Supporting Secure Services on Dynamic Aggregation of Heterogeneous Networks

SUBMITTED TO
UNIVERSITY OF SOUTHERN QUEENSLAND
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

David Tai Wai Lai
July 2010

Certification of Dissertation

I certify that the ideas, experimental work, results, analysis, software, and conclusions reported in this dissertation are entirely my own effort, except where otherwise acknowledged. I also certify that the work is original and has not been previously submitted for any other award or degree.

Signature of Candidate Date

Endorsement

Signature of Supervisor Date

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Dr. Zhongwei Zhang
(University of Southern Queensland)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Dr. Shan Suthaharan
(University of Northern Carolina at Greensboro)

Abstract

Sharing of services over IP networks prove to be an effective approach to satisfy the demand of network users when their home network cannot offer the required services. Authentication, authorization and revocation are some of the important challenges in the service sharing services over IP networks. This research address the problem associated with the authentication because it becomes more and more complicated due to the incompatible authentication schemes used by individual autonomous networks, privacy of authentication information, and the overhead in establishing the sharing. The case gets worse when a user roams from network to network.

Many efforts have been made to address these issues in the past years. Kerberos is a solution for cross realm authentication. Unfortunately, Kerberos suffers from bottle neck and single point of failure. Ad hoc aggregation cannot make use of Kerberos. Eduroam enables sharing of wireless access to users roaming between participating institutions, but only services provided by the home network is available to a user. Mobile Host Routing can route data between mobile user. But the networks are linked together in an unscalable network by network basis.

Another authentication scheme which has gained some momentum is OpenID. However, in OpenID, authentication simply means proving the ownership of an account, and there is no binding between the account and the actual user identity.

These problems and the limitations in the existing approaches inspired us to propose Service Network Graph, a service authentication infrastructure for service sharing among heterogeneous networks aggregated dynamically via self-authenticating encrypted channels. The key feature of SNG is delegation of authentication authority from one network to another. A user can use the services provided by the delegatee network as well as his home network after authenticating to the delegatee network.

When an autonomous network attaches to an SNG, not only does the network being attached delegate its authentication authority, but all authentication authorities delegated to the network also re-delegated to the attaching network. Authentication Delegation and Re-delegation makes SNG scalable.

As authentication is always done by the home network, the identity of a user can be securely bound to his account. At the same time, there is no hierarchy structure for the authentication process, autonomous networks can join an SNG in an ad hoc fashion. No

authentication bottle neck is anticipated in SNG.

The information of the authentication delegation path is stored in a Service Path which can be optimized for performance . SNG can readily extend to include mobile users.

We also proposed Dynamic Password ($DPass$) and its associated Key Exchange Scheme to be used as one of the candidate authentication schemes for SNG. $DPass$ provide strong passwords which are relatively easy to remember.

SNG together with $DPass$ provide an infrastructure for secure service sharing on dynamic aggregation of heterogenous networks. The features and feasibility of SNG and $DPass$ have been demonstrated on a simulated model of autonomous networks and an aggregate of networks in a laboratory. Our study has, to a certain extend, overcome the draw backs of the above mentioned approaches with efficiency and scalability.

Acknowledgements

I would like to thank my supervisors Dr. Zhongwei Zhang at University of Southern Queensland (USQ) and Dr. Shan Suthaharan at University of Northern Carolina at Greensboro (UNCG) for their guidance, advice, patience and encouragement. It is a pleasure to do research under their supervision. Without their help and support, I will not be able to finish my study.

The insight and lateral thinking of Dr. Zhongwei Zhang prompted me to look at problems from various perspectives and to focus on the major issues without losing sight of the details.

I am indebted to the Department of Mathematics and Computing, the Faculty of Sciences and the Computational Engineering and Science Research Centre (CESRC) for their financial support for attending conferences.

My thanks also go to Associate Professor Ron Addie. He fully supported my research and helped me to acquire the computers needed for the research.

Symbols and Abbreviations

Symbols

Symbol	Meaning
$\Rightarrow$	Speaks for
$\Rightarrow$	implies

Table 1: Symbols used in this dissertation.

Glossary

Abbreviation	Term / Meaning
SNG	$Service\ Network\ Graph$ Infrastructure which enables secure services on dynamic aggregation of heterogeneous networks.
U	$User of SNG$ A user who is entitled to use services available within an SNG.
SLS	$ServiceListingServer$ Server which lists the service available to an User on an SNG network.
S	$ServiceProvidingServer$ A server which provides services within SNG.
$HNet$	$Home\ Network$ SNG Network of which user U is a register user.
$VNet$	$Visited\ Network$ Network in which the user is currently located.
$FNet$	$Foreign\ Network$ SNG Network included in an $SPath$ which is not a $HNet$ or a $VNet$.
$RNet$	$Remote\ Network$ SNG Network not included in an $SPath$.
$ANet$	$Alien\ Network$ Network which is not part of an SNG.
$SNet$	$ServiceProvidingNetwork$ SNG Network which provides the requested service.

Table 2: Glossary of terms used in this dissertation - Part 1.

Abbreviation	Term / Meaning
$DPass$	$Dynamic\ Password$ Consists of two parts: Dynamic Part and Static Part.
$SDPass$	$Static\ Part\ of\ DPass$ First part of $DPass$ with a relatively longer life span.
$DDPass$	$Dynamic\ Part\ of\ DPass$ Second part of $DPass$ with a relatively shorter life span.
$SPath$	$ServicePath$ Network path that leads up to the server.
AS	$Authentication\ Server$ Server which authenticates user U.
SW	$Switch$ A network device.
R	$Router$ A network device.
NED	$NEtworkDescription$ language Language in $OMNeT++$
N	$Network$ An autonomous Network.
$NPMode$	$NetworkParticipationMode$ State of an autonomous network joining or leaving an SNG.
$USMode$	$UserServiceMode$ State of an autonomous network providing services to a user.
$SPath$	$ServicePath$ Service information including the network path from a user to a server.
$SAPath$	$ServiceAccessPath$ Network path from the user to the server.
$H(X)$	$Hashvalue of X$
$\{X\}_K$	$Encryption of X using key K$

Table 3: Glossary of terms used in this dissertation - Part 2.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

According to the Computer Crime and Security Survey [19] conducted by Computer Security Institute and Federal Bureau of Investigation (FBI) in 2005, the average loss per year due to unauthorized access to information is about US$300,000 per respondent of the survey. The situation has not been improved ever since - a similar survey [65] conducted in 2008 indicates that the average loss per year due to computing security incidents is only slightly under US$300,000 per respondent. These figures highlighted the severity and importance of network security, especially authentication, authorization and revocation of access rights. Without doubt, security issues become even worse and more complicated when networks are dynamically joined together to share their services. It motivated us to explore the practical solution to deal with the security of dynamic aggregations of heterogeneous networks which share services among each other.

1.1 Specific Problems

The complexity and insecurity of distributed networks system are manifested in various ways. For instance, some are related to the user authenticity; some are related to user authorization; while others are related to service availability.

When a registered user [1] of a network requests a service, the user first checks if the service is available or not. Then the second and third steps would be authenticating the user and verifying if the user is authorized to use the service or not. Furthermore, during service access, relevant parties have to check continually if the user authentication and / or authorization has been revoked or not.

[1] A registered user refers to a user who has been registered as a legitimate user to the current network. In this thesis, we always refers to a registered user wherever a user is mentioned.

When some autonomous networks are linked together for the purpose of sharing services, the situation is similar. Because the networks are autonomous, some of the important questions to ask are:

- who confirms availability of a service?
- who authenticates the user?
- who authorizes the user?
- how revocation can be executed?

1.1.1 The Availability of a Service

Within an IP network community, a user needs to register himself with a local network to enjoy all the services which are available on the network. The range of services available is often a determining factor for a user when choosing a network. However, it is difficult or impossible for one network to provide users with all the services its users want to use due to technological and financial limitations. To overcome these constraints, the range of services is supplemented by those provided by other autonomous networks. When users make a request of a particular service, they often are not sure if such a service is available or not; and even if available, where to get the service.

The services that are available on an autonomous network are usually listed and distributed to its registered users. Problems may arise when a service is shared. It is easy to understand the fact that users of the Service Providing Network ($SNet$) are aware of the services. However, users of another network which share the services may not be aware of the services. It is desirable to provide users with the services available along with related service information at the time when they approach network for services.

When a network changes its range of services shared with other networks, it is important to timely update all service lists in networks. It will be disastrous if a user requests and then waits for a service which is not offered any more. In a dynamic aggregation of networks, it is critical to keep the service list up to date as the configuration of the aggregate changes.

1.1.2 The Authentication of a User

Authenticating a user who requests a service is by no means a trivial task. It is, however, not a big deal if the user is currently located in his home network ($HNet$), and requests a service which is available from his home network. However the case is not as straight forward when the user is visiting other network ($VNet$). Using the jargons, we may say in the first case, $VNet$, $SNet$ and $HNet$ are all the same. In the second case, $VNet$, $SNet$ and $HNet$ may differ from each other.

When the user is not in his home network, who is going to authenticate the user when the user is not in his home network, and how?

- **Autonomy of Networks** To authenticate a user, a network must share some common secret authentication information with the user. As we know, authentication information repository of one network is owned by the individual network. Furthermore, individual autonomous network can have different authentication schemes which require different sets of authentication information.

  When a user requests a shared service from a service providing network $SNet$, the user has to identify himself to $SNet$. If $SNet$ happens to be the $HNet$, $SNet$ / $HNet$ can authenticate the user. Otherwise $SNet$ has to be informed and trust the result of authentication performed by another network.

  When a user is at his $HNet$ ($VNet$ equals to $HNet$), authentication can be done by $HNet$ and the result be sent to $SNet$. It only involves one layer of trusting relationship. When the user is not in his home network, ($VNet$ not equal to $HNet$), authentication request must be sent from $VNet$ to $HNet$ and the authentication result forwarded to $SNet$. If he is currently not located within his home network, he may not be able to contact his home network for authentication. Authenticating the user is not a trivial task.

  It is practically not feasible for all networks to be able to authenticate all users. Privacy, compatibility of authentication schemes, and scalability are just few of the issues which forbid such a global authentication fantasia. Authenticating a user that registered in another network is the determining factor for successful service sharing.

  Authenticating a user by password is a simple and traditional approach. This approach has limitations. Apart from the difficulty of remembering a password, another limitation of password is the scalability. To verify a password, we need to share some common secret information with the authentication server within a particular autonomous network. The shared information may not be available to authentication servers in other autonomous networks. Thus users may not be able to authenticate themselves when trying to use services in other autonomous networks.

  Setting up and updating a common set of shared secrets for all users in all networks is impracticable.

- **Scalability of Aggregation of Networks**

  It is not practical for all networks to link to every other network for service sharing. As the links in such mesh aggregation of network grows exponentially with the number of networks involved, the administrative overhead for joining such aggregation of networks become more and more expansive. If we wish to have a dynamic

aggregation of networks, we should look for a configuration that allows the aggregation to grow and the member networks can still share their services efficiently among themselves.

- **Heterogeneous Authentication Schemes**

  Different autonomous networks have different authentication schemes, different formats of shared information and different authentication client software. To access services in different networks, users may have to install various authentication client software and share different common secret information with different networks.

- **Unfriendly Passwords**

  Legitimate users need to be authenticated before using services provided by the distributed networks system. As we stated before, password is a common and readily accepted authentication token which allows user to access the networks system and makes network services available to legitimate users. Password themselves have two extremes. A friendly password is something easy to remember which usually means easy to crack. On the other hand, a strong password could be long with a short life time [53]; it means that strong passwords are harder to remember. So when composing a password, we face the dilemma of user friendly passwords are weak while strong passwords are hard to remember. This dilemma inspires us to design a password scheme [33] in which the passwords are strong but easy to remember and with a short life time, effectively a one-time password [20].

1.1.3 Authorization Relay

Service provider networks ($SNet$) may choose to deliver selected services to nominated networks only. Authorization of a user helps to filter the legitimate users from the rest.

There is another form of authorization, $sharing\ authorization$. Apart from simple sharing, a network can act as an agent for service sharing. Suppose a $SNet$ provides a service $Service$ and shared with Net_A. A third network Net_B may request to use $Service$ via Net_A. If the $SNet$ is willing to share $Service$ with all service requests coming from Net_A, it provides the service. On the other hand, if the $SNet$ is willing to share $Service$ with service requests coming from users of Net_A only, it declines the request.

1.1.4 Reluctance of Authentication for Roaming Users

Due to the lack of authentication information, service provider networks ($SNet$) are reluctant to grant authorization to users from another network. The situation is further complicated by the fact that networks are allowed to join and leave a distributed networks system at any time. This will pose great hardship for roaming users - both the move-and-stay and on-the-move users.

1.1.5 Revocation of a User

User revocation information should be passed on to a service providing network $SNet$ in a timely fashion. The common approach of updating user revocation information is either on a regular time interval basis or during initial authentication, however, this approach is not efficient and not effective [49] [70].

- **Total Revocation of Rights**

  If a user is revoked of all its right, it is equivalent to revoking its authentication status. When a user logs in, an authentication server checks the user's login information and authenticate the user as required. The login status of the user changes from "not authenticated" to "authenticated" and remains so until the login session ends. For revocation to come to effect, the authentication server has to be aware of the revocation and act accordingly. Since authentication servers check the authentication and revocation information when users log in, the earliest time to revoke a user's authentication status is the next log in even if the revocation information was updated some time ago. If a revoked user is currently logged in, (s)he can still use a requested service until the session ends.

- **Partial Revocation of Rights**

  It is common for the servers to check authentication and authorization only when requests for services are received. The servers provide service to authenticated and authorized requests until the service sessions end without checking authentication and authorization while the session is still alive. When a user is using a service with certain user rights, it retains its user rights until the service session ends.

  If a user is denied some or all of the user rights to services which it is currently accessing, it would not be denied of the rights until the server has updated its access control information and the service is accessed by the user next time.

  The time gap between revocation and actual banning of user from services may lead to potentially disastrous damage.

1.1.6 Mobility of Users

A mobile user does not stay in one place all the time. When he roams to and within another visited network, and would like to request a service which he can access in his home network, he can only contact the visited network for authentication and service. When the visited network receives such a request, there should be a protocol and mechanism so that the service request can be handled.

1.2 Related Work on the Problems

When a network provides a service to users of another network, the service is said to be shared and the network is a service providing network ($SNet$). When a network includes a shared service as part of its services, the shared service is said to be an out-sourced service and the network is a service requesting network.

A service providing network has to establish the identity of a user requesting a service. Many plausible technologies are proposed. We look at some of the prominent ones.

- **A Common Authentication Information Set:** When many autonomous networks form an aggregation for service sharing, the network administrators face a problem of authenticating users from other networks which have various authentication schemes and authentication information sets. It is obvious that enforcing a common authentication scheme is not feasible and involves substantial administrative overheads. For instance, when a network links to an aggregation of networks and subsequently detaches from the aggregation, switching to the common authentication scheme and reverting back to the original authentication scheme requires all users of the network to collect and present a different set of authentication information. Even a common authentication scheme is in force, maintaining a global set of authentication data will fail as networks may link to the aggregation or detach from the aggregation at any time. Even worse, some networks may be reluctant to disclose the authentication data for security reasons. As a direct result of this, other networks may not have the authentication data required by the common authentication scheme. A typical example is the X.500 [1] plan which has never succeeded in producing a global database of named entities.

- **Secrecy of the Private Key:** Authentication in X.509 [2] is based on the secrecy of the private key and the binding of the public key to a user name. For instance, a Certificate Authority (CA) authenticates a user and binds its name and public key in a digital certificate. If a user demonstrates he is the owner of the private key with a corresponding X.509 certificate, then he is the user named in the certificate. This authentication mechanism is built upon the unanimous trust for the Certificate Authority (CA).

  Note that an administrator of an autonomous network may decide to set up a CA for the network or empower a third party to run the CA. However, when many autonomous networks form an aggregation, they must agree on a common CA to issue all certificates or on CA certificate chaining. We envisage that the workload increases with the number of users involved.

- **Trust Agent:** Another approach is to establish a *trust* [12] [57] [5] [32] [6] [8]. *Trust* is the result of an assessment of an entity relative to a domain of action [15]

by an observer. When an observer is authorized by a network administrator to give trust recommendations [48] [67], the observer becomes a $trust\ agent$. The trust is represented by a token and each trust token is signed by the $trust\ agent$.

It is reasonable for each autonomous network to have its own set of independent trust agents. A user is asked to provide trust tokens by a few trust agents. By using the aggregated result [7] [15] of the trust tokens, the server can determine the authentication and authorization status of the user for the requested service.

This way of authentication works fine for individual networks. However, for an aggregation of networks, each autonomous network may have its own set of trust agents. Either all the networks adopt the same common set of trust agents or users have to collect trust tokens from different sets of trust agents for services out-sourced by different networks.

- **Central Authentication Server:** Kerberos [51] presented a user friendly solution in which users authenticate with a central authentication server and the authentication status can be propagated to the required servers. With one set of authentication information and one log-in, users would be able to access services available from servers within the same realm.

  The problems with this approach is that users have to keep track of the availability of services as they have to specify which server and service they want to access. Users must maintain an updated list of services available by themselves.

  Secure European System for Applications in Multi-Vendor Environment, commonly known as **SESAME**, is often considered as the Euro version of Kerberos. It is a single sign-on project funded partially by the European Commission. It uses Privilege Attribute Certificates and supports different security policies across multi-domains.

- **Eduroam:** Eduroam provides wireless service to users roaming between participating institutions. The Home network authenticates the user and the service providing network determines the authorization to use local network resources. The authentication technology used by Eduroam is based on 802.1X and Remote Authentication Dial In User Service (RADIUS) proxy servers.

  As Eduroam shares wireless access with a user, the user has to move himself to a location within the range covered by the wireless LAN in order to use the shared service. Another main drawback of Eduroam is that the shared services is still limited to public services only.

- **Mobile Host Routing:** As the name implies, Mobile Host Routing (MHR) routes data to a mobile host by establishing an Home Agent (HA) and a Foreign Agent (FA) in a network. When the user is in his home network, he authenticates himself with the

HA. When he moves to another network, he has to connect to the FA of the visited network which forwards his user information and credentials to the HA of the user's home network for authentication. The main drawback of Mobile Host Routing is that it applies to networks linked together on a network by network basis. It cannot be applied to ad hoc aggregation of networks.

- **OpenID:**

  OpenID [2] authentication is simply a confirmation process for the ownership of an Identifier which is assigned by an OpenID Provider (OP). Each OP has its own requirements for setting up an OpenID account. Most of them just require a username and password. Note that there is no binding of identity required.

  OpenID can assert that you own the $Identifier$ you presented, but what that $Identifier$ represents is in limbo.

- **Strong and Friendly Password:**

  The classic password protocol Encryption Key Exchange (EKE) by Bellovin and Merritt [9] and its successor Augmented-EKE ($A-EKE$) [10] is a collection of authentication protocols that apply encryption for key exchanges. EKE uses a shared secret indirectly to authenticate servers and users. However EKE is susceptible to the Denning-Sacco attack [69] in which a stolen session key can be used to launch a replay attack on the password. A more efficient Minimal Encrypted Key Exchange ($M - EKE$) [69] was proposed.

  Other alternatives to $A - EKE$ were proposed. Strong Password-Only Authenticated Key Exchange ($SPEKE$) [25] uses the hash of the password as the base for exponentiation instead of a fixed primitive base. Extended Password Key Exchange Protocols [26] is a family of protocols that extend the $A - EKE$ and $SPEKE$ protocols to $B - EKE$ and $B - SPEKE$ by using a second Diffie-Hellman exchange instead of a digital signature to prove that a user has knowledge of a certain password.

  Open Key Exchange (OKE) [46] eliminates the use of encryption for the user's public key. Another feature of OKE is that the user's public key can be reused as long as the private key is kept secret. Asymmetric Key Exchange (AKE) [78] is another class of protocol that exchanges keys without using encryption. Initially, each party computes a secret and generates a verifier with a one-way hash function. They swap their secrets and use them as long term swapped secrets. For each session, each party generates another session secret and swaps with the other party. The session key is generated from the swapped long term and session secrets. Authentication is completed when both parties confirm that they have the same session key.

[2] Formed in June 2007, OpenID Foundation (http://openid.net/foundation) promotes the OpenID authentication concept. More information on OpenID can be found in http://openid.net/specs/.

All the above password protocols do not offer transmission of new password.

1.3 A Practical Example of Service Sharing

In this section, we give a real life example which illustrates some of the problems stated previously.

1.3.1 Background

Hong Kong and Macau are the two Special Administrative Region (SAR) of the People's Republic of China [14]. They enjoy a high degree of autonomy and have executive, legislative and independent judicial power. In particular they maintain their customary law previously in force. As a result, Hong Kong and Macau have different and independent custom laws and regulations.

When a resident of Hong Kong leaves or enters Hong Kong, he can use the biometrics based Automated Passenger Clearance System [24] (e-channel for short) and identifies himself using his own finger-prints [55, 45]. The e-channel system matches the finger-prints captured by a scanner with a digitized version stored in the Hong Kong Custom information repository. Macau Public Security Forces Bureau still uses the traditional manual passport inspection system and is moving towards the same e-channel system as that implemented in Hong Kong [44][3]. Hong Kong residents can authorize Macau custom to get a copy of their finger-prints from Hong Kong custom. A resident of Hong Kong can use both the traditional manual passport inspection system and the e-channel system after the finger-prints are transferred to Macau custom.

1.3.2 Authentication Information Sharing

This is what happens in reality. When a resident of Hong Kong heads for Macau, he has to use the e-Channel when he leaves Hong Kong. At the custom of Macau, he can either use existing manual passport inspection system or the newly implemented e-channel. If he chooses to use the passport system, the resident has to have two sets of identity information for the two authentication systems - passport / Identity Card to be used in Macau and finger-prints to be used in Hong Kong.

If the resident wishes to use the new Passenger Clearance Service for entry to Macau, he can use a single set of identity information for both Macau and Hong Kong. The authentication information stored in Hong Kong Custom will then be copied and shared with Macau Custom. However, both Hong Kong custom and Macau custom must have an identical but

[3]More information about the hardware and the supplier can be found in http://www.nec.co.jp/press/en/0812/1101.html

independent copy of the resident's finger-prints. The transfer of identity information and finger-prints has to be done manually and takes up to three working days after the resident made an initial application for the transfer. It can be error prone and not scalable if you consider the fact that there are seven million residents in Hong Kong.

1.3.3 Service Sharing

The Passenger Clearance Service is not shared. The use of the same Passenger Clearance System hardware and having independent copies of identical information can be compared to the case of networks using the same authentication scheme. The participating networks use the same authentication scheme but each network keeps an independent copy of authentication information which are identical. It is definitely better than two autonomous networks with different authentication schemes and authentication information formats.

On the other hand, if the Passenger Clearance Service is truly shared, any Hong Kong resident can authenticate himself using the Passenger Clearance Service provided by Hong Kong and authentication information stored in Hong Kong Custom when they enter or exit Macau. No massive transfer and duplication of authentication information between two SARs are required. This will prevent security loop holes associated with updating, synchronizing and revoking authentication information.

1.4 Research Contribution

This research is targeting the authentication, authorization, revocation of rights and mobility issues of service sharing. The outcomes from this study are as follow:

1. A secure and scalable user authentication framework for service sharing within a dynamic aggregation of autonomous networks. The details can be found in the following refereed publications:

    - An infrastructure for service authentication and authorization revocation in a dynamic aggregation of networks. WSEAS Transactions on Communications, 4(8):537-547, August 2005.
    - Network service sharing infrastructure: Service authentication and authorization revocation. Proceedings, the 9thWSEAS International Conference on Communications, July 2005. (CD Proceedings)
    - Secure service sharing over networks for mobile users using service network graphs. Proceedings, Wireless Telecommunication Symposium 2006, Pamona, Ca, USA, April 2006. (CD Proceedings)
    - Achieving secure service sharing over ip networks. Proceedings, ASEE Mid-Atlantic Section Spring 2006 Conference, April 2006. (CD Proceedings)

- Towards an authentication protocol for service outsourcing over ip networks. Proceedings, the 2005 International Conference on Security and Management, (7):3-9, June 2005.
- Efficient information propagation in service routing for next generation network. Proceedings, The Fourth International Conference on Rough Set and Knowledge Technology, pages 342-349, 7 2009.

2. A relatively strong but easy password scheme to be used as a standalone authentication scheme or in conjunction with the proposed secure authentication framework for service sharing. More details can be found in the refereed publication:

   - Integrated key exchange protocol capable of revealing spoofing and resisting dictionary attacks. Technical Track Proceedings, 2nd International Conference, Applied Cryptography and Network Security, Yellow Mountain, pages 115-124, June 2004.

3. A theoretical proof for the correctness of the proposed secure authentication framework.

4. A formal proof for self-authentication of a communication channel in which the traffic is encrypted. This proof forms part of the basis for the correctness of the secure authentication framework. More details can be found in the refereed publication:

   - Self-authentication of encrypted channels in service network graph. Proceedings, 2008 IFIP International Conference on Network and Parallel Computing, pages 163-167, (NPC 2008), October 2008.

5. A formal proof that service records used in the secure authentication framework can be optimized. This proof shows that the secure authentication framework is efficient and scalable. The details can be found in the following refereed publications:

   - Improving efficiency and scalability of service network graph by re-routing. Proceedings, 1st Asian Conference on Intelligent Information and Database Systems 2009, (CD Proceedings), 4 2009.
   - Service re-routing for service network graph: Efficiency, scalability and implementation. International journal of Computer Networks Communications (IJCNC). ISBN (on-line) 0974-9322. ISBN (print) 0975-2293., 1(1), 4 2009.

6. Two case studies:

   - Application of the proposed secure authentication frame work as a model to two network simulations.

- A practical application of the basic features of the proposed secure authentication framework using C++.

1.5 Structure of the Dissertation

We have gone through from the issues associated with the service sharing over IP networks to the design of Service Network Graph, the Dynamic Password, the Key Exchange using Dynamic Password protocol and to the application of the Service Network Graph and Key Exchange using Dynamic Password protocol on aggregates of heterogeneous networks. In Chapter 2. The technology reviewed includes $Kerberos$, $Sesame$, $Eduroam$, $Mobile\ Host\ Routing$ and $OpenID$. We comment on the drawbacks of each technology when used for the purpose of service sharing for ad hoc aggregates of autonomous networks.

In Chapter 3, we propose the Service Network Graph. An Service Network Graph with the simplest logical topology is established when a network delegates its authentication authority to another network by sharing an authentication Token key with the other network. Services are then shared. Information about the shared services are recorded as Service Paths.

The next chapter, Chapter 4, focuses on a Service Path. Optimizing Service Paths makes them more scalable and efficient. In this chapter, the way to optimize an Service Path is presented and a formal proof of the optimization process is also include.

When a network joins an Service Network Graph, authentication authorities are delegated to (and by the joining network). Participating networks can track the origin and destination of authentication tokens sent and received because encrypted channels authenticates themselves. In Chapter 5, a formal proof of the self-authentication of encrypted channels is given.

Service Network Graph is designed such that individual network can have their own authentication scheme. Nevertheless, username and password pair is by far the most common authentication information in our daily life. In view of the weakness in common password schemes, we proposed Dynamic Password scheme which is effectively a onetime password scheme in Chapter 6. The associated Key Exchange using Dynamic Password scheme is also discussed.

Chapter 7 justifies Service Network Graph using Dynamic Password as a correct security protocol using the concept of Strand Space. The correctness can be proved by demonstrating that Service Network Graph using Dynamic Password has the secrecy and agreement properties. The correctness claims are presented as propositions and the proofs are built up from series of lemmas.

As an illustration, we simulate the Service Network Graph process using $OMNeT++$ simulator in Chapter 8. The simulation scenarios include the cases when a user is at his

home network and in a visited network.

In Chapter 9 we presented a simple implementation of Service Network Graph using Dynamic Password. Virtual Local Area Network and a ”Router on a stick” configuration is used for the physical networking, and the services shared are simple time, date, echo and name services.

Finally, we rounded up our discussion with a conclusion in Chapter 10.

Chapter 2

Service Sharing Technologies

In this Chapter, we take a brief look at some of the existing technologies for service sharing among autonomous networks. We comment on the draw backs of each technology when they are applied in the context service sharing.

2.1 Autonomous Network

The first technology is on an autonomous network. An autonomous network operates under the control of a single $administrative\ domain$. An autonomous network has its own authentication scheme and authentication credential repository as shown in Figure 2.1. Each network can only authenticate and authorize registered users only.

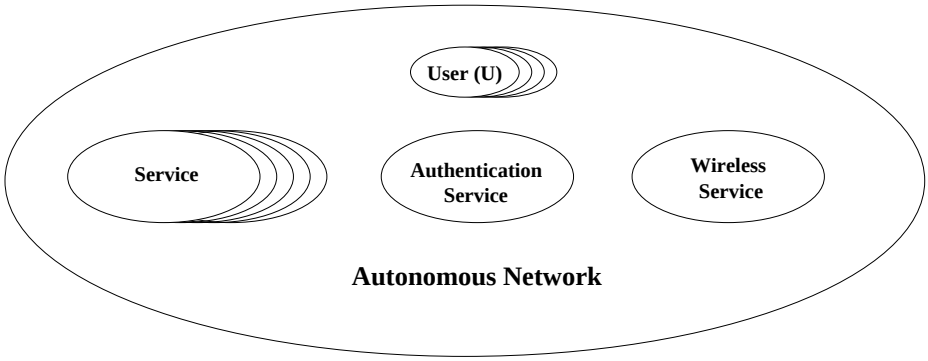


Figure 2.1: An autonomous network.

Some autonomous networks provide services only to their members; whilst some provide limited services to the public. Users must register to a given network to access all its services. Services shared across networks are limited to the public services offered by each network.

2.2 Intranet and Internet

As the second technology, we have Intranet and Internet. An Intranet usually connects networks within an organization. The inter connected networks function like an extended network sharing the same services and authentication information and scheme.

Internet is a global interconnection of networks. Basically, each network acts like an autonomous network and offers limited services to the public and other networks. In most cases, public services are provided along side with advertizement to cover the running cost, while some services gather user's personal information for commercial use. When authentication is requested by a network, the authentication schema and the authentication data repositories used are basically independent of each other. The authentication data may vary from simple user account name and password or public and private key to biometric and token authentication [24].

Sharing of services on Intranet and Internet are limited to public services only.

2.3 Kerberos

Figure 2.2: The Kerberos logo.

When a user is authenticated in his home realm (equivalent to his home network), other realms (or networks) within the Kerberos [1] system accepts the user's identity if cross-realm operation is enabled.

[1] Kerberos is the network authentication protocol developed by Massachusetts Institute of Technology (MIT) as part of Project Athena. The current version of Kerberos is V5, specified by RFC4120 dated in July 2005.

Kerberos is based in part on Needham and Schroeder's trusted third party authentication protocol [50]. Users in the Kerberos Protocol are called $principals$. The identity credential of each Principal is represented electronically as a $ticket$. The main component of Kerberos is the Key Distribution Center (KDC). A KDC consists of two servers, the Authentication Server (AS) and the Ticket Granting Server (TGS). Each entity, including principals and servers, shares a secret (a password) with the Key Distribution Center for authentication purpose. Principals whose authentication data are registered with a given Key Distribution Center are said to belong to the same realm.

Figure 2.3 depicts the basic structure of a Kerberos realm.

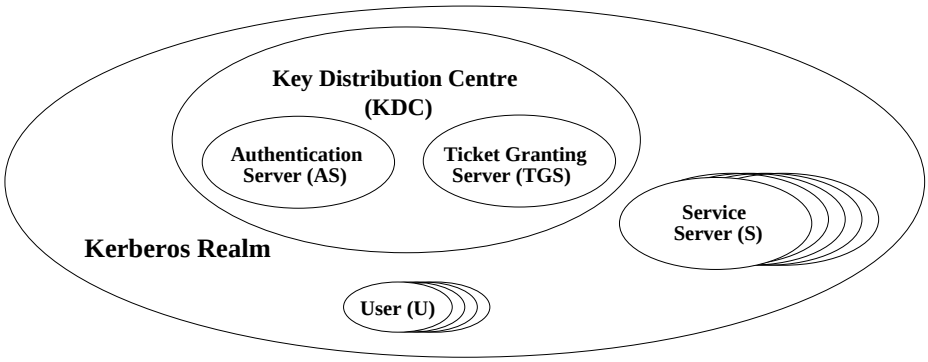


Figure 2.3: A Kerberos realm.

The Kerberos authentication process consists of six steps as shown in Figure 2.4, Figure 2.5, and Figure 2.6.

Step 1 A user sends a plain text $Authentication\ Request$ to AS for credentials to use services. Both the user and the service information are included in the request.

Step 2 An AS retrieves the shared secret of the user from the authentication data repository. It also generates a TGS session key. The AS returns two messages to the principal. The first one is an $Authentication\ Reply$ message formed by encrypting the TGS session key using the user key generated from the shared secret of the user. The second message is the TGS session key encrypted using the secret key of TGS and is called the $Ticket\ Granting\ Ticket\ (TGT)$. This TGT is the credential to access services.

Step 3 After receiving the $Authentication\ Reply$, the user can use his shared secret to recover the TGS session key. The user also uses two messages to communicate with the TGS. The first message is a $Service\ Request$ made up of service information and the TGT. The second message is an $Authenticator$ formed by encrypting the user information and a time stamp with the TGS session key just recovered from the $Authentication\ Reply$.

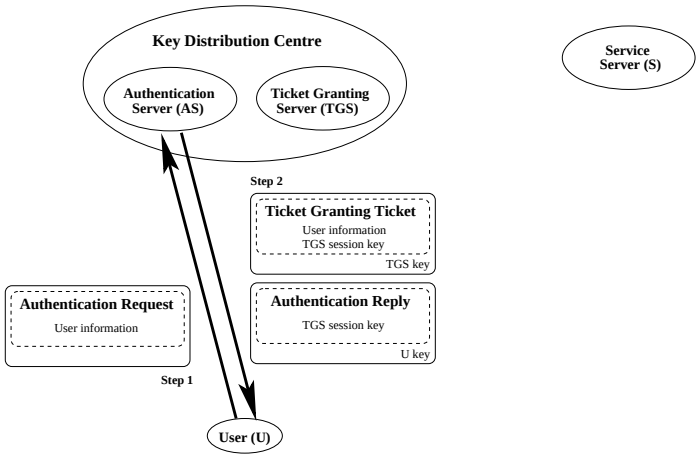


Figure 2.4: Steps 1 and 2 of Kerberos in action.

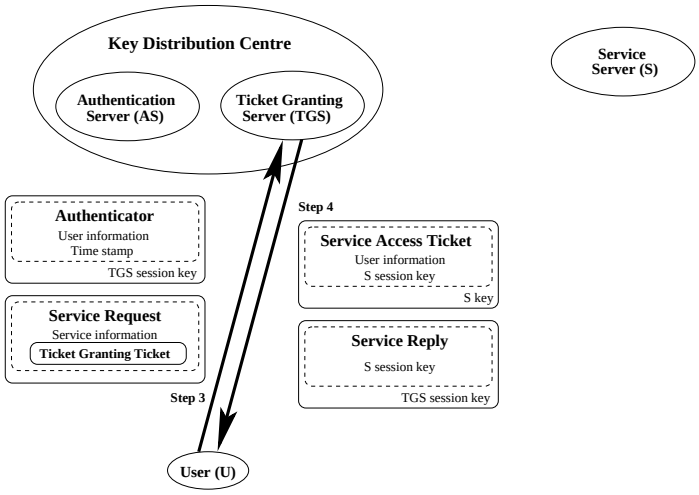


Figure 2.5: Steps 3 and 4 of Kerberos in action.

Step 4 TGS decrypts the TGT inside the $Service\ Request$ and gets the TGS session key for this session. Using the TGS session key just recovered, TGS can retrieve the user information from the $Authenticator$. Similar to AS, TGS generates a service server (S) session key; encrypts the S session key and user information using the shared secret of S to form a $Service\ Access\ Ticket$; encrypts the S session key using the TGS session key to form a $Service\ Reply$. The $Service\ Access\ Ticket$ is the credential to access the requested service of a given server.

Step 5 The user retrieves S session key using the TGS session key. The user now forwards the $Service\ Access\ Ticket$ and an $Authenticator$ similar to the $Authenticator$ in

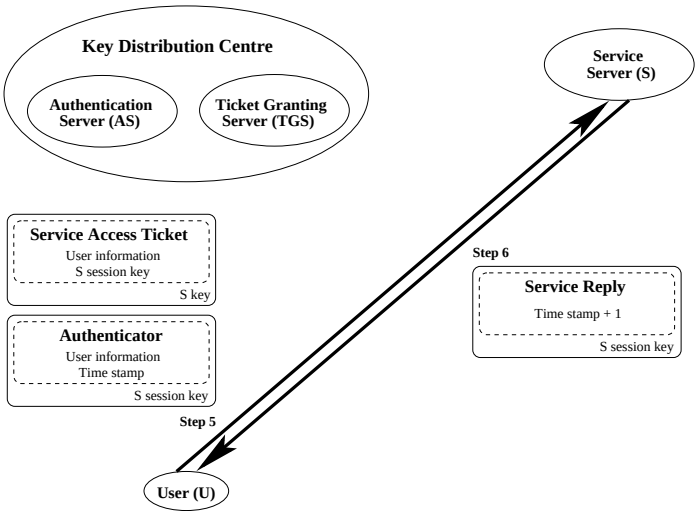


Figure 2.6: Steps 5 and 6 of Kerberos in action.

Step 3 but using the S session key.

Step 6 The service server S confirms the service is available by returning a $Service\ Reply$ fomred by encrypting the time stamp from the $Authenticator$ plus 1 with the S session key.

Upon completion of these six steps, the user can now access the service provided by S.

2.3.1 Cross-Realm Operation of Kerberos

Kerberos enables users of one realm to access services provided by another realm. This is called $cross - realm\ operation$. To facilitate the cross-realm operation, the KDC from one realm must share an inter-realm key with the KDC of another realm. When realms share inter-realm keys with another realm, they established themselves as a principal of the other realms.

When the user requests a cross-realm service, he would get a cross-realm TGT for the realm which offers the service that has been encrypted with the shared inter-realm key between the two KDCs as shown in Figure 2.7. The cross-realm TGT is forwarded to the remote TGS of the other realm which either returns a service ticket for the server in remote realm or rejects the request.

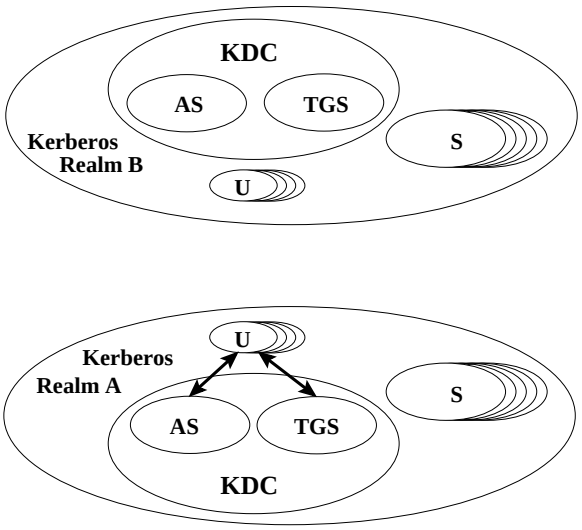


Figure 2.7: Steps 1 to 4 of multiple realm Kerberos in action.

There might be two extra forwarding steps, and the extra forwarding steps are shown as Step 4a and 4b in Figure 2.8.

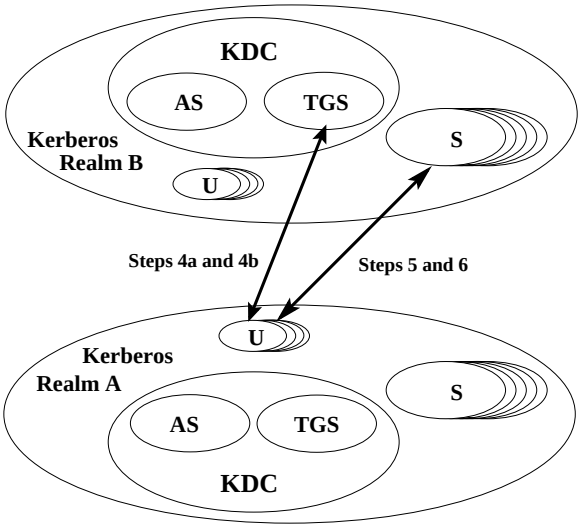


Figure 2.8: Extra steps, steps 5 and 6 of multiple realm Kerberos in action.

Note that the cross-realm operation is transitive, which means, for example, as shown in Figure 2.9 and 2.10,

- Realm A shared an inter-realm key with Realm B;
- Realm B shared an inter-realm key with Realm D;

- Realm D shared an inter-realm key with Realm C;

The TGT sequence for cross-realm authentication is from Realm A to Realm B, then to Realm D and finally to Realm C as follows:

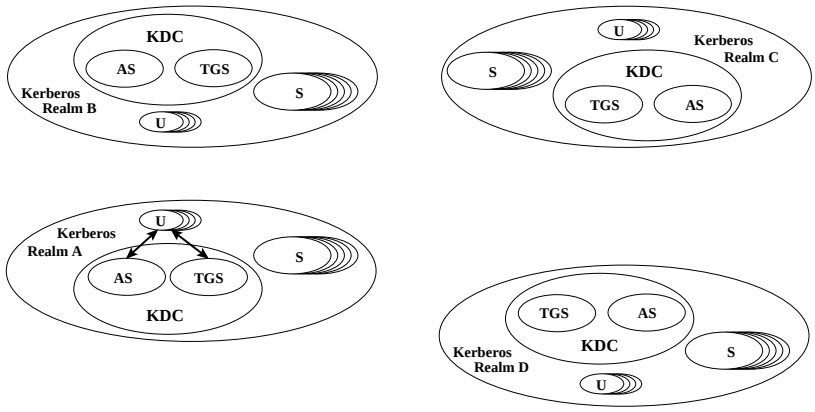


Figure 2.9: Steps 1 to 4 of multiple realm Kerberos in action.

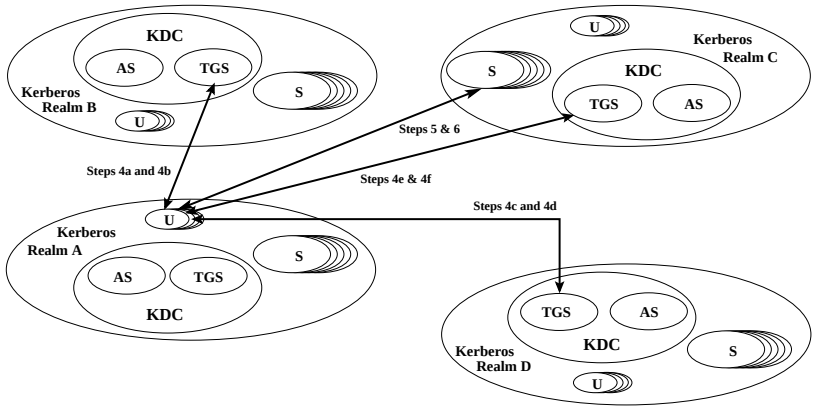


Figure 2.10: Extra steps, steps 5 and 6 of multiple realm Kerberos in action.

Steps 1 to 4 A user from Realm A authenticates himself to the AS of Realm A and gets TGT_A as shown in Figure 2.9.

Steps 4a and 4b The user forwards the TGT_A to TGS of Realm B and gets TGT_B.

Steps 4c and 4d The user forwards the TGT_B to TGS of Realm D and gets TGT_D.

Steps 4e and 4f The user forwards the TGT_D to TGS of Realm C and in return gets a $Service\ Access\ Ticket$.

Steps 5 and 6 The user uses the $Service\ Access\ Ticket$ to access service in Realm C just like accessing service in Realm A.

The extra steps are used to forward TGT between KDCs of successive communicating pairs of realm. A database may be required to keep track of the possible authentication paths. If the communicating realm pairs can be organized hierarchically as shown in Figure 2.11, authentication paths of minimal length can easily be worked out using the hierarchy.

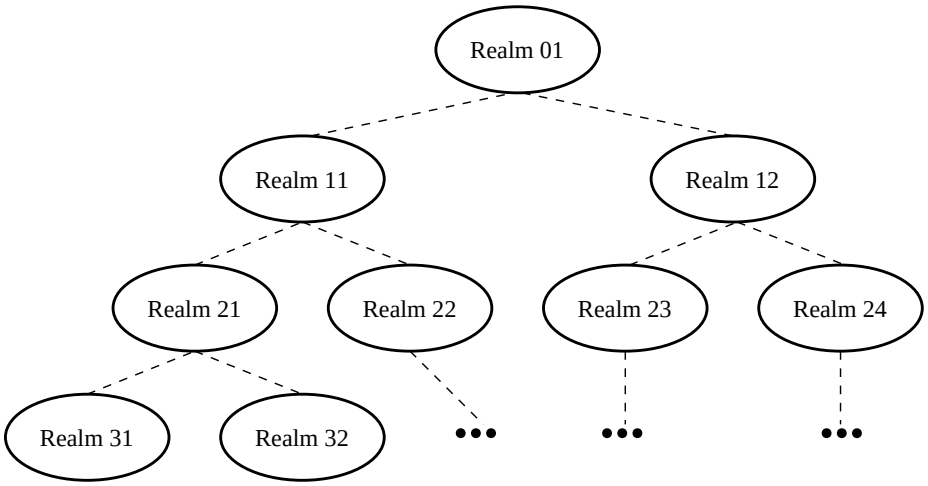


Figure 2.11: Hierarchical arrangement of Realms for cross-realm authentication

2.3.2 Kerberos V5 Applications

The functionality of Kerberos is extended by various Internet Standards such as [80, 75, 82, 83, 81, 47, 76, 29]. Popular applications based on Kerberos V5 includes:

- User interface to TELNET protocol - telnet
- Remote login - rlogin
- File transfer program - ftp
- Remote shell - rsh
- Remote file copy - rcp
- Kerberized super user - ksu

2.3.3 Drawbacks of Kerberos

Although Kerberos has been used for many purposes, there are some technical limitations.

- Each network must be Kerberized before joining, which means each network must

  1. Implement a KDC including an AS and a TGS
  2. Service Servers must accept $Service\ Access\ Tickets$ and $Authenticator$ as the credential for using services.

  This certainly limits the use of Kerberos for dynamic aggregation of heterogeneous networks.

- Each realm KDC has to keep track of the TGT paths for service tickets.
- Hierarchical organization of KDC could make the traffic heaviest at the top level.
- Hierarchical (tree like) organization of KDC could suffer single point of failure.
- Mesh organization of KDC for cross-realm authentication makes the complexity of inter-realm key management increase exponentially with the number of networks involved.
- Users have to keep track of the availability of services as they have to specify which server and service they intended to access. As a result, users must maintain an updated list of services available by themselves.
- The administrative work involved in setting up realms makes Kerberos semi-static. If autonomous networks are to attach and detach from an aggregate of networks in a dynamic fashion, we need a more efficient solution.

2.4 SESAME

In this section, we look at SESAME[2], Secure European System for Applications in Multi-vendor Environment [31, 54]. SESAME is a single sign-on project partly funded by the European Commission. It authenticates users and issues Privilege Attribute Certificates. When operating in a multi-domain environment, it supports different security policies across the domains.

The operation of SESAME is similar to Kerberos. The authentication process for SESAME version 2.0 is listed below:

[2]More details can be found in the SESAME home page https://www.cosic.esat.kuleuven.be/sesame/

Step 1 A user authenticates to an $Authentication\ Server$.

Step 2 The $Authentication\ Server$ returns an encrypted $Identity\ Token$ to user.

Step 3 The user presents the $Identity\ Token$ to a $Privilege\ Attribute\ Server$.

Step 4 The $Privilege\ Attribute\ Server$ returns a digitally signed $Privilege\ Attribute\ Certificate$ (PAC) to user.

Step 5 The user presents a PAC to the application server.

Step 6 The Application server provides services according to the security attributes in the PAC and its own security policies.

A PAC can be used more than once at more than one application server. Recall that Kerberos key distribution protocols can be used to establish session keys between users and servers.

The major feature Sesame offers which Kerberos does not is the use of PAC. With the PAC, application servers can implement their own fine grain access control and audit checks. The approach to security for SESAME was considered an alternative to the DES based security service in Distributed Computing Environment based on Kerberos [30].

Unfortunately, SESAME has similar drawbacks to that of Kerberos. We would not repeat them here.

2.5 Eduroam

Figure 2.12: The Eduroam logo.

The eduroam initiative was started in 2003 within the Trans-European Research and Education Networking Association (TEREN) Task Force on Mobility (TF-mobility). The task

force started off by setting up the roaming policy [62] and inventory of existing technologies [61, 60, 59, 58]. The Task Force eventually came up with the roaming architecture [63] and the implementation plan [64] in 2007.

Eduroam (Figure 2.12) aims to provide wireless service to users roaming between participating institutions. Authentication on the Eduroam is done by the home network and the authorization to use local network resources is determined by the network visited.

It is worth pointing out that the authentication technology used by Eduroam is based on 802.1X and Remote Authentication Dial In User Service (RADIUS) proxy servers. In more detail, when a user requests login, the authentication request is forwarded to the local authentication server.

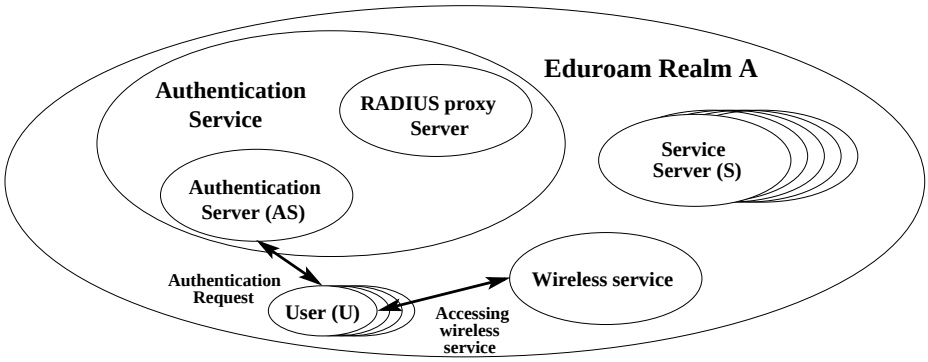


Figure 2.13: Action of Eduroam for a user in home network

If the authentication request can be processed locally, the user would be granted the authority to access the wireless services as shown in Figure 2.13.

Otherwise, the local authentication server forwards the request to a RADIUS proxy server. As all the RADIUS proxy servers are organized in a hierarchical order, it is simple to traverse the hierarchical tree of proxy servers to reach the RADIUS proxy server of user's home network. An example path taken is shown in Figure 2.14.

1. National RADIUS proxy Server 34;
2. Regional RADIUS proxy Server 22;
3. Top level RADIUS proxy Server 11;
4. Regional RADIUS proxy Server 21;
5. National RADIUS proxy Server 31;

Upon a successful authentication, the home network AS sends a message to the AS of the visited network which grants wireless service access right to the user. Depending on the

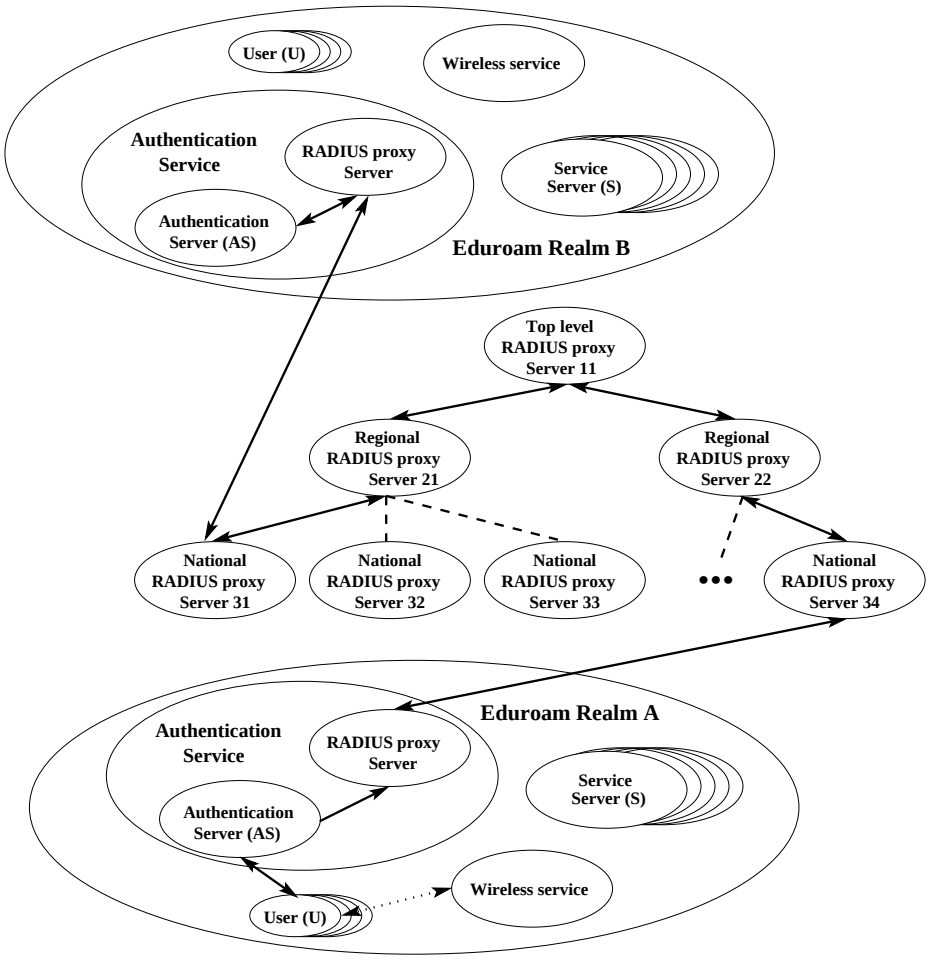


Figure 2.14: Action of Eduroam for a user in a visited network

policy of the visited network, the user may access the wireless service and other network resources of the visited network.

As we mentioned in the preceding section, Eduroam uses 802.1X which encompasses the uses of Extensible Authentication Protocol (EAP). Eduroam supports the use of Transport Layer Security (EAP-TLS), Tunneled Transport Layer Security (EAP-TTLS), and Protected Extensible Authentication Protocol using Challenge Handshake Authentication Protocol (PEAP-EAP-MS-CHAP V2) as the authentication protocols.

As an example, the Eduroam authentication protocol used in University of Southern Queensland is EAP-TTLS. A tunnel is used to provide the USQ home services to a USQ user located in the wirelsss coverage of another Eduroam institution as shown in Figure 2.15.

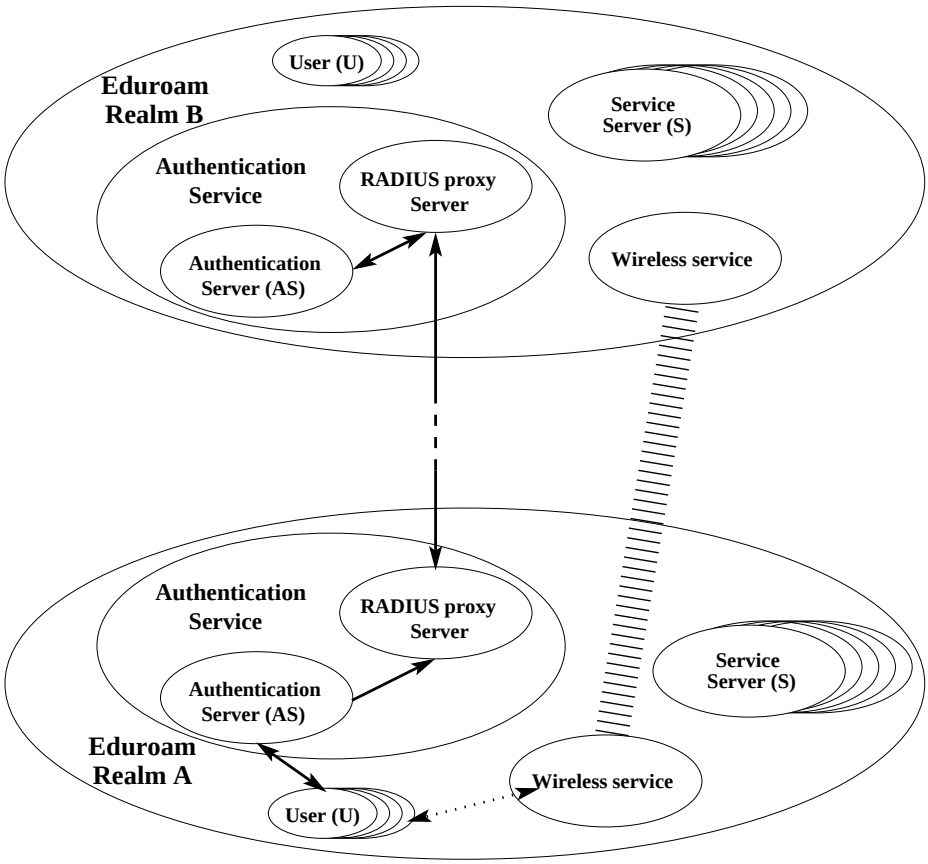


Figure 2.15: Tunneling of Eduroam for a user in a visited network

2.5.1 Drawbacks of Eduroam

There is no doublt that Eduroam has some drawbacks. The drawback include

- Sharing of Wireless access only.
- The user has to move physically within the wireless LAN coverage before he can use the shared service.
- Sharing of other network resources is limited to public services only.
- Institutions have to register to the National RADIUS proxy server
- Enjoying home services only.
- Limited to research and education institutions so far.

2.6 Mobile Host Routing

Mobile Host routing is another technology that supports service sharing.

Mobile Host Routing (MHR) [71] deals with the problem of routing data to a mobile host by establishing a Home Agent (HA) and a Foreign Agent (FA) in a network. When the user is in his home network, he authenticates himself with the HA. When he moves to another network, he becomes a guest and he has to first connect to the FA of the visited network. The FA must request that the guest provides his user information and credentials; these requested information and credentials would be forwarded to the HA of the user's home network as shown in Figure 2.16.

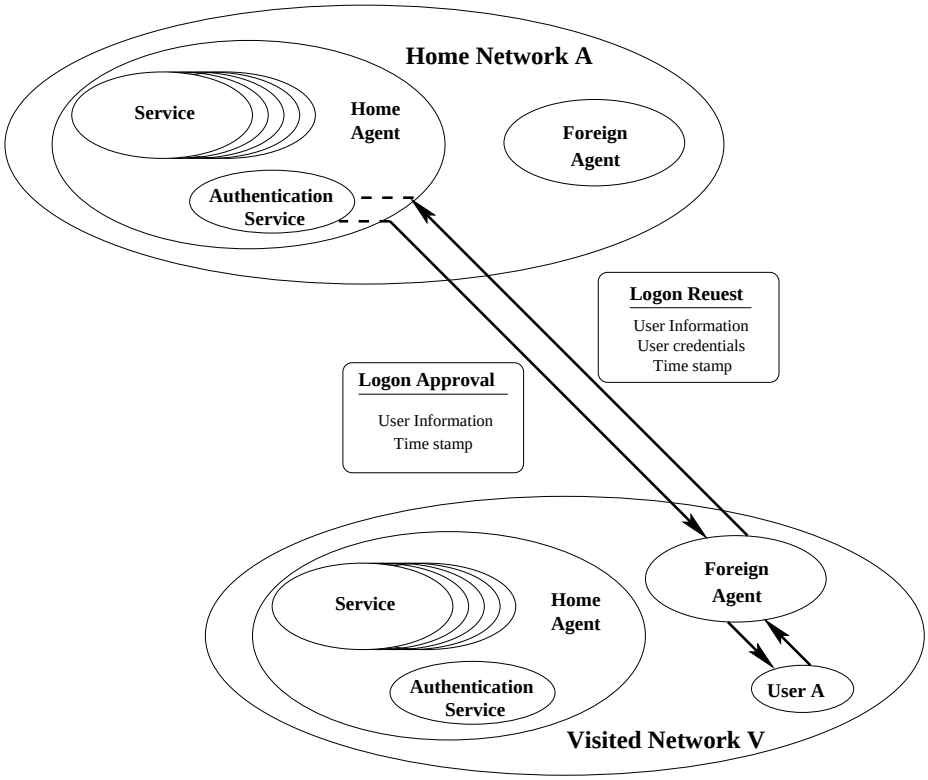


Figure 2.16: Authentication for a mobile user.

The home HA then informs the visited network FA about the result of authentication. If the home network HA is satisfied with the user's identity and credentials, the HA would request that the visited network FA registers the user and provides network access to the user. At this stage, the user uses the visited network FA as if he is using the home network HA as shown in Figure 2.17.

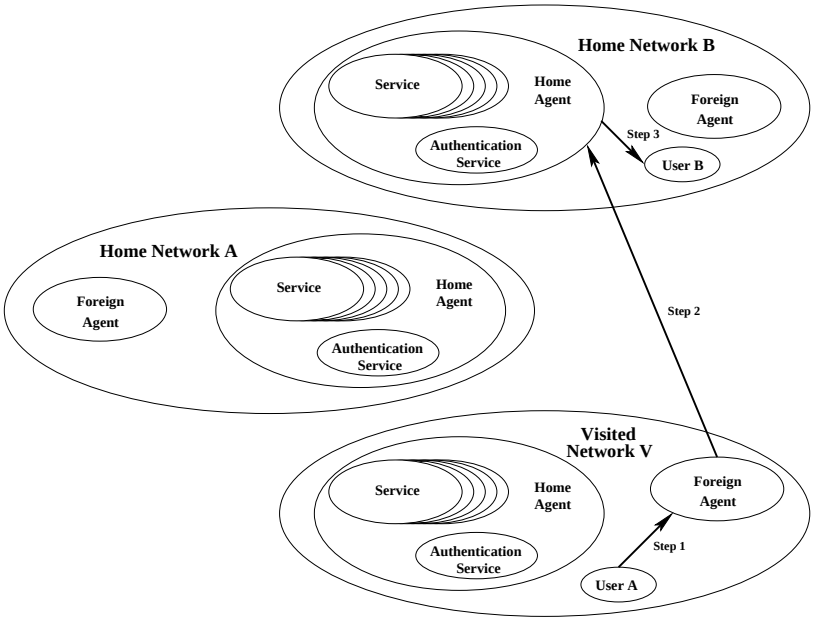


Figure 2.17: Mobile user calling a home network user.

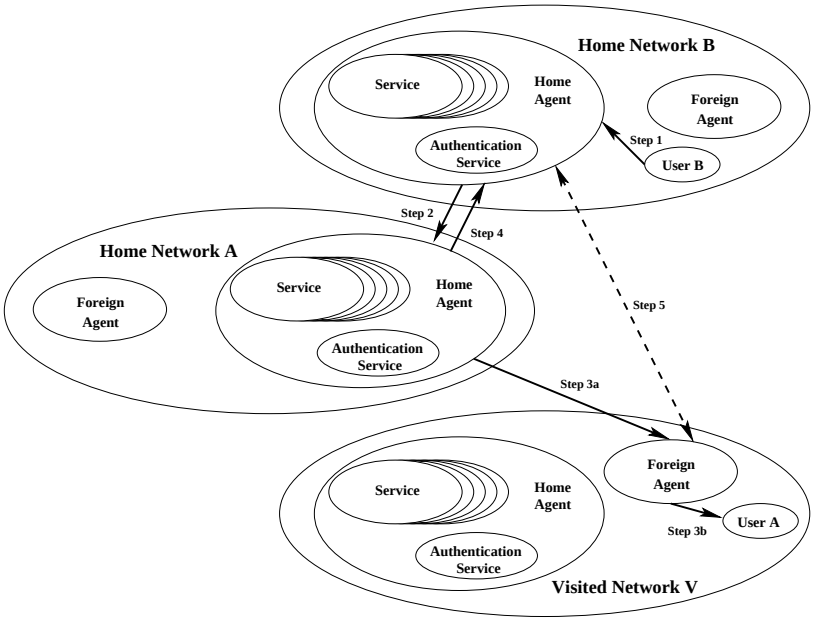


Figure 2.18: Home network user calling mobile user.

Here we look at an example. When some other user U_B wants to send data to the mobile user U_A, the steps taken are as shown in Figure 2.18:

Step 1 The U_B sends the message to his home network HA_B.

Step 2 The HA_B forwards the message to the HA_A of U_A.

Step 3 The HA_A sends the message to the FA_V which is currently handling the data transfer for U_A.

Step 4 The HA_A also informs the HA_B the address of FA_V handling the data transfer for U_A.

Step 5 The HA_B, from this point on, forwards the data from U_B to FA_V, and finally reaching U_A.

The situation is quite similar when a mobile user U_B wishes to reach another mobile user U_A. Only this time the caller U_B is using an FA to send data instead of using his HA as shown in Figure 2.19.

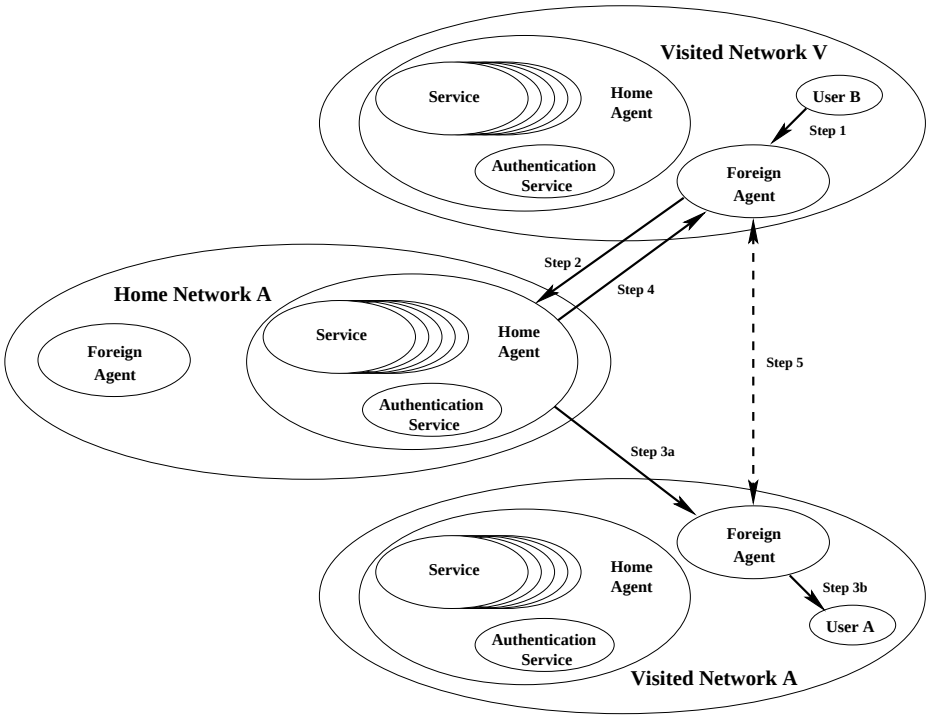


Figure 2.19: Mobile user calling another mobile user.

2.6.1 Drawbacks of Mobile Host Routing

- Dedicated Home Agents and Foreign Agents have to be set up.
- Networks are linked together on a network by network basis.
- Cannot apply to ad hoc set of networks.
- Agents of one network have to keep track of agents in all linked networks.

2.7 OpenID

OpenID [3] authentication provides a way to prove that a user has control over an identifier. It uses standard HTTP(S) requests and responses, so it does not rely on cookies or require any special capabilities on the user side. OpenID authentication enables portable, digital identity in a free and decentralized manner.

The main components in OpenID are listed in Table 2.1

We now take a look at how OpenID operates.

2.7.1 OpenID Account

The whole idea about OpenID authentication is to prove that a user controls an Identifier which, technically, is a URI assigned by an OpenID Provider (OP). So the first step in using OpenID is to set up an account at an OP such as myOpenID.com or VeriSign Labs. Each OP has its own requirements for setting up an OpenID account. Most of them just require a username and password. Note that there is no binding of identity required. Let us look at two examples as follows.

Example 1

myOpenID has an URI myopenid.com. If we choose myOpenID as our OP, and request to use $usqdemo$ as our username, myOpenID assigns an URI $usqdemo.myopenid.com$ as our Identifier.

Example 2

VeriSign Labs has an URI pip.verisignlabs.com. Again if we choose VeriSign Labs as our OP, and request to use $usqdemo$ as our username, VeriSign Labs assigns an URI $usqdemo.pip.verisignlabs.com$ as our Identifier.

We actually are able to acquire as many Identifiers as we like.

[3] Formed in June 2007, OpenID Foundation (http://openid.net/foundation) promotes the OpenID authentication concept. More information on OpenID can be found in http://openid.net/specs/.

Component	Use
Identifier	”http” or ”https” URI or an Extensible Resource Identifier (XRI).
User-Agent (UA)	The end user's Web browser.
Relying Party (RP)	A Web application requiring proof that the user controls an Identifier.
OpenID Provider (OP)	An OpenID authentication server. RP relies on OP for authentication assertion that the end user controls an Identifier.
OP Endpoint URL	The URL which accepts OpenID authentication protocol messages. It can be extracted by performing discovery on the User-Supplied Identifier.
OP Identifier	An Identifier for an OpenID Provider.
User-Supplied Identifier	An Identifier that was presented by the end user to the Relying Party.
Claimed Identifier	An Identifier that the end user claims to own.
OP-Local Identifier	An alternate Identifier for an end user that is local to a particular OP.

Table 2.1: Components of OpenID.

2.7.2 OpenID Authentication

As we knew, Web servers may provide some services to the public and some services to registered members. When we approach web sites for services which requires log-on, usually we have to supply username and password. The web sites keep all their user authentication information and perform authentication themselves. It is common that we may have to create numerous account name and password pairs.

The idea behind OpenID, however, is that for all web sites supporting OpenID, we can simply log-on to all of them using the single Identifier assigned in Section 2.7.1. The web sites do not keep user authentication information and do not do the authentication themselves. Instead they ask the OP, which issues the Identifier, to authenticate the user and to return the authentication results in the form of authentication assertions to them. Based on the authentication assertion, the web sites can proceed with the service or deny access. OpenID authentication process comes in five steps as shown in Figure 2.20. We take $usqdemo.myopenid.com$ as our Identifier in the examples which follow.

Step 1 User Agent (UA) presents User-Supplied Identifier to a Relying Party (RP) (i.e. a web site).

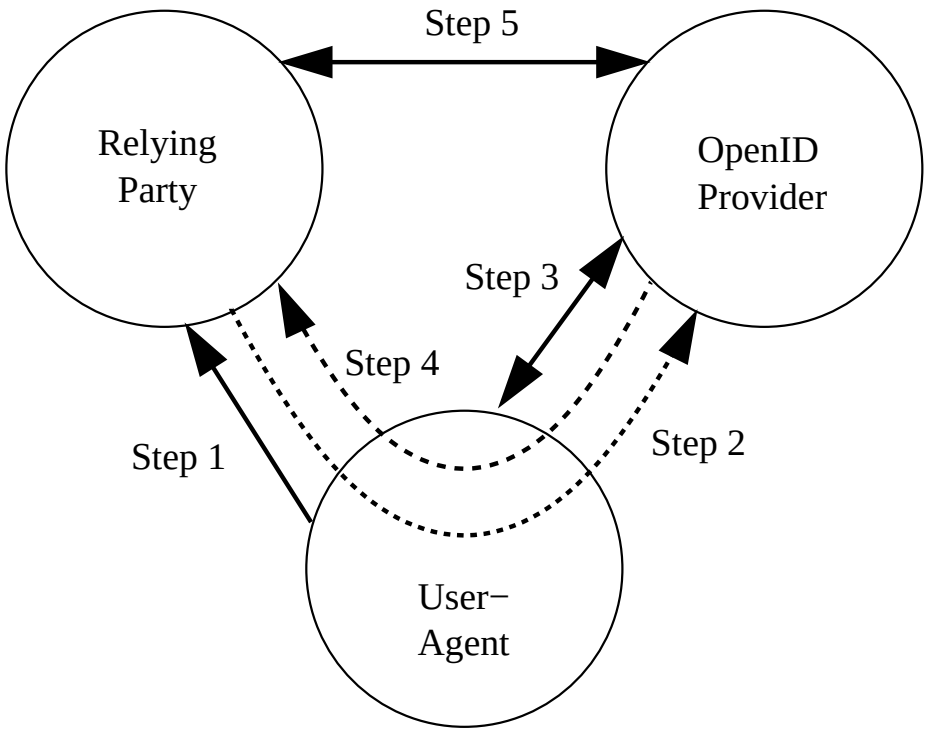


Figure 2.20: Authentication Process for OpenID

Example 3

A web browser is our UA. The web site for WikiTravel (wikitravel.com) supports OpenID for user log-on. We use it as our RP. User-Supplied Identifier is simply our Identifier $usqdemo.myopenid.com$.

Our web browser presents $usqdemo.myopenid.com$ to the log-on page of WikiTravel at $http://wikitravel.org/en/Special:OpenIDLogin$.

Step 2a RP performs discovery on the User-Supplied Identifier to establish the OP Endpoint URL.

Example 4

RP discovers, from the Identifier $usqdemo.myopenid.com$, that the OP and the OP Endpoint URL are $myopenid.com$ and $https://www.myopenid.com/$ respectively.

Step 2b RP redirects UA to OP Endpoint URL with an OpenID Authentication Request.

Example 5

UA (the browser) is redirected to point to $https : //www.myopenid.com/$ with an OpenID Authentication Request. The web page displayed at UA is now the log-on page for $myOpenID.com$, not the log-on page for $WikiTravel.com$ any more.

Step 3 OP authenticates the user.

Example 6

We have to type in the password used when setting up the OpenID account at myOpenID. myOpenID will authenticate the account using the password supplied.

Step 4 OP redirects UA back to RP with an OpenID Authentication Assertion (with approval or rejection).

Example 7

Suppose we type in the correct password. myOpenID simply redirects UA (the browser) back to WikiTravel log-on page with the OpenID Authentication Assertion (approved).

Step 5 RP sends a direct request to OP to verify the OpenID Authentication Assertion.

Example 8

RP checks

- The value of "openid.return_to" in the Authentication Assertion matches the URL of the current request.
- Discovered information such as OP Endpoint URL matches the information in the assertion.
- An assertion has not yet been accepted from this OP with the same value for "openid.response_nonce" in the returned assertion.
- The signature on the assertion is valid and all fields that are required to be signed are signed.

If the OpenID Authentication Assertion is verified successfully, the application server can start to provide service to the user.

2.7.3 Drawbacks of OpenID

OpenID has a number of major security concerns. We regard these security concerns as drawbacks.

- If DNS resolution or the transport layer is compromised, the attacker can impersonate the OP. Not only can it get the user's authentication information, it can issue its own authentication decisions.
- A special type of man-in-the-middle attack is one where the Relying Party is a rogue party acting as a Man in the Middle. The RP would perform discovery on the End User's Claimed Identifier and instead of redirecting the $UserAgent$ to the OP, would instead proxy the OP through itself. This would thus allow the RP to capture credentials the End User provides to the OP. As OpenID allows everyone to become an OP, OpenID is particularly vulnerable to this special type of Man-in-the-Middle attack.
- User-Agents in OpenID is primarily common Web browsers. Web browsers or their hosts may be infected with spyware or other malicious software, which casts doubt on $AuthenticationAssertion$ received by RP.
- OpenID is designed with web browsing in mind. When setting up an OpenID, the OP does not require any form of identity binding as in the case when you apply for a digital certificate from a Certificate Authority such as VeriSign. So what OpenID does is to confirm the ownership of the Identifier you presented, but what that Identifier represents is up to the OP to determine.

2.8 Chapter Summary

We have reviewed some of the current technologies for sharing services. We also detailed their workings and pointed out the merits and drawbacks. In the subsequent chapters, we present our solution for supporting secure services on dynamic aggregation of heterogeneous networks - the Service Network Graph (SNG) together with Dynamic Password ($DPass$) and its associate Key Exchange using Dynamic Password ($KEDP$).

Chapter 3

Service Network Graph

In this Chapter, we present an approach to service sharing using Service Network Graph [42, 35, 34]. Service Network Graph (SNG) is based on autonomous networks interconnected using encrypted channels. It begins with an introduction of the physical structure of Service Network Graph and the Distributed Network Service Authentication ($DNSA$) protocol. We then discuss how SNG uses Service Path ($SPath$) to hold the service access information. In addition, we also discuss how services are shared using SNG.

3.1 Introducing Service Network Graph

Targeting the draw backs of the existing service sharing technologies, we integrated the concepts of Authentication Delegation, Authentication Propagation into Service Network Graph.

Suppose a network N_A has delegated its authentication authority to another network N_B. Whenever N_A says user U_A is authenticated, then N_B will also say U_A is authenticated. The authentication delegation property between two networks is transitive. In other words, whenever NA delegates its authentication authority to N_B and N_B delegates its authentication authority to N_C, then we can say that N_A also delegates its authentication authority to N_C. The transitivity property of Authentication Delegation is named as Authentication Propagation.

Using Authentication Delegation and Authentication Propagation, a user can always be authenticated by his home network and can enjoy services provided his own home network as well as services shared by other networks wherever the user is in the SNG. We do not have to worry about the identity of the user as we do in OpendID, and at the same time, this will eliminate the problem of using the same authentication systems such as the ticket system in Kerberos. KDC and TGS are no longer needed. The bottleneck problem of hierarchical arrangement of cross realm KDC does not exist anymore. All services, including wireless network access service, can be shared which is significantly different

to Eduroam with which user can only enjoy shared wireless network access and services provided by his own home network. Furthermore, Authentication Propagation empowers Authentication Delegation for ad hoc aggregation of heterogeneous networks. No more mesh configuration of networks as found in Mobile Host Routing, which will greatly enhance scalability.

Another distinct feature of the SNG is that a list of all services available to a user is maintained by a Service Listing Server. Queries about service available and updating the service list is a simple task. We no long have to maintain the service list ourselves.

In the next section, we will briefly introduce the elements of a typical autonomous network which is the building block of a dynamic aggregation of networks.

3.2 Autonomous Networks

As discussed in Chapter 1, an autonomous network is a network having its own user administrative domain, policies, and authentication scheme totally independent of other autonomous networks. Furthermore, an autonomous network can provide the basic functions such as authentication, service listing and some services to a number of registered local users. Those basic functions are usually provided by a set of individual servers or by a machine which runs multiple servers, as shown in Figure 3.1.

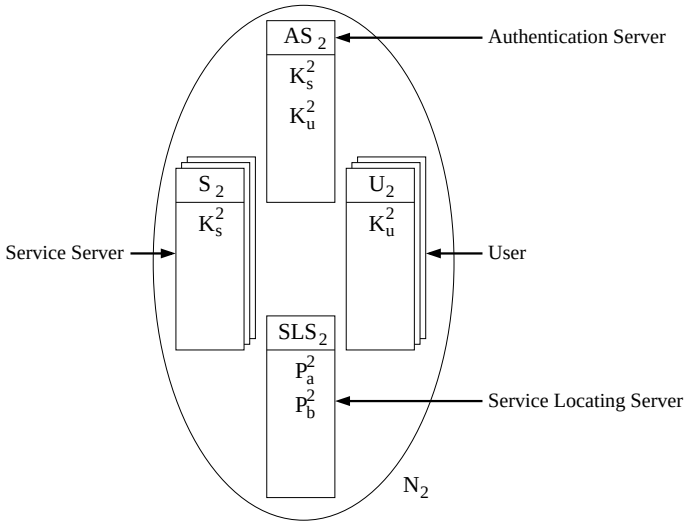


Figure 3.1: A typical autonomous network.

For instance, individual servers include:

- An Authentication Server (AS) which authenticates registered local users.

- A Service Server S which provides services.
- A Service Locating Server (SLS) which stores information about local services and shared services in the form of Service Paths.
- User (U) who requests for services.

An autonomous network N_1 can surely provide its own services to its registered local users. If another autonomous network provides a service to users of N_1, the service is called a shared service. When a registered user U_1 requests a service, the SLS_1 responds with a list of services available, locally or shared. After U_1 selects a service, the AS_1 then authenticates U_1 and S_1 provides the requested service.

3.3 Service Network Graph

A Service Network Graph (SNG) is a dynamic aggregation of some autonomous networks whose authentication servers are interconnected by $encrypted\ channels$. The encrypted channels are communication channels in which traffic are encrypted. Within an SNG, each pair of authentication servers share a common secret key and the secret key is used to establish the encrypted channel between the pair of authentication servers. All communications among the AS are encrypted using the common secret keys between the participating ASs.

3.3.1 Authentication Delegation

In an SNG, ASs are virtually linked together using the encrypted channels. The linkages enable $Authentication\ Delegation$. Let us look at how an autonomous network makes authentication delegation with an SNG.

No doubt autonomous networks authenticate their own registered users. When an autonomous network N_A delegates its authentication authority to another network N_B, then whenever N_B declares U_B as authenticated and U_B has the right to use the services which N_B provides, N_A also provides its shared services to U_B as if U_B was authenticated by N_A also. We refer to N_A as the $delegatornetwork$ and N_B is the $delegateenetwork$. N_B is said to have attached to N_A. In other words, when U_B is authenticated in N_B, he can use the services provided by N_B as well as the services provided by N_A. We say that N_A shared its services with N_B.

When a network delegates its authentication authority to another network, our concern is the authentication process. As the actual authentication is performed by the authentication server AS, we use AS of a network to represent the network itself.

When N_A delegates its authentication authority to N_B, AS_A shares a secret key with AS_B. The shared key is used to encrypt all the traffic between the two ASs. Since the

encrypted channels are self authenticating (for more details, see Chapter 5), when AS_B sends an authentication result to AS_A through an encrypted channel established with the shared key, AS_A can simply accept the authentication result as his own authentication result for accepting or rejecting a service request. The authentication delegation relationship is represented by a single arrow pointing from the delegatee network to the delegator network and is a one way relationship.

Obviously, AS_B can also share a secret key with AS_A thereby delegates its authentication authority to AS_A and forms a mutually linked relationship. Mutual linking is therefore a dual-way relationship and is represented by a double head arrow.

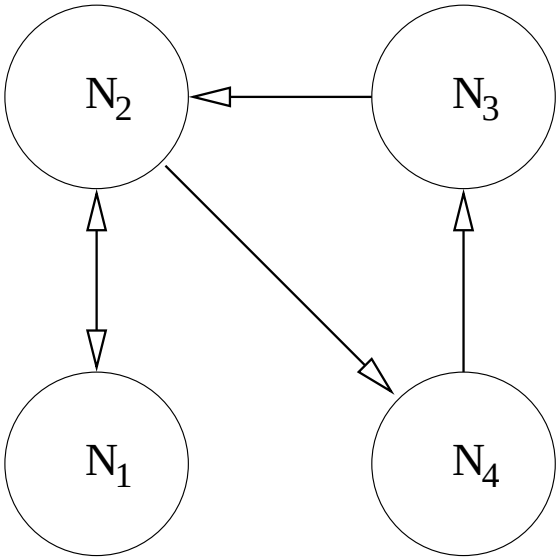


Figure 3.2: A Service Network Graph

A *Service Network Graph* can be used as a representation of many autonomous networks attached to or mutually linked with each other, in which an attachment is represented by a single-ended arrow and a mutual-link is represented by a double-end arrow as depicted in Figure 3.2.

From this moment on, we use SNG to refer to a dynamic aggregation of autonomous networks with both one way type and mutual type of links. We illustrate this concept with the following example.

Example In Figure 3.2, N_1, N_2, N_3 and N_4 are four autonomous networks where

- N_1 attaches to N_2.
  N_2 delegates its authentication authority to N_1.
  After N_1 authenticates user U_1, N_2 is happy to provide services to U_1.

- N_2 attaches to N_1 and N_4.
  N_1 delegates its authentication authority to N_2.

N_4 delegates its authentication authority to N_2.
After N_2 authenticates user U_2, N_1 and N_4 are happy to provide services to U_2.

- N_3 attaches to N_2.
  N_2 delegates its authentication authority to N_3.
  After N_3 authenticates user U_3, N_2 is happy to provide services to U_3.

- N_4 attaches to N_3.
  N_3 delegates its authentication authority to N_4.
  After N_4 authenticates user U_4, N_3 is happy to provide services to U_4.

3.3.2 The Sample Service Network Graph

In an SNG, autonomous networks can be identified with their IP network addresses and in particular, AS's IP addresses. We refer to the sample SNG shown in Figure 3.2 for our discussions. Some useful IP addresses are listed in Table 3.1.

Network	**Authentication Server**	**IP address**
N_1	AS_1	200.200.1.2
N_2	AS_2	200.200.2.2
N_3	AS_3	200.200.3.2
N_4	AS_4	200.200.4.2

Table 3.1: IP addresses of authentication servers in sample SNG.

With the SNG physical framework in place, we now proceed to look at the concept of Service Path and Distributed Networks Service Authentication protocol for SNG.

3.4 Service Path

In this section, we look at the concept of Service Path ($SPath$). In the SNG shown in Figure 3.2 on page 40, when N_4 shares its service with N_2, AS_4 shares a secret key with AS_2 and SLS_4 shares its service list in the form of $SPath$ with SLS_2. $SPath$s specify the requested service, which server provides the service, the access path, cost and the condition of sharing. Service Path ($SPath$) was originally defined in [42] as shown below. Further properties of $SPath$ are discussed in [41, 38, 39].

```
<SOpt:SAPath/SverN/SviceN>:<Cost>
```

where

SOpt (Share Option) specifies the condition of sharing, such as free (F) to share with other networks or restricted (R) for further sharing.

SAPath (Service Access Path) is the path from the home network AS to the service providing server $SverN$. To ensure that no cycle exists in $SAPath$ field, whenever a network acquires a $SPath$, it should check that its own network address does not appear in the $SAPath$ field. If the network address appears in the $SAPath$ field, the $SPath$ should be discarded.

SverN (Service Providing Network) is the name of the service providing server.

SviceN (Network Service) is the name of the requested service.

cost is the cost for using this service.

A typical $SPath$ looks like:

```
<F:./200.200.4.2/200.200.3.2/Server3/Time>:<4>
```

where

SOpt = F (free to share with others)

SAPath = ./200.200.4.2/200.200.3.2/ (from current network N_2 to N_4 and then N_3)

SverN = Server3 (name of the service providing server)

SviceN = Time (name of the requested service)

cost = 4 (cost for using this service)

3.4.1 Share Options

The first field of an $SPath$ is Share Option ($SOpt$), which indicates conditions we impose on the service. For simplicity, we define options $Free$ (F) and $Restricted$ (R). If an $SPath$ has F as the value of $SOpt$, the autonomous network shares the service to other networks when they attach to it. On the other hand, services which has R as the value of $SOpt$, the service would not be shared when other networks are attached to it.

Other options for $SOpt$ including role dependent options and time dependent options can also be defined as follows:

A Accessible only to adult users.
Service example: Streaming of X-rated video.
This option serves as a legal and moral filter for under-aged service users.
The home network may reject the authentication if the user is not an adult.

N Accessible only to network administrators.
Service example: Accessing network information and statistics.
Network administrators need to have information about neighboring networks to optimize the performance of their network. They use services which can provide information and statistics about connected networks so that optimization of their networks can be done efficiently and effectively.
Obviously these services are only available to the network administrators and not for general users.

O Accessible only during, say, 9:00 am - 5:00 pm.
Service example: Placing of orders for Stock Exchange.
Some companies only allow placing of orders within the normal trading hours. Outside the business hours, such services are not available.

H Accessible only during non-office hours.
Service example: Backing up a database.
Some services take up a lot of resources. It is reasonable to limit such services to non-office hours. In the case of backing up business data, non-office hour operation can help to ensure completeness and integrity of the data backed up.

3.4.2 Service Access Path

The second field of an $SPath$ is the Service Access Path ($SAPath$). The $SAPath$ indicates the network path to access the nominated service. We use the IP address of AS for the address for the autonomous network because we must access an AS before we can access a service.

Still looking at the SNG on page 40, a service provided by a local server has $./$ as the $SAPath$. When the local service of N_4 is shared with another network, say N_2, SLS_4 replaces $./$ with $N_4/$ before the $SPath$ is shared with SLS_2. On receiving the shared $SPath$, AS_2 then pre-pends $./$ to all the $SPath$ received. The process repeats when $SPath$s are shared further. Each sharing results in an network IP added after the $./$ in $SAPath$.

For example, when a service from N_3 is shared with N_4, $N2$ and N_1 in turn, the formats of $SAPath$ at various stage of sharing are listed in Table 3.2.

It is important to note that when a user requests a service list, the list must come from the SLS of his home network. Hence the $./$ in an $SAPath$ represents the home network of a user.

SAPath (in a network)	SAPath (when shared)
$./$ (in N_3)	$N_3/$
$./N_3/$ (in N_4)	$N_4/N_3/$
$./N_4/N_3/$ (in N_2)	$N_2/N_4/N_3/$
$./N_2/N_4/N_3/$ (in N_1)	$N_1/N_2/N_4/N_3/$

Table 3.2: Network paths to authentication server AS_3 in sample SNG.

3.4.3 Cost

The last field of $SPath$ is $Cost$. $Cost$ is in fact a measure of resources required for the service. It is used for auditing and charging purposes. Various cost metrics can be used. Commonly accepted metrics include pecuniary credits, processing time and transmission time, and bandwidth. Cost may change as the service is shared further and further away from its service providing network.

In the next section, we present a protocol which is used by SNG to authenticate users and provide services.

3.5 Distributed Networks Service Authentication Protocol

Distributed Networks Service Authentication ($DNSA$) protocol was proposed in [35, 34]. An SNG uses the $DNSA$ to authenticate and provide services to users, irrespective of the network within SNG.

The $DNSA$ protocol has two distinct operation modes. One is the *Network Participation mode* (NP mode) in which a network links to another network in an SNG. Another is the *User Service mode* (US mode) in which a user accesses a local or shared service.

3.5.1 Network Participation Mode

We refer $DNSA$ is in its Network Participation (NP) mode when autonomous networks joins an SNG. This mode is used to establish the encryption channels of an SNG.

Let us assume N_2 and N_4 are two separate autonomous networks as shown in Figure 3.3. Upon receiving the request from N_2 to attach directly to N_4, AS_4 generates and sends a secret key, $authentication\ token\ key$ (ATK_a^4) to AS_2. AS_4 accepts all service requests encrypted with ATK_a^4 as if the user was authenticated by AS_4 (see Section 3.5.2 and Chapter 5 for details). Hence we called the sharing of ATK_a^4 as $Authentication\ Delegation$.

SLS_4 also sends information about services to be shared with N_2 to SLS_2 in the form of a Service Path ($SPath$). The sharing of services makes N_2 capable of offering services

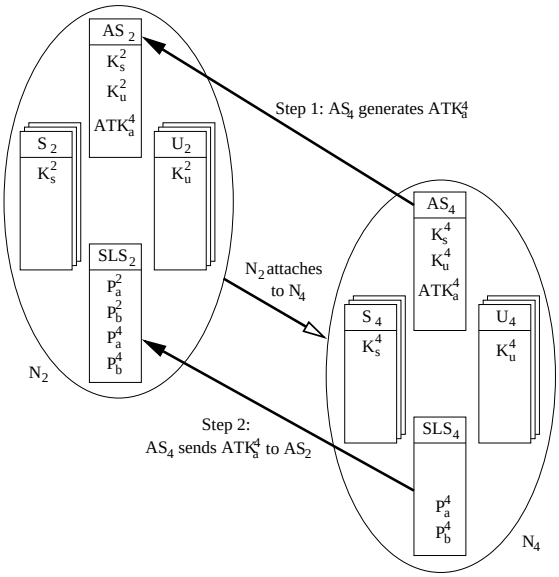


Figure 3.3: Attaching one network to another network

provided by N_4 as well as its own. Both AS_2 and SLS_2 acknowledge the receipt of information from AS_4 and SLS_4 respectively. Both AS_2 and AS_4 now have the key ATK_a^4 which was generated by AS_4. N_4 has delegated the authentication authority to N_2. Note that the reverse is not true.

Similarly, N_4 can also request to attach to N_2 and form a mutual link with N_2. This time the key ATK_a^2 is generated by AS_2.

The number of ATKs that an AS holds is equal to the number of links the AS has established. The number of ATKs that the AS generates is the same as the number of networks attached to it; and the number of ATKs the AS receives is the same as the number of network it attaches to.

3.5.2 User Service Mode

The second mode $DNSA$ is the $User\ Service\ (US)$ mode. When a local user U_2 in N_2 requests services, U_2 has to query its service locating server SLS_2 for services available. SLS_2 returns a list of services available on which each service is listed in the form of $SPath$ that includes the service access network path, server name, service name and cost metrics to the user. In Figure 3.3, the $SPath$s are shown as P_a^2, P_b^2, P_a^4 and P_b^4 in SLS_2. Here P_Y^X represents the $SPath$ of service Y provided by network N_X.

If the user U_2 is satisfied with the service, U_2 has to authenticate to AS_2 and submit the selected service path which AS_2 uses to retrieve service information from the server.

The following are four possible scenarios of service request.

Scenario 1: Local Service Request

Let us look at the simplest scenario. From the $SAPath$ of a requested service $SPath$, $DNSA$ can determine if the service is a local service or not. For instance, if the $SAPath$ is just $./$, then it is a local service. Otherwise, it is a remote service.

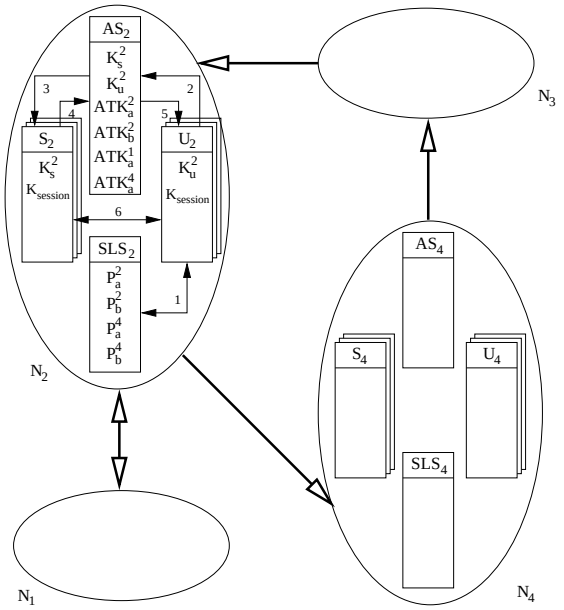


Figure 3.4: User requesting a local service

For a local service request, such as $< F : ./S2/Path >:< 1 >$, the steps for providing service as shown in Figure 3.4 are:

Step 1 U_2 requests a list of services available.

Step 2 U_2 presents authentication information and $SPath$ of selected service to AS_2 encrypted using user shared secret key K_u^2.

Step 3 If authentication is successful, AS_2 generates a session key $K_{session}$. AS_2 then encrypts $K_{session}$ using server shared secret key K_s^2 of server S_2 and sends it to S_2.

Step 4 S_2 acknowledges the session key $K_{session}$ and returns AS_2 all service information for the request encrypted using server shared secret key K_s^2.

Step 5 AS_2 relays the service information and $K_{session}$ to U_2 using user shared secret key K_u^2.

Step 6 U_2 accesses S_2 for services using the session key $K_{session}$.

Scenario 2: Remote Service Request

If the $SPath$ indicates the service is a remote service as in: $< F : ./N_4/S_4/Time >$: $< 2 >$, the steps for service accessing is show in Figure 3.5.

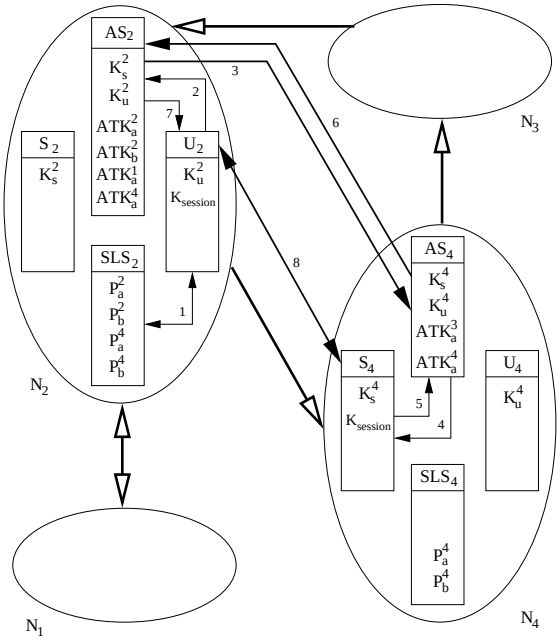


Figure 3.5: User requesting a shared service

Step 1 U_2 requests a list of services available.

Step 2 U_2 presents authentication information and $SPath$ of selected service to AS_2 encrypted using user shared secret key K_u^2.

Step 3 If authentication is successful, AS_2 retrieves the authentication token key of AS_4, ATK_a^4 and uses it to encrypt the session key $K_{session}$ and $SPath$ of requested service. $K_{session}$ and $SPath$ forms an authentication token. The encrypted token is sent to AS_4.

Step 4 AS_4 then retrieves the authentication token and encrypts it again using the shared secret key K_s^4 of server S_4 and sends it to S_4.

Step 5 S_4 acknowledges the session key $K_{session}$ and returns AS_4 all service information for the request encrypted using server shared secret key K_s^4.

Step 6 AS_4 relays the service information to AS_2 using authentication token key ATK_a^4.

Step 7 AS_2 relays the service information and $K_{session}$ to U_2 using user shared secret key K_u^2.

Step 8 U_2 accesses S_4 for services using the session key $K_{session}$.

In the case when the network path in the $SPath$ includes more than one network other than the current network, such as $< F : ./N_4/N_3/S_3/Date >:< 3 >$, the service is still a remote service but N_2 is not attached directly to the service providing network (N_3). The steps for accessing service is similar but extra steps (3a and 5a) are needed to relay the authentication token to AS_3 as shown in Figure 3.6.

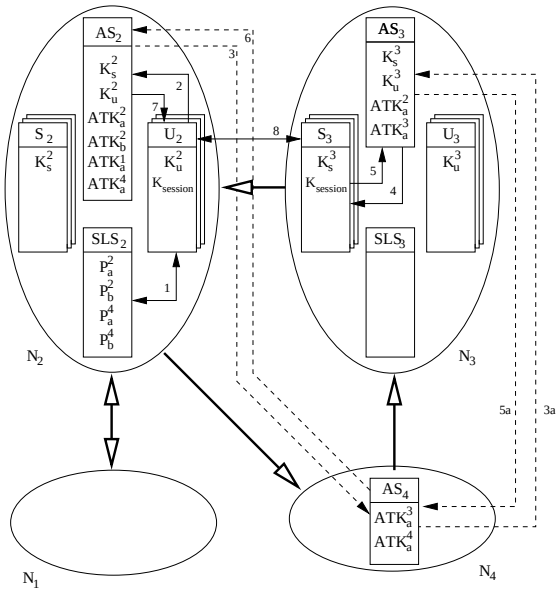


Figure 3.6: User requesting a shared service

Step 1 U_2 requests a list of services available.

Step 2 U_2 presents authentication information and $SPath$ of selected service to AS_2 encrypted using user shared secret key K_u^2.

Step 3 If authentication is successful, AS_2 retrieves the authentication token key of AS_4, ATK_a^4 and uses the key to encrypt the authentication token, and sends the encrypted token to AS_4.

Step 3a AS_4 retrieves the authentication token key of AS_3, ATK_a^3; uses the key to encrypt the authentication token; and sends the encrypted token to AS_3.

Step 4 AS_3 encrypts $K_{session}$ using server shared secrete key K_s^3 of server S_3 and send it to S_3.

Step 5 S_3 acknowledges the session key $K_{session}$ and returns AS_3 all service information for the request encrypted using server shared secret key K_s^3.

Step 5a AS_3 relays the service information to AS_4 using authentication token key ATK_a^3.

Step 6 AS_4 relays the service information to AS_2 using authentication token key ATK_a^4.

Step 7 AS_2 relays the service information and $K_{session}$ to U_2 using user shared secret key K_u^2.

Step 8 U_2 accesses S_3 for services using the session key $K_{session}$.

The extra steps show that AS_4 now acts like a middle man passing authentication tokens and service information onto the next hop along the network path in an $SPath$.

Scenario 3: Mobile Service Request

Suppose U_2 is a registered user of N_2 currently located in N_1. If the U_2 requests service $< F : ./N_4/N_3/S_3/Date >:< 3 >$ while $U - 2$ is in N_1, it is the case of mobile service request [36] as shown in Figure 3.7.

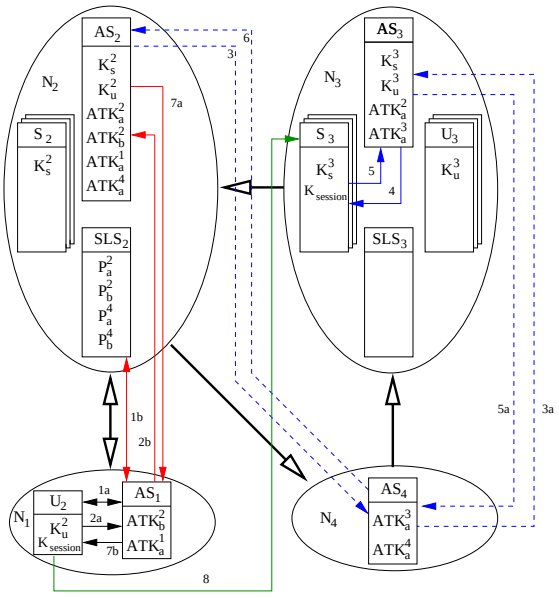


Figure 3.7: Mobile user requesting a shared service

The case is the same as remote service access except extra steps (1a, 1b, 2a, 2b, 7a and 7b) are required for relaying authentication and service information between the visited network and the home network:

Step 1a U_2 approaches AS_1 and requests a list of services available.

Step 1b AS_1 forwards the service list request to the SLS_2 in N_2. Upon receiving the service list from SLS_2, AS_1 forwards the service list to U_2.

Step 2a U_2 selects a service and send the authentication token to AS_1 using K_u^2.

Step 2b AS_1 forwards the authentication token and service request to AS_2 in N_2 using ATK_a^2.

Step 3 If authentication is successful, AS_2 retrieves the authentication token key of AS_4, ATK_a^4 and uses it to encrypt the session key $K_{session}$ and authentication token. The encrypted message is sent to AS_4.

Step 3a AS_4 retrieves the authentication token key of AS_3, ATK_a^3 and uses it to encrypt the session key $K_{session}$ and authentication token. The encrypted message is sent to AS_3.

Step 4 AS_3 then encrypts $K_{session}$ using server shared secrete key K_s^3 of server S_3 and sends it to S_3.

Step 5 S_3 acknowledges the session key $K_{session}$ and returns AS_3 with all service information for the request encrypted using server shared secret key K_s^3.

Step 5a AS_3 relays the service information to AS_4 using authentication token key ATK_a^3.

Step 6 AS_4 relays the service information to AS_2 using authentication token key ATK_a^4.

Step 7a AS_2 encrypts the service information and $K_{session}$ with K_u^2 and relays the encrypted service information and $K_{session}$ to AS_1 using the authentication token key ATK_a^2.

Step 7b AS_1 forwards the encrypted service information and session key $K_{session}$ to U_2.

Step 8 U_2 retrieves service information and $K_{session}$; and accesses S_3 for services using the session key $K_{session}$.

In Steps 1b and 7a, AS_1 and AS_2 are required to exchange messages. The access path is not recorded in $SAPath$ of an $SPath$. This information can be obtained in an SNG topology file.

The above sequence of steps works fine for transit mobile users. For non-transit mobile users, it would be better if a temporary account can be set up in N_1 for U_2. With an account on N_1, the authentication token does not need to be forwarded all the way to N_2 for authentication. Instead, authentication can be done locally.

When N_1 attaches to N_2, due to that AS_2 has delegated authentication authority to AS_1, it is acceptable that AS_1 takes over the authentication duty of AS_2 for this case even though U_2 is registered in AS_2.

In order to take over the authentication duty, AS_1 has to set up an account for U_2. Sharing a secret key between U_2 and AS_1 does not prove the identity of U_2. U_2 has to bind his identity to the shared secret. This may involve complex legal processes, lengthy period of time and plenty of documents. To bind the shared secret with the identity of U_2, AS_1 may choose an approach similar to X.509 digital certificates [2]. AS_1 can set up a transient account for U_2 by sharing a secret key with U_2 and entrusting the home authentication server AS_2 to bind the identity with the shared secret.

Entrusting AS_2 for identity binding is equivalent to delegating authentication authority to AS_2. AS_1 can assume that AS_2 endorses the identity of U_2 when AS_2 replies with a successful log-on when U_2 makes his first service request in N_1. Hence the identity of U_2 in N_1 is bound to the identity of U_2 in N_2 after to the first successful log-on and N_1 can now set up an account for U_2.

Note that setting up new accounts in N_1 (N_2 delegates authentication authority to N_1) and rely on the home network for identity binding (N_1 delegates authentication authority to N_2) is only possible when AS_1 and AS_2 have mutual authentication delegation as indicated by the double-end arrow shown in Figure 3.7.

3.6 Authentication Propagation

Complementary to authentication delegation, $Authentication\ Propagation$ is the relay of authentication request / reply of a user from the visited network to the home network [41].

For example, considering a case of mobile service request, U_2 from N_2 is now located in N_1. When U_1 requests services from N_2, AS_1 forwards the request to SLS_2 which returns a list of services available to U_2. AS_1 uses the information it acquired when it joins the SNG to pass on the message. When N_1 attaches to N_2, SLS_2 shares service information in the form of $SPath$s. The $SAPath$ field of $SPath$ tells AS_1 which network AS_1 can access via AS_2. It is simply all the network addresses in an $SAPath$ other than the $./$ and the first network address which must be the address of AS_2.

Suppose one of the shared $SPath$ has an $SAPath$

$./200.200.2.2/200.200.4.2/200.200.3.2/$ (i.e. $./N_2/N_4/N_3/$).

AS_1 reckons this is equivalent to three pieces of information:

- AS_2 is directly linked;
- Reach AS_3 via AS_2;

- Reach AS_4 via AS_2.

The path information is recorded in an SNG topology file for AS_1 as:

```
200.200.2.2 200.200.2.2
200.200.3.2 200.200.2.2
200.200.4.2 200.200.2.2
```

By going through all $SPath$ acquired, AS_1 can work out the next hop addresses of all AS accessible.

Similarly, SNG topology file for AS_2 is:

```
200.200.1.2 200.200.1.2
200.200.3.2 200.200.4.2
200.200.4.2 200.200.4.2
```

When AS forwards a service request, it always consults its SNG topology file for the next hop address.

3.7 Changes in SNG Configuration

An SNG is formed based on authentication delegations, either one-way or mutual. Changes in SNG configuration means that authentication delegation relationships in an SNG are changed [34]. This will affect the services shared among the directly linked and indirectly linked autonomous networks.

With reference to the SNG shown in Figure 3.2, a local $Date$ service in N_3 is represented by different $SPath$s in different networks when shared:

$$\begin{array}{ll} N_3 & <./S_3/Date>:<1> \\ N_4 & <./N_3/S_3/Date>:<2> \\ N_2 & <./N_4/N_3/S_3/Date>:<3> \\ N_1 & <./N_2/N_4/N_3/S_3/Date>:<4> \end{array}$$

The service is made available to N_4 when N_4 attaches to N_3. Similarly, the service is made available when N_2 and N_1 attach to N_4 and N_2 respectively. Except for the case of N_4 which directly links with N_3, all other $SPath$s for the shared $Date$ service contain the $.../N_4/N_3/...$ pattern. This pattern persists in all $SPath$s sharing $Date$ service via N_4.

Suppose N_4 wants to detach from N_3 for some reason. N_4 detaches from N_3 simply by removing the ATK_a^3 from its AS_4 and notifying N_3 about the deletion. N_3 also deletes ATK_a^3 in response to this notification. All service requests with $SPath$s having the pattern $.../N_4/N_3/...$ would be denied by AS_3 and AS_4. N_4 then broadcasts to all networks which are attached to it that all services from N_3 via N_4 are no longer available. Effectively N_4 is

telling all attached networks that $SPath$s with $.../N_4/N_3/...$ is no longer valid and should be removed from their service lists.

When a network receives such a notice, it first checks its service list. If $SPath$s with $.../N_4/N_3/...$ exist, it deletes the $SPath$s and passes the notice further on to all networks attached to it. If it does not have any $SPath$ with $.../N_4/N_3/...$, either the network does not share any N_3 service made available via N_4 or the message reaches the network via a loop. In this case, the network which has received the message simply discards the message.

After all the networks discard the message to delete an $SPath$, the SNG is ready for normal service again.

The case is pretty much similar when N_3 does not want to share services with N_4 anymore. AS_3 simply informs AS_4 that the services from N_3 are no longer available and removes the ATK_a^3 it shared with N_4. N_4 then proceeds as if it initiates the detaching action.

3.8 Changes in User Authorization

Whenever user authorization is changed, it is recorded somewhere in the identity repository of his home network. The changes in authorization would be problematic when the user is still accessing services while the changes occurred [35].

Suppose U_2 is a registered user of N_2. He has requested and authorized to access a local service ($Path$) and a remote service ($Date$):
$< ./S_2/Path >:< 1 >$
$< ./N_4/N_3/S_3/Date >:< 1 >$
Suppose the right to use $Path$ service and $Date$ service are removed from U_2 while he is accessing both services.

When AS_2 is alert to any authorization changes for U_2, it makes some changes in its own identity repository. AS_2 pushes the authorization changes to the server concerned as shown in the service path using a *revocation token*. The revocation token can be forwarded further to another network as indicated by the $SPath$. The content of a revocation token is similar to that of an authentication token. Only this time the service is flagged as $deny$. The steps for revoking the $Date$ service are shown in Figure 3.8:

Step 1 AS_2 sends revocation token to AS_4.

Step 2 AS_4 forwards the revocation token to AS_3.

Step 3 AS_3 passes the revocation token to S_3.

On receiving such an authorization change message, the server terminates the service according to predefined access control policy.

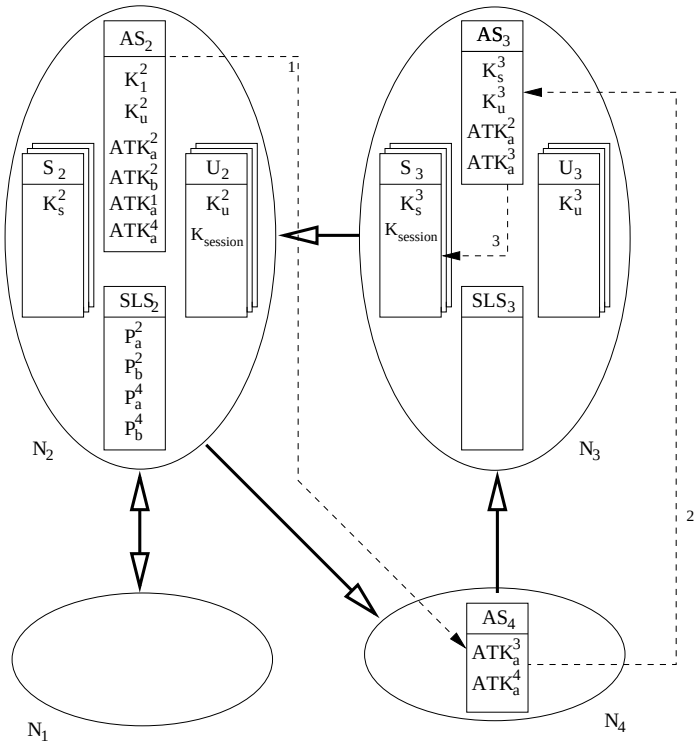


Figure 3.8: Changes in user authorization is pushed to server

3.9 Chapter Summary

In this chapter, we introduced SNG as a dynamic aggregation of autonomous networks. The networks are linked together using the encrypted channels which indicates authentication delegation is in place between the networks. We also presented $DNSA$, the protocol for authenticating a local user or mobile user requesting a local or remote service. In the next Chapter, we continue our discussion on $SPath$ optimization for efficiency and scalability.

Chapter 4

Efficiency and Scalability of Service Path

In Chapter 3, the concept of service and SNG were introduced; and $DNSA$ was discussed. In addition, Service Path ($SPath$) was also defined. The $SPath$ was mainly used to hold the information about a service which is shared between autonomous networks in an SNG. We noticed that each time when a service is shared, the $SAPath$ field of an $SPath$ increase by one IP address. The problem is that the overhead for establishing a service increases as $SAPath$ increases and poses a potential constraint for the scalability of $SPath$ and SNG. In this Chapter, we look at the efficiency and scalability of $SPath$ by presenting a method for optimizing $SPath$ [39].

4.1 Overview

We re-visit Figure 3.2 on page 40. When a service from N_3 is shared with N_4, N_2 and N_1 in turn, the $SAPath$ field of an $SPath$ grows each time it is shared. If $SPath$ and indeed SNG are to be used for ad hoc aggregation of autonomous networks, the $SPath$ would become very long and complicated, therefore, inefficient. Another constraint is that $SPath$ is not scalable. Thus it is desirable to be able to optimize the $SAPath$ field of an $SPath$.

First of all, we make a proposition as follows.

Service Paths of the form:

$$< SOpt : SAPath/Ser/Srv >:< Cost >$$

can always have the $SAPath$ optimized to a two-address format

$$add_{N_{home}}/add_{N_{service}}/$$

and the resulting $SPath$s have the form:

$$< SOpt : add_{N_{home}}/add_{N_{service}}/Ser/Srv >:< Cost >$$

We prove our proposition in the following section.

4.2 Basic Axioms and Theorems

In subsequent sections, we use Axiom P10 and Theorem P8 from Lampson et al in [43] to demonstrate how $SPath$ can be optimized. We adapted Theorem P8 by qualifying the Principals to act as "Authentication Agent".

Axiom P10

Listed in Appendix A.1, this axiom says that principal P_A can establish a "speak for" relationship with principal P_B when he declares P_B speaks for him.

Theorem P8

This theorem, listed in Appendix A.2, tells us that if principal P_A speaks for principal P_B, then whenever P_A says something, P_B would have said the same thing.

In our research, we are concerned with authentication authority of a principal. Hence we adapt Theorem P8 by qualifying the "speak for" relation with the role "as Authentication Agent" ("as AA" for short) as shown in Theorem 4.1 below.

Theorem 4.1 $\vdash (P_{A\ as\ AA} \looparrowright P_{B\ as\ AA}) \Rightarrow ((P_{A\ as\ AA}\ \mathbf{says}\ s) \Rightarrow (P_{B\ as\ AA}\ \mathbf{says}\ s))$

The symbols used are given in Table 4.1.

Symbol	Meaning
s	a statement
$\vdash s$	s is an axiom of the theory or s is provable form of the axioms.
$\looparrowright$	speak for
$\Rightarrow$	imply
$\wedge$	and

Table 4.1: List of symbols used in this chapter

4.3 Optimization of Service Path

In this section, we demonstrate how Service Paths can be optimized. We use a particular case as an example.

Suppose N_A provides a freely sharable FTP service at a cost of 215 units. The AS_A for N_A has an IP address of 10.1.1.1. The S_A is called FTPSer.

When network N_A offers a service, the service is listed as an $SPath$ of the form

$$< SOpt : SAPath/Ser/Srv >:< C >$$

The $SAPath$ field in this case is simply the network address of the authentication server (AS) of N_A.

$$< SOpt : add_{N_A}/Ser/Srv >:< C >$$

So the $SPath$ of our example FTP service listed in the service providing network N_A looks like:

$$< F : ./FTPSer/FTP >:< 215 >$$

or equivalently,

$$< F : 10.1.1.1/FTPSer/FTP >:< 215 >$$

When network N_B joins an SNG by attaching to network N_A, N_A delegates its authentication authority to N_B. N_A also passes the $SPath$ of the FTP service to N_B. Home users of N_B can now use the FTP service offered by N_A if they are authenticated by N_B. N_B lists all the shared service as $SPath$s by pre-pending the address of its authentication server to the $SAPath$ fields of all $SPath$s shared by N_A.

$$< SOption : add_{N_B}/add_{N_A}/Ser/Srv >:< Cost >$$

The $SPath$ for FTP service in N_B looks like:

$$< F : 10.1.2.1/10.1.1.1/FTPSer/FTP >:< 215 >$$

As the $SAPath$ field is pre-pended with a network address every time it is shared with another network, the $SAPath$ of an $SPath$ gets longer each time the service is shared with another network. When users try to authenticate and access a service, obviously, the overhead for authentication and setting up a service gets larger as the $SAPath$ gets longer. It is imperative to keep the $SAPath$ to an optimal length for the purpose of efficiency and scalability. To optimize an $SPath$ is the transformation of the $SAPath$ field of an arbitrary $SPath$ to its optimal form. In the next subsection, we discuss the theoretical basis of optimizing $SPath$s.

4.3.1 Optimization of SPath

We start to prove that $SPaths$ ($SAPath$ field) can be optimized with some formal definitions regarding Authentication Delegation in SNG context.

Definition 4.2 ***Authentication Delegation***
If network N_B attaches to network N_A which is a member of an SNG, then we define N_A delegates its authentication authority to network N_B. If a network N_B delegates its authentication authority to another network N_A, then we represent it as

$$N_{A\ as\ AA}\ \mathbf{says}\ (N_{B\ as\ AA} \twoheadrightarrow N_{A\ as\ AA})$$

When authentication authority is delegated, we have to keep track of the delegatee and the delegator relationships. The relationships are recorded in Authentication Delegation Paths which has the same format as SPath. Every time when a delegation occurs, the new delegatee address is pre-pended to the Authentication Delegation Path. So an Authentication Delegation Path would have the address of the delegatee network as the leftmost address and the delegator network address as the right most address. In between are intermediate networks which were delegatee networks at certain time in the authentication delegation process.

Definition 4.3 ***Authentication Delegation Path for Self-Authentication***
The Authentication Delegation Path of network N_A in network N_A itself is defined as:

$$add_{N_A}/$$

It simply means N_A performs authentication itself.

Definition 4.4 ***Authentication Delegation Path in Remote Networks***
If N_A delegates its authentication authority to another network N_B, then we define the Authentication Delegation Path for N_A in N_B to be

$$add_{N_B}/add_{N_A}/$$

within the SNG context. The delegated authentication authority can further be delegated. That is to say, if N_A delegates authentication authority to N_B which in turn delegates the authentication authority of N_A to another network N_C, the Authentication Delegation Path looks like

$$add_{N_C}/(add_{N_B}/add_{N_A}/)$$

which is equivalent to

$$add_{N_C}/add_{N_B}/add_{N_A}/$$

Hence we can generalize our Authentication Delegation Path definition to the following definition.

Definition 4.5 ***Authentication Delegation Path***

The Authentication Delegation Path is defined as the network path which traces the authentication delegation sequence from the delegator network to the final delegatee network in the form of

$$add_{N_{delegatee}}/.../add_{N_2}/add_{N_1}/add_{N_{delegator}}/$$

With the definitions 4.2 - 4.5 in place, we can now make the proposition that $SPath$s can be optimized.

Proposition 4.6 ***(Optimization of SPath)***

Service Path of the form

$$< SOpt : SAPath/Ser/Srv >:< Cost >$$

can always have the $SAPath$ optimized to a two-address format

$$add_{N_{home}}/add_{N_{service}}/$$

and the resulting $SPaths$ have the form

$$< SOpt : add_{N_{home}}/add_{N_{service}}/Ser/Srv >:< Cost >$$

We prove this proposition by working through a sequence of Lemmas.

Lemma 4.7 ***(Transitivity of "speak for" relation)***

$$\vdash (N_{A\ as\ AA} \rightsquigarrow N_{B\ as\ AA}) \wedge (N_{B\ as\ AA} \rightsquigarrow N_{C\ as\ AA}) \Rightarrow (N_{A\ as\ AA} \rightsquigarrow N_{C\ as\ AA})$$

PROOF:

From the first condition in the Lemma and Theorem 4.1, we have

$$(N_{A\ as\ AA}\ \mathbf{says}\ s) \Rightarrow (N_{B\ as\ AA}\ \mathbf{says}\ s)$$

Similarly, the second condition in the Lemma yields

$$(N_{B\ as\ AA}\ \mathbf{says}\ s) \Rightarrow (N_{C\ as\ AA}\ \mathbf{says}\ s)$$

So the predicate of the logic becomes:

$$((N_{A\ as\ AA}\ \mathbf{says}\ s)\ \Rightarrow\ (N_{B\ as\ AA}\ \mathbf{says}\ s)) \wedge ((N_{B\ as\ AA}\ \mathbf{says}\ s)\ \Rightarrow\ (N_{C\ as\ AA}\ \mathbf{says}\ s))$$

Transitive property of the "⇒" relation allows us to replace the predicate with

$$((N_{A\ as\ AA}\ \mathbf{says}\ s)\ \Rightarrow\ (N_{C\ as\ AA}\ \mathbf{says}\ s))$$

which is precisely what we get when we apply Theorem 4.1 to the conclusion of the Lemma. ■

Lemma 4.8 ***(Transitivity of Authentication Delegation)***

Suppose N_A delegates the authentication authority to N_B. When N_B delegates the authentication authority to N_C, the authentication authority for N_A is also delegated to N_C.

PROOF

When N_A delegates authentication authority to N_B, by Definition 4.2 and Axiom P10 from [43], we have

$$(N_{B\ as\ AA}\ \rightarrowtail\ N_{A\ as\ AA})$$

When N_B delegates authentication authority to N_C, by Definition 4.2 and Axiom P10 from [43], we have

$$(N_{C\ as\ AA}\ \rightarrowtail\ N_{B\ as\ AA})$$

These two authentication delegations satisfied the conditions of Lemma 4.7 and so we can conclude from Lemma 4.7 that

$$(N_{C\ as\ AA}\ \rightarrowtail\ N_{A\ as\ AA})\ \blacksquare$$

Lemma 4.9 ***(Authentication Delegation Path)***

If $N_{A\ as\ AA}$ delegates its authentication authority to another network $N_{B\ as\ AA}$, then N_B acquires the Authentication Delegation Path for N_A, and all Authentication Delegation Paths N_A has, with the address of N_B pre-pended:

$$add_{N_B}/add_{N_A}/$$

$$add_{N_B}/add_{N_A}/.../$$

PROOF

When N_A delegates its authentication authority to N_B, by Definition 4.4, N_B has a Authentication Delegation Path

$add_{N_B}/add_{N_A}/$

Similarly, when N_B delegates its authentication authority to N_C, from Lemma 4.8, Definition 4.4 and Definition 4.5, N_C has two Authentication Delegation Paths

$add_{N_C}/add_{N_B}/$

$add_{N_C}/add_{N_B}/add_{N_A}/$

We keep on applying Lemma 4.8 and Definition 4.4 and 4.5 every time an autonomous network delegates its authentication authority to another network, until, finally N_A delegates its authentication authority to network N_B. From Lemma 4.8, all authentication authority already delegated to N_A, and the authentication authority of N_A itself, are delegated to N_B. When authentication authorities are delegated to N_B all the Authentication Delegation Paths have the address of N_A pre-pended by Definition 4.4. ■

Lemma 4.10 ***(Equivalence of Authentication Delegation)***
Authentication Delegation Path of the form

$add_{N_H}/.../add_{N_3}/add_{N_2}/add_{N_1}/add_{N_S}/$

is equivalent to

$add_{N_H}/add_{N_S}/$

PROOF
Authentication Delegation Path

$add_{N_H}/.../add_{N_3}/add_{N_2}/add_{N_1}/add_{N_S}/$

indicates N_S delegates its authentication authority to N_1 (Definition 4.5) which in turn delegates its authentication authority to N_2. From Lemma 4.8, the authentication authority for N_S is also delegated to N_2. By Definition 4.5 the Authentication Delegation Path of N_S in N_2 is

$add_{N_2}/add_{N_1}/add_{N_S}/$

When the authentication authority for N_S is delegated to N_2, using Definition 4.2, we have

$N_{S\ as\ AA}$ **says** $(N_{2\ as\ AA} \hookrightarrow N_{S\ as\ AA})$

By Axiom P10 from [43], we have

$(N_{2\ as\ AA} \rightarrowtail N_{S\ as\ AA})$

And by Definition 4.4, the Authentication Delegation Path for the delegation listed above is

$add_{N_2}/add_{N_S}/$

Hence we can transform Authentication Delegation Path from

$add_{N_2}/add_{N_1}/add_{N_S}/$

to a shorter form

$add_{N_2}/add_{N_S}/$

Every time we apply the argument to the address triplets on the left hand side of an Authentication Delegation Path, we get one address less. By repeating the process just described to an Authentication Delegation Path argument, we can arrive at its optimized form:

$add_{N_H}/add_{N_S}/$ ■

PROOF of Proposition 4.6
$SAPath$ inside a $SPath$ is the authentication Delegation Path of the service to the user's home network. Hence by Lemma 4.10, all Authentication Delegation Path can be reduced to the form

$add_{N_H}/add_{N_S}/$

and hence we have the optimized form of an $SPath$. ■

4.4 Implementing SPath Optimization

In this section, we discuss how to achieve the $SPath$ optimization in an SNG context.

For heterogeneous aggregation of networks in an SNG, it is common to have a service shared with many neighboring networks. The shared service may, in turn, share with more next neighbors. When a shared service is made available to a network, it may come with a different Service Path even though the service is provided by same unique server.

When the Service Path is optimized, information about the Authentication Path is lost. Furthermore, if a network withdraws from the SNG, all the Service Paths it shared with other networks would be invalidated. The optimized form of $SPath$ does not reveal the intermediate networks involved. The same problem occurs when a networks re-joins an SNG, members of the SNG have to determine which $SPath$s should be validated.

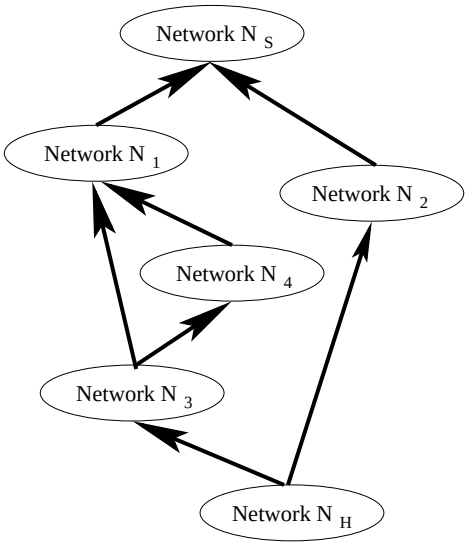


Figure 4.1: Sharing of key before SPath optimization

4.4.1 Selecting Service Paths

We discuss these issues along with the implementation methodology using Figure 4.1.

Figure 4.1 shows three possible ways to obtain a service. The $SPaths$ are:

$$< SOpt : N_H/N_3/N_1/N_S/Ser/Srv >:< C1 >$$

$$< SOpt : N_H/N_3/N_4/N_1/N_S/Ser/Srv >:< C2 >$$

$$< SOpt : N_H/N_2/N_S/Ser/Srv >:< C3 >$$

Obviously the three $SPaths$ should be optimized to the same $SPath$:

$$< SOpt : N_H/N_S/Ser/Srv >:< C4 >$$

Note that the choice for the $Cost$ is the minimum of all $SPath\ Costs$. In this case, $C4$ is the minimum of $C1$, $C2$ and $C3$.

The optimization essentially summarizes $SPaths$ of the same service with different service access paths to a single $SPath$. This helps to minimize the resources used and provides a more scalable way of deployment.

4.4.2 Service Optimization Table and Authentication Path Network Lookup Table

We have seen that three distinct $SPaths$ for the same service can be optimized and form a single $SPath$ for the service in Section 4.4.1. It is true that by optimizing the $SPath$, the

$SPath$ is more efficient and scalable. But the lost of detailed $SPath$ information results in two problems.

As the $SPath$ serves as the path for the user home network to get service information, the optimized $SPath$ does not have explicit hop by hop information for the user home network to reach the server for service information. Secondly, if a network stops to participate in an SNG, the Authentication Delegation chain is broken and all the $SPath$s that involve the network are no longer valid and should be removed from the Service List. With the $SPath$s optimized, there is no way to tell which $SPath$ is to be removed from the Service List.

The problems can be solved if

- the $SPath$ is shared in full form
- the $SPath$ in full form and the corresponding optimized form are recorded in a $SPath$ Optimization Table (SOT)
- the optimized form is listed in the Authentication Path Network Lookup Table ($APNLT$).

An $APNLT$ lists all full $SPath$s associated with an optimized $SPath$ and the networks involved in all the full $SPath$s. A typical SOT and $APNLT$ are shown in Table 4.2 and Table 4.3.

Consider the two $SPath$s in network N_H:

$$< SOpt : N_H/N_3/N_1/N_S/Ser/Srv >:< C1 >$$

$$< SOpt : N_H/N_2/N_S/Ser/Srv >:< C3 >$$

The last one is optimized to the form

$$< SOpt : N_H/N_S/Ser/Srv >:< C3 >$$

Here we assume that $C3$ is less than $C1$ and $C3$ is chosen as the cost for the optimized $SPath$.

The SOT for network N_H look like Table 4.2 and the corresponding $APNLT$ look like Table 4.3.

Optimized SPath	Full SPath	Status of full SPath
$< SOpt : N_H/N_S/Ser/Srv >:< C3 >$	$< SOpt : N_H/N_3/N_1/N_S/Ser/Srv >:< C1 >$ $< SOpt : N_H/N_2/N_S/Ser/Srv >:< C3 >$	Standby Chosen

Table 4.2: SPath Optimization Table for N_H

It is easy to see that the Authentication Path for the optimized $SPath$ is $N_H/N_2/N_S/$ from SOT. The $APNLT$ provides a quick way to check if an optimized $SPath$ is affected or not when a network withdraws from an SNG. There are three cases as follow.

SPath	N_1	N_2	N_3	N_4	N_5	...	N_S	...
$< SOpt : N_H/N_S/Ser/Srv >:< C3 >$								
$< SOpt : N_H/N_3/N_1/N_S/Ser/Srv >:< C1 >$	T	F	T	F	F	...	T	...
$< SOpt : N_H/N_2/N_S/Ser/Srv >:< C3 >$	F	T	F	F	F	...	T	...

Table 4.3: Authentication Path Network Lookup Table for N_H

Case 1

This is the case in which N_1 withdraws from the SNG.

From $APNLT$, column N_1 we see that the entry at the row for

$$< SOpt : N_H/N_3/N_1/N_S/Ser/Srv >:< C1 >$$

is "T" and the entry at the row for

$$< SOpt : N_H/N_2/N_S/Ser/Srv >:< C3 >$$

is "F". It means that

$$< SOpt : N_H/N_3/N_1/N_S/Ser/Srv >:< C1 >$$

is invalid now as it uses N_1 as part of the $SAPath$ while

$$< SOpt : N_H/N_2/N_S/Ser/Srv >:< C3 >$$

is not affected as it does not use N_1 in its $SAPath$.

So $SPath$

$$< SOpt : N_H/N_3/N_1/N_S/Ser/Srv >:< C1 >$$

should be removed from the $APNLT$ and SOT, and

$$< SOpt : N_H/N_2/N_S/Ser/Srv >:< C3 >$$

promoted to $Chosen$ as shown in Table 4.4 and Table 4.5.

Optimized SPath	Full SPath	Status of full SPath
$< SOpt : N_H/N_S/Ser/Srv >:< C3 >$	$< SOpt : N_H/N_2/N_S/Ser/Srv >:< C3 >$	Chosen

Table 4.4: SPath Optimization Table for N_H

SPath	N_1	N_2	N_3	N_4	N_5	...	N_S	...
$< SOpt : N_H/N_S/Ser/Srv >:< C3 >$								
$< SOpt : N_H/N_2/N_S/Ser/Srv >:< C3 >$	F	T	F	F	F	...	T	...

Table 4.5: Authentication Path Network Lookup Table for N_H

Case 2

This is the case in which N_2 Instead of N_1 withdraws from the SNG.

Column N_2 of $APNLT$ has a "F" at the row for

$$< SOpt : N_H/N_3/N_1/N_S/Ser/Srv >:< C1 >$$

and "T" at the row for

$$< SOpt : N_H/N_2/N_S/Ser/Srv >:< C3 >$$

Now it is

$$< SOpt : N_H/N_2/N_S/Ser/Srv >:< C3 >$$

which is invalid, and

$$< SOpt : N_H/N_3/N_1/N_S/Ser/Srv >:< C1 >$$

is not affected.

$$< SOpt : N_H/N_2/N_S/Ser/Srv >:< C3 >$$

would be removed from the $APNLT$ and SOT while

$$< SOpt : N_H/N_3/N_1/N_S/Ser/Srv >:< C1 >$$

in SOT would be promoted to ***Chosen*** as it provides the $SPath$ with minimum cost. Note the $Cost$ for the optimized $SPath$ changes to $C1$ as the $SPath$ which has a $Cost$ of $C3$ is no longer available. The updated SOT and $APNLT$ are shown in Table 4.6 and Table 4.7.

Optimized SPath	Full SPath	Status of full SPath
$< SOpt : N_H/N_S/Ser/Srv >:< C1 >$	$< SOpt : N_H/N_3/N_1/N_S/Ser/Srv >:< C1 >$	Chosen

Table 4.6: SPath Optimization Table for N_H

SPath	N_1	N_2	N_3	N_4	N_5	...	N_S	...
$< SOpt : N_H/N_S/Ser/Srv >:< C1 >$								
$< SOpt : N_H/N_3/N_1/N_S/Ser/Srv >:< C1 >$	T	F	T	F	F	...	T	...

Table 4.7: Authentication Path Network Lookup Table for N_H

Case 3

This is the case in which both N_1 and N_2 stay in the SNG and a new $SPath$ for the same service $N_S/Ser/Srv$ is shared with N_H.

$$< SOpt : N_H/N_3/N_4/N_1/N_S/Ser/Srv >:< C2 >$$

If $C2$ is larger than $C3$, the new $SPath$ becomes a backup $SPath$ and assumes a status of *Standby* as shown in Table 4.8 and Table 4.9.

Optimized SPath	Full SPath	Status of full SPath
$< SOpt : N_H/N_S/Ser/Srv >:< C3 >$	$< SOpt : N_H/N_3/N_1/N_S/Ser/Srv >:< C1 >$	Standby
	$< SOpt : N_H/N_2/N_S/Ser/Srv >:< C3 >$	Chosen
	$< SOpt : N_H/N_3/N_4/N_1/N_S/Ser/Srv >:< C2 >$	Standby

Table 4.8: SPath Optimization Table for N_H

SPath	N_1	N_2	N_3	N_4	N_5	...	N_S	...
$< SOpt : N_H/N_S/Ser/Srv >:< C3 >$								
$< SOpt : N_H/N_3/N_1/N_S/Ser/Srv >:< C1 >$	T	F	T	F	F	...	T	...
$< SOpt : N_H/N_2/N_S/Ser/Srv >:< C3 >$	F	T	F	F	F	...	T	...
$< SOpt : N_H/N_3/N_4/N_1/N_S/Ser/Srv >:< C2 >$	T	F	T	T	F	...	T	...

Table 4.9: Authentication Path Network Lookup Table for N_H

On the other hand, if $C2$ is less than $C3$, the new $SPath$ becomes the preferred $SPath$ and assume a status of ***Chosen*** while

$$< SOpt : N_H/N_2/N_S/Ser/Srv >:< C3 >$$

is down-graded to *Standby*. The $APNLT$ remains the same as before, but the SOT is changed as shown in Table 4.8 and Table 4.10.

In the extreme case when an optimized $SPath$ has no valid full $SPath$ in the SOT, the optimized $SPath$ is no longer valid. Hence it can be removed from the SOT, $APNLT$ and service list.

Optimized SPath	Full SPath	Status of full SPath
$< SOpt : N_H/N_S/Ser/Srv >:< C3 >$	$< SOpt : N_H/N_3/N_1/N_S/Ser/Srv >:< C1 >$	Standby
	$< SOpt : N_H/N_2/N_S/Ser/Srv >:< C3 >$	Standby
	$< SOpt : N_H/N_3/N_4/N_1/N_S/Ser/Srv >:< C2 >$	Chosen

Table 4.10: SPath Optimization Table for N_H

With the SOT and $APNLT$ in place, we can optimize $SPath$ without loss of Authentication Path information and can check how the optimized $SPath$s are affected when an autonomous network withdraws from an SNG.

When an AS uses an optimized $SPath$, there may not be any direct authentication delegation relationship for the home network and the service providing network ($SNet$) and encrypted channel cannot be established.

4.4.3 Authentication Re-Delegation

Optimization mechanism guarantees the scalability of $SPath$. However, after the optimization is applied, if a network wishes to reach the service providing network for service information, there may not be any encrypted channel for the process. The service providing network may not have delegated the authentication authority to the service requesting network. To overcome this, we introduce the Authentication Re-delegation [40].

Remember that when network N_A joins an SNG, it shares a secret key K_1 with a member network, N_B of the SNG for establishing a self-authenticating encryption channel. In N_A, the Authentication Delegation Path for N_B is

$$add_{N_A}/add_{N_B}/$$

When another network N_C links with N_A to join the SNG, the key shared between N_A and N_C is K_2. In N_C the Authentication Delegation Paths are

$$add_{N_C}/add_{N_A}/$$

$$add_{N_C}/add_{N_A}/add_{N_B}/$$

Proposition 4.6 allows us to optimize the second Authentication Delegation Path to

$$add_{N_C}/add_{N_B}/$$

N_C has a shared key with N_A. N_B has a shared key with N_A and N_B has no shared key with N_C. The optimized $SPath$ $add_{N_C}/add_{N_B}/$ works only when there is a shared key between N_C and N_B. The shared key is used to establish a self-authenticating encryption

channel between N_C and N_B. So N_C must share a key K_3 with N_B before optimizing any $SPath$ in which the service is provided by N_B.

The sharing of a key between N_B and N_C is called Authentication re-Delegation from N_B to N_C given that N_B has delegated authentication authority to N_A and N_A has delegated authentication authority to N_C.

Since Authentication Delegation in SNG is transitive (Lemma 4.8), N_B can share a secret key with N_C without further administrative work. This can be done via the original encrypted Authentication Delegation Path or simply uses the same procedure as when N_C initially links with N_A. By sharing a key with N_B, N_C is now linked directly with N_B.

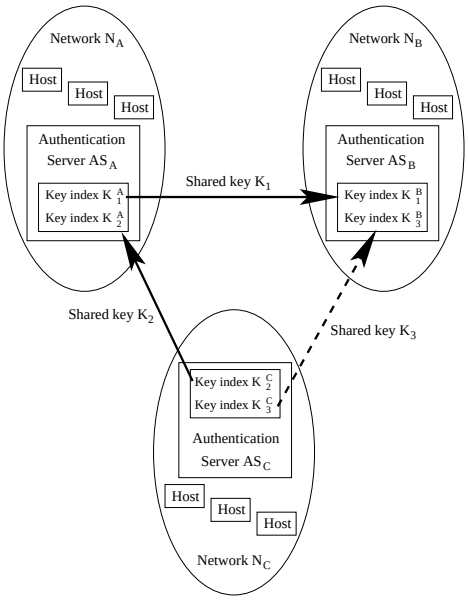


Figure 4.2: Sharing of key before SPath optimization

Let us use an example here to explain the concept of Authentication Delegation Path.

> Figure 4.2 shows that N_A shares a key K_1 with N_B and the key indices for K_1 are K_1^A and K_1^B in N_A and N_B respectively. The shared key between N_A and N_C is K_2. The key indices for K_2 are K_2^A and K_2^C in N_A and N_C respectively. After Authentication Re-delegation, N_C and N_B have acquired K_3^B and K_3^C respectively.

The implementation is valid for all $SPath$s which has the $SAPath$ field optimized to the two-address format. add_{N_C} and add_{N_B} are now used in place of add_{N_H} and add_{N_S}. The N_H has to initiate the sharing of a key with the N_S which is exactly the same as when N_H is trying to attach to the N_S. The extra addresses which appear in the original $SAPath$ have no affect on the optimization process as shown in Figure 4.3.

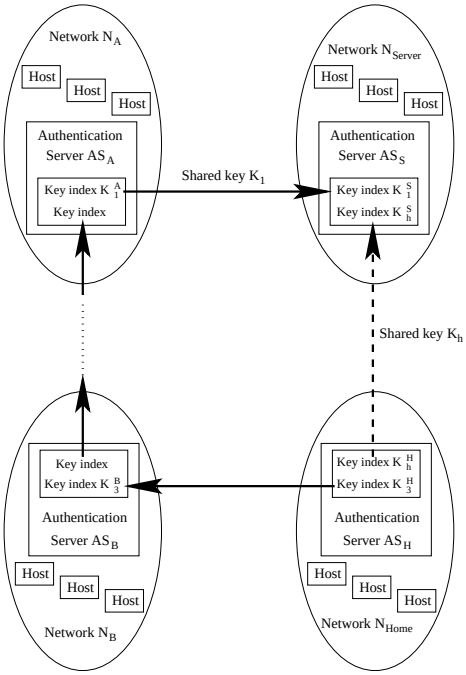


Figure 4.3: SPath optimization in general

4.4.4 Revocation of Authentication Delegation

At any point of time, if a network wishes to revoke its authentication delegation [40] to a certain network, all it needs to do is to send a revocation message containing its own network identity as the delegator and the network identity of the delegatee network. Each member network of SNG adjusts its service list, the SOT and the $APNLT$ to reflect the revocation.

Starting with $APNLT$ and SOT as shown in Table 4.9 and Table 4.10, and assuming N_H receives the following revocation message:

$Delegator : N_4$

$Delegatee : N_3$

$Service : N_S/Ser/Srv$

N_H immediately looks up the SOT and the $APNLT$, removing all entries of full $SPath$s that contain N_3/N_4 in their $SAPath$s; and has $N_S/Ser/Srv$ as the service and service provider. Note that only $SPath$s with the pattern N_3/N_4 in their $SAPath$s are removed, $SPath$s with the pattern N_4/N_3 in their $SAPath$ should not be removed as N_4/N_3 means

$Delegator : N_3$

$Delegatee : N_4$

and is not affected by the revocation of authentication delegation from N_4 to N_3.

Similarly, $SPath$s with $SAPath$ pattern $N_3/.../N_X/N_4$ are not removed as N_4 has not revoked its authentication delegation to N_X.

Statuses of the remaining full $SPath$s in SOT are updated as shown in Table 4.11 and Table 4.12.

Optimized SPath	Full SPath	Status of full SPath
$< SOpt : N_H/N_S/Ser/Srv >:< C3 >$	$< SOpt : N_H/N_3/N_1/N_S/Ser/Srv >:< C1 >$	Standby
	$< SOpt : N_H/N_2/N_S/Ser/Srv >:< C3 >$	Chosen

Table 4.11: SPath Optimization Table for N_H

SPath	N_1	N_2	N_3	N_4	N_5	...	N_S	...
$< SOpt : N_H/N_S/Ser/Srv >:< C3 >$								
$< SOpt : N_H/N_3/N_1/N_S/Ser/Srv >:< C1 >$	T	F	T	F	F	...	T	...
$< SOpt : N_H/N_2/N_S/Ser/Srv >:< C3 >$	F	T	F	F	F	...	T	...

Table 4.12: Authentication Path Network Lookup Table for N_H

4.5 Chapter Summary

In this chapter, we presented the basics and implementation for $SPath$ Optimization. To enable $SPath$ Optimization, we also introduced Authentication Re-Delegation. We listed three cases where a network withdrew from or joined to an SNG and the affect of such changes. In the next chapter, we discuss how an encrypted channel in SNG can authenticate itself which is required for authentication delegation in SNG.

Chapter 5

Self-authenticating Channel

In Chapter 3, we have discussed that SNG is a dynamic aggregation of networks linked together using encrypted channels and encrypted channels are self-authenticated. In Chapter 4, we have addressed the efficiency and scalability of SNG, and the optimization of $SPath$s. In this Chapter, we focus on the encrypted channels and theoretically prove that encrypted channels are self-authenticating [37].

5.1 Authentication Delegation in SNG

As we know, Authentication delegation in SNG is the delegation of authentication authority from one network (AS) to another. When an autonomous network joins an SNG, it has to obtain the trust of at least one autonomous network in the SNG and obtains the authentication delegation from the network. An SNG is a representation of dynamically aggregated networks, which are linked by encrypted channels.

Whether the encrypted channels are self-authenticated is critically important for the secure service sharing on the SNG. In this Chapter, we prove that an encrypted channel is self-authenticating. Let us once more use the SNG given in Chapter 3 on page 40. If the encryption key is shared by AS_1 with AS_2 only, AS_1 is assured of the origin of a message if it is encrypted with the shared key. So when AS_2 says something regarding the identity of a user, encrypted with the shared key, AS_1 which has delegated the authentication authority to AS_2, is comfortable to say the same.

It is reasonable that symbols and approach similar to that presented by Lampson [43] are used to give a formal proof of self-authentication for encrypted channels in SNG. Like in Chapter 4, we start with some axioms and theorems from Guttman et al [17] and from Lampson [43] first. We then prove the proposition that encrypted channels authenticate themselves by proposing and proving three lemmas.

5.2 Self-Authentication of Encrypted Channels

The objective in this section is to prove that an encrypted channel in SNG authenticates itself. Axiom 1 from [17] (listed in Appendix B) tells us about free encryption. Axiom P10, Theorem P8, and Theorem P11 from [43] (listed in Appendix A) established the foundation for delegation of authorities.

When an autonomous network joins an SNG, its authentication server (AS_A) shares a unique and secret key with the authentication server (AS_B) of a member network of the SNG. The secret key would be stored in the key database of AS_B and has a key index. The shared key is used by AS_A to establish an encrypted channel with AS_B. In so doing, AS_A joins AS_B and becomes part of the SNG.

The SNG one-way joining process forms the basis of the assumptions of Proposition 5.4. The conclusion of Proposition 5.4 is that whatever encrypted message delivered by the encrypted channel is originated from AS_A. In other words, encrypted channels formed when joining an SNG are self-authenticating.

The proof starts with three definitions and then three lemmas. We then apply the lemmas to prove our Proposition 5.4.

Definition 5.1 *K^r is defined as a key identifier if it allows a receiver r to know what the decryption key of an encrypted channel is but doesn't disclose anything about it to others.*

In the above definition, a key identifier can be one of the following:

- an index to a database of keys kept by the receiver; or
- $\{K^{-1}\}_{K_m}$ where K_m is a secret master key kept by the receiver and K^{-1} is the decryption key; or
- a pair $(K^x, \{K^{-1}\}_{K'})$, where K^x is a key identifier of key K' which encrypts the decryption key K^{-1}.

Definition 5.2 *When principal P_A shares a unique secret key K_A with principal P_B, key K_A and a key index K_A^B is added to the database DB_B of keys in P_B, then we define DB_B* **says** $(K_A^B \leftrightarrow P_A)$.

It follows from the assumption that the key is unique and secret. The database in P_B is happy to say the speak for relationship assuming a one-to-one (K_A^B to P_A) mapping. Note that the mapping from P_A to K_A^B can be one-to-many.

This definition describes what happens when a unique key of one principal is shared with another principal.

Definition 5.3 *When P_A establishes an encrypted channel with P_B using a unique shared key K_A, then we define P_A* **says** $(DB_B \leftrightarrow P_A)$.

Note that a unique secret key has to be shared before an encrypted channel can be established.

The above definitions do not exclude the cases when a pair such as $(K_A^{B'}, \{s\}_{K_A})$ is received in place of the expected pair $(K_A^B, \{s\}_{K_A})$.

5.2.1 One-way Self-authentication of Encrypted Channels

In this subsection, we demonstrate how an encrypted channel can have the property of one-way self-authentication.

Proposition 5.4 *Suppose:*

1. *Principal P_A shares a unique secret key, K_A, with principal P_B;*
2. *P_A uses K_A to establish an encryption channel with P_B;*
3. *K_A^B is the key index of K_A^{-1} in P_B.*

then when P_B receives $(K_A^B, \{s\}_K)$, P_B can infer that P_A $\mathbf{says}$ s.

We prove this by working through a sequence of lemmas.

Lemma 5.5 *Suppose there is only one unique decryption key corresponding to each encryption key used in an encrypted channel. The holder of a decryption key K^{-1} of an encrypted channel can infer that "K^{-1} $\mathbf{says}$ s" on receiving $\{s\}_K$.*

$$\vdash (K^{-1}, \{s\}_K) \Rightarrow (K^{-1} \ \mathbf{says} \ s)$$

PROOF: It follows from the assumption in Lemma 5.5 and Axiom 1 from [17] that there is only one unique key which can get back message s. We can safely associate the message s with the decryption key (index). ■

Lemma 5.6 *Suppose K^r is a key identifier. When a pair $(K^r, \{s\}_K)$ was received, the receiver can infer that "K^r $\mathbf{says}$ s".*

$$\vdash (K^r, \{s\}_K) \Rightarrow (K^r \ \mathbf{says} \ s)$$

PROOF: It is a direct application of Definition 5.1 to Lemma 5.5. ■

Lemma 5.7 *Suppose:*

1. *Principal P_A shares a unique key K_A with principal P_B;*
2. *P_A uses K_A to establish an encryption channel with P_B;*

3. *K_A^B is the key index of K_A^{-1} in P_B.*

For P_B, $(K_A^B \mapsto A)$.

PROOF: According to assumption 1, and Definition 5.2, DB_B **says** $(K_A^B \mapsto P_A)$.

Assumption 2 and Definition 5.3 tells us that P_A **says** $DB_B \mapsto P_A$ which is the same as $DB_B \mapsto P_A$ by Axiom P10 from [43].

Since both conditions listed above are assumed to be true, we can link them into a compound predicate $(DB \mapsto P_A) \wedge (DB$ **says** $(K_A^B \mapsto P_A))$. This is the predicate in the Handoff Rule from Lampson (Theorem P11 from [43]):
$\vdash (DB \mapsto P_A) \wedge (DB$ **says** $(K_A^B \mapsto P_A)) \Rightarrow (K_A^B \mapsto P_A)$

We may now conclude that $(K_A^B \mapsto P_A)$ ■

PROOF of Proposition 5.4

The assumptions in Proposition 5.4 give $(K_A^B \mapsto P_A)$ using Lemma 5.7.

When P_B receives $(K_A^B, \{s\}_K)$, P_B can infer that $(K_A^B$ **says** $s)$ using Lemma 5.6.

According to Theorem P8 from [43], we have
$\vdash (K_A^B \mapsto P_A) \Rightarrow ((K_A^B$ **says** $s) \Rightarrow (P_A$ **says** $s))$ And hence when P_B receives $(K_A^B, \{s\}_K)$, P_B can infer that P_A **says** s. ■

5.2.2 Two-Way Self-authentication of Encrypted Channels

In Section 5.2.1, Proposition 5.4 shows that traffic delivered to AS_B via the encrypted channel established with the key shared by AS_A originates from AS_A. This is a one-way self-authentication. It would be desirable if we can extend Proposition 5.4 to cover a two-way self-authentication. A two-way self-authentication means that messages exchanged between AS_A and AS_B using encrypted channels established with shared keys can be assured of the origin. Those messages heading for AS_A are coming from AS_B and those messages going to AS_B should originate from AS_A. To extend Proposition 5.4, we need to add two assumptions and modify another assumption.

Proposition 5.8 *Suppose:*

1. *Principal P_A shares a unique secret key, K_A, with principal P_B;*
2. *Principal P_B shares the same unique secret key, K_A, with principal P_A;*
3. *P_A and P_B use K_A to establish an encryption channel between them;*
4. *K_A^B is the key index of K_A^{-1} in P_B.*
5. *K_A^A is the key index of K_A^{-1} in P_A.*

*then when P_B receives $(K_A^B, \{s\}_K)$, P_B can infer that P_A **says** s; and when P_A receives $(K_A^A, \{s\}_K)$, P_A can infer that P_B **says** s.*

PROOF: Using assumptions 1, 3, 4, and Proposition 5.4 in Section 5.2.1, we can say that when P_B receives $(K_A^B, \{s\}_K)$, P_B can infer that P_A **says** s. With assumption 2, 3, 5 and Proposition 5.4 we can say that when P_A receives $(K_A^A, \{s\}_K)$, P_A can infer that P_B **says** s. ■

5.3 Chapter Summary

In this chapter, we have theoretically proved that encrypted channels in SNG authenticate themselves. With this property, when Principal P_A and Principal P_B set up an encrypted channel using a shared key, P_A is assured that the messages he receives from the channel come from P_B. P_B also has the same assurance that the messages he receives from the same channel come from P_A. This self-authenticating property of encrypted channels enables two communicating parties to achieve both one-way and two-way self-authentication in SNG.

Chapter 6

Authentication Protocol for SNG

In Chapters 3, 4 and 5 we proposed SNG and proved that the SNG linked by the encrypted channels can achieve the secure service sharing over the autonomous networks. In this chapter we propose a new verifier-based password authentication protocol using Dynamic Passwords. The protocol features mutual authentication, integrated session key exchange and is resistant to both dictionary attacks and replay attacks. It also reveals any successful spoofing.

The protocol is most suitable for mobile users such as SNG in $DNSA - US$ mode.

6.1 Introduction

Among the many authentication protocols available, verifying password is the most primitive and simplest form of authentication. Other forms of authentication may rely on using something that a user owns or is part of the user. In many cases, assistance from dedicated hardware is needed. An authentication protocol that requires assistance of dedicated hardware is less flexible, and may have compatibility and portability problems when implemented across different platforms. Using passwords, however, users only have to remember their own account name and password in order to access protected resources. Password authentication is easy to use and straight forward to implement. However it has some limitations.

In general, strong passwords are comparatively long. Memorizing a long password that is made up of unrelated characters and digits or special symbols can be a daunting task. This entices people to choose easy passwords such as birthdays, names of kin or common English words [74], and keep using the passwords for as long as possible. These weak passwords could be the targets of dictionary attack. In a dictionary attack, possible passwords are common dates, names or dictionary words. The results are validated using some publicly known information about the password and the authentication protocol used.

A second problem with weak password is called replay attack. Replay attack occurs

when an adversary captures a legitimate login packet and replays it later to gain access to resources as a legal user.

Third problem is so called spoofing. With a stolen password, an adversary can login as the password owner. The owner of the stolen password is not alerted at all.

With all these problems in mind, we propose the concept of Dynamic Passwords [33]. A Dynamic Password consists of two parts. The first part changes each time the Dynamic Password is used and is called the dynamic part. The second part has a relatively longer life time and length than the first part and is called the static part. If a Dynamic Password is changed with every use, a replay attack is prevented. At the same time, spoofing attack can easily be detected.

To deliver the new dynamic part of a Dynamic Password, we propose Key Exchange using Dynamic Passwords ($KEDP$). This protocol is portable for mobile users, resists replay and dictionary attacks, and it provides mutual authentication and integrates key exchange with authentication. $KEDP$ also offers a detection mechanism when spoofing happens

6.2 Strength of Passwords

The commonly accepted metric for strength of a password is its life time without being compromised. We can work out the life time of a password by assuming an adversary can make continuous guesses over the Internet. The Federal Information Processing Standards Publication 112 [53] outlines how to determine password length and lifetime.

The probability of correctly guessing a password is proportional to the number of guesses an adversary can make before the password expires (and changed) and inversely proportional to the total possible number of guesses. The number of guesses an adversary can make depends on the password life time and the guess rate. The total number of possible password guesses depends on the length of the password and the size of the character set which forms the password. We base our calculation for the life time of a password on the fact that the probability of a correct guess should be less than or equal to a certain value, say, 1 in 1,000,000.

Let us derive a mathematical formula to represent the probability. But first we use the following symbols. ie.

A = size of character set for password item
M = length of password
L = password lifetime
R = guess rate
S = size of password space
P = acceptable probability limit of guessing a password during its lifetime
G = total number of guesses made during a password's lifetime

We need the basic combination formula for total number of choices when choosing n events out of k possible events is:

$$C_k^n = \frac{n!}{k!(n-k)!} \tag{6.1}$$

The three equations from [53] we used are listed below.

Equation 8 of Appendix E.6 in [53]:

$$P = \frac{G}{S} \tag{6.2}$$

Equation 2 of Appendix E.3 in [53]:

$$S = A^M \tag{6.3}$$

Equation 1 of Appendix E.6 in [53]:

$$G = L * R \tag{6.4}$$

Substituting Equation 6.3 and Equation 6.4, Equation 6.2 becomes:

$$\begin{aligned} P &= \frac{L * R}{A^M} \\ L &= \frac{P * A^M}{R} \end{aligned} \tag{6.5}$$

Or, equivalently,

$$\begin{aligned} P &= \frac{L * R}{A^M} \\ A^M &= L * R/P \\ MlogA &= logL + logR - logP \\ M &= \frac{logL + logR - logP}{logA} \end{aligned} \tag{6.6}$$

As an illustration, we assume the variables have values shown in Table 6.1. The corresponding calculated life time for passwords of various length are listed in Table 6.2.

Item	Value
A	62 (upper and lower case letters + digits
P	1/1,000,000 (1 correct guess per million guesses)
Internet speed	512 kbps
Bits per guess	200
Guess rate (R)	= Modem speed / bits per guess
	= 2620 guesses per second

Table 6.1: Parameters for password life time calculations.

Number of Characters in password	Life time of password
5	0.4 s
6	21 s
7	1340 s
8	0.96 day
9	60 day
10	10 year

Table 6.2: Typical life time of passwords.

6.3 Dynamic Password

Normally, length, composition and life time [53] are three important design factors for passwords. It is obvious that a password can be made stronger by increasing its length or by decreasing its lifetime and enlarging the character set.

In [53], there is a guideline for determining the length and lifetime of passwords that are safe against on-line brute force attacks. From Table 6.1 and Table 6.2, assuming a password guess consists of sending 200 bits and using a modem with speed of 512 kbps, a password of 5 characters drawn from a 62-character set can have a lifetime of less than half of a second. A password of 8 characters long can have a lifetime of approximately one day as shown in Table 6.2. In common sense, memorizing a long password that consists of unrelated characters is not easy, and is even not practical when the password changes frequently. Users more readily accept longer passwords when the passwords have a longer lifetime. Towards this end, we design a new schema for relatively stronger but easy to remember password.

Such a password is called Dynamic Password ($DPass$). A Dynamic Password consists of two parts. The static part ($SDPass$) has a longer lifetime and also a longer length

while the dynamic part ($DDPass$) has a shorter lifetime and shorter length. Combining a $SDPass$ and a $DDPass$ together satisfies the length and lifetime requirements of a strong password. A longer lifetime alleviates the effect of the longer length of $SDPass$. The short lifetime of $DDPass$ is mitigated by its relatively shorter length. The shorter lifetime of $DDPass$ also becomes the effective life time of a Dynamic Password as a whole since the Dynamic Password is effectively changed when $DDPass$ changes.

6.3.1 Strength of Dynamic Passwords

In this Subsection, we look into the strength of the proposed Dynamic Passwords. Dynamic Passwords have a dynamic part attached to Static part. We can consider the static part a normal password and the dynamic part adds to the length and shortens the life time of the password. As $DDPass$ has a shorter life time, the resulting Dynamic Password has the same shorter life time as the $DDPass$.

Next, we discuss it mathematically. The notations we used are as follows:

A_S	character set for static part of a Dynamic Password
A_D	character set for dynamic part of a Dynamic Password
M_S	length of static part of a Dynamic Password
M_D	length of dynamic part of a Dynamic Password
L	password lifetime
R	guess rate
S	size of password space
P	acceptable probability limit of guessing a password during its lifetime
G	total number of guesses made during a password's lifetime

We re-write Equation 6.3 as

$$S = (A_S)^{M_S} * (A_D)^{M_D} \tag{6.7}$$

Substituting equation 6.7 and equation 6.4, equation 6.2 becomes:

$$\begin{aligned} P &= \frac{L * R}{(A_S)^{M_S} * (A_D)^{M_D}} \\ M_S &= \frac{(logL + logR - logP - M_D * logA_D)}{logA_S} \end{aligned} \tag{6.8}$$

When M_D = 0, Equation 6.8 is reduced to Equation 6.6. This simply means that when the $DDPass$ of a $DPass$ is null then the whole $DPass$ is simply a normal password.

If we use $dictionary\ words$ as the character set, A_S and A_D are dramatically increased. It also makes both $DDPAss$ and $SDPass$, and in turn, Dynamic Password, more user-friendly. This increases the effective life time of $DDPass$ and $SDPass$; or equivalently reduce the "length" of $DDPass$ and $SDPass$ for a given effective life time. If four to

six letter words are used, the character sets now consist of approximately 23,000 entities. A $DDPass$ or $SDPass$ of length 2 means the $DDPass$ or $SDPass$ is made up of two words of 4 to 6 letters. Table 6.3 shows the calculated life time of $DPass$ with A_S and A_D = 23000.

Number of characters in normal password	**Effective Life time**	**Number of characters in**	
		SDPass	**DDPass**
5	0.4 s	1	1
6	21 s	2	1
7	1340 s	2	1
8	0.96 day	3	1
9	60 day	3	1
10	10 year	4	1

Table 6.3: Dynamic Password length using 23000-character set.

Apparently, with every new $DDPass$, a new Dynamic Password is formed even though $SDPass$ remains the same. If the $DDPass$ is assigned a lifetime until next login, the Dynamic Password formed is effectively a one-time password [20].

If we assume users login every day, a one-time Dynamic Password has a lifetime of 1 day. From Table 6.3 a normal password with a lifetime of 1 day is 8 characters long. In comparison, a Dynamic Password of lifetime 1 day has a $DDPass$ of one 4-6 letter dictionary word and $SDPass$ of three 4-6 letter dictionary words. As the $SDPass$ remains the same for an extended period of time, users could remember the static $SDPass$ after going through a few login sessions. The daily task of memorizing 8 unrelated digits and alphabets is reduced to memorizing 1 new word only. Both $SDPass$ and $DDPass$ can be lengthened to enhance the strength of Dynamic Passwords formed.

Fact 1: Replay attack is not possible with Dynamic Passwords.

As the $DDPass$ of a Dynamic Password changes with every use of the password, a replay of login request will definitely fail.

Fact 2: Spoofing is not possible either with Dynamic Passwords.

If an adversary impersonates a legitimate user and gains access to an account, $DDPass$ is changed for the next login. Legitimate users are alerted if spoofing has occurred since they cannot login using the correct but expired $DDPass$.

Fact 3: Both $SDPass$ and $DDPass$ when used individually are weak passwords.

When $SDPass$ and $DDPass$ are combined as a Dynamic Password, they form a much stronger password that is not hard to remember.

6.4 Key Exchange using Dynamic Passwords

When we use Dynamic Passwords we need a protocol that authenticates a user and transmits the new $DDPass$ of a Dynamic Password after each login. We first review the existing key exchange schemes in Sections 6.4.1 and 6.4.2 and then present Key Exchange using Dynamic Passwords ($KEDP$) in Section 6.4.3.

6.4.1 Encryption Key Exchange

The classic protocol of Encryption Key Exchange (EKE) by Bellovin and Merritt [9] and its successor Augmented-EKE (A–EKE) [10] initiated a collection of authentication protocols.

EKE uses password as a shared secret to authenticate servers and users indirectly. During EKE message exchanges, passwords are not sent across the communication channel, but are used as a symmetric encryption key. Both parties must have access to the shared password. If the server is compromised and passwords are stolen, an adversary can impersonate a legitimate user. EKE is secure against dictionary attacks as the contents of the messages are generated randomly. It does not provide any clue for guessing the password. However EKE is susceptible to the Denning-Sacco attack [69] in which a stolen session key can be used to launch a replay attack on the password. A method to strengthen EKE against Denning-Sacco Attacks was proposed. A more efficient Minimal EKE (M–EKE) was also proposed [69].

If a server is compromised, all passwords stored in a server may be stolen. To prevent passwords from being stolen from compromised servers, Augmented Encryption Key Exchange (A–EKE) [11] was proposed to strengthen the security of passwords stored in a server. In A–EKE, a one-way hash value of the password is used as the shared secret. The symmetric encryption key is the one-way hashed value of the password instead of the password itself. However, a stolen session key can be used to launch a dictionary attack on the password in A–EKE.

Other alternatives to A–EKE are proposed. Simple Password Exponential Key Exchange ($SPEKE$) [25] uses the hash of the password as the base for exponentiation instead of a fixed primitive base. Extended Password Key Exchange Protocols [26] are a family of protocols that extend the A–EKE and $SPEKE$ protocols to $B-EKE$ and $B-SPEKE$ by using a second Diffie-Hellman exchange instead of a digital signature to prove that a user has the knowledge of a certain password.

Open Key Exchange (OKE) [46] eliminates the use of encryption for the user's public key. Another feature of OKE is that the user's public key can be reused as long as the private key is kept secret.

6.4.2 Asymmetric Key Exchange

Another key exchange scheme is Asymmetric Key Exchange AKE. Based on the swapped-secret approach, Asymmetric Key Exchange (AKE) [78] is another class of protocol that exchanges keys without using encryption. Initially, each party computes a secret and generates a verifier with a one-way hash function. They swap their secrets and use them as long term swapped secrets. For each session, each party generates another session secret and swaps with the other party. The session key is generated from the swapped long term and session secrets. Authentication is completed when both parties confirm that they have the same session key.

AKE is similar to the Diffie-Hellman key exchange approach. Session keys are formed based on the swapped secrets. The difference is that AKE combines some shared secrets in forming the session key. The shared secrets make AKE free from Man-in-the-Middle Attacks. No encryption is used in AKE and it takes only four steps. It is not suitable for mobile users, as both parties must store the verifier of the other party.

6.4.3 Key Exchange using Dynamic Passwords

Neither EKE nor AKE offer transmission of new passwords. The $DDPass$ of a one-time Dynamic Password is actually renewed with every login. In essence, Key Exchange using Dynamic Passwords ($KEDP$) is a password authentication protocol which authenticates users, delivers new $DDPass$ to users and establishes session keys.

Abbr.	Meaning
$DDass$	Dynamic Part of a Dynamic Password
$DDPass_n$	New $DDPass$ selected by Server
$SDPass$	Static Part of a Dynamic Password
$H(X)$	One way hash value of X
R_n	The n^{th} random number
M_n	The n^{th} message
SK	Session Key
$\{X\}_Y$	Symmetric encryption of X using key Y
$\{X\}^Y$	Symmetric decryption of X using key Y
$RandomGen()$	Random string generator
$SDPasswdMap(X)$	Maps X from random string space to $SDPass$ password space with character set [[4-6]letter word]
$DDPasswdMap(X)$	Maps X from random string space to $DDPass$ password space with character set [A-Za-z0-9]

Table 6.4: List of abbreviations for $KEDP$

$KEDP$ consists of two phases: Registration and Login phases. For the ease of description, we list all the abbreviations for $KEDP$ in Table 6.4.

Phase 1: Registration

The registration process is performed at the $KEDPS$ - a server running $KEDP$. The user first selects a unique account name. $KEDPS$ then generates two random numbers to form $SDPass$ and $DDPass$ respectively. The $SDPass$ and $DDPass$ are then transferred to the user via a secure channel. $KEDPS$ uses some one-way function to form hash values of $SDPass$ and $DDPass$. For simplicity, here we assume that the $H(SDPass)$ and $H(DDPass)$ are stored in $KEDPS$ and both $SDPass$ and $DDPass$ are erased from $KEDPS$. The process of registration is shown in Algorithm 1.

Algorithm 1: Registration Phase of Integrated Key Exchange Using Dynamic Passwords

Begin (Server Side)

$Account\ Name \longleftarrow User\ Input$
$H(SDPass) \longleftarrow SDPass \longleftarrow SDPasswdMap(RandomGen())$
$H(DDPass) \longleftarrow DDPass \longleftarrow DDPasswdMap(RandomGen())$
$Secure\ File\ Storage \longleftarrow H(SDPass), H(DDPass)$
$User \longleftarrow DDPass, SDPass$

end

After registration, the server just has $H(SDPass)$ and $H(DDPass)$. The user has to remember $SDPass$ and $DDPass$ while $DDPass$ is changed with every successful login.

Phase 2: Login

An execution of the protocol is shown in Algorithm 2. The authentication can be broken into four processes. We will use:

- $\{X\}_Y$ denotes X is encrypted using key Y.
- $\{X\}^Y$ denotes X is decrypted using key Y.

Process 1: To start a login session, the user inputs the account name, $SDPass$ and $DDPass$ to $KEDPC$ - an application running $KEDP$ on user side. $KEDPC$ generates a random string R_1, $H(SDPass)$, $H(DDPass)$ and forms message M_1:

$$M_1 = \{R_1\}_{H(SDPass)} \tag{6.9}$$

$KEDPC$ deletes $SDPass$, $DDPass$ and $H(SDPass)$ because they are not needed by $KEDPC$ any more. However $KEDPC$ holds R_1 and $H(DDPass)$ until the session key is established. M_1 and the account name are sent to $KEDPS$.

Algorithm 2: An execution of Key Exchange using Dynamic Passwords

Begin (Process 1, Client Side)
$DDPass,\ SDPass \longleftarrow UserInput$
$R_1 \longleftarrow RandomGen()$
$M_1 \longleftarrow \{R_1\}_{H(SDPass)}$
end

$M_{from\ Client} \longrightarrow Server$

Begin (Process 2, Server Side)
$H(DDPass),\ H(SDPass) \longleftarrow Secure\ File\ Storage$
$R_1 \longleftarrow \{M_1\}^{H(SDPass)}$
$R_2 \longleftarrow RandomGen()$
$DDPass_n \longleftarrow DDPasswdMap(RandomGen())$
$M_2 \longleftarrow \{R_2, DDPass_n\}_{R_1}$
end

$Client \longleftarrow M_2(fromServer)$

Begin (Process 3, Client Side)
$R_2,\ DDPass_n \longleftarrow \{M_2\}^{R_1}$
$R_3 \longleftarrow RandomGen()$
$M_3 \longleftarrow \{R_3, H(DDPass)\}_{R_2}$
$SK \longleftarrow H(R_1, R_2, R_3)$
$DDPass \longleftarrow DDPass_n$ Upon successful use of SK
end

$M_3(fromClient) \longrightarrow Server$

Begin (Process 4, Server Side)
$R_3,\ H(DDPass) \longleftarrow \{M_3\}^{R_2}$
Compare $H(DDPass)$ with stored value
$SK \longleftarrow H(R_1, R_2, R_3)$
$H(DDPass) \longleftarrow H(DDPass_n)$ Upon successful use of SK
end

Process 2: When $KEDPS$ receives M_1 and the account name from $KEDPC$, $KEDPS$ retrieves the corresponding $H(SDPass)$. $KEDPS$ decrypts M_1 and gets R_1. $KEDPS$ generates a new $DDPass_n$, a random string R_2 and forms message M_2 which is sent to $KEDPC$.

$$M_2 = \{R_2, DDPass_n\}_{R_1} \tag{6.10}$$

Process 3: Upon receiving M_2, $KEDPC$ decrypts M_2 using the stored value of R_1 and extracts the random string R_2. $KEDPC$ generates a random string R_3 and forms message

M_3. M_3 is sent to $KEDPS$.

$$M_3 = \{R_3, H(DDPass)\}_{R_2} \tag{6.11}$$

$$SK = H(R_1, R_2, R_3) \tag{6.12}$$

Upon successful use of SK, $KEDPC$ would display $DDPass_n$ to the user. The user has to use this new $DDPass_n$ in place of the old $DDPass$ If $KEDPC$ does not receive a valid M_2 after a predefined period of time, $KEDPC$ may assume that $KEDPS$ has not received the login message M_1. So $KEDPC$ would delete all stored information about the user and prompt the user for a new login.

Process 4: After receiving M_3, $KEDPS$ decrypts M_3 and extracts R_3 and $H(DDPass)$. $H(DDPass)$ is compared with the corresponding stored hash values. If they match, $KEDPS$ then forms SK as in Equation 6.12.

Upon successful use of SK, $KEDPS$ would replace the stored hash value $H(DDPass)$ with the new value $H(DDPass_n)$. If $KEDPS$ does not receive a valid M_3 after a predefined period of time, $KEDPS$ may assume that $KEDPC$ has not received the message M_2. So $KEDPC$ then delete all stored information about the user and prompt the user for a new login. $KEDPC$ cannot communicate with $KEDPS$ using SK and so do not display $DDPass_n$ to the user as the new $DDPass$.

6.5 Features of Proposed Key Exchange using DPass

Apparently $KEDP$ is a **Mutual Authentication** protocol. $KEDP$ provides explicit user authentication and implicit server authentication. If the user does not provide the proper $DDPass$ in M_3, user authentication fails. If the user cannot communicate with the server using SK, it means that one or more of the random strings R_1, R_2 and R_3 do not have the same corresponding values in the server. It implies that the server has not use the proper $H(SDPass)$ to decrypt M_1. So server authentication fails.

Fact 4: $KEDP$ is free from **Dictionary Attacks**.

If the messages exchanged in the authentication process contain publicly known information that can be used to validate a password guess, dictionary attacks are possible. However, in $KEDP$, the contents of messages M_1, M_2 and M_3 are listed in equations (6.9, 6.10, 6.11). Due to the character set of $DDPass$ and $DDPass_n$ is [A-Za-z0-9], R_1, R_2 and R_3 are randomly generated, so the content of M_1, M_2 and M_3 do not provide any extra information for the adversary to validate their password guesses.

Fact 5: $KEDP$ is resistant to on-line Brute Force Attack.

After each successful login, $KEDPS$ stores a new $H(DDPass_n)$ in its secret files. The user is prompted to remember the new $DDPass_n$ displayed on the screen for next login. If an adversary replays any previous login messages, the $DDPass$ used in M_3 cannot

form the same hash value as the $H(DDPass)$ stored in $KEDPS$. User authentication fails. This prevents adversaries from launching replay attacks.

Increasing the length of $SDPass$ and decreasing the lifetime of $DDPass$ increase the strength of the Dynamic Password formed.

Three-strike rule mandates an account be suspended temporarily after three consecutive log-in failure. Keeping the user identity list private [18], implementing Dynamic Password and three-strike rule helps to protect against **Denial of Service Attack** .

Fact 6: $KEDP$ makes **Spoofing** much harder to succeed.

The easiest way to compromise passwords is by looking over the shoulder of users when they enter their passwords. $KEDP$ cannot stop shoulder surfing from adversaries. But $KEDP$ makes such an attack harder with $DDPass$ in the Dynamic Password. An adversary has to look over the user's shoulder while the user enters the $SDPass$ and also when the new $DDPass$ is displayed by $KEDPC$. The spying action is thus made more conspicuous and suspicious. The chance that an adversary can get both the $SDPass$ and the new $DDPass$ without alerting the user is much less than with simple password entry.

Fact 7: Revealing Spoofing can easily be review when $KEDP$ is use.

If an adversary has knowledge of a valid account name and password, he can login to the account and just read information without changing anything. It is hard for a user to discover that an adversary has gained access to his account.

As the $DDPass$ of the user changes for every successful login, the $DDPass$ is changed by the spoofed login. The legitimate user still assumes and uses the expired $DDPass$ to initiate a login session after spoofing. Obviously the user cannot login. At this point, the user must contact the server administrator and re-initialize his $SDPass$ and $DDPass$. He can then check with the server administrator as to the time of his last successful login. The last login time may reveal whether spoofing has occurred or not.

6.6 Application of KEDP

To apply $KEDP$, all we have to do is to determine the lengths of the $SDPass$ and the $DDPass$, the one-way hash algorithm and the symmetric encryption algorithm.

There are many hash algorithms and encryption algorithms which can be used in $KEDP$. One common one-way hash algorithm is the MD5 [66] algorithm which produces 128 bit hash values. SHA1 produces 160 bit hash values. Triple DES may be used as the symmetric encryption algorithm. $SDPass$ can be 4 random words from a set of 4-6 letter dictionary words. $DDPass$ can be 4 or more random characters from the character set [A-Za-z0-9].

6.6.1 Distributed Computer Networks Systems

We can apply $KEDP$ in its simplest form without any adaptation. In a typical client-server situation, $KEDP$ can provide the necessary mutual authentication between client

and server.

In a distributed computer networks system, a user can login to an authentication server and obtain session keys or service tokens to be used with other servers within the system. Using $KEDP$, the traffic between the authentication server and the user are protected by symmetric encryptions that have different keys for every message. The random nature of the authentication message content makes the authentication process immune to dictionary attacks. The ability of $KEDP$ to reveal spoofing makes the user more confident about personal privacy and system security within the distributed computer networks system.

6.6.2 KEDP Ported to Kerberos

We can also port $KEDP$ to other authentication schemes using passwords. For instance, Kerberos is an authentication protocol for accessing distributed resources across the network. Kerberos authenticates and grants tickets to users. Tickets enable users to use different services within a certain period of time without any further need to authenticate themselves. The only major drawback is its vulnerability to dictionary attack. $KEDP$ is practically immune from dictionary attack and complements the password authentication part of Kerberos. Porting $KEDP$ to Kerberos involves some minimal changes to the message exchange steps in Kerberos.

In the following Section, we give a detailed illustration of how $KEDP$ is applied on Kerberos. The abbreviations used in the Kerberos message exchange are listed in Table 6.5. During Kerberos message exchange, an adversary can easily pretend to be a legitimate user

Abbr.	Meaning
C	Account User Name
T	Ticket Granting Server name
KDC	Key Distribution Center
K_X	Secret key of entity X
K_{XY}	Session key between entities X and Y
$\{M\}_K$	Message M encrypted with key K

Table 6.5: List of abbreviations used in Kerberos message exchange.

and sends KDC an initial plain request. KDC just returns message 2 to the adversary without any authentication. The session key between the TGS and the client is encrypted with the secret key of the client, which is derived from the user password using some known function. The adversary can then start dictionary attacks or brute-force attacks on the encrypted session key and validate his guesses against some known format data encrypted along with the session key.

$KEDP$ can be applied to Kerberos. The shared secret changes from a secret symmetric key to hashes HSP and $HHDP$. Message M_1 and message M_2 need to be changed with

an added step between message M_2 and message M_3. The rest remains the same.

$$M_1 = C, T \quad \text{Plain request from client to server changes to} \tag{6.13}$$

$$M_1 = \{R_3, H(DDPass)\}_{H(SDPass)} \tag{6.14}$$

$$M_2 = \{K_{CT}\}_{K_C} \quad \text{The } KDC \text{ encrypted reply message changes to} \tag{6.15}$$

$$M_2 = \{K_{CT}, DDPass_n\}_{R_1} \tag{6.16}$$

An added message M_{2a} is sent from the user to KDC:

$$M_{2a} = \{R_3, H(DDPass)\}_{H(SDPass)} \tag{6.17}$$

6.7 Chapter Summary

In this chapter, we introduced Dynamic Password and the associated key exchange process. Dynamic Password allows us to have the benefit of user friendly strong passwords. The Dynamic password and $KEDP$ have many desirable features. The authentication using Dynamic Password is immune to replay and dictionary attacks. The associated key exchange process also reveals spoofing and is resistant to social engineering and spoofing.

Chapter 7

KEDP in Service Network Graph

The Dynamic Password ($DPass$) and its associated Key Exchange using Dynamic Password protocol ($KEDP$) presented in Chapter 6 provide users with user friendly strong passwords but are relatively easy to remember.

In this chapter, we apply $DPass$ and $KEDP$ to SNG. A correctness proof of the $KEDP$ protocol using Strand Spaces and Bundles is also discussed in this chapter. For clarity, we use DPS to represent $DPass$ and $KEDP$ together.

We have established the proposition that an encrypted channel speaks for a client [37] using Lampson's theory in Chapter 5. So we begin this chapter with the definitions for strand spaces. Then we extend the definitions to strand space for Dynamic Password and $KEDP$ when applied to SNG. We then proceed to prove the correctness of Dynamic Password and $KEDP$ by demonstrating that both parties are guaranteed of the agreement and secrecy of information exchanged.

This chapter is organized as follows. The basic notions of terms, subterm, free encryption, free message algebra, Strand Spaces and penetrator strands [17] are given in Section 7.1. In Section 7.3, we map the DPS protocol to the corresponding DPS strand space. The agreement and secrecy properties of DPS strand space are shown in Section 7.4.

7.1 Strand Spaces and Related Notions

It is a common practice to specify the actions and events in a protocol using some formal specification language such as BAN [13, 4] or Communicating Sequential Processes (CSP) [22, 23], and analyze the protocol at different levels of abstraction. Various techniques and models for proving correctness of security protocols are proposed [56, 3, 68]. For example, $The\ Inductive\ Approach$ which is based on predicate calculus, has been used to prove public key protocol $Needham - Schroeder$, shared key protocol $Otway-Rees$, and recursive protocol $Recursive\ Authentication\ Protocol$. $Strand\ Spaces$ model

was proposed by Thayer, Herzog and Guttman [16, 17] in 1998. The Strand Spaces approach leads to a precise characterization of the validity of security protocols. Strand Spaces captures causally relevant information which simplifies inductive arguments and aids in isolating the properties the protocol satisfies [16]. In the Strand Spaces model, behavior of system penetrators are explicitly stated; the semantics of assumptions about freshness and uniqueness of nonce and session keys are also clearly stated. Based on the advantages of using Strand Spaces, we choose to use the Strand Spaces approach to prove the correctness of $KEDP$ and DPS when applied to SNG.

In this section, an overview of strand spaces base on [17] is given. We start with a basic notion, $term$.

Definition 7.1 Term

A term is either a text message, a key, a password, a hash value, or an encrypted message.

When a term R is used as a key, the corresponding decryption key is denoted by R^{-1}.

Definition 7.2 Encrypted Term

When a text message g is encrypted with a key K, the encrypted message is also a term represented by: $\{g\}_K$.

Let $\mathcal{A}$ be the set of possible messages exchanged between two principals in a protocol run. Other sets of terms used in our discussion includes:

1. $\mathcal{T}$ is a set of text messages and $\mathcal{T} \subset \mathcal{A}$
2. $\mathcal{E}$ is a set of encrypted messages and $\mathcal{E} \subset \mathcal{A}$
3. $\mathcal{K}_p$ is a set of compromised keys and $\mathcal{K}_p \subset \mathcal{A}$

In particular, the conditions and operators associated with the sets are:

- $\mathcal{T}$, $\mathcal{K}_p$ and $\mathcal{E}$ are disjoint.
- $\mathcal{A}$ has three operations:
  - $hash : \mathcal{A} \rightarrow \mathcal{K}_p$
  - $encr : \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{E}$ and
  - $join : \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$

Definition 7.3 Sign of a term

When a principal transmits a term, that occurrence of the term has a positive sign. A negative term indicates that the term was received by a principal.

Axiom 7.4 *For* $m_0,\ m_0',\ m_1,\ m_1' \in \mathcal{A}$ *and* $K,\ K' \in \mathcal{T}$

1. $(m_0 m_1 = m_0' m_1') \Rightarrow (m_0 = m_0') \wedge (m_1 = m_1')$
2. $m_0 m_1 \neq \{m_0\}_K \neq hash(m_0)$
3. $m_0 m_1 \notin \mathcal{T}$
4. $\{m_0\}_K \notin \mathcal{T}$

Definition 7.5 Strands

A $strand$ *is a sequence of message transmissions and receptions for a single principal. It is a sequential process with no internal nor external choice [23].*

We also use axioms, definitions, notational conventions and propositions from [17] listed in Table 7.1 and Table 7.2.

Item	**About**	**Listed in Appendix**
Axiom 1	It is about free encryption.	B.1
Axiom 2	It is about free algebra. We use an adapted version as shown in Axiom 7.4.	B.4
Proposition 2.12	It is about the properties of a subterm.	B.5
Notational Convention 2.5	Convention for specifying a bundle.	B.9
Lemma 2.7	Each bundle has a minimal member.	B.11
Lemma 2.8	Minimal member of a bundle has a positive term.	B.12
Lemma 2.9	The term of the minimal member of a bundle originates from the minimal member.	B.13
Proposition 3.3	It is about the ability of a penetrator in compromising encryption and decryption keys.	B.15

Table 7.1: Axioms, notational conventions and propositions from [17]

Item	About	Listed in Appendix
Definition 2.1	It defines a signed term.	B.2
Definition 2.11	It defines the subterm relationship.	B.3
Definition 2.2	It defines a strand space.	B.6
Definition 2.3	It lists the properties of a strand space.	B.7
Definition 2.4	It lists the properties of a bundle.	B.8
Definition 2.6	It defines the properties of edges in a bundle.	B.10
Definition 3.1	It defines a penetrator trace.	B.14
Definition 3.2	This defines an infiltrated strand space.	B.16

Table 7.2: Definitions from [17]

7.2 Authenticating an Encrypted Channel

Generally, a computer system communicates with the outside world through some channel, such as a network connection, a system call from user applications, or a serial connection from a terminal. If the traffic is encrypted, the channel is referred to as an encrypted channel. Authenticating a channel amounts to identifying the user with whom the system is communicating.

We have shown how encrypted channels authenticate themselves [37] in Chapter 5. So we now proceed to map DPS to a strand space and prove its correctness.

7.3 Mapping DPS to a Strand Space

In Chapter 6, Dynamic Password and its associated $KEDP$ (DPS) [33] was proposed as an authentication protocol which can reveal any successful spoofing. Suppose A, B are two principals in a Strand Space. The sequence of messages exchanged are reviewed as follows,

Step 1 A initiates a request with a nonce R_1:

$$\{R_1\}_{H(SP)_{AB}}$$

and the message is sent from A to B.

Step 2 B responds with another nonce R_2 and a new dynamic part of DPS $DDPass_{new}$:

$$\{R_2 DDPass_{new}\}_{R_1}$$

and the message is sent from B to A.

Step 3 A responds with another nonce R_3 and the hash of original dynamic part of DPS to confirm to B that $DDPass_{new}$ was received successfully:

$$\{R_3 H(DDPass)_{AB}\}_{R_2}$$

and the message is sent from A to B. This message also serves as a spoofing attack indicator.

Due to the fact that DPS takes advantage of symmetric keys for encryption and hence no specific cryptographic system are required, we assume free encryption and free message algebra.

First, let us take a look at the relationship between the events in DPS key exchange protocol and Strand Space model. In a typical execution of DPS key exchange protocol, both A and B undergo a sequence of events which have causal relationship with each other as shown in Figure 7.1.

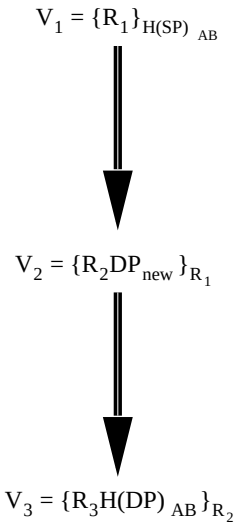


Figure 7.1: DPS events for a principal

Figure 7.2 shows the causal links between events that happen in A and those that happen in B. Note that we use $\Rightarrow$ to indicate the causal link between events pertaining to a single principal, and $\rightarrow$ to show the causal link between events among principals.

Apparently, DPS events fit easily into the Strand Space model with the group of events that happen in A forming a strand and the group of events that happen in B forming another strand. The strands are linked and form a bundle which represents a typical protocol run as shown in Figure 7.3

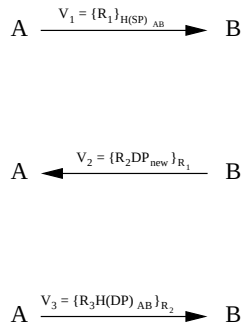


Figure 7.2: Causal links between DPS events for principal A and B

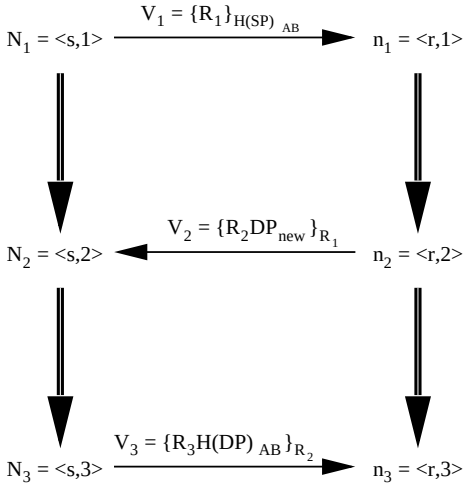


Figure 7.3: A DPS Bundle

7.3.1 DPS Strand Spaces

We can now give a formal definition for the DPS strand space assuming free algebra and free encryption hold. Referring to Figure 7.3, we now define an infiltrated DPS strand space:

Definition 7.6 *An infiltrated strand space Σ, $\mathcal{P}$ is a DPS space if Σ is the union of three kinds of strands:*

1. *Penetrator strands $p \in \mathcal{P}$;*
2. *"Initiator strands from A to B" $s \in$ Init$[R_1, R_2, R_3, DDPass_{new}, H(DDPass)_{AB}]$ with trace:*

$$\langle +\{R_1\}_{H(SDPass)_{AB}}, -\{R_2 DDPass_{new}\}_{R_1}, +\{R_3 H(DDPass)_{AB}\}_{R_2} \rangle$$

And

- *Init[*$R_1, R_2, R_3, DDPass_{new}, H(DDPass)_{AB}$*] denotes the set of all strands with the trace shown.*
- *The principal associated with this strand is A.*
- $H(SDPass)_{AB}^{-1}, R_1, R_1^{-1}, R_2, R_2^{-1}, R_3, H(DDPass)_{AB}, DDPass_{new} \notin \mathcal{K}_p$*;*
- $R_1 \neq R_2 \neq R_3 \neq DDPass_{new} \neq H(DDPass)_{AB} \neq H(SDPass)_{AB}$*;*
- $R_1,\ R_2,\ R_3, DDPass_{new},\ H(DDPass)_{AB},\ andH(SDPass)_{AB}$ *are uniquely originating in* Σ*;*

3. *Complementary "Responder strands from B to A"*
   r ∈ *Resp[*$R_1, R_2, R_3, DDPass_{new}, H(DDPass)_{AB}$*] with trace:*

$$\langle -\{R_1\}_{H(SDPass)_{AB}}, +\{R_2 DDPass_{new}\}_{R_1}, -\{R_3 H(DDPass)_{AB}\}_{R_2} \rangle$$

   And

   - *Resp[*$R_1, R_2, R_3, DDPass_{new}, H(DDPass)_{AB}$*] denotes the set of all strands with the trace shown.*
   - *The principal associated with this strand is B.*
   - $H(SDPass)_{AB}^{-1}, R_1, R_1^{-1}, R_2, R_2^{-1}, R_3, H(DDPass)_{AB}, DDPass_{new} \notin \mathcal{K}_p$*;*
   - $R_1 \neq R_2 \neq R_3 \neq DDPass_{new} \neq H(DDPass)_{AB} \neq H(SDPass)_{AB}$*;*
   - $R_1,\ R_2,\ R_3, DDPass_{new},\ H(DDPass)_{AB},\ andH(SDPass)_{AB}$ *are uniquely originating in* Σ*;*

In order to prove the correctness of DPS, we need to define two correctness concepts according to Woo and Lam [77] and Fabrega, Herzog and Guttman [17]: agreement and secrecy.

Definition 7.7 Agreement

A protocol guarantees agreement to a principal B (the responder) for certain data item x if:

> *Each time a principal B completes a run of the protocol as a responder using x, which to B appears to be a run with A, then there is a* unique *run of the protocol with the principal A as initiator using x, which to A appears to be a run with B.*

From this definition, we can prove agreement by showing that whenever a bundle C contains a strand representing a responder run using x, then C also contains a *unique* strand representing an initiator run that corresponds in the sense that it also uses x. The agreement proof can be split into two tasks:

1. Show that such a corresponding initiator strand exists in C.

2. Show that the initiator strand is unique.

Another requirement for an authentication protocol is secrecy of data exchanged.

Definition 7.8 Secrecy
A value x is secret in a bundle C if for every n ∈ C, term(n) ≠ x.

In simple terms, a value is never "said on line" (as term(n)), then it is secret. It means that not only the participants never send x (on line), the penetrator cannot send x (on line) also. Now if a penetrator can derive x from values he receives, the penetrator is capable of sending it. If a penetrator cannot send x, he cannot derive it from values that it receives.

7.4 Correctness Proof of DPS

In this section, we show the secrecy property first, followed by the agreement property.

7.4.1 Secrecy Guarantee

In this section, we show the secrecy of R_1, R_2, R_3, $DDPass_{new}$ and $H(DDPass)$.

Secrecy of R_1

Secrecy of R_1 is stated in Proposition 7.9.

Proposition 7.9 *Suppose:*

1. *Σ is a DPS space, C is a bundle in Σ, and s is an initiator strand in Init[$R_1, R_2, R_3, DDPass_{new}, H(DDPass)_{AB}$];*

2. *$R_1^{-1}, R_2^{-1}, H(SDPass)_{AB}^{-1} \notin \mathcal{K}_p$;*

3. *$R_1,\ R_2,\ R_3, DDPass_{new},\ and\ H(DDPass)_{AB}$ are uniquely originating in Σ;*

Then for all nodes m ∈ C such that $R_1 \sqsubset$ term(m), either $V_1 \sqsubset$ term(m) or $V_2 \sqsubset$ term(m). In particular, $R_1 \neq$ term(m).

This proposition states that R_1 only appears in

$V_1 = \{R_1\}_{H(SDPass)_{AB}}$ or $V_2 = \{R_2 DDPass_{new}\}_{R_1}$

only and so remain secret to adversaries. In other words, we show that R_1 appears in V_1 or V_2 only, and thus remains secret.

The set of nodes which makes Proposition 7.9 invalid is defined as $\mathcal{S}$.

$\mathcal{S} = \{m \in C : (R_1 \sqsubset term(m)) \wedge (V_1 \not\sqsubset term(m)) \wedge (V_2 \not\sqsubset term(m))\}$

If $\mathcal{S}$ is non-empty, then it has at least one $\preceq$-minimal element by Lemma 2.7 in [17]. We prove that the minimal element can neither be regular node nor penetrator node and thus it does not exist and so $\mathcal{S}$ is empty. In other words, Proposition 7.9 holds.

To prove Proposition 7.9 we use the following four steps:

Step 1 R_1 originates at N_1.

Step 2 A minimal node n of $\mathcal{S}$ is not a regular node.

Step 3 A minimal node n of $\mathcal{S}$ is not a penetrator node.

Step 4 $\mathcal{S}$ is empty. R_1 does not appear as $R_1 \sqsubset term(m)$ and secrecy of R_1 is guaranteed.

Step 1 Lemma 7.10 states that R_1 originates at N_1.

Lemma 7.10 *R_1 originates at N_1.*

PROOF: From Definition2.3 in [17] about strands properties, if R_1 originates at n, then $R_1 \sqsubset$ term(n), the sign of n is positive and no other preceding node in the same strand can have R_1 as subterm. From the DPS Strand definition, Definition 7.6, the positive nodes are N_1, n_2, and N_3 with $R_1 \sqsubset term(N_1)$ and $R_1 \sqsubset term(n_2)$.

1. Node n_2
   $n_1 \prec n_2$ and $R_1 \sqsubset$ term(n_1). This contradicts the definition of origin of a term in strands. Hence R_1 does not originates at n_2.

2. Node N_3
   $R_1 \not\sqsubset$ term(N_3) and so R_1 does not originates at N_3.

3. Node N_1
   There is no preceding node in the (initiator) strand such that $R_1 \sqsubset$ term(n). Hence N_1 does not get R_1 from some preceding node and re-transmit it.

The conclusion is that R_1 originates from N_1. ■

Step 2 Lemma 7.11 states that the minimal member of $\mathcal{S}$ does not originates from a regular node.

Lemma 7.11 *No minimal member of $\mathcal{S}$ is a regular node.*

PROOF: Suppose $m \in \mathcal{S}$ is minimal and regular. By Lemma 2.8 in [17] , the sign of m is positive.

For m to be regular, m can either be on initiator strand s; on another initiator strand s'; on a responder strand r; or another responder strand r'. We show that none of the cases is true and so minimal member of $\mathcal{S}$ cannot be a regular node.

1. If m is on s, the positive nodes are $< s, 1 >$ and $< s, 3 >$.

   - Term($< s, 1 >$) = V_1 and so m cannot be $< s, 1 >$.
   - Term($< s, 3 >$) = V_3. As $R_1 \not\sqsubset V_3$, m cannot be $< s, 3 >$.

   So m is not on s.

2. If m is on another initiator strand $s' \neq$ s and

   (a) m = $< s', 1 >$: $R_1 \sqsubset$ term($< s', 1 >$), so R_1 originates from $< s', 1 >$. This contradicts Lemma 7.10 which states that R_1 originates from N_1.

   (b) m = $< s', 3 >$: $R_1 \sqsubset$ term($< s', 3 >$) and so $R_1 \sqsubset$ term($< s', 2 >$). As $< s', 2 > \prec < s', 3 >$, it contradicts the minimality of m.

3. If m is on a responder strand r, the only positive node is $< r, 2 >$. Term($< r, 2 >$) = V_2 and so m cannot be $< r, 2 >$.

4. If m is on another responder strand r', the only positive node is $< r', 2 >$. Suppose m = $< r', 2 >$, then $R_1 \sqsubset$ term($< r', 2 >$) and $R_1 \sqsubset$ term($< r', 1 >$). Hence $< r', 1 > \prec < r', 2 >$, and contradicts the minimality of m.

Hence we can conclude that m cannot be a regular node. ■

Step 3 Lemma 7.12 states that the minimal member of $\mathcal{S}$ does not originate from a penetrator node either.

Lemma 7.12 *No minimal member of $\mathcal{S}$ is a penetrator node.*

PROOF: Suppose $m \in \mathcal{S}$ is minimal and is a penetrator node in one of the following penetrator traces.

Trace P_M. Text message: $< +t >$ where t $\in \mathcal{T}$

If m is a node in Trace P_M, then $R_1 \sqsubset t$ which means R_1 originates from m. This contradicts Lemma 7.10 that R_1 originates uniquely from N_1.

Trace P_F. Flushing: $< -g >$

Since m is positive and there is no positive node in this trace, m cannot be a node in this trace.

Trace P_T. Tee: $< -g, +g, +g >$

The positive nodes re-transmit the term that is received by a previous node. $< P_T, 1 > \prec m$ and so m (the positive nodes) cannot be minimal. Hence m cannot be a node in this trace.

Trace P_C. Concatenation $< -g, -h, +gh >$

If m is a node in Trace P_C, $(R_1 \sqsubset gh) \wedge (V_1 \not\sqsubset gh) \wedge (V_2 \not\sqsubset gh)$.

If $R_1 \sqsubset gh$ then either $R_1 \sqsubset g$ or $R_1 \sqsubset h$. On the other hand, $V_1 \not\sqsubset gh$ means $V_1 \not\sqsubset g$ and $V_1 \not\sqsubset h$ and similarly, $V_2 \not\sqsubset gh$ means $V_2 \not\sqsubset g$ and $V_2 \not\sqsubset h$. Hence if $R_1 \sqsubset g$, $< P_E, 1 > \prec m$ or if $R_1 \sqsubset h$, $< P_E, 2 > \prec m$. Thus m cannot be a node in this trace.

Trace P_K. Key: $< +K_0 >$ where $K_0 \in \mathcal{K}_p$

This trace requires $R_1 \in \mathcal{K}_p$. This contradicts assumptions of a DPS strand that $R_1 \notin \mathcal{K}_p$.

Trace P_E. Encryption: $< -K_0, -h, +\{h\}_{K_0} >$

Suppose $m = < P_E, 3 >$, in which case

$$R_1 \sqsubset \{h\}_{K_0}) \wedge (V_1 \not\sqsubset \{h\}_{K_0}) \wedge (V_2 \not\sqsubset \{h\}_{K_0}$$

As $R_1 \sqsubset \{h\}_{K_0}$, then $R_1 \sqsubset h$ or $R_1 = K_0$. Moreover, $V_1 \not\sqsubset \{h\}_{K_0}$ means $V_1 \not\sqsubset h$ and $V_1 \neq K_0$; $V_2 \not\sqsubset \{h\}_{K_0}$ means $V_2 \not\sqsubset h$ and $V_2 \neq K_0$. Hence $< P_E, 2 > \prec m$ and contradicts the minimality of m.

Trace P_D. Decryption: $< -K_0^{-1}, -\{h\}_{K_0}, +h >$ where $K_0 \in \mathcal{K}_p$

Suppose $(R_1 \sqsubset h) \wedge (V_1 \not\sqsubset h) \wedge (V_2 \not\sqsubset h)$. Consequently, either $V_1 = \{h\}_{K_0}$ or $V_2 = \{h\}_{K_0}$.

$V_1 = \{h\}_{K_0}$**:** Free encryption requires $h = R_1$ and K_0 = $H(SDPass)_{AB}$. Hence there must be a node w which precedes m with $H(SDPass)^{-1}_{AB} \sqsubset$ term(w). In DPS, $H(SDPass)^{-1}_{AB}$ = $H(SDPass)_{AB}$. As $H(SDPass)^{-1}_{AB} \notin \mathcal{K}_p$ and together with Proposition3.3 in [17] , we can infer that $H(SDPass)^{-1}_{AB}$ originates

from a regular node. We cannot find a regular node originating $H(SDPass)^{-1}_{AB}$. Thus node w does not exist in this case and $V_1 \neq \{h\}_{K_0}$.

$V_2 = \{h\}_{K_0}$**:** Free encryption requires $h = R_2 DDPass_{new}$ and K_0 = R_1. Hence there must be a node w which precedes m with $R_1^{-1} \sqsubset$ term(w). As $R_1^{-1} \notin \mathcal{K}_p$ and together with Proposition3.3 in [17] , we can infer that R_1^{-1} originates from a regular node.

For DPS, R_1^{-1} is the same as R_1. The regular node from which R_1 originates is N_1 and the message is V_1. As V_1 is encrypted with $H(SDPass)_{AB}$ and $H(SDPass)^{-1}_{AB} \notin \mathcal{K}_P$, penetrator cannot extract R_1. Thus $V_2 \neq \{h\}_{K_0}$.

As both $V_1 \neq \{h\}_{K_0}$ and $V_2 \neq \{h\}_{K_0}$ are not valid, m cannot be a node on the D penetrator trace.

Trace P_S**.** Separation: $< -gh, +g, +h >$

Assuming term(m) = g. A similar argument applies to term(m) = h.

As m is minimal in $\mathcal{S}$, $R_1 \sqsubset g$, $V_1 \not\sqsubset g$ and $V_2 \not\sqsubset g$. Consequently, either $V_1 \sqsubset gh$ implying $V_1 \sqsubset h$ or $V_2 \sqsubset gh$ implying $V_2 \sqsubset h$.

We are going to show that no node can send gh to node m and hence m is not in Trace S.

Let $\mathcal{Z} = \{X \in C : (X \prec m) \wedge (gh \sqsubset term(m))\}$.
$\mathcal{Z}$ has at least one element $< P_S, 1 >$; and $\mathcal{Z}$ has a minimal element X by Lemma 2.7 in [17] ; and X must be positive (Lemma 2.8 in [17]).

No regular node have an encrypted term as a subterm of the form gh and so X cannot be a regular node. We are going to show that X cannot be a penetrator node either.

Trace P'_M**.** Text message: $< +t' >$ where t $\in \mathcal{T}$

It requires $R_1 \sqsubset t'$ and originates from $< P'_M, 1 >$, contradicting Lemma 7.10 which states that R_1 originates in N_1.

Trace P'_F**.** Flushing: $< -g' >$

There is no positive node, and so m cannot be on this trace.

Trace P'_T**.** Tee: $< -g', +g', +g' >$

If X = $< P'_T, 2 >$ or X = $< P'_T, 2 >$, it requires $gh \sqsubset < P'_T, 1 >$ which means $< P'_T, 1 > \prec X$ and contradicts the minimality of X.

Trace P'_K**.** Key: $< +K'_0 >$

It requires $R_1 \sqsubset K'_0$ and originates from $< P'_K, 1 >$, contradicting Lemma 7.10 which states that R_1 originates in N_1.

Trace P'_S**.** Separation: $< -g'h', +g', +h' >$

Suppose X = $< P'_S, 3 >$ (or X = $< P'_S, 2 >$), then $gh \sqsubset$ term($< P'_S, 1 >$) which implies X is not the minimal element in $\mathcal{Z}$.

Trace P'_E. Encryption: $< -K'_0, -h', +\{h'\}_{K'_0} >$
Suppose $X = < P'_E, 3 >$, then $gh \sqsubset$ term($< P'_E, 2 >$) which implies X is not the minimal element in $\mathcal{Z}$.

Trace P'_D. Decryption: $< -K_0'^{-1}, -\{h'\}_{K'_0}, +h' >$
Suppose $X = < P'_D, 3 >$, then $gh \sqsubset$ term($< P'_D, 2 >$) which implies X is not the minimal element in $\mathcal{Z}$.

Trace P'_C. Concatenation: $< -g', -h', +g'h' >$
Suppose $X = < P'_C, 3 >$, then $gh \sqsubset$ term($< P'_D, 3 >$). So P'_C has trace $< -g, -h, +gh >$. Hence term($< P'_C, 1 >$) = term(m). However, $< P'_C, 1 > \prec m$ which contradicts the minimality of m in $\mathcal{S}$.

None of the penetrator cases ($P'_M, P'_F, P'_T, P'_K, P'_S, P'_E, P'_D, and P'_C$) is true, and so X cannot be a node on the penetrator P_S.

As none of the penetrator cases ($P_M, P_F, P_T, P_K, P_S, P_E, P_D, and P_C$) is true, and so m cannot be a penetrator node. ■

Step 4 Lemma 7.11 says m cannot be a regular node. By Lemma 7.12, m cannot be a penetrator node either. So $\mathcal{S}$ must be empty. Hence we can conclude that R_1 does not appear as $R_1 \sqsubset term(m)$ except $R_1 \sqsubset V_1$ and $R_1 \sqsubset V_2$. The secrecy of R_1 is guaranteed.

PROOF of Proposition 7.9 From Lemma 7.11 and Lemma 7.12, we can draw the conclusion that $\mathcal{S}$ is empty and hence Proposition 7.9 holds. ■

R_1 remains secret during a DPS protocol run.

Secrecy of R_2, R_3, $DDPass_{new}$, and $H(DDPass)$

The proof for secrecy of R_2, R_3, $DDPass_{new}$, **and** $H(DDPass)$ are pretty much the same as R_1. We can follow similar steps as used in R_1 but with different set $\mathcal{S}$. The set $\mathcal{S}$ for each term is listed in Table 7.3.

The Propositions used are also similar to Proposition 7.9. Without repeating the common assumptions, the proposition for each term is listed in Table 7.4.

When proving the secrecy of R_1, R_2, R_3, $DDPass_{new}$ and $H(DDPass)$, we need to establish Lemmas similar to Lemmas 7.10, 7.11, and 7.12. We state two of the Lemmas which is referred to later in the proof. Similar to Lemma 7.10, Lemma 7.13 states that R_2 originates at n_2 and Lemma 7.14 states that $H(DDPass)$ originates at N_3. Proofs of the Lemmas are much the same as Lemma 7.10.

Lemma 7.13 *R_2 originates at n_2.*

Lemma 7.14 *$H(DDPass)$ originates at N_3.*

Term	Set $\mathcal{S}$
R_1	$\mathcal{S} = \{m \in C : (R_1 \sqsubset term(m)) \wedge (V_1 \not\sqsubset term(m)) \wedge (V_2 \not\sqsubset term(m))\}$
R_2	$\mathcal{S} = \{m \in C : (R_2 \sqsubset term(m)) \wedge (V_2 \not\sqsubset term(m)) \wedge (V_3 \not\sqsubset term(m))\}$
R_3	$\mathcal{S} = \{m \in C : (R_2 \sqsubset term(m)) \wedge (V_3 \not\sqsubset term(m))\}$
$DDPass_{new}$	$\mathcal{S} = \{m \in C : (DDPass_{new} \sqsubset term(m)) \wedge (V_2 \not\sqsubset term(m))\}$
$H(DDPass)$	$\mathcal{S} = \{m \in C : (H(DDPass) \sqsubset term(m)) \wedge (V_3 \not\sqsubset term(m))\}$

Table 7.3: $\mathcal{S}$ for R_1, R_2, R_3, $DDPass_{new}$ and $H(DDPass)$

Term	Proposition
R_1	For all nodes m $\in \mathcal{C}$ such that $R_1 \sqsubset$ term(m), either $V_1 \sqsubset$ term(m) or $V_2 \sqsubset$ term(m). In particular, $R_1 \neq$ term(m).
R_2	For all nodes m $\in \mathcal{C}$ such that $R_2 \sqsubset$ term(m), either $V_2 \sqsubset$ term(m) or $V_3 \sqsubset$ term(m). In particular, $R_2 \neq$ term(m).
R_3	For all nodes m $\in \mathcal{C}$ such that $R_3 \sqsubset$ term(m), $V_3 \sqsubset$ term(m). In particular, $R_3 \neq$ term(m).
$DDPass_{new}$	For all nodes m $\in \mathcal{C}$ such that $DDPass_{new} \sqsubset$ term(m), $V_2 \sqsubset$ term(m). In particular, $DDPass_{new} \neq$ term(m).
$H(DDPass)$	For all nodes m $\in \mathcal{C}$ such that $H(DDPass) \sqsubset$ term(m), $V_3 \sqsubset$ term(m). In particular, $H(DDPass) \neq$ term(m).

Table 7.4: Propositions for R_1, R_2, R_3, $DDPass_{new}$ and $H(DDPass)$

7.4.2 DPS Agreement Guarantee

To provide an agreement guarantee to the responder, we have to show that when the responding principal B completes the protocol run using, say, R_2, then there must be a unique run of protocol with initiating principal A using the same R_2.

We can show the agreement guarantee using the steps listed below:

Step 1. Show that R_2 originates at node n_2 on a responder strand r.

Step 2. Show that the minimal node n other than n_2 with R_2 as a subterm and $V_2 \not\sqsubset$ term(n) is regular and the sign of n is positive.

Step 3. Show that the node preceding n is m and term(m) = V_2.

Step 4. Show the regular strand containing both m and n is an initiator strand.

Step 5. Show that the initiator strand is unique.

Similarly, to provide an agreement guarantee to the initiator, we have to show that when the initiating principal A completes the protocol run using, say, R_1, then there must be a unique run of protocol with responding principal B using the same R_1.

We can show the agreement guarantee using the steps listed below:

Step 1. Show that R_1 originates at node N_1 on an initiator strand s.

Step 2. Show that the minimal node n other than N_1 with R_1 as a subterm and $V_2 \not\sqsubset$ term(n) is regular and the sign of n is positive.

Step 3. Show that term(n) = V_2.

Step 4. Show the regular strand containing n is a responder strand.

Step 5. Show that the responder strand is unique.

We show the agreement guarantee for the responder first, followed by the agreement guarantee for the initiator.

Agreement Guarantee for the Responder

We make two propositions for the agreement property of the DPS protocol from the responder's point of view. Proposition 7.15 states that for a responder strand, there exists an initiator strand in $\mathcal{C}$ and Proposition 7.20 states that the initiator strand is unique. The two proposition together state that responder and initiator strand has a one to one correspondence. So proving the two propositions proved the two aspects of agreement property and thus established the agreement property of the DPS protocol for the responder.

Proposition 7.15 *Suppose:*

1. *Σ is an DPS space, C is a bundle in Σ, and r is a responder strand in Resp[$R_1, R_2, R_3, DDPass_{new}, H(DDPass)_{AB}$];*
2. *$R_1^{-1}, R_2^{-1}, H(SDPass)_{AB}^{-1}, \notin \mathcal{K}_p$;*
3. *$R_1,\ R_2,\ R_3, DDPass_{new},\ H(DDPass)_{AB},\ and H(SDPass)_{AB}$ are uniquely originating in Σ;*

Then $\mathcal{C}$ contains an initiator strand s $\in$ Init[$R_1, R_2, R_3, DDPass_{new}, H(DDPass)_{AB}$].

This proposition simply states that for a responder strand, there is a corresponding initiator strand. To prove the proposition, we first establish the existence of a node in an initiator strand with R_2 as subterm. Note that Lemma 7.13 states that R_2 originates at n_2.

Lemma 7.16 *The set $\mathcal{S} = \{n \in C$: $R_2 \sqsubset$ term(n) $\wedge$ $V_2 \not\sqsubset$ term(n) $\}$ has a $\preceq$-minimal node m. Then the node m is regular, and the sign of m is positive.*

PROOF: We would show that $\mathcal{S}$ is non-empty, has a minimal element and positive. Then we eliminate the possibility of penetrator nodes, and establish the fact that it is a regular node.

As $n_3 \in \mathcal{C}$; $R_2 \sqsubset$ term(n_3); and $V_2 \not\sqsubset$ term(n_3); $\mathcal{S}$ is non-empty. So $\mathcal{S}$ has at least one $\preceq$-minimal node m by Lemma 2.7 in [17] and m is positive by Lemma 2.8 in [17] .

We now show that m cannot be a penetrator node and hence establish the fact that m is regular.

m cannot be a node in any one of the penetrator traces listed below.

Trace P_M. Text message: $< +t >$ where t $\in \mathcal{T}$

If m is a node in Trace P_M, then $R_2 \sqsubset t$ which means R_2 originates from m. This contradicts Lemma 7.13 that R_2 originates uniquely from n_2.

Trace P_F. Flushing: $< -g >$

Since m is positive and there is no positive node in this trace, m cannot be a node in this trace.

Trace P_T. Tee: $< -g, +g, +g >$

The positive nodes re-transmit the term that is received by a previous node. $< P_T, 1 > \prec m$ and so m (the positive nodes) cannot be minimal. Hence m cannot be a node in this trace.

Trace P_C. Concatenation $< -g, -h, +gh >$

If m is a node in Trace P_C, $(R_2 \sqsubset gh) \wedge (V_2 \not\sqsubset gh)$.

If $R_2 \sqsubset gh$ then either $R_2 \sqsubset g$ or $R_2 \sqsubset h$. On the other hand, $V_2 \not\sqsubset gh$ means $V_2 \not\sqsubset g$ and $V_2 \not\sqsubset h$. Hence if $R_2 \sqsubset g$, $< P_E, 1 > \prec m$; and if $R_2 \sqsubset h$, $< P_E, 2 > \prec m$. Thus m cannot be a node in this trace.

Trace P_K. Key: $< +K_0 >$ where $K_0 \in \mathcal{K}_p$

This trace requires $R_2 \in \mathcal{K}_p$. This contradicts assumptions of a DPS strand that $R_2 \notin \mathcal{K}_p$.

Trace P_E. Encryption: $< -K_0, -h, +\{h\}_{K_0} >$

Suppose $m = < P_E, 3 >$, in which case $(R_2 \sqsubset \{h\}_{K_0}) \wedge (V_2 \not\sqsubset \{h\}_{K_0})$.
As $R_2 \sqsubset \{h\}_{K_0}$, then $R_2 \sqsubset h$ or $R_2 = K_0$. Moreover, $V_2 \not\sqsubset \{h\}_{K_0}$ means $V_2 \not\sqsubset h$ and $V_2 \neq K_0$. Hence $< P_E, 2 > \prec m$ and contradicts the minimality of m.

Trace P_D. Decryption: $< -K_0^{-1}, -\{h\}_{K_0}, +h >$ where $K_0 \in \mathcal{K}_p$

Suppose $(R_2 \sqsubset h) \wedge (V_2 \not\sqsubset h) \wedge (V_3 \not\sqsubset h)$. Consequently, $V_2 = \{h\}_{K_0}$.

Free encryption requires $h = R_2 DDPass_{new}$ and K_0 = R_1. Hence there must be a node w which precedes m with $R_1^{-1} \sqsubset$ term(w). As $R_1^{-1} \notin \mathcal{K}_p$ and together with Proposition3.3 in [17] , we can infer that R_1^{-1} originates from a regular node.

For DPS, R_1^{-1} is the same as R_1. The regular node from which R_1 originates is N_1 and the message is V_1. Penetrator cannot extract R_1 as V_1 is encrypted with $H(SDPass)_{AB}$ and $H(SDPass)_{AB}^{-1} \notin \mathcal{K}_P$. Thus $V_2 \neq \{h\}_{K_0}$.

As $V_2 \neq \{h\}_{K_0}$, m cannot be a node on the D penetrator trace.

Trace P_S. Separation: $< -gh, +g, +h >$

Assuming term(m) = g. A similar argument applies to term(m) = h.

As m is minimal in $\mathcal{S}$, $H(DDPass) \sqsubset g$, $V_3 \not\sqsubset g$. Consequently, $V_3 \sqsubset gh$ implying $V_3 \sqsubset h$.

We show that no node can send gh to node m and hence m is not in Trace S.

Let $\mathcal{Z} = \{X \in C : (X \prec m) \wedge (gh \sqsubset term(m))\}$.
$\mathcal{Z}$ has at least one element $< P_S, 1 >$; and $\mathcal{Z}$ has a minimal element X by Lemma 2.7 in [17] ; and X must be positive (Lemma 2.8 in [17]).

No regular node have an encrypted term as a subterm of the form gh and so X cannot be a regular node. We are going to show that X cannot be a penetrator node either.

Trace P'_M. Text message: $< +t' >$ where t $\in \mathcal{T}$

It requires $H(DDPass) \sqsubset t'$ and originates from $< P'_M, 1 >$, contradicting Lemma 7.14 which states that $H(DDPass)$ originates in N_3.

Trace P'_F. Flushing: $< -g' >$

There is no positive node, and so m cannot be on this trace.

Trace P'_T. Tee: $< -g', +g', +g' >$

If $X = < P'_T, 2 >$ or $X = < P'_T, 2 >$, it requires $gh \sqsubset < P'_T, 1 >$ which means $< P'_T, 1 > \prec X$ and contradicts the minimality of X.

Trace P'_K. Key: $< +K'_0 >$

It requires $H(DDPass) \sqsubset K'_0$ and originates from $< P'_K, 1 >$, contradicting Lemma 7.14 which states that $H(DDPass)$ originates in N_3.

Trace P'_S. Separation: $< -g'h', +g', +h' >$

Suppose $X = < P'_S, 3 >$ (or $X = < P'_S, 2 >$), then $gh \sqsubset$ term($< P'_S, 1 >$) which implies X is not the minimal element in $\mathcal{Z}$.

Trace P'_E. Encryption: $< -K'_0, -h', +\{h'\}_{K'_0} >$

Suppose $X = < P'_E, 3 >$, then $gh \sqsubset$ term($< P'_E, 2 >$) which implies X is not the minimal element in $\mathcal{Z}$.

Trace P'_D. Decryption: $< -K_0'^{-1}, -\{h'\}_{K'_0}, +h' >$
Suppose $X = < P'_D, 3 >$, then $gh \sqsubset$ term$(< P'_D, 2 >)$ which implies X is not the minimal element in $\mathcal{Z}$.

Trace P'_C. Concatenation: $< -g', -h', +g'h' >$
Suppose $X = < P'_C, 3 >$, then $gh \sqsubset$ term$(< P'_D, 3 >)$. So P'_C has trace $< -g, -h, +gh >$. Hence term$(< P'_C, 1 >)$ = term(m). However, $< P'_C, 1 > \prec m$ which contradicts the minimality of m in $\mathcal{S}$.

None of the penetrator cases ($P'_M, P'_F, P'_T, P'_K, P'_S, P'_E, P'_D, and P'_C$) is true, and so X cannot be a node on the penetrator P_S.

As none of the penetrator cases ($P_M, P_F, P_T, P_K, P_S, P_E, P_D, and P_C$) is true, and so m cannot be a penetrator node. ■

As we have shown that m is not in any of the penetrator's Traces, we can conclude that m is a regular node. Because of its minimality, it is also positive.■

Lemma 7.16 can be re-worded as Definition 7.17:

Definition 7.17 *Fix some m that is $\preceq$-minimal in $\mathcal{S}$ = $\{n \in C$: $R_2 \sqsubset$ term(n) $\wedge$ $V_2 \not\sqsubset$ term(n) $\}$, and is therefore regular and of positive sign.*

Now we have the fact that R_2 originates from a node n_2 in a responder strand, and there exists a minimal node m in $\mathcal{S}$, we show that there exists another node N preceding m with $R_2 \sqsubset$ term(N).

Lemma 7.18 *A node N precedes m on the same strand t as m and term(N) = V_2.*

PROOF: It was shown that R_2 originates uniquely at n_2 (Lemma 7.13). As R_2 does not originates at m, there exists another node N preceding m with $R_2 \sqsubset$ term(N). Because m is minimal in $\mathcal{S}$, the only possible case is $V_2 \sqsubset$ term(N) so that N does not contradict the minimality of m. As no regular node has encrypted term as subterm, we can conclude that V_2 = term(N).■

Now we show m and N are located in an initiator strand.

Lemma 7.19 *The regular strand t containing m and N is an initiator strand and is contained in $\mathcal{C}$.*

PROOF: As term(N) = V_2 and precedes m on the same strand, we have the following cases:

Responder Strand the node after N with term(N) = V_2 is negative and contradicts the fact that m is positive.

Initiator Strand the node after N with term(N) = V_2 is positive and we can conclude that t is an initiator strand and $N = N_2$; $m = N_3$.

The two nodes N and m are the second and third node of an initiator strand, and so t is an initiator strand.■

PROOF of Proposition 7.15
Proposition 7.15 follows from Lemma 7.16, Lemma 7.18 and Lemma 7.19. ■

We have shown that there is an initiator strand in $\mathcal{C}$ to each responder strand. The only remaining thing to show that the initiator strand is unique.

Proposition 7.20 *If Σ is a DPS space, and R_1 is uniquely originating in Σ, then there is at most one strand $t \in$ Init[$R_1,\ R_2,\ R_3,\ H(DDPass)_{AB},\ DDPass_{new}$]$_{AB}$ for any R_2.*

PROOF: By Lemma 7.10, R_1 originates at N_1 which is $< t, 1 >$. If we assume that R_1 uniquely originates in Σ as in assumption clause 4 of Proposition 7.15, then there can only be at most one such initiator strand t. ■

Proposition 7.15 and Proposition 7.20 together show that for a responder strand in $\mathcal{C}$ there is an unique corresponding initiator strand in $\mathcal{C}$. This is the Agreement Guarantee for the responder.

Agreement Guarantee for the Initiator

We follow the same approach as the Agreement Guarantee for Responder and make two propositions initiator's point of view.

Proposition 7.21 *Suppose:*

1. *Σ is an DPS space, C is a bundle in Σ, and s is an initiator strand in Init[$R_1, R_2, R_3, DDPass_{new}, H(DDPass)_{AB}$];*
2. *$R_1^{-1}, R_2^{-1}, H(SDPass)_{AB}^{-1} \notin \mathcal{K}_p$;*
3. *$R_1,\ R_2,\ R_3, DDPass_{new},\ H(DDPass)_{AB},\ and H(SDPass)_{AB}$ are uniquely originating in Σ;*

Then $\mathcal{C}$ contains an responder strand r $\in$ Resp[$R_1, R_2, R_3, DDPass_{new}, H(DDPass)_{AB}$].

We now establish the existence of a node in a responder strand with R_1 as subterm. Note that we have already shown that R_1 originates at N_1 in Lemma 7.10.

Lemma 7.22 *The set $\mathcal{S} = \{n \in C$: $R_1 \sqsubset$ term(n) $\wedge\ V_1 \not\sqsubset$ term(n) $\}$ has a $\preceq$-minimal node m. Then the node m is regular, and the sign of m is positive.*

PROOF: We show that $\mathcal{S}$ is non-empty, has a minimal element and positive. Then we eliminate the possibility of penetrator nodes, and establish the fact that it is a regular node.

As $N_2 \in C$ and $R_1 \sqsubset$ term(N_2) and $V_1 \not\sqsubset$ term(N_2), $\mathcal{S}$ is non-empty. So $\mathcal{S}$ has at least one $\preceq$-minimal node m by Lemma 2.7 in [17] and m is positive by Lemma 2.8 in [17] .

We now show that m cannot be a penetrator node and hence establish the fact that m is regular.

m cannot be a node in any one of the penetrator traces listed below.

Suppose $m \in \mathcal{S}$ is minimal and is a penetrator node in one of the following penetrator traces.

Trace P_M. Text message: $< +t >$ where t $\in \mathcal{T}$

If m is a node in Trace P_M, then $R_1 \sqsubset t$ which means R_1 originates from m. This contradicts Lemma 7.10 that R_1 originates uniquely from N_1.

Trace P_F. Flushing: $< -g >$

Since m is positive and there is no positive node in this trace, m cannot be a node in this trace.

Trace P_T. Tee: $< -g, +g, +g >$

The positive nodes re-transmit the term that is received by a previous node. $< P_T, 1 > \prec m$ and so m (the positive nodes) cannot be minimal. Hence m cannot be a node in this trace.

Trace P_C. Concatenation $< -g, -h, +gh >$

If m is a node in Trace P_C, $(R_1 \sqsubset gh) \wedge (V_1 \not\sqsubset gh) \wedge (V_2 \not\sqsubset gh)$.

If $R_1 \sqsubset gh$ then either $R_1 \sqsubset g$ or $R_1 \sqsubset h$. On the other hand, $V_1 \not\sqsubset gh$ means $V_1 \not\sqsubset g$ and $V_1 \not\sqsubset h$ and similarly, $V_2 \not\sqsubset gh$ means $V_2 \not\sqsubset g$ and $V_2 \not\sqsubset h$. Hence if $R_1 \sqsubset g$, $< P_E, 1 > \prec m$ or if $R_1 \sqsubset h$, $< P_E, 2 > \prec m$. Thus m cannot be a node in this trace.

Trace P_K. Key: $< +K_0 >$ where $K_0 \in \mathcal{K}_p$

This trace requires $R_1 \in \mathcal{K}_p$. This contradicts assumptions of a DPS strand that $R_1 \notin \mathcal{K}_p$.

Trace P_E. Encryption: $< -K_0, -h, +\{h\}_{K_0} >$

Suppose $m =< P_E, 3 >$, in which case

$R_1 \sqsubset \{h\}_{K_0}) \wedge (V_1 \not\sqsubset \{h\}_{K_0}) \wedge (V_2 \not\sqsubset \{h\}_{K_0}$

As $R_1 \sqsubset \{h\}_{K_0}$, then $R_1 \sqsubset h$ or $R_1 = K_0$. Moreover, $V_1 \not\sqsubset \{h\}_{K_0}$ means $V_1 \not\sqsubset h$ and $V_1 \neq K_0$; $V_2 \not\sqsubset \{h\}_{K_0}$ means $V_2 \not\sqsubset h$ and $V_2 \neq K_0$. Hence $< P_E, 2 > \prec m$ and contradicts the minimality of m.

Trace P_D. Decryption: $< -K_0^{-1}, -\{h\}_{K_0}, +h >$ where $K_0 \in \mathcal{K}_p$

Suppose $(R_1 \sqsubset h) \wedge (V_1 \not\sqsubset h) \wedge (V_2 \not\sqsubset h)$. Consequently, either $V_1 = \{h\}_{K_0}$ or $V_2 = \{h\}_{K_0}$.

$V_1 = \{h\}_{K_0}$**:** Free encryption requires $h = R_1$ and K_0 = $H(SDPass)_{AB}$. Hence there must be a node w which precedes m with $H(SDPass)_{AB}^{-1} \sqsubset$ term(w). In DPS, $H(SDPass)_{AB}^{-1}$ = $H(SDPass)_{AB}$. As $H(SDPass)_{AB}^{-1} \notin \mathcal{K}_p$ and together with Proposition3.3 in [17] , we can infer that $H(SDPass)_{AB}^{-1}$ originates from a regular node. We cannot find a regular node originating $H(SDPass)_{AB}^{-1}$. Thus node w does not exist in this case and $V_1 \neq \{h\}_{K_0}$.

$V_2 = \{h\}_{K_0}$**:** Free encryption requires $h = R_2 DDPass_{new}$ and K_0 = R_1. Hence there must be a node w which precedes m with $R_1^{-1} \sqsubset$ term(w). As $R_1^{-1} \notin \mathcal{K}_p$ and together with Proposition3.3 in [17] , we can infer that R_1^{-1} originates from a regular node.

For DPS, R_1^{-1} is the same as R_1. The regular node from which R_1 originates is N_1 and the message is V_1. As V_1 is encrypted with $H(SDPass)_{AB}$ and $H(SDPass)_{AB}^{-1} \notin \mathcal{K}_P$, penetrator cannot extract R_1. Thus $V_2 \neq \{h\}_{K_0}$.

As both $V_1 \neq \{h\}_{K_0}$ and $V_2 \neq \{h\}_{K_0}$ are not valid, m cannot be a node on the D penetrator trace.

Trace P_S. Separation: $< -gh, +g, +h >$

Assuming term(m) = g. A similar argument applies to term(m) = h.

As m is minimal in $\mathcal{S}$, $R_1 \sqsubset g$, $V_1 \not\sqsubset g$ and $V_2 \not\sqsubset g$. Consequently, either $V_1 \sqsubset gh$ implying $V_1 \sqsubset h$ or $V_2 \sqsubset gh$ implying $V_2 \sqsubset h$.

We show that no node can send gh to node m and hence m is not in Trace S.

Let $\mathcal{Z} = \{X \in C : (X \prec m) \wedge (gh \sqsubset term(m))\}$.
$\mathcal{Z}$ has at least one element $< P_S, 1 >$; and $\mathcal{Z}$ has a minimal element X by Lemma 2.7 in [17] ; and X must be positive (Lemma 2.8 in [17]).

No regular node have an encrypted term as a subterm of the form gh and so X cannot be a regular node. We are going to show that X cannot be a penetrator node either.

Trace P'_M. Text message: $< +t' >$ where t $\in \mathcal{T}$

It requires $R_1 \sqsubset t'$ and originates from $< P'_M, 1 >$, contradicting Lemma 7.10 which states that R_1 originates in N_1.

Trace P'_F. Flushing: $< -g' >$
There is no positive node, and so m cannot be on this trace.

Trace P'_T. Tee: $< -g', +g', +g' >$
If $X = < P'_T, 2 >$ or $X = < P'_T, 2 >$, it requires $gh \sqsubset < P'_T, 1 >$ which means $< P'_T, 1 > \prec X$ and contradicts the minimality of X.

Trace P'_K. Key: $< +K'_0 >$
It requires $R_1 \sqsubset K'_0$ and originates from $< P'_K, 1 >$, contradicting Lemma 7.10 which states that R_1 originates in N_1.

Trace P'_S. Separation: $< -g'h', +g', +h' >$
Suppose $X = < P'_S, 3 >$ (or $X = < P'_S, 2 >$), then $gh \sqsubset$ term($< P'_S, 1 >$) which implies X is not the minimal element in $\mathcal{Z}$.

Trace P'_E. Encryption: $< -K'_0, -h', +\{h'\}_{K'_0} >$
Suppose $X = < P'_E, 3 >$, then $gh \sqsubset$ term($< P'_E, 2 >$) which implies X is not the minimal element in $\mathcal{Z}$.

Trace P'_D. Decryption: $< -K'^{-1}_0, -\{h'\}_{K'_0}, +h' >$
Suppose $X = < P'_D, 3 >$, then $gh \sqsubset$ term($< P'_D, 2 >$) which implies X is not the minimal element in $\mathcal{Z}$.

Trace P'_C. Concatenation: $< -g', -h', +g'h' >$
Suppose $X = < P'_C, 3 >$, then $gh \sqsubset$ term($< P'_D, 3 >$). So P'_C has trace $< -g, -h, +gh >$. Hence term($< P'_C, 1 >$) = term(m). However, $< P'_C, 1 > \prec m$ which contradicts the minimality of m in $\mathcal{S}$.

None of the penetrator cases ($P'_M, P'_F, P'_T, P'_K, P'_S, P'_E, P'_D, and P'_C$) is true, and so X cannot be a node on the penetrator P_S.

As none of the penetrator cases ($P_M, P_F, P_T, P_K, P_S, P_E, P_D, and P_C$) is true, and so m cannot be a penetrator node. ■

As we have shown that m is not in any of the penetrator's Traces, we can conclude that m is a regular node. Because of its minimality, it is also positive.■

Lemma 7.16 can be re-worded as Definition 7.17:

Definition 7.23 *Fix some m that is $\preceq$-minimal in $\mathcal{S}$ = {n $\in$ C: $R_1 \sqsubset$ term(n) $\wedge$ $V_1 \not\sqsubset$ term(n) }, and is therefore regular and of positive sign.*

Now we have the fact that R_1 originates from a node N_1 in an initiator strand, and there exists a minimal node m in $\mathcal{S}$, we show that there exists another node N preceding m with $R_1 \sqsubset$ term(N).

Lemma 7.24 *A node N precedes m on the same strand t as m and term(N) = V_1.*

PROOF: It was shown that R_1 originates uniquely at N_1 (Lemma 7.10). As R_1 does not originates at m, there exists another node N preceding m with $R_1 \sqsubset$ term(N). Because m is minimal in $\mathcal{S}$, the only possible case is $V_1 \sqsubset$ term(N) so that N does not contradict the minimality of m. As no regular node has encrypted term as subterm, we can conclude that V_1 = term(N).■

Now we show m and N are located in a responder strand.

Lemma 7.25 *The regular strand t containing m and N is a responder strand and is contained in $\mathcal{C}$.*

PROOF: As term(N) = V_1 and precedes m on the same strand, we have the following cases:

Initiator Strand the node after N with term(N) = V_1 is negative and contradicts the fact that m is positive.

Responder Strand the node after N with term(N) = V_1 is positive and we can conclude that t is a responder strand and $N = n_1$; $m = n_2$.

The two nodes N and m are the first and second node of a responder strand, t is a responder strand.■

PROOF of Proposition 7.21

Proposition 7.21 follows from Lemma 7.22, Lemma 7.24 and Lemma 7.25. ■

We have shown that there is a responder strand in $\mathcal{C}$ to each initiator strand. The only remaining thing to show that the responder strand is unique.

Proposition 7.26 *If Σ is a DPS space, and R_2 is uniquely originating in Σ, then there is at most one strand $t \in$ Resp[$R_1,\ R_2,\ R_3,\ H(DDPass)_{AB},\ DDPass_{new}]_{AB}$ for any R_1.*

PROOF: By Lemma 7.13, R_2 originates at n_2 which is $< t, 2 >$. If we assume that R_2 uniquely originates in Σ as in assumption clause 4 of Proposition 7.21, then there can only be at most one such responder strand t. ■

Proposition 7.21 and Proposition refDPSProp4 together show that for a initiator strand in $\mathcal{C}$ there is an unique corresponding responder strand in $\mathcal{C}$. This is the Agreement Guarantee for the initiator.

7.4.3 Correctness Proof of DPS

In the proof for Propositions 7.15, 7.20, 7.21, and 7.26, the identity of the interlocutors are not explicitly specified in the message. As we use encrypted channels for communication, and it is common to use out-of-band signals to exchange the information about the

interlocutors before the encrypted communication starts, we can assume that the identity of the communicating parties are known before a DPS protocol run by Proposition 5.4. The encrypted channel authenticates the users of the channel and Man-in-the-Middle attack is not feasible in this case.

Now let us put the pieces of puzzle together:

- Proposition 5.4 established the identity of the interlocutors;
- Proposition 7.9 assures the secrecy of the term R_1 used in the message.
- We can prove the secrecy of the term R_2 used in the message.
- We can prove the secrecy of the term R_3 used in the message.
- We can prove the secrecy of the term $DDPass_{new}$ used in the message.
- We can prove the secrecy of the term $H(DDPass)$ used in the message.
- Proposition 7.15 and Proposition 7.20 provide the responder agreement property;
- Proposition 7.21 and Proposition 7.26 provide the initiator agreement property;

Hence we may say that DPS satisfied the requirements of correctness for a security protocol.

7.5 Chapter Summary

In this chapter, we have shown how Dynamic Password and its associated Key Exchange using Dynamic Password DPS can be applied to SNG. The terms used in messages of DPS remains secret to the adversaries and the identity of the interlocutors of the protocol run can be identified and confirmed. Thus the correctness of DPS when applied to SNG is assured.

Chapter 8

Application of SNG on Simulated Networks

In preceding chapters, we have developed a secure authentication protocol for SNG. In Chapter 7, we have presented a theoretic proof of the encrypted channel by the use of Strand Space theory. In this chapter, we present a case study of Service Network Graph on a simulated aggregate of networks. The study is about two scenarios, first when the user is located at his home network and second when a user is located at a visited network away from his home.

8.1 Simulation Platform Overview

Open source Objective Modular NEtwork Testbed in C++ (OMNeT++)[1] is an extensible, modular, component-based C++ simulation library and framework[2]. We simulate a case on OMNeT++ using a four-network SNG model.

Setting up a simulation in OMNeT++ is to define a network model using the NEwork Description (NED) language. The building blocks are modules. Each module represents a router, a switch, a server or a user and the modules are defined with NED. The actual work each module does is specified in the module class method *activity()* specified in the file module_name.cc. OMNeT++ system configuration file and configuration files for SNG simulation are shown in a hierarchical fashion in Figure 8.1.

[1] More details can be found in http://www.omnest.com/

[2] OMNEST is the commercial version of the OMNeT++ simulation environment. More details can be found in http://www.omnest.com/

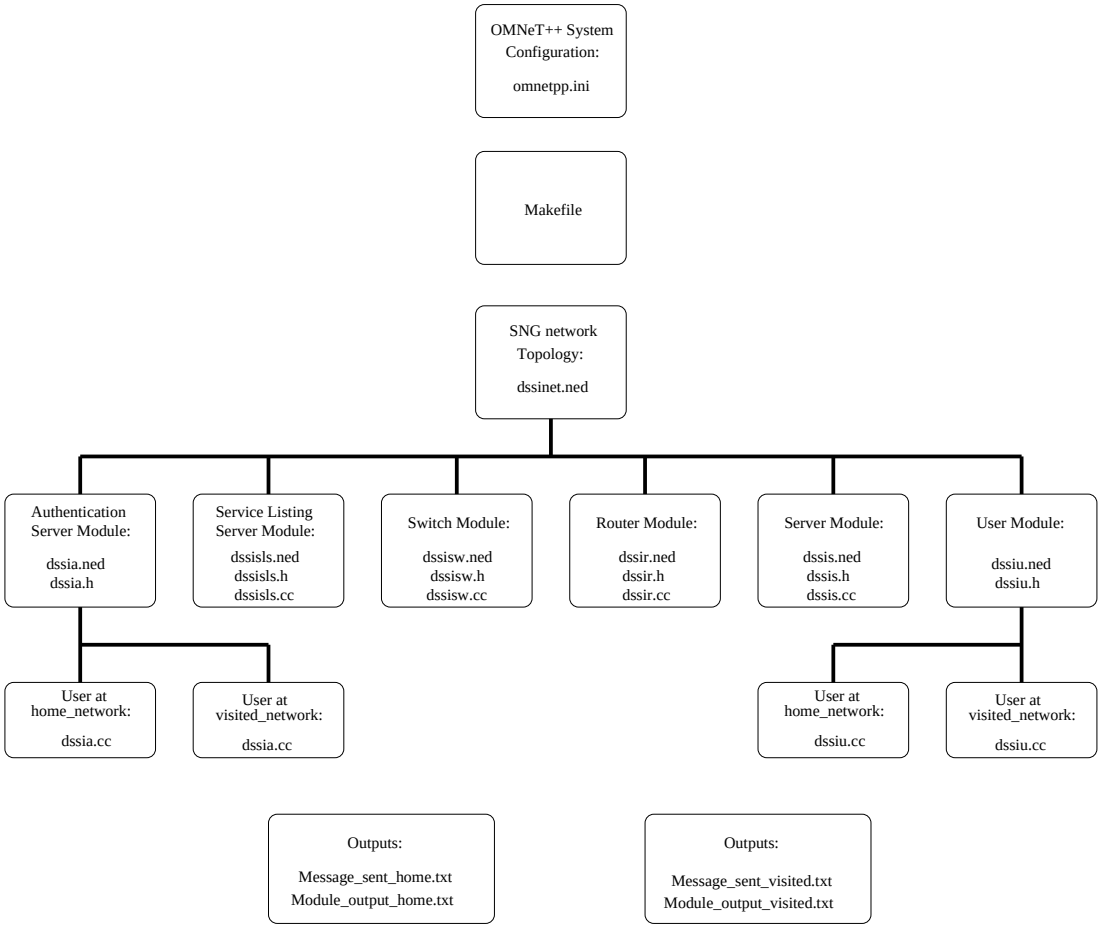


Figure 8.1: OMNeT++ configuration files for SNG simulation.

8.1.1 Configuring OMNeT++ for the SNG Model

A typical autonomous network consists of authentication server, service server, service listing server, and users. Routers and switches are used to interconnect the autonomous networks to form an SNG. We use a four autonomous network SNG for our simulation. The OMNeT++ configuration file and the corresponding Makefile used to generate the executable file are listed in Table 8.1.

	Configuration file	Listed in Appendix
OMNeT++ System	omnetpp.ini	C.1
Makefile	Makefile	C.2

Table 8.1: OMNeT++ configuration files.

8.1.2 Specifying SNG Model for OMNet++

The SNG network model we used consists of four networks, N_1, N_2, N_3, and N_4 as shown in Figure 8.2 and is defined by dssinet.ned (listed in Appendix C.3).

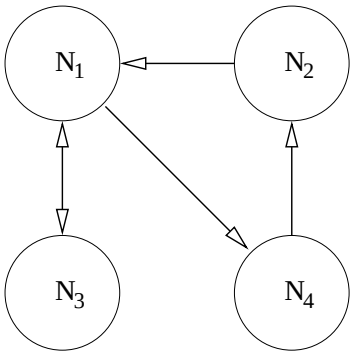


Figure 8.2: OMNeT++ SNG network model for user at home network.

The four autonomous networks are linked together in an SNG context as indicated by the single and double arrows for the one-way and two-way authentication delegation. Note that the link between N_1 and N_2 are two-way while the others are one-way. Home network for the user is N_1. When U is located at N_1, he is at the home network. When U is at N_3, he is at a visited network.

To access a service, a user U initiates a service query to SLS of the current network he is located at. The SLS returns a service list available to the U. U can then select a service and make a service request. If he is at the Home Network, then the AS authenticates him; gets the service information from the server, perhaps via other ASs in the SNG; and returns the service information to him. U can then access the service directly without involving any AS for message relay.

If U is at a visited network for the first time, he has to identify himself to the AS of the current network which gets a service list from the SLS of U's Home Network. The newly acquired service list, which would be added to the service available to the U from the current network, is sent to U. These extra steps are represented by OMNeT++ as an extra initial message in indigo color. U then makes a normal service query to the SLS of

the current network. The AS of the current network sends the service request to U's Home Network for authentication. The Home Network AS gets the service information from the appropriate server and returns the service information to U via the AS of the visited network.

Module	**Module files**	**Listed in Appendix**
Server module	dssis.ned	C.4.1
	dssis.h	C.4.2
	dssis.cc	C.4.3
SLS server module	dssisls.ned	C.5.1
	dssisls.h	C.5.2
	dssisls.cc	C.5.3
Switch module	dssisw.ned	C.6.1
	dssisw.c	C.6.2
	dssisw.cc	C.6.3
Router module	dssir.ned	C.7.1
	dssir.h	C.7.2
	dssir.cc	C.7.3
User module	dssiu.ned	C.8.1
	dssiu.h	C.8.2
	dssiu.cc (when user at home network)	C.8.3
	dssiu.cc (when user at visited network)	C.8.4
AS module	dssia.ned	C.9.1
	dssia.h	C.9.2
	dssia.cc (when user at home network)	C.9.3
	dssia.cc (when user at visited network)	C.9.4

Table 8.2: Module files for the SNG used in OMNeT++.

8.1.3 Component Modules of the SNG Model

We specify the configuration and functionality of each OMNeT++ module using three files: module.ned, module.h and module.cc. Module.ned is responsible for the module configuration while module.h and module.cc provides the functionality of the module in C++

language. Each module has many variables and a number of functions. Among the functions, activity() is very important which outlines what actions the module would take.

All instances of the same module share the same set of module files. For example, all routers share the same set of router module files dssir.ned, dssir.h and dssir.cc. The modules and the corresponding module files are listed in Table 8.2.

Note that when we simulate the case when user is at his home network, we use the "at home network" version of dssiu.cc and use the "at visited network" version when the user is not at his home network.

8.1.4 OMNet++ Outputs

OMNeT++ records all outputs of a model run in output vectors. The output lists information we added to messages in the form of parameters including the message source and destination. A summary of the output is listed in Table 8.3.

	Output file	**Listed in Appendix**
User at home network	Message_sent_home.txt	C.10.1
	Module_output_home.txt	C.10.2
User at visited network	Message_sent_foreign.txt	C.10.3
	Module_output_foreign.txt	C.10.4

Table 8.3: OMNet++ output files.

Note that the "at home network" version of output corresponds to the case when user is at his home network, and "at visited network" version of output is for the case when user is away from his home network. We used different versions of dssiu.cc and dssia.cc for the two cases.

8.2 Service Access

To understand how the service messages are routed in OMNeT++, we shall first explain the parameters we added to each message and how they are used for routing a message to its destination.

The messages in the SNG model have eight parameters to which we can assign values. The parameters are listed in Table 8.4. They are appended to a message as C++ strings.

	parameter	Description
1	address	Host address of the device
2	server	Address of the service providing server
3	src	Current address of an user
4	dest	Destination address of the message
5	start	Source addresses of AS or $servers$ for each segment of the service access path
6	stop	Destination addresses of AS or $servers$ for each segment of the service access path
7	Message Label	Label indicating the use of the message
8	Message Kind	Color code for various message type

Table 8.4: Addresses of devices used in the Simulation.

8.2.1 Address

The parameter $address$ is used to hold the host address of the device.

Four networks with network number 100, 200, 300 and 400 are used. Inside each network, there are six hosts. They are assigned a host number according to their roles. The numbers 1 to 6 are assigned to SLS, AS, S, U, switch and router respectively. The resulting host address for a device follows pretty much the same as the TCP/IP addressing scheme of (network address + host address). Take an SLS as an example. SLS in network 200 has a host number 1 and network number 200, and the resulting host address is 201 (200 + 1).

Table 8.5 shows a summary of the network and host addresses used.

Network	Host Address					
	SLS	AS	S	U	$Switch$	$Router$
100	101	102	103	104	105	106
200	201	202	203	204	205	206
300	301	302	303	304	305	306
400	401	402	403	404	405	406

Table 8.5: Addresses of devices used in the Simulation.

8.2.2 Server, Src, Dest

When a user U_1 roams to another network, it has to acquire an address in the visited network in order to talk to the AS of the visited network. The acquired address would be different from his home address. Parameter src is the current address of U_1, which is 104 (host address in home network) for the $User\ at\ home\ network$ case; and is 304 (acquired in visited network) for $User\ at\ visited\ network$ case.

Parameters $dest$ and $server$ are the destination address of the message and address of the service providing server respectively. Table 8.6 shows the possible destination addresses for messages in the simulation.

Address	Use
101	A service query from U_1, $User\ at\ home\ network$ case.
102	A service request from U_1, $User\ at\ home\ network$ case.
301	A service query from U_1, $User\ at\ visited\ network$ case.
302	A service request from U_1, $User\ at\ visited\ network$ case.
103	A service access from U_1 to S_1.
203	A service access from U_1 to S_2.
403	A service access from U_1 to S_4.

Table 8.6: Destination addresses of a message in the simulation.

8.2.3 Start, Stop

When U initiate a service query, the message may pass a number of AS before reaching the application server for service information. Parameters $start$ and $stop$ are the source and destination addresses of AS or $servers$ for each segment of the path when requesting and returning service information.

For instance, the service query for S_1 for the $User\ at\ visited\ network$ is listed in Table 8.7. As shown in Table 8.7, $server\ src$ and $dest$ remain the same while $start$ and $stop$ are changed at AS_3, AS_1, AS_4 and S_4. Note that switches and routers do not change any of the variables $start$, $stop$, src or $dest$.

From	Server	start	stop	src	dest	Comment
U_1	103	304	302	304	302	From U_1 to AS_3
SW_3	103	304	302	304	302	
AS_3	103	302	102	304	302	From AS_3 to AS_1
SW_3	103	302	102	304	302	
R_3	103	302	102	304	302	
R_1	103	302	102	304	302	
SW_1	103	302	102	304	302	
AS_1	103	102	103	304	302	From AS_1 to S_1
SW_1	103	102	103	304	302	
S_1	103	103	102	304	302	From S_1 to AS_1
SW_1	103	103	102	304	302	
AS_1	103	102	302	304	302	From AS_1 to AS_3
SW_1	103	102	302	304	302	
R_1	103	102	302	304	302	
R_3	103	102	302	304	302	
SW_3	103	102	302	304	302	
AS_3	103	302	304	304	302	From AS_3 to U_1
SW_3	103	302	304	304	302	

Table 8.7: Example of message variable values from S_4 to U.

8.2.4 Message Label

We use the $Message\ Label$ to indicate the use of the message. We simply include the source and destination address in the $Message\ Label$. Table 8.8 shows all possible values of $Message\ Label$ for the $User\ at\ visited\ network$.

8.2.5 Message Kind

OMNeT++ associates different color to different Message Kind to differentiate various type of messages. With the color code, message flow can easily identified. Table 8.9 lists all the message kinds and the associated message colors used in our simulations.

Label	Use	User at
104to301	A service query from U_1	Home Network
104to302	A service request from U_1	Home Network
104to103	A service access from U_1 to S_1	Home Network
104to203	A service access from U_1 to S_2	Home Network
104to403	A service access from U_1 to S_4	Home Network
304to301	A service query from U_1	visited network
304to302	A service request from U_1	visited network
304to103	A service access from U_1 to S_1	visited network
304to203	A service access from U_1 to S_2	visited network
304to403	A service access from U_1 to S_4	visited network

Table 8.8: List of message labels.

Number	Color	Use in our simulation
0	Red	Not used
1	Green	Service query
2	Blue	Service request
3	White	Service access
4	Yellow	Not used
5	Indigo	Hello message
6	Magenta	Not used
7	Black	Not used

Table 8.9: List of message kinds and color.

8.3 Routing Operation

The SNG model is defined by dssinet.ned. Similar to the routes in a real router, the connections between networks and hosts are defined by the $connection$ section in dssinet.ned. For instance,

```
SW2.inputs[3] <-- U2.outputs[0];
```

simply means the outputs[0] of the $U2$ module is connected to inputs[3] of the $SW2$ module and the arrow shows the direction of the message flow. Note that $nocheck$ modifier makes OMNeT++ to assume all input and output gates are connected. A few lines from the $connection$ section of dssinet.ned are shown below:

```
connections nocheck:
    U2.inputs[0] <-- SW2.outputs[3];
    SW2.inputs[1] <-- AS2.outputs[0];
    SW2.inputs[3] <-- U2.outputs[0];
    ...
    R1.inputs[1] <-- R2.outputs[0] display "o=red,2";
    R2.inputs[0] <-- R1.outputs[1] display "o=grey,0";
    ...
```

Routing or Message delivery in OMNeT++ is handled by the $activity()$ method of each module.

8.3.1 User Module

To be precise, the $activity()$ method in an user module does not route messages. It is responsible for message generation. Possible messages generated are listed in Table 8.10. The generated message is always sent to the connecting switch as specified in dssinet.ned.

One Time Only Message

Message Label	Use
304to302	Hello message to AS_3.

Repeating Messages

Message Label	Use
104to101	A service query to SLS_1.
104to102	A service request to AS_1.
104to103	A service access to S_1 in network 100.
104to203	A service access to S_2 in network 200.
104to403	A service access to S_3 in network 400.
304to301	A service query to SLS_3.
304to302	A service request to AS_3.
304to103	A service access to S_1 in network 100.
304to203	A service access to S_2 in network 200.
304to403	A service access to S_4 in network 400.

Table 8.10: List of messages generated by user module U_1.

8.3.2 Server Module and Service Listing Server Module

The $activity()$ method of the server and service listing server modules are simple. As they must be destination of a received message, all they have to do is return the message back to the sender which is the *stop* by swapping the $start$ and *stop* parameter of the incoming message as shown in Table 8.11. The message is sent out via the appropriate output gates with information from the $connection$ section of dssinet.ned.

Code	Action
msg = receive();	receive a message
start = msg->par("start");	get the parameter $start$
stop = msg->par("stop");	get the parameter $stop$
msg->par("start") = stop; msg->par("stop") = start;	Swap the parameters $start$ and $stop$
...	
send(msg, "outputs", 0);	send it to $outputs[0]$

Table 8.11: Code used by routing messages in server and service listing server modules.

8.3.3 Switch Module

Routing for switches is also straight forward. A message is either sent to the connected router or the connected host. If the message is heading to the connected host, the addresses of the switch and *stop* parameter would have the same network component. Otherwise, the message is sent from the connected host to another device in SNG. A segment of the code is shown below:

```
//Receive a message
msg = receive();
// Get "stop" parameter
stop = msg->par("stop");
// Extract the network part of "stop"
stopNet = stop / 100;
// Extract the host part of "stop"
stopHost = stop % 100;
    ...
// To connected host?
if (stopNet != (address / 100))
```

```
    // NO: send to router
    { send( msg, "to_router" );}
else
    // YES: send to host
    { send(msg, "outputs", stopHost-1); }
```

The message would be sent out via the corresponding output gates with information from the *connection* section of dssinet.ned.

8.3.4 Router Module

Routers handle messages in a similar fashion that switches do. If the message is heading to the connected host, the addresses of the router and *stop* parameter would have the same network component. Otherwise, the message is sent from the connected host to another device in SNG. A segment of the code is shown below:

```
// receive a message
msg = receive();
// get "stop" parameter
stop = msg->par("stop");
// extract the network part of "stop"
stopNet = stop / 100;
// extract the host part of "stop"
stopHost = stop % 100;
    ...
// to connected host?
if (stopNet != (address / 100))
    // NO: send to another router
    {    send( msg, "outputs", stopNet-1 );}
else
    // YES: send to connected switch
    { send(msg, "to_switch); }
```

The message would be sent out via the corresponding output gates with information from the *connection* section of dssinet.ned. Note that the index for outputs[] is determined by the network component of the *stop* parameter.

8.3.5 Authentication Server Module

As a message changes its intermediate destination at AS and reverses direction at SLS and S, the *start* and *stop* variables would also be changed at these devices. The $activity()$

method in SLS and S simply swap the values for the $start$ and $stop$ variables. The $activity()$ method is more elaborated when it comes to AS.

The $activity()$ method in AS is responsible for keeping track of the service authentication path and the service path. When a message arrives at an AS, the message would be passed on to the next hop according to the $SPath$ of the message. Message variables $start$ keeps record of where it comes from and $stop$ keeps track of where it should go.

In the SNG model, authentication delegations are specified in the $connection$ section of dssinet.ned. Table 8.12 gives the list of the SNG model connections in a generic format.

Authentication Delegation		
To Network		**From Network**
Network 1	→	Network 3
Network 2	→	Network 1
Network 4	→	Network 2
Network 1	→	Network 4

Table 8.12: List Network linkage for the simulation cases.

Table 8.12 does not tell us explicitly what is the next network the message should go. For example, going to Network 2 from Network 1 requires the message to travel from Network 1 to Network 4 and then it reaches Network 2. Instead of working out where to send the message every time from the linkage relationship, we optimize the routing process with hard coded routes listed in Table 8.13. So if a message at Network 1 is heading to Network 2 in a forward direction, the next stop should be Network 4 and the $stop$ variable should hold the address of the AS in Network 4 which is 402. For a message heading to Network 2 at Network 1 in a backward direction would have Network 4 as the next network and the $stop$ variable would be assigned a value of 402.

When the message is traveling in a forward direction,

To Network = $server$ / 100
Current Network = $stop$ /100

Similarly, a backward moving message has

To Network = src / 100
Current Network = $stop$ /100

The routing algorithm for AS module is listed in Algorithm 3.

Forward			Backward		
To Network	**Current Network**	**Next Network**	**To Network**	**Current Network**	**Next Network**
Network 1	Network 2	Network 1	**Network 1**	**Network 2**	**Network 4**
Network 1	Network 3	Network 1	Network 1	Network 3	Network 1
Network 1	Network 4	Network 2	Network 1	Network 4	Network 1
Network 2	**Network 1**	**Network 4**	Network 2	Network 1	Network 2
Network 2	Network 3	Network 1	Network 2	Network 3	Network 1
Network 2	Network 4	Network 2	Network 2	Network 4	Network 1
Network 3	Network 1	Network 3	Network 3	Network 1	Network 3
Network 3	Network 2	Network 1	Network 3	Network 2	Network 4
Network 3	Network 4	Network 2	Network 3	Network 4	Network 1
Network 4	Network 1	Network 4	Network 4	Network 1	Network 2
Network 4	Network 2	Network 1	Network 4	Network 2	Network 4
Network 4	Network 3	Network 1	Network 4	Network 3	Network 1

Table 8.13: List of SNG routes.

Algorithm 3: Routing Algorithm for AS module

```
If AS is in the same network as the server
    If message comes from server
        If AS is in the same network as the user, send to user
        else send to another AS
    else send to server
else
    if AS is in the same network as the user
        If the message for the user send to user
    else send to another AS
end
```

A graphical representation of the logic is also shown in Figure 8.3.

8.4 Simulation of the SNG Model

To test the feasibility and practicability of SNG, we test the SNG model under two different scenarios: (1) when user is at home network and (2) when user is at visited network.

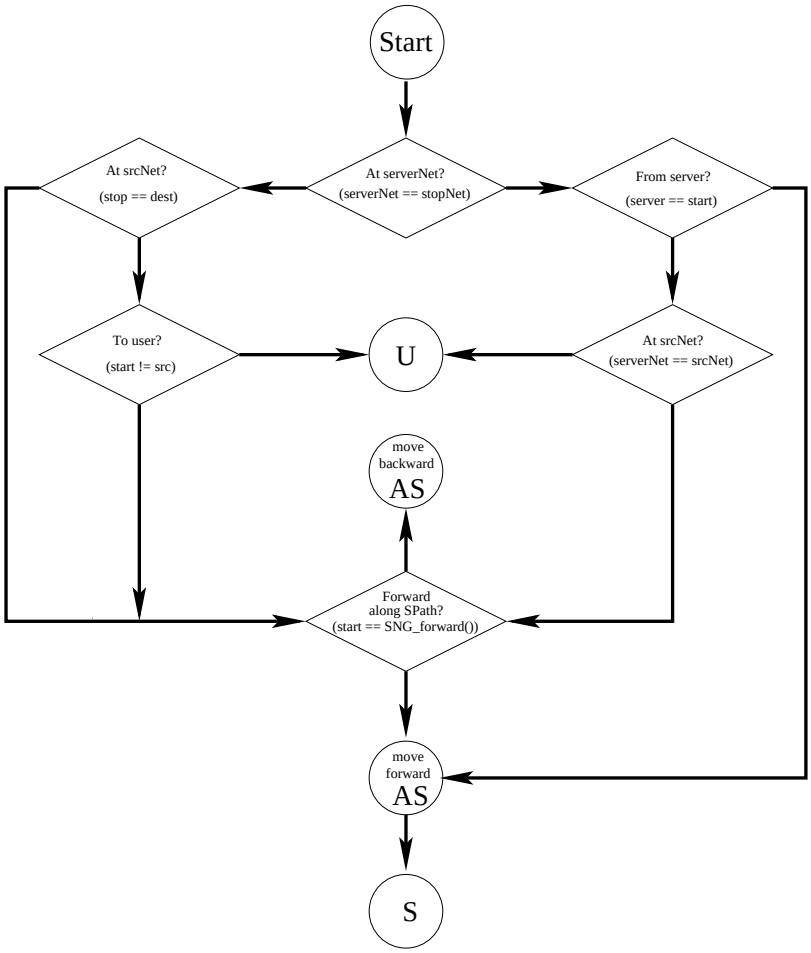


Figure 8.3: Routing logic for the simulation.

8.4.1 Scenario 1: User at Home Network

The topology used is shown in Figure 8.4. Note that user U_1 is in Network 1, his home network.

U_1 starts with a service query (in green) to SLS_1. Then he decided to send a service request (in blue) to AS_1 asking for a service from S_1. The service request is sent to AS_1 for authentication. In real cases, a few more message exchange between AS_1 and U_1 may be involved. For simplicity, we assume that this message alone is enough to authenticate U_1.

After authenticating U_1, AS_1 asks for the service information from S_1 by passing the service request to S_1. The service information is relayed back to U_1 via AS_1. The actual service access is represented by three direct message exchanges (in white) between U_1 and S_1.

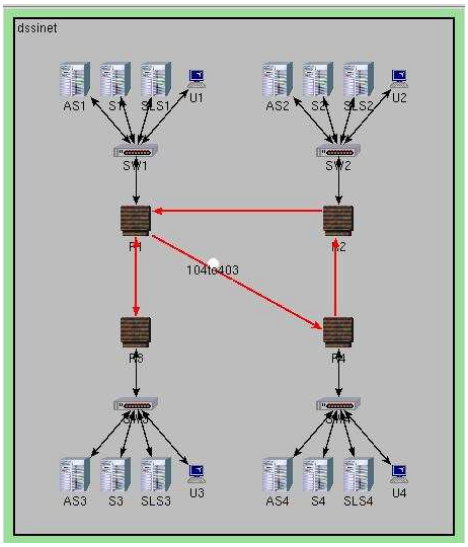


Figure 8.4: Simulation Topology for user at home network.

The second service request from U_1 is directed to S_4. The process is similar to the first request. U_1 starts with a service query (in green) to SLS_1 and then a service request (in blue) to AS_1. AS_1 then relays the request to AS_4 which eventually gets the service information from S_4. The return path for the service information is from S_4 to AS_4, to AS_1 and finally to U_1. The access for service (in white) is direct message exchanges between U_1 and S_4.

The third service request from U_1 is targeted at S_2. The whole process is similar to the second requests except the path for service information is longer by one hop - AS_2.

8.4.2 Scenario 2: User at Visited Network

The topology used is shown in Figure 8.5. Note that user U_1 is now located at Network 3, a visited network, instead of Network 1, his home network.

Initially, a Hello message (in indigo) is sent from U_1 to AS_3. Because the home network for U_1 is Network 1, AS_3 gets the service list from SLS_1 via AS_1 and passes it on to U_1. AS_3 also informs SLS_3 of the service list from SLS_1 so that U_1 does not have to repeat this initial step. U_1 now makes service query (in green) to SLS_3 for services from SLS_1 and SLS_3 available to him.

After the Hello session, the process is similar to the case when U is at his Home Network. U_1 initiates a service query to SLS_3 and decides to use service provided by S_1. A service request (in blue) is sent from U_1 to AS_3 which passes on the service request to AS_1 for authentication. In real cases, a few more message exchange between AS_1 and U_1 via AS_3 may be involved. For simplicity, we assume that this message alone is enough to

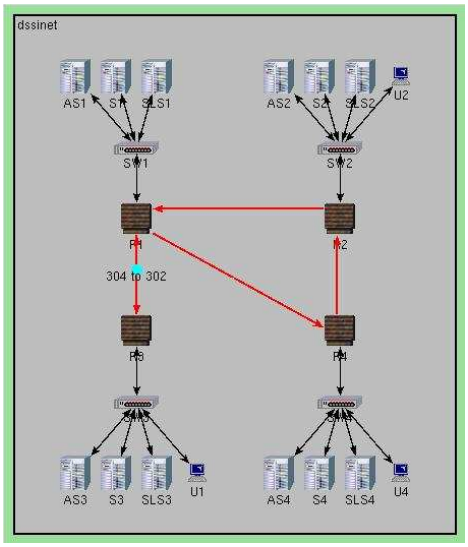


Figure 8.5: Simulation Topology for user at a visited network.

authenticate U_1.

After authenticating U_1, AS_1 asks the service information from S_1 by passing the service request to S_1. The service information is relayed back to U_1 via AS_3. The actual service is represented by three direct message (in white) exchanges between U_1 and S_1.

Access services provide by S_4 and S_2 are similar to the case illustrated above except AS_4 and AS_2 are involved in getting the service information.

8.5 Chapter Summary

In this chapter, we have presented a practical application of SNG using simulation technology. Two scenarios are shown. The first one is when a user is at his home network and the second scenarios is when the user is at visited network. Successful application of an SNG model to a simulated environment leads to the implement of a four-network SNG using C++ detailed in next Chapter.

Chapter 9

Practicality of SNG on Aggregated Networks

In Chapter 1, we stated that the objective of our study is design secure service sharing protocol for aggregated autonomous networks. We have so far theoretically proven that SNG and $DPass$ can be used to establish the secure channel and the case study of simulation has demonstrated its feasibility in Chapter 8. In this chapter, we verified that SNG is a practical solution to an aggregate of autonomous networks by implementing the basic features of SNG on an aggregate of four autonomous networks.

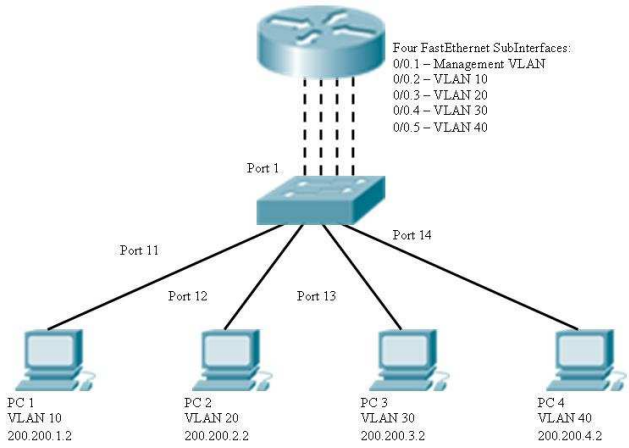
9.1 The Autonomous Network Aggregate

We implemented the basic features of SNG with the following assumptions:

1. Four networks are interconnected.
2. All hosts in a single network are implemented using a single machine with a single IP address.
3. Servers of a particular network share the same IP address but listen to different ports.
4. Dynamic Password is used as the authentication protocol.

We used an isolated network system of four networks interconnected by Cisco routers and switches. To reduce the resources required, AS, SLS, U, and S of each network are all implemented in a single machine with a single IP address. Virtual Local Area Network ($VLAN$) technology [73] is employed to interconnect the networks. The use of $VLAN$ reduces the networking devices required to a pair of router and switch only. "Router on a stick" configuration is used to connect the switch hosting all the $VLAN$s to the router [72]. The actual hardware used in shown in Figure 9.1(a).

(a) Hardware used in the SNG.



(b) Logical network diagram for the SNG.

Figure 9.1: SNG topology used

We labeled the computers as PC1 to PC4 as shown in Figure 9.1(a) and 9.1(b). The two devices at the top of the stack are the router and switch.

The IP addresses of the personal computers are listed in Table 9.1.

PC	IP Address
1	200.200.1.2 / 24
2	200.200.2.2 / 24
3	200.200.3.2 / 24
4	200.200.4.2 / 24

Table 9.1: IP addresses of PCs used in the SNG.

Table 9.2 lists some of the network and $VLAN$ information of the SNG topology for easy reference.

The configuration files for the router and the switch used are shown in Appendix D.10.

$VLAN$ Number	Network Address / Mask	Switch Port	Router SubInterface	Router IP Address
10	200.200.1.0 / 24	0/11	0/0.2	200.200.1.1
20	200.200.2.0 / 24	0/12	0/0.3	200.200.2.1
30	200.200.3.0 / 24	0/13	0/0.4	200.200.3.1
40	200.200.4.0 / 24	0/14	0/0.5	200.200.4.1

Table 9.2: Summary of network and $VLAN$ data.

9.2 SNG Topology and Data Files

The topology of the simple SNG used is shown in Figure 9.2. Note that the networks are physically connected as a mesh and all hosts can reach all other hosts directly. This mimics the real Internet for SNG implementation.

To store the data for the SNG, we used the data files shown in Table 9.3. The content of the data files are listed in Appendix D.9.1.

File Name	Used by	Format
PCxMap.data	ASx to forward authentication data.	{destination, next hop}
PCxSP.data	SLSx for $SPath$ information.	{serviceID, port, serverName, $SPath$, cost, restriction}
dp.data	List of possible $DDPass$.	
port.data	AS for the port which a ServiceGroup server listens to.	{server type, port}
userDP.data	AS for user authentication information.	{userName, userSDPass, userDDPass}
userHDP.data	AS for user authentication information.	{userName, userHSDPass, userHDDPass}

Table 9.3: Data files used in the implementation.

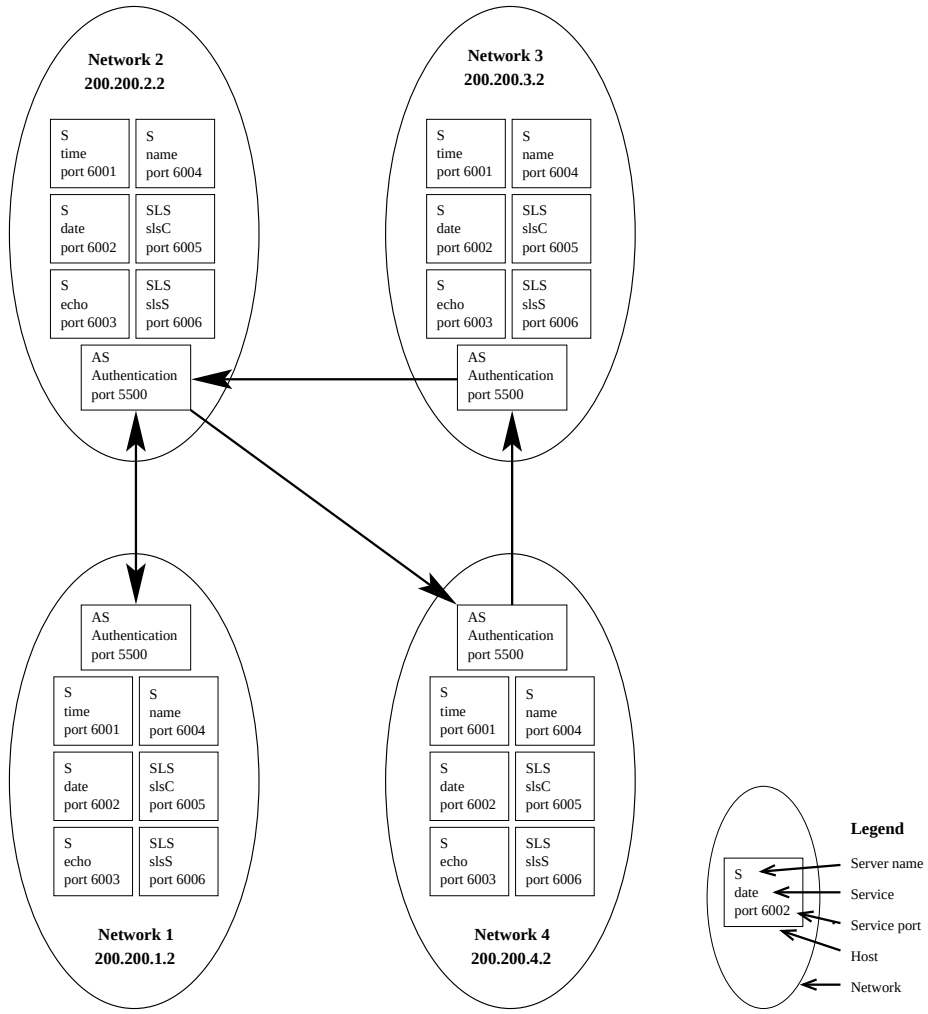


Figure 9.2: Logical topology of SNG showing the authentication delegations.

9.3 Service Request Cases

There are four general types of service request, from home network to foreign network / remote network or vice versa, as listed in Table 9.4. In the first case, as shown in Figure 9.3 the request does not leave the Home Network. Home AS just gets the service information from the Home Network server and returns it to the user.

Case	User at	Service
1	Home Network	Home network
2	Home Network	Foreign networks
3	Foreign network	Home and foreign networks
4	Remote network	Home and foreign networks

Table 9.4: Types of Service Request.

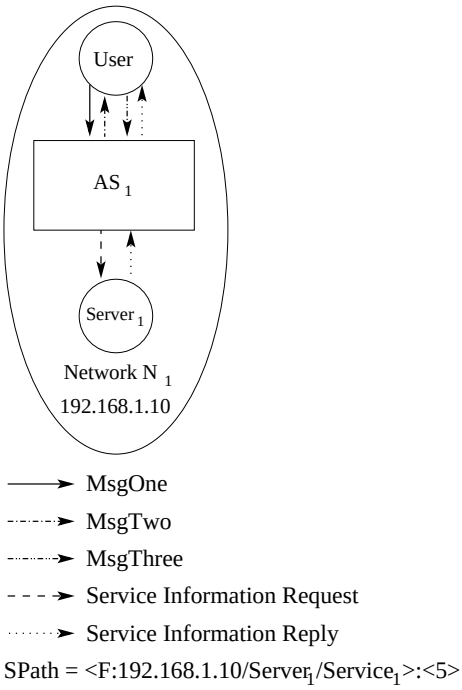


Figure 9.3: User accessing Home Network services at Home Network.

In the other cases the Homes AS has to pass the request to other AS. Each AS has to make use of $SPath$. Figure 9.4 shows tha case of user accessing Foreign Network services at Home Network and Figure 9.5 shows tha case of user accessing Network services at a Foreign Network.

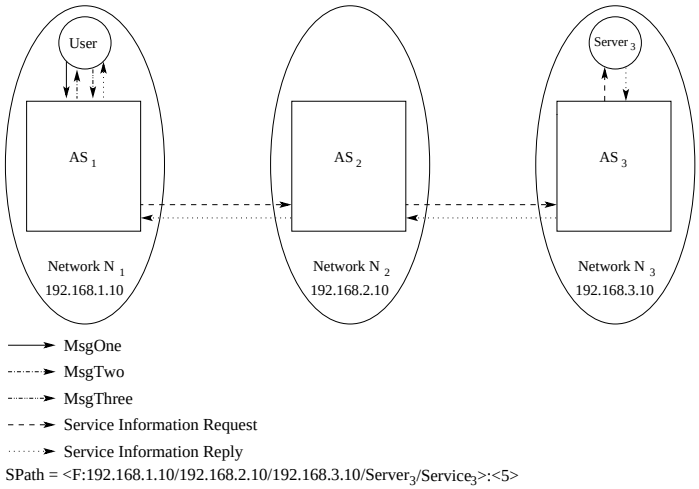


Figure 9.4: User accessing Foreign Network services at Home Network.

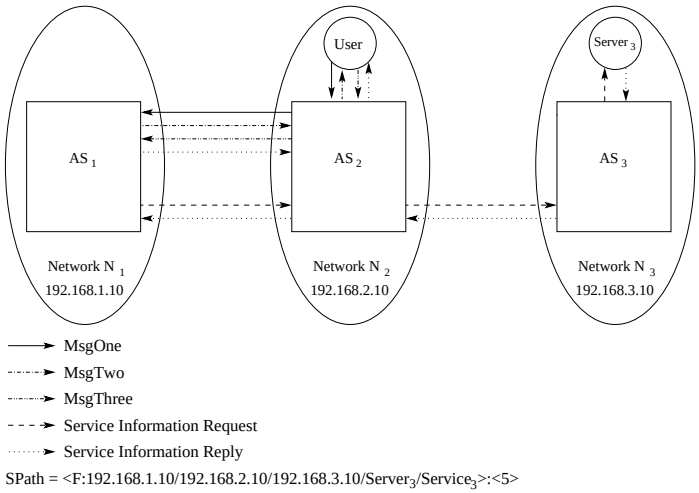


Figure 9.5: User accessing Network services at a Foreign Network.

For the last case, as shown in Figure 9.6, user is located in a remote network. The $SPath$ does not hold any path information to Home Network from the current (remote) network. The current AS has to look up PCxMap.data which is generated or updated whenever $SPath$s are acquired and optimized by an AS. For this version of implementation, all PCxMap.data files are manually created and maintained. Automatic creation of PCxMap.data by AS is left for future work. Format of PCxMap.data is simple. For instance, assuming the current network is N_2 and the Home Network is N_3, the line

```
200.200.3.2      200.200.4.2
```

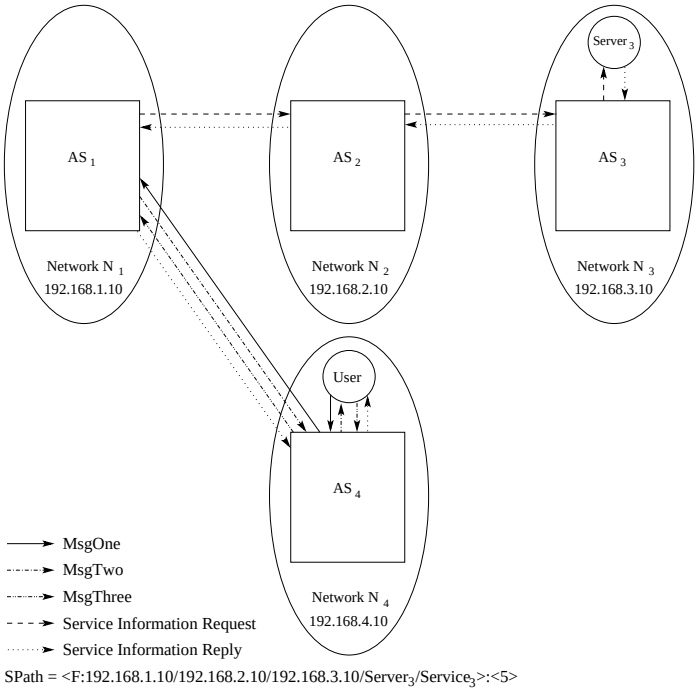


Figure 9.6: User accessing Network services at a Remote Network.

in PC2Map.data tells the current AS (AS_2) that in order to reach the final destination in N_3, send the authentication information to N_4. Note that an entry in PC2Map.data means that the networks are enrolled in the SNG. So service request from an alien network user is not processed.

9.4 SNG Users and Servers

The major entities in an SNG are U, S, SLS, and AS. The implementation of U and AS may make use of the server-client paradigm, in which U is the client and AS is the server. Similarly, U and S are also implemented as server-client model.

When a service request reaches AS of the service providing network ($SNet$), the request would be passed to a $Service\ Group\ Server$ which listens to a fixed port. Passing the service request to $Service\ Group\ Server$ instead of passing the request directly to the service providing server is to add another layer of security. The $Service\ Group\ Server$ invokes an instance of an application, the $Service\ Server$, to listen to a randomly generated port. The port number is returned to AS as part of the service information. So only authenticated users requesting a service can have the proper service information on top of the session key for encrypted communication.

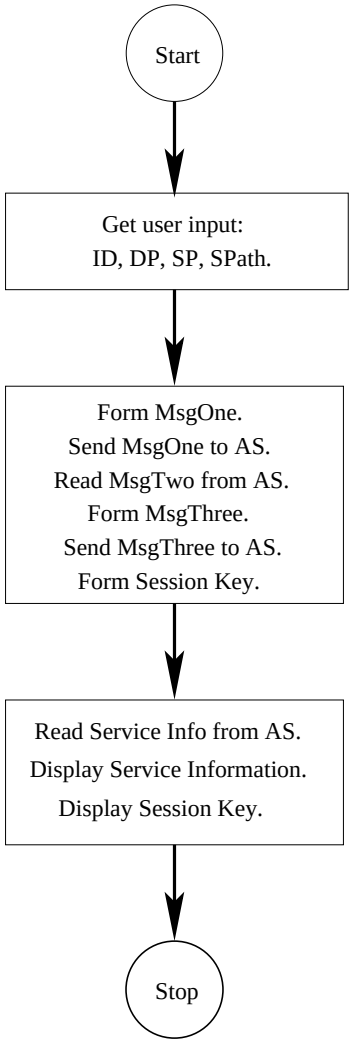


Figure 9.7: Logic diagram of the client program.

A user is implemented as a client program. When started, the client program asks authentication information and requested $SPath$ from the user and send the authentication information to AS using port 5500. After completing the authentication process, the client program displays service information returned from the AS including service port and session key to be used. A logic diagram of the client application is shown in Figure 9.7. Listing of asClient.cpp can be found in Appendix D.2.

9.4.1 SNG Authentication Server

Unlike the user, an AS is implemented as a server. Primarily, it authenticates a user and provides service information to the user. During the authentication process, a session key is generated which would be used for encrypting the traffic for service access. A random port is assigned for the service access session. The randomness of the port adds security and accountability to the service access.

The algorithm used by AS is listed in Algorithm 4 and a more detailed diagram is shown in Figure 9.8. AS server is implemented as asServer.cpp. Listing of asServer.cpp can be found in Appendix D.1.

Algorithm 4: Algorithm for AS servers

```
Get authentication information from user
If user is a local user
{
    Authenticate user
    If service is a local service
    {
        Get service information
        Send service information to user
    }
    else
    {
        Send service request to next AS in the SPAth
    }
}
else
{
    If service is a local service
    {
        Get service information
        Send service information to user
    }
    else
    {
        Send service request to next AS in the SPAth
    }
}
end
```

Note that we use a $service\ code$ to determine if the message is heading towards the user or away from the user during a service request.

AS server is implemented as asServer.cpp. Listing of asServer.cpp can be found in Appendix D.1.

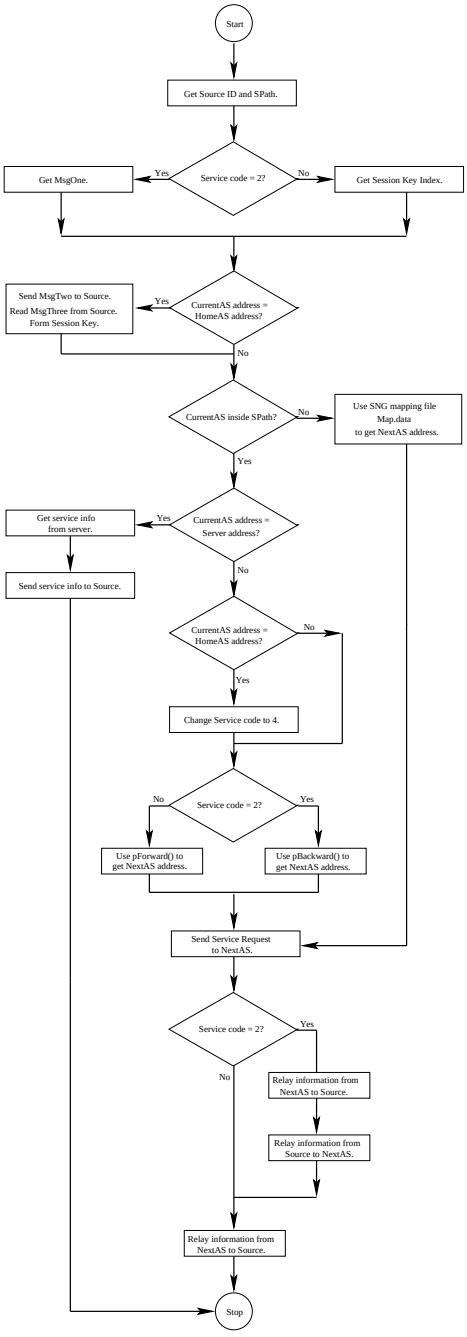


Figure 9.8: Diagram of algorithm used by the asServer program.

9.4.2 SNG Application Servers

After authentication the AS contacts the $ServiceGroup\ server$ for service information. The $ServiceGroup\ server$ starts an instance of the $Service\ server$ listening to a randomly generated port. The main piece of service information returned to the AS is the randomly generated port number which the $Service\ server$ listens to. The user can only access a service successfully when he has the service port and session key.

To provide service, an application server uses the standard Unix system call $execl()$ to start an instance of an application and so off-shelf applications can easily be hooked up as a ready-to-serve service.

The logic diagram of $ServiceGroup\ server$ is shown in Figure 9.9 and logic diagram of $Service\ server$ is shown in Figure 9.10.

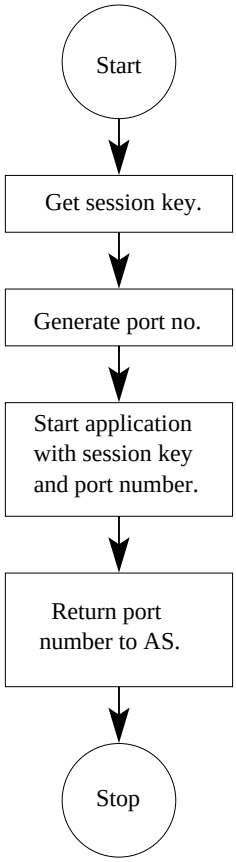


Figure 9.9: Diagram of algorithm used by an appGroup program.

Listings of $ServiceGroup\ servers$ and $Service\ servers$ are given in Table 9.5.

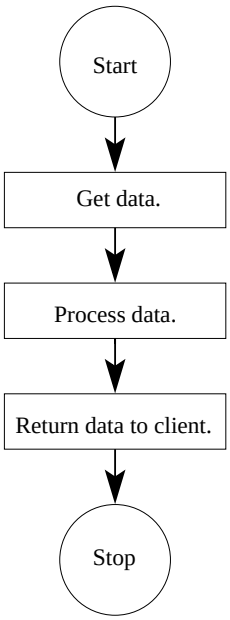


Figure 9.10: Diagram of algorithm used by an application program.

Service	*ServiceGroup server*	*Service server*
Date	Appendix D.3.1	Appendix D.3.2
echo	Appendix D.4.1	Appendix D.4.2
name	Appendix D.5.1	Appendix D.5.2
time	Appendix D.6.1	Appendix D.6.2
slsS	Appendix D.7.1	Appendix D.7.2
slsC	Appendix D.7.3	Appendix D.7.4

Table 9.5: Listings of $ServiceGroup\ servers$ and $Service\ servers$.

9.4.3 SNG Service Listing Servers

A Service Listing Server works in a similar fashion as the Application Servers. We offered two versions of Service Listing as shown in Table 9.6. The Single-service listing allows the

Service	Implementation files	Program Listing
Single-service listing	slsSGroup.cpp	Appendix D.7.1
	slsSServer.cpp	Appendix D.7.2
All-services listing	slsCGroup.cpp	Appendix D.7.3
	slsCServer.cpp	Appendix D.7.4

Table 9.6: Data files used in the implementation.

user to specify which service to list by entering the service ID. A valid service ID includes $time$, $name$, $date$, $echo$, $slsS$ and $slsC$. The All-services listing simple dump all the services available without providing any search capability to enhance the performance.

We modified and enhanced the Binary Search Tree bst module to provide our $SPath$ search.

9.5 Service Path Technology

In order to use $SPath$ efficiently, we coded the sp and $path$ modules to create and access $SPaths$. It is worth mentioning the pForward() and the pBackward() functions which are used by AS to determine the next hop for user authentication information. They extract the relevant information from the $SPath$ passed to them and return an integer to indicate completion of the process. The algorithms used by pForward() and pBackwrad are shown in Algorithm 5 and Algorithm 6

Algorithm 5: Algorithm for pForward()

```
Get SAPath from SPath
Get the position of current AS in the SAPath
If the next entry in the SAPath is not an IP address
{
    Next hop address is set to null
    Return 0 to caller
}
else
{
    Set the next entry in the SAPath as the next hop address
    Return 1 to caller
}
end
```

Algorithm 6: Algorithm for pBackward()

```
Get SAPath from SPath
Get the position of current AS in the SAPath
If the privious entry in the SAPath is not an IP address
{
    Next hop address is set to null
    Return 0 to caller
}
else
{
    Set the privious entry in the SAPath as the next hop address
    Return 1 to caller
}
end
```

9.6 Dynamic Password and Encryption

The Dynamic Password is used as the authentication protocol for the implementation. We designed the $security$ module to perform the traffic encryption as well as session encryption. Two encryption methods, $Round\ Robin$ and $Substitution$ are available and user can use either one or both of them. For the $Round\ Robin$ method, we need to specify the amount of displacement from the original character set. For the $Substitution$ method, we need to choose which one of the five substitution tables is to be used.

To handle the messages involved in the Dynamic Password protocol, we implemented in the $security$ module, all functions listed in Table 9.7.

Function	Use
msgOne()	Create msgOne.
msgTwo()	Create msgTwo.
msgThree()	Create msgThree.
parseMsgOne()	Access information in msgOne.
parseMsgTwo()	Access information in msgTwo.
parseMsgThree()	Access information in msgThree.

Table 9.7: Functions used by Dynamic Password.

We downloaded a $md5ADT$ module from public domain[1] to provide the required hashing function.

9.7 Chapter Summary

We have looked at how to use SNG with $DPass$ on a real aggregate of autonomous networks platform. The implementation can handle the cases when user is at his home network accessing services offered by his home network as well as other service providing networks. The user can also enjoy the same level of service sharing even when he is not in his home network and roaming in another network with the SNG. At the same time, off-shelf applications can easily be offered as a shared service. The implementation demonstrated that SNG with $DPass$ is indeed a practical service sharing protocol for dynamic aggregates of autonomous networks.

[1]MD5 - Command Line Message Digest Utility from http://www.fourmilab.ch/md5/. The web site is maintained by J. Walker.

Chapter 10

Conclusion and Future Direction

In Chapter 2 we have presented some existing approach to the service sharing over the IP networks. We elaborated each one of them with the workings and the drawbacks. The review showed that it was so imperative to develop a new protocol for service sharing among dynamically aggregated autonomous networks. Following the review is our design of SNG, $DPass$, and $KEDP$. We also proved that the message such as the service list and users authentication information on the encrypted channel are securely transmitted across the SNG. In Chapter 7, we proved that SNG using $DPass$ is correct as a secure authentication protocol. In this chapter, we summarize our study and list out a number of open questions.

10.1 Problems and Challenges in Service Sharing

Service sharing has becomes increasingly popular in past decades. There exists some approaches to address the issues risen in the area of authentication, authorization and revocation while sharing services among many autonomous networks

The major issues associated with service sharing are

- who confirms the availability of a services?
- who authenticates the user?
- who authorizes the user?
- how revocation can be executed?

In our study, we proposed the Service Network Graph (SNG) in Chapter 3. An SNG with the simplest logical topology is established when a network N_1 delegates its authentication authority to another network N_2 by sharing an Authentication Token Key with

N_2. BY delegating its authentication authority to N_2, N_1 trusts any user authenticated and authorized by N_2 for service access.

All service access information including access path, cost, and options, is listed in a $SPath$. Users can browse service available to him by requesting a list of all $SPath$ available. When N_1 delegates its authentication authority to N_2, it also shares all its $SPath$s with N_2. N_2 then makes all $SPath$s, including its own and those shared by N_1, available to its users on request. N_2 users can now access all services available from N_2 as well as from N_1.

The SNG handles the service requests from mobile users in a similar way. The extra step for service request is to route the service request and authentication information to the home network for authentication and service information using an encrypted channel.

When the authentication or authorization status of a user is changed, his home network pushes an revocation token to all servers currently providing services to the user so that they can terminate their services according to their own predefine control policies.

Network N_1 can revoke its authentication delegation to network N_2 by pushing a revocation message to all members of the SNG. All networks then adjust their SOT and $APNLT$ to reflect the changes.

In Chapter 6, $DPass$ is proposed as a candidate authentication to be used in SNG. It features a strong but easy password capable of revealing spoofing and resisting Dictionary attacks.

For better performance and scalability, we have shown that $SPath$s are optimized. We have also shown that by encrypting a message with the authentication token key, identity of the message sender can be assured. A theoretically proof for the correctness of SNG has been given in Chapter 7. As an illustration, we simulate an SNG model using $OMNeT++$ network simulator in Chapter 8. The simulation scenarios include the cases when a user is at his home network and in a visited network.

Finally, we round up our discussion about SNG with a simple implementation of SNG using $DPass$, Virtual Local Area Network ($VLAN$) and inter-VLAN routing in Chapter 9.

10.2 Methodology

Inspired by the demand and the advance in technology, we tackled the service sharing issues by designing and developing SNG and $DPass$. The features of SNG and $SPath$ are as follows:

1. There is no need to establish dedicated key control entities such as KDC and TGS in Kerberos. In SNG, an AS only keeps track of the authentication token key of all network it attaches to and networks attached to it.

2. Service providing servers do not have to be re-programmed to accept tickets as required by Kerberos. SNG only requires them to use session keys and conform to the service information they give to the users.

3. There is no need for dedicated TGS to keep track of the service paths. SNG uses $SPath$ to keep track of the path to reach a services. $SPath$ is particularly useful for ad hoc aggregation of networks for sharing of services.

4. There is no need to go through an hierarchy of entities such as an hierarchy of KDC to obtain service approval. With $SPath$ Optimization, the traffic in SNG is only two stops, end-to-end flow for both authenticating users and accessing service, thus eliminating traffic bottle neck and single point of failure.

5. With Authentication Delegation, a member network of an SNG may have only one shared authentication token key only and be able to access all the services provided by the authentication delegator network. This makes service sharing readily available to ad hoc aggregate of networks.

6. In SNG availability of a service is indicated by the existence of a corresponding $SPath$. The list if service path is maintained by a Service Listing Server. A user does not have to remember which services are available.

7. In SNG, service sharing is enabled by Authentication Delegation and Authentication Re-delegation. There is no domain within an SNG. Networks can join or leave an SNG in a dynamic and ad hoc way.

8. Authentication delegation and $SPath$ remove any constrain as to which service SNG can share. From relatively simple services like Date and Time service to national security services such as Passenger Clearance Service.

9. User can access services not only provided by the home network, services other networks shared with his home network are also accessible.

10. As long as authentication and service requests can be routed to the home network, SNG users can roam among networks in an SNG and still enjoy the services as if he stays in the home network.

11. A network is only required to attach to a member of SNG to participate in the service sharing. There is no central registration required.

12. While the linkages in Mobile Host Routing is one-to-one, linkages in SNG can be one-to-many. When a network attaches to an SNG authentication re-delegation extends the linkage and connects a single attachment to a graph of links. Administrative overhead for establishing service sharing is thus reduced.

13. An SNG member are required to have, at the minimum, one shared key with another member of the SNG. This will keep the administrative overhead of participating AS to a minimum.

14. SNG is not vulnerable to DNS attack as in OpenID.

15. In SNG authentication is delegated. The home network still does the authentication. Each network may set up their own identity check before registering a user. Home networks can implement the same identity binding check as VeriSign so that a user is actually the one whom he claims to be.

10.3 Outcomes

We have seen how existing technologies handle service sharing. To mitigate limitations of the existing technologies, we proposed SNG and $DPass$ in our study. The outcomes of our study are shown below.

- Service Network Graph which provides service sharing for dynamic aggregation of autonomous networks using Authentication Delegation, Authentication Re-delegation, $SPath$ and $SPath$ Optimization.
- Dynamic Password and its associated Key Exchange Protocol as a candidate authentication scheme in SNG.
- Application of a 4-network SNG model to two OMNeTT++ simulations.
- A practical application of the basic features of a 4-network SNG using C++.
- A formal proof for the correctness of SNG using $DPass$.
- A formal proof for self-authentication of an encrypted channel.
- A formal proof that $SPath$ can be optimized.

10.4 Problems for Future Studies

The work on service sharing and SNG is far from complete. We need to consider issues like:

- When a network attaches to an SNG, the sharing of authentication token key is to use another secure channel. An automatic and secure mechanism is desirable.

- When a network N_1 revokes its authentication delegation to another network N_2, N_1 has to broadcast its revocation message to all members of the SNG. It would be more efficient if N_1 is only required to notify N_2 and trusting N_2 passes the information on to other networks attached to N_2 thereby removing the need for a broadcast.

- $SOpt$ imposed conditions when a service is shared. Role based conditions can only be enforced when user role is known. The system of naming roles and the role hierarchy used differs from network to network. How to evaluate the roles and make a user role available for checking during authentication is a high priority issue.

- We need to investigate the possibility of optimizing the topology of SNG so that it can better scale to large network aggregations.

Let us take a look again at the example we gave in Chapter 1, Section 1.3 as one of our challenge in the near future. In the example, Hong Kong and Macau can be considered as two autonomous network regarding custom clearance. They implemented the same Automatic Passenger Clearance System using finger print for identity matching. The two regions are in fact operating the custom clearance independently. The only thing that relates the two regions regarding custom clearance is that a copy of user finger prints can be transferred from one region to another to be used as the common authentication information for a user entering either one of the regions.

We can consider Hong Kong as one network HK with resident U_{HK} and Macau as another network MA with resident U_{MA}. We can look at the custom clearance process from an SNG context. Suppose HK attaches to MA. MA then delegates its custom clearance authority to HK. When U_{HK} request entry to MA, U_{HK} is now a roaming user. MA can simply forward the authentication information of U_{HK} to HK and allow U_{HK} entry when HK confirms the identity of U_{HK}. No more waiting for the error prune manual transfer of authentication information and synchronizing two versions of the information for the same person.

Instead of passwords, finger prints are used in the custom clearance process. We need to pass a minutiae map of at least twelve minutiae points [27, 79] so that a definite matching [52, 28, 21] can be done. Our initial consideration is a random combination of the possible minutiae points to form a minutiae map for a particular authentication session to avoid the possible Dictionary attack.

In summary, SNG is a practical, efficient and scalable service sharing infrastructure which needs fine tuning and added features. Much work is needed to be done in the future.

Bibliography

[1] X.500 (11/08) information technology - open systems interconnection - the directory: Overview of concepts, models and services. *International Telecommunication Union ITU-T Recommendations X series*, 11 2008.

[2] X.509 (11/08) information technology - open systems interconnection - the directory: Public-key and attribute certificate frameworks. *International Telecommunication Union ITU-T Recommendations X series*, 11 2008.

[3] M. Abadi and A. D. Gordon. Reasoning about cryptographic protocols in the spi calculus. *CONCUR 97, Lecture Notes in Computer Science, Springer-Verlag*, pages 59–73, July 1997.

[4] M. Abadi and M. R. Tuttle. A semantics for a logic of authentication (extended abstract). *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, pages 201–216, August 1991.

[5] A. Abdul-Rahman and S. Hailes. Using recommendations for managing trust in distributed systems. In *Proc. of IEEE Malaysia International Conference on Communication (MICC'97), Kuala Lumpur*, 1997.

[6] A. Abdul-Rahman and S. Hailes. Supporting trust in virtual communities. *Hawaii Int. Conference on System Sciences 33 , Maui, Hawaii, January 2000*, 6:6007, January 2000.

[7] A. Abdul-Rahman and S. Halles. A distributed trust model. *Proceedings of New Security Paradigms Workshops 1997*, pages 48–60, 1997.

[8] A. R. Au, M. Looi, and P. Ashley. Automated cross organisational trust establishment on extranets. *Proceedings of the workshop on information technology for virtual enterprises, 2001*, (7):3–11, January 2001.

[9] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. *Proceedings of the IEEE Symposium on Security and Privacy, Oakland, California*, pages 72–84, May 1992.

[10] S. M. Bellovin and M. Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. *ACM Conference on Computer and Communications Security*, pages 244–250, 1993.

[11] S. M. Bellovin and M. Merritt. Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and password file compromise. pages 244–250, 1993.

[12] T. Beth, M. Borcherding, and B. Klien. Valuation of trust in open networks. *In ESORICS '94: Proceedings of the Third European Symposium on Research in Computer Security (1994), pp. 3-18.*, pages 3–18, 1994.

[13] M. Burrows, M. Abadi, and R. M. Neednam. A logic of authentication. *ACM Transactions on Computer Systems*, 8:18–36, 1990.

[14] Basic Law Consulative Committee. The basic law of the hong kong special administrative region of the people's republic of china. 4 April 1990.

[15] D. Denning. A new paradigm for trusted systems. *Proceedings of 1992-1993 ACM SIGSAC New Security Paradigms Workshop*, pages 36–41, 1993.

[16] F. J. T. Fabrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocols correct? *Proceedings, 1998 IEEE Symposium on Security and Privacy*, May 1998.

[17] F. J. T. Fabrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security, 7(1999)*, pages 191–230, 1999.

[18] D. Florencio, C. Herley, and B. Coskun. Do strong passwords accomplish anything? *Proceedings, USENIX Workshop on Hot Topics in Security 2007*, pages 1–6, 2007.

[19] L. A. Gordon, M. P. Loeb, W. Lucyshyn, and R. Richardson. 2005 csi/fbi computer crime and security survey. *Computer and Security Institute and Federal Bureau of Investigation*, 2005.

[20] N. Haller, C. Metz, P. Nezzerr, and M. Straw. A one-time password system. *RFC2289, Internet Engineering Task Force*, Feb 1998.

[21] Z. Han and C. Liu. *Fingerprint Classification Based on Statistical Features and Singular Point Information*. Lecture Notes in Computer Science. Lecture Notes in Computer Science, SpringerLink, 2005.

[22] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[23] T. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.

[24] Government of Hong Kong SAR Immigration Department. Automated passenger clearance system. *http://www.immd.gov.hk/ehtml/20041216.htm*, 16 December 2004.

[25] D. P. Jablon. Strong password-only authenticated key exchange. *Computer Communication Review*, 26(5):5–26, 1996.

[26] D. P. Jablon. Extended password key exchange protocols immune to dictionary attack. *Proceedings of the WETICE'97 Workshop on Enterprise Security, Cambridge, MA, USA.*, 1997.

[27] A. Jain, Y. Chen, and M. Demirkus. Pores and ridges: Fingerprint matching using level 3 features. volume 477-480, pages 1046–1049, 2006.

[28] A. Jain and S. Pankanti. Fingerprint classification and matching. In *Handbook for Image and Video Processing*. Academic Press, 2000.

[29] S. Josefsson. Extended kerberos version 5 key distribution center (kdc) exchanges over tcp. *RFC5021, Internet Engineering Task Force*, Aug 2007.

[30] P. Kaijser. A review of the sesame development. In *Information Security and Privacy*, volume 1438/1998 of *Lecture Notes in Computer Science*, pages 1–8. Springer Berlin / Heidelberg, 1998.

[31] P. Kaijser, T. Parker, and D. Pinkas. Sesame: The solution to security for open distributed systems. volume 17, pages 501–518, 1994.

[32] R. Kohlas and U. Maurer. Confidence valuation in a public-key infrastructure based on uncertain evidence. In *In Public Key Cryptography*, pages 93–112. Springer, 2000.

[33] D. Lai and Z. Zhang. Integrated key exchange protocol capable of revealing spoofing and resisting dictionary attacks. *Technical Track Proceedings, 2nd International Conference, Applied Cryptography and Network Security, Yellow Mountain*, pages 115–124, June 2004.

[34] D. Lai and Z. Zhang. An infrastructure for service authentication and authorization revocation in a dynamic aggregation of networks. *WSEAS Transactions on Communications*, 4(8):537–547, August 2005.

[35] D. Lai and Z. Zhang. Network service sharing infrastructure: Service authentication and authorization revocation. *Proceedings, the 9th WSEAS International Conference on Communications. (CD Proceedings)*, July 2005.

[36] D. Lai and Z. Zhang. Secure service sharing over networks for mobile users using service network graphs. *Proceedings, Wireless Telecommunication Syposium 2006, Pamona, Ca, USA. (CD Proceedings)*, April 2006.

[37] D. Lai and Z. Zhang. Self-authentication of encrypted channels in service network graph. *Proceedings, 2008 IFIP International Conference on Network and Parallel Computing, (NPC 2008)*, pages 163–167, October 2008.

[38] D. Lai and Z. Zhang. Efficient information propagation in service routing for next generation network. *Proceedings, The Fourth International Conference on Rough Set and Knowledge Technology*, pages 342–349, 7 2009.

[39] D. Lai and Z. Zhang. Improving efficiency and scalability of service network graph by re-routing. *Proceedings, 1st Asian Conference on Itelligent Information and Database Systems 2009. (CD Proceedings)*, 4 2009.

[40] D. Lai and Z. Zhang. Service re-routing for service network graph: Efficiency, scalability and implementation. *International journal of Computer Networks Communications (IJCNC). ISBN (on-line) 0974-9322. ISBN (print) 0975-2293.*, 1(1), 4 2009.

[41] D. Lai, Z. Zhang, and C. Shen. Achieving secure service sharing over ip networks. *Proceedings, ASEE Mid-Atlantic Section Spring 2006 Conference. (CD Proceedings)*, April 2006.

[42] D. Lai, Z. Zhang, and H. Wang. Towards an authentication protocol for service outsourcing over ip networks. *Proceedings, the 2005 International Conference on Security and Management*, (7):3–9, June 2005.

[43] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, 1992.

[44] D. Lee. Macau crossings to take just 2 seconds. *http://www.thestandard.com.hk/news_detail.asp?pp_cat=30&art_id=91069&sid=26206847&con_type=1*, 25 Nov 2009.

[45] X. Liang and T. Asano. Fingerprint matching using minutia polygons. pages 1046–1049, 2006.

[46] S. Lucks. Open key exchange: How to defeat dictionary attacks without encrypting public keys. *Ecole Normale Suprieure, Paris*, pages 79–90, April 1997.

[47] Melnikov. The kerberos v5 (gssapi) simple authentication and security layer (sasl) mechanism. *RFC4752, Internet Engineering Task Force*, Nov 2006.

[48] M. Montaner, B. Lopez, and J. L. Rosa. Developing trust in recommender agents. pages 304–305, 2002.

[49] M. Naor and K. Nissim. Certificate revocation and certificate update. *Proceedings 7th USENIX Security Symposium (San Antonio, Texas).*, January 1998.

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

[51] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The kerberos network authentication service (v5). *RFC4120, Internet Engineering Task Force*, July 2005.

[52] National Institute of Standards and Technology. Matching up fingerprints: Solving crimes, improving security. *http://www.nist.gov/public_affairs/licweb/fingerprint_int/nist_fingerprint.html*.

[53] National Institute of Standards and Technology. Password usage. *Federal Information Processing Standards Publication*, 99(112):1–100, May 1985.

[54] T.A. Parker. A secure european system for applications in a multi-vendor environment (the sesame project). volume 3072/2004, pages 139–156. Chapman and Hall, Ltd. London, UK, UK, 1993.

[55] G. Parziale1 and A. Niel1. A fingerprint matching using minutiae triangulation. In *Biometric Authentication*, volume 3072/2004 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin / Heidelberg, 2004.

[56] L. C. Paulson. Proving properties of security protocols by induction. *10th IEEE Computer Security Foundation Workshop*, pages 70–83, 1997.

[57] M. Reiter and S. Stubblebine. Authentication metric analysis and design. *ACM Transactions on Information and System Security*, 2(2):138–158, January 1999.

[58] Trans-European Research and Education Networking Association (TEREN) Task Force on Mobility (TF-mobility). Inventory of 802.1x-based solutions for inter-nrens roaming. *http://www.terena.org/activities/tf-mobility/deliverables/delD/DelD_v1.2-f.pdf*, 2004.

[59] Trans-European Research and Education Networking Association (TEREN) Task Force on Mobility (TF-mobility). Inventory of vpn-based solutions for inter-nren roaming. *http://www.terena.org/activities/tf-mobility/deliverables/delE/DeliEv4.4-np.pdf*, 2004.

[60] Trans-European Research and Education Networking Association (TEREN) Task Force on Mobility (TF-mobility). Inventory of web-based solution for inter-nren roaming. *http://www.terena.org/activities/tf-mobility/deliverables/delF/DelF-f.pdf*, 2004.

[61] Trans-European Research and Education Networking Association (TEREN) Task Force on Mobility (TF-mobility). Preliminary selection for inter-nren roaming. *http://www.terena.org/activities/tf-mobility/deliverables/delG/DelG-final.pdf*, 2004.

[62] Trans-European Research and Education Networking Association (TEREN) Task Force on Mobility (TF-mobility). Tf-mobility roaming policy document. *http://www.terena.org/activities/tf-mobility/deliverables/delI/Roaming_policy_document_v.1.2.pdf*, 2004.

[63] Trans-European Research and Education Networking Association (TEREN) Task Force on Mobility (TF-mobility). Inter-nren roaming architecture: Description and development items. *http://www.eduroam.org/downloads/docs/GN2-06-137v5-Deliverable_DJ5-1-4_Inter-NREN_Roaming_Technical_Specification_20060908164149.pdf*, Sept 2006.

[64] Trans-European Research and Education Networking Association (TEREN) Task Force on Mobility (TF-mobility). Eduroam service definition and implementation plan. *http://www.eduroam.org/downloads/docs/GN2-07-327v2-DS5_1_1-_eduroam_Service_Definition.pdf*, Jan 2007.

[65] R. Richardson. 2008 csi computer crime and security survey. *Computer and Security Institute*, 2008.

[66] R. Rivest. The md5 message-digest algorithm. In *RFC1321, Internet Engineering Task Force*, April.

[67] S. Robles, J. Borrell, J. Bigham, L. Tokarchuk, and L. Cuthbert. Design of a trust model for a secure multi-agent marketplace. *Proceedings of the fifth international conference on Autonomous agents*, pages 77–78, 2001.

[68] S. Schneider. Verifying authentication protocols with csp. *Proceedings of the 10th IEEE Computer Security Foundations Worshop, IEEE Computer Society Press, 1997*, pages 3–17, 1997.

[69] M. Steiner, G. Tsudik, and M. Waidner. Refinement and extension of encrypted key exchange. *Operating Systems Review*, 29(3):22–30.

[70] S. G. Stubblebine. Recent-secure authentication: Enforcing revocation in distributed systems. *IEEE Computer Society Symposium on Security and Privacy, Oakland, California.*, May 1995.

[71] A. Tanenbaum. *Computer Networks, Fourth Edition*. Pearson Eduaction International, 2003.

[72] Inc. Technology Support, Cisco Systems. Inter-vlan routing. *http://www.cisco.com/en/US/tech/tk389/tk815/tk857/tsd_technology_support_sub-protocol_home.html*, May 2003.

[73] Inc. Technology Support, Cisco Systems. Virtual lans/vlan trunking protocol (vlans/vtp). *http://www.cisco.com/en/US/tech/tk389/tk689/tsd_technology_support_protocol_home.html*, March 2005.

[74] Ashlee Vance. If your password is 123456, just make it hackme. *The New York Times*, 20 Jan 2010.

[75] N. Williams. A pseudo-random function (prf) for the kerberos v generic security service application program interface (gss-api) mechanism. *RFC4402, Internet Engineering Task Force*, Feb 2006.

[76] N. Williams. Generic security service application program interface (gss-api) domain-based service names mapping for the kerberos v gss mechanism. *RFC5179, Internet Engineering Task Force*, May 2008.

[77] T. Y. C. Woo and S. S. Lam. Verifying authentication protocols: Methodology and example. *Proc. Int. Conference on Network Protocols*, October 1993.

[78] T. Wu. The secure remote password protocol. *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium, San Diego, CA*, pages 97–111, 1998.

[79] W. Xu, X. Chen, and J. Feng. A robust fingerprint matching approach: Growing and fusing of local structures. In *Advances in Biometrics*, volume 4642/2007 of *Lecture Notes in Computer Science*, pages 134–143, 2007.

[80] L. Zhu, K. Jaganathan, and S. Hartman. A pseudo-random function (prf) for the kerberos v generic security service application program interface (gss-api) mechanism. *RFC4121, Internet Engineering Task Force*, July 2005.

[81] L. Zhu, K. Jaganathan, and N. Williams. Online certificate status protocol (ocsp) support for public key cryptography for initial authentication in kerberos (pkinit). *RFC4557, Internet Engineering Task Force*, June 2006.

[82] L. Zhu, P. Leach, and K. Jaganathan. Kerberos cryptosystem negotiation extension. *RFC4537, Internet Engineering Task Force*, June 2006.

[83] L. Zhu and B. Tung. Public key cryptography for initial authentication in kerberos (pkinit). *RFC4556, Internet Engineering Task Force*, June 2006.

Appendices

Appendix A

The following Axioms and Theorem appear in Lampson et al [43].
The symbols used are:

A.1 Axiom P10

$\vdash (P_A \ \mathbf{says} \ (P_B \ \Rightarrow\!\!\!\!\!\!\rightarrow \ P_A)) \ \Rightarrow \ (P_B \ \Rightarrow\!\!\!\!\!\!\rightarrow \ P_A)$

A.2 Theorem P8

$\vdash (P_A \ \Rightarrow\!\!\!\!\!\!\rightarrow \ P_B) \ \Rightarrow \ ((P_A \ \mathbf{says} \ s) \ \Rightarrow \ (P_B \ \mathbf{says} \ s))$

A.3 Theorem P11

$\vdash ((P_C \ \Rightarrow\!\!\!\!\!\!\rightarrow \ P_A) \ \wedge \ (P_C \ \mathbf{says} \ (P_B \ \Rightarrow\!\!\!\!\!\!\rightarrow \ P_A))) \Rightarrow \ (P_B \ \Rightarrow\!\!\!\!\!\!\rightarrow \ P_A)$

Appendix B

The following axioms, definitions, lemmas, notational convention, propositions and theorems appear in Guttman et al [17].

Let $\mathcal{A}$ be a set of plain text messages with elements m, m'; and $\mathcal{K}$ be a set of unique encryption keys with elements K, K'.

B.1 Axiom 1

For $m,\ m'\ \in \mathcal{A}$ and $K,\ K'\ \in \mathcal{K}$,

$$\{m\}_K\ =\ \{m'\}_{K'}\ \Rightarrow\ m\ =\ m'\ \wedge\ K\ =\ K'$$

B.2 Definition 2.1

A **signed term** is a pair $< \sigma, a >$ with $a\ \in\ \mathcal{A}$ and σ one of the symbols $+$, $-$. We will write a signed term as $+$t or $-$t. $(\pm\mathcal{A})$* is the set of finite sequences of signed terms. We will denote a typical element of $(\pm\mathcal{A})$* by $<< \sigma_1, a_1 >, ..., < \sigma_n a_n >>$.

B.3 Definition 2.11

The subterm relation $\sqsubset$ is defined inductively, as the smallest relation such that:

1. $a\ \sqsubset\ a$
2. $a\ \sqsubset\ \{g\}_K$ iff $(a\ \sqsubset\ g)$ or $(a\ =\ K)$ or $(a\ =\ \{g\}_K)$
3. $a\ \sqsubset\ gh$ if $(a\ \sqsubset\ g)$ or $(a\ \sqsubset\ h)$

B.4 Axiom 2

For $m_0,\ m'_0,\ m_1,\ m'_1\ \in \mathcal{A}$ and $K,\ K'\ \in \mathcal{T}$

1. $(m_0 m_1\ =\ m'_0 m'_1)\ \Rightarrow\ (m_0\ =\ m'_0)\ \wedge\ (m_1\ =\ m'_1)$
2. $m_0 m_1\ \neq\ \{m_0\}_K\ \neq\ hash(m_0)$
3. $m_0 m_1\ \notin\ \mathcal{T}$
4. $\{m_0\}_K\ \notin\ \mathcal{T}$

B.5 Proposition 2.12

Suppose $K\ \neq\ K'$ and $\{h'\}_{K'}\ \sqsubset\ \{h\}_K$. Then $\{h'\}_{K'}\ \sqsubset\ h$

B.6 Definition 2.2

A $Strand\ Space$ over $\mathcal{A}$ is a set Σ together with a trace mapping $tr\ :\ \Sigma\ \rightarrow\ (\pm\mathcal{A})$*.

B.7 Definition 2.3

Fix a strand space Σ

1. A **node** is a pair $<\ s, i\ >$, with $s\ \in\ \Sigma$ and i is an integer satisfying $1\ \leq\ i\ \leq\ length(tr(s))$. The set of nodes is denoted by $\mathcal{N}$. We will say the node $< s, i >$ belongs to the strand s. Clearly, every node belongs to a unique strand.

2. If $n\ =\ <\ s, i\ >\ \in\ \mathcal{N}$ then $index(n) = i$ and $strand(n) = s$. Define $term(n)$ to be $(tr(s))_i$, i.e. the i th signed term in the trace of s. Similarly, uns_term(n) is $((tr(s))_i)_2$ unsigned part of the ith signed term in the trace of s.

3. There is an edge $n_1\ \rightarrow\ n_2$ iff term(n_1) $=\ +a$ and term(n_2) $=\ -a$ for some $a\ \in\ \mathcal{A}$. Intuitively, the edge means that node n_1 sends the message a, which is received by n_2, recording a potential causal link between those strands.

4. When $n_1\ =\ <\ s, i\ >$ and $n_2\ =\ <\ s, i+1\ >$ are members of $\mathcal{N}$, there is an edge $n_1\ \Rightarrow\ n_2$. Intuitively, the edge expresses that n_1 is an immediate causal predecessor of n_2 on the strand s. We write n' $\Rightarrow^+$ n to mean that n' precedes n (not necessarily immediately) on the same strand.

5. An unsigned term t occurs in n $\in \mathcal{N}$ iff t $\sqsubset$ term(n).

6. Suppose I is a set of unsigned terms. The nodes $n \in \mathcal{N}$ is an **entry point** for I iff term(n) $= +t$ for some $t \in I$, and whenever n' $\Rightarrow^+ n$, term(n') $\notin$ I.

7. An unsigned term t **originates** on $n \in \mathcal{N}$ iff n is an entry point for the set $I = \{t' : t \sqsubset t'\}$.

8. An unsigned term t is uniquely originating iff t originates on a unique n $\in \mathcal{N}$.

B.8 Definition 2.4

Suppose

- $\rightarrow_C \subset \rightarrow$;
- $\Rightarrow_C \subset \Rightarrow$;
- $\mathcal{C} = <\mathcal{N}_C, (\rightarrow_C \cup \Rightarrow_C)>$ is a subgraph of $<\mathcal{N}, (\rightarrow \cup \Rightarrow)>$.

$\mathcal{C}$ is a bundle if:

1. $\mathcal{C}$ is finite.
2. If $n_2 \in \mathcal{N}_C$ and term(n_2) is negative, then there is a unique n_1 such that $n_1 \rightarrow_C n_2$.
3. If $n_2 \in \mathcal{N}_C$ and $n_1 \Rightarrow n_2$ then $n_1 \Rightarrow_C n_2$.
4. $\mathcal{C}$ is acyclic.

B.9 Notational Convention 2.5

A node n is in a bundle $\mathcal{C} = <\mathcal{N}_C, (\rightarrow_C \cup \Rightarrow_C)>$, written $n \in \mathcal{C}$, if $n \in \mathcal{N}_C$; a strand s is in $\mathcal{C}$ if all its nodes are in $\mathcal{N}_c$.

If $\mathcal{C}$ is a bundle, then the $\mathcal{C}$-height of a strand s is the largest i such that $< s, i > \in \mathcal{C}$. $\mathcal{C}$-trace(s) = $< tr(s)(1), \ldots, tr(s)(m) >$, where $m = \mathcal{C}$-height(s).

B.10 Definition 2.6

If $\mathcal{S}$ is a set of edges, i.e. $\mathcal{S} \subset \rightarrow \cup \Rightarrow$ then $\prec_S$ is the transitive closure of $\mathcal{S}$, and $\preceq_S$ is the reflexive, transitive closure of $\mathcal{S}$

B.11 Lemma 2.7

Suppose $\mathcal{C}$ is a bundle. Then $\preceq_C$ is a partial order, i.e. a reflexive, antisymmetric, transitive relation. Every non-empty subset of the nodes in $\mathcal{C}$ has $\preceq_C$-minimal members.

B.12 Lemma 2.8

Suppose $\mathcal{C}$ is a bundle, and $\mathcal{S} \subseteq \mathcal{C}$ is a set of nodes such that

$$\forall \text{m, m' uns\_term(m)} = \text{uns\_term(m')} \text{ implies } (m \in \mathcal{S} \text{ iff m'} \in \mathcal{S})$$

If n is a $\preceq_c$-minimal member of $\mathcal{S}$, then the sign of n is positive.

B.13 Lemma2.9

Suppose $\mathcal{C}$ is a bundle, t $\in \mathcal{A}$ and n $\in \mathcal{C}$ is a $\preceq_C$-minimal element of $\{m \in \mathcal{C} : t \sqsubset term(m)\}$. The node n is an originating occurrence for t.

B.14 Definition 3.1

A penetrator trace is one of the following:

M. Text message: $< +t >$ where $t \in \mathcal{T}$

F. Flushing: $< -g >$

T. Tee: $< -g, +g, +g >$

C. Concatenation: $< -g, -h, +gh >$

S. Separation into components: $< -gh, +g, +h >$

K. Key: $< +K >$ where $K \in \mathcal{K}_P$

E. Encryption: $< -K, -h, +\{h\}_K >$

D. Decryption: $< -K^{-1}, -\{h\}_K, +h >$

B.15 Proposition 3.3

Let $\mathcal{C}$ be a bundle, and let K $\in \mathcal{K} \setminus \mathcal{K}_P$.

If K never originates on a regular node, then K $\not\sqsubset$ term(n) for any node n $\in \mathcal{C}$. In particular, for any penetrator node p $\in \mathcal{C}$, K $\not\sqsubset$ term(p).

B.16 Definition 3.2

An infiltrated strand space is a pair $(\Sigma, \mathcal{P})$ with Σ a strand space and $\mathcal{P} \subseteq \Sigma$ such that tr(p) is a penetrator trace for all p $\in \mathcal{P}$.

A strand s $\in \Sigma$ is a penetrator strand if it belongs to $\mathcal{P}$, and a node is a penetrator node if the strand it lies on is a penetrator strand. Otherwise we will call it a non-penetrator or regular strand or node.

A node n is an **M, F,** etc. node if it lies on a penetrator strand with a trace of kind **M, F,** etc.

Appendix C

C.1 Listing of omnetpp.ini

```
[General]
    network = dssinet

[Cmdenv]
    runs-to-execute = 1
   #express-mode = yes
    module-messages = yes
    event-banners = yes

[Tkenv]
    default-run = 1
    use-mainwindow = yes
    print-banners = yes
    slowexec-delay = 300ms

[Parameters]
    dssinet.dssia.num_requests = 1
    dssinet.dssis.num_requests = 1
    dssinet.dssisls.num_requests = 1
    dssinet.dssiu.num_requests = 1
```

C.2 Listing of Makefile

```
#  Makefile for dssi #
# ** This file was manually written #
# Name of target to be created (-o option) TARGET = dssi
# User interface (uncomment one) (-u option)
# USERIF_LIBS=$(CMDENV_LIBS) USERIF_LIBS=$(TKENV_LIBS)
# uncomment 1 of the 3 lines to support either serial or
#  parallel operation KERNEL_LIBS=$(STD_KERNEL_LIBS)
# KERNEL_LIBS=$(PVM_KERNEL_LIBS)
# KERNEL_LIBS=$(MPI_KERNEL_LIBS)
# .ned or .h include paths with -I INCLUDE_PATH=
# misc additional object and library files to link
# EXRA_OBJS=
# object files in subdirectories SUBDIR_OBJS=
# Additional libraries (-L option -l option) LIBS=
#------------------------------------------------------------

NEDC=nedc
MSGC=opp_msgc
CXX=g++
CC=gcc
AR=ar cr
SHLIB_LD=g++ -shared -fPIC
MAKEDEPEND=opp_makedep -Y --objdirtree

CFLAGS=-O3 -D
NDEBUG=1  -fPIC
NEDCFLAGS=-Wno-unused
LDFLAGS=
EXE_SUFFIX=

OMNETPP_INCL_DIR=/home/mcsci/lai/omnet/omnetpp-2.3/include
OMNETPP_LIB_DIR=/home/mcsci/lai/omnet/omnetpp-2.3/lib

TK_LIBS=-L/usr/X11R6/lib -lX11 -ltk8.3 -ltcl8.3
MPI_LIBS=
PVM_LIBS=-lgpvm3 -lpvm3
SYS_LIBS=-ldl -lstdc++
SYS_LIBS_PURE=-lm
$(shell $(CXX) -print-file-name=libstdc++.a)

# User interface libs CMDENV_LIBS=-lenvir -lcmdenv
TKENV_LIBS=-lenvir -ltkenv $(TK_LIBS)

# Simulation kernel STD_KERNEL_LIBS=-lsim_std
MPI_KERNEL_LIBS=-lsim_mpi $(MPI_LIBS)
PVM_KERNEL_LIBS=-lsim_pvm
$(PVM_LIBS)
```

```
# Simulation kernel and user interface libraries
OMNETPP_LIBS=-L$(OMNETPP_LIB_DIR) $(USERIF_LIBS)
$(KERNEL_LIBS) $(SYS_LIBS)

COPTS=$(CFLAGS) $(INCLUDE_PATH) -I$(OMNETPP_INCL_DIR)
NEDCOPTS=$(CFLAGS) $(NEDCFLAGS) $(INCLUDE_PATH)
-I$(OMNETPP_INCL_DIR)

#------------------------------------------------------

# Object files from this directory to link
OBJS= dssinet_n.o dssia_n.o dssis_n.o dssisls_n.o
      dssiu_n.o dssir_n.o dssisw_n.o dssia.o dssis.o
      dssisls.o dssiu.o dssisw.o dssir.o

$(TARGET): $(OBJS) $(EXRA_OBJS) Makefile
    $(CXX) $(LDFLAGS) $(OBJS) $(EXRA_OBJS) $(SUBDIR_OBJS)
    $(LIBS) $(OMNETPP_LIBS) -o $(TARGET)

purify: $(OBJS) $(EXRA_OBJS)   Makefile
    purify $(CXX) $(LDFLAGS) $(OBJS) $(EXRA_OBJS)
                  $(SUBDIR_OBJS) $(LIBS) -L
                  $(OMNETPP_LIB_DIR) $(KERNEL_LIBS)
                  $(USERIF_LIBS) $(SYS_LIBS_PURE) -o
                  $(TARGET).pure

dssinet_n.o: dssinet_n.cc
    $(CXX) -c $(NEDCOPTS) dssinet_n.cc

dssinet_n.cc: dssinet.ned
    $(NEDC) $(INCLUDE_PATH) dssinet.ned

dssia_n.o: dssia_n.cc
    $(CXX) -c $(NEDCOPTS) dssia_n.cc

dssia_n.cc: dssia.ned
    $(NEDC) $(INCLUDE_PATH) dssia.ned

dssis_n.o: dssis_n.cc
    $(CXX) -c $(NEDCOPTS) dssis_n.cc

dssis_n.cc: dssis.ned
    $(NEDC) $(INCLUDE_PATH) dssis.ned

dssisls_n.o: dssisls_n.cc
    $(CXX) -c $(NEDCOPTS) dssisls_n.cc

dssisls_n.cc: dssisls.ned
    $(NEDC) $(INCLUDE_PATH) dssisls.ned

dssiu_n.o: dssiu_n.cc
    $(CXX) -c $(NEDCOPTS) dssiu_n.cc

dssiu_n.cc: dssiu.ned
    $(NEDC) $(INCLUDE_PATH) dssiu.ned

dssisw_n.o: dssisw_n.cc
    $(CXX) -c $(NEDCOPTS) dssisw_n.cc

dssisw_n.cc: dssisw.ned
    $(NEDC) $(INCLUDE_PATH) dssisw.ned

dssir_n.o: dssir_n.cc
    $(CXX) -c $(NEDCOPTS) dssir_n.cc

dssir_n.cc: dssir.ned
    $(NEDC) $(INCLUDE_PATH) dssir.ned

dssia.o: dssia.cc
    $(CXX) -c $(COPTS) dssia.cc

dssis.o: dssis.cc
    $(CXX) -c $(COPTS) dssis.cc

dssisls.o: dssisls.cc
    $(CXX) -c $(COPTS) dssisls.cc

dssiu.o: dssiu.cc
    $(CXX) -c $(COPTS) dssiu.cc

dssisw.o: dssisw.cc
    $(CXX) -c $(COPTS) dssisw.cc

dssir.o: dssir.cc
    $(CXX) -c $(COPTS) dssir.cc

doc: neddoc.html htmldocs

neddoc.html: dssinet.ned dssia.ned dssis.ned dssisls.ned
```

```
            dssiu.ned dssisw.ned dssir.ned
 @opp_neddoc dssinet.ned dssia.ned dssis.ned dssisls.ned
            dssiu.ned dssisw.ned dssir.ned > neddoc.html
 @echo File  neddoc.html generated.

htmldocs:
@doxygen -g- | sed "s/^PROJECT_NAME.*/PROJECT_NAME
   = fifo1/;\
s|^INPUT *=.*|INPUT = . |;\
s/^EXTRACT_ALL *=.*/EXTRACT_ALL = yes/;\
s/^EXTRACT_PRIVATE *=.*/EXTRACT_PRIVATE = yes/;\
s/^EXCLUDE_PATTERNS *=.*/EXCLUDE_PATTERNS
   = *_n.cc *_n.h/;\
s/^ALPHABETICAL_INDEX *=.*/ALPHABETICAL_INDEX = yes/;\
s/^HTML_OUTPUT *=.*/HTML_OUTPUT = htmldoc/;\
s/^GENERATE_LATEX *=.*/GENERATE_LATEX = no/;\
s/^GENERATE_TREEVIEW *=.*/GENERATE_TREEVIEW = yes/;\
s/^HIDE_UNDOC_RELATIONS *=.*/HIDE_UNDOC_RELATIONS = no/;\
s|^TAGFILES *=.*|TAGFILES
   = /home/andras/omnetpp-2.3/doc/api/
   opptags.xml=/home/andras/omnetpp-2.3/doc/api|;\
s|^GENERATE_TAGFILE *=.*|GENERATE_TAGFILE
   = htmldoc/tags.xml|;\
s/^QUIET *=.*/QUIET = yes/" > doxy.cfg
@doxygen doxy.cfg
@echo Code documentation generated. Now, point your web
   browser to ./htmldoc/index.html.

clean:
    rm -f *.o *_n.cc *_n.h *_m.cc *_m.h $(TARGET)
       $(EXE_SUFFIX)
    rm -f *.vec *.sca
    rm -rf neddoc.html htmldoc

depend:
    $(MAKEDEPEND) $(INCLUDE_PATH) -- *.cc
    # $(MAKEDEPEND) $(INCLUDE_PATH) -fMakefile.in -- *.cc

re-makemake:
    opp_makemake -f    #recreate Makefile

re-makemake-m:
    opp_makemake -f -m    #recreate Makefile.in
```

```
# DO NOT DELETE THIS LINE -- make depend depends on it.
dssia.o:
dssia.cc \
  dssia.h
dssis.o: dssis.cc \
  dssis.h
dssisls.o: dssisls.cc \
  dssisls.h
dssiu.o: dssiu.cc \
 dssiu.h
dssisw.o: dssisw.cc \
 dssisw.h
dssir.o: dssir.cc \
 dssir.h
%
```

C.3 Listing of dssinet.ned

```
//---------------------------------------------------------
// file: dssinet.ned
// ---------------------------------------------------------
// Specify modules used
import
    "dssisls",
    "dssis",
    "dssiu",
    "dssia",
    "dssisw",
    "dssir";

module DssiGraph
    submodules:
// Network 1 (the home network)
        AS1: DSSI_a
            parameters:
                address = 102,   // Address for the node
                dim = 4;         // Used in testing
            gatesizes:
                inputs[1],
                outputs[1];
            display: "p=70,70;i=server1";
            // specify the icon and location
        S1: DSSI_s
            parameters:
```

```
                address = 103,
                dim = 4;
            gatesizes:
                inputs[1],
                outputs[1];
            display: "p=110,70;i=server1";
        SLS1: DSSI_sls
            parameters:
                address = 101,
                dim = 4;
            gatesizes:
                inputs[1],
                outputs[1];
            display: "p=150,70;i=server1";
        SW1: DSSI_sw
            parameters:
                address = 105,
                dim = 4;
            gatesizes:
                inputs[4],
                outputs[4];
            display: "p=130,140;i=switch1";
        R1: DSSI_r
            parameters:
                address = 106,
                dim = 4;
            gatesizes:
                inputs[4],
                outputs[4];
            display: "p=130,210;i=router";
// Network 2
        AS2: DSSI_a
            parameters:
                address = 202,
                dim = 4;
            gatesizes:
                inputs[1],
                outputs[1];
            display: "p=270,70;i=server1";
        S2: DSSI_s
            parameters:
                address = 203,
                dim = 4;
            gatesizes:
                inputs[1],
                outputs[1];
            display: "p=310,70;i=server1";
        SLS2: DSSI_sls
            parameters:
                address = 201,
                dim = 4;
            gatesizes:
                inputs[1],
                outputs[1];
            display: "p=350,70;i=server1";
        U2: DSSI_u
            parameters:
                address = 204,
                dim = 4;
            gatesizes:
                inputs[1],
                outputs[1];
            display: "p=390,70;i=comp2_s";
        SW2: DSSI_sw
            parameters:
                address = 205,
                dim = 4;
            gatesizes:
                inputs[4],
                outputs[4];
            display: "p=330,140;i=switch1";
        R2: DSSI_r
            parameters:
                address = 206,
                dim = 4;
            gatesizes:
                inputs[4],
                outputs[4];
            display: "p=330,210;i=router";
// Network 3 (the foreign network)
        AS3: DSSI_a
            parameters:
                address = 302,
                dim = 4;
            gatesizes:
                inputs[1],
                outputs[1];
            display: "p=70,460;i=server1";
        S3: DSSI_s
            parameters:
```

```
            address = 303,
            dim = 4;
        gatesizes:
            inputs[1],
            outputs[1];
        display: "p=110,460;i=server1";
    SLS3: DSSI_sls
        parameters:
            address = 301,
            dim = 4;
        gatesizes:
            inputs[1],
            outputs[1];
        display: "p=150,460;i=server1";
    U1: DSSI_u
        parameters:
            address = 304,
            dim = 4;
        gatesizes:
            inputs[1],
            outputs[1];
        display: "p=190,460;i=comp2_s";
    SW3: DSSI_sw
        parameters:
            address = 305,
            dim = 4;
        gatesizes:
            inputs[4],
            outputs[4];
        display: "p=130,390;i=switch1";
    R3: DSSI_r
        parameters:
            address = 306,
            dim = 4;
        gatesizes:
            inputs[4],
            outputs[4];
        display: "p=130,320;i=router";
// Network 4
    AS4: DSSI_a
        parameters:
            address = 402,
            dim = 4;
        gatesizes:
            inputs[1],
            outputs[1];
        display: "p=270,460;i=server1";
    S4: DSSI_s
        parameters:
            address = 403,
            dim = 4;
        gatesizes:
            inputs[1],
            outputs[1];
        display: "p=310,460;i=server1";
    SLS4: DSSI_sls
        parameters:
            address = 401,
            dim = 4;
        gatesizes:
            inputs[1],
            outputs[1];
        display: "p=350,460;i=server1";
    U4: DSSI_u
        parameters:
            address = 404,
            dim = 4;
        gatesizes:
            inputs[1],
            outputs[1];
        display: "p=390,460;i=comp2_s";
    SW4: DSSI_sw
        parameters:
            address = 405,
            dim = 4;
        gatesizes:
            inputs[4],
            outputs[4];
        display: "p=330,390;i=switch1";
    R4: DSSI_r
        parameters:
            address = 406,
            dim = 4;
        gatesizes:
            inputs[4],
            outputs[4];
        display: "p=330,320;i=router";
// module connections
    connections nocheck:
        AS1.inputs[0] <-- SW1.outputs[1];
```

```
S1.inputs[0] <-- SW1.outputs[2];
SLS1.inputs[0] <-- SW1.outputs[0];
SW1.inputs[1] <-- AS1.outputs[0];
SW1.inputs[2] <-- S1.outputs[0];
SW1.inputs[0] <-- SLS1.outputs[0];
R1.from_switch <-- SW1.to_router;
SW1.from_router <-- R1.to_switch;

AS2.inputs[0] <-- SW2.outputs[1];
S2.inputs[0] <-- SW2.outputs[2];
SLS2.inputs[0] <-- SW2.outputs[0];
U2.inputs[0] <-- SW2.outputs[3];
SW2.inputs[1] <-- AS2.outputs[0];
SW2.inputs[2] <-- S2.outputs[0];
SW2.inputs[0] <-- SLS2.outputs[0];
SW2.inputs[3] <-- U2.outputs[0];
R2.from_switch <-- SW2.to_router;
SW2.from_router <-- R2.to_switch;

AS3.inputs[0] <-- SW3.outputs[1];
S3.inputs[0] <-- SW3.outputs[2];
SLS3.inputs[0] <-- SW3.outputs[0];
U1.inputs[0] <-- SW3.outputs[3];
SW3.inputs[1] <-- AS3.outputs[0];
SW3.inputs[2] <-- S3.outputs[0];
SW3.inputs[0] <-- SLS3.outputs[0];
SW3.inputs[3] <-- U1.outputs[0];
R3.from_switch <-- SW3.to_router;
SW3.from_router <-- R3.to_switch;

AS4.inputs[0] <-- SW4.outputs[1];
S4.inputs[0] <-- SW4.outputs[2];
SLS4.inputs[0] <-- SW4.outputs[0];
U4.inputs[0] <-- SW4.outputs[3];
SW4.inputs[1] <-- AS4.outputs[0];
SW4.inputs[2] <-- S4.outputs[0];
SW4.inputs[0] <-- SLS4.outputs[0];
SW4.inputs[3] <-- U4.outputs[0];
R4.from_switch <-- SW4.to_router;
SW4.from_router <-- R4.to_switch;

R1.inputs[1] <-- R2.outputs[0] display "o=red,2";
R1.inputs[2] <-- R3.outputs[0] display
                 "m=m,50,0,50,0;o=red,2";
R1.inputs[3] <-- R4.outputs[0] display "o=grey,0";
        R2.inputs[0] <-- R1.outputs[1] display "o=grey,0";
        R2.inputs[2] <-- R3.outputs[1] display "o=grey,0";
        R2.inputs[3] <-- R4.outputs[1] display "o=red,2";
        R3.inputs[0] <-- R1.outputs[2] display
                         "m=m,50,0,50,0;o=red,2";
        R3.inputs[1] <-- R2.outputs[2] display "o=grey,0";
        R3.inputs[3] <-- R4.outputs[2] display "o=grey,0";
        R4.inputs[0] <-- R1.outputs[3] display
                         "m=m,0,40,0,40;o=red,2";
        R4.inputs[1] <-- R2.outputs[3] display "o=grey,0";
        R4.inputs[2] <-- R3.outputs[3] display "o=grey,0";

endmodule
network dssinet : DssiGraph
endnetwork
```

C.4 Server Module

C.4.1 Listing of dssis.ned

```
//---------------------------------------------------------
// file: dssis.ned
//---------------------------------------------------------
simple DSSI_s
    parameters:
      address,
      dim;
    gates:
      in: inputs[];
      out: outputs[];
endsimple
```

C.4.2 Listing of dssis.h

```
//---------------------------------------------------------
// file: dssis.h
//---------------------------------------------------------
#ifndef __DSSIS_H #define __DSSIS_H #include <omnetpp.h>

class DSSI_s : public cSimpleModule {
    Module_Class_Members(DSSI_s,cSimpleModule,16384)
    virtual void activity();
};
#endif
```

C.4.3 Listing of dssis.cc

```
//---------------------------------------------------------
// file: dssis.cc
//---------------------------------------------------------
#include "dssis.h"
Define_Module( DSSI_s );

void DSSI_s::activity() {
    cMessage *msg;
    int start, stop, address;
    address = par("address");

    for (;;)
    {
        msg = receive();
        start = msg->par("start");
        stop = msg->par("stop");
        msg->par("start") = stop;
        msg->par("stop") = start;
        ev << "From S server/start/stop/src/dest = ";
        ev << msg->par("server") << "/";
        ev << msg->par("start") << "/";
        ev << msg->par("stop") << "/";
        ev << msg->par("src") << "/";
        ev << msg->par("dest") << "/";
        ev << endl;
        send ( msg, "outputs", 0);
    }
}
```

C.5 Service Listing Server Module

C.5.1 Listing of dssisls.ned

```
//---------------------------------------------------------
// file: dssisls.ed
//---------------------------------------------------------
simple DSSI_sls
    parameters:
      address,
      dim;
    gates:
      in: inputs[];
      out: outputs[];
endsimple
```

C.5.2 Listing of dssisls.h

```
//---------------------------------------------------------
// file: dssisls.h
//---------------------------------------------------------
#ifndef __DSSISLS_H #define __DSSISLS_H
#include <omnetpp.h>

class DSSI_sls : public cSimpleModule {
    Module_Class_Members(DSSI_sls,cSimpleModule,16384)
    virtual void activity();
};
#endif
```

C.5.3 Listing of dssisls.cc

```
//---------------------------------------------------------
// file: dssisls.cc
//---------------------------------------------------------
#include "dssisls.h"
Define_Module( DSSI_sls );

void DSSI_sls::activity() {
    cMessage *msg;
    int start, stop;
    char * cname;

    for (;;)
    {
        msg = receive();
        start = msg->par("start");
        stop   = msg->par("stop");
        msg->par("start") = stop;
        msg->par("stop") = start;
        ev << "From SLS server/start/stop/src/dest/ = ";
        ev << msg->par("server") << "/";
        ev << msg->par("start") << "/";
        ev << msg->par("stop") << "/";
        ev << msg->par("src") << "/";
        ev << msg->par("dest") << "/";
        ev << endl;
```

```
        send( msg, "outputs", 0);
    }
}
```

C.6 Switch Module

C.6.1 Listing of dssisw.ned

```
//---------------------------------------------------------
// file: dssisw.ned
//---------------------------------------------------------
simple DSSI_sw
    parameters:
      address,
      dim;
    gates:
      in: inputs[], from_router;
      out: outputs[], to_router;
endsimple
```

C.6.2 Listing of dssisw.h

```
//---------------------------------------------------------
// file: dssisw.h
//---------------------------------------------------------
#ifndef __DSSISW_H #define __DSSISW_H
#include <omnetpp.h>

class DSSI_sw : public cSimpleModule {
    Module_Class_Members(DSSI_sw,cSimpleModule,16384)
    virtual void activity();
};
#endif
```

C.6.3 Listing of dssisw.cc

```
//---------------------------------------------------------
// file: dssisw.cc
//---------------------------------------------------------
#include "dssisw.h"
Define_Module( DSSI_sw );

void DSSI_sw::activity() {
    cMessage *msg;
    int address, stop, stopNet, stopHost;

    address = par("address");
    for (;;)
    {
        msg = receive();
        stop = msg->par("stop");
        stopNet = stop / 100;
        stopHost = stop % 100;
        ev << "From SW server/start/stop/src/dest = ";
        ev << msg->par("server") << "/";
        ev << msg->par("start") << "/";
        ev << msg->par("stop") << "/";
        ev << msg->par("src") << "/";
        ev << msg->par("dest") << "/";
        ev << endl;
        if (stopNet != (address / 100)){
            send( msg, "to_router" );
        } else {
            send(msg, "outputs", stopHost-1);
        }
    }
}
```

C.7 Router Module

C.7.1 Listing of dssir.ned

```
//---------------------------------------------------------
// file: dssir.ned
//---------------------------------------------------------
simple DSSI_r
    parameters:
      address,
      dim;
    gates:
      in: inputs[], from_switch;
      out: outputs[], to_switch;
endsimple
```

C.7.2 Listing of dssir.h

```
//---------------------------------------------------------
// file: dssir.h
//---------------------------------------------------------
#ifndef __DSSIR_H #define __DSSIR_H
#include <omnetpp.h>

class DSSI_r : public cSimpleModule {
    Module_Class_Members(DSSI_r,cSimpleModule,16384)
    virtual void activity();
};

#endif
```

C.7.3 Listing of dssir.cc

```
//---------------------------------------------------------
// file: dssir.cc
//---------------------------------------------------------
#include "dssir.h"
Define_Module( DSSI_r );

void DSSI_r::activity() {
    cMessage *msg;
    int address, addressNet, stop, stopNet;
    address = par("address");
    addressNet = address / 100;

    for (;;)
    {
        msg = receive();
        stop = msg->par("stop");
        stopNet = stop / 100;
        ev << "From R server/start/stop/src/dest = ";
        ev << msg->par("server") << "/";
        ev << msg->par("start") << "/";
        ev << msg->par("stop") << "/";
        ev << msg->par("src") << "/";
        ev << msg->par("dest") << "/";
        ev << endl;
        if (addressNet != stopNet) {
            // To different network, pass to other router
            send( msg, "outputs", stopNet-1 );
        } else {
            send( msg, "to_switch" );
        }
    }
}
```

C.8 User Module

C.8.1 Listing of dssiu.ned

```
//---------------------------------------------------------
// file: dssiu.ned
//---------------------------------------------------------
simple DSSI_u
    parameters:
      address,
      dim;
    gates:
      in: inputs[];
      out: outputs[];
endsimple
```

C.8.2 Listing of dssiu.h

```
//---------------------------------------------------------
// file: dssiu.h
//---------------------------------------------------------
#ifndef __DSSIU_H #define __DSSIU_H
#include <omnetpp.h>

class DSSI_u : public cSimpleModule {
    Module_Class_Members(DSSI_u,cSimpleModule,16384)
    virtual void activity();
};

#endif
```

C.8.3 Listing of dssiu.cc for User at Home Network

```
//---------------------------------------------------------
// file: dssiu.cc
//---------------------------------------------------------
#include "dssiu.h"
```

```
Define_Module( DSSI_u );

void DSSI_u::activity() {
    char msgname[32];
    cMessage *msg;
    int network=200, i, j, k, address, src, dest, start,
        stop, target, count;

    address = par("address");
    /* SLS address = nn1
       AS  address = nn2
       S   address = nn3
       U   address = nn4
       where nn = 10, 20, 30, 40 fro different networks
       msgname is nnxtommy where nnx is src, mmy is dest
       start, stop are the intermediate hosts.
    */
    // Only U1 sends messages address == 104
    if (address==104){

        while (true){
            /* A sequence of repeating messages:
               network to 100 for first service request
               network to 400 for second service request
               network to 200 for third service request
            */
            // network 100, 400, 200
            switch (network){
                case 100:
                    network = 400;
                    break;
                case 400:
                    network = 200;
                    break;
                case 200:
                    network = 100;
                    break;
                default:
                    network = 200;
                    break;
            }

            for (i=1; i<4; i++){
            /* i = 1 for Service query
                 = 2 for Service request
                 = 3 for Service access
            */
            // Service access three times.
             if (i==3) count =3;
             else count =1;
             for(j=0; j<count; j++){
                 /* Service query always to SLS1.
                    Service request always to AS1.
                      target specifies the network of
                        SLS1 and AS1 (and U1).
                      target = 101 Service query.
                      target = 102 Service request.
                    Service access has network
                      specified  by the "network".
                 */
                 target = 100 + i;
                 if (i == 3){
                     // to server
                     sprintf( msgname, "%dto%d",
                              address, network + 3);
                     msg = new cMessage( msgname, i );
                     msg->addPar("stop") = network + 3;
                     msg->addPar("dest") = network + 3;

                 } else {
                     // to other hosts
                     sprintf( msgname, "%dto%d",
                              address, target);
                     msg = new cMessage( msgname, i );
                     msg->addPar("stop") = target;
                     msg->addPar("dest") = target;
                 }

                 msg->addPar("src") = address;
                 msg->addPar("start") = address;
                 msg->addPar("server") = network +3;

                 ev << "From U server/start/stop/src
                        /dest = ";
                 ev << msg->par("server") << "/";
                 ev << msg->par("start") << "/";
                 ev << msg->par("stop") << "/";
                 ev << msg->par("src") << "/";
                 ev << msg->par("dest") << "/";
                 ev << endl;
```

```
                    send( msg, "outputs", 0 );
                    msg = receive();
                }
            }
        }
    }
}
```

C.8.4 Listing of dssiu.cc for User at Foreign Network

```
//---------------------------------------------------------
// file: dssiu.cc
//---------------------------------------------------------
#include "dssiu.h"
Define_Module( DSSI_u );

void DSSI_u::activity() {
    char msgname[32];
    cMessage *msg;
    int network=300, i, j, k, address, src, dest, start,
        stop, target, count;

    address = par("address");
    /* SLS address = net1
       AS  address = net2
       S   address = net3
       U   address = net4
       SW  address = net5
       R   address = net6
       where net = 10, 20, 30, 40 for different networks
    */
    // Only U1 sends messages and U1 address == 304
    // Other users do not initial service request
    if (address==304){
        /* Hello message to AS3.
           AS3 will get the service from SLS1 and
               return the service list to SLS3 and U1.
           This Hello process is a one-off process.
        */

        // to AS3
        sprintf( msgname, "304 to 302");
        msg = new cMessage( msgname, 5 );
        msg->addPar("stop") = 302;
        msg->addPar("dest") = 302;
        msg->addPar("src") = 304;
        msg->addPar("start") = 304;
        msg->addPar("server") = 303;
        send( msg, "outputs", 0 );
        // from AS3
        // path = (U1-AS3-SLS1-AS3-U1)
        msg = receive();

        while (true){
            /* initial network = 300 for Hello message
               Then a sequence of repeating messages:
               network to 100 for first service request
               network to 400 for second service request
               network to 200 for third service request
            */
            switch (network){
                case 100:
                    network = 400;
                    break;
                case 400:
                    network = 200;
                    break;
                case 200:
                    network = 300;
                    break;
                case 300:
                    network = 100;
                    break;
                default:
                    network = 200;
                    break;
            }

            for (i=1; i<4; i++){
                /* i = 1 for Service query
                     = 2 for Service request
                     = 3 for Service access
                */
                // Service access three times.
                if (i==3) count =3;
                else count =1;
                for(j=0; j<count; j++){
                    /* Service query always to SLS3.
```

```
                Service request always to AS3.
                  target specifies the network of
                  SLS3 and AS3 (and U1).
                  target = 301 Service query.
                  target = 302 Service request.
                Service access will have network
                  specified  by the "network".
              */
              target = 300 + i;
              if (i == 3){
                  // Service access
                  sprintf( msgname, "%dto%d",
                           address, network + 3);
                  msg = new cMessage( msgname, i );
                  msg->addPar("stop") = network + 3;
                  msg->addPar("dest") = network + 3;
              } else {
                  // Service query or Service request
                  sprintf( msgname, "%dto%d",
                           address, target);
                  msg = new cMessage( msgname, i );
                  msg->addPar("stop") = target;
                  msg->addPar("dest") = target;
              }

              msg->addPar("src") = address;
              msg->addPar("start") = address;
              msg->addPar("server") = network +3;

              ev << "From U server/start/stop/src
                     /dest = ";
              ev << msg->par("server") << "/";
              ev << msg->par("start") << "/";
              ev << msg->par("stop") << "/";
              ev << msg->par("src") << "/";
              ev << msg->par("dest") << "/";
              ev << endl;
              send( msg, "outputs", 0 );
              msg = receive();
          }
      }
  }
}
```

C.9 Authentication Server Module

C.9.1 Listing of dssia.ned

```
//---------------------------------------------------------
// file: dssia.ned
//---------------------------------------------------------
simple DSSI_a
    parameters:
      address,
      dim;
    gates:
      in: inputs[];
      out: outputs[];
endsimple
```

C.9.2 Listing of dssia.h

```
//---------------------------------------------------------
// file: dssia.h
//---------------------------------------------------------

#ifndef __DSSIA_H #define __DSSIA_H
#include <omnetpp.h>
class DSSI_a : public cSimpleModule {
    Module_Class_Members(DSSI_a,cSimpleModule,16384)
    virtual void activity();
};

#endif
```

C.9.3 Listing of dssia.cc for user at Home Network

```
//---------------------------------------------------------
// file: dssia.cc
//---------------------------------------------------------

#include "dssia.h"


Define_Module( DSSI_a );
int SNG_forward (int serverNet, int stopNet){
    int stop;
```

```
/* SNG config as shown in Fig 9 of WSEAS journal:
   if serverNet == 1, stopNet = 2 then stop = 102
   if serverNet == 1, stopNet = 3 then stop = 102
   if serverNet == 1, stopNet = 4 then stop = 202
   if serverNet == 2, stopNet = 1 then stop = 402
   if serverNet == 2, stopNet = 3 then stop = 102
   if serverNet == 2, stopNet = 4 then stop = 202
   if serverNet == 3, stopNet = 1 then stop = 302
   if serverNet == 3, stopNet = 2 then stop = 102
   if serverNet == 3, stopNet = 4 then stop = 202
   if serverNet == 4, stopNet = 1 then stop = 402
   if serverNet == 4, stopNet = 2 then stop = 102
   if serverNet == 4, stopNet = 3 then stop = 102
*/
switch (serverNet){
    case 1:
        switch (stopNet){
            case 2:
                stop = 102;
                break;
            case 3:
                stop = 102;
                break;
            case 4:
                stop = 202;
                break;
        }
        break;
    case 2:
        switch (stopNet){
            case 1:
                stop = 402;
                break;
            case 3:
                stop = 102;
                break;
            case 4:
                stop = 202;
                break;
        }
        break;
    case 3:
        switch (stopNet){
            case 1:
                stop = 302;
                break;
            case 2:
                stop = 102;
                break;
            case 4:
                stop = 202;
                break;
        }
        break;
    case 4:
        switch (stopNet){
            case 1:
                stop = 402;
                break;
            case 2:
                stop = 102;
                break;
            case 3:
                stop = 102;
                break;
        }
        break;
    }
    return stop;
}

int SNG_backward (int srcNet, int stopNet){
    int stop;

    /* SNG config as shown in Fig 9 of WSEAS journal:
       if srcNet == 1, stopNet = 2 then stop = 402
       if srcNet == 1, stopNet = 3 then stop = 102
       if srcNet == 1, stopNet = 4 then stop = 102
       if srcNet == 2, stopNet = 1 then stop = 202
       if srcNet == 2, stopNet = 3 then stop = 102
       if srcNet == 2, stopNet = 4 then stop = 102
       if srcNet == 3, stopNet = 1 then stop = 302
       if srcNet == 3, stopNet = 2 then stop = 402
       if srcNet == 3, stopNet = 4 then stop = 102
       if srcNet == 4, stopNet = 1 then stop = 202
       if srcNet == 4, stopNet = 2 then stop = 402
       if srcNet == 4, stopNet = 3 then stop = 102
    */
    switch (srcNet){
        case 1:
```

```
    switch (stopNet){
        case 2:
            stop = 402;
            break;
        case 3:
            stop = 102;
            break;
        case 4:
            stop = 102;
            break;
    }
    break;
case 2:
    switch (stopNet){
        case 1:
            stop = 202;
            break;
        case 3:
            stop = 102;
            break;
        case 4:
            stop = 102;
            break;
    }
    break;
case 3:
    switch (stopNet){
        case 1:
            stop = 302;
            break;
        case 2:
            stop = 402;
            break;
        case 4:
            stop = 102;
            break;
    }
    break;
case 4:
    switch (stopNet){
        case 1:
            stop = 202;
            break;
        case 2:
            stop = 402;
                    break;
                case 3:
                    stop = 102;
                    break;
            }
            break;
    }
    return stop;
}


void DSSI_a::activity()
{
    cMessage *msg;
    int tmp, address, start, stop, src, dest, server,
        serverNet, startNet,
        stopNet, srcNet;
    address = par("address");

    for (;;)
    {
        msg = receive();

        start = msg->par("start");
        stop = msg->par("stop");
        src = msg->par("src");
        dest = msg->par("dest");
        server = msg->par("server");
        serverNet = server / 100;
        stopNet = stop / 100;
        srcNet = src / 100;

        if (serverNet != stopNet){
            if ((stop == dest) && (start != src)){
//            if (start != src){
                // from remote AS, pass to U
                msg->par("start") = stop;
                msg->par("stop") = src;
                send ( msg, "outputs", 0);
            } else {
                //to remote AS using SNG config
                msg->par("start") = stop;
                tmp = SNG_forward(serverNet, stopNet);
                if (start == tmp){
                    // backward trip
```

```
                tmp = SNG_backward(srcNet, stopNet);
            }
            msg->par("stop") = tmp;
            send ( msg, "outputs", 0);
        }
    } else {
        // local
        if (server == start){
            // from server
            if (serverNet == srcNet){
                // local service request, to user
                msg->par("start")=stop;
                msg->par("stop")=src;
                send ( msg, "outputs", 0 );
            } else {
                // remote service request,
                // to AS using SNG config
                msg->par("start")=stop;
                msg->par("stop")=
                     SNG_backward(srcNet, stopNet);
                send ( msg, "outputs", 0 );
            }
        } else {
            // from U, pass to server
            msg->par("start")=stop;
            msg->par("stop")=server;
            send ( msg, "outputs", 0 );
        }
    }

    ev << "From AS server/start/stop/src/dest = ";
    ev << msg->par("server") << "/";
    ev << msg->par("start") << "/";
    ev << msg->par("stop") << "/";
    ev << msg->par("src") << "/";
    ev << msg->par("dest") << "/";
    ev << endl;
  }
}
```

C.9.4 Listing of dssia.cc for user at Foreign Network

```
//---------------------------------------------------------
// file: dssia.cc
//---------------------------------------------------------

#include "dssia.h"

Define_Module( DSSI_a );

// Determines next address to pass the service request
int SNG_forward (int serverNet, int stopNet){
    int stop;

    /* SNG config as defined in dssinet.ned:
       if serverNet == 1, stopNet = 2 then stop = 102
       if serverNet == 1, stopNet = 3 then stop = 102
       if serverNet == 1, stopNet = 4 then stop = 202
       if serverNet == 2, stopNet = 1 then stop = 402
       if serverNet == 2, stopNet = 3 then stop = 102
       if serverNet == 2, stopNet = 4 then stop = 202
       if serverNet == 3, stopNet = 1 then stop = 302
       if serverNet == 3, stopNet = 2 then stop = 102
       if serverNet == 3, stopNet = 4 then stop = 202
       if serverNet == 4, stopNet = 1 then stop = 402
       if serverNet == 4, stopNet = 2 then stop = 102
       if serverNet == 4, stopNet = 3 then stop = 102
    */
    switch (serverNet){
        case 1:
            switch (stopNet){
                case 2:
                    stop = 102;
                    break;
                case 3:
                    stop = 102;
                    break;
                case 4:
                    stop = 202;
                    break;
            }
            break;
        case 2:
```

```
            switch (stopNet){
                case 1:
                    stop = 402;
                    break;
                case 3:
                    stop = 102;
                    break;
                case 4:
                    stop = 202;
                    break;
            }
            break;
        case 3:
            switch (stopNet){
                case 1:
                    stop = 302;
                    break;
                case 2:
                    stop = 102;
                    break;
                case 4:
                    stop = 202;
                    break;
            }
            break;
        case 4:
            switch (stopNet){
                case 1:
                    stop = 402;
                    break;
                case 2:
                    stop = 102;
                    break;
                case 3:
                    stop = 102;
                    break;
            }
            break;
    }
    return stop;
}

// Determines next address to return the service request
int SNG_backward (int srcNet, int stopNet){
    int stop;

    /* SNG config as shown in Fig 9 of WSEAS journal:
       if srcNet == 1, stopNet = 2 then stop = 402
       if srcNet == 1, stopNet = 3 then stop = 102
       if srcNet == 1, stopNet = 4 then stop = 102
       if srcNet == 2, stopNet = 1 then stop = 202
       if srcNet == 2, stopNet = 3 then stop = 102
       if srcNet == 2, stopNet = 4 then stop = 102
       if srcNet == 3, stopNet = 1 then stop = 302
       if srcNet == 3, stopNet = 2 then stop = 402
       if srcNet == 3, stopNet = 4 then stop = 102
       if srcNet == 4, stopNet = 1 then stop = 202
       if srcNet == 4, stopNet = 2 then stop = 402
       if srcNet == 4, stopNet = 3 then stop = 102
    */
    switch (srcNet){
        case 1:
            switch (stopNet){
                case 2:
                    stop = 402;
                    break;
                case 3:
                    stop = 102;
                    break;
                case 4:
                    stop = 102;
                    break;
            }
            break;
        case 2:
            switch (stopNet){
                case 1:
                    stop = 202;
                    break;
                case 3:
                    stop = 102;
                    break;
                case 4:
                    stop = 102;
                    break;
            }
            break;
        case 3:
            switch (stopNet){
                case 1:
```

```
                    stop = 302;
                    break;
                case 2:
                    stop = 402;
                    break;
                case 4:
                    stop = 102;
                    break;
            }
            break;
        case 4:
            switch (stopNet){
                case 1:
                    stop = 202;
                    break;
                case 2:
                    stop = 402;
                    break;
                case 3:
                    stop = 102;
                    break;
            }
            break;
    }
    return stop;
}

void DSSI_a::activity() {
    cMessage *msg;
    int tmp, address, start, stop, src, dest, server,
        serverNet, startNet,
        stopNet, srcNet, mobile = 1;
    address = par("address");

    for (;;)
    {
        msg = receive();
        // Hello message handling for AS3
        if (address == 302) {
            /* mobile = 1 Hello message from U1
               mobile = 2 Service list from SLS1
               mobile = 3 Not Hello process
            */
            if (mobile == 1){
                msg->par("stop") = 101;
                msg->par("dest") = 101;
                msg->par("src") = 302;
                msg->par("start") = 302;
                msg->par("server") = 303;
                mobile = 2;
                send( msg, "outputs", 0 );
                msg = receive();
            }
            if (mobile == 2){
                msg->par("stop") = 304;
                msg->par("dest") = 302;
                msg->par("src") = 304;
                msg->par("start") = 302;
                msg->par("server") = 303;
                mobile = 3;
                send( msg, "outputs", 0 );
                msg = receive();
            }
        }
        start = msg->par("start");
        stop = msg->par("stop");
        src = msg->par("src");
        dest = msg->par("dest");
        server = msg->par("server");
        serverNet = server / 100;
        stopNet = stop / 100;
        srcNet = src / 100;

        /* serverNet != stopNet
           The AS is at not the same network as the
           service providing server. AS either send the
           service request / information to another AS
           using SNG_forward() or SNG_backward() to
           determine the next address or send the
           service information to user U1.
           All service query
               src = 304 dest = 301  server = net3
           All service request
               src = 304 dest = 302  server = net3
           All service request
               src = 304 dest = net3 server = net3
        */
        if (serverNet != stopNet){
        // AS NOT in same network as server.
            if ((stop == dest) && (start != src)){
```

```
    /* (stop == dest) means at AS3.
       (start != src) means the return of service
       information not the initial service
       request. So AS has to pass service
       information to user U1
    */
        msg->par("start") = stop;
        msg->par("stop") = src;
        ev << "case 1" << endl;
        send ( msg, "outputs", 0);
    } else {
    // AS has to pass to next hop using
    // sng_forward() or SNG_backward().
        msg->par("start") = stop;
        tmp = SNG_forward(serverNet, stopNet);
        if (start == tmp){
            // backward trip
            tmp = SNG_backward(srcNet, stopNet);
        }
        msg->par("stop") = tmp;
        ev << "case 2 & 3" << endl;
        send ( msg, "outputs", 0);
    }
} else {
    // AS is in the network as server
    ev << "local cases" << endl;
    if (server == start){
    // This is service information from server
        if (serverNet == srcNet){
        // local (net 300) service request,
        // send to user U1 directly
            msg->par("start")=stop;
            msg->par("stop")=src;
            send ( msg, "outputs", 0 );
        } else {
        // Remote service request.
        // Send to U1 via next hop using
        // SNG_backward()
            msg->par("start")=stop;
            msg->par("stop")=
                 SNG_backward(srcNet, stopNet);
            send ( msg, "outputs", 0 );
        }
    } else {
    // AS is in the same network as the server.
            // From user U1.
            // Therefore send to server.
                // from U, pass to server
                msg->par("start")=stop;
                msg->par("stop")=server;
                send ( msg, "outputs", 0 );
            }
            ev << "Other cases" << endl;
        }

        ev << "From AS server/start/stop/src/dest = ";
        ev << msg->par("server") << "/";
        ev << msg->par("start") << "/";
        ev << msg->par("stop") << "/";
        ev << msg->par("src") << "/";
        ev << msg->par("dest") << "/";
        ev << endl;
    }
}
```

C.10 Simulation Output

C.10.1 Listing of Message_sent_home.txt

```
% Message_sent_home.txt
% Comment lines added start with '%'
% Service query to SLS1:
SENT: 104to101 src=dssinet.U1   dest=dssinet.SW1
SENT: 104to101 src=dssinet.SW1  dest=dssinet.SLS1
SENT: 104to101 src=dssinet.SLS1 dest=dssinet.SW1
SENT: 104to101 src=dssinet.SW1  dest=dssinet.U1

% Authentication and S4 Service information:
SENT: 104to102 src=dssinet.U1   dest=dssinet.SW1
SENT: 104to102 src=dssinet.SW1  dest=dssinet.AS1
SENT: 104to102 src=dssinet.AS1  dest=dssinet.SW1
SENT: 104to102 src=dssinet.SW1  dest=dssinet.S1
SENT: 104to102 src=dssinet.S1   dest=dssinet.SW1
SENT: 104to102 src=dssinet.SW1  dest=dssinet.AS1
SENT: 104to102 src=dssinet.AS1  dest=dssinet.SW1
SENT: 104to102 src=dssinet.SW1  dest=dssinet.U1

% Three rounds of S1 Service access:
```

```
SENT: 104to103 src=dssinet.U1   dest=dssinet.SW1
SENT: 104to103 src=dssinet.SW1  dest=dssinet.S1
SENT: 104to103 src=dssinet.S1   dest=dssinet.SW1
SENT: 104to103 src=dssinet.SW1  dest=dssinet.U1

SENT: 104to103 src=dssinet.U1   dest=dssinet.SW1
SENT: 104to103 src=dssinet.SW1  dest=dssinet.S1
SENT: 104to103 src=dssinet.S1   dest=dssinet.SW1
SENT: 104to103 src=dssinet.SW1  dest=dssinet.U1

SENT: 104to103 src=dssinet.U1   dest=dssinet.SW1
SENT: 104to103 src=dssinet.SW1  dest=dssinet.S1
SENT: 104to103 src=dssinet.S1   dest=dssinet.SW1
SENT: 104to103 src=dssinet.SW1  dest=dssinet.U1

% Service query to SLS1:
SENT: 104to101 src=dssinet.U1   dest=dssinet.SW1
SENT: 104to101 src=dssinet.SW1  dest=dssinet.SLS1
SENT: 104to101 src=dssinet.SLS1 dest=dssinet.SW1
SENT: 104to101 src=dssinet.SW1  dest=dssinet.U1

% Authentication and S4 Service information:
SENT: 104to102 src=dssinet.U1   dest=dssinet.SW1
SENT: 104to102 src=dssinet.SW1  dest=dssinet.AS1
SENT: 104to102 src=dssinet.AS1  dest=dssinet.SW1
SENT: 104to102 src=dssinet.SW1  dest=dssinet.R1
SENT: 104to102 src=dssinet.R1   dest=dssinet.R4
SENT: 104to102 src=dssinet.R4   dest=dssinet.SW4
SENT: 104to102 src=dssinet.SW4  dest=dssinet.AS4
SENT: 104to102 src=dssinet.AS4  dest=dssinet.SW4
SENT: 104to102 src=dssinet.SW4  dest=dssinet.S4
SENT: 104to102 src=dssinet.S4   dest=dssinet.SW4
SENT: 104to102 src=dssinet.SW4  dest=dssinet.AS4
SENT: 104to102 src=dssinet.AS4  dest=dssinet.SW4
SENT: 104to102 src=dssinet.SW4  dest=dssinet.R4
SENT: 104to102 src=dssinet.R4   dest=dssinet.R1
SENT: 104to102 src=dssinet.R1   dest=dssinet.SW1
SENT: 104to102 src=dssinet.SW1  dest=dssinet.AS1
SENT: 104to102 src=dssinet.AS1  dest=dssinet.SW1
SENT: 104to102 src=dssinet.SW1  dest=dssinet.U1

% Three rounds of Service access:
SENT: 104to403 src=dssinet.U1   dest=dssinet.SW1
SENT: 104to403 src=dssinet.SW1  dest=dssinet.R1
SENT: 104to403 src=dssinet.R1   dest=dssinet.R4
```

```
SENT: 104to403 src=dssinet.R4   dest=dssinet.SW4
SENT: 104to403 src=dssinet.SW4  dest=dssinet.S4
SENT: 104to403 src=dssinet.S4   dest=dssinet.SW4
SENT: 104to403 src=dssinet.SW4  dest=dssinet.R4
SENT: 104to403 src=dssinet.R4   dest=dssinet.R1
SENT: 104to403 src=dssinet.R1   dest=dssinet.SW1
SENT: 104to403 src=dssinet.SW1  dest=dssinet.U1

SENT: 104to403 src=dssinet.U1   dest=dssinet.SW1
SENT: 104to403 src=dssinet.SW1  dest=dssinet.R1
SENT: 104to403 src=dssinet.R1   dest=dssinet.R4
SENT: 104to403 src=dssinet.R4   dest=dssinet.SW4
SENT: 104to403 src=dssinet.SW4  dest=dssinet.S4
SENT: 104to403 src=dssinet.S4   dest=dssinet.SW4
SENT: 104to403 src=dssinet.SW4  dest=dssinet.R4
SENT: 104to403 src=dssinet.R4   dest=dssinet.R1
SENT: 104to403 src=dssinet.R1   dest=dssinet.SW1
SENT: 104to403 src=dssinet.SW1  dest=dssinet.U1

SENT: 104to403 src=dssinet.U1   dest=dssinet.SW1
SENT: 104to403 src=dssinet.SW1  dest=dssinet.R1
SENT: 104to403 src=dssinet.R1   dest=dssinet.R4
SENT: 104to403 src=dssinet.R4   dest=dssinet.SW4
SENT: 104to403 src=dssinet.SW4  dest=dssinet.S4
SENT: 104to403 src=dssinet.S4   dest=dssinet.SW4
SENT: 104to403 src=dssinet.SW4  dest=dssinet.R4
SENT: 104to403 src=dssinet.R4   dest=dssinet.R1
SENT: 104to403 src=dssinet.R1   dest=dssinet.SW1
SENT: 104to403 src=dssinet.SW1  dest=dssinet.U1

% Service query to SLS1:
SENT: 104to101 src=dssinet.U1   dest=dssinet.SW1
SENT: 104to101 src=dssinet.SW1  dest=dssinet.SLS1
SENT: 104to101 src=dssinet.SLS1 dest=dssinet.SW1
SENT: 104to101 src=dssinet.SW1  dest=dssinet.U1

% Authentication and S2 Service information:
SENT: 104to102 src=dssinet.U1  dest=dssinet.SW1
SENT: 104to102 src=dssinet.SW1 dest=dssinet.AS1
SENT: 104to102 src=dssinet.AS1 dest=dssinet.SW1
SENT: 104to102 src=dssinet.SW1 dest=dssinet.R1
SENT: 104to102 src=dssinet.R1  dest=dssinet.R4
SENT: 104to102 src=dssinet.R4  dest=dssinet.SW4
SENT: 104to102 src=dssinet.SW4 dest=dssinet.AS4
SENT: 104to102 src=dssinet.AS4 dest=dssinet.SW4
```

```
SENT: 104to102 src=dssinet.SW4 dest=dssinet.R4
SENT: 104to102 src=dssinet.R4  dest=dssinet.R2
SENT: 104to102 src=dssinet.R2  dest=dssinet.SW2
SENT: 104to102 src=dssinet.SW2 dest=dssinet.AS2
SENT: 104to102 src=dssinet.AS2 dest=dssinet.SW2
SENT: 104to102 src=dssinet.SW2 dest=dssinet.S2
SENT: 104to102 src=dssinet.S2  dest=dssinet.SW2
SENT: 104to102 src=dssinet.SW2 dest=dssinet.AS2
SENT: 104to102 src=dssinet.AS2 dest=dssinet.SW2
SENT: 104to102 src=dssinet.SW2 dest=dssinet.R2
SENT: 104to102 src=dssinet.R2  dest=dssinet.R4
SENT: 104to102 src=dssinet.R4  dest=dssinet.SW4
SENT: 104to102 src=dssinet.SW4 dest=dssinet.AS4
SENT: 104to102 src=dssinet.AS4 dest=dssinet.SW4
SENT: 104to102 src=dssinet.SW4 dest=dssinet.R4
SENT: 104to102 src=dssinet.R4  dest=dssinet.R1
SENT: 104to102 src=dssinet.R1  dest=dssinet.SW1
SENT: 104to102 src=dssinet.SW1 dest=dssinet.AS1
SENT: 104to102 src=dssinet.AS1 dest=dssinet.SW1
SENT: 104to102 src=dssinet.SW1 dest=dssinet.U1

% Three rounds of S2 Service access:
SENT: 104to203 src=dssinet.U1  dest=dssinet.SW1
SENT: 104to203 src=dssinet.SW1 dest=dssinet.R1
SENT: 104to203 src=dssinet.R1  dest=dssinet.R2
SENT: 104to203 src=dssinet.R2  dest=dssinet.SW2
SENT: 104to203 src=dssinet.SW2 dest=dssinet.S2
SENT: 104to203 src=dssinet.S2  dest=dssinet.SW2
SENT: 104to203 src=dssinet.SW2 dest=dssinet.R2
SENT: 104to203 src=dssinet.R2  dest=dssinet.R1
SENT: 104to203 src=dssinet.R1  dest=dssinet.SW1
SENT: 104to203 src=dssinet.SW1 dest=dssinet.U1

SENT: 104to203 src=dssinet.U1  dest=dssinet.SW1
SENT: 104to203 src=dssinet.SW1 dest=dssinet.R1
SENT: 104to203 src=dssinet.R1  dest=dssinet.R2
SENT: 104to203 src=dssinet.R2  dest=dssinet.SW2
SENT: 104to203 src=dssinet.SW2 dest=dssinet.S2
SENT: 104to203 src=dssinet.S2  dest=dssinet.SW2
SENT: 104to203 src=dssinet.SW2 dest=dssinet.R2
SENT: 104to203 src=dssinet.R2  dest=dssinet.R1
SENT: 104to203 src=dssinet.R1  dest=dssinet.SW1
SENT: 104to203 src=dssinet.SW1 dest=dssinet.U1

SENT: 104to203 src=dssinet.U1  dest=dssinet.SW1
SENT: 104to203 src=dssinet.SW1 dest=dssinet.R1
SENT: 104to203 src=dssinet.R1  dest=dssinet.R2
SENT: 104to203 src=dssinet.R2  dest=dssinet.SW2
SENT: 104to203 src=dssinet.SW2 dest=dssinet.S2
SENT: 104to203 src=dssinet.S2  dest=dssinet.SW2
SENT: 104to203 src=dssinet.SW2 dest=dssinet.R2
SENT: 104to203 src=dssinet.R2  dest=dssinet.R1
SENT: 104to203 src=dssinet.R1  dest=dssinet.SW1
SENT: 104to203 src=dssinet.SW1 dest=dssinet.U1
```

C.10.2 Listing of Module_output_home.txt

```
% Module_output_home.txt
% Comment lines added start with '%'
% Service query to SLS1:
From U server/start/stop/src/dest    = 103/104/101/104/101/
From SW server/start/stop/src/dest   = 103/104/101/104/101/
From SLS server/start/stop/src/dest/ = 103/101/104/104/101/
From SW server/start/stop/src/dest   = 103/101/104/104/101/

% Authentication and S1 Service information:
From U server/start/stop/src/dest    = 103/104/102/104/102/
From SW server/start/stop/src/dest   = 103/104/102/104/102/
From AS server/start/stop/src/dest   = 103/102/103/104/102/
From SW server/start/stop/src/dest   = 103/102/103/104/102/
From S server/start/stop/src/dest    = 103/103/102/104/102/
From SW server/start/stop/src/dest   = 103/103/102/104/102/
From AS server/start/stop/src/dest   = 103/102/104/104/102/
From SW server/start/stop/src/dest   = 103/102/104/104/102/

% Three rounds of S1 Service access:
From U server/start/stop/src/dest    = 103/104/103/104/103/
From SW server/start/stop/src/dest   = 103/104/103/104/103/
From S server/start/stop/src/dest    = 103/103/104/104/103/
From SW server/start/stop/src/dest   = 103/103/104/104/103/

From U server/start/stop/src/dest    = 103/104/103/104/103/
From SW server/start/stop/src/dest   = 103/104/103/104/103/
From S server/start/stop/src/dest    = 103/103/104/104/103/
From SW server/start/stop/src/dest   = 103/103/104/104/103/

From U server/start/stop/src/dest    = 103/104/103/104/103/
From SW server/start/stop/src/dest   = 103/104/103/104/103/
From S server/start/stop/src/dest    = 103/103/104/104/103/
```

```
From SW server/start/stop/src/dest   = 103/103/104/104/103/

% Service query to SLS1:
From U server/start/stop/src/dest    = 403/104/101/104/101/
From SW server/start/stop/src/dest   = 403/104/101/104/101/
From SLS server/start/stop/src/dest/ = 403/101/104/104/101/
From SW server/start/stop/src/dest   = 403/101/104/104/101/

% Authentication and S4 Service information:
From U server/start/stop/src/dest    = 403/104/102/104/102/
From SW server/start/stop/src/dest   = 403/104/102/104/102/
From AS server/start/stop/src/dest   = 403/102/402/104/102/
From SW server/start/stop/src/dest   = 403/102/402/104/102/
From R server/start/stop/src/dest    = 403/102/402/104/102/
From R server/start/stop/src/dest    = 403/102/402/104/102/
From SW server/start/stop/src/dest   = 403/102/402/104/102/
From AS server/start/stop/src/dest   = 403/402/403/104/102/
From SW server/start/stop/src/dest   = 403/402/403/104/102/
From S server/start/stop/src/dest    = 403/403/402/104/102/
From SW server/start/stop/src/dest   = 403/403/402/104/102/
From AS server/start/stop/src/dest   = 403/402/102/104/102/
From SW server/start/stop/src/dest   = 403/402/102/104/102/
From R server/start/stop/src/dest    = 403/402/102/104/102/
From R server/start/stop/src/dest    = 403/402/102/104/102/
From SW server/start/stop/src/dest   = 403/402/102/104/102/
From AS server/start/stop/src/dest   = 403/102/104/104/102/
From SW server/start/stop/src/dest   = 403/102/104/104/102/

% Three rounds of S4 Service access:
From U server/start/stop/src/dest    = 403/104/403/104/403/
From SW server/start/stop/src/dest   = 403/104/403/104/403/
From R server/start/stop/src/dest    = 403/104/403/104/403/
From R server/start/stop/src/dest    = 403/104/403/104/403/
From SW server/start/stop/src/dest   = 403/104/403/104/403/
From S server/start/stop/src/dest    = 403/403/104/104/403/
From SW server/start/stop/src/dest   = 403/403/104/104/403/
From R server/start/stop/src/dest    = 403/403/104/104/403/
From R server/start/stop/src/dest    = 403/403/104/104/403/
From SW server/start/stop/src/dest   = 403/403/104/104/403/

From U server/start/stop/src/dest    = 403/104/403/104/403/
From SW server/start/stop/src/dest   = 403/104/403/104/403/
From R server/start/stop/src/dest    = 403/104/403/104/403/
From R server/start/stop/src/dest    = 403/104/403/104/403/
From SW server/start/stop/src/dest   = 403/104/403/104/403/
```

```
From S server/start/stop/src/dest    = 403/403/104/104/403/
From SW server/start/stop/src/dest   = 403/403/104/104/403/
From R server/start/stop/src/dest    = 403/403/104/104/403/
From R server/start/stop/src/dest    = 403/403/104/104/403/
From SW server/start/stop/src/dest   = 403/403/104/104/403/

From U server/start/stop/src/dest    = 403/104/403/104/403/
From SW server/start/stop/src/dest   = 403/104/403/104/403/
From R server/start/stop/src/dest    = 403/104/403/104/403/
From R server/start/stop/src/dest    = 403/104/403/104/403/
From SW server/start/stop/src/dest   = 403/104/403/104/403/
From S server/start/stop/src/dest    = 403/403/104/104/403/
From SW server/start/stop/src/dest   = 403/403/104/104/403/
From R server/start/stop/src/dest    = 403/403/104/104/403/
From R server/start/stop/src/dest    = 403/403/104/104/403/
From SW server/start/stop/src/dest   = 403/403/104/104/403/

% Service query to SLS1:
From U server/start/stop/src/dest    = 203/104/101/104/101/
From SW server/start/stop/src/dest   = 203/104/101/104/101/
From SLS server/start/stop/src/dest/ = 203/101/104/104/101/
From SW server/start/stop/src/dest   = 203/101/104/104/101/

% Authentication and S2 Service information:
From U server/start/stop/src/dest    = 203/104/102/104/102/
From SW server/start/stop/src/dest   = 203/104/102/104/102/
From AS server/start/stop/src/dest   = 203/102/402/104/102/
From SW server/start/stop/src/dest   = 203/102/402/104/102/
From R server/start/stop/src/dest    = 203/102/402/104/102/
From R server/start/stop/src/dest    = 203/102/402/104/102/
From SW server/start/stop/src/dest   = 203/102/402/104/102/
From AS server/start/stop/src/dest   = 203/402/202/104/102/
From SW server/start/stop/src/dest   = 203/402/202/104/102/
From R server/start/stop/src/dest    = 203/402/202/104/102/
From R server/start/stop/src/dest    = 203/402/202/104/102/
From SW server/start/stop/src/dest   = 203/402/202/104/102/
From AS server/start/stop/src/dest   = 203/202/203/104/102/
From SW server/start/stop/src/dest   = 203/202/203/104/102/
From S server/start/stop/src/dest    = 203/203/202/104/102/
From SW server/start/stop/src/dest   = 203/203/202/104/102/
From AS server/start/stop/src/dest   = 203/202/402/104/102/
From SW server/start/stop/src/dest   = 203/202/402/104/102/
From R server/start/stop/src/dest    = 203/202/402/104/102/
From R server/start/stop/src/dest    = 203/202/402/104/102/
From SW server/start/stop/src/dest   = 203/202/402/104/102/
```

```
From AS server/start/stop/src/dest   = 203/402/102/104/102/
From SW server/start/stop/src/dest   = 203/402/102/104/102/
From R server/start/stop/src/dest    = 203/402/102/104/102/
From R server/start/stop/src/dest    = 203/402/102/104/102/
From SW server/start/stop/src/dest   = 203/402/102/104/102/
From AS server/start/stop/src/dest   = 203/102/104/104/102/
From SW server/start/stop/src/dest   = 203/102/104/104/102/

% Three rounds of S2 Service access:
From U server/start/stop/src/dest    = 203/104/203/104/203/
From SW server/start/stop/src/dest   = 203/104/203/104/203/
From R server/start/stop/src/dest    = 203/104/203/104/203/
From R server/start/stop/src/dest    = 203/104/203/104/203/
From SW server/start/stop/src/dest   = 203/104/203/104/203/
From S server/start/stop/src/dest    = 203/203/104/104/203/
From SW server/start/stop/src/dest   = 203/203/104/104/203/
From R server/start/stop/src/dest    = 203/203/104/104/203/
From R server/start/stop/src/dest    = 203/203/104/104/203/
From SW server/start/stop/src/dest   = 203/203/104/104/203/

From U server/start/stop/src/dest    = 203/104/203/104/203/
From SW server/start/stop/src/dest   = 203/104/203/104/203/
From R server/start/stop/src/dest    = 203/104/203/104/203/
From R server/start/stop/src/dest    = 203/104/203/104/203/
From SW server/start/stop/src/dest   = 203/104/203/104/203/
From S server/start/stop/src/dest    = 203/203/104/104/203/
From SW server/start/stop/src/dest   = 203/203/104/104/203/
From R server/start/stop/src/dest    = 203/203/104/104/203/
From R server/start/stop/src/dest    = 203/203/104/104/203/
From SW server/start/stop/src/dest   = 203/203/104/104/203/

From U server/start/stop/src/dest    = 203/104/203/104/203/
From SW server/start/stop/src/dest   = 203/104/203/104/203/
From R server/start/stop/src/dest    = 203/104/203/104/203/
From R server/start/stop/src/dest    = 203/104/203/104/203/
From SW server/start/stop/src/dest   = 203/104/203/104/203/
From S server/start/stop/src/dest    = 203/203/104/104/203/
From SW server/start/stop/src/dest   = 203/203/104/104/203/
From R server/start/stop/src/dest    = 203/203/104/104/203/
From R server/start/stop/src/dest    = 203/203/104/104/203/
From SW server/start/stop/src/dest   = 203/203/104/104/203/
```

C.10.3 Listing of Message_sent_foreign.txt

```
% Message_sent_foreign.txt
% Comment lines added start with '%'
% Identify user to AS3
SENT: 304to302 src=dssinet.U1 dest=dssinet.SW3
SENT: 304to302 src=dssinet.SW3  dest=dssinet.AS3

% AS3 use another message to get Service List from SLS1
SENT: 304to302 src=dssinet.AS3  dest=dssinet.SW3
SENT: 304to302 src=dssinet.SW3  dest=dssinet.R3
SENT: 304to302 src=dssinet.R3   dest=dssinet.R1
SENT: 304to302 src=dssinet.R1   dest=dssinet.SW1
SENT: 304to302 src=dssinet.SW1, dest=dssinet.SLS1
SENT: 304to302 src=dssinet.SLS1 dest=dssinet.SW1
SENT: 304to302 src=dssinet.SW1  dest=dssinet.R1
SENT: 304to302 src=dssinet.R1   dest=dssinet.R3
SENT: 304to302 src=dssinet.R3   dest=dssinet.SW3
SENT: 304to302 src=dssinet.SW3  dest=dssinet.AS3
SENT: 304to302 src=dssinet.AS3  dest=dssinet.SW3

% AS3 uses the original message to reply U1
SENT: 304to302 src=dssinet.SW3  dest=dssinet.U1

% Service query to SLS3:
SENT: 304to301 src=dssinet.U1   dest=dssinet.SW3
SENT: 304to301 src=dssinet.SW3  dest=dssinet.SLS3
SENT: 304to301 src=dssinet.SLS3 dest=dssinet.SW3
SENT: 304to301 src=dssinet.SW3  dest=dssinet.U1

% Authentication and S1 Service information:
SENT: 304to302 src=dssinet.U1   dest=dssinet.SW3
SENT: 304to302 src=dssinet.SW3  dest=dssinet.AS3
SENT: 304to302 src=dssinet.AS3  dest=dssinet.SW3
SENT: 304to302 src=dssinet.SW3  dest=dssinet.R3
SENT: 304to302 src=dssinet.R3   dest=dssinet.R1
SENT: 304to302 src=dssinet.R1   dest=dssinet.SW1
SENT: 304to302 src=dssinet.SW1  dest=dssinet.AS1
SENT: 304to302 src=dssinet.AS1  dest=dssinet.SW1
SENT: 304to302 src=dssinet.SW1  dest=dssinet.S1
SENT: 304to302 src=dssinet.S1   dest=dssinet.SW1
SENT: 304to302 src=dssinet.SW1  dest=dssinet.AS1
SENT: 304to302 src=dssinet.AS1  dest=dssinet.SW1
SENT: 304to302 src=dssinet.SW1  dest=dssinet.R1
SENT: 304to302 src=dssinet.R1   dest=dssinet.R3
SENT: 304to302 src=dssinet.R3   dest=dssinet.SW3
SENT: 304to302 src=dssinet.SW3  dest=dssinet.AS3
SENT: 304to302 src=dssinet.AS3  dest=dssinet.SW3
```

```
SENT: 304to302 src=dssinet.SW3  dest=dssinet.U1

% Three rounds of S1 Service access:
SENT: 304to103 src=dssinet.U1   dest=dssinet.SW3
SENT: 304to103 src=dssinet.SW3  dest=dssinet.R3
SENT: 304to103 src=dssinet.R3   dest=dssinet.R1
SENT: 304to103 src=dssinet.R1   dest=dssinet.SW1
SENT: 304to103 src=dssinet.SW1  dest=dssinet.S1
SENT: 304to103 src=dssinet.S1   dest=dssinet.SW1
SENT: 304to103 src=dssinet.SW1  dest=dssinet.R1
SENT: 304to103 src=dssinet.R1   dest=dssinet.R3
SENT: 304to103 src=dssinet.R3   dest=dssinet.SW3
SENT: 304to103 src=dssinet.SW3  dest=dssinet.U1

SENT: 304to103 src=dssinet.U1   dest=dssinet.SW3
SENT: 304to103 src=dssinet.SW3  dest=dssinet.R3
SENT: 304to103 src=dssinet.R3   dest=dssinet.R1
SENT: 304to103 src=dssinet.R1   dest=dssinet.SW1
SENT: 304to103 src=dssinet.SW1  dest=dssinet.S1
SENT: 304to103 src=dssinet.S1   dest=dssinet.SW1
SENT: 304to103 src=dssinet.SW1  dest=dssinet.R1
SENT: 304to103 src=dssinet.R1   dest=dssinet.R3
SENT: 304to103 src=dssinet.R3   dest=dssinet.SW3
SENT: 304to103 src=dssinet.SW3  dest=dssinet.U1

SENT: 304to103 src=dssinet.U1   dest=dssinet.SW3
SENT: 304to103 src=dssinet.SW3  dest=dssinet.R3
SENT: 304to103 src=dssinet.R3   dest=dssinet.R1
SENT: 304to103 src=dssinet.R1   dest=dssinet.SW1
SENT: 304to103 src=dssinet.SW1  dest=dssinet.S1
SENT: 304to103 src=dssinet.S1   dest=dssinet.SW1
SENT: 304to103 src=dssinet.SW1  dest=dssinet.R1
SENT: 304to103 src=dssinet.R1   dest=dssinet.R3
SENT: 304to103 src=dssinet.R3   dest=dssinet.SW3
SENT: 304to103 src=dssinet.SW3  dest=dssinet.U1

% Service query to SLS3:
SENT: 304to301 src=dssinet.U1   dest=dssinet.SW3
SENT: 304to301 src=dssinet.SW3  dest=dssinet.SLS3
SENT: 304to301 src=dssinet.SLS3 dest=dssinet.SW3
SENT: 304to301 src=dssinet.SW3  dest=dssinet.U1

% Authentication and S4 Service information:
SENT: 304to302 src=dssinet.U1   dest=dssinet.SW3
SENT: 304to302 src=dssinet.SW3  dest=dssinet.AS3
SENT: 304to302 src=dssinet.AS3  dest=dssinet.SW3
SENT: 304to302 src=dssinet.SW3  dest=dssinet.R3
SENT: 304to302 src=dssinet.R3   dest=dssinet.R1
SENT: 304to302 src=dssinet.R1   dest=dssinet.SW1
SENT: 304to302 src=dssinet.SW1  dest=dssinet.AS1
SENT: 304to302 src=dssinet.AS1  dest=dssinet.SW1
SENT: 304to302 src=dssinet.SW1  dest=dssinet.R1
SENT: 304to302 src=dssinet.R1   dest=dssinet.R4
SENT: 304to302 src=dssinet.R4   dest=dssinet.SW4
SENT: 304to302 src=dssinet.SW4  dest=dssinet.AS4
SENT: 304to302 src=dssinet.AS4  dest=dssinet.SW4
SENT: 304to302 src=dssinet.SW4  dest=dssinet.S4
SENT: 304to302 src=dssinet.S4   dest=dssinet.SW4
SENT: 304to302 src=dssinet.SW4  dest=dssinet.AS4
SENT: 304to302 src=dssinet.AS4  dest=dssinet.SW4
SENT: 304to302 src=dssinet.SW4  dest=dssinet.R4
SENT: 304to302 src=dssinet.R4   dest=dssinet.R1
SENT: 304to302 src=dssinet.R1   dest=dssinet.SW1
SENT: 304to302 src=dssinet.SW1  dest=dssinet.AS1
SENT: 304to302 src=dssinet.AS1  dest=dssinet.SW1
SENT: 304to302 src=dssinet.SW1  dest=dssinet.R1
SENT: 304to302 src=dssinet.R1   dest=dssinet.R3
SENT: 304to302 src=dssinet.R3   dest=dssinet.SW3
SENT: 304to302 src=dssinet.SW3  dest=dssinet.AS3
SENT: 304to302 src=dssinet.AS3  dest=dssinet.SW3
SENT: 304to302 src=dssinet.SW3  dest=dssinet.U1

% Three rounds of S4 Service access:
SENT: 304to403 src=dssinet.U1   dest=dssinet.SW3
SENT: 304to403 src=dssinet.SW3  dest=dssinet.R3
SENT: 304to403 src=dssinet.SW4  dest=dssinet.S4
SENT: 304to403 src=dssinet.R3   dest=dssinet.R4
SENT: 304to403 src=dssinet.R4   dest=dssinet.SW4
SENT: 304to403 src=dssinet.S4   dest=dssinet.SW4
SENT: 304to403 src=dssinet.SW4  dest=dssinet.R4
SENT: 304to403 src=dssinet.R4   dest=dssinet.R3
SENT: 304to403 src=dssinet.R3   dest=dssinet.SW3
SENT: 304to403 src=dssinet.SW3  dest=dssinet.U1

SENT: 304to403 src=dssinet.U1   dest=dssinet.SW3
SENT: 304to403 src=dssinet.SW3  dest=dssinet.R3
SENT: 304to403 src=dssinet.R3   dest=dssinet.R4
SENT: 304to403 src=dssinet.R4   dest=dssinet.SW4
SENT: 304to403 src=dssinet.SW4  dest=dssinet.S4
SENT: 304to403 src=dssinet.S4   dest=dssinet.SW4
```

```
SENT: 304to403 src=dssinet.SW4  dest=dssinet.R4
SENT: 304to403 src=dssinet.R4   dest=dssinet.R3
SENT: 304to403 src=dssinet.R3   dest=dssinet.SW3
SENT: 304to403 src=dssinet.SW3  dest=dssinet.U1

SENT: 304to403 src=dssinet.U1   dest=dssinet.SW3
SENT: 304to403 src=dssinet.SW3  dest=dssinet.R3
SENT: 304to403 src=dssinet.R3   dest=dssinet.R4
SENT: 304to403 src=dssinet.R4   dest=dssinet.SW4
SENT: 304to403 src=dssinet.SW4  dest=dssinet.S4
SENT: 304to403 src=dssinet.S4   dest=dssinet.SW4
SENT: 304to403 src=dssinet.SW4  dest=dssinet.R4
SENT: 304to403 src=dssinet.R4   dest=dssinet.R3
SENT: 304to403 src=dssinet.R3   dest=dssinet.SW3
SENT: 304to403 src=dssinet.SW3  dest=dssinet.U1

% Service query to SLS3:
SENT: 304to301 src=dssinet.U1   dest=dssinet.SW3
SENT: 304to301 src=dssinet.SW3  dest=dssinet.SLS3
SENT: 304to301 src=dssinet.SLS3 dest=dssinet.SW3
SENT: 304to301 src=dssinet.SW3  dest=dssinet.U1

% Authentication and S2 Service information:
SENT: 304to302 src=dssinet.U1   dest=dssinet.SW3
SENT: 304to302 src=dssinet.SW3  dest=dssinet.AS3
SENT: 304to302 src=dssinet.AS3  dest=dssinet.SW3
SENT: 304to302 src=dssinet.SW3  dest=dssinet.R3
SENT: 304to302 src=dssinet.R3   dest=dssinet.R1
SENT: 304to302 src=dssinet.R1   dest=dssinet.SW1
SENT: 304to302 src=dssinet.SW1  dest=dssinet.AS1
SENT: 304to302 src=dssinet.AS1  dest=dssinet.SW1
SENT: 304to302 src=dssinet.SW1  dest=dssinet.R1
SENT: 304to302 src=dssinet.R1   dest=dssinet.R4
SENT: 304to302 src=dssinet.R4   dest=dssinet.SW4
SENT: 304to302 src=dssinet.SW4  dest=dssinet.AS4
SENT: 304to302 src=dssinet.AS4  dest=dssinet.SW4
SENT: 304to302 src=dssinet.SW4  dest=dssinet.R4
SENT: 304to302 src=dssinet.R4   dest=dssinet.R2
SENT: 304to302 src=dssinet.R2   dest=dssinet.SW2
SENT: 304to302 src=dssinet.SW2  dest=dssinet.AS2
SENT: 304to302 src=dssinet.AS2  dest=dssinet.SW2
SENT: 304to302 src=dssinet.SW2  dest=dssinet.S2
SENT: 304to302 src=dssinet.S2   dest=dssinet.SW2
SENT: 304to302 src=dssinet.SW2  dest=dssinet.AS2
SENT: 304to302 src=dssinet.AS2  dest=dssinet.SW2
SENT: 304to302 src=dssinet.SW2  dest=dssinet.R2
SENT: 304to302 src=dssinet.R2   dest=dssinet.R4
SENT: 304to302 src=dssinet.R4   dest=dssinet.SW4
SENT: 304to302 src=dssinet.SW4  dest=dssinet.AS4
SENT: 304to302 src=dssinet.AS4  dest=dssinet.SW4
SENT: 304to302 src=dssinet.SW4  dest=dssinet.R4
SENT: 304to302 src=dssinet.R4   dest=dssinet.R1
SENT: 304to302 src=dssinet.R1   dest=dssinet.SW1
SENT: 304to302 src=dssinet.SW1  dest=dssinet.AS1
SENT: 304to302 src=dssinet.AS1  dest=dssinet.SW1
SENT: 304to302 src=dssinet.SW1  dest=dssinet.R1
SENT: 304to302 src=dssinet.R1   dest=dssinet.R3
SENT: 304to302 src=dssinet.R3   dest=dssinet.SW3
SENT: 304to302 src=dssinet.SW3  dest=dssinet.AS3
SENT: 304to302 src=dssinet.AS3  dest=dssinet.SW3
SENT: 304to302 src=dssinet.SW3  dest=dssinet.U1

% Three rounds of S2 Service access:
SENT: 304to203 src=dssinet.U1   dest=dssinet.SW3
SENT: 304to203 src=dssinet.SW3  dest=dssinet.R3
SENT: 304to203 src=dssinet.R3   dest=dssinet.R2
SENT: 304to203 src=dssinet.R2   dest=dssinet.SW2
SENT: 304to203 src=dssinet.SW2  dest=dssinet.S2
SENT: 304to203 src=dssinet.S2   dest=dssinet.SW2
SENT: 304to203 src=dssinet.SW2  dest=dssinet.R2
SENT: 304to203 src=dssinet.R2   dest=dssinet.R3
SENT: 304to203 src=dssinet.R3   dest=dssinet.SW3
SENT: 304to203 src=dssinet.SW3  dest=dssinet.U1

SENT: 304to203 src=dssinet.U1   dest=dssinet.SW3
SENT: 304to203 src=dssinet.SW3  dest=dssinet.R3
SENT: 304to203 src=dssinet.R3   dest=dssinet.R2
SENT: 304to203 src=dssinet.R2   dest=dssinet.SW2
SENT: 304to203 src=dssinet.SW2  dest=dssinet.S2
SENT: 304to203 src=dssinet.S2   dest=dssinet.SW2
SENT: 304to203 src=dssinet.SW2  dest=dssinet.R2
SENT: 304to203 src=dssinet.R2   dest=dssinet.R3
SENT: 304to203 src=dssinet.R3   dest=dssinet.SW3
SENT: 304to203 src=dssinet.SW3  dest=dssinet.U1

SENT: 304to203 src=dssinet.U1   dest=dssinet.SW3
SENT: 304to203 src=dssinet.SW3  dest=dssinet.R3
SENT: 304to203 src=dssinet.R3   dest=dssinet.R2
SENT: 304to203 src=dssinet.R2   dest=dssinet.SW2
SENT: 304to203 src=dssinet.SW2  dest=dssinet.S2
```

```
SENT: 304to203 src=dssinet.S2   dest=dssinet.SW2
SENT: 304to203 src=dssinet.SW2  dest=dssinet.R2
SENT: 304to203 src=dssinet.R2   dest=dssinet.R3
SENT: 304to203 src=dssinet.R3   dest=dssinet.SW3
SENT: 304to203 src=dssinet.SW3  dest=dssinet.U1
```

C.10.4 Listing of Module_output_foreign.txt

```
% Module_output_foreign.txt
% Comment lines added start with '%'
% Identify user to AS3 which retrieves the Service List from SLS1
From U server/start/stop/src/dest    = 303/304/302/304/302/
From SW server/start/stop/src/dest   = 303/304/302/304/302/

% AS3 use another message to get Service List from SLS1
From AS server/start/stop/src/dest   = 303/302/101/302/101/
From SW server/start/stop/src/dest   = 303/302/101/302/101/
From R server/start/stop/src/dest    = 303/302/101/302/101/
From R server/start/stop/src/dest    = 303/302/101/302/101/
From SW server/start/stop/src/dest   = 303/302/101/302/101/
From SLS server/start/stop/src/dest/ = 303/101/302/302/101/
From SW server/start/stop/src/dest   = 303/101/302/302/101/
From R server/start/stop/src/dest    = 303/101/302/302/101/
From R server/start/stop/src/dest    = 303/101/302/302/101/
From SW server/start/stop/src/dest   = 303/101/302/302/101/
From As server/start/stop/src/dest   = 303/101/302/302/101/

% AS3 uses the original message to reply U1
From SW server/start/stop/src/dest   = 303/302/304/302/304/

% Service query to SLS3:
From U server/start/stop/src/dest    = 103/304/301/304/301/
From SW server/start/stop/src/dest   = 103/304/301/304/301/
From SLS server/start/stop/src/dest/ = 103/301/304/304/301/
From SW server/start/stop/src/dest   = 103/301/304/304/301/

% Authentication and S1 Service information:
From U server/start/stop/src/dest    = 103/304/302/304/302/
From SW server/start/stop/src/dest   = 103/304/302/304/302/
From AS server/start/stop/src/dest   = 103/302/102/304/302/
From SW server/start/stop/src/dest   = 103/302/102/304/302/
From R server/start/stop/src/dest    = 103/302/102/304/302/
From R server/start/stop/src/dest    = 103/302/102/304/302/
From SW server/start/stop/src/dest   = 103/302/102/304/302/
From AS server/start/stop/src/dest   = 103/102/103/304/302/
From SW server/start/stop/src/dest   = 103/102/103/304/302/
From S server/start/stop/src/dest    = 103/103/102/304/302/
From SW server/start/stop/src/dest   = 103/103/102/304/302/
From AS server/start/stop/src/dest   = 103/102/302/304/302/
From SW server/start/stop/src/dest   = 103/102/302/304/302/
From R server/start/stop/src/dest    = 103/102/302/304/302/
From R server/start/stop/src/dest    = 103/102/302/304/302/
From SW server/start/stop/src/dest   = 103/102/302/304/302/
From AS server/start/stop/src/dest   = 103/302/304/304/302/
From SW server/start/stop/src/dest   = 103/302/304/304/302/

% Three rounds of S1 Service access:
From U server/start/stop/src/dest    = 103/304/103/304/103/
From SW server/start/stop/src/dest   = 103/304/103/304/103/
From R server/start/stop/src/dest    = 103/304/103/304/103/
From R server/start/stop/src/dest    = 103/304/103/304/103/
From SW server/start/stop/src/dest   = 103/304/103/304/103/
From S server/start/stop/src/dest    = 103/103/304/304/103/
From SW server/start/stop/src/dest   = 103/103/304/304/103/
From R server/start/stop/src/dest    = 103/103/304/304/103/
From R server/start/stop/src/dest    = 103/103/304/304/103/
From SW server/start/stop/src/dest   = 103/103/304/304/103/

From U server/start/stop/src/dest    = 103/304/103/304/103/
From SW server/start/stop/src/dest   = 103/304/103/304/103/
From R server/start/stop/src/dest    = 103/304/103/304/103/
From R server/start/stop/src/dest    = 103/304/103/304/103/
From SW server/start/stop/src/dest   = 103/304/103/304/103/
From S server/start/stop/src/dest    = 103/103/304/304/103/
From SW server/start/stop/src/dest   = 103/103/304/304/103/
From R server/start/stop/src/dest    = 103/103/304/304/103/
From R server/start/stop/src/dest    = 103/103/304/304/103/
From SW server/start/stop/src/dest   = 103/103/304/304/103/

From U server/start/stop/src/dest    = 103/304/103/304/103/
From SW server/start/stop/src/dest   = 103/304/103/304/103/
From R server/start/stop/src/dest    = 103/304/103/304/103/
From R server/start/stop/src/dest    = 103/304/103/304/103/
From SW server/start/stop/src/dest   = 103/304/103/304/103/
From S server/start/stop/src/dest    = 103/103/304/304/103/
From SW server/start/stop/src/dest   = 103/103/304/304/103/
From R server/start/stop/src/dest    = 103/103/304/304/103/
From R server/start/stop/src/dest    = 103/103/304/304/103/
From SW server/start/stop/src/dest   = 103/103/304/304/103/
```

```
% Service query to SLS3:
From U server/start/stop/src/dest    = 403/304/301/304/301/
From SW server/start/stop/src/dest   = 403/304/301/304/301/
From SLS server/start/stop/src/dest/ = 403/301/304/304/301/
From SW server/start/stop/src/dest   = 403/301/304/304/301/

% Authentication and S4 Service information:
From U server/start/stop/src/dest    = 403/304/302/304/302/
From SW server/start/stop/src/dest   = 403/304/302/304/302/
From AS server/start/stop/src/dest   = 403/302/102/304/302/
From SW server/start/stop/src/dest   = 403/302/102/304/302/
From R server/start/stop/src/dest    = 403/302/102/304/302/
From R server/start/stop/src/dest    = 403/302/102/304/302/
From SW server/start/stop/src/dest   = 403/302/102/304/302/
From AS server/start/stop/src/dest   = 403/102/402/304/302/
From SW server/start/stop/src/dest   = 403/102/402/304/302/
From R server/start/stop/src/dest    = 403/102/402/304/302/
From R server/start/stop/src/dest    = 403/102/402/304/302/
From SW server/start/stop/src/dest   = 403/102/402/304/302/
From AS server/start/stop/src/dest   = 403/402/403/304/302/
From SW server/start/stop/src/dest   = 403/402/403/304/302/
From S server/start/stop/src/dest    = 403/403/402/304/302/
From SW server/start/stop/src/dest   = 403/403/402/304/302/
From AS server/start/stop/src/dest   = 403/402/102/304/302/
From SW server/start/stop/src/dest   = 403/402/102/304/302/
From R server/start/stop/src/dest    = 403/402/102/304/302/
From R server/start/stop/src/dest    = 403/402/102/304/302/
From SW server/start/stop/src/dest   = 403/402/102/304/302/
From AS server/start/stop/src/dest   = 403/102/302/304/302/
From SW server/start/stop/src/dest   = 403/102/302/304/302/
From R server/start/stop/src/dest    = 403/102/302/304/302/
From R server/start/stop/src/dest    = 403/102/302/304/302/
From SW server/start/stop/src/dest   = 403/102/302/304/302/
From AS server/start/stop/src/dest   = 403/302/304/304/302/
From SW server/start/stop/src/dest   = 403/302/304/304/302/

% Three rounds of S4 Service access:
From U server/start/stop/src/dest    = 403/304/403/304/403/
From SW server/start/stop/src/dest   = 403/304/403/304/403/
From R server/start/stop/src/dest    = 403/304/403/304/403/
From R server/start/stop/src/dest    = 403/304/403/304/403/
From SW server/start/stop/src/dest   = 403/304/403/304/403/
From S server/start/stop/src/dest    = 403/403/304/304/403/
From SW server/start/stop/src/dest   = 403/403/304/304/403/
From R server/start/stop/src/dest    = 403/403/304/304/403/
From R server/start/stop/src/dest    = 403/403/304/304/403/
From SW server/start/stop/src/dest   = 403/403/304/304/403/

From U server/start/stop/src/dest    = 403/304/403/304/403/
From SW server/start/stop/src/dest   = 403/304/403/304/403/
From R server/start/stop/src/dest    = 403/304/403/304/403/
From R server/start/stop/src/dest    = 403/304/403/304/403/
From SW server/start/stop/src/dest   = 403/304/403/304/403/
From S server/start/stop/src/dest    = 403/403/304/304/403/
From SW server/start/stop/src/dest   = 403/403/304/304/403/
From R server/start/stop/src/dest    = 403/403/304/304/403/
From R server/start/stop/src/dest    = 403/403/304/304/403/
From SW server/start/stop/src/dest   = 403/403/304/304/403/


From U server/start/stop/src/dest    = 403/304/403/304/403/
From SW server/start/stop/src/dest   = 403/304/403/304/403/
From R server/start/stop/src/dest    = 403/304/403/304/403/
From R server/start/stop/src/dest    = 403/304/403/304/403/
From SW server/start/stop/src/dest   = 403/304/403/304/403/
From S server/start/stop/src/dest    = 403/403/304/304/403/
From SW server/start/stop/src/dest   = 403/403/304/304/403/
From R server/start/stop/src/dest    = 403/403/304/304/403/
From R server/start/stop/src/dest    = 403/403/304/304/403/
From SW server/start/stop/src/dest   = 403/403/304/304/403/

% Service query to SLS3:
From U server/start/stop/src/dest    = 203/304/301/304/301/
From SW server/start/stop/src/dest   = 203/304/301/304/301/
From SLS server/start/stop/src/dest/ = 203/301/304/304/301/
From SW server/start/stop/src/dest   = 203/301/304/304/301/

% Authentication and S2 Service information:
From U server/start/stop/src/dest    = 203/304/302/304/302/
From SW server/start/stop/src/dest   = 203/304/302/304/302/
From AS server/start/stop/src/dest   = 203/302/102/304/302/
From SW server/start/stop/src/dest   = 203/302/102/304/302/
From R server/start/stop/src/dest    = 203/302/102/304/302/
From R server/start/stop/src/dest    = 203/302/102/304/302/
From SW server/start/stop/src/dest   = 203/302/102/304/302/
From AS server/start/stop/src/dest   = 203/102/402/304/302/
From SW server/start/stop/src/dest   = 203/102/402/304/302/
From R server/start/stop/src/dest    = 203/102/402/304/302/
From R server/start/stop/src/dest    = 203/102/402/304/302/
From SW server/start/stop/src/dest   = 203/102/402/304/302/
```

```
From AS server/start/stop/src/dest   = 203/402/202/304/302/
From SW server/start/stop/src/dest   = 203/402/202/304/302/
From R server/start/stop/src/dest    = 203/402/202/304/302/
From R server/start/stop/src/dest    = 203/402/202/304/302/
From SW server/start/stop/src/dest   = 203/402/202/304/302/
From AS server/start/stop/src/dest   = 203/202/203/304/302/
From SW server/start/stop/src/dest   = 203/202/203/304/302/
From S server/start/stop/src/dest    = 203/203/202/304/302/
From SW server/start/stop/src/dest   = 203/203/202/304/302/
From AS server/start/stop/src/dest   = 203/202/402/304/302/
From SW server/start/stop/src/dest   = 203/202/402/304/302/
From R server/start/stop/src/dest    = 203/202/402/304/302/
From R server/start/stop/src/dest    = 203/202/402/304/302/
From SW server/start/stop/src/dest   = 203/202/402/304/302/
From AS server/start/stop/src/dest   = 203/402/102/304/302/
From SW server/start/stop/src/dest   = 203/402/102/304/302/
From R server/start/stop/src/dest    = 203/402/102/304/302/
From R server/start/stop/src/dest    = 203/402/102/304/302/
From SW server/start/stop/src/dest   = 203/402/102/304/302/
From AS server/start/stop/src/dest   = 203/102/302/304/302/
From SW server/start/stop/src/dest   = 203/102/302/304/302/
From R server/start/stop/src/dest    = 203/102/302/304/302/
From R server/start/stop/src/dest    = 203/102/302/304/302/
From SW server/start/stop/src/dest   = 203/102/302/304/302/
From AS server/start/stop/src/dest   = 203/302/304/304/302/
From SW server/start/stop/src/dest   = 203/302/304/304/302/

% Three rounds of S2 Service access:
From U server/start/stop/src/dest    = 203/304/203/304/203/
From SW server/start/stop/src/dest   = 203/304/203/304/203/
From R server/start/stop/src/dest    = 203/304/203/304/203/
From R server/start/stop/src/dest    = 203/304/203/304/203/
From SW server/start/stop/src/dest   = 203/304/203/304/203/
From S server/start/stop/src/dest    = 203/203/304/304/203/
From SW server/start/stop/src/dest   = 203/203/304/304/203/
From R server/start/stop/src/dest    = 203/203/304/304/203/
From R server/start/stop/src/dest    = 203/203/304/304/203/
From SW server/start/stop/src/dest   = 203/203/304/304/203/

From U server/start/stop/src/dest    = 203/304/203/304/203/
From SW server/start/stop/src/dest   = 203/304/203/304/203/
From R server/start/stop/src/dest    = 203/304/203/304/203/
From R server/start/stop/src/dest    = 203/304/203/304/203/
From SW server/start/stop/src/dest   = 203/304/203/304/203/
From S server/start/stop/src/dest    = 203/203/304/304/203/
From SW server/start/stop/src/dest   = 203/203/304/304/203/
From R server/start/stop/src/dest    = 203/203/304/304/203/
From R server/start/stop/src/dest    = 203/203/304/304/203/
From SW server/start/stop/src/dest   = 203/203/304/304/203/

From U server/start/stop/src/dest    = 203/304/203/304/203/
From SW server/start/stop/src/dest   = 203/304/203/304/203/
From R server/start/stop/src/dest    = 203/304/203/304/203/
From R server/start/stop/src/dest    = 203/304/203/304/203/
From SW server/start/stop/src/dest   = 203/304/203/304/203/
From S server/start/stop/src/dest    = 203/203/304/304/203/
From SW server/start/stop/src/dest   = 203/203/304/304/203/
From R server/start/stop/src/dest    = 203/203/304/304/203/
From R server/start/stop/src/dest    = 203/203/304/304/203/
From SW server/start/stop/src/dest   = 203/203/304/304/203/
```

Appendix D

D.1 asServer.cpp

```
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else
#include <sys/types.h>
#include <sys/socket.h>
#include sys/wait.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "security.h"
#include "md5ADT.h"
#include "bst.h"
#include "sp.h"
#include "path.h"

#define PROTOPORT    5500      /* default asServer port number */
#define QLEN         6         /* size of request queue        */

void sig_chld(int signo);

int main(int argc, char* argv[]){
    struct    hostent  *ptrh;  /* ptr to a host table entry     */
    struct    protoent *ptrp;  /* ptr to a protocol table entry */
    struct    sockaddr_in sad; /* struct for server's address   */
    struct    sockaddr_in cad; /* struct for client's address   */
    int       sd, sd2;         /* socket descriptors            */
    int       port;            /* protocol port number          */
    int       len;
    socklen_t alen;
    char      buf[1000];       /* buffer for str server sends   */
    char      msg_buf[1000];
    char      msg_buf2[1000];
    int       visit = 0;
    int       counter = 0;
    FILE      *fPtr;
    char      userID[100] ;
    char      userIDfile[100];
    char      keyIndex[100];
    int       flag = 0;
    int       n, i, pid, passLen, nameLen;
    int       service;
    char      location;
    char      serviceName[100];
    char      servicePath[100];
    char      server[100];
    char      serverPort[100];
    char      serverAdd[100];
    char      thisIP[100];
    char      R1[100];
    char      R2[100];
    char      R3[100];
    char      DP[100];
    char      SP[100];
    char      DPnew[100];
    char      HDP[100];
    char      HDPfile[100];
    char      HDPnew[100];
    char      HSP[100];
    char      HSPfile[100];
    char      msg_one[1000];
    char      msg_two[1000];
    char      msg_three[1000];
```

```
int       sKey;
char      combineR[300];
char      hashR[100];
char      asPort[100];
char      asServerAddfile[100];
char      homeASAdd[100];
char      nextIP[100];
char      term1[1000];
char      term2[1000];
char      answer[1000];
char      *host;
char      localhostname[10];
char      filename[20];

#ifdef WIN32
    WSADATA wsaData;
    WSAStartup(0x0101, &wsaData);
#endif

if (argc < 4){
   printf ("Usage: asServer localhost 5500 <eth0 or eth1>\n");
exit(1);
}
fd = socket(PF_INET, SOCK_STREAM, 0);

// For IP of asServer using ioctl
struct ifreq ifr;
struct sockaddr_in saddr;
int fd;
int partAll, part1, part2, part3, part4;
strcpy(ifr.ifr_name, argv[3]);
// "eth0" or "eth1" is the name of the interface to be checked.
// "lo" will give 127.0.0.1
ioctl(fd, SIOCGIFADDR, &ifr);
saddr = *((struct sockaddr_in *)(&(ifr.ifr_addr)));
part4 = saddr.sin_addr.s_addr % 256;
part3 = (saddr.sin_addr.s_addr / 256) %256;
part2 = (saddr.sin_addr.s_addr / 256 /256 ) %256;
part1 = (saddr.sin_addr.s_addr / 256 /256 /256);
sprintf (thisIP, "%d.%d.%d.%d", part4, part3, part2, part1);
// thisIP is the IP address of the asServer

memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr      */
sad.sin_family = AF_INET;           /* set to Internet     */
sad.sin_addr.s_addr = INADDR_ANY;   /* set local IP add    */

if (argc > 2){
    port = atoi(argv[2]);
} else {
    port = PROTOPORT;        /* use default port number  */
}
sad.sin_port = htons((u_short)port);

/* Map TCP transport protocol name to protocol number */
if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
    fprintf(stderr, "cannot map \"tcp\" to protocol number");
    exit(1);
}

/* Create a socket */
sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
if (sd < 0) {
    fprintf(stderr, "socket creation failed\n");
    exit(1);
}

/* Bind a local address to the socket */
if (bind(sd, (struct sockaddr *)&sad,
    (socklen_t)(sizeof(sad))) < 0) {
    fprintf(stderr,"bind failed\n");
    exit(1);
}

/* Specify size of request queue */
if (listen(sd, QLEN) < 0) {
    fprintf(stderr,"listen failed\n");
    exit(1);
}

signal(SIGCHLD, sig_chld);

/* Main server loop - accept and handle requests */
while (1) {
    alen = (socklen_t)(sizeof(cad));
    // We use INADDR_ANY and so socket can accept
    // connections from all interfaces
    if ((sd2=accept(sd, (struct sockaddr *)&cad, &alen))< 0){
        fprintf(stderr, "accept failed\n");
        exit(1);
    }
    visit++;
```

```
/* fork() to handle multiple connections */
if ((pid = fork()) == 0) {
    close(sd); // child not listening

    /* Receive request from client
       Service 1 is just for testing socket connections
       Service 2 is for service access
       Service 4 is asServer internal service code
    */
    msg_buf[0] = 0;
    while(n=read(sd2, buf, sizeof(buf))){
        decRoundRobin_len_all(buf, n, 15);
        buf[n] =0;
        strcat(msg_buf, buf);
        if (buf[n-1] == '#') {
            msg_buf[strlen(msg_buf)] = 0;
            break;
        }
    }
    service = msg_buf[0] - '0';
    printf ("msg = <%s>\n", msg_buf);

    /* Service 1 is just for testing socket connection
       Service 2 is the normal service request
                 moving in the backward
                 (towards the homeAS in SPath by default)
                 until the homeAS is reached and will
                 change to service 4
       Service 4 is for messages between asServers
                 moving in the forward direction
                 (away from the homeAS in SPath)
                 until the server is reached.
    */
    switch (service) {
        case 1:
            sprintf(buf, "Welcome to Service Network Graph
                          Service Center!\n
                          You are customer number %i\n#",
                          visit++);
            encRoundRobin_len_all(buf, strlen(buf), 15);
            write(sd2, buf, strlen(buf));
            break;
        case 2:
        case 4:
```

```
if (service == 2){
    // Service 2
    sscanf(&msg_buf[2], "%[^+]%*c%[^+]%*c%32c",
           userID, servicePath, msg_one);
    msg_one[32] = 0;
    printf ("msg_buf: %s\n", msg_buf);
} else {
    // Service 4
    sscanf(&msg_buf[2],"%[^+]%*c%[^+]%*c%[^#]",
           userID, servicePath, keyIndex);
    printf ("keyIndex: %s\n", keyIndex);
}
sscanf(&servicePath[3], "%[^/]", homeASAdd);
pInformation(servicePath , server,
             serviceName, serverPort,
             serverAdd);
printf ("msg_buf: %s\n", msg_buf);
printf ("servicePath: %s\n", servicePath);
printf ("Service Name: %s\n", serviceName);
printf ("Server: %s\n", server);
printf ("Server port: %s\n", serverPort);
printf ("Server address: %s\n", serverAdd);
printf ("msg_one: %s\n", msg_one);
printf ("userID: %s\n", userID);
printf ("Home AS Server Address: <%s>\n",
         homeASAdd);
printf ("thisIP: <%s>\n", thisIP);

if (strcmp(homeASAdd, thisIP) == 0){
    // authentication with user directly
    // get HSP from userHDP.data
    fPtr = fopen("userHDP.data", "r");
    do {
        fscanf (fPtr, "%s %s %s", userIDfile,
                HSPfile, HDPfile);
    } while (strcmp(userID, userIDfile) != 0);

    fclose(fPtr);
    parseMsgOne(R1, HSPfile, msg_one);
    msgTwo(R1, R2, DPnew, msg_two, &len);
    sprintf (buf, "%s#", msg_two);
    printf ("msg_two: %s\n", msg_two);
    encRoundRobin_len_all(buf, strlen(buf),15);
    write(sd2, buf, strlen(buf));
    printf ("Finish writing to sd2\n");
```

```
msg_buf2[0] = 0;
while(n=read(sd2, buf, sizeof(buf))){
    decRoundRobin_len_all(buf, n, 15);
    buf[n] =0;
    strcat(msg_buf2, buf);
    if (buf[n-1] == '#') {
        msg_buf2[strlen(msg_buf2)-1] = 0;
        break;
    }
}

printf("finish reading from sd2\n");
sscanf (msg_buf2, "%64c", msg_three);
msg_three[64] = '\0';

printf ("msg_three: %s\n", msg_three);
parseMsgThree(R2, R3, HDP, msg_three);
if (strcmp(HDP, HDPfile)!=0) {
    printf("Sorry, Service is not
            available!\n");
    exit(1);
}

// Upadte DPnew
md5_action(DPnew, HDPnew);
fPtr = fopen("userHDP.data", "r+");
do {
    fscanf (fPtr, "%s %s %s", userIDfile,
            HSPfile, HDPfile);
} while (strcmp(userID, userIDfile) != 0);
fseek(fPtr, -32, SEEK_CUR);
fprintf(fPtr, "%s", HDPnew);
fclose(fPtr);

fPtr = fopen("userDP.data", "r+");
do {
    fscanf (fPtr, "%s %s %s", userIDfile,
            HSPfile, HDPfile);
} while (strcmp(userID, userIDfile) != 0);
len = strlen (HDPfile);
len *= -1;
fseek(fPtr, len, SEEK_CUR);
fprintf(fPtr, "%s", DPnew);
fclose(fPtr);
```

```
    // session key
    strcpy(combineR, R1);
    strcat(combineR, R2);
    strcat(combineR, R3);
    md5_action(combineR, hashR);
    sKey = md5ToKey(hashR);
    printf("Session key index: %i\n", sKey);
    //sprintf(buf, "#%s#%i##", userID, sKey);
    sprintf(keyIndex, "%i", sKey);

}

/*************** Socket variables ************/
struct  sockaddr_in slsad;
int slssd;
memset((char *)&slsad,0,sizeof(slsad));
slsad.sin_family = AF_INET;
/*****************************************/

if (pInsidePath(servicePath, thisIP)){
    if (strcmp(thisIP, serverAdd)==0){
        // This is the port
        port = atoi(serverPort);
        if (port > 0)
             slsad.sin_port =
                 htons((u_short)port);
        else {
            // print error message and exit
            fprintf(stderr, "bad port number
                             %i\n", port);
            exit(1);
        }

        // This is the IP
        ptrh = gethostbyname("localhost");
        if ( ((char *)ptrh) == NULL ) {
            fprintf(stderr,"invalid host:
                    %s\n", "localhost");
            exit(1);
        }
        memcpy(&slsad.sin_addr, ptrh->h_addr,
               ptrh->h_length);

        /* Map TCP transport protocol name
```

```
   to protocol number. */
if (((int)(ptrp =
    getprotobyname("tcp"))) == 0) {
    fprintf(stderr, "cannot map \"tcp\"
            to protocol number");
    exit(1);
}

/* Create a socket. */
slssd = socket(PF_INET, SOCK_STREAM,
               ptrp->p_proto);
if (sd < 0) {
    fprintf(stderr, "socket creation
                     failed\n");
    exit(1);
}

/* Connect the socket to the
   specified server. */
if (connect(slssd, (struct sockaddr *)
    &slsad, sizeof(slsad)) < 0) {
    fprintf(stderr,"connect failed\n");
    exit(1);
}

// prepare the message to server
// either keyIndex already prepared
// during authentication or get it from
// msg_buf with a forward code 4.
printf ("msg_buf: %s\n", msg_buf);
if (msg_buf[0] == '4'){
    // either authenticated just above
    // or in forward direction
    // msg_buf : userID, servicePath,
    // keyIndex in forward direction
    sscanf (&msg_buf[2], "%*[^+]%*c%*
            [^+]%*c%[^#]", keyIndex);
}
sprintf(answer, "%s+%s#", userID,
                keyIndex);
len = strlen(answer);
answer[len] = '\0';
printf ("Message to appGroup: %s\n",
        answer);
encRoundRobin_len_all(answer,
        strlen(answer), 15);
write(slssd, answer, strlen(answer));

while(n=read(slssd, buf, sizeof(buf))){
    decRoundRobin_len_all(buf, n, 15);
    if (buf[n-1] == '#') {
        buf[n-1] = '\0';
        printf("Port = %s\n", buf);
        buf[n-1] = '#';
        encRoundRobin_len_all(buf,
                              n, 15);
        write(sd2, buf, n);
        // so that asClient detects '#'
        break;
    } else {
        write(1, buf, n);
        encRoundRobin_len_all(buf,
                              n, 15);
        write(sd2, buf, n);
    }
}
close(slssd);
break;
// end of service
/***********************************/
} else {
    /* thisIP in SP.
       Check if at HomeAS, if yes, change
       direction from back to forward
       Determine nextIP using pForward()
       or pBackward
       service = 2 is a default service
                 request and move backward
       service = 4 is the forward (4ward)
                 equivalence of service 2
                 pass msg_buf as is.
       printf ("Inside (thisIP!=serverAdd),
                msg_buf: %s\n", msg_buf);
    */
    if (strcmp(thisIP, homeASAdd)==0){
        service = 4;
        sprintf (msg_buf, "4+%s+%s+%s#",
            userID, servicePath, keyIndex);
        printf ("**** msg_buf: %s\n",
                 msg_buf);
```

```
            // and a content change -
            // no authentication,
            // key index included.
        }
        if (msg_buf[0] == '2'){
            // default, backward
            // inside SP, not homeAS
            // (changed direction if at homeAS)
            // and so must be able to backward
            printf ("Inside '2'\n");
            pBackward(servicePath, thisIP,
                      nextIP);
            printf ("NextIP: %s\n", nextIP);
        } else {
            printf ("Inside '4'\n");
            // inside SP, not serverAdd and
            // so must be able to forward
            pForward(servicePath,thisIP,nextIP);
            printf ("NextIP: %s\n", nextIP);
        }
    }
} else {
    // thisIP not in SP, use sngMap to
    // deliver to HomeAS.
    gethostname(localhostname, 10);
    sprintf(filename, "%sMap.data",
            localhostname);
    printf("filename = %s\n", filename);
    if ((fPtr = fopen(filename, "r"))==NULL){
        printf("Can't open file!\n");
        exit(0);
    }
    do {
        fscanf (fPtr, "%s %s", term1, term2);
    } while (strcmp(term1, homeASAdd)!=0);
    strcpy(nextIP, term2);
    fclose(fPtr);
}
printf ("nextIP: %s\n", nextIP);
// start socket slssd and read write to
// slssd twice, to sd2 twice
printf ("End of case 2, nextIP: %s\n", nextIP);
port = 5500; // for AS
if (port > 0){
    slsad.sin_port = htons((u_short)port);
} else {
    fprintf(stderr,"bad port number %d\n",port);
    exit(1);
}

// socket stuff
// host = nextIP;
// printf("port: %d\nHost: %s\n", port, host);
ptrh = gethostbyname(nextIP);
if ( ((char *)ptrh) == NULL ) {
    fprintf(stderr,"invalid host: %s\n", host);
    exit(1);
}
memcpy(&slsad.sin_addr, ptrh->h_addr,
       ptrh->h_length);

/* Map TCP transport protocol name to
   protocol number. */
if ( ((int)(ptrp=getprotobyname("tcp")))==0){
    fprintf(stderr, "cannot map \"tcp\"
            to protocol number");
    exit(1);
}

/* Create a socket. */
slssd = socket(PF_INET, SOCK_STREAM,
               ptrp->p_proto);
if (sd < 0) {
    fprintf(stderr,"socket creation failed\n");
    exit(1);
}

/* Connect socket to the specified server.*/
if (connect(slssd, (struct sockaddr *)&slsad,
    sizeof(slsad)) < 0) {
    fprintf(stderr,"connect failed\n");
    exit(1);
}

// prepare the message to server
// If service is 2, read and write twice.
// msg_buf straight through.
// If service is 4, read write once.
// msg_buf straight through.
printf("msg_buf in socket stuff:%s\n",msg_buf);
```

```
strcpy(answer, msg_buf);
len = strlen(answer);
answer[len] = '\0';
printf ("Message to other AS: %s\n", answer);

// to slssd, from slssd to sd2, first cycle
encRoundRobin_len_all(answer,
    strlen(answer), 15);
write(slssd, answer, strlen(answer));
printf ("finish writing to slssd\n");
while(n=read(slssd, buf, sizeof(buf))){
    write(sd2, buf, n);
    decRoundRobin_len_all(buf, n, 15);
    write(1, buf, n);
    if (buf[n-1] == '#') {
        break;
    }
}
printf ("Finish reading from slsssd and
         writing to sd2\n");
/* service == 2
   Current AS has to pass the service request
   to next hop AS (MsgOne);
     pass the reply from next hop AS
       to source AS (MsgTwo);
     pass the reply from source AS
       to next hop AS (MsgThree);
     pass the reply from next hop AS
       to source AS (service information).
   service == 4
   Current AS has to pass the service request
     to next hop AS
     (request for service information);
     pass the reply from next hop AS
       to source AS (servcie information);
*/
// AS task completed if service code == 4

if (service == 2){
    while(n=read(sd2, buf, sizeof(buf))){
        write(slssd, buf, n);
        decRoundRobin_len_all(buf, n, 15);
        write(1, buf, n);
        if (buf[n-1] == '#') {
            break;
        }
    }
    printf ("Finish read from sd2,
             write to slssd\n");
    // from slssd to sd2
    while(n=read(slssd, buf, sizeof(buf))){
        write(sd2, buf, n);
        decRoundRobin_len_all(buf, n, 15);
        write(1, buf, n);
        if (buf[n-1] == '#') {
            break;
        }
    }
        printf ("Finish read from slssd,
                 write to sd2\n");
        close(slssd);
    break;
default:
    printf ("Customer number %4i - Request format
        error!\n");
    sprintf(msg_buf, "Request format error!\n
        Thank you! You are customer number %i\n
        Service request unknown!\n##", visit);
    encRoundRobin_len_all(msg_buf, strlen(msg_buf),
                          15);
    write(sd2, msg_buf, strlen(msg_buf));
    break;
        }
        close(sd2);
        exit(0);
    }
    close(sd2);
  }
}
void sig_chld(int signo){
    int pid;
    int stat;
    while((pid = waitpid(-1, &stat, WNOHANG)) > 0)
        printf("child %d terminated.\n", pid);
    return;
}
```

D.2 asClient.cpp

```
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else
#define closesocket close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#endif

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "security.h"
#include "md5ADT.h"

#define PROTOPORT    5500  /* default asServer port number */
#define BUFFSIZE     1024

extern  int             errno;
char    localhost[] =   "localhost";
/* default host name            */

int main(int argc, char *argv[]) {
    struct  hostent  *ptrh;  /* ptr to a host table entry       */
    struct  protoent *ptrp;  /* ptr to a protocol table entry   */
    struct  sockaddr_in sad; /* struct for IP address           */
    int     sd;              /* socket descriptor               */
    int     port;            /* protocol port number            */
    char    *host;           /* ptr to host name                */
    int     n;               /* number of characters read       */
    char    buf[1000];       /* buffer for data from the server */
    char    msg_buf[1000];
    char    userID[1000];
    char    keyBuf[1000];
    char    password[20];
    int     menuChoice;
    char    answer[100];
    char    tempAnswer[100];
    char    service[100];
    int     len;
    char    R1[100];
    char    R2[100];
    char    R3[100];
    char    DP[100];
    char    SP[100];
    char    DPnew[100];
    char    HDP[100];
    char    HSP[100];
    char    msg_one[100];
    char    msg_two[100];
    char    msg_three[100];
    int     i, j, sKey;
    char    servicePath[100];
    char    combineR[300];
    char    hashR[100];
    char    homeASAdd[100];

#ifdef WIN32
    WSADATA wsaData;
    WSAStartup(0x0101, &wsaData);
#endif
    memset((char *)&sad,0,sizeof(sad));
    sad.sin_family = AF_INET;

    if (argc > 2) {
        port = atoi(argv[2]);
    } else {
        port = PROTOPORT;
    }
    if (port > 0)
        sad.sin_port = htons((u_short)port);
    else {
        fprintf(stderr,"bad port number %s\n",argv[2]);
        exit(1);
    }

    if (argc > 1) {
        host = argv[1];
    } else {
        host = localhost;
    }

    /* Convert host name to equivalent IP add and copy to sad. */
```

```
ptrh = gethostbyname(host);
if ( ((char *)ptrh) == NULL ) {
    fprintf(stderr,"invalid host: %s\n", host);
    exit(1);
}
memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);

/* Map TCP transport protocol name to protocol number. */
if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
    fprintf(stderr, "cannot map \"tcp\" to protocol number");
    exit(1);
}

/* Create a socket. */
sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
if (sd < 0) {
    fprintf(stderr, "socket creation failed\n");
    exit(1);
}

/* Connect the socket to the specified server. */
if (connect(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0) {
    fprintf(stderr,"connect failed\n");
    exit(1);
}

printf ("SNG Service Menu:\n"
        "1. Using Service Network Graph\n"
        "2. Service Access\n"
    "3. Exit\n"
        "Please select a service: ");
scanf ("%i", &menuChoice);
switch (menuChoice) {
    case 1:  strcpy(answer, "1#");
             break;
    case 2:
    case 4:
      printf("Please enter the userID: ");
      scanf ("%s", userID);
      printf("Please enter your Dynamic Password SP part: ");
      scanf ("%s", SP);
      printf("Please enter your Dynamic Password DP part: ");
      scanf ("%s", DP);
      printf("Please enter your preferred Service Path: ");
      scanf ("%s", servicePath);
```

```
msgOne(R1, SP, msg_one, &len);
printf ("Message One: %s\n", msg_one);
sprintf(answer, "%d+%s+%s+%s#", menuChoice, userID,
    servicePath, msg_one);
printf ("answer: %s\n", answer);
encRoundRobin_len_all(answer, strlen(answer), 15);

  // To cause a delay for rand()
  for (j=0; j<2500; j++){
      for (i=0; i<2500; i++){
          strcpy (msg_two, msg_one);
      }
  }
  // End of delay

  write(sd, answer, strlen(answer));

  msg_buf[0] = 0;
  while(n=read(sd, buf, sizeof(buf))){
      decRoundRobin_len_all(buf, n, 15);
      buf[n] =0;
      strcat(msg_buf, buf);
      if (buf[n-1] == '#') {
          msg_buf[strlen(msg_buf)-1] = 0;
          break;
      }
  }
  strcpy(msg_two, msg_buf);
  printf ("msg_two received: %s\n", msg_two);
  parseMsgTwo(R1, R2, DPnew, msg_two);
  printf ("R1 used    : %s\n", R1);
  printf ("R2 received: %s\n", R2);
  printf ("DP received: %s\n", DPnew);

  // Another delay for rand()
  for (j=0; j<5000; j++){
      for (i=0; i<5000; i++){
          strcpy (msg_two, msg_one);
      }
  }
  msgThree(R2, R3, DP, msg_three, &len);
  printf ("R3 generated: %s\n", R3);
  printf ("DP sent: %s\n", DP);
  printf ("ServicePath: %s\n", servicePath);
```

```
            sprintf(answer, "%s#", msg_three);
            printf ("msgThree before encryption: %s\n", answer);
            break;
        case 3:  closesocket(sd);
            exit(0);
        default: break;
    }
    printf("answer send: %s\n", answer);
    encRoundRobin_len_all(answer, strlen(answer), 15);
    write(sd, answer, strlen(answer));
    printf ("finish writing msg_three\n");
    while(n=read(sd, buf, sizeof(buf))){
        decRoundRobin_len_all(buf, n, 15);
        buf[n] = 0;
        if (buf[n-1] == '#') {
        buf[n-1] = 0;
        printf ("%s\n", buf);
        break;
    }
    printf ("%s", buf);
    }

    closesocket(sd);
    if (menuChoice != 1){
        strcpy(combineR, R1);
        strcat(combineR, R2);
        strcat(combineR, R3);
        md5_action(combineR, hashR);
        sKey = md5ToKey(hashR);

        printf("\nSession key index as calculated in asClient:
                %i\n", sKey);
    }

    exit(0);
}
```

D.3 Date Server

D.3.1 dateGroup.cpp

```
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "security.h"
#include "md5ADT.h"

#define PROTOPORT    6002    /* default timeServer port number   */
#define QLEN         6       /* size of request queue            */

void sig_chld(int signo);

int main(){
    struct    hostent  *ptrh;  /* ptr to a host table entry      */
    struct    protoent *ptrp;  /* ptr to a protocol table entry  */
    struct    sockaddr_in sad; /* struct for server's address    */
    struct    sockaddr_in cad; /* struct for client's address    */
    int       sd, sd2;         /* socket descriptors             */
    int       port;            /* protocol port number           */
    socklen_t alen;
    char      buf[1000];       /* buffer for str server sends    */
    char      msg_buf[1000];
    char      port_buf[100];
    char      keyBuf[10];
    char      userID[500];
    char      userToken [500];
    int       rNum;
    int       visit = 0;
    time_t    time_now;
    char      time_str[50];

    FILE      *fPtr;
```

```
    char      username[128] ;
    char      password[128];
    int       flag = 0;
    int       n, i, pid, passLen, nameLen, buf_len;
    int       keyIndex;
    /* generate keyIndex, send to AS and server */

#ifdef WIN32
    WSADATA wsaData;
    WSAStartup(0x0101, &wsaData);
#endif
    memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr struct*/
    sad.sin_family = AF_INET;           /* set to Internet      */
    sad.sin_addr.s_addr = INADDR_ANY;   /* set the local IP add */
    port = PROTOPORT;                   /* use default port no. */
    sad.sin_port = htons((u_short)port);

    /* Map TCP transport protocol name to protocol number */
    if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
        fprintf(stderr, "cannot map \"tcp\" to protocol number");
        exit(1);
    }

    /* Create a socket */
    sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
    if (sd < 0) {
        fprintf(stderr, "socket creation failed\n");
        exit(1);
    }

    /* Bind a local address to the socket */
    if (bind(sd, (struct sockaddr *)&sad,
        (socklen_t)(sizeof(sad))) < 0) {
        fprintf(stderr,"bind failed\n");
        exit(1);
    }

    /* Specify size of request queue */
    if (listen(sd, QLEN) < 0) {
        fprintf(stderr,"listen failed\n");
        exit(1);
    }

    /* To avoid zombie */
    signal(SIGCHLD, sig_chld);
```

```
    /* Main server loop - accept and handle requests */
    while (1) {
        alen = (socklen_t)(sizeof(cad));
        if ( (sd2=accept(sd, (struct sockaddr *)&cad,
            &alen)) < 0){
            fprintf(stderr, "accept failed\n");
            exit(1);
        }
        visit++;

        /* fork() to handle multiple connections */
        if ((pid = fork()) == 0) {
            close(sd); // child not listening

            /* read all from client */
            msg_buf[0] = 0;
            while(n=read(sd2, buf, sizeof(buf))){
                decRoundRobin_len_all(buf, n, 15);
                buf[n] =0;
                strcat(msg_buf, buf);
                if (buf[n-1] == '#') {
                    msg_buf[strlen(msg_buf)-1] = 0;
                    break;
                }
            }
            sscanf(msg_buf, "%[^+]%*c%s", userID, keyBuf);
            srand(time(NULL));
            rNum = random()%999;
            /* Create port */
            sprintf(buf, "%i", 8000+rNum);
            strcpy(port_buf, buf);
            buf_len = strlen(buf);
            buf[buf_len] = '#';
            buf[buf_len+1] = 0;
            encRoundRobin_len_all(buf,strlen(buf), 15);
            write(sd2, buf, strlen(buf));

            /* Call the service */
            execl("/home/linux/dateServer", "./dateServer",
                "localhost", port_buf, userID, keyBuf, NULL);

            exit(0);
        }   // fork()
        close(sd2);
```

```
    }
}

void sig_chld(int signo){
    int pid;
    int stat;
    while((pid = waitpid(-1, &stat, WNOHANG)) > 0)
        printf("child %d terminated.\n", pid);
    return;
}
```

D.3.2 dateServer.cpp

```
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "security.h"
#include "md5ADT.h"
#define PROTOPORT    6002      /* default dateServer port number */
#define QLEN         6         /* size of request queue          */

void sig_chld(int signo);

int main(int argc, char * argv[]){
    struct    hostent  *ptrh;  /* ptr to a host table entry      */
    struct    protoent *ptrp;  /* ptr to a protocol table entry  */
    struct    sockaddr_in sad; /* struct for server's address    */
    struct    sockaddr_in cad; /* struct for client's address    */
    int       sd, sd2;         /* socket descriptors             */
    int       port;            /* protocol port number           */
    socklen_t alen;
    char      buf[1000];       /* buffer for str server sends    */
    char      msg_buf[1000];
    time_t    time_now;
    char      time_str[50];
    int       keyIndex;

    FILE     *fPtr;
    char     username[128] ;
    char     password[128];
    int      flag = 0;
    int      n, i, pid, passLen, nameLen, buf_len;
    char     localhostname[10];


#ifdef WIN32
    WSADATA wsaData;
    WSAStartup(0x0101, &wsaData);
#endif
    memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr struct */
    sad.sin_family = AF_INET;           /* set to Internet       */
    sad.sin_addr.s_addr = INADDR_ANY;   /* set local IP add      */

    if (argc < 5){
        fprintf(stderr, "Usage: dateServer hostAdd(future_use)
            port_no username encr_key\n");
    exit(1);
    }

    if (argc > 2){
        port = atoi(argv[2]);
    } else {
        port = PROTOPORT;        /* use default port number  */
    }
    sad.sin_port = htons((u_short)port);

    /* Map TCP transport protocol name to protocol number */
    if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
        fprintf(stderr, "cannot map \"tcp\" to protocol number");
        exit(1);
    }

    /* Create a socket */
    sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
    if (sd < 0) {
```

```
    fprintf(stderr, "socket creation failed\n");
    exit(1);
}

/* Bind a local address to the socket */
if (bind(sd, (struct sockaddr *)&sad,
    (socklen_t)(sizeof(sad))) < 0) {
    fprintf(stderr,"bind failed\n");
    exit(1);
}

/* Specify size of request queue */
if (listen(sd, QLEN) < 0) {
    fprintf(stderr,"listen failed\n");
    exit(1);
}
signal(SIGCHLD, sig_chld);

/* Main server loop - accept and handle requests */
alen = (socklen_t)(sizeof(cad));
if ( (sd2=accept(sd, (struct sockaddr *)&cad, &alen)) < 0){
    fprintf(stderr, "accept failed\n");
    exit(1);
}

/* Set the encr/decr key is specified as argv[3] */
if (argc > 4){
    keyIndex = atoi(argv[4]);
} else {
    keyIndex = 15;
}

/*read from client until '#' */
username[0] = 0;
while(n=read(sd2, buf, sizeof(buf))){
    decRoundRobin_len_all(buf, n, keyIndex);
    buf[n] =0;

    strcpy(username, buf);
    if (buf[n-1] == '#') {
        username[strlen(username)-1] = 0;
        break;
    }
}
```

```
    /* check authentication token.
       Get token from file.
       keyIndex specified by dateAppServer when execl().
    */
    if (strcmp(username, argv[3]) == 0){
        time_now = time(NULL);
        strftime(msg_buf, 49, "%d/%m/%Y", localtime(&time_now));
        gethostname(localhostname, 10);
        sprintf(buf, "From %s: Current date is %s\n#",
            localhostname, msg_buf);
        encRoundRobin_len_all(buf, strlen(buf), keyIndex);
        write(sd2, buf, strlen(buf));
    } else {
        printf ("Request format error!\n");
        sprintf(msg_buf, "Request format error!\n
            Service request unknown!\n##");
        encRoundRobin_len_all(msg_buf, strlen(msg_buf), keyIndex);
        write(sd2, msg_buf, strlen(msg_buf));
    }
    close(sd2);
}

void sig_chld(int signo){
    int pid;
    int stat;
    while((pid = waitpid(-1, &stat, WNOHANG)) > 0)
        printf("child %d terminated.\n", pid);
    return;
}
```

D.4 Echo Server

D.4.1 echoGroup.cpp

```
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
```

```
#include <unistd.h>
#include <netinet/in.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "security.h"
#include "md5ADT.h"

#define PROTOPORT    6003    /* default timeServer port number   */
#define QLEN         6       /* size of request queue            */

void sig_chld(int signo);

int main(){
    struct    hostent  *ptrh;  /* ptr to a host table entry      */
    struct    protoent *ptrp;  /* ptr to a protocol table entry  */
    struct    sockaddr_in sad; /* struct for server's address    */
    struct    sockaddr_in cad; /* struct for client's address    */
    int       sd, sd2;         /* socket descriptors             */
    int       port;            /* protocol port number           */
    socklen_t alen;
    char      buf[1000];       /* buffer for str server sends    */
    char      port_buf[100];
    char      msg_buf[1000];
    char      keyBuf[10];
    char      userID[500];
    char      userToken [500];
    int   rNum;
    int   visit = 0;
    time_t   time_now;
    char   time_str[50];

    FILE   *fPtr;
    char   username[128] ;
    char   password[128];
    int    flag = 0;
    int    n, i, pid, passLen, nameLen, buf_len;
    int    keyIndex;
    /* generate keyIndex, send to AS and server */

#ifdef WIN32
    WSADATA wsaData;
    WSAStartup(0x0101, &wsaData);
#endif
    memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr struct */
    sad.sin_family = AF_INET;           /* set to Internet       */
    sad.sin_addr.s_addr = INADDR_ANY;   /* set local IP add      */
    port = PROTOPORT;                   /* use default port no.  */
    sad.sin_port = htons((u_short)port);

    /* Map TCP transport protocol name to protocol number */
    if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
        fprintf(stderr, "cannot map \"tcp\" to protocol number");
        exit(1);
    }

    /* Create a socket */
    sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
    if (sd < 0) {
        fprintf(stderr, "socket creation failed\n");
        exit(1);
    }

    /* Bind a local address to the socket */
    if (bind(sd, (struct sockaddr *)&sad,
        (socklen_t)(sizeof(sad))) < 0) {
        fprintf(stderr,"bind failed\n");
        exit(1);
    }

    /* Specify size of request queue */
    if (listen(sd, QLEN) < 0) {
        fprintf(stderr,"listen failed\n");
        exit(1);
    }

    /* To avoid zombie */
    signal(SIGCHLD, sig_chld);

    /* Main server loop - accept and handle requests */
    while (1) {
        alen = (socklen_t)(sizeof(cad));
        if ( (sd2=accept(sd, (struct sockaddr *)&cad,
            &alen)) < 0){
            fprintf(stderr, "accept failed\n");
            exit(1);
```

```
        }
        visit++;

        /* fork() to handle multiple connections */
        if ((pid = fork()) == 0) {
            close(sd); // child not listening

            /* read all from client */
            msg_buf[0] = 0;
            while(n=read(sd2, buf, sizeof(buf))){
                decRoundRobin_len_all(buf, n, 15);
                buf[n] =0;
                strcat(msg_buf, buf);
                if (buf[n-1] == '#') {
                    msg_buf[strlen(msg_buf)-1] = 0;
                    break;
                }
            }
            sscanf(msg_buf, "%[^+]%*c%s]", userID, keyBuf);
            srand(time(NULL));
            rNum = random()%999;
            /* Create port */
            sprintf(buf, "%i", 9000+rNum);
            strcpy(port_buf, buf);
            buf_len = strlen(buf);
            buf[buf_len] = '#';
            buf[buf_len+1] = 0;
            encRoundRobin_len_all(buf,strlen(buf), 15);
            write(sd2, buf, strlen(buf));

            /* Call the service */
            execl("/home/linux/echoServer", "./echoServer",
                "localhost", port_buf, userID, keyBuf, NULL);
            exit(0);
        }   // fork()
        close(sd2);
    }
}

void sig_chld(int signo){
    int pid;
    int stat;
    while((pid = waitpid(-1, &stat, WNOHANG)) > 0)
        printf("child %d terminated.\n", pid);
    return;
}
```

D.4.2 echoServer.cpp

```
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "security.h"
#include "md5ADT.h"
#define PROTOPORT    6003    /* default dateServer port number   */
#define QLEN         6       /* size of request queue            */

void sig_chld(int signo);

int main(int argc, char * argv[]){
    struct    hostent  *ptrh;  /* ptr to a host table entry      */
    struct    protoent *ptrp;  /* ptr to a protocol table entry  */
    struct    sockaddr_in sad; /* struct for server's address    */
    struct    sockaddr_in cad; /* struct for client's address    */
    int       sd, sd2;         /* socket descriptors             */
    int       port;            /* protocol port number           */
    socklen_t alen;
    char      buf[1000];       /* buffer for str server sends    */
    char      msg_buf[1000];
    time_t   time_now;
    char     time_str[50];
    int      keyIndex;
    char     userLine[1000];
```

```
    FILE     *fPtr;
    char     username[128] ;
    char     password[128];
    int      flag = 0;
    int      n, i, pid, passLen, nameLen, buf_len;
    char     localhostname[10];


#ifdef WIN32
    WSADATA wsaData;
    WSAStartup(0x0101, &wsaData);
#endif
    memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr struct */
    sad.sin_family = AF_INET;           /* set to Internet       */
    sad.sin_addr.s_addr = INADDR_ANY;   /* set local IP add      */
    if (argc < 5){
        fprintf(stderr, "Usage: dateServer hostAdd(future_use)
            port_no username encr_key\n");
        exit(1);
    }

    if (argc > 2){
        port = atoi(argv[2]);
    } else {
        port = PROTOPORT;        /* use default port number  */
    }
    sad.sin_port = htons((u_short)port);


    /* Map TCP transport protocol name to protocol number */
    if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
        fprintf(stderr, "cannot map \"tcp\" to protocol number");
        exit(1);
    }

    /* Create a socket */
    sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
    if (sd < 0) {
        fprintf(stderr, "socket creation failed\n");
        exit(1);
    }

    /* Bind a local address to the socket */
    if (bind(sd, (struct sockaddr *)&sad,
        (socklen_t)(sizeof(sad))) < 0) {
        fprintf(stderr,"bind failed\n");
        exit(1);
    }
    /* Specify size of request queue */
    if (listen(sd, QLEN) < 0) {
        fprintf(stderr,"listen failed\n");
        exit(1);
    }

    signal(SIGCHLD, sig_chld);

    /* Main server loop - accept and handle requests */
    alen = (socklen_t)(sizeof(cad));
    if ( (sd2=accept(sd, (struct sockaddr *)&cad, &alen)) < 0){
        fprintf(stderr, "accept failed\n");
        exit(1);
    }

    /* Set the encr/decr key is specified as argv[3] */
    if (argc > 4){
        keyIndex = atoi(argv[4]);
    } else {
        keyIndex = 15;
    }

    /*read from client until '#' */
    msg_buf[0] = 0;
    while(n=read(sd2, buf, sizeof(buf))){
        decRoundRobin_len_all(buf, n, keyIndex);
        buf[n] =0;
        strcat(msg_buf, buf);
        if (buf[n-1] == '#') {
            msg_buf[strlen(msg_buf)-1] = 0;
            break;
        }
    }
    sscanf(msg_buf, "%[^+]%*c%[^\n]", username, userLine);

    /* check authentication token.
       Get token from file.
       keyIndex specified by dateAppServer when execl().
    */
    if (strcmp(username, argv[3]) == 0){
        gethostname(localhostname, 10);
        sprintf(buf, "From %s: Your line is: %s\n#",
```

```
            localhostname, userLine);
        encRoundRobin_len_all(buf, strlen(buf), keyIndex);
        write(sd2, buf, strlen(buf));
    } else {
        printf ("Request format error!\n");
        sprintf(msg_buf, "Request format error!\n
            Service request unknown!\n##");
        encRoundRobin_len_all(msg_buf, strlen(msg_buf), keyIndex);
        write(sd2, msg_buf, strlen(msg_buf));
    }
    close(sd2);
}

void sig_chld(int signo){
    int pid;
    int stat;
    while((pid = waitpid(-1, &stat, WNOHANG)) > 0)
        printf("child %d terminated.\n", pid);
    return;
}
```

D.5 Name Server

D.5.1 nameGroup.cpp

```
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "security.h"
#include "md5ADT.h"

#define PROTOPORT    6004    /* default timeServer port number   */
#define QLEN         6       /* size of request queue            */

void sig_chld(int signo);

int main(){
    struct    hostent  *ptrh;  /* ptr to a host table entry      */
    struct    protoent *ptrp;  /* ptr to a protocol table entry  */
    struct    sockaddr_in sad; /* struct for server's address    */
    struct    sockaddr_in cad; /* struct for client's address    */
    int       sd, sd2;         /* socket descriptors             */
    int       port;            /* protocol port number           */
    socklen_t alen;
    char      buf[1000];       /* buffer for str server sends    */
    char      port_buf[100];
    char      msg_buf[1000];
    char      keyBuf[10];
    char      userID[500];
    char      userToken [500];
    int       rNum;
    int       visit = 0;
    time_t    time_now;
    char      time_str[50];

    FILE      *fPtr;
    char      username[128] ;
    char      password[128];
    int       flag = 0;
    int       n, i, pid, passLen, nameLen, buf_len;
    int       keyIndex;
    /* generate keyIndex, send to AS and server */

#ifdef WIN32
    WSADATA wsaData;
    WSAStartup(0x0101, &wsaData);
#endif
    memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr struct */
    sad.sin_family = AF_INET;           /* set to Internet       */
    sad.sin_addr.s_addr = INADDR_ANY;   /* set local IP add      */
    port = PROTOPORT;                   /* use default port no.  */
    sad.sin_port = htons((u_short)port);
```

```
/* Map TCP transport protocol name to protocol number */
if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
    fprintf(stderr, "cannot map \"tcp\" to protocol number");
    exit(1);
}

/* Create a socket */
sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
if (sd < 0) {
    fprintf(stderr, "socket creation failed\n");
    exit(1);
}

/* Bind a local address to the socket */
if (bind(sd, (struct sockaddr *)&sad,
    (socklen_t)(sizeof(sad))) < 0) {
    fprintf(stderr,"bind failed\n");
    exit(1);
}

/* Specify size of request queue */
if (listen(sd, QLEN) < 0) {
    fprintf(stderr,"listen failed\n");
    exit(1);
}

/* To avoid zombie */
signal(SIGCHLD, sig_chld);

/* Main server loop - accept and handle requests */
while (1) {
    alen = (socklen_t)(sizeof(cad));
    if ( (sd2=accept(sd, (struct sockaddr *)&cad,
        &alen)) < 0){
        fprintf(stderr, "accept failed\n");
        exit(1);
    }
    visit++;

    /* fork() to handle multiple connections */
    if ((pid = fork()) == 0) {
        close(sd); // child not listening

        /* read all from client */
        msg_buf[0] = 0;
            while(n=read(sd2, buf, sizeof(buf))){
                decRoundRobin_len_all(buf, n, 15);
                buf[n] =0;
                strcat(msg_buf, buf);
                if (buf[n-1] == '#') {
                    msg_buf[strlen(msg_buf)-1] = 0;
                    break;
                }
            }
            sscanf(msg_buf, "%[^+]%*c%s", userID, keyBuf);
            srand(time(NULL));
            rNum = random()%999;
            /* Create port */
            sprintf(buf, "%i", 10000+rNum);
            strcpy(port_buf, buf);
            buf_len = strlen(buf);
            buf[buf_len] = '#';
            buf[buf_len+1] = 0;
            encRoundRobin_len_all(buf,strlen(buf), 15);
            write(sd2, buf, strlen(buf));

            /* Call the service */
            execl("/home/linux/nameServer", "./nameServer",
                "localhost", port_buf, userID, keyBuf, NULL);

            exit(0);
        }
        close(sd2);
    }
}

void sig_chld(int signo){
    int pid;
    int stat;
    while((pid = waitpid(-1, &stat, WNOHANG)) > 0)
        printf("child %d terminated.\n", pid);
    return;
}
```

D.5.2 nameServer.cpp

```
#ifndef unix
#define WIN32
#include <windows.h>
```

```
#include <winsock.h>
#else

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "security.h"
#include "md5ADT.h"
#define PROTOPORT    6004    /* default dateServer port number   */
#define QLEN         6       /* size of request queue            */

void sig_chld(int signo);

int main(int argc, char * argv[]){
    struct    hostent  *ptrh;  /* ptr to a host table entry      */
    struct    protoent *ptrp;  /* ptr to a protocol table entry  */
    struct    sockaddr_in sad; /* struct for server's address    */
    struct    sockaddr_in cad; /* struct for client's address    */
    int       sd, sd2;         /* socket descriptors             */
    int       port;            /* protocol port number           */
    socklen_t alen;
    char      buf[1000];       /* buffer for str server sends    */
    char      msg_buf[1000];
    time_t    time_now;
    char      time_str[50];
    int       keyIndex;

    FILE      *fPtr;
    char      username[128] ;
    char      password[128];
    int       flag = 0;
    int       n, i, pid, passLen, nameLen, buf_len;
    char      localhostname[10];


#ifdef WIN32
    WSADATA wsaData;
    WSAStartup(0x0101, &wsaData);
#endif
    memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr struct */
    sad.sin_family = AF_INET;           /* set to Internet       */
    sad.sin_addr.s_addr = INADDR_ANY;   /* set local IP add      */

    if (argc < 5){
        fprintf(stderr, "Usage: dateServer hostAdd(future_use)
            port_no username encr_key\n");
        exit(1);
    }

    if (argc > 2){
        port = atoi(argv[2]);
    } else {
        port = PROTOPORT;        /* use default port number  */
    }
    sad.sin_port = htons((u_short)port);

    /* Map TCP transport protocol name to protocol number */
    if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
        fprintf(stderr, "cannot map \"tcp\" to protocol number");
        exit(1);
    }

    /* Create a socket */
    sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
    if (sd < 0) {
        fprintf(stderr, "socket creation failed\n");
        exit(1);
    }

    /* Bind a local address to the socket */
    if (bind(sd, (struct sockaddr *)&sad,
        (socklen_t)(sizeof(sad))) < 0) {
        fprintf(stderr,"bind failed\n");
        exit(1);
    }
    /* Specify size of request queue */
    if (listen(sd, QLEN) < 0) {
        fprintf(stderr,"listen failed\n");
        exit(1);
    }
```

```
signal(SIGCHLD, sig_chld);

/* Main server loop - accept and handle requests */
alen = (socklen_t)(sizeof(cad));
if ( (sd2=accept(sd, (struct sockaddr *)&cad, &alen)) < 0){
    fprintf(stderr, "accept failed\n");
    exit(1);
}

/* Set the encr/decr key is specified as argv[3] */
if (argc > 4){
    keyIndex = atoi(argv[4]);
} else {
    keyIndex = 15;
}

/*read from client until '#' */
username[0] = 0;
while(n=read(sd2, buf, sizeof(buf))){
    decRoundRobin_len_all(buf, n, keyIndex);
    buf[n] =0;
    strcat(username, buf);
    if (buf[n-1] == '#') {
        username[strlen(username)-1] = 0;
        break;
    }
}
/* check authentication token.
   Get token from file.
   keyIndex specified by dateAppServer when execl().
*/
if (strcmp(username, argv[3]) == 0){
    gethostname(localhostname, 10);
    sprintf(buf, "From %s: Name of this Server is \"%s\"\n#",
        localhostname, &argv[0][2]);
    encRoundRobin_len_all(buf, strlen(buf), keyIndex);
    write(sd2, buf, strlen(buf));

} else {
    printf ("Request format error!\n");
    sprintf(msg_buf, "Request format error!\n
        Service request unknown!\n##");
    encRoundRobin_len_all(msg_buf, strlen(msg_buf), keyIndex);
    write(sd2, msg_buf, strlen(msg_buf));
}
    close(sd2);
}

void sig_chld(int signo){
    int pid;
    int stat;
    while((pid = waitpid(-1, &stat, WNOHANG)) > 0)
        printf("child %d terminated.\n", pid);
    return;
}
```

D.6 Time Server

D.6.1 timeGroup.cpp

```
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include time.h>
#include "security.h"
#include "md5ADT.h"

#define PROTOPORT    6001    /* default timeServer port number   */
#define QLEN         6       /* size of request queue            */

void sig_chld(int signo);

int main(){
```

```
    struct    hostent  *ptrh;  /* ptr to a host table entry      */
    struct    protoent *ptrp;  /* ptr to a protocol table entry  */
    struct    sockaddr_in sad; /* struct for server's address    */
    struct    sockaddr_in cad; /* struct for client's address    */
    int       sd, sd2;         /* socket descriptors             */
    int       port;            /* protocol port number           */
    socklen_t alen;
    char      buf[1000];       /* buffer for str server sends    */
    char      msg_buf[1000];
    char      port_buf[100];
    char      keyBuf[10];
    char      userID[500];
    char      userToken [500];
    int       rNum;
    int       visit = 0;
    time_t    time_now;
    char      time_str[50];

    FILE      *fPtr;
    char      username[128] ;
    char      password[128];
    int       flag = 0;
    int       n, i, pid, passLen, nameLen, buf_len;
    int       keyIndex;
    /* generate keyIndex, send to AS and server */

#ifdef WIN32
    WSADATA wsaData;
    WSAStartup(0x0101, &wsaData);
#endif
    memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr struct */
    sad.sin_family = AF_INET;           /* set to Internet       */
    sad.sin_addr.s_addr = INADDR_ANY;   /* set local IP add      */
    port = PROTOPORT;                   /* use default port no.  */
    sad.sin_port = htons((u_short)port);

    /* Map TCP transport protocol name to protocol number */
    if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
        fprintf(stderr, "cannot map \"tcp\" to protocol number");
        exit(1);
    }

    /* Create a socket */
    sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
    if (sd < 0) {
        fprintf(stderr, "socket creation failed\n");
        exit(1);
    }

    /* Bind a local address to the socket */
    if (bind(sd, (struct sockaddr *)&sad,
        (socklen_t)(sizeof(sad))) < 0) {
        fprintf(stderr,"bind failed\n");
        exit(1);
    }

    /* Specify size of request queue */
    if (listen(sd, QLEN) < 0) {
        fprintf(stderr,"listen failed\n");
        exit(1);
    }

    /* To avoid zombie */
    signal(SIGCHLD, sig_chld);

    /* Main server loop - accept and handle requests */
    while (1) {
        alen = (socklen_t)(sizeof(cad));
        if ( (sd2=accept(sd, (struct sockaddr *)&cad, &alen)) < 0){
            fprintf(stderr, "accept failed\n");
            exit(1);
        }
        visit++;

        /* fork() to handle multiple connections */
        if ((pid = fork()) == 0) {
            close(sd); // child not listening

            /* read all from client */
        msg_buf[0] = 0;
            while(n=read(sd2, buf, sizeof(buf))){
                decRoundRobin_len_all(buf, n, 15);
        buf[n] =0;
            strcat(msg_buf, buf);
            if (buf[n-1] == '#') {
                msg_buf[strlen(msg_buf)-1] = 0;
                    break;
            }
        }
            sscanf(msg_buf, "%[^+]%*c%s", userID, keyBuf);
```

```
            srand(time(NULL));
            rNum = random()%999;
            /* Create port */
            sprintf(buf, "%i", 7000+rNum);
            strcpy(port_buf, buf);
            buf_len = strlen(buf);
            buf[buf_len] = '#';
            buf[buf_len+1] = '\0';
            encRoundRobin_len_all(buf,strlen(buf), 15);
            write(sd2, buf, strlen(buf));

            /* Call the service */
            printf("%d\n", execl("/home/linux/timeServer",
                "timeServer", "localhost", port_buf, userID,
                keyBuf, NULL));
            exit(0);
        }
        close(sd2);
    }
}

void sig_chld(int signo){
    int pid;
    int stat;
    while((pid = waitpid(-1, &stat, WNOHANG)) > 0)
        printf("child %d terminated.\n", pid);
    return;
}
```

D.6.2 timeServer.cpp

```
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netdb.h>
#endif
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "security.h"
#include "md5ADT.h"
#define PROTOPORT    6001    /* default dateServer port number        */
#define QLEN         6       /* size of request queue                 */

void sig_chld(int signo);

int main(int argc, char * argv[]){
    struct    hostent  *ptrh;  /* pointer to a host table entry       */
    struct    protoent *ptrp;  /* pointer to a protocol table entry   */
    struct    sockaddr_in sad; /* structure to hold server's address  */
    struct    sockaddr_in cad; /* structure to hold client's address  */
    int       sd, sd2;         /* socket descriptors                  */
    int       port;            /* protocol port number                */
    socklen_t alen;
    char      buf[1000];       /* buffer for string the server sends  */
    char      msg_buf[1000];
    time_t    time_now;
    char      time_str[50];
    int       keyIndex;

    FILE     *fPtr;
    char     username[128] ;
    char     password[128];
    int      flag = 0;
    int      n, i, pid, passLen, nameLen, buf_len;
    char     localhostname[10];


#ifdef WIN32
    WSADATA wsaData;
    WSAStartup(0x0101, &wsaData);
#endif
    memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
    sad.sin_family = AF_INET;           /* set family to Internet   */
    sad.sin_addr.s_addr = INADDR_ANY;   /* set the local IP address */

    if (argc < 5){
        fprintf(stderr, "Usage: dateServer hostAdd(future_use)
            port_no username encr_key\n");
```

```
exit(1);
}

if (argc > 2){
    port = atoi(argv[2]);
} else {
    port = PROTOPORT;        /* use default port number  */
}
sad.sin_port = htons((u_short)port);

/* Map TCP transport protocol name to protocol number */
if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
    fprintf(stderr, "cannot map \"tcp\" to protocol number");
    exit(1);
}

/* Create a socket */
sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
if (sd < 0) {
    fprintf(stderr, "socket creation failed\n");
    exit(1);
}

/* Bind a local address to the socket */
if (bind(sd, (struct sockaddr *)&sad,
    (socklen_t)(sizeof(sad))) < 0) {
    fprintf(stderr,"bind failed\n");
    exit(1);
}
/* Specify size of request queue */
if (listen(sd, QLEN) < 0) {
    fprintf(stderr,"listen failed\n");
    exit(1);
}

signal(SIGCHLD, sig_chld);

/* Main server loop - accept and handle requests */
alen = (socklen_t)(sizeof(cad));
if ( (sd2=accept(sd, (struct sockaddr *)&cad, &alen)) < 0){
    fprintf(stderr, "accept failed\n");
    exit(1);
}

/* Set the encr/decr key is specified as argv[3] */
    if (argc > 4){
        keyIndex = atoi(argv[4]);
    } else {
        keyIndex = 15;
    }

    /*read from client until '#' */
    username[0] = 0;
    while(n=read(sd2, buf, sizeof(buf))){
        decRoundRobin_len_all(buf, n, keyIndex);
    buf[n] =0;
        strcat(username, buf);
        if (buf[n-1] == '#') {
        username[strlen(username)-1] = 0;
            break;
        }
    }
    /* check authentication token.
       Get token from file.
       keyIndex specified by dateAppServer when execl().
    */
    if (strcmp(username, argv[3]) == 0){
        time_now = time(NULL);
        strftime(msg_buf, 49, "%X", localtime(&time_now));
        gethostname(localhostname, 10);
        sprintf(buf, "From %s: Current time is %s\n#",
            localhostname, msg_buf);
        encRoundRobin_len_all(buf, strlen(buf), keyIndex);
        write(sd2, buf, strlen(buf));
    } else {
        printf ("Request format error!\n");
        sprintf(msg_buf, "Request format error!\n
            Service request unknown!\n##");
        encRoundRobin_len_all(msg_buf, strlen(msg_buf), keyIndex);
        write(sd2, msg_buf, strlen(msg_buf));
    }
    close(sd2);
}

void sig_chld(int signo){
    int pid;
    int stat;
    while((pid = waitpid(-1, &stat, WNOHANG)) > 0)
        printf("child %d terminated.\n", pid);
    return;
```

```
}
```

D.7 Service Listing Server

D.7.1 slsSGroup.cpp

```
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "security.h"
#include "md5ADT.h"

#define PROTOPORT    6006    /* default timeServer port number   */
#define QLEN         6       /* size of request queue            */

void sig_chld(int signo);

int main(){
    struct    hostent  *ptrh;  /* ptr to a host table entry      */
    struct    protoent *ptrp;  /* ptr to a protocol table entry  */
    struct    sockaddr_in sad; /* struct for server's address    */
    struct    sockaddr_in cad; /* struct for client's address    */
    int       sd, sd2;         /* socket descriptors             */
    int       port;            /* protocol port number           */
    socklen_t alen;
    char      buf[1000];       /* buffer for str server sends    */
    char      port_buf[100];
    char      msg_buf[1000];
    char      keyBuf[10];
    char      userID[500];
    char      userToken [500];
    int       rNum;
    int       visit = 0;
    time_t    time_now;
    char      time_str[50];

    FILE      *fPtr;
    char      username[128] ;
    char      password[128];
    int       flag = 0;
    int       n, i, pid, passLen, nameLen, buf_len;
    int       keyIndex;
    /* generate keyIndex, send to AS and server */

#ifdef WIN32
    WSADATA wsaData;
    WSAStartup(0x0101, &wsaData);
#endif
    memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr struct */
    sad.sin_family = AF_INET;           /* set to Internet       */
    sad.sin_addr.s_addr = INADDR_ANY;   /* set local IP add      */
    port = PROTOPORT;                   /* use default port no.  */
    sad.sin_port = htons((u_short)port);

    /* Map TCP transport protocol name to protocol number */
    if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
        fprintf(stderr, "cannot map \"tcp\" to protocol number");
        exit(1);
    }

    /* Create a socket */
    sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
    if (sd < 0) {
        fprintf(stderr, "socket creation failed\n");
        exit(1);
    }

    /* Bind a local address to the socket */
    if (bind(sd, (struct sockaddr *)&sad,
        (socklen_t)(sizeof(sad))) < 0) {
        fprintf(stderr,"bind failed\n");
        exit(1);
```

```
}

/* Specify size of request queue */
if (listen(sd, QLEN) < 0) {
    fprintf(stderr,"listen failed\n");
    exit(1);
}

/* To avoid zombie */
signal(SIGCHLD, sig_chld);

/* Main server loop - accept and handle requests */
while (1) {
    alen = (socklen_t)(sizeof(cad));
    if ( (sd2=accept(sd, (struct sockaddr *)&cad, &alen)) < 0){
        fprintf(stderr, "accept failed\n");
        exit(1);
    }
    visit++;

    /* fork() to handle multiple connections */
    if ((pid = fork()) == 0) {
        close(sd); // child not listening

        /* read all from client */
    msg_buf[0] = 0;
        while(n=read(sd2, buf, sizeof(buf))){
            decRoundRobin_len_all(buf, n, 15);
            buf[n] =0;
            strcat(msg_buf, buf);
            if (buf[n-1] == '#') {
                msg_buf[strlen(msg_buf)-1] = 0;
                break;
            }
        }
    sscanf(msg_buf, "%[^+]%*c%s]", userID, keyBuf);
        srand(time(NULL));
        rNum = random()%999;
        /* Create port */
        sprintf(buf, "%i", 12000+rNum);
        strcpy(port_buf, buf);
        buf_len = strlen(buf);
        buf[buf_len] = '#';
        buf[buf_len+1] = 0;
        encRoundRobin_len_all(buf,strlen(buf), 15);
            write(sd2, buf, strlen(buf));
            printf ("userID: %s\nkeyIndex: %s\nPort: %s\n",
                userID, keyBuf, port_buf);
            /* Call the service */
            execl("/home/linux/slsSServer", "./slsSServer",
                "localhost", port_buf, userID, keyBuf, NULL);
            exit(0);
        }   // fork()
        close(sd2);
    }
}

void sig_chld(int signo){
    int pid;
    int stat;
    while((pid = waitpid(-1, &stat, WNOHANG)) > 0)
        printf("child %d terminated.\n", pid);
    return;
}
```

D.7.2 slsSServer.cpp

```
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "security.h"
#include "md5ADT.h"
#include "bst.h"
#include "sp.h"
```

```
#define PROTOPORT    5000    /* default slsServer port number    */
#define QLEN         6       /* size of request queue            */

BstPrint sngFree;
BstPrint fn;
BstList fnList;
void sig_chld(int signo);

int main(int argc, char * argv[]){
    struct    hostent  *ptrh;  /* ptr to a host table entry      */
    struct    protoent *ptrp;  /* ptr to a protocol table entry  */
    struct    sockaddr_in sad; /* struct for server's address    */
    struct    sockaddr_in cad; /* struct for client's address    */
    int       sd, sd2;         /* socket descriptors             */
    int       port;            /* protocol port number           */
    socklen_t alen;
    char      buf[1000];       /* buffer for str server sends    */
    char      msg_buf[1000];
    int       visit = 0;
    int       keyIndex;
    char      userLine[1000];

    FILE      *fPtr;
    char      username[128] ;
    char      password[128];
    int       flag = 0;
    int       n, i, pid, passLen, nameLen;

    char      ID[100];
    int       ServicePort;
    char      Server[100];
    char      ServicePath[100];
    int       Cost;
    char      Restriction[10];
    char      localhostname[10];
    char      filename[20];

#ifdef WIN32
    WSADATA wsaData;
    WSAStartup(0x0101, &wsaData);
#endif
    memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr struct */
    sad.sin_family = AF_INET;           /* set  to Internet      */
    sad.sin_addr.s_addr = INADDR_ANY;   /* set local IP add      */

    if (argc > 2){
        port = atoi(argv[2]);
    } else {
        port = PROTOPORT;        /* use default port number      */
    }
    sad.sin_port = htons((u_short)port);

    /* Map TCP transport protocol name to protocol number */
    if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
        fprintf(stderr, "cannot map \"tcp\" to protocol number");
        exit(1);
    }

    /* Create a socket */
    sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
    if (sd < 0) {
        fprintf(stderr, "socket creation failed\n");
        exit(1);
    }

    /* Bind a local address to the socket */
    if (bind(sd, (struct sockaddr *)&sad,
        (socklen_t)(sizeof(sad))) < 0) {
        fprintf(stderr,"bind failed\n");
        exit(1);
    }

    FILE * spFile;
    gethostname(localhostname, 10);
    sprintf (filename, "%sSP.data", localhostname);

    spFile = fopen(filename, "r");

    Bst * spTree = BstConstruct(spCompare);
    sp * SP_node = spCreate();
    sp * SP_search = spCreate();
    sp * SP_result;

    fscanf(spFile,"%s", ID);
    while (strcmp(ID, "end") != 0){
        fscanf(spFile, "%i %s %s %i %s",
            &ServicePort, Server, ServicePath, &Cost, Restriction);
        setID (SP_node, ID);
        setServicePort(SP_node, ServicePort);
        setServer (SP_node, Server);
```

```
    setServicePath (SP_node, ServicePath);
    setCost (SP_node, Cost);
    setRestriction (SP_node, Restriction);
    BstInsert(spTree, SP_node);
    SP_node = spCreate();
    fscanf(spFile,"%s", ID);
}

/* Specify size of request queue */
if (listen(sd, QLEN) < 0) {
    fprintf(stderr,"listen failed\n");
    exit(1);
}

signal(SIGCHLD, sig_chld);

/* Main server loop - accept and handle requests */
alen = (socklen_t)(sizeof(cad));
if ( (sd2=accept(sd, (struct sockaddr *)&cad, &alen)) < 0){
    fprintf(stderr, "accept failed\n");
    exit(1);
}

/* Set the encr/decr key is specified as argv[3] */
if (argc > 4){
    keyIndex = atoi(argv[4]);
} else {
    keyIndex = 15;
}

/*receive request from client*/
msg_buf[0] = 0;
while(n=read(sd2, buf, sizeof(buf))){
    decRoundRobin_len_all(buf, n, keyIndex);
    buf[n] =0;
    strcat(msg_buf, buf);
    if (buf[n-1] == '#') {
        msg_buf[strlen(msg_buf)-1] = 0;
        break;
    }
}
sscanf(msg_buf, "%[^+]%*c%[^\n]", username, userLine);

/* check authentication token.
 *     Get token from file.
 *         keyIndex specified by dateAppServer when execl().
 */
    if (strcmp(username, argv[3]) == 0){
        setID (SP_search, userLine);
    SP_result = (sp*)BstSearch(spTree, SP_search);
    if (SP_result != NULL){
            sprintf(buf, "Service ID  = %s\n"
                         "ServicePort = %i\n"
                         "Server Name = %s\n"
                         "Service Path = %s\n"
                         "Service Cost = %i\n"
                         "Service Restriction = %s\n#",
                         getID (SP_result),
                    getServicePort (SP_result),
                    getServer (SP_result),
             getServicePath (SP_result),
             getCost (SP_result),
             getRestriction (SP_result));
            encRoundRobin_len_all(buf, strlen(buf), keyIndex);
            write(sd2, buf, strlen(buf));
        } else {
        sprintf(buf, "Service not available!\n##");
            encRoundRobin_len_all(buf, strlen(buf), keyIndex);
            write(sd2, buf, strlen(buf));
        }
    } else {
        printf ("Request format error!\n");
        sprintf(msg_buf, "Request format error!\n
            Service request unknown!\n##");
    encRoundRobin_len_all(msg_buf, strlen(msg_buf), keyIndex);
        write(sd2, msg_buf, strlen(msg_buf));
    }
    BstTraverse(spTree, sngFree, in);
    BstDestroy(spTree);
    close(sd2);
}
void sngFree(const void* item){
    sp* SP = (sp*)item;
    free(SP);
    return;
}
void fn(const void * item){
    sp * SP = (sp*)item;
    printf("Service in BST node = %s\n",  getID(SP));
```

```
}

void fnList(const void * item, int sockNum){
    sp* SP = (sp*)item;
    char buf[500];

    sprintf(buf, "Service ID  = %s\n" "ServicePort = %i\n"
                 "Server Name = %s\n"
                 "Service Path = %s\n" "Service Cost = %i\n"
                 "Service Restriction = %s\n\n",
                 getID(SP), getServicePort (SP), getServer (SP),
                 getServicePath (SP), getCost (SP),
                 getRestriction (SP));
    encRoundRobin_len_all(buf, strlen(buf), 15);
    write(sockNum, buf, strlen(buf));
}

void sig_chld(int signo){
    int pid;
    int stat;
    while((pid = waitpid(-1, &stat, WNOHANG)) > 0)
        printf("child %d terminated.\n", pid);
    return;
}
```

D.7.3 slsCGroup.cpp

```
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "security.h"
#include "md5ADT.h"

#define PROTOPORT    6005    /* default timeServer port number   */
#define QLEN         6       /* size of request queue            */

void sig_chld(int signo);

int main(){
    struct    hostent  *ptrh;  /* ptr to a host table entry      */
    struct    protoent *ptrp;  /* ptr to a protocol table entry  */
    struct    sockaddr_in sad; /* struct for server's address    */
    struct    sockaddr_in cad; /* struct for client's address    */
    int       sd, sd2;         /* socket descriptors             */
    int       port;            /* protocol port number           */
    socklen_t alen;
    char      buf[1000];       /* buffer for str server sends    */
    char      port_buf[100];
    char      msg_buf[1000];
    char      keyBuf[10];
    char      userID[500];
    char      userToken [500];
    int       rNum;
    int       visit = 0;
    time_t    time_now;
    char      time_str[50];

    FILE      *fPtr;
    char      username[128] ;
    char      password[128];
    int       flag = 0;
    int       n, i, pid, passLen, nameLen, buf_len;
    int       keyIndex;
    /* generate keyIndex, send to AS and server */

#ifdef WIN32
    WSADATA wsaData;
    WSAStartup(0x0101, &wsaData);
#endif
    memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr struct */
    sad.sin_family = AF_INET;           /* set to Internet       */
    sad.sin_addr.s_addr = INADDR_ANY;   /* set local IP add      */
    port = PROTOPORT;                   /* use default port no.  */
    sad.sin_port = htons((u_short)port);
```

```
/* Map TCP transport protocol name to protocol number */
if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
    fprintf(stderr, "cannot map \"tcp\" to protocol number");
    exit(1);
}

/* Create a socket */
sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
if (sd < 0) {
    fprintf(stderr, "socket creation failed\n");
    exit(1);
}

/* Bind a local address to the socket */
if (bind(sd, (struct sockaddr *)&sad,
    (socklen_t)(sizeof(sad))) < 0) {
    fprintf(stderr,"bind failed\n");
    exit(1);
}

/* Specify size of request queue */
if (listen(sd, QLEN) < 0) {
    fprintf(stderr,"listen failed\n");
    exit(1);
}

/* To avoid zombie */
signal(SIGCHLD, sig_chld);

/* Main server loop - accept and handle requests */
while (1) {
    alen = (socklen_t)(sizeof(cad));
    if ( (sd2=accept(sd, (struct sockaddr *)&cad, &alen)) < 0){
        fprintf(stderr, "accept failed\n");
        exit(1);
    }
    visit++;

    /* fork() to handle multiple connections */
    if ((pid = fork()) == 0) {
        close(sd); // child not listening

        /* read all from client */
    msg_buf[0] = 0;
            while(n=read(sd2, buf, sizeof(buf))){
                decRoundRobin_len_all(buf, n, 15);
                buf[n] =0;
                strcat(msg_buf, buf);
                if (buf[n-1] == '#') {
                    msg_buf[strlen(msg_buf)-1] = 0;
                    break;
                }
            }
            sscanf(msg_buf, "%[^+]%*c%s]", userID, keyBuf);
            srand(time(NULL));
            rNum = random()%999;
            /* Create port */
            sprintf(buf, "%i", 11000+rNum);
            strcpy(port_buf, buf);
            buf_len = strlen(buf);
            buf[buf_len] = '#';
            buf[buf_len+1] = 0;
            encRoundRobin_len_all(buf,strlen(buf), 15);
            write(sd2, buf, strlen(buf));
            printf ("userID: %s\nkeyIndex: %s\nPort: %s\n", userID,
                     keyBuf, port_buf);
            /* Call the service */
            execl("/home/linux/slsCServer", "./slsCServer",
                "localhost", port_buf, userID, keyBuf, NULL);
            exit(0);
        }   // fork()
        close(sd2);
    }
}

void sig_chld(int signo){
    int pid;
    int stat;
    while((pid = waitpid(-1, &stat, WNOHANG)) > 0)
        printf("child %d terminated.\n", pid);
    return;
}
```

D.7.4 slsCServer.cpp

```
#ifndef unix
#define WIN32
#include <windows.h>
```

```
#include <winsock.h>
#else

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "security.h"
#include "md5ADT.h"
#include "bst.h"
#include "sp.h"

#define PROTOPORT    5000    /* default slsServer port number    */
#define QLEN         6       /* size of request queue            */

BstPrint fn;
BstList fnList;
void sig_chld(int signo);

int main(int argc, char * argv[]){
    struct    hostent  *ptrh;  /* ptr to a host table entry      */
    struct    protoent *ptrp;  /* ptr to a protocol table entry  */
    struct    sockaddr_in sad; /* struct for server's address    */
    struct    sockaddr_in cad; /* struct for client's address    */
    int       sd, sd2;         /* socket descriptors             */
    int       port;            /* protocol port number           */
    socklen_t alen;
    char      buf[1000];       /* buffer for str server sends    */
    char      msg_buf[1000];
    int       visit = 0;
    int       keyIndex;
    char      userLine[1000];

    FILE     *fPtr;
    char     username[128] ;
    char     password[128];
    int      flag = 0;
    int      n, i, pid, passLen, nameLen;

    char     ID[100];
    int      ServicePort;
    char     Server[100];
    char     ServicePath[100];
    int      Cost;
    char     Restriction[10];
    char     localhostname[10];
    char     filename[20];

#ifdef WIN32
    WSADATA wsaData;
    WSAStartup(0x0101, &wsaData);
#endif
    memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr struct */
    sad.sin_family = AF_INET;           /* set to Internet       */
    sad.sin_addr.s_addr = INADDR_ANY;   /* set local IP add      */
    if (argc > 2){
        port = atoi(argv[2]);
    } else {
        port = PROTOPORT;        /* use default port number      */
    }

    sad.sin_port = htons((u_short)port);

    /* Map TCP transport protocol name to protocol number */
    if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
        fprintf(stderr, "cannot map \"tcp\" to protocol number");
        exit(1);
    }

    /* Create a socket */
    sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
    if (sd < 0) {
        fprintf(stderr, "socket creation failed\n");
        exit(1);
    }

    /* Bind a local address to the socket */
    if (bind(sd, (struct sockaddr *)&sad,
        (socklen_t)(sizeof(sad))) < 0) {
        fprintf(stderr,"bind failed\n");
        exit(1);
    }
```

```
FILE * spFile;
gethostname(localhostname, 10);
sprintf (filename, "%sSP.data", localhostname);

spFile = fopen(filename, "r");

Bst * spTree = BstConstruct(spCompare);
sp * SP_node = spCreate();
sp * SP_search = spCreate();
sp * SP_result;

fscanf(spFile,"%s", ID);
while (strcmp(ID, "end") != 0){
    fscanf(spFile, "%i %s %s %i %s",
        &ServicePort, Server, ServicePath, &Cost, Restriction);
    setID (SP_node, ID);
    setServicePort(SP_node, ServicePort);
    setServer (SP_node, Server);
    setServicePath (SP_node, ServicePath);
    setCost (SP_node, Cost);
    setRestriction (SP_node, Restriction);
    BstInsert(spTree, SP_node);
    SP_node = spCreate();
    fscanf(spFile,"%s", ID);
}

/* Specify size of request queue */
if (listen(sd, QLEN) < 0) {
    fprintf(stderr,"listen failed\n");
    exit(1);
}

signal(SIGCHLD, sig_chld);

/* Main server loop - accept and handle requests */
alen = (socklen_t)(sizeof(cad));
if ( (sd2=accept(sd, (struct sockaddr *)&cad, &alen)) < 0){
    fprintf(stderr, "accept failed\n");
    exit(1);
}

/* Set the encr/decr key is specified as argv[4] */
if (argc > 4){
    keyIndex = atoi(argv[4]);
    } else {
        keyIndex = 15;
    }

    /*receive request from client*/
    msg_buf[0] = 0;
    while(n=read(sd2, buf, sizeof(buf))){
        decRoundRobin_len_all(buf, n, keyIndex);
        buf[n] =0;
        strcat(msg_buf, buf);
        if (buf[n-1] == '#') {
            msg_buf[strlen(msg_buf)-1] = 0;
            break;
        }
    }
    sscanf(msg_buf, "%[^#]", username);
    /* check authentication token.
     *     Get token from file.
     *         keyIndex specified by dateAppServer when execl().
     */
    if (strcmp(username, argv[3]) == 0){
        BstDump(spTree, fnList, sd2, in, keyIndex);
        buf[0] = '#';
        encRoundRobin_len_all(buf,2, keyIndex);
        write(sd2, buf, 2);
    } else {
        printf ("Request format error!\n");
        sprintf(msg_buf, "Request format error!\n
            Service request unknown!\n##");
        encRoundRobin_len_all(msg_buf, strlen(msg_buf), keyIndex);
        write(sd2, msg_buf, strlen(msg_buf));
    }
    close(sd2);
}

void fn(const void * item){
    sp * SP = (sp*)item;
    printf("Service in BST node = %s\n",  getID(SP));

}

void fnList(const void * item, int sockNum, int keyIndex){
    sp* SP = (sp*)item;
    char buf[500];
```

```
    sprintf(buf, "Service ID  = %s\n" "ServicePort = %i\n"
                 "Server Name = %s\n"
                 "Service Path = %s\n" "Service Cost = %i\n"
                 "Service Restriction = %s\n\n",
                 getID(SP), getServicePort (SP), getServer (SP),
                 getServicePath (SP), getCost (SP),
                 getRestriction (SP));
    encRoundRobin_len_all(buf, strlen(buf), keyIndex);
    write(sockNum, buf, strlen(buf));
}

void sig_chld(int signo){
    int pid;
    int stat;
    while((pid = waitpid(-1, &stat, WNOHANG)) > 0)
        printf("child %d terminated.\n", pid);
    return;
}
```

D.8 Utility Files

D.8.1 GNUmakefile

```
# NOTE:
# Many files in this GNUmakefile will not be used for the actual
#     implementation.
# They are used for testing individual servers only.
# Examples: dateGroup  is tested with dateGPClient
#           dateServer is tested with dateClient
# Only dateGroup and dateServer will be used in the final
#    implementation. dateGPClient and dateClient will not
#    be used in the final implementation.
#

all: slsSServer slsCServer asServer asClient timeServer timeClient
     timeGroup timeGPClient dateServer dateClient dateGroup
     dateGPClient echoServer echoClient echoGroup echoGPClient
     nameServer nameClient nameGroup nameGPClient IPServer
     userDP_generate userHDP_generate dict combinedict exec
     md5_app security_test sp_test  slsCClient slsSClient slsCGroup
     slsCGPClient slsSGroup slsSGPClient sngCreate

clean:
    /bin/rm -f *.o

IPServer : IPServer.o security.o md5ADT.o
    g++  IPServer.o security.o md5ADT.o -o IPServer

slsCClient : slsCClient.o security.o md5ADT.o bst.o sp.o
    g++  slsCClient.o security.o md5ADT.o bst.o sp.o -o slsCClient

slsSClient : slsSClient.o security.o md5ADT.o bst.o sp.o
    g++  slsSClient.o security.o md5ADT.o bst.o sp.o -o slsSClient

timeClient : timeClient.o security.o md5ADT.o bst.o sp.o
    g++  timeClient.o security.o md5ADT.o bst.o sp.o -o timeClient

dateClient : dateClient.o security.o md5ADT.o bst.o sp.o
    g++  dateClient.o security.o md5ADT.o bst.o sp.o -o dateClient

echoClient : echoClient.o security.o md5ADT.o bst.o sp.o
    g++  echoClient.o security.o md5ADT.o bst.o sp.o -o echoClient

asClient : asClient.o security.o md5ADT.o bst.o sp.o
    g++  asClient.o security.o md5ADT.o bst.o sp.o -o asClient

asServer : asServer.o security.o md5ADT.o bst.o sp.o path.o
    g++ asServer.o security.o md5ADT.o bst.o sp.o path.o -o
        asServer

slsCServer : slsCServer.o security.o md5ADT.o bst.o sp.o
    g++ slsCServer.o security.o md5ADT.o bst.o sp.o -o slsCServer

slsSServer : slsSServer.o security.o md5ADT.o bst.o sp.o
    g++ slsSServer.o security.o md5ADT.o bst.o sp.o -o slsSServer

timeServer : timeServer.o security.o md5ADT.o bst.o sp.o
    g++ timeServer.o security.o md5ADT.o bst.o sp.o -o timeServer

dateServer : dateServer.o security.o md5ADT.o bst.o sp.o
    g++ dateServer.o security.o md5ADT.o bst.o sp.o -o dateServer

echoServer : echoServer.o security.o md5ADT.o bst.o sp.o
    g++ echoServer.o security.o md5ADT.o bst.o sp.o -o echoServer

nameServer : nameServer.o security.o md5ADT.o bst.o sp.o
```

```
    g++ nameServer.o security.o md5ADT.o bst.o sp.o -o nameServer

nameClient : nameClient.o security.o md5ADT.o bst.o sp.o
    g++ nameClient.o security.o md5ADT.o bst.o sp.o -o nameClient

dateGroup : dateGroup.o security.o md5ADT.o bst.o sp.o
    g++ dateGroup.o security.o md5ADT.o bst.o sp.o -o dateGroup

slsCGroup : slsCGroup.o security.o md5ADT.o bst.o sp.o
    g++ slsCGroup.o security.o md5ADT.o bst.o sp.o -o slsCGroup

slsSGroup : slsSGroup.o security.o md5ADT.o bst.o sp.o
    g++ slsSGroup.o security.o md5ADT.o bst.o sp.o -o slsSGroup

slsCGPClient : slsCGPClient.o security.o md5ADT.o bst.o sp.o
    g++ slsCGPClient.o security.o md5ADT.o bst.o sp.o -o
        slsCGPClient

slsSGPClient : slsSGPClient.o security.o md5ADT.o bst.o sp.o
    g++ slsSGPClient.o security.o md5ADT.o bst.o sp.o -o
        slsSGPClient

dateGPClient : dateGPClient.o security.o md5ADT.o bst.o sp.o
    g++ dateGPClient.o security.o md5ADT.o bst.o sp.o -o
        dateGPClient

userDP_generate : userDP_generate.o security.o md5ADT.o
    g++ userDP_generate.o security.o md5ADT.o -o userDP_generate

userHDP_generate : userHDP_generate.o security.o md5ADT.o
    g++ userHDP_generate.o security.o md5ADT.o -o userHDP_generate

userDP_generate.o : userDP_generate.cpp
    g++ -c userDP_generate.cpp

sngCreate : sngCreate.o bst.o
    g++ sngCreate.o bst.o -o sngCreate

sngCreate.o : sngCreate.cpp
    g++ -c sngCreate.cpp

userHDP_generate.o : userHDP_generate.cpp
    g++ -c userHDP_generate.cpp

dict : dict.o
    g++ dict.o -o dict

dict.o : dict.cpp
    g++ -c dict.cpp

combinedict : combinedict.o
    g++ combinedict.o -o combinedict

combinedict.o : combinedict.cpp
    g++ -c combinedict.cpp

exec : exec.o
    g++ exec.o -o exec

slsCGroup.o : slsCGroup.cpp
    g++ -c slsCGroup.cpp

slsSGroup.o : slsSGroup.cpp
    g++ -c slsSGroup.cpp

slsCGPClient.o : slsCGPClient.cpp
    g++ -c slsCGPClient.cpp

slsSGPClient.o : slsSGPClient.cpp
    g++ -c slsSGPClient.cpp

exec.o : exec.cpp
    g++ -c exec.cpp

md5_app : md5_app.o md5ADT.o
    g++ md5_app.o md5ADT.o -o md5_app

md5_app.o : md5_app.cpp
    g++ -c md5_app.cpp

security_test : security_test.o security.o md5ADT.o
    g++ security_test.o security.o md5ADT.o -o security_test

security_test.o : security_test.cpp
    g++ -c security_test.cpp

sp_test : sp_test.o sp.o
    g++ sp_test.o sp.o -o sp_test

sp_test.o : sp_test.cpp
```

```
    g++ -c sp_test.cpp

IPServer.o : IPServer.cpp
    g++ -c IPServer.cpp

dateGPClient.o : dateGPClient.cpp
    g++ -c dateGPClient.cpp

dateGroup.o : dateGroup.cpp
    g++ -c dateGroup.cpp

timeGroup : timeGroup.o security.o md5ADT.o bst.o sp.o
    g++ timeGroup.o security.o md5ADT.o bst.o sp.o -o timeGroup

timeGPClient : timeGPClient.o security.o md5ADT.o bst.o sp.o
    g++ timeGPClient.o security.o md5ADT.o bst.o sp.o -o
        timeGPClient

timeGPClient.o : timeGPClient.cpp
    g++ -c timeGPClient.cpp

timeGroup.o : timeGroup.cpp
    g++ -c timeGroup.cpp

echoGroup : echoGroup.o security.o md5ADT.o bst.o sp.o
    g++ echoGroup.o security.o md5ADT.o bst.o sp.o -o echoGroup

slsGroup : slsGroup.o security.o md5ADT.o bst.o sp.o
    g++ slsGroup.o security.o md5ADT.o bst.o sp.o -o slsGroup

echoGPClient : echoGPClient.o security.o md5ADT.o bst.o sp.o
    g++ echoGPClient.o security.o md5ADT.o bst.o sp.o -o
        echoGPClient

slsGPClient : slsGPClient.o security.o md5ADT.o bst.o sp.o
    g++ slsGPClient.o security.o md5ADT.o bst.o sp.o -o slsGPClient

echoGPClient.o : echoGPClient.cpp
    g++ -c echoGPClient.cpp

echoGroup.o : echoGroup.cpp
    g++ -c echoGroup.cpp

slsGPClient.o : slsGPClient.cpp
    g++ -c slsGPClient.cpp

slsGroup.o : slsGroup.cpp
    g++ -c slsGroup.cpp

nameGroup : nameGroup.o security.o md5ADT.o bst.o sp.o
    g++ nameGroup.o security.o md5ADT.o bst.o sp.o -o nameGroup

nameGPClient : nameGPClient.o security.o md5ADT.o bst.o sp.o
    g++ nameGPClient.o security.o md5ADT.o bst.o sp.o -o
        nameGPClient

nameGPClient.o : nameGPClient.cpp
    g++ -c nameGPClient.cpp

nameGroup.o : nameGroup.cpp
    g++ -c nameGroup.cpp

nameClient.o : nameClient.cpp
    g++ -c nameClient.cpp

nameServer.o : nameServer.cpp
    g++ -c nameServer.cpp

slsCClient.o : slsCClient.cpp
    g++ -c slsCClient.cpp

slsSClient.o : slsSClient.cpp
    g++ -c slsSClient.cpp

timeClient.o : timeClient.cpp
    g++ -c timeClient.cpp

dateClient.o : dateClient.cpp
    g++ -c dateClient.cpp

echoClient.o : echoClient.cpp
    g++ -c echoClient.cpp

asClient.o : asClient.cpp
    g++ -c asClient.cpp

asServer.0 : asServer.cpp
    g++ -c asServer.cpp

slsCServer.o : slsCServer.cpp
```

```
    g++ -c slsCServer.cpp

slsSServer.o : slsSServer.cpp
    g++ -c slsSServer.cpp

timeServer.o : timeServer.cpp
    g++ -c timeServer.cpp

dateServer.o : dateServer.cpp
    g++ -c dateServer.cpp

echoServer.o : echoServer.cpp
    g++ -c echoServer.cpp

security.o : security.cpp
    g++ -c security.cpp

md5ADT.o : md5ADT.cpp
    g++ -c md5ADT.cpp

bst.o : bst.cpp
    g++ -c bst.cpp

path.o : path.cpp
    g++ -c path.cpp

sp.o : sp.cpp
    g++ -c sp.cpp
```

D.8.2 security.h

```
#ifndef __ENCRYPTION_H
#define __ENCRYPTION_H

// isPrint() to check. rely on '\0' to stop
bool encRoundRobin(char * message, int key);
bool decRoundRobin(char * message, int key);
bool encSubstitution(char * message, int tableNum);
bool decSubstitution(char * message, int tableNum);

int md5ToKey(char *);
int md5ToTableNum(char*);

// As the srand() is seeded with time, calling staticDPS()
// more than once within a program is not advisable.
// It can be used with each run of a program separated at
// least by 1 second.
void staticDPS(int len, char * password);
void dynamicDPS(char * filename, char * passowrd);
void msgOne(char* R1, char* SP, char* msg_one, int* msg_len);
void msgTwo(char* R1, char* R2, char* DP, char* msg_two,
            int* msg_len);
void msgThree(char* R2, char* R3, char* DP, char* msg_three,
            int* msg_len);
void parseMsgOne(char* R1, char* SP, char* msg_one);
void parseMsgTwo(char* R1, char* R2, char* DP, char* msgTwo);
void parseMsgThree(char* R2, char* R3, char* DP, char* msgThree);

// isPrint() to check. len specified.
bool encRoundRobin_len(char * message, int len, int key);
bool decRoundRobin_len(char * message, int len, int key);
bool encSubstitution_len(char * message, int len, int tableNum);
bool decSubstitution_len(char * message, int len, int tableNum);

int md5ToKey_len(char *, int len);
int md5ToTableNum_len(char*, int len);

// '`' substitute '\n'. isPrint() to check. len specified.
bool encRoundRobin_len_all(char * message, int len, int key);
bool decRoundRobin_len_all(char * message, int len, int key);

#endif
```

D.8.3 security.cpp

```
#include <string.h>
#include <ctype.h>
#include <time.h>
#include <stdlib.h>
#include <stdio.h>

#include "security.h"
#include "md5ADT.h"

// Tables randomly generated.
char etable1[95] = {89,73,113,106,124,118,66,96,107,80,83,70,82,
```

```
                    119,112,94,84,114,61,45,63,47,48,95,117,40,109,
                    33,77,116,51,55,108,103,44,37,87,100,105,72,86,
                    49,126,76,43,58,68,41,53,97,93,62,38,123,90,
                    110,74,59,64,91,101,52,42,54,69,39,71,125,57,
                    32,81,120,34,60,50,98,78,65,46,56,88,92,115,
                    102,36,121,79,35,75,99,85,111,67,104,122};
char dtable1[95] = {101,59,104,119,116,67,84,97,57,79,94,76,66,51,
                    110,53,54,73,106,62,93,80,95,63,111,100,77,89,
                    105,50,83,52,90,109,38,124,78,96,43,98,71,33,
                    88,120,75,60,108,118,41,102,44,42,48,122,72,68,
                    112,32,86,91,113,82,47,55,39,81,107,121,69,92,
                    115,65,125,70,35,40,64,58,87,123,46,34,49,114,
                    61,56,37,45,103,117,126,85,36,99,74};
char etable2[95] = {123,37,99,48,96,75,103,83,56,70,105,89,47,60,
                    40,107,66,45,51,58,90,69,63,55,115,36,109,43,
                    77,111,80,82,42,67,120,39,94,35,32,100,126,
                    117,88,91,113,87,61,98,93,85,81,95,112,119,
                    121,38,97,116,108,102,68,33,71,79,124,114,101,
                    78,41,52,74,54,53,92,50,118,125,57,73,76,84,65,
                    86,64,122,34,46,62,59,104,72,49,44,110,106};
char dtable2[95] = {70,93,117,69,57,33,87,67,46,100,64,59,124,49,
                    118,44,35,123,106,50,101,104,103,55,40,109,51,
                    120,45,78,119,54,115,113,48,65,92,53,41,94,122,
                    110,102,37,111,60,99,95,62,82,63,39,112,81,114,
                    77,74,43,52,75,105,80,68,83,36,88,79,34,71,98,
                    91,38,121,42,126,47,90,58,125,61,84,76,97,56,
                    89,73,107,85,66,86,116,32,96,108,72};
char etable3[95] = {72,47,55,94,79,70,89,67,80,122,126,64,113,92,
                    108,60,58,87,115,106,117,102,78,111,35,65,83,
                    82,121,84,32,114,34,73,48,56,44,52,74,97,105,
                    107,50,61,125,118,68,62,37,109,33,119,91,43,49,
                    51,85,104,69,53,63,101,90,59,77,88,39,93,66,99,
                    112,57,41,110,81,71,46,98,45,75,36,42,103,116,
                    86,76,96,120,38,123,40,100,54,95,124};
char dtable3[95] = {62,82,64,56,112,80,120,98,122,104,113,85,68,
                    110,108,33,66,86,74,87,69,91,124,34,67,103,48,
                    95,47,75,79,92,43,57,100,39,78,90,37,107,32,65,
                    70,111,117,96,54,36,40,106,59,58,61,88,116,49,
                    97,38,94,84,45,99,35,125,118,71,109,101,123,93,
                    53,114,89,72,51,73,46,81,105,55,102,44,63,50,
                    115,52,77,83,119,60,41,121,126,76,42};
char etable4[95] = {79,99,39,41,61,118,96,111,42,32,67,71,93,65,84,
                    82,57,75,102,46,77,59,43,116,66,36,126,87,90,
                    86,101,124,120,97,68,92,119,50,44,104,112,74,
                    33,117,51,76,49,80,94,121,108,48,34,70,91,122,
                    63,109,106,73,45,64,60,114,56,38,78,105,53,55,
                    85,40,72,95,69,110,100,47,81,98,125,83,37,58,
                    89,35,113,123,62,52,54,88,115,103,107};
char dtable4[95] = {41,74,84,117,57,114,97,34,103,35,40,54,70,92,
                    51,109,83,78,69,76,121,100,122,101,96,48,115,
                    53,94,36,120,88,93,45,56,42,66,106,85,43,104,
                    91,73,49,77,52,98,32,79,110,47,113,46,102,61,
                    59,123,116,60,86,67,44,80,105,38,65,111,33,108,
                    62,50,125,71,99,90,126,82,89,107,39,72,118,95,
                    124,55,75,37,68,64,81,87,119,63,112,58};
char etable5[95] = {69,48,121,52,56,78,112,67,57,97,58,115,113,41,
                    117,47,107,70,38,63,53,71,94,123,96,86,44,125,
                    35,99,36,83,95,88,46,106,122,68,89,119,80,64,
                    116,37,90,49,50,32,87,62,73,60,111,33,72,120,
                    92,104,109,118,85,82,103,105,124,101,108,91,77,
                    114,93,126,55,43,110,84,75,76,45,79,61,39,65,
                    100,51,34,74,59,98,81,42,40,66,102,54};
char dtable5[95] = {79,85,117,60,62,75,50,113,123,45,122,105,58,
                    110,66,47,33,77,78,116,35,52,126,104,36,40,42,
                    119,83,112,81,51,73,114,124,39,69,32,49,53,86,
                    82,118,108,109,100,37,111,72,121,93,63,107,92,
                    57,80,65,70,76,99,88,102,54,64,56,41,120,61,
                    115,97,125,94,89,95,67,48,98,90,106,84,38,44,
                    101,43,74,46,91,71,87,34,68,55,96,59,103};

char* encTable[5] = {etable1,etable2,etable3,etable4,etable5};
char* decTable[5] = {dtable1,dtable2,dtable3,dtable4,dtable5};

bool encRoundRobin(char * message, int key){
    int  change = key % 26;
    int  i, len = strlen(message);
    char msgChar;

    for (i=0; i<len; i++){
        msgChar = message[i];
        if (isprint(msgChar)){
            // printable ASCII 32 - 126 total of 95 char.
            message[i] = (((msgChar-32) + change) % 95) + 32;
        } else {
            return false;
        }
    }
    return true;
}
bool encRoundRobin_len(char * message, int len, int key){
```

```
    int  change = key % 26;
    int  i;
    char msgChar;

    for (i=0; i<len; i++){
        msgChar = message[i];
        if (isprint(msgChar)){
            // printable ASCII 32 - 126 total of 95 char.
            message[i] = (((msgChar-32) + change) % 95) + 32;
        } else {
            return false;
        }
    }
    return true;
}
bool encRoundRobin_len_all(char * message, int len, int key){
    int  change = key % 26;
    int  i;
    char msgChar;

    for (i=0; i<len; i++){
        msgChar = message[i];
        if (isprint(msgChar)){
            // printable ASCII 32 - 126 total of 95 char.
            message[i] = (((msgChar-32) + change) % 95) + 32;
        }
    }
    return true;
}

bool decRoundRobin(char * message, int key){
    int  change = key % 26;
    int  i, len = strlen(message);
    char msgChar;

    for (i=0; i<len; i++){
        msgChar = message[i];
        if (isprint(msgChar)){
            // printable ASCII 32 - 126 total of 95 char.
            message[i] = (((msgChar-32) + 95 - change) % 95) + 32;
        } else {
            return false;
        }
    }
    return true;
}

bool decRoundRobin_len(char * message, int len, int key){
    int  change = key % 26;
    int  i;
    char msgChar;

    for (i=0; i<len; i++){
        msgChar = message[i];
        if (isprint(msgChar)){
            // printable ASCII 32 - 126 total of 95 char.
            message[i] = (((msgChar-32) + 95 - change) % 95) + 32;
        } else {
            return false;
        }
    }
    return true;

}
bool decRoundRobin_len_all(char * message, int len, int key){
    int  change = key % 26;
    int  i;
    char msgChar;

    for (i=0; i<len; i++){
        msgChar = message[i];
        if (isprint(msgChar)){
            // printable ASCII 32 - 126 total of 95 char.
            message[i] = (((msgChar-32) + 95 - change) % 95) + 32;
        }
    }
    return true;

}

bool encSubstitution(char * message, int tableNum){
    int  i, len = strlen(message);
    char msgChar;

    for (i=0; i<len; i++){
        msgChar = message[i];
        if (isprint(msgChar)){
            // printable ASCII 32 - 126 total of 95 char.
            message[i] = *(encTable[tableNum-1] + (msgChar-32));
```

```
        } else {
            return false;
        }
    }
    return true;
}
bool encSubstitution_len(char * message, int len, int tableNum){
    int  i;
    char msgChar;

    for (i=0; i<len; i++){
        msgChar = message[i];
        if (isprint(msgChar)){
            // printable ASCII 32 - 126 total of 95 char.
            message[i] = *(encTable[tableNum-1] + (msgChar-32));
        } else {
            return false;
        }
    }
    return true;
}

bool decSubstitution(char * message, int tableNum){
    int  i, len = strlen(message);
    char msgChar;

    for (i=0; i<len; i++){
        msgChar = message[i];
        if (isprint(msgChar)){
            // printable ASCII 32 - 126 total of 95 char.
            message[i] = *(decTable[tableNum-1] + (msgChar-32));
        } else {
            return false;
        }
    }
    return true;
}
bool decSubstitution_len(char * message, int len, int tableNum){
    int  i;
    char msgChar;

    for (i=0; i<len; i++){
        msgChar = message[i];
        if (isprint(msgChar)){
            // printable ASCII 32 - 126 total of 95 char.
            message[i] = *(decTable[tableNum-1] + (msgChar-32));
        } else {
            return false;
        }
    }
    return true;
}

int md5ToKey(char * hash){
    int i, result = 0, len = strlen(hash);

    for (i=0; i<len; i++){
        result += hash[i];
    }
    return (result % 94) +1;
}

int md5ToTableNum(char* hash){
    int i, result = 0, len = strlen(hash);

    for (i=0; i<len; i++){
        result += hash[i];
    }
    return (result % 5) + 1;
}

void staticDPS(int len, char * password){
    int i, j, repeat = 1, randNum;
    char randChar;

    srand(time(NULL));
    for (i=0; i<len; i++){
        repeat = 1;
        randNum = rand() % 95;
        randChar = randNum + 32;

        for (j=0; (j<i) && (repeat == 1); j++){
            if (randChar == password[j]){
                repeat = 0;
                break;
            }
        }
        if (repeat) password[i] = randChar;
        else i--;
    }
```

```
    password[len] = 0;
}

/*
    filename is the file of [potential passwords.
    the first line is the number of choices.
 */
void dynamicDPS(char * filename, char * password){
    int i, count;
    FILE * datafile;
    int randNum;

    datafile = fopen(filename, "r");
    fscanf (datafile, "%i", &count);
    srand(time(NULL));
    randNum = rand() % count;

    for (i=0; i<randNum-1; i++){
        fscanf (datafile, "%*s");
    }
    fscanf(datafile, "%s", password);
    fclose(datafile);
    return;
}
void msgOne(char* R1, char* SP, char* msgOne, int* len){
    int key;
    char SPDigest[100];

    // generate R1
    staticDPS(32, R1);
    R1[32] = '\0';
    printf("R1 generated: %s\n", R1);

    // form H(SP)
    md5_action(SP, SPDigest);

    // form {R1}H(SP)
    key = md5ToKey(SPDigest);
    strcpy(msgOne, R1);
    msgOne[32] = '#';
    msgOne[32] = '\0';
    *len = strlen(msgOne);
    encRoundRobin_len_all(msgOne, *len, key);
}
```

```
void parseMsgOne(char* R1, char* HSP, char* msgOne){
    int len, key;
    char SPDigest[100];

    key = md5ToKey(HSP);
    len = strlen(msgOne);
    decRoundRobin_len_all(msgOne, len, key);
    strcpy(R1, msgOne);
}

void msgTwo(char* R1, char* R2, char* DP, char* msgTwo, int* len ){
    int key;

    // generate R2
    staticDPS(32, R2);
    dynamicDPS("dp.data", DP);

    // form {R2DP(new)}R1
    key = md5ToKey(R1);
    strcpy(msgTwo, R2);
    strcat(msgTwo, DP);
    *len = strlen(msgTwo);
    encRoundRobin_len_all(msgTwo, *len, key);
}
void parseMsgTwo(char* R1, char* R2, char* DP, char* msgTwo){
    int len, key;

    // form key
    key = md5ToKey(R1);
    len = strlen(msgTwo);
    decRoundRobin_len_all(msgTwo, len, key);
    sscanf(msgTwo,"%32c%s", R2, DP);
    R2[32] = '\0';
}

void msgThree(char* R2,char* R3,char* DP,char* msgThree,int* len){
    int key;
    char SDDigest[100];

    // generate R3
    staticDPS(32, R3);
    // form h(DP)
    md5_action(DP, SDDigest);
    // form {R2DP(new)}R1
    key = md5ToKey(R2);
```

```
    strcpy(msgThree, R3);
    strcat(msgThree, SDDigest);
    *len = strlen(msgThree);
    encRoundRobin_len_all(msgThree, *len, key);
}
void parseMsgThree(char* R2, char* R3, char* HDP, char* msgThree){
    int len, key;
    char SDDigest[100];

    key = md5ToKey(R2);
    len = strlen(msgThree);
    decRoundRobin_len_all(msgThree, len, key);
    sscanf(msgThree,"%32c%s", R3, HDP);
    R3[32] = '\0';
}
```

D.8.4 md5ADT.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

typedef unsigned long mULONG;
typedef unsigned char mUCHAR;

typedef struct {
        mULONG hash[4];
        mULONG bits[2];
        mUCHAR data[64];
} MD5Context;

/* Basic MD5 functions */
#define F1(x, y, z) (z ^ (x & (y ^ z)))
#define F2(x, y, z) (y ^ (z & (x ^ y)))
#define F3(x, y, z) (x ^ y ^ z)
#define F4(x, y, z) (y ^ (x | ~z))

/* This is the central step in the MD5 algorithm. */
#define TRANSFORM(f, w, x, y, z, data, s) \
        ( w += f(x, y, z) + data,  w = w<<s | w>>(32-s),  w += x )

/* returnable errors */
#define MD5_SYNTAX_ERROR        1
#define MD5_FILE_ERROR          2
#define MD5_FILE_IO_ERROR       3
#define MD5_MISSING_INPUT       4
#define MD5_ERROR_OUT_CLOSE     5
#define MD5_SYNTAX_TOOMANYARGS  6
#define MD5_TEST_FAILURE        7

#define MD5_BUFSIZE 4096

/* md5_transform
**
** The MD5 "basic transformation". Updates the hash
** based on the data block passed.
*/
static void md5_transform(mULONG hash[4], const mULONG data[16]);

/* md5_init
**
** Initialise md5 context structure
*/
void md5_init( MD5Context *ctx );

/* md5_update
**
** Update context with the next buffer from the stream of data.
** Call with each block of data to update the md5 hash.
*/
void md5_update(MD5Context *ctx, const mUCHAR* buf, mULONG buflen);

/* md5_final
**
** Finalize creation of md5 hash and copy to digest buffer.
*/
void md5_final( MD5Context *ctx, mUCHAR digest[ 16 ] );

/* md5_digest_string
**
** Supply the digest and a buffer for the string.
** This routine will populate the buffer and
** return the value as a C string.
*/
char *md5_digest_string( mUCHAR d[16], char digest_string[33] );

/* md5_message
**
```

```
** Gather text messages in one place.
*/
char *md5_message( int errcode );

/* md5_file
**
** Compute hash on an open file handle.
*/
int md5_file( FILE *infile, mUCHAR digest[ 16 ] );

/* md5_filename
**
** Compute the hash on a file.
*/
int md5_filename( const char *infilename, mUCHAR digest[ 16 ] );

/* md5_verify_filename
**
** Verify that a given file has a given hash.
*/
int md5_verify_filename( char *infilename, char *hash );

/* md5_buffer
**
** Compute the md5 hash on a buffer.
*/
void md5_buffer(const mUCHAR *buf, int buflen, mUCHAR digest[16]);

/* LCaseHex
**
** Forces Hex string to lowercase.
**
** Make sure that we do apples to apples
** comparison of input hash to calculated hash.
*/
static char *LCaseHex( char *h );

/* showuse
**
** display usage information, help, version info
*/
void showuse( int morehelp );

/* Standard test vectors and expected hashes.
** Data below is used in the self-test function.
**
** The standard test vectors are augmented by
** a couple more cases to ease regression test.
*/
#define NUM_VECTORS 7
#define NUM_CASES   9

/* GenAndTestFileCases()
**
** Generates CASE data files for regression test
** and runs md5_verify_filename check.
*/
int GenAndTestFileCases();

/*
** md5_self_test
**
** Tests the core hashing functions against the
** test suite given in the RFC.
*/
int md5_self_test( void );


/* md5_action
**
** The function to form md5 from arguments string.
*/
char * md5_action(char * input, char digest_string[33] );
```

D.8.5 md5ADT.cpp

```
#include "md5ADT.h"

char *test_vector[] = {
  "",
  "a",
  "abc",
  "message digest",
  "abcdefghijklmnopqrstuvwxyz",
  "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789",
  "12345678901234567890123456789012345678901234567890123456789012345
  5678901234567890",

  "Replace CASE7.DAT with your own 10MB to 100MB file\n",
```

```
  "Replace CASE8.DAT with your own 100MB+  file\n"
};

char *test_hash[] = {
    "d41d8cd98f00b204e9800998ecf8427e",
    "0cc175b9c0f1b6a831c399e269772661",
    "900150983cd24fb0d6963f7d28e17f72",
    "f96b697d7cb7938d525a2f31aaf161d0",
    "c3fcd3d76192e4007dfb496cca67e13b",
    "d174ab98d277d9f5a5611c2c9f419d9f",
    "57edf4a22be3c955ac49da2e2107b67a",

    "04842407f40f4fc21f159bf89a7bf63e",
    "472c5207d61df9454c1e732930d1c496"
};

static void md5_transform( mULONG hash[4], const mULONG data[16])
{
    mULONG a = hash[0], b = hash[1], c = hash[2], d = hash[3];

    /* Round 1 */
    TRANSFORM( F1, a, b, c, d, data[ 0] + 0xd76aa478,  7);
    TRANSFORM( F1, d, a, b, c, data[ 1] + 0xe8c7b756, 12);
    TRANSFORM( F1, c, d, a, b, data[ 2] + 0x242070db, 17);
    TRANSFORM( F1, b, c, d, a, data[ 3] + 0xc1bdceee, 22);
    TRANSFORM( F1, a, b, c, d, data[ 4] + 0xf57c0faf,  7);
    TRANSFORM( F1, d, a, b, c, data[ 5] + 0x4787c62a, 12);
    TRANSFORM( F1, c, d, a, b, data[ 6] + 0xa8304613, 17);
    TRANSFORM( F1, b, c, d, a, data[ 7] + 0xfd469501, 22);
    TRANSFORM( F1, a, b, c, d, data[ 8] + 0x698098d8,  7);
    TRANSFORM( F1, d, a, b, c, data[ 9] + 0x8b44f7af, 12);
    TRANSFORM( F1, c, d, a, b, data[10] + 0xffff5bb1, 17);
    TRANSFORM( F1, b, c, d, a, data[11] + 0x895cd7be, 22);
    TRANSFORM( F1, a, b, c, d, data[12] + 0x6b901122,  7);
    TRANSFORM( F1, d, a, b, c, data[13] + 0xfd987193, 12);
    TRANSFORM( F1, c, d, a, b, data[14] + 0xa679438e, 17);
    TRANSFORM( F1, b, c, d, a, data[15] + 0x49b40821, 22);

    /* Round 2 */
    TRANSFORM( F2, a, b, c, d, data[ 1] + 0xf61e2562,  5);
    TRANSFORM( F2, d, a, b, c, data[ 6] + 0xc040b340,  9);
    TRANSFORM( F2, c, d, a, b, data[11] + 0x265e5a51, 14);
    TRANSFORM( F2, b, c, d, a, data[ 0] + 0xe9b6c7aa, 20);
    TRANSFORM( F2, a, b, c, d, data[ 5] + 0xd62f105d,  5);
    TRANSFORM( F2, d, a, b, c, data[10] + 0x02441453,  9);
    TRANSFORM( F2, c, d, a, b, data[15] + 0xd8a1e681, 14);
    TRANSFORM( F2, b, c, d, a, data[ 4] + 0xe7d3fbc8, 20);
    TRANSFORM( F2, a, b, c, d, data[ 9] + 0x21e1cde6,  5);
    TRANSFORM( F2, d, a, b, c, data[14] + 0xc33707d6,  9);
    TRANSFORM( F2, c, d, a, b, data[ 3] + 0xf4d50d87, 14);
    TRANSFORM( F2, b, c, d, a, data[ 8] + 0x455a14ed, 20);
    TRANSFORM( F2, a, b, c, d, data[13] + 0xa9e3e905,  5);
    TRANSFORM( F2, d, a, b, c, data[ 2] + 0xfcefa3f8,  9);
    TRANSFORM( F2, c, d, a, b, data[ 7] + 0x676f02d9, 14);
    TRANSFORM( F2, b, c, d, a, data[12] + 0x8d2a4c8a, 20);

    /* Round 3 */
    TRANSFORM( F3, a, b, c, d, data[ 5] + 0xfffa3942,  4);
    TRANSFORM( F3, d, a, b, c, data[ 8] + 0x8771f681, 11);
    TRANSFORM( F3, c, d, a, b, data[11] + 0x6d9d6122, 16);
    TRANSFORM( F3, b, c, d, a, data[14] + 0xfde5380c, 23);
    TRANSFORM( F3, a, b, c, d, data[ 1] + 0xa4beea44,  4);
    TRANSFORM( F3, d, a, b, c, data[ 4] + 0x4bdecfa9, 11);
    TRANSFORM( F3, c, d, a, b, data[ 7] + 0xf6bb4b60, 16);
    TRANSFORM( F3, b, c, d, a, data[10] + 0xbebfbc70, 23);
    TRANSFORM( F3, a, b, c, d, data[13] + 0x289b7ec6,  4);
    TRANSFORM( F3, d, a, b, c, data[ 0] + 0xeaa127fa, 11);
    TRANSFORM( F3, c, d, a, b, data[ 3] + 0xd4ef3085, 16);
    TRANSFORM( F3, b, c, d, a, data[ 6] + 0x04881d05, 23);
    TRANSFORM( F3, a, b, c, d, data[ 9] + 0xd9d4d039,  4);
    TRANSFORM( F3, d, a, b, c, data[12] + 0xe6db99e5, 11);
    TRANSFORM( F3, c, d, a, b, data[15] + 0x1fa27cf8, 16);
    TRANSFORM( F3, b, c, d, a, data[ 2] + 0xc4ac5665, 23);

    /* Round 4 */
    TRANSFORM( F4, a, b, c, d, data[ 0] + 0xf4292244,  6);
    TRANSFORM( F4, d, a, b, c, data[ 7] + 0x432aff97, 10);
    TRANSFORM( F4, c, d, a, b, data[14] + 0xab9423a7, 15);
    TRANSFORM( F4, b, c, d, a, data[ 5] + 0xfc93a039, 21);
    TRANSFORM( F4, a, b, c, d, data[12] + 0x655b59c3,  6);
    TRANSFORM( F4, d, a, b, c, data[ 3] + 0x8f0ccc92, 10);
    TRANSFORM( F4, c, d, a, b, data[10] + 0xffeff47d, 15);
    TRANSFORM( F4, b, c, d, a, data[ 1] + 0x85845dd1, 21);
    TRANSFORM( F4, a, b, c, d, data[ 8] + 0x6fa87e4f,  6);
    TRANSFORM( F4, d, a, b, c, data[15] + 0xfe2ce6e0, 10);
    TRANSFORM( F4, c, d, a, b, data[ 6] + 0xa3014314, 15);
    TRANSFORM( F4, b, c, d, a, data[13] + 0x4e0811a1, 21);
    TRANSFORM( F4, a, b, c, d, data[ 4] + 0xf7537e82,  6);
    TRANSFORM( F4, d, a, b, c, data[11] + 0xbd3af235, 10);
    TRANSFORM( F4, c, d, a, b, data[ 2] + 0x2ad7d2bb, 15);
```

```
    TRANSFORM( F4, b, c, d, a, data[ 9] + 0xeb86d391, 21);

    hash[ 0 ] += a;
    hash[ 1 ] += b;
    hash[ 2 ] += c;
    hash[ 3 ] += d;
}

void md5_init( MD5Context *ctx )
{
    ctx->hash[ 0 ] = 0x67452301;
    ctx->hash[ 1 ] = 0xefcdab89;
    ctx->hash[ 2 ] = 0x98badcfe;
    ctx->hash[ 3 ] = 0x10325476;

    ctx->bits[ 0 ] = 0;
    ctx->bits[ 1 ] = 0;
}

void md5_update(MD5Context *ctx, const mUCHAR *buf, mULONG buflen)
{
    mULONG idx;

    /* Update bitcount */

    idx = ctx->bits[ 0 ];
    ctx->bits[ 0 ] = idx + (buflen << 3);
    if( ctx->bits[ 0 ] < idx ) {
        ctx->bits[ 1 ]++;         /* Carry from low to high */
    }
    ctx->bits[ 1 ] += buflen >> 29;

    idx = (idx >> 3) & 0x3f;    /* Bytes already in ctx->data */

    /* Handle any leading odd-sized chunks */

    if( idx != 0 ) {
        mUCHAR *p = (mUCHAR *) ctx->data + idx;

        idx = 64 - idx;
        if( buflen < idx ) {
            memcpy( p, buf, (size_t) buflen );
        }
        else {
            memcpy( p, buf, (size_t) idx );
            md5_transform( ctx->hash, (mULONG *) ctx->data );
            buf += idx;
            buflen -= idx;
        }
    }
    if( buflen >= idx ) {
        while( buflen >= 64 ) {
            memcpy( ctx->data, buf, 64 );
            md5_transform( ctx->hash, (mULONG *) ctx->data );
            buf += 64;
            buflen -= 64;
        }
        memcpy( ctx->data, buf, (size_t) buflen );
    }
}

void md5_final( MD5Context *ctx, mUCHAR digest[ 16 ] )
{
    mULONG count;
    mUCHAR *pad;

    count = (ctx->bits[ 0 ] >> 3)&0x3F; /* N0. of bytes mod 64 */
    pad = ctx->data + count;
    *pad++ = 0x80;

    /* Bytes of padding needed to make 64 bytes */
    count = 64 - 1 - count;

    /* Pad out to 56 mod 64 */
    if( count < 8 ) {
        /* Two lots of padding:  Pad the first block to 64 bytes */
        memset( pad, 0, (size_t) count );
        md5_transform( ctx->hash, (mULONG *) ctx->data );

        /* Now fill the next block with 56 bytes */
        memset( ctx->data, 0, 56 );
    } else {
        /* Pad block to 56 bytes */
        memset( pad, 0, (size_t) (count - 8) );
    }

    /* Append length in bits and transform */
    ((mULONG *) ctx->data)[ 14 ] = ctx->bits[ 0 ];
    ((mULONG *) ctx->data)[ 15 ] = ctx->bits[ 1 ];
```

```
    md5_transform( ctx->hash, (mULONG *) ctx->data );
    memcpy( digest, ctx->hash, 16 );
}

char *md5_digest_string( mUCHAR d[ 16 ], char digest_string[ 33 ] )
{
    int i;

    digest_string[ 32 ] = 0;
    for( i = 0; i < 16; i++ ) {
        sprintf( &(digest_string[ i * 2 ]), "%2.2x", d[ i ] );
    }

    return( digest_string );
}

char *md5_message( int errcode )
{
    #define MD5_MAX_MESSAGES 8
    char *msgs[ MD5_MAX_MESSAGES ] = {
        "md5:000:Invalid Message Code.",
        "md5:001:Syntax Error -- check help for usage.",
        "md5:002:File Error Opening/Creating Files.",
        "md5:003:File I/O Error -- Note: file cleanup not done.",
        "md5:004:Missing input -- nothing to hash.",
        "md5:005:Error on output file close.",
        "md5:006:Syntax: Too many arguments.",
        "md5:007:Test Failure."
    };
    char *msg = msgs[ 0 ];

    if( errcode > 0 && errcode < MD5_MAX_MESSAGES ) {
        msg = msgs[ errcode ];
    }

    return( msg );
}

int md5_file( FILE *infile, mUCHAR digest[ 16 ] )
{
    int retcode;
    MD5Context ctx;
    mUCHAR buf[ MD5_BUFSIZE ];
    mULONG bytes_read;

    md5_init( &ctx );

    while( (bytes_read = fread (buf, sizeof (mUCHAR),
          MD5_BUFSIZE, infile)) > 0 ) {
        md5_update( &ctx, buf, bytes_read );
    }
    if( ferror( infile ) ) {
        retcode = MD5_FILE_IO_ERROR;
    }
    else {
        md5_final( &ctx, digest );
        retcode = 0;
    }

    return( retcode );
}

int md5_filename( const char *infilename, mUCHAR digest[ 16 ] )
{
    FILE *infile;
    int retcode = MD5_FILE_ERROR;

    if( !infilename ) {
        infile = stdin;
    }
    else {
        infile = fopen( infilename, "rb" );
    }
    if( !infile ) {
        printf( "Error: FileName='%s' -- %s\n",
                 infilename, strerror( errno ) );
    }
    else {
        retcode = md5_file( infile, digest );
        if( infile != stdin ) {
            if( fclose( infile ) != 0 ) {
                char *ErrM = md5_message( MD5_ERROR_OUT_CLOSE );
                printf( "Error: %s -- %s\n", ErrM,
                         strerror( errno ) );
                retcode = MD5_FILE_IO_ERROR;
            }
        }
    }

    return( retcode );
```

```
}

int md5_verify_filename( char *infilename, char *hash )
{
    int retcode = 0;
    mUCHAR digest[ 16 ];
    char digest_string[ 33 ];
    char *hd;

    md5_filename( infilename, digest );
    hd = md5_digest_string( digest, digest_string );
    printf( "%s << Expected Hash Value\n%s << Actual Hash Value\n",
             hash, hd );
    if( strcmp( hash, hd ) ){
        retcode = MD5_TEST_FAILURE;
    }

    return( retcode );
}

void md5_buffer(const mUCHAR *buf, int buflen, mUCHAR digest[16])
{
    MD5Context ctx;

    md5_init( &ctx );
    md5_update( &ctx, buf, buflen );
    md5_final( &ctx, digest );
}

int GenAndTestFileCases()
{
    int i;
    char outfilename[16];
    FILE *outfile;
    int retcode = 0;

    for( i = 0; i < NUM_CASES && retcode == 0; i++ ) {
        sprintf( outfilename, "CASE%d.DAT", i );
        printf( "Writing %s\n", outfilename );
        outfile = fopen( outfilename, "wb" );
        if( !outfile ) {
            printf( "Error: FileName='%s' -- %s\n",
                     outfilename, strerror( errno ) );
        }
        else {
            if( fprintf( outfile, "%s", test_vector[ i ] ) < 0 ) {
                printf( "Error: FileName='%s' -- %s\n",
                         outfilename, strerror( errno ) );
                retcode = MD5_FILE_IO_ERROR;
            }
            if( fclose( outfile ) != 0 ) {
                char *ErrM = md5_message( MD5_ERROR_OUT_CLOSE );
                printf( "Error: %s -- %s\n", ErrM,
                         strerror( errno ) );
                retcode = MD5_FILE_IO_ERROR;
            }
            else {
                if( i < NUM_VECTORS ) {
                    retcode = md5_verify_filename( outfilename,
                        test_hash[ i ] );
                }
            }
        }
    }

    return( retcode );
}

int md5_self_test( void )
{
    int i, retcode = 0;
    mUCHAR digest[ 16 ];
    char digest_string[ 33 ];
    char *tv;
    char *hd;

    for( i = 0; i < NUM_VECTORS && retcode == 0; i++ ) {
        tv = test_vector[ i ];
        md5_buffer( (mUCHAR *) tv, strlen( tv ), digest );
        hd = md5_digest_string( digest, digest_string );
        printf( "%s << Expected Hash Value\n%s <<
            Actual Hash Value\n", test_hash[ i ], hd );
        if( strcmp( hd, test_hash[ i ] ) ) {
            retcode = MD5_TEST_FAILURE;
        }
    }
    if( retcode == 0 ) {
        retcode = GenAndTestFileCases();
    }
```

```
    return( retcode );
}

static char *LCaseHex( char *h )
{
    char *p;

    for( p = h; *p; p++ ) {
        *p = (char) (*p | 0x20);
    }

    return( h );
}

void showuse( int morehelp )
{
    {
        printf( "\n" );
        printf( "  md5      (create md5 hash)
            Bob Trower 03/11/05 \n" );
        printf( "           (C) Copr Bob Trower 1986-2005.
            Version 0.00B \n" );
        printf( "  Usage:   md5 [-option] <Input> [testhash]\n" );
        printf( "  Purpose: This program is a simple utility
            that\n" );
        printf( "           implements the md5 hashing algorithm
            (RFC1321).\n" );
    }
    if( !morehelp ) {
        printf( "           Use -h option for additional help.\n" );
    }
    else {
        printf( "  Options: -f  Input is filename. -s
            Input is string.\n" );
        printf( "           -h  This help text.    -?
            This help text.\n" );
        printf( "           -t  Self Test.         -v
            Verify file hash.\n" );
        printf( "  Note:    -s  Is the default.
            It hashes the input as a\n" );
        printf( "               string\n" );
        printf( "  Returns: 0 = success.
            Non-zero is an error code.\n" );
        printf( "  ErrCode: 1 = Bad Syntax
            2 = File Open error\n" );
        printf( "           3 = File I/O error
            4 = Missing input\n" );
        printf( "  Example: md5 -s Some_String
            <- hash that string.\n" );
        printf( "           md5 -f SomeFileName
            <- hash that file.\n" );
        printf( "           md5 -t
            <- Perform Self Test.\n" );
        printf( "           md5 -v filename, testhash
            <- Verify file hash.\n" );
        printf( "  Source:  Source code and latest releases
            can be found at:\n" );
        printf( "           http://toogles.sourceforge.net\n" );
        printf( "  Release: 0.00.00, Fri Mar 11 03:30:00 2005,
            ANSI-SOURCE C\n" );
    }
}

char * md5_action(char * input, char digest_string[33] ){
    mUCHAR digest[ 16 ];

    md5_buffer( (mUCHAR*)input, strlen(input), digest);
    return md5_digest_string( digest, digest_string );
}
```

D.8.6 bst.h

```
/************************************************************
  COMMONWEALTH OF AUSTRALIA Copyright Regulations 1969

                    WARNING

  This material has been copied and communicated to you
  by or on behalf of The University of Southern Queensland
  pursuant to Part VA of the Copyright Act 1968 (the Act).

  The material in this communication may be subject to
  copyright under the Act. Any further copying or
  communication of this material by you may be the subject
  of copyright protection under the Act.

  Do not remove this message
************************************************************/
```

```
#ifndef _BST_H_
#define _BST_H_ struct Bst;

enum BstTraverseOrder {pre, in, post};
typedef int BstCompare(const void *item1, const void *item2);
typedef void BstPrint(const void *item);
typedef void BstList(const void *item, int sockNum, int keyIndex);

Bst *BstConstruct(BstCompare *fn);  // NULL if fails bool
BstInsert(Bst *tree, const void *item);
bool BstReplace(Bst *tree, const void *item, BstCompare *fn,
    const void* dup);
const void *BstSearch(const Bst *tree, const void *item);
 // NULL if not found
void BstDestroy(Bst *tree);
void BstTraverse(Bst *tree, BstPrint *fn, BstTraverseOrder order);
void BstDump(Bst *tree, BstList *fn, int sockNum,
    BstTraverseOrder order, int keyIndex);

#endif
```

D.8.7 bst.cpp

```
/*************************************************************
  COMMONWEALTH OF AUSTRALIA Copyright Regulations 1969

                  WARNING

  This material has been copied and communicated to you
  by or on behalf of The University of Southern Queensland
  pursuant to Part VA of the Copyright Act 1968 (the Act).

  The material in this communication may be subject to
  copyright under the Act. Any further copying or
  communication of this material by you may be the subject
  of copyright protection under the Act.

  Do not remove this message

*************************************************************/

#include <stdlib.h> #include "bst.h"

struct BstNode {
    const void *item;
    BstNode *left;
    BstNode *right;
};

struct Bst {
    BstNode *root;
    BstCompare *compare;
};

Bst *BstConstruct(BstCompare *compare) {
    Bst *tree;

    tree = (Bst *)malloc(sizeof(Bst));
    if (tree == NULL) {
        return NULL;
    }
    tree->root = NULL;  // Initially empty
    tree->compare = compare;
    return tree;
}

bool BstInsert(Bst *tree, const void *item) {
    BstNode *current = tree->root;
        BstNode *parent = NULL, * newnode;
    int result;

    while (current != NULL) {
        parent = current;
        result = tree->compare(item, current->item);
        if (result < 0) {
            current = current->left;
        } else if (result > 0) {
            current = current->right;
        } else {
            return false;  // Found
        }
    }
    // Found insertion point so create new node and attach it
    if ((newnode = (BstNode *)malloc(sizeof(BstNode))) == NULL) {
        return false;  // No memory
    }
        newnode->item = item;
        newnode->left = newnode->right = NULL;
        if (tree->root == NULL) {  // empty tree
```

```
                tree->root = newnode;
    }
        else if (result < 0) {
                parent->left = newnode;
    }
    else {
                parent->right = newnode;
    }

        return true;
}

bool BstReplace(Bst *tree, const void *item, BstCompare *fn, const
void* dup) {
        BstNode *current = tree->root;
        BstNode *parent = NULL, * newnode;
        int result;

        while (current != NULL) {
            parent = current;
            result = tree->compare(item, current->item);
            if (result < 0) {
                current = current->left;
            } else if (result > 0) {
                current = current->right;
            } else {
                if (fn(current->item, item)){
                    dup = current->item;
                    current->item = item;
            }
            return false;  // Found with same hop count
        }
    }
    // Found insertion point so create new node and attach it
    if ((newnode = (BstNode *)malloc(sizeof(BstNode))) == NULL) {
            return false;  // No memory
    }
    newnode->item = item;
    newnode->left = newnode->right = NULL;
    if (tree->root == NULL) {  // empty tree
           tree->root = newnode;
    }
    else if (result < 0) {
            parent->left = newnode;
    }
    else {
            parent->right = newnode;
    }
    return true;
}

const void *BstSearch(const Bst *tree, const void *item) {
    BstNode *current = tree->root;
    int result;

    while (current != NULL) {
        result = tree->compare(item, current->item);
        if (result < 0) {
            current = current->left;
        } else if (result > 0) {
            current = current->right;
        } else {
            return current->item;  // Found
        }
    }
    return NULL;  // Not found
}

static void Traverse(BstNode *ptr, BstPrint *fn, BstTraverseOrder
order) {
    if (ptr != NULL) {
        switch (order) {
        case pre:
            fn(ptr->item);
            Traverse(ptr->left, fn, pre);
            Traverse(ptr->right, fn, pre);
            break;
        case in:
            Traverse(ptr->left, fn, in);
            fn(ptr->item);
            Traverse(ptr->right, fn, in);
            break;
        case post:
            Traverse(ptr->left, fn, post);
            Traverse(ptr->right, fn, post);
            fn(ptr->item);
            break;
        default:
            break;
        }
```

```
    }
}

static void List(BstNode *ptr, BstList *fn, int sockNum,
    BstTraverseOrder order, int keyIndex) {
        if (ptr != NULL) {
            switch (order) {
                case pre:
                  fn(ptr->item, sockNum, keyIndex);
                  List(ptr->left, fn, sockNum, pre, keyIndex);
                  List(ptr->right, fn, sockNum, pre, keyIndex);
                  break;
                case in:
                  List(ptr->left, fn, sockNum, in, keyIndex);
                  fn(ptr->item, sockNum, keyIndex);
                  List(ptr->right, fn, sockNum, in, keyIndex);
                  break;
                case post:
                  List(ptr->left, fn, sockNum, post, keyIndex);
                  List(ptr->right, fn, sockNum, post, keyIndex);
                  fn(ptr->item, sockNum, keyIndex);
                  break;
                default:
                  break;
            }
        }
}

void BstTraverse(Bst *tree, BstPrint *fn, BstTraverseOrder order) {
    Traverse(tree->root, fn, order);
}

void BstDump(Bst *tree, BstList *fn, int sockNum, BstTraverseOrder
order, int keyIndex) {
        List(tree->root, fn, sockNum, order, keyIndex);
}

static void DeleteSubtree(BstNode *ptr) {
    // Recursively delete nodes
    if (ptr == NULL) {
        return;
    }
    DeleteSubtree(ptr->left);
    DeleteSubtree(ptr->right);
    free(ptr);
}

void BstDestroy(Bst *tree) {
    DeleteSubtree(tree->root);
    free(tree);
}
```

D.8.8 sp.h

```
struct sp;

int   spCompare(const void *item1, const void *item2);
sp*   spCreate();
void  spDestroy(sp* SP);

char* getID (sp* SP);
int   getServicePort (sp* SP);
char* getServer (sp* SP);
char* getServicePath (sp* SP);
int   getCost (sp* SP);
char* getRestriction (sp* SP);
char* getPreviousHop (sp* SP);
char* getNextHop(sp* SP);
void  setID (sp* SP, char* ID);
void  setServicePort(sp* SP, int Port);
void  setServer (sp* SP, char* Server);
void  setServicePath (sp* SP, char*ServicePath);
void  setCost (sp* SP, int Cost);
void  setRestriction (sp* SP, char* Restriction);
void  setPreviousHop (sp* SP, char* PreviousHop);
void  setNextHop(sp* SP, char* next);
```

D.8.9 sp.cpp

```
#include "sp.h"
#include "bst.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

struct sp {
    char ID[100];
```

```
    int  ServicePort;
    char Server[100];
    char ServicePath[100];
    int  Cost;
    char Restriction[10];
    char PreviousHop[100];
    char NextHop[100];
};
int spCompare(const void *item1, const void *item2){
    sp * SP1 = (sp*)item1;
    sp * SP2 = (sp*)item2;
    return strcmp(SP1->ID, SP2->ID);
}
sp * spCreate(){
    return (sp*)malloc(sizeof(sp));
}
void spDestroy(sp* SP){
    free(SP);
}
char * getID (sp* SP){
    return SP->ID;
}
int getServicePort (sp* SP){
    return SP->ServicePort;
}
char * getServer (sp* SP){
    return SP->Server;
}
char * getServicePath (sp* SP){
    return SP->ServicePath;
}
int getCost (sp* SP){
    return SP->Cost;
}
char * getRestriction (sp* SP){
    return SP->Restriction;
}
char * getPreviousHop (sp* SP){
    return SP->PreviousHop;
}
char * getNextHop(sp* SP){
    return SP->NextHop;
}

void setID (sp* SP, char* ID){
    strcpy(SP->ID,ID);
}
void setServicePort(sp* SP, int Port){
    SP->ServicePort = Port;
}
void setServer (sp* SP, char* Server){
    strcpy(SP->Server, Server);
}
void setServicePath (sp* SP, char*ServicePath){
    strcpy(SP->ServicePath, ServicePath);
}
void setCost (sp* SP, int Cost){
    SP->Cost = Cost;
}
void setRestriction (sp* SP, char* Restriction){
    strcpy(SP->Restriction, Restriction);
}
void setPreviousHop (sp* SP, char* PreviousHop){
    strcpy(SP->PreviousHop, PreviousHop);
}
void setNextHop(sp* SP, char* next){
    strcpy(SP->NextHop, next);
}
```

D.8.10 path.h

```
void pInformation(char* p, char* server, char* service,
                  char* port, char* serverAdd);

// return 1 when inside
// return 0 when not
int pInsidePath(char* p, char* thisIP);

//check with pInsidePath() first!
// return 1 when there is a forward AS; nextIP assigned.
// return 0 when there is no forward AS; next IP not assigned.
int pForward(char* p, char* thisIP, char* nextIP);

// check with pInsidePath() first!
// return 1 when there is a backward AS; nextIP assigned.
// return 0 when there is no backward AS; nextIP not assigned.
int pBackward(char* p, char* thisIP, char* nextIP);
```

D.8.11 path.cpp

```
#include "path.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

void pInformation(char* p, char* server, char* service,
                  char* port, char* serverAdd){
    char term1[100];
    char term2[100];
    char term3[100];
    FILE * portFile;

    sscanf (p, "%*[^:]%*c%s", term3);
    do {
        sscanf (term3, "%[^/]%*c%s", term1, term2);
        strcpy(term3, term2);
    } while (isdigit(term2[0]));

    strcpy(serverAdd, term1);
    sscanf(term2, "%[^/]%*c%[^>]", server, service);
    if ((portFile = fopen("port.data", "r"))==NULL){
        printf("Can't open file!\n");
        exit(0);
    }
    do {
        fscanf (portFile, "%s %s", term1, term2);
    } while (strcmp(term1, service)!=0);
    strcpy(port, term2);
    fclose(portFile);
}
int pInsidePath(char* p, char* thisIP){
    // return 1 when thisIP appears in path
    // return 0 when thisIP does not appear in Path
    char term1[100];
    char term2[100];
    char term3[100];

    sscanf (p, "%*[^:]%*c%s", term3);
    do {
        sscanf (term3, "%[^/]%*c%s", term1, term2);
    if (!isdigit(term2[0])) break;
        strcpy(term3, term2);
    } while (strcmp(term1, thisIP)!=0);
    if (strcmp(term1, thisIP)){
        return 0;
    } else {
        return 1;
    }
}
int pForward(char* p, char* thisIP, char* nextIP){
    //check with pInsidePath() first!
    // return 1 when there is a forward AS; nextIP assigned.
    // return 0 when there is no forward AS; next IP not assigned.
    char term1[100];
    char term2[100];
    char term3[100];

    sscanf (p, "%*[^:]%*c%s", term3);
    do {
        sscanf (term3, "%[^/]%*c%s", term1, term2);
        if (!isdigit(term2[0])) break;
        strcpy(term3, term2);
    } while (strcmp(term1, thisIP)!=0);
    if (!isdigit(term2[0])){
        nextIP[0] = '\0';
        return 0;
    } else {
        sscanf(term2, "%[^/]", nextIP);
        return 1;
    }
}

int pBackward(char* p, char* thisIP, char* nextIP){
    // check with pInsidePath() first!
    // return 1 when there is a backward AS; nextIP assigned.
    // return 0 when there is no backward AS; nextIP not assigned.

    char term1[100];
    char term2[100];
    char term3[100];
    char previous[100];

    sscanf (p, "%*[^:]%*c%s", term3);
    term1[0] = 0;
    do {
        strcpy(previous, term1);
        sscanf (term3, "%[^/]%*c%s", term1, term2);
```

```
    if (!isdigit(term2[0])) break;
        strcpy(term3, term2);
    } while (strcmp(term1, thisIP)!=0);
    if (previous[0] != 0){
        strcpy (nextIP, previous);
    return 1;
    } else {
        nextIP[0] = 0;
        return 0;
    }
}
```

D.9 Data Files

D.9.1 PC1Map.data

```
200.200.2.2 200.200.2.2
200.200.3.2 200.200.2.2
200.200.4.2 200.200.2.2
```

D.9.2 PC2Map.data

```
200.200.1.2 200.200.1.2
200.200.3.2 200.200.4.2
200.200.4.2 200.200.4.2
```

D.9.3 PC3Map.data

```
200.200.1.2 200.200.2.2
200.200.2.2 200.200.2.2
200.200.4.2 200.200.2.2
```

D.9.4 PC4Map.data

```
200.200.1.2 200.200.3.2
200.200.2.2 200.200.3.2
200.200.3.2 200.200.3.2
```

D.9.5 PC1SP.data

```
time
6001
timeServer
200.200.1.2/timeServer/time
18
F
date
6002
dateServer
200.200.1.2/dateServer/date
18
F
echo
6003
echoServer
200.200.1.2/200.200.2.2/202.200.4.2/echoServer/echo
18
F
name
6004
nameServer
200.200.1.2/200.200.2.2/200.200..3.2/200.200.4.2/nameServer/name
18
F
slsC
6005
slsCServer
200.200.1.2/slsCServer/slsC
18
F
slsS
6006
slsSServer
200.200.1.2/lsServer/slsS
18
F
end
```

D.9.6 PC2SP.data

```
time
6001
timeServer
```

```
200.200.2.2/timeServer/time
18
F
date
6002
dateServer
200.200.2.2/dateServer/date
18
F
echo
6003
echoServer
200.200.2.2/202.200.4.2/echoServer/echo
18
F
name
6004
nameServer
200.200.2.2/200.200.4.2/200.200.3.2/nameServer/name
18
F
slsC
6005
slsCServer
200.200.2.2/slsCServer/slsC
18
F
slsS
6006
slsSServer
200.200.2.2/lsServer/slsS
18
F
end
```

D.9.7 PC3SP.data

```
time
6001
timeServer
200.200.3.2/timeServer/time
18
F
date
6002
dateServer
200.200.3.2/dateServer/date
18
F
echo
6003
echoServer
200.200.3.2/200.200.2.2/202.200.4.2/echoServer/echo
18
F
name
6004
nameServer
200.200.3.2/nameServer/name
18
F
slsC
6005
slsCServer
200.200.3.2/slsCServer/slsC
18
F
slsS
6006
slsSServer
200.200.3.2/lsServer/slsS
18
F
end
```

D.9.8 PC4SP.data

```
time
6001
timeServer
200.200.4.2/timeServer/time
18
F
date
6002
dateServer
200.200.4.2/dateServer/date
18
F
echo
```

```
6003
echoServer
202.200.4.2/echoServer/echo
18
F
name
6004
nameServer
200.200.4.2/200.200.3.2/nameServer/name
18
F
slsC
6005
slsCServer
200.200.4.2/slsCServer/slsC
18
F
slsS
6006
slsSServer
200.200.4.2/lsServer/slsS
18
F
end
```

D.9.9 dp.data

```
3854 aahs aals abas abba abbe abed abet able ably abos abri abut
abye abys aced aces ache achy acid acme acne acre acta acts acyl
adds adit ados adze aeon aero aery afar agar agas aged agee ager
ages agha agin agio agly agma agog agon ague ahem ahoy aide aids
ails aims ains airn airs airt airy aits ajar ajee akee akin alae
alan alar alas alba albs alec alee alef ales alfa alga alif alit
alky alls ally alma alme alms aloe alow alps also alto alts alum
amah amas ambo amen amia amid amie amin amir amis ammo amok amps
amus amyl anal anas ands anes anew anga anil anis ankh anna anoa
anon ansa anta ante anti ants anus aped aper apes apex apod apse
aqua arak arbs arch arco arcs area ares arfs aria arid aril arks
arms army arse arts arty arum arvo aryl asci asea ashy asks asps
atap ates atma atom atop auks auld aunt aura auto aver aves avid
avos avow away awed awee awes awls awns awny awol awry axal axed
axel axes axil axis axle axon ayah ayes ayin azan azon azon baal
baas baba babe babu baby bach back bade bads baff bags baht bail
bait bake bald bale balk ball balm bals bams band bane bang bani
bank bans baps barb bard bare barf bark barm barn bars base bash
bask bass bast bate bath bats batt baud bawd bawl bays bead beak
beam bean bear beat beau beck beds bedu beef been beep beer bees
beet begs bell bels belt bema bend bene bens bent berg berm best
beta beth bets bevy beys bhut bias bibb bibs bice bide bids bier
biff bigs bike bile bilk bill bima bind bine bins bint bios bird
birk birl birr bise bisk bite bits bitt bize blab blae blah blam
blat blaw bleb bled blet blew blin blip blob bloc blot blow blub
blue blur boar boas boat bobs bock bode bods body boff bogs bogy
boil bola bold bole boll bolo bolt bomb bond bone bong bonk bony
boob book boom boon boor boos boot bops bora bore born bort bosh
bosk boss bota both bots bott bout bowl bows boxy boyo boys bozo
brad brae brag bran bras brat braw bray bred bree bren brew brie
brig brim brin brio bris brit broo bros brow brrr brut bubo bubs
buck buds buff bugs buhl buhr bulb bulk bull bumf bump bums bund
bung bunk bunn buns bunt buoy bura burd burg burl burn burp burr
burs bury bush busk buss bust busy bute buts butt buys buzz byes
byre byrl byte byte cabs caca cade cadi cads cafe caff cage cagy
caid cain cake caky calf calk call calm calo calx came camp cams
cane cans cant cape caph capo caps carb card care cark carl carn
carp carr cars cart casa case cash cask cast cate cats caul cave
cavy caws cays ceca cede cedi cees ceil cell cels celt cent cepe
ceps cere cero cess cete chad cham chao chap char chat chaw chay
chef chew chez chia chic chid chin chip chis chit chon chop chow
chub chug chum ciao cine cion cire cist cite city clad clag clam
clan clap claw clay clef clew clip clod clog clon clop clot cloy
club clue coal coat coax cobb cobs coca cock coco coda code cods
coed coff coft cogs coho coif coil coin coir coke cola cold cole
cols colt coly coma comb come comp cone coni conk conn cons cony
coof cook cool coon coop coos coot cope cops copy cord core corf
cork corm corn cory cosh coss cost cosy cote cots coup cove cowl
cows cowy coxa coys cozy crab crag cram crap craw crew crib cris
croc crop crow crud crus crux cube cubs cuds cued cues cuff cuif
cuke cull culm cult cunt cups curb curd cure curf curl curn curr
curs curt cusk cusp cuss cute cuts cwms cyan cyma cyme cyst czar
czar dabs dace dada dado dads daff daft dago dags dahl dahs dais
daks dale dals dame damn damp dams dang dank daps darb dare dark
darn dart dash data date dato daub daut davy dawk dawn daws dawt
days daze dead deaf deal dean dear debs debt deck deco deed deem
deep deer dees deet defi deft defy deil deke dele delf deli dell
dels deme demo demy dene dens dent deny dere derm desk deva devs
dews dewy dexy deys dhak dhal dhow dial dibs dice dick dido didy
died diel dies diet digs dike dill dime dims dine ding dink dins
dint diol dips dipt dire dirk dirl dirt disc dish disk diss dita
dite dits ditz diva dive djin doat doby dock docs dodo doer does
doff doge dogs dogy doit dojo dole doll dols dolt dome doms dona
```

done dong dons doom door dopa dope dopy dore dork dorm dorp dorr
dors dory dose doss dost dote doth dots doty doum dour doux dove
down dows doxy doze dozy drab drag dram drat draw dray dree dreg
drek drew drib drip drop drub drug drum drys duad dual dubs duce
duci duck duct dude duds duel dues duet duff dugs duit duke dull
duly duma dumb dump dune dung dunk duns dunt duos dupe dups dura
dure durn duro durr dusk dust duty dyad dyed dyer dyes dyke dyne
dyne each earl earn ears ease east easy eath eats eaux eave ebbs
ebon eche echo ecru ecus eddo eddy edge edgy edhs edit eels eely
eery effs efts egad egal eger eggs eggy egis egos eide eked ekes
elan elds elhi elks ells elms elmy else emes emeu emfs emic emir
emit emus emyd ends engs enol enow envy eons epee epha epic epos
eras ergo ergs erne erns eros errs erst eses espy etas etch eths
etic etna etui euro even ever eves evil ewer ewes exam exec exes
exit exon expo eyas eyed eyen eyer eyes eyne eyra eyre eyry eyry
face fact fade fado fads fags fail fain fair fake fall falx fame
fane fang fano fans fard fare farl farm faro fart fash fast fate
fats faun faux fava fave fawn fays faze feal fear feat feck feds
feed feel fees feet fehs fell felt feme fems fend fens feod fere
fern fess feta fete fets feud feus fiar fiat fibs fice fico fido
fids fief fife figs fila file fill film filo fils find fine fink
fino fins fire firm firn firs fisc fish fist fits five fixt fizz
flab flag flak flam flan flap flat flaw flax flay flea fled flee
flew flex fley flic flip flit floc floe flog flop flow flub flue
flus flux foal foam fobs foci foes fogs fogy fohn foil foin fold
folk fond fons font food fool foot fops fora forb ford fore fork
form fort foss foul four fowl foxy foys fozy frae frag frap frat
fray free fret frig frit friz froe frog from frow frug fubs fuci
fuck fuds fuel fugs fugu fuji full fume fumy fund funk funs furl
furs fury fuse fuss futz fuze fuzz fyce fyke fyke gabs gaby gadi
gads gaed gaen gaes gaff gaga gage gags gain gait gala gale gall
gals gama gamb game gamp gams gamy gane gang gaol gape gaps gapy
garb gars gash gasp gast gate gats gaud gaum gaun gaur gave gawk
gawp gays gaze gear geck geds geed geek gees geez geld gels gelt
gems gene gens gent genu germ gest geta gets geum ghat ghee ghis
gibe gibs gids gied gien gies gift giga gigs gild gill gilt gimp
gink gins gips gird girl girn giro girt gist gits give glad gled
glee gleg glen gley glia glib glim glob glom glop glow glue glug
glum glut gnar gnat gnaw gnus goad goal goas goat gobo gobs goby
gods goer goes gogo gold golf gone gong good goof gook goon goop
goos gore gorp gory gosh gout gowd gowk gown goys grab grad gram
gran grat gray gree grew grey grid grig grim grin grip grit grog
grot grow grub grue grum guan guar guck gude guff guid gulf gull
gulp guls gums gunk guns guru gush gust guts guvs guys gybe gyms
gyps gyre gyri gyro gyve gyve haaf haar habu hack hade hadj haed
haem haen haes haet haft hags haha hahs haik hail hair haji hajj
hake hale half hall halm halo halt hame hams hand hang hank hant
haps hard hare hark harl harm harp hart hash hasp hast hate hath
hats haul haut have hawk haws hays haze hazy head heal heap hear
heat hebe heck heed heel heft hehs heil heir held hell helm helo
help heme hemp hems hens hent herb herd here herl herm hern hero
hers hest heth hets hewn hews hick hide hied hies high hike hila
hili hill hilt hind hins hint hips hire hisn hiss hist hits hive
hoar hoax hobo hobs hock hods hoed hoer hoes hogg hogs hoke hold
hole holk holm holp hols holt holy home homo homy hone hong honk
hons hood hoof hook hoop hoot hope hops hora horn hose host hots
hour hove howe howf howk howl hows hoya hoys hubs huck hued hues
huff huge hugs huic hula hulk hull hump hums hung hunh hunk huns
hunt hurl hurt hush husk huts hwan hyla hymn hype hypo hyps hyte
hyte iamb ibex ibis iced ices ichs icky icon idea idem ides idle
idly idol idyl iffy iglu ikat ikon ilea ilex ilia ilka ilks ills
illy imam imid immy impi imps inby inch info inia inks inky inly
inns inro inti into ions iota ired ires irid iris irks iron isba
isle isms itch item iwis ixia izar izar jabs jack jade jagg jags
jail jake jamb jams jane jape jarl jars jato jauk jaup java jaws
jays jazz jean jeed jeep jeer jees jeez jefe jehu jell jeon jerk
jess jest jete jets jeux jews jiao jibb jibe jibs jiff jigs jill
jilt jimp jink jinn jins jinx jism jive jobs jock joes joey jogs
john join joke joky jole jolt josh joss jota jots jouk jowl jows
joys juba jube judo juga jugs juju juke jump junk jupe jura jury
just jute juts juts kaas kabs kadi kaes kafs kagu kaif kail kain
kaka kaki kale kame kami kana kane kaon kapa kaph karn kart kata
kats kava kayo kays kbar keas keck keef keek keel keen keep keet
kefs kegs keir kelp kemp keno kens kent kepi keps kept kerb kerf
kern keto keys khaf khan khat khet khis kibe kick kids kief kier
kifs kike kill kiln kilo kilt kina kind kine king kink kino kins
kips kirk kirn kirs kiss kist kite kith kits kiva kiwi knap knar
knee knew knit knob knop knot know knur koan koas kobo kobs koel
kohl kola kolo konk kook koph kops kore kors koss koto kris kudo
kudu kues kuru kvas kyak kyar kyat kyte kyte labs lace lack lacs
lacy lade lads lady lags laic laid lain lair lake lakh laky lall
lama lamb lame lamp lams land lane lang lank laps lard lari lark
lars lase lash lass last late lath lati lats laud lava lave lavs
lawn laws lays laze lazy lead leaf leak leal lean leap lear leas
lech leek leer lees leet left legs lehr leis leke leks leku lend
leno lens lent lept less lest lets leud leva levo levy lewd leys
liar libs lice lich lick lido lids lied lief lien lier lies lieu
life lift like lilt lily lima limb lime limn limo limp limy line
ling link linn lino lins lint liny lion lips lira lire liri lisp
list lite lits litu live load loaf loam loan lobe lobo lobs loca

loch loci lock loco lode loft loge logo logs logy loin loll lone
long loof look loom loon loop loos loot lope lops lord lore lorn
lory lose loss lost lota loth loti lots loud loup lour lout love
lowe lown lows luau lube luce luck lude lues luff luge lugs lull
lulu lump lums luna lune lung lunk lunt luny lure lurk lush lust
lute lutz luvs luxe lwei lyes lynx lyre lyse lyse mabe mace mach
mack macs made mads maes mage magi mags maid mail maim main mair
make mako male mall malm malt mama mana mane mano mans many maps
marc mare mark marl mars mart mash mask mass mast mate math mats
matt maud maul maun maut mawn maws maxi maya mayo mays maze mazy
mead meal mean meat meed meek meet meld mell mels melt memo mems
mend meno menu meou meow mere merk merl mesa mesh mess meta mete
meth mewl mews meze mhos mibs mica mice mick midi mids mien miff
migg migs mike mild mile milk mill milo mils milt mime mina mind
mine mini mink mint minx mire miri mirk mirs miry mise miso miss
mist mite mitt mity mixt moan moas moat mobs mock mocs mode modi
mods mogs moil mojo moke mola mold mole moll mols molt moly mome
momi moms monk mono mons mony mood mool moon moor moos moot mope
mops mopy mora more morn mors mort mosk moss most mote moth mots
mott moue move mown mows moxa mozo much muck muds muff mugg mugs
mule mull mumm mump mums mumu muni muns muon mura mure murk murr
muse mush musk muss must mute muts mutt myna myth myth naan nabe
nabs nada nags naif nail name nana nans naoi naos nape naps narc
nard nark nary nave navy nays nazi neap near neat nebs neck need
neem neep neif nema nene neon nerd ness nest nets nett neuk neum
neve nevi news newt next nibs nice nick nide nidi nigh nill nils
nims nine nipa nips nisi nite nits nixe nixy nobs nock node nodi
nods noel noes nogg nogs noil noir nolo noma nome noms nona none
nook noon nope nori norm nose nosh nosy nota note noun nous nova
nows nowt nubs nude nuke null numb nuns nurd nurl nuts nuts oafs
oaks oars oast oath oats obes obey obia obis obit oboe obol ocas
odds odea odes odic odor odyl ofay offs ogam ogee ogle ogre ohed
ohia ohms oils oily oink okas okay okeh okes okra olds oldy olea
oleo oles olio olla omen omer omit once ones only onto onus onyx
oohs oops oots ooze oozy opah opal oped open opes opts opus orad
oral orbs orby orca orcs ordo ores orgy orle orra orts oryx orzo
osar oses ossa otic otto ouch ouds ouph ours oust outs ouzo oval
oven over ovum owed owes owls owns owse oxen oxes oxid oxim oyer
oyes oyez oyez paca pace pack pacs pact padi pads page paid paik
pail pain pair pale pall palm palp pals paly pams pane pang pans
pant papa paps para pard pare park parr pars part pase pash pass
past pate path pats paty pave pawl pawn paws pays peag peak peal
pean pear peas peat pech peck pecs peds peed peek peel peen peep
peer pees pegs pehs pein peke pele pelf pelt pend pens pent peon
pepo peps peri perk perm pert peso pest pets pews pfft pfui phat

phew phis phiz phon phot phut pial pian pias pica pice pick pics
pied pier pies pigs pika pike piki pile pili pill pily pima pimp
pina pine ping pink pins pint piny pion pipe pips pipy pirn pish
piso piss pita pith pits pity pixy plan plat play plea pleb pled
plew plie plod plop plot plow ploy plug plum plus pock poco pods
poem poet pogy pois poke poky pole poll polo pols poly pome pomp
poms pond pone pong pons pony pood poof pooh pool poon poop poor
pope pops pore pork porn port pose posh post posy pots pouf pour
pout pows pram prao prat prau pray pree prep prex prey prez prig
prim proa prod prof prog prom prop pros prow psis psst pubs puce
puck puds puff pugh pugs puja puke pula pule puli pull pulp puls
puma pump puna pung punk puns punt puny pupa pups pure puri purl
purr purs push puss puts putt putz pyas pyes pyic pyin pyre pyre
qaid qats qoph quad quag quai quay quey quid quin quip quit quiz
quod quod race rack racy rads raff raft raga rage ragi rags raia
raid rail rain raja rake raki rale rami ramp rams rand rang rani
rank rant rape raps rapt rare rase rash rasp rate rath rato rats
rave raws raya rays raze razz read real ream reap rear rebs reck
recs redd rede redo reds reed reef reek reel rees refs reft regs
reif rein reis rely rems rend rent repo repp reps resh rest rete
rets revs rhea rhos rhus rial rias ribs rice rich rick ride rids
riel rife riff rifs rift rigs rile rill rime rims rimy rind ring
rink rins riot ripe rips rise risk rite ritz rive road roam roan
roar robe robs rock rocs rode rods roes roil role rolf roll romp
roms rood roof rook room root rope ropy rose rosy rota rote roti
rotl roto rots roue roup rout roux rove rows rube rubs ruby ruck
rudd rude rued ruer rues ruff ruga rugs ruin rule ruly rump rums
rune rung runs runt ruse rush rusk rust ruth ruts ryas ryes ryke
rynd ryot ryot sabe sabs sack sacs sade sadi safe saga sage sago
sags sagy said sail sain sake saki sale sall salp sals salt same
samp sand sane sang sank sans saps sard sari sark sash sass sate
sati saul save sawn saws says scab scad scag scam scan scar scat
scop scot scow scry scud scum scup scut seal seam sear seas seat
secs sect seed seek seel seem seen seep seer sees sego segs seif
seis self sell sels seme semi send sene sent sept sera sere serf
sers seta sets sett sewn sews sext sexy shad shag shah sham shat
shaw shay shea shed shes shew shim shin ship shit shiv shmo shod
shoe shog shoo shop shot show shri shul shun shut sial sibb sibs
sice sick sics side sift sigh sign sike sild silk sill silo silt
sima simp sims sine sing sinh sink sins sipe sips sire sirs site
sith sits size sizy skag skas skat skee skeg skep skew skid skim
skin skip skis skit skua slab slag slam slap slat slaw slay sled
slew slid slim slip slit slob sloe slog slop slot slow slub slue
slug slum slur slut smew smit smog smug smut snag snap snaw sned
snib snip snit snob snog snot snow snub snug snye soak soap soar

```
sobs sock soda sods sofa soft soil soja soke sola sold sole soli
solo sols soma some sone song sons sook soon soot soph sops sora
sorb sord sore sori sorn sort soth sots souk soul soup sour sous
sown sows soya soys spae span spar spas spat spay spaz spec sped
spew spic spik spin spit spiv spot spry spud spue spun spur sris
stab stag star stat staw stay stem step stet stew stey stir stoa
stob stop stow stub stud stum stun stye suba subs such suck sudd
suds sued suer sues suet sugh suit sulk sulu sumo sump sums sung
sunk sunn suns supe sups suqs sura surd sure surf suss swab swag
swam swan swap swat sway swig swim swob swop swot swum sybo syce
syke syli sync syne syph syph tabs tabu tace tach tack taco tact
tads tael tags tahr tail tain taka take tala talc tale tali talk
tall tame tamp tams tang tank tans taos tapa tape taps tare tarn
taro tarp tars tart task tass tate tats taus taut tavs taws taxa
taxi teak teal team tear teas teat teds teed teel teem teen tees
teff tegs tela tele tell tels temp tend tens tent tepa term tern
test teth tets tews text thae than that thaw thee them then thew
they thin thio thir this thou thro thru thud thug thus tick tics
tide tidy tied tier ties tiff tike tiki tile till tils tilt time
tine ting tins tint tiny tipi tips tire tirl tiro titi tits tivy
toad toby tods tody toea toed toes toff toft tofu toga togs toil
toit toke tola told tole toll tolu tomb tome toms tone tong tons
tony took tool toom toon toot tope toph topi tops tora torc tore
tori torn toro torr tors tort tory tosh toss tost tote tots tour
tout town tows towy toyo toys trad tram trap tray tree tref trek
tret trey trig trim trio trip trod trop trot trow troy true trug
tsar tsks tuba tube tubs tuck tufa tuff tuft tugs tuis tule tump
tuna tune tung tuns tups turd turf turk turn tush tusk tuts tutu
twae twas twat twee twig twin twit twos tyee tyer tyes tyke tyne
type typo typp typy tyre tyro tzar tzar udos ughs ugly ukes ulan
ulna ulus ulva umbo umps unai unau unbe unci unco unde undo undy
unit unto upas upby updo upon urbs urds urea urge uric urns ursa
urus used user uses utas uvea uvea uvea wabs wack wade wadi wads
wady waes waff waft wage wags waif wail wain wair wait wake wale
walk wall waly wame wand wane wans want wany waps ward ware wark
warm warn warp wars wart wary wash wasp wast wats watt wauk waul
waur wave wavy wawl waws waxy ways weak weal wean wear webs weds
weed week weel ween weep weer wees weet weft weir weka weld well
welt wend wens went wept were wert west wets wham whap what whee
when whet whew whey whid whig whim whin whip whir whit whiz whoa
whom whop whys wich wick wide wife wigs wild wile will wilt wily
wimp wind wine wing wink wino wins winy wipe wire wiry wise wish
wisp wiss wist wite with wits wive woad woes wogs woke woks wold
wolf womb wonk wons wont wood woof wool woos wops word wore work
worm worn wort wost wots wove wows wrap wren writ wuss wych wyes
wyle wynd wynn wyns wyte wyte xyst xyst yack yaff yagi yaks yald
yams yang yank yaps yard yare yarn yaud yaup yawl yawn yawp yaws
yays yeah yean year yeas yech yegg yeld yelk yell yelp yens yerk
yeti yett yeuk yews yids yill yins yipe yips yird yirr ylem yobs
yock yodh yods yoga yogh yogi yoke yoks yolk yond yoni yore your
yowe yowl yows yuan yuca yuch yuck yuga yuks yule yups yurt ywis
ywis zags zany zaps zarf zeal zebu zeds zees zein zeks zerk zero
zest zeta zigs zill zinc zing zins zips ziti zits zoea zoic zone
zonk zoom zoon zoos zori zyme zyme
```

D.9.10 port.data

```
time 6001
date 6002
echo 6003
name 6004
slsC 6005
slsS 6006
```

D.9.11 userDP.data

```
david 'vM(Ay dato
peter Arz`+I earn
paul 0.W3QL pfft
```

D.9.12 userHDP.data

```
david 172522ec1028ab781d9dfd17eaca4427 96320642f0f38ced3f68fa125bdb16ee
Peter d9c14c0d88239177585d6d7122b8d991 fe01ce2a7fbac8fafaed7c982a04e229
Paul ce2711b847b138dcb5a8a0d60c589c0c b03f608e81ecd2ad921f2226bd241907
```

D.10 Configuration Files

D.10.1 Switch Configuration

```
hostname Switch ! no aaa new-model ip subnet-zero ! no file verify
auto spanning-tree mode pvst spanning-tree extend system-id ! vlan
internal allocation policy ascending ! interface FastEthernet0/1
 switchport mode trunk
! interface FastEthernet0/2 ! interface FastEthernet0/3 ! interface
FastEthernet0/4 ! interface FastEthernet0/5 ! interface
FastEthernet0/6 ! interface FastEthernet0/7 ! interface
FastEthernet0/8 ! interface FastEthernet0/9 ! interface
```

```
FastEthernet0/10 ! interface FastEthernet0/11
 switchport access vlan 10
! interface FastEthernet0/12
 switchport access vlan 20
! interface FastEthernet0/13
 switchport access vlan 30
! interface FastEthernet0/14
 switchport access vlan 40
! interface FastEthernet0/15 ! interface FastEthernet0/16 !
interface FastEthernet0/17 ! interface FastEthernet0/18 ! interface
FastEthernet0/19 ! interface FastEthernet0/20 ! interface
FastEthernet0/21 ! interface FastEthernet0/22 ! interface
FastEthernet0/23 ! interface FastEthernet0/24 ! interface
GigabitEthernet0/1 ! interface GigabitEthernet0/2 ! interface Vlan1
 ip address 200.200.5.2 255.255.255.0
 no ip route-cache
! ip default-gateway 200.200.5.1 ip http server ! control-plane !
line con 0 line vty 0 4
 no login
line vty 5 15
 no login
!
```

D.10.2 Router Configuration

```
service timestamps debug datetime msec service timestamps log
datetime msec no service password-encryption ! hostname Router !
boot-start-marker boot-end-marker ! no aaa new-model ! resource
policy ! ip subnet-zero ! ip cef ! interface FastEthernet0/0
 no ip address
 duplex auto
 speed auto
! interface FastEthernet0/0.1
 encapsulation dot1Q 1 native
 ip address 200.200.5.1 255.255.255.0
 no snmp trap link-status
! interface FastEthernet0/0.2
 encapsulation dot1Q 10
 ip address 200.200.1.1 255.255.255.0
 no snmp trap link-status
! interface FastEthernet0/0.3
 encapsulation dot1Q 20
 ip address 200.200.2.1 255.255.255.0
 no snmp trap link-status
! interface FastEthernet0/0.4
 encapsulation dot1Q 30
 ip address 200.200.3.1 255.255.255.0
 no snmp trap link-status
! interface FastEthernet0/0.5
 encapsulation dot1Q 40
 ip address 200.200.4.1 255.255.255.0
 no snmp trap link-status
! interface FastEthernet0/1
 no ip address
 shutdown
 duplex auto
 speed auto
! interface Serial0/0/0
 no ip address
 shutdown
 no fair-queue
 clock rate 125000
! interface Serial0/0/1
 no ip address
 shutdown
 clock rate 125000
! ip classless ! ip http server ! ! control-plane ! ! line con 0
line aux 0 line vty 0 4
 login
! scheduler allocate 20000 1000 !
```

Appendix E

E.1 port1.data

```
time 6001
date 6002
echo 6003
name 6004
slsC 6005
slsS 6006
```

- End of dissertation -