Security in Mobile Ad hoc Networks (MANET)

Developed by

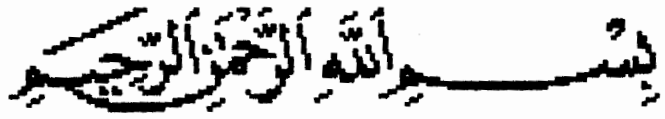
Imran Hameed

Supervised by

Prof. Dr. Khalid Rashid

Department of Computer Science, Faculty of Applied Sciences, International Islamic University, Islamabad

(2004)



**In the name of ALMIGHTY ALLAH,
The most Beneficent, the most
Merciful.**

Department of Computer Science, Faculty of Applied Science, International Islamic University, Islamabad.

Dated: 16 · 8 · **2004**

Final Approval

It is certified that we have read the thesis; entitled **"Security in Mobile Ad hoc Networks (MANET)"** submitted by **Imran Hameed** under University Reg. No. **31-CS/MS/01**. It is our judgment that this thesis is of sufficient standard to warrant its acceptance by the International Islamic Univeristy, Islamabad, for the degree of **MS (Computer Science)**.

Committee

External Examiner
Dr. Abdus Sattar
H# 143, St. 60, I-8/3,
Islamabad.

Internal Examiner
Mr. Mohsin Ali
Faculty Member
Dept't of Computer Science
IIUI.
(Resigned)

Supervisor
Prof. Dr. Khalid Rashid
Dean,
Faculty of Applied Sciences,
International Islamic University,
Islamabad.

A thesis submitted to the
Department of Computer Science,
Faculty of Applied Sciences,
International Islamic University, Islamabad
as a partial fulfillment of the requirements
for the award of the degree of
MS (Computer Sciences)

Declaration

I hereby declare that this software, neither as a whole nor as a part thereof has been copied out from any source. It is further declared that I have developed this software and thesis entirely on the basis of my personal efforts made under the sincere guidance of my teachers. No portion of the work presented in this thesis has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

Imran Hameed
31-CS/MS/01

Dedication

Dedicated to **The Holy Prophet Muhammad** (Allah's grace and peace be upon him) lord of the world and the thereafter. We offer our humblest thanks to him, who made us aware of our creator and guided us to the track, which leads to the success, Who is a symbol of love and affection for all the creatures of Allah.

Acknowledgements

All praise to the Almighty Allah, the most Merciful, the most Gracious, without whose help and blessings, I was unable to complete the project.

Thanks to my Parents who helped me during my most difficult times and it is due to their unexplainable care and love that I am at this position today.

I feel highly privileged in taking opportunity to express my profound gratitude and sense of devotion to my supervisor **Prof. Dr. Kahlid Rashid**, Dean, Faculty of Applied Sciences, International Islamic University, Islamabad for his inspiring guidance, consistent encouragement, sympathetic attitude and dynamic supervision.

I acknowledge my teachers and friends for their help in the project.

Imran Hameed

Project in Brief

Project Title:	Security in Mobile Ad hoc Networks (MANET).
Objective:	To Develop an efficient security mechanism for Mobile communication.
Undertaken By:	**Imran Hameed**
Supervised By:	**Prof. Dr. Khalid Rashid** Dean, Faculty of Applied Sciences & Faculty of Management Sciences, International Islamic University, Islamabad.
Technologies Used:	Microsoft® Visual C++ 6.0, Bluetooth or 802.11 Standard Devices.
System Used:	Pentium® III
Operating System Used:	Microsoft® Windows® 2000 Professional
Date Started:	November, 2002
Date Completed:	March, 2004

Abstract

Mobile ad hoc networks are unique generation of networks offering unrestricted mobility without relying on any infrastructure. In these kinds of networks, hosts rely on each other to keep the network connected, sharing responsibility of network formation and management. Security is a factor of great importance in decentralized communication systems like mobile ad hoc networks. This paper deals with the problem of secure communication and authentication in ad hoc wireless networks. We focus on security mechanisms and notably on the key management mechanisms. Our authentication protocol is inspired from the work of (D. Balfanz & H. Chi Wong, 2002) with a novel diversity. A reliable solution for securing wireless commercial transactions is presented which is practical and industry demanding. In this approach, devices exchange a small amount of public information over location-constrained channel, which will later allow them to complete key exchange over wireless link. It is secure against passive attacks on location-constrained channel and all kinds of wireless attacks on wireless link. Instead of using traditional public key algorithms like RSA, we have included Elliptic cryptography in our approach for generating public/private key pair because it is more suitable for constrained environments (low memory wireless devices).

TABLE OF CONTENTS

Chapter 1
Introduction

1. INTRODUCTION

Ad hoc networking is a wireless networking paradigm for self-organizing networks that until recently has mainly been associated with military battlefield networks. However, with the availability of wireless technologies such as Bluetooth and 802.11 and the development of the next generation networks, civilian applications that exploit the advantages of ad hoc networking are being envisioned.

The advent of ad hoc networks brought with it a flurry of research focused on communication and routing protocols; peer-to-peer, many-to-many, and sensor network protocols were quickly adapted to harness such research. These developments enabled an increased number of applications to harness the benefits of decentralized networks. Examples of such applications range from simple chat programs to shared whiteboards and other collaborative applications. Although intended for diverse audiences and contexts, many of these applications share a common quality: they are information centric. The information transferred may be an ordinary conversation between friends, confidential meeting notes shared among corporate executives, or mission–critical military information. Despite the deployment of information–driven applications such as these, the call for ad hoc network security remains largely unanswered.

Ad hoc security is not, however, a concern that has slipped through the cracks unnoticed: numerous research initiatives have been launched to surmount the challenge. Unfortunately, many of these approaches have been largely ineffective due to their limited scope, unrealistic computational requirements, or inability to address core security issues.

So far most of the research that has been done on ad hoc networking has focused on routing. Other issues such as security and network addressing have received considerably less attention and these issues need to be addressed before any successful applications will appear.

1.1 Ad hoc Networks

Ad hoc networks are an emergent, decentralized communication paradigm that were initially developed explicitly for battlefield settings, where military personnel required a communication medium that did not rely on a centralized entity or wired infrastructure, and had the capability to support lightweight, mobile nodes, such as handheld radios and telephones. More recently, application developers from a variety of domains have embraced the salient features of the ad hoc networking paradigm. Although these applications vary considerably in terms of their use of ad hoc networks, verities of the underlying communication pattern have remained:

- ***Decentralized:*** Central servers, specialized hardware, and fixed infrastructures are necessarily absent. The lack of infrastructure precludes the deployment of hierarchical host relationships; instead, nodes uphold ephemeral and egalitarian relationships. That is, they assume a contributory, collaborative role in the network rather than of dependence.

- ***Amorphous:*** Node mobility and wireless connectivity allow nodes to enter and leave the network spontaneously. Fixed topologies and infrastructures are, therefore, inapplicable.
- ***Broadcast communication:*** The underlying protocols used in ad hoc networking employ broadcast rather than unicast communication.
- ***Addressless messages***: Dynamic network membership necessitates content–based rather than address–based messages. That is, nodes cannot rely on a specific node to provide a desired service; instead, the node must request the service of all nodes currently in the network; nodes capable of providing service respond accordingly.
- ***Lightweight nodes:*** Ad hoc enabled nodes are mobile and, therefore, often small and lightweight in terms of energy and computational capabilities.
- ***Transient:*** The energy restraints and application domains of ad hoc networks often require temporal network sessions. Perhaps the most notable variant in ad hoc network–based applications is network area—the perimeter of the network and number of nodes contained therein. Many research initiatives have envisioned ad hoc networks that encompass thousands of nodes across a wide area. Because wireless nodes are capable of communicating on a very short distance, this vision motivated extensive, often complicated routing protocols. In contrast, we envision ad hoc networks with small areas and a limited number of nodes. Collaborative whiteboards, shared notebooks, and serverless chat applications, which are often used in small, group contexts, are supportive of this vision.

1.1.1 Security Requirements of Ad hoc Networks

The security services of ad hoc networks are not altogether different than those of other network communication paradigms. Specifically, an effective security paradigm must ensure the following security primitives:

- Identity Verification ensures that a malicious node cannot masquerade as a trusted network node.
- Data confidentiality is a core security primitive for ad hoc networks. It ensures that the desired recipient(s) of a given message cannot be understood by anyone else. Data confidentiality in ad hoc networks is typically enabled by symmetric key cryptosystems.
- Data integrity denotes the authenticity of data sent from one node to another. That is, it ensures that a message sent from node A to node B was not modified by a malicious node, C, during transmission. If a robust confidentiality mechanism is employed, ensuring data integrity may be a simple as adding one–way hashes to encrypted messages.
- Availability ensures that the desired network services are available whenever they are needed. Systems that ensure availability seek to combat denial of service and energy starvation attacks.

1.1.2 Security Solution Constraints

Historically, network security personnel have adopted a centralized, largely protective paradigm to satisfy the aforementioned requirements. This paradigm has been effective because the privileges of every node in the network are managed by dedicated machines—authentication servers, firewalls, etc.—and the professionals who maintain them. Membership in such a network allows individual nodes to operate in an open fashion—sharing sensitive files, allowing incoming network connections—because they are implicitly guaranteed that malicious users from the outside world will not be allowed to access their node(s).

Although they were attempted early in the evolution of ad hoc networks, attempts to adapt similar client–server solutions to a decentralized environment were largely ineffective.

- ***Lightweight:*** Solutions must minimize the amount of computation and communication required to ensure the security services to accommodate the limited energy and computational resources of mobile, ad hoc–enabled devices.
- ***Decentralized:*** Like ad hoc networks themselves, attempts to secure them must be ad hoc: they must establish security without a priori knowledge or reference to centralized, persistent entities. Instead, security paradigms must levy the cooperation of all trustworthy nodes in the network.
- ***Reactive:*** Ad hoc networks are dynamic: nodes—trustworthy and malicious—may enter and leave the network spontaneously and unannounced. Security paradigms must react to changes in network state; they must seek to detect compromises and vulnerabilities; they must be reactive, not protective.
- ***Fault–Tolerant:*** Wireless transfer mediums are known to be unreliable; nodes are likely to leave or be compromised without warning. The communication requirements of security solutions should be designed with such faults in mind; they mustn't rely on message delivery or ordering.

Naturally, these are not stringent requirements: specific applications may relax some or all of the above requirements based on their domain and the sensitivity of the information involved. Moreover, many ad hoc network applications do not require 2–party secure communication; instead, achieving broadcast or group security may be all that is needed.

1.1.3 Security Attacks

Security attacks can be classified in the following two categories [3] depending on the nature of the attacker:

- ***Passive attacks:*** The attacker can only eavesdrop or monitor the network traffic. Typically this is the easiest form of attack and can be performed without difficulty

in many networking environments, e.g. broadcast type networks such as Ethernet and wireless networks.

- ***Active attacks:*** The attacker is not only able to listen to the transmission but is also able to actively alter or obstruct it. Furthermore depending on the attacker's actions, the following subcategories can be used to cover the majority of attacks.
- ***Eavesdropping:*** This attack is used to gain knowledge of the transmitted data. This is a passive attack which is easily performed in many networking environments as mentioned above. However this attack can easily be prevented by using an encryption scheme to protect the transmitted data.
- ***Traffic analysis:*** The main goal of this attack is not to gain direct knowledge about the transmitted data, but to extract information from the characteristics of the transmission, e.g. amount of data transmitted, identity of the communicating nodes etc. This information may allow the attacker to deduce sensitive information, e.g. the roles of the communicating nodes, their position etc. Unlike the previously described attack this one is more difficult to prevent.
- ***Impersonation:*** Here the attacker uses the identity of another node to gain unauthorized access to a resource or data. This attack is often used as a prerequisite to eavesdropping. By impersonating a legitimate node the attacker can try to gain access to the encryption key used to protect the transmitted data. Once this key is known by the attacker, she can successfully perform the eavesdropping attack.
- ***Modification:*** This attack modifies data during the transmission between the communicating nodes, implying that the communicating nodes do not share the same view of the transmitted data. An example could be when the transmitted data represents a financial transaction where the attacker has modified the transactions value.
- ***Insertion:*** This attack involves an unauthorized party, who inserts new data claiming that it originates from a legitimate party. This attack is related to that of impersonation.
- ***Replay:*** The attacker retransmits data previously transmitted by a legitimate node.
- ***Denial of service:*** This active attack aims at obstructing or limiting access to a certain resource. This resource could be a specific node or service or the whole network.

1.1.4 Security Mechanisms

Most of the security services previously mentioned can be provided using different cryptographic techniques. The following subsections give an overview of which techniques are used to provide each of the services.

- ***Confidentiality:*** The confidentiality service can be of two different types. The most common type of confidentiality requirement is that transmitted information

should not be exposed to any unauthorized entities. A more strict confidentiality requirement is that the very existence of the information should not be revealed to any unauthorized entities.
The first type of confidentiality requirement only requires protection from eavesdropping attacks and can be provided using an encryption scheme. The stricter requirement implies that the service must also provide protection against traffic analysis. Such a service will typically require additional mechanisms along with some encryption scheme.

- ***Integrity:*** The integrity service can be provided using cryptographic hash functions along with some form of encryption. When dealing with network security the integrity service is often provided implicitly by the authentication service.
- ***Authentication:*** Authentication can be provided using encryption along with cryptographic hash functions.
- ***Non-repudiation:*** Non-repudiation requires the use of public key cryptography to provide digital signatures. Along with digital signatures a trusted third party must be involved.
- ***Availability:*** The availability is typically ensured by redundancy, physical protection and other non cryptographic means, e.g. use of robust protocols.

1.2 Literature Survey

This section provides a survey of dominant research initiatives in the ad hoc security arena, past and present. Each problem area is addressed except secure routing, which is not aligned with our vision of ad hoc networks.

1.2.1 Authentication

Authentication denotes the accurate, absolute identification of users who wish to participate in the network. Historically, authentication has been accomplished by a central, well-known authentication server. The role of the server is to maintain a database of entities, or users, and their corresponding, unique id. The id may be a digital certificate, public key, or both. When someone wishes to join a network session—for example a secure chat session—the applicant proposes its identity, which is then verified by the authentication server. Unfortunately, the much known notion of a central server contradicts the essence of ad hoc networks. It is, therefore, necessary to define an authentication paradigm that does not entail a central entity to verify node identity.

Techniques for authentication in ad hoc networks can generally be grouped into three categories, namely Trusted Third Parties, Chain of Trust and Location-Constrained Authentication.

1.2.1.1 Trusted Third Parties

One of the most rudimentary approaches to authentication in ad hoc networks uses a trusted third-party (TTP). Every node that wishes to participate in an ad hoc

network obtains a certificate from a universally trusted third-party. When two nodes wish to communicate, they first check to see if the other node has a valid certificate.

Although popular, the TTP approach is laden with flaws. Foremost, it probably isn't reasonable to require all ad hoc network-enabled devices to have a certificate. Second, each node would have to have a unique name. Although this is reasonable in large networks, it is a bit too restrictive in an ad hoc setting.

Recent research has introduced more suitable variations of TTPs. Preliminary phases of this thesis have been reserved for a close analysis of these solutions.

1.2.1.2 Chain of Trust

The TTP model essentially relies on a fixed entity to ensure the validity of all nodes' identities. In contrast, the chain of trust paradigm relies on any node in the network to perform authentication. That is, if a node wishes to enter a network session, it may ask any of the nodes to enter. If the asked node is able and willing to authenticate the requesting node, the node is allowed into the session.

Most implementations of the chain of trust require that a node have the signed public key of the requesting node before it can authenticate. In other words, a node must have communicated with the requesting node previously in order to authenticate it.

This model assumes that all nodes in the current network session are trustworthy, which may not be a valid assumption for all ad hoc applications. Further, the chain of trust does not handle the situation wherein none of the authenticated nodes can authenticate the new node.

1.2.1.3 Location-Constrained Authentication

Location–Constrained authentication levies the fact that most ad hoc networks exist in a small area. Authentication is performed between two nodes that are necessarily close to one another—physically touching in some implementations. The proximity of nodes is often dictated by the protocol used to perform the authentication. Bluetooth and infrared are two of the most widely used protocols for this form of authentication. Although it may not seem obvious, location-constrained authentication is potentially very secure. The security is obtained from physical assurance and tamper-detection. That is, the authenticating node can be reasonably certain that the node, it thinks, it is authenticating is the node actually to be authenticated (i.e., there is no man in the middle) by physical indications—the transfer light on the requesting node is blinking, the person operating the device is physically present, etc.

Although location-constrained authentication is well-suited for most applications with a single end-point, it is not feasible for large, group-based settings.

1.2.2 Key Management

Data confidentiality is often enabled by a symmetric key. Symmetric implies that the same key and algorithm is used to encrypt and decrypt messages. Although originally developed for 2–party settings, symmetric keys are appropriate in many–to–many

contexts as well. In these settings, each member has knowledge of the shared, or group key, and each message addressed to the group is first encrypted with the key. When a member receives the message, it decrypts it using the group key. Unlike public or asymmetric keys, symmetric key cryptosystems are potentially well–suited for ad hoc networks due to their relative efficiency. Unfortunately, distributing the group key to all group members is a non–trivial problem. Key management paradigms address this problem. The figure below illustrates the Digital signatures process.

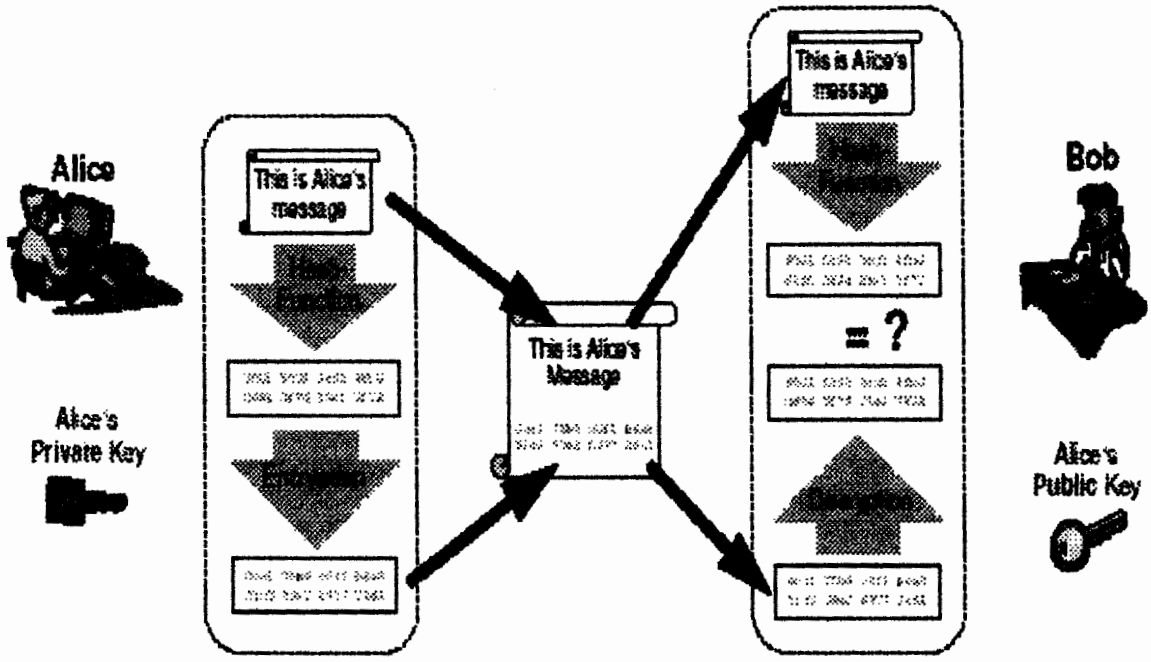


Figure 1.1 Illustrations of the Digital Signatures

1.2.3 Generalized Diffie-Hellman

- **Overview**

Several years after its inception, the well–known Diffie–Hellman key agreement protocol was generalized to n participants—thereby allowing an arbitrary number of nodes to contribute to and share a session key. The revised protocol, henceforth referred to as Generalized Diffie–Hellman, is nearly identical to its predecessor: members agree on an a priori G and Z^p; each member then generates its own secret Ni → G.

Generalized Diffie–Hellman consists of two stages—upflow and down flow. Each member's contributions are collected during the upflow stage, and the resultant intermediate values are broadcast to the group in the down flow stage.

The setup for GDH is identical to that of two–party Diffie–Hellman: all participants, M_1... M_n, choose a cyclic group, G, of order q, and a generator, in G; each member then chooses a secret share, Ni → G.

- **Upflow**

During the upflow, each member, Mi performs a single exponentiation and appends it to the flow, and forwards the flow to Mi.

$$(N_k/K\ [i,j])/j\ [l,i]$$

$$M_i \longrightarrow M_{i+1}$$

The upflow stage terminates and the downflow commences when $M_n = M_i$ — when the last member has received the upflow.

$$(N_1, N_1N_2, \ldots, N_1N_2 \ldots N_{n-1})$$

$$M_{n-1} \longrightarrow M_n$$

Upon receipt of the upflow, M_n calculates the new group key, Kn by taking exponentiation of the last intermediate value in the flow:

$$K = K_n = (N_1N_2 \ldots N_{n-1})\ N_n$$

Once Kn has been calculated, Mn commences the downflow.

- **Downflow**

The downflow is initially comprised of n - 1 intermediate values, N_1N_n, $N_1N_2N_n$,..., $N_1N_2 \ldots N_{n-2}N_n$ exponentiated to the nth group member's secret, N_n. M_n sends the downflow to M_{n-1}.

$$(N_1N_n, N_1N_2N_n, \ldots, N_1N_2 \ldots N_{n-2}N_n)$$

$$M_{n-1} \longleftarrow M_n$$

Upon receipt of the downflow, each member, Mi removes its own intermediate value, $N_1N_n, \ldots N_{i-1}N_{i+1} \ldots N_n$, calculates the group key, $K_n = (N_1N_n, \ldots N_{i-1}N_{i+1} \ldots N_n)N_i$ exponentiates the remaining i - 1 intermediate values in the flow, and forwards the flow to its predecessor, M_{i-1}.

$$N_1N_nN_{n-1} \ldots N_{i+1}N_i$$
$$N_1N_nN_nN_{n-1} \ldots N_{i+1}N_i$$
$$\ldots$$
$$N_1N_2 \ldots N_{i-2}N_{n-2}N_n \ldots N_{i+1}N_i$$

$$M_{i-1} \longleftarrow M_i$$

The downflow terminates when $M_i = M_1$.

- **Analysis**

Although Generalized Diffie–Hellman was developed for a wired environment, it is surprisingly well–suited for some ad hoc environments. Its ability to accommodate dynamic group membership without a secure communication medium is particularly desirable. Although Diffie–Hellman does not scale particularly well, it is sufficiently efficient for small group contexts. Additionally, the developers of GDH have since designed several variants of the original GDH protocol. The resulting protocols are more efficient in terms of the number of rounds required and message size and quantity.

None of this is to say that GDH is a panacea, however. Because it was developed for a unicast environment, it uses point-to-point communication patterns. Point–to–point techniques such as these have share common shortcoming: network faults. That is, what if a member leaves, dies, or is otherwise compromised during key generation? Unfortunately, the protocol does not sufficiently deal with these failures. In addition, authentication is assumed in GDH, as it is with many key management paradigms.

1.2.4 Semi–Centralized Key Management—The Resurrecting Duckling

- **Overview**

Unlike GDH, The Resurrecting Duckling[4] is a key management system that was developed explicitly for ad hoc networks. We have deemed this approach "semi–centralized" as it is, for the most part, an adaptation of a popular form of centralized key management used in client–server architectures.

- **Design**

The Resurrecting Duckling employs a public key infrastructure with a replicated Certificate Authority (CA). Each node in the network has a public/private key pair. In keeping with the public key paradigm, the public key is widely-distributed, while the private key is known only to the node to which it corresponds. The CA is responsible for signing certificates which bind public keys to individual nodes. Because ad hoc networks are prone to failure, and the CA must be persistent, the CA is replicated and signatures are created using a (n, t + 1) threshold cryptosystem. (n, t + 1) threshold signature schemes allow n parties to share the ability to sign the public key when only t + 1 members are present.

In the proposed solution, the CA is replicated n times, thereby allowing up to t servers to be compromised without loss of the signature service. Each server stores a "share" of the private key (s_1 , s_2 , …, s_n). There is yet another specialized node in the infrastructure—the combiner. The combiner, c is responsible for combining each of the shares and validating the contribution of each of the n servers. If a contribution is deemed invalid, it is discarded by the combining node.

Once the public key infrastructure is in place, the group key can be generated and distributed to the group securely: the server nodes need only encrypt the group key once for every member using that member's public key. When the member receives the message, it simply decrypts the message using its private key.

- **Analysis**

Despite its widespread acceptance in the ad hoc domain, the Resurrecting Duckling is not very feasible. Most notably, it requires persistent server nodes. Although replication and threshold schemes are employed for these servers, their existence remains paramount to the integrity of the key management system; and compromising t + 1 of them will compromise the system.

Further, the protocol requires a reliable, addressable transfer medium. In defense of [4], the protocol does not require a synchronous transfer medium—there is no bound on message delivery or processing times—which makes the protocol slightly more desirable for the ad hoc context. Nonetheless, this largely adaptive approach is unnecessarily centralized, and therefore, undesirable.

1.2.5 Tree-based Key Management

Tree structures provide a method of key agreement and management that has several desirable qualities. The very nature of being in a tree formation reduces the complexity from O(N) to O(N log N) and provides a well-suited structure for dynamic membership changes. As elements change membership status, the tree requires only a few key-node updates to affect the whole tree. This means only a few messages are passed, reducing computational complexity and bandwidth. For this discussion, consider a purely multicast network of N nodes, physically arranged in any order, with all members being within range of all others (maximal reachability). As with "standard" key agreement protocols, trees must also ensure perfect forward and backward secrecy and key freshness.

1.2.5.1 Logical Key Hierarchy (LKH)

This scheme places all group members at the leaves of an order-m B-Tree, and consists of a series of intermediate subgroup keys. Built from the bottom up, intermediate keys higher in the tree serve greater numbers of group members, where the root of the tree is the group key.

The keys delivered to each member are "shared", in essence, with no other nodes with the exception of the root. All intermediate keys, however, are shared by all dependents (children in this subtree) of the inner-tree node.

This group tree, in most LKH implementations, is stored on one designated host. It is clear from the role that this node must be a nearly permanent fixture in the model, as failure would place the group in a state of near total confusion.

Membership changes initially affect only those keys which are above and to the most direct path to the root. As these intermediate values change, a secondary flow will change all other members' keys to guarantee perfect forward secrecy and key freshness.

This minimal-impact affect of trees initiated by membership changes helps key distribution problems immensely.

1.2.5.2 Extensions to Logical Key Hierarchy (LKH)—LKH++

An extension to Logical Key Hierarchy that uses hashing and intermediate key values is described in "Efficient and Secure keys management for wireless mobile communications" by Roberto Di Pietro, Luigi V. Mancini, and Sushil Jajodia. Using a key distribution node, the "center", this protocol attempts to require a minimal amount of network communication and message computation to generate and share the group key.

As in other techniques, intermediate values between any given node and the root are required to be maintained by each node.

The main difference between this and other protocols is the use of hashing. They describe how a new key, K' is to be sent to the group, encrypted with the current group key, K. The center computes the one-way hash of K (defined as H (K)), followed by the Xor of the new key and the hash of the current $K_h = H(K) \_ K_0$, and sends this to the group. Group members then recover the new key by computing $K_0 = H(K) \_ K_h$. By hashing the values, the key is never sent "in the clear" or even as a collection of partial values, rather it is sent in a "pseudo-encrypted" form—only those network elements that know the current key will be able to decode and retrieve the new key. The benefit of this technique is a general reduction in resource consumption. They noted a 50% drop in computations done by the key center, and a 50% reduction in required bandwidth for group formation.

1.2.5.3 Tree-Based Group Diffie-Hellman (TGDH)

Work done by Kim, Perrig and Tsudik [1] introduces a fairly simple and flexible structure for keys using singly-rooted binary trees that are easily distributed throughout the group. Using the elegance of trees and the provable guarantees of Diffie-Hellman key exchange, this is generally a simple construct. They point out that their structure provides a set of desirable qualities others do not; ability to manage cascaded events, and the decoupling of physical location in the group versus the logical location in the tree structure. This helps ensure a highly flexible organization, and prevents unbalanced trees related to "membership sponsors" found in some key organization techniques. (See the Octopus protocol for a basic example of this type of issue.)

As with two-party Diffie-Hellman, key material in this scheme is divided into a public set and a private set of values. Each node uses a pair of values, a key and a blinded key, that is it is computationally infeasible to derive the key material from the final key, and knows the path from itself to the root node. This path also provides a set of key values from itself to the group key, located at the root. Computation of any intermediate key value requires the knowledge of the public value of one child node, and the private value of the other. This is a mutually-blind requirement, in that one child cannot compute the others' value, and a third party can not determine either private value, as in two-party Diffie-Hellman key agreement. Given a leaf nodes' private value and all intermediate blind key values from itself to the root, a particular node can compute the group key with relative ease. They also mention that if all group members know all blind key values from the tree, membership changes are handled in a more robust manner.

Compared to Generalized Diffie-Hellman, version 3 (GDH.3), Tree-based Group Diffie-Hellman performed very well. The research in [1] showed improved communication and computation over GDH.3, minimizing many O (N) operations to O(N log N) in complexity. With 1024 group members, random membership changes yielded nearly linear results that appear to behave well with both larger and smaller groups. Their test results led them to assert their design would work best under low- to medium-delay networks.

1.3 The RSA Signature Scheme

The RSA signature scheme was discovered by Rivest, Shamir, and Adleman [51]. It was the first practical signature scheme based on public-key techniques. The underlying computationally hard mathematical problem for the RSA signature scheme is the integer factorization problem (IFP)[41]:

Given a composite number n that is the product of two large prime numbers p and q, find p and q.

Since it is part of the public key, an adversary has access to the modulus n. Once p and q are computed, the system is broken and the attacker can use Algorithm 1.3.1 to determine the secret key.

Algorithm 1.3.1 summarizes how a key pair, i.e. a public and the corresponding private key, for the RSA signature scheme can be generated. The steps for signing a message and the steps for verifying a message can be found in Algorithm 1.3.3.

- **Key generation for the RSA signature scheme**

OUTPUT: The public key (n, e) and the private key d.

1. Generate two large distinct random primes p and q, each roughly the same size.
2. Compute $n = pq$ and $\Phi = (p - 1)(q - 1)$.
3. Use the extended Euclidean algorithm to compute the unique integer d, $1 < d < \Phi$, such that $ed \equiv 1 \pmod{\Phi}$.
4. The public key is (n, e); the private key is d.

- **RSA signature generation**

INPUT: The message m, the public key (n; e), and the private key d.
OUTPUT: The digital signature s.

- Compute the hash value of the message $\tilde{m} = h(m)$, an integer in the range [0, n -1].
- Compute $s = m^d \bmod n$.
- The signature for m is s.

- **RSA signature Verification**

INPUT: The message m, the public key (n, e) of the signer and the signature s on m.

- Compute $\sim m = S^e \bmod n$.
- Verify that $\sim m = h(m)$; if not, reject the signature.

1.4 The Elliptic Curve Digital Signature Algorithm (ECDSA)

A variant of DSA based on elliptic curves is ECDSA. It was first proposed in 1992 by Scott Vanstone [63]. The underlying computationally hard mathematical problem is the Elliptic Curve Discrete Logarithm Problem (ECDLP)[41]:

Given an elliptic curve E defined over $\mathbf{F_q}$, a point P ∈ E($\mathbf{F_q}$) of order n, and a point Q ∈ E ($\mathbf{F_q}$), determine the integer l, $0 \leq l \leq n - 1$, such that Q = lP, provided that such an integer exists.

This discrete logarithm problem over elliptic curves is considered to be significantly harder than the DLP over Zp, which is the mathematical basis for DSA. Therefore, the strength per-key-bit is substantially higher than in DSA and, hence, smaller parameters (keys) can be used for elliptic curve cryptosystems to achieve equivalent levels of security.

In order to facilitate interoperability, the domain parameters for ECDSA, which are the parameters of the curve E, the underlying finite field Fq and a base point G ∈ E (Fq), have to be negotiated and agreed upon by the communication partners. The curve is usually determined by its two parameters a and b and the curve equation. For the finite field F_2 m, the curve equation is given by the equation

$$y^2 + xy = x^3 + ax^2 + b$$

which is the same for all m. The base point G is defined by its affine coordinate's xG and yG .Usually, the order n of the point G is also part of the domain parameters. Algorithm 1.4.1 summarizes how a key pair, i.e. a public and the corresponding private key, for the ECDSA signature scheme can be generated. The steps for signing a message are given in Algorithm 1.4.2 and the steps for verifying a message can be found in Algorithm 1.4.3. Signature generation and verification requires the computation of the hash value of the message using the Secure Hash Algorithm (SHA-1) [45], which was proposed by the U.S. National Institute for Standards and Technology (NIST).

- **Key Generation for the ECDSA**

INPUT: The elliptic curve domain parameters.
OUTPUT: The public key Q and the private key d.

- Select a random integer d in the interval [1; n - 1].
- Compute Q = dG.
- The public key is Q and the private key is d.

- **ECDSA Signature Generation**

INPUT: The message m, the elliptic curve domain parameters, the public key Q, and the private key d.
OUTPUT: The digital signature (r; s).

- Select a random integer k; $0 < k < n$.
- Compute $kG = (x_1, y_1)$ and convert x_1 to an integer.
- Compute $r = x_1 \bmod n$. If $r = 0$ then go to Step 1.
- Compute $k^{-1} \bmod n$.
- Compute SHA-1(m) and convert this bit string to an integer e.
- Compute $s = k^{-1} (e + dr) \bmod n$. If $s = 0$ then go to Step 1.
- The signature for the message m is (r; s).

- **ECDSA Signature Verification**

INPUT: The elliptic curve domain parameters, the message m, the public key Q of the signer and the signature (r; s).

- Verify that r and s are integers in the interval [1; n - 1]
- Compute SHA-1(m) and convert this bit string to an integer e.
- Compute $w = s^{-1} \bmod n$.
- Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
- Compute $X = u_1G + u_2Q$.
- If $X = 0$, then reject the signature. Otherwise, convert the x-coordinate of X to an integer x_1, and compute $v = x_1 \bmod n$.
- Accept the signature if and only if $v = r$.

1.5 Project

Imagine a situation in which bank customers are standing in a queue and two or three bank employees are dealing them in making transactions. The customers have to spend a lot of time and also have to wait for their turn in the queue. The bank needs more employees for giving better service to its customers. This scenario, both for the bank and the customers, can be improved by deploying ad hoc networking using Resurrecting Duckling Policy[5,6], with some extensions in our idea. Here, we have also taken inspiration from the work of D. Balfanz [22].

Our idea is elaborated as: A user (Bank customer) enters the bank, with his laptop or PDA, to perform some transactions. The bank has mounted the Infrared Bar Code at one side in the bank and also owns a wireless radio link network (e.g. Bluetooth [18] or 802.11). The user or customer enters the bank premises, would walk up to the Infrared bar code and briefly establish physical contact between infrared bar code and his laptop or PDA. This process is termed as ***Pre-authentication***[1], which is done using a ***link-constrained channel***[2] [22].

[1] First phase in which duckling and mother exchange some secret information over location-constrained channel.

[2] This notion of location-constrained channel was introduced by Stajno and Anderson [3, 4], as a part of "Resurrecting Duckling" model for ad hoc networks.

During Pre-authentication, their public keys will be exchanged. Now the user can sit in the bank premises and his laptop or PDA can then perform standard SSL/TLS[19] key exchange with the bank server over the wireless link (e.g. Bluetooth[18] or 802.11), since he owns a secret key to perform secure transactions and can establish an authenticated and secret communication channel.

Such an exchange of pre-authenticated data ensures the user that he wants to communicate with the device (infrared bar code) by using a special, link-constrained side channel to exchange a small amount of cryptographic information. That information can be used to authenticate standard key exchange protocols performed over wireless link.

Once the pre-authentication is completed, the devices proceed to establish a secure connection between them over the main wireless link. To this end, they can use any established public-key-based key exchange protocol which requires them to prove possession of a particular private key (e.g., SSL/TLS, SKEME IKE, etc.), which will correspond to the public key committed to in the pre-authentication step.

After secret key exchange, the customer or user will send his/her account number along with the password or secret code (which can be issued at the time of opening an account in the bank and can be changed at any time through bank website) to the bank server. The A/C No. plus code or password will be encrypted with secret key (Session key) exchanged during pre-authentication. After this phase, the user can perform his transactions securely over the wireless link (e.g. Bluetooth[18] or 802.11).

We can summarize the basic scheme for pre-authentication as follows.

A) Pre-authentication over location-constrained channel

$$X \longrightarrow Y: H(KU_X)$$

$$Y \longrightarrow X: H(KU_Y)$$

B) Authentication over wireless channel with SSL/TLS

$$X \longrightarrow Y: TLS\_CLIENT\_HELLO$$

..... And so on.

The various symbols denote:

X: Customer's Laptop

Y: Bar code device

KU_X, KU_Y: Public key belonging to X and Y respectively.

$H(KU_X)$, $H(KU_Y)$: one-way hash of encoding of corresponding keys.

1.5.1 Characteristics of Location-constrained Channel

Communication technologies that have inherent physical limitations in their transmissions are good candidates. For example, audio (both the audible and ultrasonic range), which has limited transmission range and broadcast characteristics, can be used by a group of PDA's in a room to demonstratively identify each other. For situations that require a single communication endpoint, channels with directionality such as infrared are natural candidates.

The channel be impervious (or resistant) to eavesdropping. For example, Anderson and Stajano use secret data, such as a symmetric key, exchanged across the location-constrained channel to allow participants to authenticate each other. As a result, that authentication protocol is vulnerable to a passive attacker capable of eavesdropping on the location-constrained channel, thereby obtaining the secrets necessary to impersonate one of the legitimate participants. A location-constrained channel used to exchange such secret pre-authentication data must therefore be very resistant to eavesdropping.

We therefore propose that any physically limited channel suitable for demonstrative identification, on which it is difficult to transmit without being detected by at least one legitimate participant (human or device), is a candidate for use as a pre-authentication channel. Such candidates include: contact, infrared, near-field signaling across the body, and sound (both audible and ultrasound). The amount of data exchanged across the pre-authentication channel is only a small fraction of that sent across the main wireless link, and so we can use channel media capable only of low data rates.

1.5.2 Advantages of Pre-authentication

Because legitimate participants would only communicate with entities from which they had received pre-authentication data, we would now require an attacker to perform an active attack – to be able to transmit –not only in the main wireless medium, but also in the location-constrained channel. Because of the physical limitations of transmission on location-constrained channels, it is significantly harder for an attacker to passively eavesdrop on them, not to mention to actively transmit.

For such an active attack to succeed, the attacker must not only transmit on the location-constrained channel, but must do so without being detected by any legitimate participant.

The difficulty of monitoring a pre-authentication for such unwanted participation depends on the type of channel used and the number of legitimate parties involved. The more directed the channel and the smaller the number of parties, the easier it is to monitor. Note that, because of the physical limitations of the channels used and this monitoring requirement, it is only possible to use our techniques to pre-authenticate devices that are physically co-located at the time of first introduction.

1.5.3 Advantages of our Approach

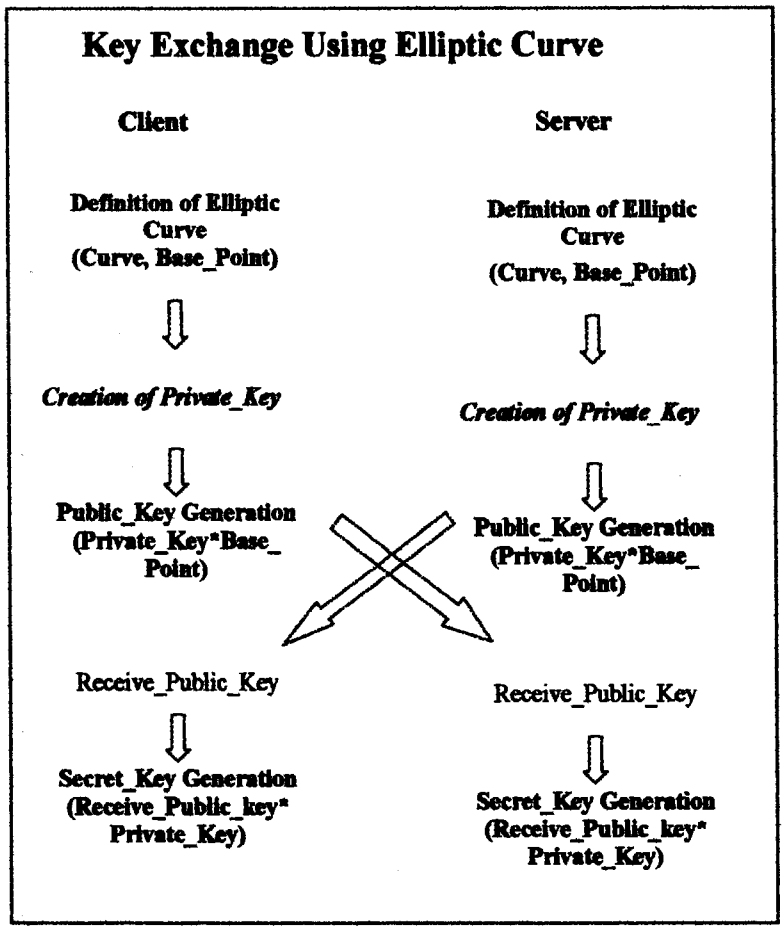
This idea of pre-authentication has been generalized to secure arbitrary peer-to-peer ad hoc interactions using a wide variety of key exchange protocols. We have introduced the use of Elliptic Curve Cryptography and can remove the secrecy requirements on link-constrained channels used to authenticate key exchange protocols. More importantly, it allows us to expand the range of key-exchange protocols, which can be authenticated in this manner to include almost any standard public-key-based protocol. As a result, this approach can also be used with an enormous range of devices, protocols and applications.

This approach is significantly more secure than previous approaches; as we force an adversary to mount an active attack on the location-constrained channel itself in order to successfully subvert an ad-hoc exchange. Previous approaches (e.g., use of unauthenticated Diffie-Hellman key exchange) are either vulnerable to either active attacks in the main wireless channel, or, in the case of Anderson and Stajano [5, 6], to passive (eavesdropping) attacks in the location-constrained side channel.

1.5.4 *ECC-based SSL In Our Banking Scenario*

In our approach, we have integrated ECC into the Secure Socket Layer Protocol. We chose SSL because it is the most popular and trusted security protocol on the Web. In the form of HTTPS (HTTP secured using SSL), SSL is single handedly responsible for the widespread adoption of e-commerce and many emerging wireless devices too now have SSL capabilities.

We have added ECC support to OpenSSL [10], the most widely used open source implementation of SSL. We have used the OpenSSL0.9.6b cryptographic library for our implementation. We selected ECC in place of RSA because ECC offers the highest strength per bit of any known public-key cryptosystem. ECC not only uses small keys for equivalent strength compared to traditional public key cryptosystems like RSA, the key size disparity grows a security needs increase. This makes it attractive especially for constrained wireless devices because smaller keys result in power, bandwidth and computational savings


Key Exchange Using Elliptic Curve
Client
Server
Definition of Elliptic Curve (Curve, Base_Point)
Definition of Elliptic Curve (Curve, Base_Point)
Creation of Private_Key
Creation of Private_Key
Public_Key Generation (Private_Key*Base_ Point)
Public_Key Generation (Private_Key*Base_ Point)
Receive_Public_Key
Receive_Public_Key
Secret_Key Generation (Receive_Public_key* Private_Key)
Secret_Key Generation (Receive_Public_key* Private_Key)

Chapter 2
Analysis

2. SYSTEM ANALYSIS

At a technical level, software engineering begins with a series of modeling tasks that lead to a complete specification of requirements and a comprehensive design representation for the software to be built. The Analysis model, actually a set of models, is the first technical representation of a system. Over the years many methods have been proposed for analysis modeling. However just two models now dominate the analysis modeling landscape. The first, *structured analysis* is a classical modeling method and the other approach is *object oriented* method. We have used the first modeling technique for the analysis of the software Authentication & Authorization in Mobile Ad hoc networks

2.1 Object-Oriented Analysis

It is a method of analysis that examines the requirements of end-user from the perspective of objects and classes found in the vocabulary of problem domain.

2.1.1 A Unified Approach to Object-Oriented Analysis

Over the past decade, Grady Booch, James Rumbaugh, and Ivar Jacobson have collaborated to combine the best features of their individual object-oriented analysis and design methods into a unified method. The result, called the *Unified Modeling Language* (UML), has become widely used throughout the industry.

UML allows a software engineer to express an analysis model using a modeling notation that is governed by a set of syntactic, semantic, and pragmatic rules.

In UML a system is represented using five different "views" that describe the system from distinctly different perspectives. Each view is defined by a set of diagrams. The following views are presented in UML:

- *User Model View:* This view represents the system (product) from the user's (called *actors* in UML) perspective. The use-case is the modeling approach of choice for the user model view. This important analysis representation describes a usage scenario from the end-user's perspective.
- *Structural Model View:* Data and functionality are viewed from inside the system. That is, static structure (classes, objects, and relationships) is modeled.
- *Behavioral Model View:* This part of the analysis model represents the dynamic or behavioral aspects of the system. It also depicts the interactions or collaborations between various structural elements described in the user model and structural model views.
- *Implementation Model View:* The structural and behavioral aspects of the system are represented as they are to be built.
- *Environment Model View:* The structural and behavior aspects of the environment in which the system is to be implemented are represented

In general, UML analysis modeling focuses on the user model and structural model views of the system. UML design modeling addresses the behavioral model, implementation model, and environmental model views.

2.1.2 Domain Analysis

Analysis for object-oriented systems can occur at many different levels of abstractions. At the business or enterprise level, the techniques associated with OOA can be coupled with a business process engineering approach in an effort to define classes, objects, relationships, and behaviors that model the entire business. At the business area level, an object model that describes the workings of a particular business area (or a category of products or systems) can be defined. At an application level, the object model focuses on specific customer requirements as those requirements affect an application to be built.

We will conduct OOA at a middle level of abstraction. This activity, called *domain analysis*, is performed when an organization wants to create a library of reusable classes (components) that will be broadly applicable to an entire category of applications.

2.1.2.1 Reuse and Domain Analysis

Object-technologies are leveraged through reuse. The benefits derived from reuse are consistency and familiarity. Patterns within the software will become more consistent, leading to better maintainability. Be certain to establish a set of reuse "design rules" so that these benefits are achieved.

2.1.2.2 The Domain Analysis Process

Software domain analysis is the identification, analysis and specification of common requirements form a specific application domain, typically for reuse on multiple projects within that application domain ... [Object-oriented domain analysis is] the identification, analysis, and specification of common, reusable capabilities within a specific application domain, in terms of common objects, classes, subassemblies, and frameworks.

The goal of domain analysis is straightforward: to find or create those classes that are broadly applicable, so that they may be reused.

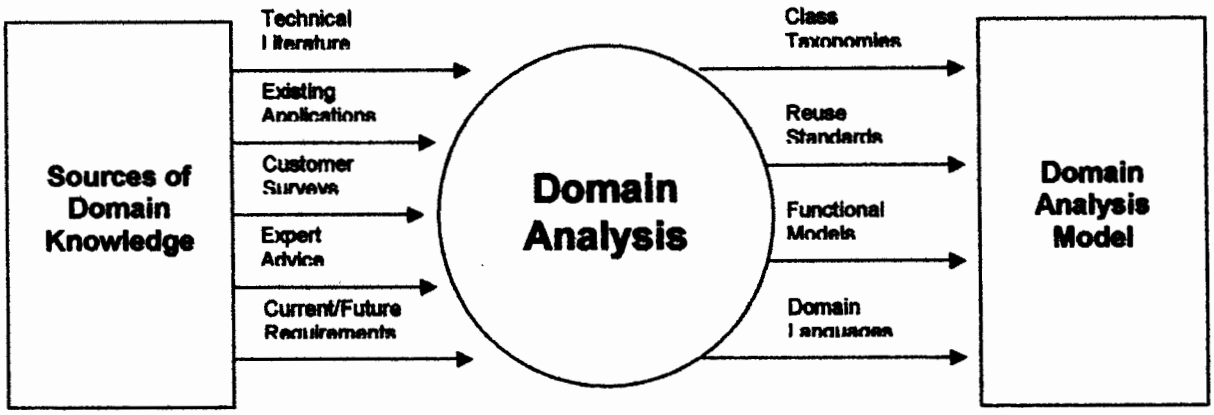


Figure 2.10 Input and Output for Domain Analysis

Domain analysis may be viewed as an umbrella activity for the software process. By this we mean that domain analysis is an ongoing software engineering activity that is not connected to any one software project. In a way, the role of a domain analyst is similar to the role of master tool smith in a heavy manufacturing environment. The job of the tool smith is to design and build tools that may be used by many people doing similar but not necessarily the same jobs. The role of the domain analyst is to design and build reusable components that may be used by many people. Working on similar but not necessarily the same applications.

Figure 2-10 illustrates key inputs and outputs for the domain analysis process. Sources of domain knowledge are surveyed in an attempt to identify objects that can be reused across the domain. In essence domain analysis is quite similar to knowledge engineering. The knowledge engineer investigates a specific area of interest in an attempt to extract key facts that may be of use in creating an expert system of artificial neural network. During domain analysis, *object* (and class) *extraction* occurs.

The following series of activities are performed during the domain analysis process:

2.1.2.3 The Domain to be Investigated

To accomplish this we isolated the business area, system type, or product category of interest. Next, both OO and non-OO "items" were extracted. OO items include specifications, designs, and code for existing OO application classes; support classes (e.g. GUI classes); commercial off-the-shelf (COTS) component libraries that are relevant to the domain; and test cases. Non-OO items encompass policies, procedures, plans, standards, and guidelines; parts of existing non-OO applications; and COTS non-OO software.

2.1.2.4 Categorization of Items Extracted from the Domain

The items were organized into categories and the general defining characteristics of the category were defined. A classification scheme for the categories was proposed and naming conventions for each item were defined. Classification hierarchies were established.

2.1.2.5 Collection of Representative Sample of Application in the Domain

To accomplish this activity, we ensured that the application in question had items that fit into the categories that have already been defined. During the early stages of use of object-technologies, a software organization has few if any OO applications. Therefore, we identified the conceptual (as opposed to physical) objects in each [non-OO] application.

2.1.2.6 Development of Analysis Model for the Objects

The analysis model served as the basis for design and construction of the domain objects.

2.2 The Object-Oriented Analysis Process

The OOA process doesn't begin with a concern for objects. Rather, it begins with an understanding of the manner in which the system will be used—by people, if the system is human-interactive; by machines, if the system is involved in process control; or by other programs, if the system coordinates and controls applications. Once the scenario of usage has been defined, the modeling of the software begins.

A series of techniques is used to gather basic customer requirements and then define an analysis model for an object-oriented system.

2.2.1 Use-Cases

Use-cases model the system from the end-user's point of view. Created during requirements elicitation, use-cases should achieve the following objectives:

- To define the functional and operational requirements of the system (product) by defining a scenario of usage that is agreed upon by the end-user and the software engineering team.
- To provide a clear and unambiguous description of how the end-user and the system interact with one another.
- To provide a basis for validation testing.

2.2.1.1 Use-Cases Diagram

Use-Case diagram in Figure 2-11 shows some of the use-cases in the system, some of the actors in the system, and the relationships between them.

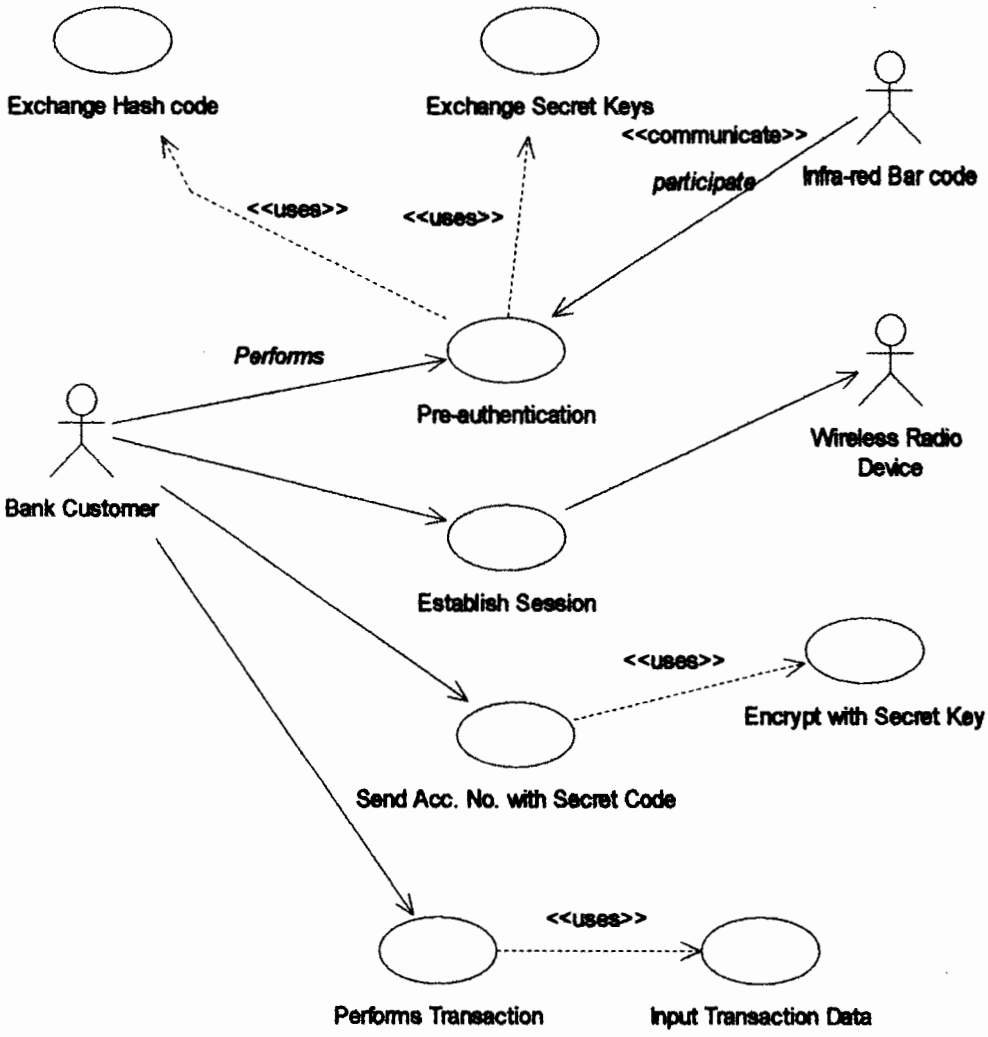


Figure 2.11 Use-Case Diagram for Secure Banking: Authenticating peer-to-peer Ad hoc networks

2.2.1.2 Use-Cases In the System

A use-case is a high level piece of functionality that the system provides.

- *Pre-authentication*
- *Exchange Secret Keys*
- *Exchange Hash code*
- *Establish Session*
- *Send Acc. No. with Secret code*
- *Encrypt with Secret Key*
- *Performs Transaction*
- *Input Transaction Data*

2.2.1.3 Expanded Use-Cases Format

- **Use-Case: Pre-authentication**

  Actors: Bank Customer, Infra-red Bar code

  Subject: Exchanges pre-authentication data.

  Summary: The Bank customer exchanges secret keys with infra-red bar code and get a secret key for further transactions.

  Type: Essential, Primary

  Typical Course of Actions:

  1. When a Bank customer goes near to the bar-code and sends its secret code to the bar-code.
  2. The infra-red bar-code assigns a secret key to the bank customer.
  3. It performs key exchange.
  4. Sends user secret key
  5. Gets a secret key from infra-red bar-code.
  6. Stores secret key in bank customer's laptop or PDA.
  7. Completes pre-authentication step.

Alternative Courses of Actions:

3a. Secret key exchange failed, exit the thread.

- **Use-Case: Exchange Hash Code**

Actors: None

Subject: Exchanges hash code.

Summary: The Pre-authentication use-case uses the Exchange Hash-code use-case to perform exchange of hash code of secret keys.

Type: Essential, Primary

Typical Course of Actions:

1. This Use-Case begins when the Pre-authentication use-case uses this use-case.
2. It responds by exchanging hash code of secret keys generated by Hash function.

Alternative Courses of Actions:

2a. Hash code function failed, Exit the function.

- **Use-Case: Exchange Secret keys**

Actors: None

Subject: Exchanges secret keys.

Summary: The Pre-authentication use-case uses the Exchange Secret keys use-case to exchange the secret keys between bank customer and Infra-red bar-code.

Type: Essential, Primary

Typical Course of Actions:

1. This Use-Case begins when the Pre-authentication use-case uses this use-case.
2. It responds by exchanging the secret keys between the two actors.

Alternative Courses of Actions:

2a.The secret key exchange failed, Exit the function.

- **Use-Case: Establish Session**

  Actors: Bank Customer, Wireless Radio Device

  Subject: Establishes connection.

  Summary: This use-case is called when the Bank customer establishes connection with the bank server via Wireless radio device.

  Type: Essential, Primary

  Typical Course of Actions:

  1. This Use-Case begins when Bank customer runs the connection function.
  2. It responds by establishing connection with the wireless device.
  3. It initializes Active-x component.
  4. Establishes a secure communication channel using SSL protocol over TCP-IP.

  Alternative Courses of Actions:

  4a. Failed to establish a connection, exit the application.

- **Use-Case: Send Acc. No with Secret code**

  Actors: Bank Customer

  Subject: Bank customer sends its information.

  Summary: This use-case starts after establish session use-case in which bank customer sends its account number with secret code to bank server to authenticate itself.

  Type: Essential, Primary

  Typical Course of Actions:

  1. This Use-Case begins when the bank customer wants to authenticate itself using secure connection.
  2. It sends bank customer's account no. along with the secret key to the bank server.

  Alternative Courses of Actions:

  2a. Failed to deliver the data, exit the service.

- **Use-Case: Encrypt with Secret Key**

  Actors: None

  Subject: Encrypt Acc. No. with secret key.

  Summary: Send Acc. No. with Secret key use-case uses this use-case to encrypt the data before sending it to the bank server.

  Type: Essential, Primary

  Typical Course of Actions:

  1. This Use-Case begins when the Send Acc. No. with Secret key use-case uses this use-case.
  2. It responds by encrypting the user information with the secret key exchanged during pre-authentication.

  Alternative Courses of Actions:

  2a. Encryption function failed, exit.

- **Use-Case: Perform Transaction**

  Actors: Bank Customer

  Subject: Performs transaction.

  Summary: This use-case begins to perform a transaction with the bank server after connection establishment and authentication process.

  Type: Essential, Primary

  Typical Course of Actions:

  1 This Use-Case begins when the Bank customer performs transactions with the bank server using secure channel.

  2 It responds by performing the transaction successfully and executing the command on server for changes.

  Alternative Courses of Actions:

  2a. Transaction failed, again perform the transaction.

- **Use-Case: Input Transaction Data**

  Actors: None

  Subject: Takes input from bank customer.

  Summary: This use-case begins when Perform transaction use-case uses it to take input from bank customer before performing the transaction.

  Type: Essential, Primary

  Typical Course of Actions:

  1 This Use-Case begins when the Perform Transaction use-case uses this use-case.

  2 It responds by taking input from the bank customer.

Chapter 3
Design

3. DESIGN

Design is an iterative process transforming requirements into a "blueprint" for constructing the software. It is the first step in the development phase for any engineered product or system. It can also be defined as "the process of applying various techniques and principles for the purpose of defining a device, a process or a a system in sufficient detail to permit its physical realization.

The designer's goal is to produce a model or representation of an entity that will later be built. The process by which the model is developed combines intuition and judgment based on experience in building similar entities, a set of principles and/or heuristics that guide the way in which the model evolves, a ultimately leads to a final design representation.

3.1 Object-Oriented Design

It includes a process of object-oriented decomposition and a notation for representing logical and physical as well as static and dynamic models of the system under design.

The four layers of object-oriented design pyramid are:

- **The Subsystem Layer** contains a representation of each of the subsystems that enable the software to achieve its customer-defined requirements and to implement the technical infrastructure that supports customer requirements. The subsystem design is derived by considering overall customer requirements (represented with use-cases) and the events and states that are externally observable (the object-behavior model).

- **The Class and Object Layer** contains the class hierarchies that enable the system to be created using generalizations and increasingly more targeted specializations. This layer also contains representations of each object. Class and object design is mapped from the description of attributes, operations, and collaborations contained in the CRC model.

- **The Message Layer** contains the design details that enable each object to communicate with its collaborators. This layer establishes the external and internal interfaces for the system. Message design is driven by the object-relationship model.

- **The Responsibilities Layer** contains the data structure and algorithmic design for all attributes and operations for each object. Responsibilities design is derived using the attributes, operations, and collaborations described in the CRC model.

The design pyramid focuses exclusively on the design of a specific product or system. It should be noted, however that another "layer" of design exists, and this layer forms the foundation on which the pyramid rests. The foundation layer focuses on the design of *domain objects* (called *design patterns*). Domain objects play a key role in building the infrastructure for the OO system by providing support for human/computer interface activities, task management, and data management. Domain objects can also be used to flesh out the design of the application itself.

3.2 Design Patterns

The best engineers in any field have an uncanny ability to see patterns that characterize a problem and corresponding patterns that can be combined to create a solution. Throughout the OOD process, a software engineer should look for every opportunity to reuse existing design patterns (when they meet the needs of the design) rather than creating new ones.

3.2.1 Describing Design Patterns

All design patterns can be described by specifying the following information:

- The name of the pattern
- The intent of the pattern
- The "design forces" that motivate the pattern
- The solution that mitigates these forces
- The classes that are required to implement the solution
- The responsibilities and collaboration among solution classes
- Guidance that leads to effective implementation
- Example source code or source code templates
- Cross-references to related design patterns

The design pattern name is itself an abstraction that conveys significant meaning once the applicability and intent are understood. *Design forces* describe the data, functional, or behavioral requirements associated with part of the software for which the pattern is to be applied. In addition forces define the constraints that may restrict the manner in which the design is to be derived. In essence, design forces describe the environment and conditions that must exist to make the design pattern applicable. The pattern characteristics (classes, responsibilities, and collaborations) indicate the attributes of the design that may be adjusted to enable the pattern to accommodate a variety of problems. These attributes represent characteristics of the design that can be searched (e.g. via a database) so that an appropriate pattern can be found. Finally, guidance associated with the use of a design pattern provides an indication of the ramifications of design decisions.

3.2.2 Using Patterns in Design

In an object-oriented system, design patterns can be used by applying two different mechanisms: inheritance and composition. Using *inheritance*, an existing design pattern becomes a template for new subclass. The attributes and operations that exist in the pattern become part of the subclass.

Composition is a concept that leads to aggregate objects. That is, a problem may require objects that have complex functionality (in the extreme, a subsystem accomplishes this). The complex object can be assembled by selecting a set of design patterns and composing the appropriate object (or subsystem). Each design pattern is treated as a black box, and communicates among the patterns occurs only via well defined interfaces.

3.3 Object-Oriented Design Process

UML design modeling addresses the structural model, behavioral model, implementation model, and environmental model views.

3.3.1 Structural Model

Data and functionality are viewed from inside the system. That is, static structure (classes, objects, and relationships) is modeled.

3.3.1.1 What is a Class?

A *class* is something that encapsulates information and behavior. Traditionally we've approached systems with the idea that we have the information over here on the database side, and the behavior over there on the application side. One of the differences with the object-oriented approach is the joining of a little bit of information with the behavior that affects the information. We take a little bit of information and a little bit of behavior, and encapsulate them into something called a class.

3.3.1.2 Finding a Class

A good place to start when finding classes is the flow of events for the use-cases. Looking at the nouns in the flow of events will allow us to know what some of the classes are. When looking at the nouns, they will be one of four things:

- An actor
- A class
- An attribute of a class
- An expression that is not an actor , class, or attribute

By filtering out all of the nouns except for the classes, we have found classes identified for our system.

3.3.1.3 Class Diagram

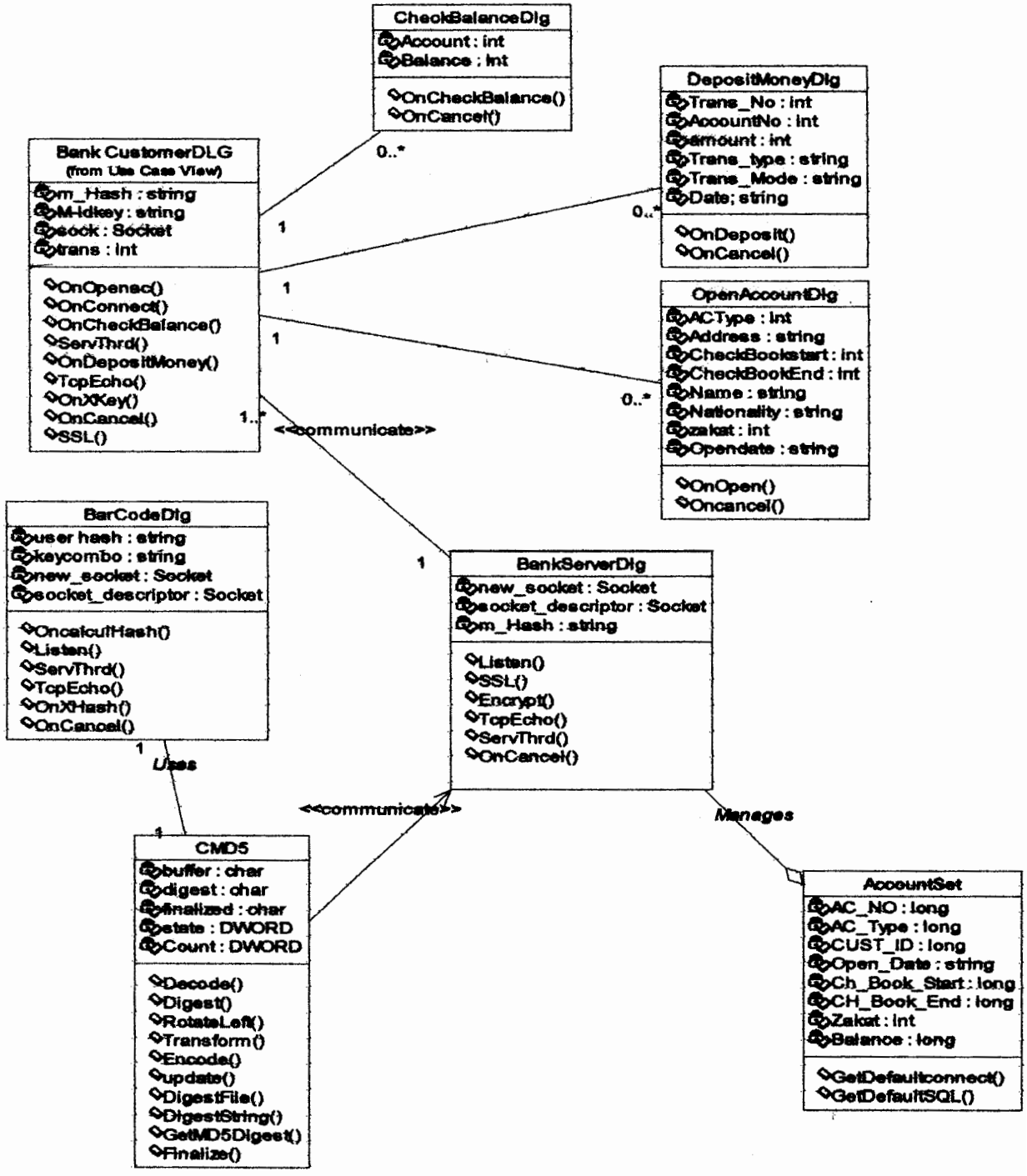


***Figure 3.6** Class Diagram for Secure Banking: Authenticating peer-to-peer Ad hoc networks*

A *Class diagram* in Figure 3-6 is used to display some of the classes and packages of classes in the system. It gives a static picture of the pieces in the system, and of the relationships between them.

3.3.2 Behavioral Model

This part of the analysis model represents the dynamic or behavioral aspects of the system. It also depicts the interactions or collaborations between various structural elements described in the user model and structural model views.

3.3.2.1 Interaction Diagrams

An *Interaction diagram* shows, step-by-step, one of the flows through a use-case. There are two types of Interaction diagrams:

- *Sequence Diagram* represents dynamic behavior which time oriented. It can show the focus of control.
- *Collaboration Diagram* represents dynamic behavior which message oriented. It can show the data flow.

3.3.2.2 Sequence Diagram

A Sequence diagram shown in Figure 3-7 is an interaction diagram, which is ordered by time; it is read from the top to the bottom.

We can read this diagram by looking at the objects and messages. The objects that participate in the flow are shown in rectangles across the top of the diagram.

The actor objects, involved in the use-case are also shown in the diagram.

Each object has a *lifeline*, drawn as a vertical dashed line below the object. A message is drawn between the lifelines of two objects to show that the objects communicate. Each message represents one object making a function call of another.

Messages can also be reflexive, showing that an object is calling one of its own operations.

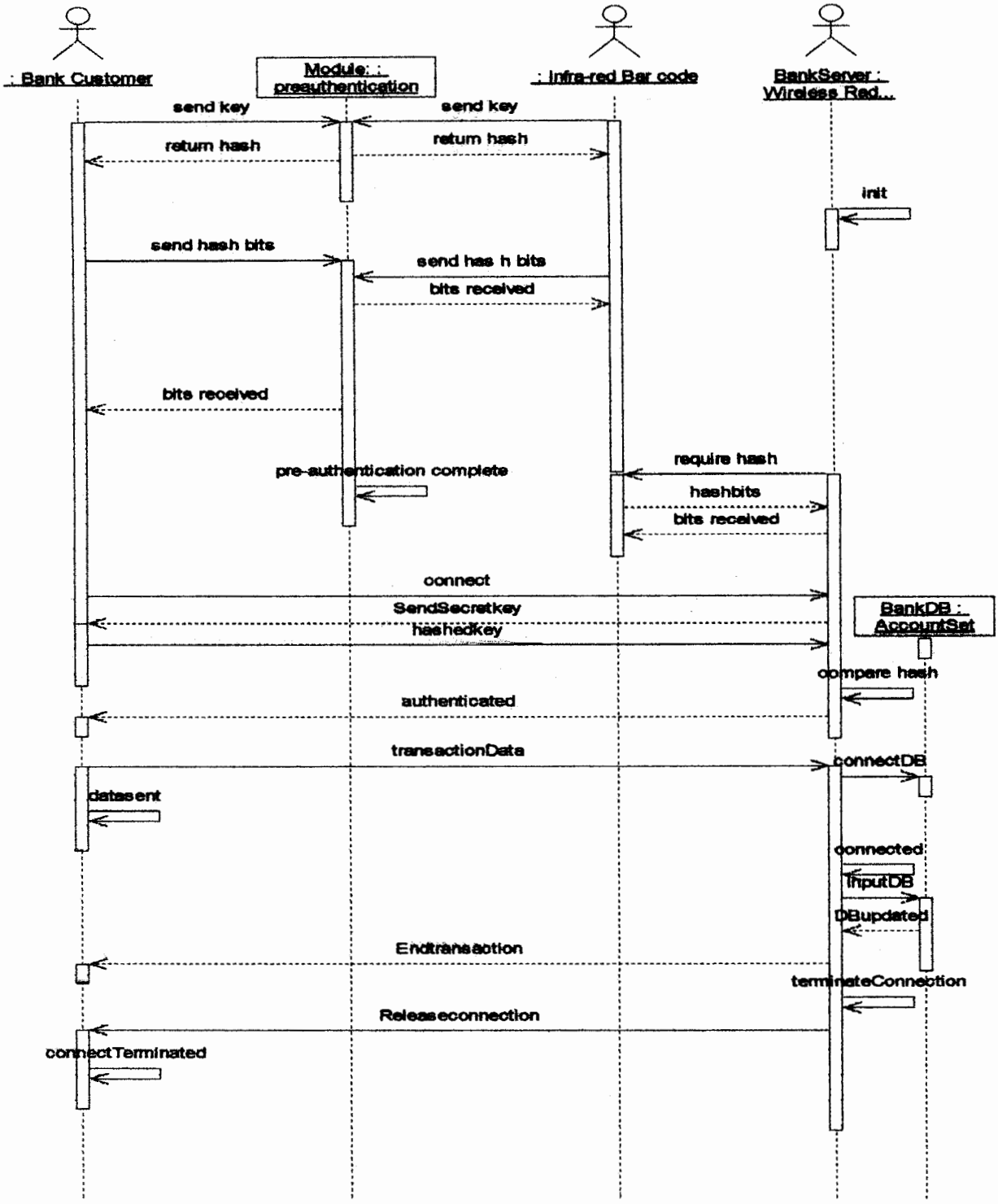


Figure 3.7 Sequence Diagram Secure Banking: Authenticating peer-to-peer ad hoc networks

3.3.2.3 State Transition Diagram

State Transition diagrams in Figures 3-8, 3-9 and 3-10 shows the life cycle of a single object, from the time it is created until it is destroyed.

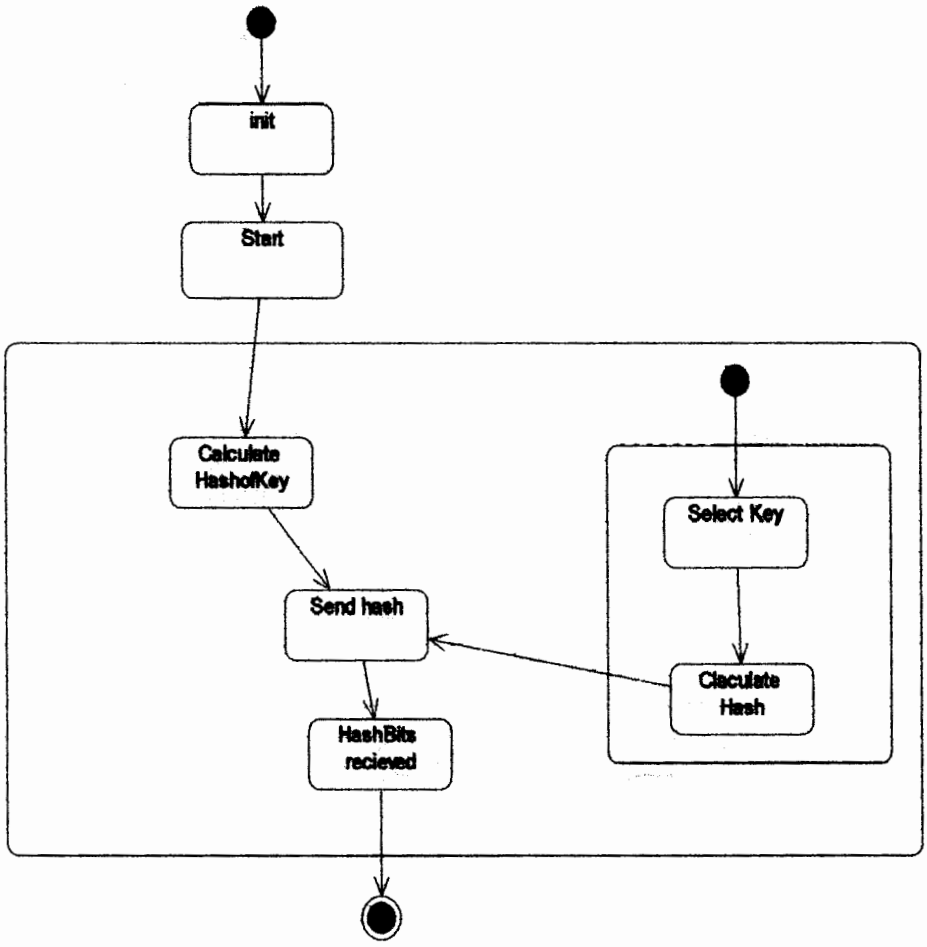


Figure 3.8 State Transition Diagram for Use-Case Pre-authentication

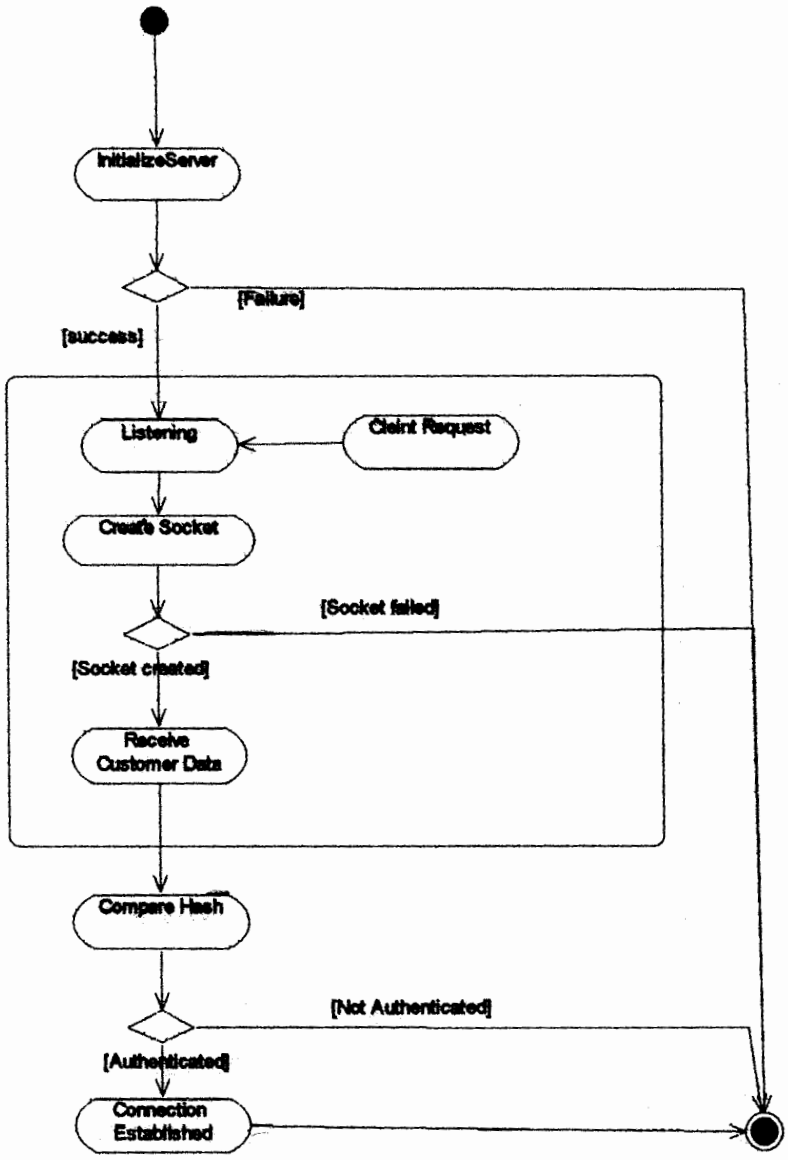


Figure 3.9 State Transition Diagram for Use-Case Establish Session

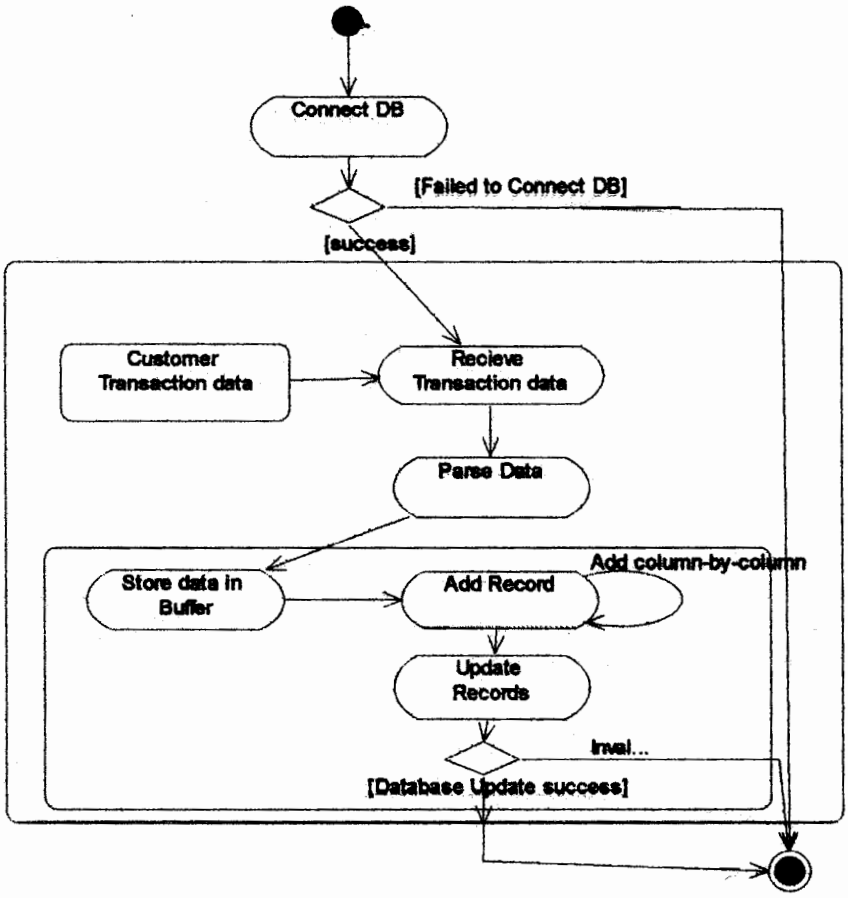


Figure 3.10 State Transition Diagram for Use-Case Perform Transaction

Chapter 4
Implementation

4. IMPLEMENTATION

This is very important phase in the software engineering paradigm because no matter how efficiently analysis has been done? Or how brilliantly the design has been prepared? Although programming is an outgrowth of analysis and design, all the programming and implementation skills have to be applied here, because any inefficiency on part of the programmer will hammer the quality of the software. Another important aspect of this phase is that, although this phase is succeeded by the testing phase, but during the implementation phase the programmer is best equipped for glass box testing of the software, because at this stage he has the access to the code.

4.1 Implementation Techniques

The following techniques were used during the implementation of our project.

4.1.1 Object-Oriented Programming

Although all areas of object technologies have received significant attention within the software community, no subject has produced more books, more discussion, and more debate than *object-oriented programming* (OOP). Hundreds of books have been wirtten on C++ and Java programming, and hundreds more are dedicated to less widely used OO languages.

The software engineering viewpoint stresses OOA and OOD and considers OOP (coding) an important, but secondary, activity that is an outgrowth of analysis and design. The reason for this is simple. As the complexity of systems increases, the design architecture of the end product has a significantly stronger influence on its success than the programming language that has been used. And yet, "language wars" continue to rage.

4.1.2 Component-Based Programming

In the software engineering context, reuse is an idea both old and new programmers have stressed upon, since the earliest days of computing, but the early approach to reuse was ad hoc. Today, complex, high-quality computer-based systems must be built in very short time periods. This mitigates toward a more organized approach to reuse.

Computer-based software engineering (CBSE) is a process that emphasizes the design and construction of computer-based systems using reusable software "components."

4.1.2.1 Microsoft COM

Microsoft has developed a component object model (COM) that provides a specification for using components produced by various vendors within a single application running under the Windows based operating system. COM encompasses two elements: COM interfaces (implemented as COM objects) and a set of mechanisms for

registering and passign messages between COM interfaces. From the point of view of the application, "the focus is not on how [COM objects are] implemented, only on the fact that the object has an interface that it registers with the system, and that uses the component system to communicate with other COM objects".

4.2 Implementation Tools

Our simultaion software is developed using two tools Microsoft Visual C++ and Microsoft Access 2002 (for database). There are some reasons to select Visual C++ and Microsoft Access. The basic reason is that the DDK provides a platform for developing device driver for Windows based operating system, but using Visual C++ with DDK is a necessity.

4.2.1 Microsoft Visual C++

Microsoft Visual C++.NET 2003 provides the dynamic development environment for creating Microsoft Windows–based and Microsoft .NET–based applications, dynamic Web applications, and XML Web services using the C++ development language. Visual C++ .NET includes the industry-standard Active Template Library (ATL) and Microsoft Foundation Class (MFC) libraries, advanced language extensions, and powerful integrated development environment (IDE) features that enable developers to edit and debug source code efficiently.

It provides developers with a proven, object-oriented language for building powerful and performance-conscious applications. With advanced template features, low-level platform access, and an optimizing compiler, Visual C++.NET delivers superior functionality for generating robust applications and components. The product enables developers to build a wide variety of solutions, including Web applications, smart-client Microsoft Windows-based applications, and solutions for thin-client and smart-client mobile devices. C++ is the world's most popular systems-level language, and Visual C++.NET 2003 gives developers a world-class tool with which to build software.

Following are the reason to select Visual C++ for our Interface Development.

- Visual C++ is a powerful tool for building 32-bit applications for Window 95, and Windows NT.
- Provide graphical user interface.
- With its code generating Wizards it can produce the shell of a working Windows application in seconds.
- The class library included with Visual C++, the Microsoft Foundation Classes, has become the industry standard for Windows software development.
- The visual editing tools make layout of menus and dialogs a snap.
- You can create a Windows application in minutes by telling AppWizard to make you a "starter app" with all the Windows boilerplate code you want.

- Many of the methods correspond closely to API (Application Programming Interface) functions that are already familiar to Windows programmers.
- Very Powerful compilers that not only tell the error but also try to give u message so that u can easily understand what wrong u have done.
- Easy to use linking facility so that you can link your program with other programs who has provide u with Library to interact or call their program function in your programs.
- It can be efficiently used for object-oriented programming.
- It supports COM.

4.3 SSL (Secure Socket layer Protocol)

The primary goal of the SSL Protocol is to provide privacy and reliability between two communicating applications. The protocol is composed of two layers. At the lowest level, layered on top of some reliable transport protocol (e.g., TCP[TCP]), is the *SSL Record Protocol*. The SSL Record Protocol is used for encapsulation of various higher level protocols. One such encapsulated protocol, the *SSL Handshake Protocol*, allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data.

4.3.1 ECC-based SSL Handshake

Since ECC is a public-key cryptographic mechanism, only the handshake protocol is affected by incorporating ECC into SSL. Figure 4.1 shows the operation of an ECC-based SSL handshake. Through the firrst two messages (processed in the same way as for RSA) the client and server negotiate an ECC based cipher suite (for example, ECDH-ECDSA-RC4-SHA). However, the ServerCertificate message contains the server's Elliptic Curve Diffie-Hellman (ECDH) public key signed by a certificate authority using the Elliptic Curve Digital Signature Algorithm (ECDSA). After validating the ECDSA signature, the client conveys its ECDH public-key to the server in the ClientKeyExchange. The SSL Handshake Protocol is one of the defined higher level clients of the SSL Record Protocol. This protocol is used to negotiate the secure attributes of a session. Handshake messages are supplied to the SSL Record Layer, where they are encapsulated within one or more SSLPlaintext structures, which are processed and transmitted as specified by the current active session state.

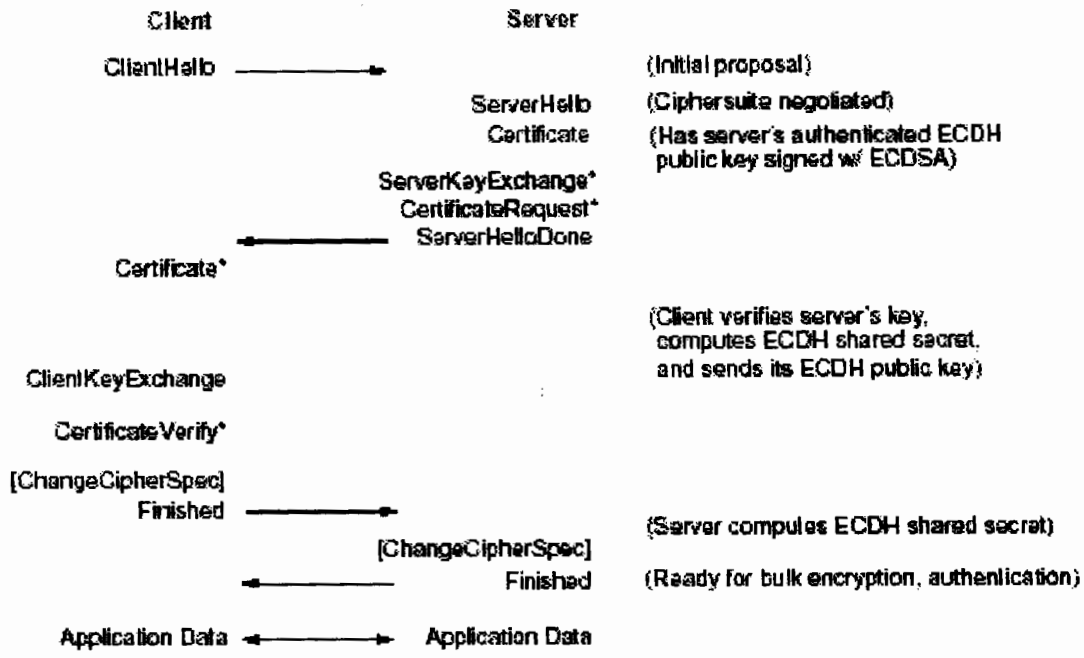


Figure 4.1 ECC-based SSL-Handshake

4.3.2 SSL Record Layer Protocol

The SSL Record Layer receives uninterpreted data from higher layers in non-empty blocks of arbitrary size.

- ***Fragmentation***

  The record layer fragments information blocks into SSLPlaintext records of 2^14 bytes or less. Client message boundaries are not preserved in the record layer (i.e., multiple client messages of the same Content Type may be coalesced into a single SSLPlaintext record).

```
struct {
    uint8 major, minor;
} ProtocolVersion;
enum {
    change_cipher_spec(20), alert(21), handshake(22),
    application_data(23), (255)
} ContentType;
struct {
    ContentType type;
    ProtocolVersion version;
    uint16 length;
    opaque fragment[SSLPlaintext.length]; } SSLPlaintext;
```

- ***Record compression and decompression***

All records are compressed using the compression algorithm defined in the current session state. There is always an active compression algorithm, however initially it is defined as CompressionMethod.null. The compression algorithm translates an SSLPlaintext structure into an SSLCompressed structure. Compression functions erase their state information whenever the CipherSpec is replaced.

```
struct {
    ContentType type;       /* same as SSLPlaintext.type */
    ProtocolVersion version;/* same as SSLPlaintext.version */
    uint16 length;
    opaque fragment[SSLCompressed.length];
} SSLCompressed;
```

4.4 EC Domain Parameters

The elliptic curve domain parameters specify the elliptic curve used. The parameters consist:

Bit Size	Curve Name
163	NIST_B_163
233	NIST_B_233
283	NIST_B_283
409	NIST_B_409
571	NIST_B_571

***Table 4.1** NIST Recommended Curves*

4.4.1 The Finite Field ($GF2^m$)

borZoi uses a binary field of the form 2^m over a polynomial basis. This is specified by m, an unsigned long, basis, an int, which can be 1 (Gaussian Basis: not supported in borZoi), 2 (Trinomial Basis: $x^m + x^k + 1$) or 3 (Pentanomial Basis: $x^m + x^{k3} + x^{k2} + x^{k1} + 1$), trinomial_k, an unsigned long, representing the k power of the trinomial polynomial, pentanomial_k3, an unsigned long, representing the k3 power of the pentanomial polynomial, pentanomial_k2, an unsigned long, representing the k2 power of the pentanomial polynomial and pentanomial_k1, an unsigned long, representing the k_1 power of the pentanomial polynomial.

4.4.2 The Elliptic Curve (E : $y^2 + xy = x^3 + ax^2 + b$)

The elliptic curve is specified by a, a finite field element (F2M), b, a finite field element (F2M), r, a large positive prime integer (BigInt) which divides the number of points on the curve, G, a point on the curve (Point, see borzoi_util.h) which is the generator of a subgroup (of the points on the curve) of order r and k, a positive prime integer (BigInt) called the cofactor which is equal to the number of points on the curve divided by r.

4.4.3 Constructors

- EC_Domain_Parameters ( );
- EC_Domain_Parameters (m, basis, trinomial_k, c, r, G, k);
- EC_Domain_Parameters (m, basis, pentanomial_k3, pentanomial_k2, pentanomial_k1). Note - c is of type Curve.

4.4.4 Methods

- bool valid ( );

This checks if the EC domain parameters are valid using a subset (steps 6.4 to 7) of the method in section A.16.8 of the IEEE P1363 standard:

– Check that $b \neq 0$ in $GF2^m$.

– Check that $G \neq O$.

– Check that G_X and G_Y are elements of $GF2^m$.

– Check that $y^2 + xy = x^3 + ax^2 + b$ in $GF2^m$.

– Check that $rG = O$.

– Check that the curve is not subject to the MOV reduction attack.

- EC_Domain_Parameters& operator = (const EC_Domain_Parameters& dp);

  Assignment

- std::ostream& put (std : : ostream& s) const;

  Output

4.5 EC Keys

4.5.1 EC Private Keys

Elliptic Curve Private Keys have two member variables: dp, the EC domain parameters and s, the private key which is a large integer (BigInt) and must be kept secret.

4.5.1.1 Constructors

- ECPrivKey (dp);

Generate an EC private key object with EC domain parameters dp and a random private key.

- ECPrivKey (dp, s);

Generate an EC private key object with EC domain parameters dp and a private key s.

4.5.2 EC Public Keys

Elliptic Curve Public keys have two member variables: dp, the EC domain parameters and W, the public key which is a point on the curve (Point).

4.5.2.1 Constructors

- ECPubKey ( );

  Create an empty EC public key object.

- ECPubKey (sk);

  Calculate an EC public key object from an EC private key sk.

- ECPubKey (dp, W);

  Create an EC public key object with EC domain parameters dp and public key W.

4.5.2.2 Methods

- bool valid ( );

  Checks if the EC public key object is valid

- =

  Assignment

4.6 ECKAS-DH1

In ECKAS-DH1 (the Elliptic Curve Key Agreement Scheme, Diffie-Hellman 1), each party combines its own private key with the other party's public key to calculate a shared secret key which can then be used as the key for a symmetric encryption algorithm such as AES. Other (public or private) information known to both parties may be used as key derivation parameters to ensure that a di_erent secret key is generated every session.

4.6.1 Functions

- OCTETSTR ECKAS_DH1 (dp, s, Wi);

  Calculates a 128 bit secret key from EC domain parameters dp, private key s and public key Wi. s belongs to one party, Wi belongs to the other and dp is common to both of them.

- OCTETSTR ECKAS_DH1 (dp, s, Wi, P);

  Calculates a 128 bit secret key from EC domain parameters dp, private key s, public key Wi and key derivation parameter P (an octet string). s belongs to one party, Wi belongs to the other and dp and P are common to both of them.

- Example

  In this example (Figure 4.1) party A and party B have previously decided to use the NIST_B_163 curve.

User A	**User B**
use_NIST_B_163 ( ) ;	use_NIST_B_163 ( ) ;
EC_Domain_Parameters dp = NIST_B_163;	EC_Domain_Parameters dp = NIST_B_163;
ECPrivKey skA (dp); // generate private key	ECPrivKey skB (dp); // generate private key
ECPubKey pkA (skA); // calculate public key	ECPubKey pkB (skB); // calculate public key
// Send pkA to B and obtain pkB	// Send pkB to A and obtain pkA
OCTETSTR K = ECKAS_DH1 (dp, skA.s, pkB.W)	OCTETSTR K = ECKAS_DH1 (dp, skB.s, pkA.W)
// Use K with a symmetric encryption algorithm	// Use K with a symmetric encryption algorithm

Figure 4.1 ECKAS_DH1 Example

4.7 SHA-1 Hash Algorithm

These functions implement the SHA-1 hash algorithm as specified in FIPS 180-1.

4.7.1 Functions

- OCTETSTR SHA1 (x);

  Calculate the SHA-1 hash of octet string x.

- OCTETSTR SHA1 (x);

  Calculate the SHA-1 hash of string x.

- OCTETSTR SHA1 (x);

  Calculate the SHA-1 hash of BigInt x.

4.7.2 Example

```
std :: string M ( " Message to be hashed " );

OCTETSTR hash = SHA1 ( M );
```

4.8 Elliptic Curve Digital Signature Algorithm (ECDSA)

ECDSA (the Elliptic Curve Digital Signature Algorithm) is used to generate a digital signature of a message digest or hash. The signature consists of c and d which are two large integers (BigInt).

4.8.1 Constructors

- ECDSA ( )

  Create an empty ECDSA signature object.

- ECDSA (c , d)

  Create an ECDSA signature object from c (BigInt) and d (BigInt).

- ECDSA (sk , f)

Generate an ECDSA signature object from message digest f (BigInt) and private key sk (ECPrivKey).

4.8.2 Methods

- Verify (pk , f)

Verify the signature with message digest f (BigInt) and public key pk (ECPubKey).

- Put ( ) : Output

4.8.3 Example (generating a signature)

In this example, the user has previously generated a private key sk:

```
Std :: string M ("message");
ECDSA sig (sk, OS2IP ( SHA1 (M)));
```

4.8.4 Example (verifying a signature)

In this example, the user tries to verify the signature generated in the previous example:

```
if ( sig.verify ( pk , OS2IP (SHA1 (M))))
{ // Valid Signature }
else
{ // Invalid Signature }
```

4.9 Database Connectivity and Transactions

We accessed the bank database using ODBC. The Bank database is linked with the Bank server. The implementation of accessing the database and performing transactions is as under:

4.9.1 CAccountSet

```
IMPLEMENT_DYNAMIC(CAccountSet, CRecordset)
CAccountSet::CAccountSet(CDatabase* pdb)
    : CRecordset(pdb)
{   //{{AFX_FIELD_INIT(CAccountSet)
    m_AC_NO = 0;
    m_AC_TYPE = 0;
    m_CUST_ID = 0;
    m_OPEN_DATE = _T("");
    m_CHECKBOOK_START = 0;
    m_CHECKBOOK_END = 0;
    m_ZAKAT = 0;
    m_BALANCE = 0;
    m_nFields = 8;
    //}}AFX_FIELD_INIT
    m_nDefaultType = dynaset;
}
```

Chapter 5

Testing

5. TESTING AND RESULTS

The overall objective of the testing process is to identify the maximum number of errors in the code with a minimum amount of efforts. Finding an error is thus considered a success rather than failure. On finding an error, efforts are made to correct it.

5.1 Testing Process

Test consists of a number of test cases, where different aspects of the part of the project under test are checked. Each test case tells what to do, what data to use, and what results to expect. When conducting the test, the results including deviations from the planned test cases are not in a test protocol. Normally a deviation indicates an error in the system (although some times the test case is wrong, and the system is right). An error is noted and described in a test report for removal or directly removed by the programmer who developed that part.

5.2 General Types of Errors

Error can be of following types:

- Functional error (e.g. function is not working correctly or missing).
- Non-Functional error (e.g. performance is slow)
- Logical error (e.g. error in algorithm, user interface errors is not considered as a logical error).

5.3 Testing Strategies

A strategy for software testing may be viewed as the spiral (Figure 5-1). Unit testing begins at the vortex of the spiral and concentrates on each unit (i.e., component) of the software as implemented in source code. Testing progresses by moving outward along the spiral to integration testing, where the focus is on design and the construction of the software architecture. Taking another turn outward on the spiral, we encounter validation testing, where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally, we arrive at system testing, where the software and other system elements are tested as a whole. To test computer software, we spiral out along stream-lines that broaden the scope of testing with each turn.

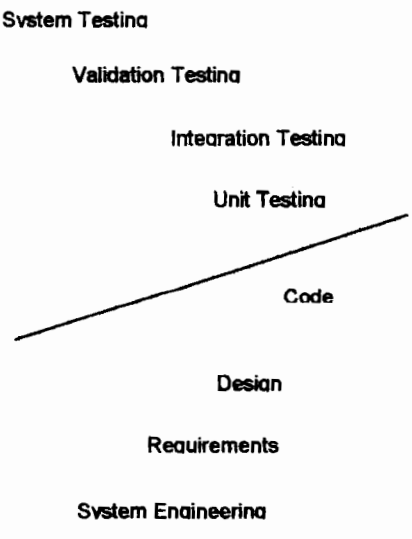


Figure 5-1: Testing Strategy

5.3.1 Unit Testing

Unit testing focuses verification effort on the smallest unit of software design—the software component of module. Using the component-level design description as guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests is uncovered errors in limited by the constrained scope established for unit testing. The unit test is white-box oriented, and the step can be conducted in parallel for multiple components.

5.3.2 Integration Testing

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested components and build a program structure that has been dictated by design.

5.3.2.1 Top-down Integration

Top-down integration testing is an incremental approach to construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main control module (main program). Modules subordinates (and ultimately subordinate) to the main control module control module are incorporated into the structure in either a depth-first or breath-first manner.

5.3.2.2 Bottom-up Integration

Bottom-up integration testing, as the name implies, begins construction and testing with atomic modules (i.e., components at the lowest levels in the program structure). Because components are integrated from the bottom up, processing required for components subordinate to a given level is always available and the need for stubs is eliminated.

5.3.2.3 Regression Testing

Each time a new module is added as part of integration testing, the software changes. New data flow paths are established, new I/O may occur, and new control logic is invoked. These changes may cause problems with functions that previously worked flawlessly. In the context of an integration test strategy, regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.

5.3.3 Validation Testing

Validation can be defined in many ways but a simple definition is that validation succeeds when software functions in a manner that can be reasonably expected by the customer.

5.3.4 System Testing

System testing is a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that system elements have been properly integrated and perform allocated functions.

5.3.4.1 Security Testing

Security testing attempts to verify that protection mechanisms built into a system will, in fact, protect it from improper penetration.

5.3.4.2 Stress Testing

Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency or volume.

5.3.4.3 Performance Testing

Performance testing is designed to test the run-time performance of software within the context of an integrated system. Performance testing occurs throughout all steps in the testing process. Even at the unit level, the performance of an individual module may be assessed as white-box tests are conducted. However, it is not until all system elements are fully integrated that the true performance of a system can be ascertained.

5.4 Object-Oriented Testing Strategies

The classical strategy for testing computer software begins with "testing in the small" and works outward "testing in the large." We begin with unit testing, then progress towards integration testing, and culminate with validation and system testing. In conventional applications, unit testing focuses on the smallest compile able program unit—the subprogram (e.g., modules, subroutine, procedure, component). Once each of these units has been tested individually, it is integrated into a program structure while a series of regression tests are run to uncover errors due to interfacing between the modules and side effects caused by the addition of new units. Finally, the system as a whole is tested to ensure that errors in requirements are uncovered.

5.4.1 Unit Testing in the OO Context

Class testing for OO software is the equivalent of unit testing for conventional software. Unlike unit testing of conventional software, which tends to focus on the algorithmic detail of a module and the data that flow across the module interface, class testing for OO software is driven by the operations encapsulated by the class and the state behavior of the class.

5.4.2 Integration Testing in the OO Context

Because object-oriented software does not have a hierarchical control structure, conventional top-down and bottom-up integration strategies have little meaning. In addition, integrating operations one at a time into a class is often impossible because of the "direct and indirect interactions of the components that make up the class". There are two different strategies for integration testing of OO systems:

5.4.2.1 Thread-based Testing

Thread-based testing, integrates the set of classes required to respond to one input or event for the system. Each thread is integrated and tested individually.

5.4.2.2 Use-based Testing

Use-based testing, begins the construction of the system by testing those classes (called *independent classes*) that use very few (if any) of server classes. After the independent classes are tested, the next layers of classes, called *dependent classes* that use the independent classes are tested. This layer of testing layers of dependent classes continues until the entire system is constructed.

5.4.2.3 Cluster Testing

Cluster testing is one step in the integration testing of OO software. Here, a cluster of collaborating classes (determined by examining the CRC and object-relationship model) is exercised by designing test cases that attempt to uncover errors in the collaborations.

5.4.3 Validation Testing in the OO Context

At the validation or system level, the details of class connections disappear. Like conventional validation, the validation of OO software focuses on user-visible actions and user-recognizable output from the system.

5.5 Test Plan

Test plan provides an overview of the testing effort for the product.

5.5.1 Secure Banking Test Plan

- **Introduction**

  Our testing effort descriptions summarize *IEEE 829-1983 for Software Test Documentation*, which attempts to define a common set of test documents, to be used across the industry.

- **Test Items**

  1. Bar-code device
  2. Timing Results for Field and Elliptic curve operations
  3. SSL Record-Layer Protocol
  4. SSL Handshake Protocol
  5. Database Access and Update (Transaction success)
  6. Performance comparison of RSA, DSA and ECDSA

- **Features to be Tested**

  Our *Test Design Specification* (Section 5.6) will summarize the features to be tested.

- **Item Pass/Fail Criteria**

  Item will be considered as pass if the test lead to an expected result or the behavior of the module is according to expectations. If otherwise the item will be considered as failed.

- **Suspension Criteria and Resumption Requirements**

  If the test leads to a BSOD (Blue Screen of Death) or a severe error message from the operating system or a lethal software crash, further testing will be ceased. The test will be redone, incase the bug can't be reproduced.

- **Test Deliverables**

  None

- **Environmental Needs**

  1. A Pentium® III or higher machine.
  2. Windows 2000 (Family)
  3. Microsoft® Visual C++ 6.0

5.6 Test Design Specification

This specifies how a feature or group of features will be tested according to Standard 829.

5.6.1 Secure Banking Test Design Specifications

- **Features to be Tested**

  This specification includes:

  1. User Interface aspects of Bar-Code device and Bank Customer
  2. Performance issues of SSL Protocol
  3. Stability of SSL Record Layer Protocol
  4. Stability of SSL Handshake Protocol
  5. Validation of Client-Server Connection
  6. Performance issues of Encryption Algorithm
  7. Validation of database connectivity
  8. Stability of database
  9. Performance issues of database transactions

- **Test Identification**

  Our *Test Case Specifications* (Section 5.7) will define each test associated with this design.

- **Feature Pass/Fail Criteria**

  A feature or a combination of features will be considered as pass if the tests lead to an expected result, or the behavior of the module is according to expectations. If otherwise the feature will be considered as failed.

5.7 Test Case Specification

This defines a test case. According to the Standard 829, the test case specification includes the following sections:

5.7.1 Performance Test for SSL Session and Connection States

- **Test Items**

  1. SSL Record Layer Protocol

- **Input Specifications**

  When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL handshake protocol. Thereafter the final cipher text block from each record is preserved for use with the following record.

- **Output Specifications**

  Sequence numbers are of type uint64 and may not exceed 2^64-1.

- **Environmental Needs**

  1. A Pentium® III or higher machine.
  2. Windows 2000 (Family)

- **Special Procedural Requirements**

  Sequence numbers are of type uint64 and may not exceed 2^64-1.

- **Inter-case Dependencies**

  None

5.7.2 Load Test for SSL Session and Connection States

- **Test Items**

  1. Record Layer Protocol

- **Input Specifications**

  Client message boundaries are not preserved in the record layer (i.e., multiple client messages of the same ContentType may be coalesced into a single SSLPlaintext record).

- **Output Specifications**

  SSLPlaintext.fragment length should not exceed 2^14.

- **Environmental Needs**

  1. A Pentium® III or higher machine.
  2. Windows 2000 (Family)

- **Special Procedural Requirements**

  SSLPlaintext.fragment length should not exceed 2^14.

- **Inter-case Dependencies**

  None

5.7.3 Stress Test for SSL Session and Connection States

- **Test Items**

  1. SSL Record Layer Protocol

- **Input Specifications**

  Compression must be lossless and may not increase the content length by more than 1024 bytes. If the decompression function encounters an SSLCompressed.fragment that would decompress to a length in excess of 2^14 bytes, it should issue a fatal decompression_failure alert.

- **Output Specifications**

  The length (in bytes) of the SSLCompressed.fragment should not exceed 2^14 + 1024.

- **Environmental Needs**

  1. A Pentium® III or higher machine.
  2. Windows 2000 (Family)

- **Special Procedural Requirements**

  The length (in bytes) of the SSLCiphertext.fragment may not exceed 2^14 + 2048.

- **Inter-case Dependencies**

  1. Performance Test for SSL Record layer protocol.

5.7.4 Performance Test for SSL Handshake Protocol

- **Test Items**

  1. SSL handshake protocol

- **Input Specifications**

  Application data may not be sent before a finished message has been sent.

- **Output Specifications**

  Server must send a server hello message in response to client hello message, or else a fatal error will occur and the connection will fail.

- **Environmental Needs**

  1. A Pentium® III or higher machine.
  2. Windows 2000 (Family)

- **Special Procedural Requirements**

  None

- **Inter-case Dependencies**

  None

5.7.5 Load Test for SSL Handshake Protocol

- **Test Items**

  1. SSL Handshake protocol

- **Input Specifications**

  The server will send its certificate, if it is to be authenticated.

- **Output Specifications**

  Server Key Exchange message is not used if the server certificate contains Diffie-Hellman [DH] parameters.

- **Environmental Needs**

  1. A Pentium® III or higher machine.
  2. Windows 2000 (Family)

- **Special Procedural Requirements**

  If the server is authenticated, it may request a certificate from the client, if that is appropriate to the cipher suite selected.

5.7.6 Stress Test for SSL Handshake Protocol

- **Test Items**

  1. SSL Handshake protocol

- **Input Specifications**

  The client key exchange message is sent after Client's Certificate (if required), and the content of that message will depend on the public key algorithm selected between the client hello and the server hello.

- **Output Specifications**

  None

- **Environmental Needs**

  1. A Pentium® III or higher machine.

5.7.7 Performance Comparison of RSA, DSA and ECDSA

Before comparing the performance of different signature schemes, i.e. the execution times of the signature generation and signature verification, we have to agree which key sizes provide a comparable level of security. This is necessary, because the computational hardness of the underlying mathematical problems is different and some schemes need smaller key sizes than others for achieving the same level of security.

In a Standard for Efficient Cryptography document [44] a list of elliptic curves with different key sizes is given. The list also contains RSA / DSA key sizes that achieve comparable levels of security. Table 5.1 contains these values.

Curve Name	Curve Type	Key Size	equiv. RSA / DSA Key Size
sect113r1 sect113r2	random	113	512
sect131r1 sect131r2	random	131	704
sect163k1	Koblitz	163	1024
sect163r1 sect163r2	random	163	1024
sect193r1 sect193r2	random	193	1536
sect233k1	Koblitz	233	2240
sect233r1	random	233	2240
sect239k1	Koblitz	239	2304
sect283k1	Koblitz	283	3456
sect283r1	random	283	3456

***Table 5.1** SECG recommended elliptic curves over F_2^m [44]*

Clearly, the key sizes for elliptic curves are significantly smaller than those for RSA /DSA. Additionally, the key size does not increase as fast as the one of RSA / DSA. Figure 5.1 gives a good impression of this behavior. Hence, ECDSA has a major advantage for designs that might need an increased level of security in the future.

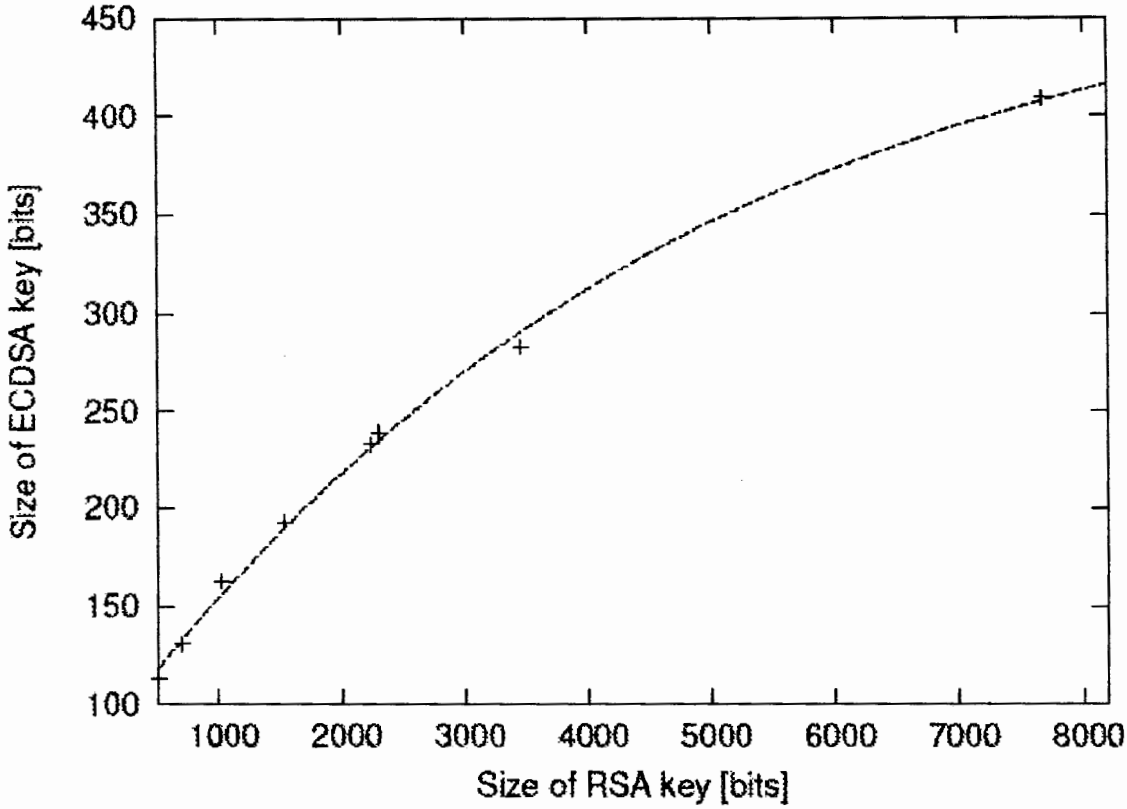


***Figure 5.1** Size of the ECDSA key compared to the size of the RSA / DSA key providing a similar level of security*

	RSA-1024 (e = 3)	DSA-1024	ECDSA-168 (OVER F_{168})
Sign	43	7	5
Verify	0.6	27	19
Key generation	1100	7	7

***Table 5.2** Digital signature timings (milliseconds on a 200 MHz Pentium Pro)*

Chapter 6

Conclusion

6. CONCLUSION

There is no any single authentication model which ensures authentication in every environment. While for a group meeting in small conference room the Password-based key exchange will work perfectly, for a network defined by hierarchy of trust relationships, the Resurrecting-Duckling policy is an ideal alternative. For guaranteed secure transaction certainly the choice is to use public key system involving the hassle of group certificates.

In this paper we have presented new approaches for peer-to-peer authentication in mobile ad hoc networks. We have constructed our schema on previous work by Anderson, Stajano and others, and presented to perform pre-authentication over location-limited channels. Our schema is public key infrastructure less and resolves naming problem that plagues traditional authentication systems.

Unique location-limited channels: Instead of limiting the concept of imprinting a duckling device with its mother's secret key, we propose to use location-limited channel to bootstrap a wide range of key-exchange protocols. This approach can be experimented to audio, infrared, and contact-based channels, but other media are certainly imaginable. Kernel-resident native file systems are those that interact directly with lower level media such as disks and networks. Writing such file systems is difficult because it requires deep understanding of specific operating system internals, especially the interaction with device drivers and the virtual memory system. Once such a file system is written, porting it to another operating system is just as difficult as the initial implementation, because specifics of different operating systems vary significantly.

Composed pre-authentication protocols: we provide a composed recipe for enhancing existing key exchange protocols with a pre-authentication step. And we explained how passive as well as active attacks can be detected by human user or by the system.

No dependency on Public key infrastructure: Key exchange and key agreement protocols depend on authentication step to verify the user. We have suggested a way to solve this problem. A reliance on pre-existing third party naming and trust infrastructures is unnecessary if one can briefly bring communicating parties within close physical proximity. In such a case, our pre-authentication protocols can be used in place of a PKI.

Appendix (Research Paper)

Securing Transactions in Mobile Ad Hoc Networks Using Location-Limited Channel

Dr. Khalid Rasheed (drkhalid@iiu.edu.pk) & Imran Hameed (hameedi@hotmail.com)
Faculty of Applied Sciences, International Islamic University
Islamabad, Pakistan

Abstract: Mobile ad hoc networks are unique generation of networks offering unrestricted mobility without relying on any infrastructure. In these kinds of networks, hosts rely on each other to keep the network connected, sharing responsibility of network formation and management. An important challenge in design of these networks is their vulnerability to security attacks. Security is a factor of great importance in decentralized communication systems like mobile ad hoc networks. In this paper we address the problem of secure communication and authentication in ad hoc wireless networks. This paper focuses on security mechanisms and notably on the key management mechanisms. Our authentication protocol is inspired from work of (D. Balfanz & H. Chi Wong, 2002) with a novel diversity. We present a reliable solution for securing wireless commercial transactions, which are practical and industry demanding requirements. In our approach, devices exchange a small amount of public information over location-constrained channel, which will later allow them to complete key exchange over wireless link. This approach does not require any public key infrastructure as is required in MANET. Moreover our approach is secure against passive attacks on location-limited channel and all kinds of wireless attacks on wireless link. Instead of using traditional public key algorithms like RSA, we have included Elliptic cryptography in our approach for generating public/private key pair because it is more suitable for constrained environments (low memory wireless devices). Since applications of Mobile ad hoc networks are very limited so we have also tried to find out applications of this area.

Key words: MANET, pre-authentication, location-constrained channel, elliptic curve cryptography, Resurrecting Duckling, SSL handshake, ECC-based handshake

Introduction

With the advancement in radio technologies like Bluetooth, IEEE 802.11 or Hiperlan, a new concept of networking has emerged. This is known as ad hoc networking where potential mobile users arrive within the common perimeter of radio link and participate in setting up the network topology for communication. Nodes within ad hoc network are mobile and they communicate with other nodes that are within radio range through direct wireless links and with those that are beyond the direct radio range through multi-hop routing.

The build up of ad hoc network can be envisaged where support of a wired backbone is not feasible. Ad hoc wireless network does not have any predefined infrastructure and all network services are configured and created on the fly. Thus it is obvious that with lack of infrastructural support and susceptible wireless link attacks, security in ad hoc network becomes inherent weakness. Achieving security within ad hoc networking is challenging due to following reasons:

- **Dynamic topologies and membership**

The topology of Ad-hoc networks can change very fast, depending on the movement of the nodes. This results in regular changes of the available routes, changes in the reach ability of different nodes and changing participants.

- **No Central Entities**

Due to the dynamic nature of the Ad-hoc network, the functionality of all nodes should be equal. Central servers are not recommended for most application scenarios because of the complete break-down of the service when the server becomes unreachable for any reason. Regarding security services, also the high physical vulnerability of a single node should be taken into account.

- **Unidirectional Links**

Due to different power and transmission ranges of the nodes, unidirectional links can occur.

- **Constraints regarding**

  - **Processing Power**: Most of the scenarios for Ad-hoc networks assume mobile devices with small processors.
  - **Battery power**: If battery powered devices are used, the transmission power and the processor utilization will directly effect the battery lifetime.
  - **Bandwidth**: The scarcest resource in wireless world is bandwidth. This is especially true for mobile Ad-hoc networks that have to cope with a lot of additional signaling information and also have to act as relay-stations for neighboring network nodes.

- **Scalability**

Due to the high demands on decentralization and self-organization also scalability is an issue. Straightforward security mechanisms can have a high negative impact on system scalability.

Security issues in Ad hoc Networks

Ad-hoc networks have same security issues as infrastructure networks. The emphases on these issues just differ from those of more conventional systems. To secure an ad hoc network, we consider following services: confidentiality, Authentication, Integrity, Access control, Availability and Non-repudiation.

- **Confidentiality**: Confidentiality is protection of transmitted data from passive attacks. It ensures us that our information can't be disclosed to unauthorized entities. The other aspect of confidentiality is the protection of traffic flow from analysis. This requires that an attacker is unable to observe the source and destination, frequency, length or other characteristics of the traffic on a communications facility. The challenge in confidentiality is not only protecting data transported by network but also data stored on device.
- **Authentication**: It enables a node to ensure the identity of the peer node it is communicating with. Without authentication, an adversary could masquerade a node, thus gaining unauthorized access to resource and sensitive information and interfering with the operation of other nodes.
- **Integrity**: It assures that messages are received as sent with no duplication, modification, reordering or replays.
- **Access Control**: It is the ability to limit and control the access to host systems and applications via communication links. To achieve this control, each entity trying to gain access must first be identified, or authenticated, so that access rights can be tailored to the individual.
- **Non-repudiation**: It prevents either sender or receiver from denying a transmitted message. Non-repudiation is useful for detection and isolation of compromised nodes.

- **Availability:** It ensures the survivability of network services despite denial of service attacks. A denial of service attack can occur at any layer of ad hoc network. For example, on network layer an adversary could disrupt the routing protocol. On higher layer, an opponent could corrupt high level services too. One such target is the key management service, which is essential for any security framework.

Threats Faced by Ad hoc Networks

The most important threats that mobile ad hoc network has to face are:

- Attack on basic mechanism of the ad hoc network, such as routing. Prevention of these attacks requires security mechanisms that are often based on cryptographic algorithms. Routing mechanisms are more vulnerable in ad hoc networks than in conventional networks since in ad hoc network each device acts as a relay. This means, that an adversary who hijacks an ad hoc node could paralyze the entire network by disseminating false routing information. A less dramatic but subtler malicious behavior is node selfishness. Moreover, weakness in protocols can be exploited to perform malicious neighbor discovery.
- Attack on security mechanisms and especially on the key management mechanism. Key management is certainly not a problem limited to ad hoc networks; however, because of the peculiarities of ad hoc networks, its solution requires specific attention. Examples of attacks on security mechanism are: Public keys can be maliciously replaced; some keys can be compromised; if there is a distributed trusted server, it can fall under the control of a malicious party.

Previous Works

This section presents various security models for ad hoc networks. Each security model and its scope will be discussed one by one.

Resurrecting Duckling Policy

Authenticity can be achieved in mobile ad hoc networks (which involve only two devices or less computing devices) by Resurrecting Duckling policy, which was introduced by Frank Stajano and Ross Anderson in (F. Stajano and R. Anderson, 1999) and extended in (Stajano, 2000). It is particularly suited to devices without display and embedding a processor too weak for public-key operations. The fundamental authentication problem is a secure transient association between two devices establishing a master-slave relationship. It is secure in the sense that master and slave share a common secret and transient because the master can solve the association only. Also a master can always identify the slave in a set of devices.

The proposed solution is called the Resurrecting Duckling model. In this paper a slave has two exclusive states: imprinted and imprintable. The master controls his slave and they are bound to together with a shared secret that is originally transferred from master to slave over a non-wireless, confidential and integrated, channel. The slave is imprinted and made imprintable by his master. The slave becomes imprintable as a consequence of conclusion of a transaction or by an order of his master. An example of this policy is the usage of a ther[illegible]meter. The thermometer is calibrated once in six months. A doctor picks up any valid th[illegible]ter and wants it to communicate with e.g. her palmtop in order to receive the me[illegible] result. After this action the thermometer is free to be used by any other doctor.

The original "The resurrecting duckling" security policy was extended to cover also a peer-to-peer interaction (R. Anderson & Kuhn, 1996). In the extension a master can also be a human in addition to devices. In case of human master he can imprint the slave device by typing a PIN. A master can also be a millimeter-sized "dust mote" of which a system is consist of. There are a lot of dust motes in monitored area and each of them has battery, solar cell, sensors and some digital computing equipment. It may also have an active transmitter and receiver. These

can be used in military purpose. Another feature of this extension is that the master does not have to be unique. The slave can be imprinted also by another master that has a credential valid to that slave at the moment. The slave has a principle master, but it can receive some kind of orders also from other masters. In extension of the resurrecting duckling model the master can upload a new policy in slave, too.

The Resurrecting Duckling scheme is an appropriate model for a well-defined hierarchy of trust relationships. It particularly suits cheap devices that do not need a display or a processor to perform public-key operations. It perfectly works for a set of home devices. However, more flexible ad-hoc networks may not contain explicit trust relationships between each pair of nodes or to a centralized entity like the mother duck. Deploying a comprehensive network consisting of a hierarchy of a global mother duck and multiple subsidiary local mother duck is very similar to a public-key infrastructure, where the mother duck correspond to Certification Authority (CA), with all its advantages and drawbacks. Even the battlefield scenario raises some problems. Here the soldiers are siblings and obey their mother, the general. If one soldier device wants to authenticate to another device it has to present its credentials. The second device can then check the Credentials by using its policy. But what happens if all soldiers' do not use the same credentials, i.e. the same secret key to prevent it to be stolen by the enemy. If all devices use the same key the other side might invest considerable effort doing some physical attack (R. Anderson & Kuhn, 1996) to recover the key because it would compromise all nodes. Since the devices cannot hold a list of all valid credentials it seems that a further authentication method is needed.

Password-based Key Agreement

The work developed in (Asokan & Ginzboorg, 2000) addresses the scenario of a group of people who wants to set up a secure session in a meeting room without any support infrastructure. People physically present in the room know and trust one another personally. However they do not have any prior means of digitally identifying and authenticating one another, such as shared secret or mutually verifiable public key certificate chains or access to trusted key distribution centers. An attacker can monitor or modify all traffic on the wireless communication channel and may also attempt to send messages purporting to come from those who are inside the room. There is no secure communication channel to connect the computers. Desirable properties of a protocol that solves this problem should be:

- **Secrecy**: The basic requirement of secrecy is that only those entities who know an initial password should be able to learn the resulting session key. An observer must not be able to get any information about the session key.
- **Perfect forward secrecy** requires that an attacker who succeeds in compromising one member of the group and learns about his permanent secret information will still be unable to recover the session keys resulting from previous runs of the protocol.
- **Contributory key agreement** The resulting session key is established by the contribution from all entities participating in the meeting. This ensures that if only one entity chooses its contribution key randomly all other entities will not be able to make the key space smaller.
- **Tolerance to disruption attempts** The strongest attacker can disrupt any protocol by jamming the radio channel or modifying the contents of messages among legitimate members. The protocol must not be vulnerable to an attacker who is able to insert messages. It is assumed that the possibility of modifying or deleting messages in such an ad-hoc network is very unlikely.

The work describes and introduces several password-based key-exchange methods that meet these requirements. A weak password is sent to the group members. Each member then contributes to part of the key and signs this data by using the weak password. Finally a secure session key to set up a secure channel is derived without any central trust authority or support infrastructure.

Firstly, a well-known two-party protocol for password authenticated key exchange is described, based on the protocol called Encrypted key exchange (EKE) (Steven M. Bellovin & Michael Merit, 1992) and extends the work from two-party to multiple parties by electing a leader. The main drawback of the multiparty version is that the leader chooses the common session key unilaterally: the key agreement scheme is non-contributory. The protocol is then modified to extend it to a contributory multi-party protocol.

Secondly, Diffie-Hellman exchange (DH), a classic two-party key agreement protocol is extended to support multi-party password authenticated key exchange. An extension of unpublished protocol by Steiner et al (1996), in which each member shares a different password with an authentication server, is used. This new protocol provides perfect forward secrecy to all players. It is also resilient to disruptions. The leader can disrupt the protocol completely. Any other member attempting to disrupt the protocol by sending out a random quantity will not be able to compute the session key.

Finally, a fault-tolerant Diffie-Hellman exchange on a d-cube is extended to handle failures which were based on the idea proposed by Becker and Wille.

Password-Based key agreement model perfectly works for small groups. Authentication is done outside the IT system, e.g. the group members authenticate themselves by showing their passport or common knowledge. This model does not suffice anymore for more complicated environments, though. Groups of people who do not know each other or number of people who want to have confidential exchanges without bringing in knowledge of the rest of the group be able to eavesdrop on the channel, are two examples. Another problem arises for large groups or groups at different locations. The secure channel to distribute the initial password is not available anymore. It seems that existing support infrastructure is required to set up a secure channel.

Distributed Public-Key Management

Key management for established public-key systems requires a centralized trusted entity called Certificate Authority (CA). The CA issues certificates by binding a public key to a node's identity. One constraint is that the CA should always be available because certificates might be renewed or revoked. Replicating the CA improves availability. However, a central service goes against the distributed structure of ad-hoc networks.

L. Zhou and Z.J. Haas (1996) proposes to distribute trust to a set of nodes by letting them share the key management service, in particular the ability to sign certificates. This is done using threshold cryptography (Desmedt & Frankel, 1990). An (n, t + 1) threshold cryptography scheme allows n parties to share the ability to perform a cryptographic operation so that any t+1 parties can perform this operation jointly whereas it is infeasible for at most t parties to do so. Using this scheme the private key k of the CA is divided into n shares (S1, S2, Sn), each share being assigned to each special node. Using this share a set of t + 1 special node is able to generate a valid certificate. As long as t or less special nodes are compromised and do not participate in generating certificates the service can operate.

Even if compromised nodes deliver incorrect data the service is able to sign certificates. Threshold cryptography can be applied to well use signature schemes like the Digital Signature Standard (DSS) (Gennaro et al., 1996).

Another approach introduced in (Hubaux et al., 2001) presents a self-organized public-key infrastructure. The system replaces the centralized CA by certificate chains. Users issue certificates if they are confident about the identity, i.e. if they believe that a given public key belongs to a given user. Each user stores a list of certificates in its own repository. To obtain the certificate of another entity the user builds a certificate chain using his repository list and implicitly trusted entity's lists, until a path to an entity that has the desired certificate in its repository is found.

The Threshold Key Management system is a way to distribute a public-key system. For high-value transactions public-key systems are certainly the only way to provide a satisfactory and legal security framework. Trust should, as much as possible, be based on knowledge. Since two entities that never met before cannot have common knowledge - a shared secret - they both have to trust a central entity, e.g. a CA to substitute this knowledge. When a user wants to prove his identity to CA, he goes there with his public key and shows his passport. The CA proves his identity and then binds his identity to his public key and signs the certificate. How it can be done when the CA is distributed? The user must prove his identity to all special nodes to prevent that a compromised node passes on faulty information. In this case friendship or just knowing each other is considered as common knowledge. The CA signs certificates without proving the identity, that's why; this model cannot be used for high-value transactions.

The self-organized public-key infrastructure (Hubaux et al., 2001) shows similar problems. To make the system bullet-proven, the entity's identity had to be checked in real world before users issue certificates. Furthermore it is assumed that the certificate requester trusts each node in the recommendation chain. Finally a significant computing power and time is consumed to obtain a certificate going through the certificate chain. Each node in the chain has to perform public-key operations, first to check the received certificate for authentication (signature verification) and then to sign it before forwarding it (signature generation). This cannot be done in parallel but only one after the other.

Despite its centralized nature a central CA is preferable for applications with high-security demand. To ensure high availability the CA can be replicated. The replicated CA's are as secure as the original CA as long as the replication process is not vulnerable to attacks. The private key of the CA does not get weaker after replication. Much research has been done about efficient public-key systems - for example, a public-key system for mobile systems is presented in (G. Horn & B. Preneel, 1998).

Proposed System Model

Imagine a situation in which bank customers are standing in a queue and two or three bank employees are dealing them in making transactions. The customers have to spend a lot of time and also have to wait for their turn in the queue. The bank needs more employees for giving better service to its customers. This scenario, both for the bank and the customers, can be improved by deploying ad hoc networking using Resurrecting Duckling Policy (Stajano and Anderson, 1999) and (Stajano, 2000), with some extensions in our idea. Here, we have also taken inspiration from work of D. Balfanz (Balfanz & Wong, 2002).

Our idea is elaborated as: A user (Bank customer) enters the bank, with his laptop or PDA, to perform some transactions. The bank has mounted the Infrared Bar Code at one side in the bank and also owns a wireless radio link network (e.g. Bluetooth (www.bluetooth.com) or 802.11). The user or customer enters the bank premises, would walk up to the Infrared bar code and briefly establish physical contact between infrared bar code and his laptop or PDA. This process is termed as ***Pre-authentication***[1], which is done using a ***link-constrained channel***[2].

During Pre-authentication, their public keys will be exchanged. Now the user can sit in the bank premises and his laptop or PDA can then perform standard SSL/TLS (Dierks & Allen, 1999) key exchange with the bank server over the wireless link (e.g. Bluetooth or 802.11), since he owns a secret key to perform secure transactions and can establish an authenticated and secret communication channel.

---

[1] First phase in which duckling and mother exchange some secret information over location-limited channel.

[2] This notion of location-limited channel was introduced by Stajno and Anderson , as a part of "Resurrecting Duckling" model for ad hoc networks.

Such an exchange of pre-authenticated data ensures the user that he wants to communicate with the device (infrared bar code) by using a special, link-constrained side channel to exchange a small amount of cryptographic information. That information can be used to authenticate standard key exchange protocols performed over wireless link.

Once the pre-authentication is completed, the devices proceed to establish a secure connection between them over the main wireless link. To this end, they can use any established public-key-based key exchange protocol which requires them to prove possession of a particular private key (e.g., SSL/TLS, SKEME IKE, etc.), which will correspond to the public key committed to in the pre-authentication step.

After secret key exchange, the customer or user will send his/her account number along with the password or secret code (which can be issued at the time of opening an account in the bank and can be changed at any time through bank website) to the bank server. The A/C No. plus code or password will be encrypted with secret key (Session key) exchanged during pre-authentication. After this phase, the user can perform his transactions securely over the wireless link (e.g. Bluetooth or 802.11).
We can summarize the basic scheme for pre-authentication as follows.

A) Pre-authentication over location-limited channel.

$$1\text{- } X \longrightarrow Y\text{: } H(KU_X)$$
$$2\text{- } Y \longrightarrow X\text{: } H(KU_Y)$$

B) Authentication over wireless channel with SSL/TLS.

$$X \longrightarrow Y\text{: TLS\_CLIENT\_HELLO ..... And so on.}$$

The various symbols denote:
X: Customer's Laptop
Y: Bar code device
KU_x, KU_Y: Public key belonging to X and Y respectively.
$H(KU_x)$, $H(KU_Y)$: one-way hash of encoding of corresponding keys.

Using RSA for public-key Cryptography

In our imagined scenario (Banking Scenario), initially we used SSL (P. V. Jani, 2002) (Secure socket layer) protocol with RSA-based public-key cryptography. SSL offers encryption, source authentication and integrity protection for data exchanged over insecure, public networks. It operates above a reliable transport service such as TCP and has the flexibility to accommodate different cryptographic algorithms for key agreement, encryption and hashing.

The two main components of SSL are the Handshake protocol and the Record Layer protocol. The Handshake protocol allows an SSL client and server to negotiate a common cipher suite, authenticate each other, and establish a shared master secret using public-key algorithms. The Record Layer derives symmetric keys from the master secret and uses them with faster symmetric-key algorithms for bulk encryption and authentication of application data. Public-key cryptographic operations are the most computationally expensive portion of SSL processing, and speeding them up remains an active area for research and development.

RSA-based Handshake

The client and server exchange random nonces (used for replay protection) and negotiate a cipher suite with ClientHello and ServerHello messages. The server then sends its signed RSA public-key either in the Certificate message or the ServerKeyExchange message. The client verifies the RSA signature, generates a 48-byte random number (the pre-master secret) and sends it encrypted with the server's public-key in the ClientKeyExchange. The server uses its RSA private key to decrypt the pre-master secret. Both end-points then use the pre-master

secret to create a master secret, which, along with previously exchanged nonces, is used to derive the cipher keys, initialization vectors and MAC (Message Authentication Code) keys for bulk encryption by the Record Layer.

The server can optionally request client authentication by sending a CertificateRequest message listing acceptable certificate types and certificate authorities. In response, the client sends its private key in the Certificate and proves possession of the corresponding private key by including a digital signature in the CertificateVerify message.

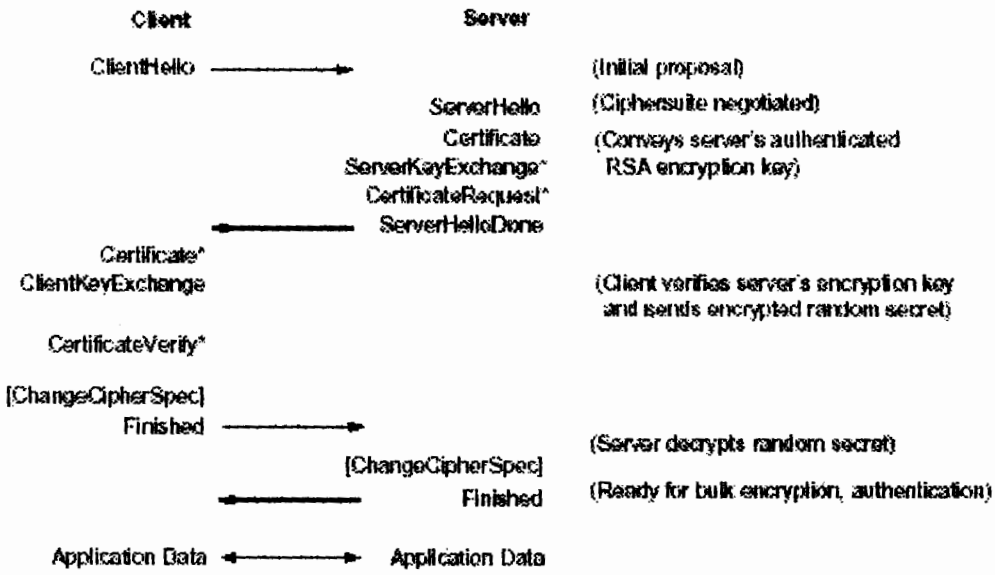


Fig. 1 SSL Handshake for an RSA-based Cipher Suit

Using ECC for Public-key cryptography

Elliptic curve cryptography

Elliptic curve cryptography was proposed by Victor Miller (Miller, V., 1986) and Neal Koblitz (Koblitz, N., 1987) in the mid 1980's.Now Elliptic Curve Cryptography (ECC) has evolved into a mature public-key cryptosystem. Extensive research has been done on the underlying math, its security strength, and efficient implementations.

At the foundation of every public key cryptosystem is a hard mathematical problem that is computationally infeasible to solve. For instance, RSA and Diffie- Hellman rely on the hardness of integer factorization and the discrete logarithm problem respectively. Unlike these cryptosystem which operate over integer fields, the Elliptic Curve Cryptosystems (ECC) operates over points on an elliptic curve.

Elliptic curve cryptography (Miller, 1986), (Lopez & Dahab, 2000), (Koblitz, 1987) and (Koblitz, 1994) provides a methodology for obtaining high-speed, efficient, and scalable implementations of network security protocols. The security of these protocols depends on the difficulty of computing elliptic curve discrete logarithm in the elliptic curve group. The group operations utilize the arithmetic of points which are elements of the set of solutions of an elliptic curve equation defined over a finite field. The arithmetic of elliptic curve operations depends on the arithmetic on the underlying finite field. The standards suggest the use of GF(p) and GF(2k).

The security of ECC relies on the hardness of solving the Elliptic Curve Discrete Logarithm Problem (ECDLP), which states that given P and Q = kp, it is hard to find k. While a brute-force

approach is to compute all multiples of P until Q is found, k would be so large in a real cryptographic application that it would be infeasible to determine k in this way.

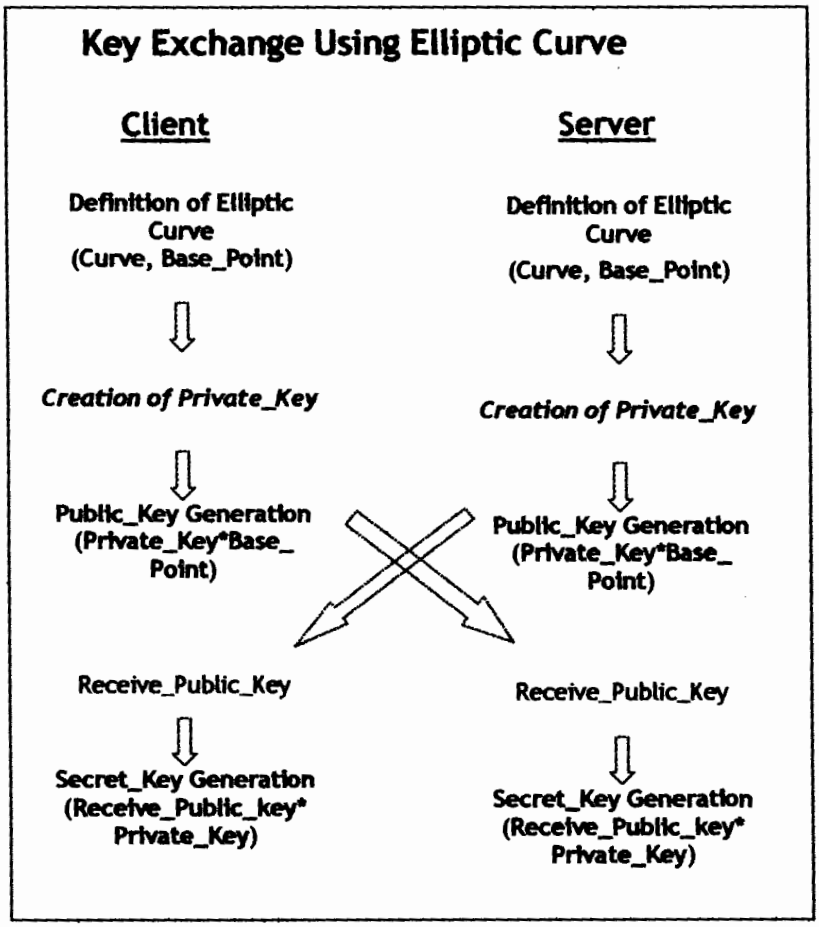


Fig. 2 Key exchange using Elliptic curve Cryptography

ECC-based Handshake

The processing of the first two messages is the same as for RSA but the Certificate message contains the server's Elliptic Curve Diffie-Hellman (ECDH) public key signed with the Elliptic Curve Digital Signature Algorithm (ECDSA). After validating the ECDSA signature, the client conveys its ECDH public key in the ClientKeyExchange message. Next, each entity uses its own ECDH private key and the other's public key to perform an ECDH operation and arrive at a shared pre-master secret. The derivation of the master secret and symmetric keys is unchanged compared to RSA. Client authentication is still optional and the actual message exchange depends on the type of certificate a client possesses.

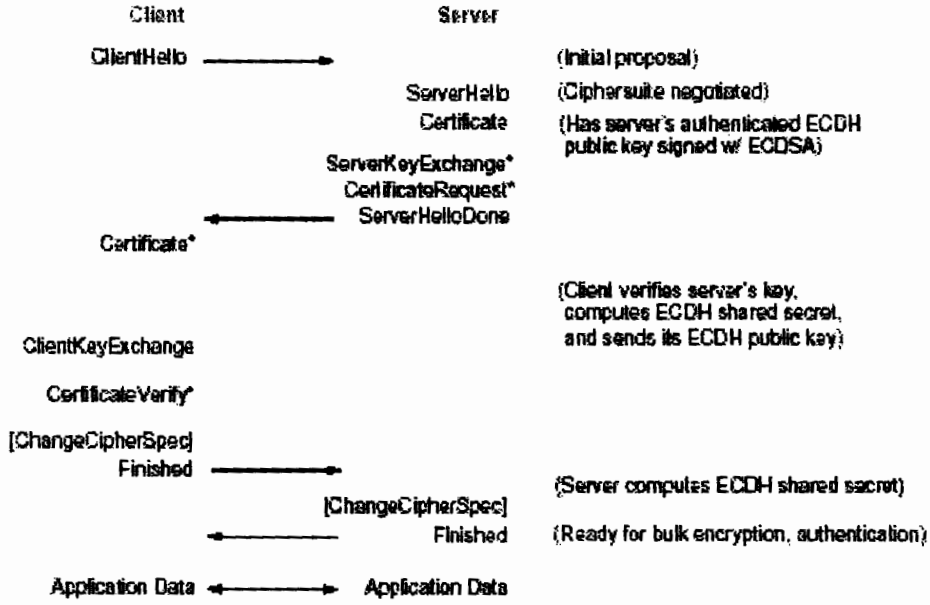


Fig. 3 ECC-based SSL Handshake

Simulation

We have divided our work in to three major areas. The first step is to implement exchange of pre-authentication data. We have exchanged hash of public keys. Hash code is generated by using SHA. Public/Private key pair is generated by using RSA but later we have implemented ECC (Koblitz, N., 2000) for this purpose, as it is more suitable for small devices.

In second step, we have implemented SSL with ECC support, for complete exchange of keys and these keys are authenticated by using pre-authentication data.

For these two steps we have used socket programming. The server sockets listens for a connection on both location-limited channel and the primary link, but only admits primary-links connection from clients, who have performed pre-authentication on location-limited channel. Currently we have used serial cable for location-limited channel.

The third step is to transfer data from client to server. For this purpose we have utilized Microsoft Access for database and this whole framework is implemented in VC++6.

Simulation Results

As part of adding ECC support to OpenSSL (www.openssl.org), the most widely used open source implementation of SSL. We have enhanced the OpenSSL0.9.6b cryptographic library to support ECDH and ECDSA. We have also added the ability to generate and process X.509 certificates containing ECC keys.

Table 1 show the measured performance of primitive RSA, ECDH, and ECDSA operations using the OpenSSL0.9.6b speed program (enhanced to include ECC) on a Linux PDA equipped with a 200MHz Strong ARM processor.

	$RSA_{encypt,verify}$	$RSA_{decrypt,verify}$	$ECDSA_{verify}$	$ECDSA_{sign}$
Linux	10.8	188.7	46.5	24.5
PDA	39.1	1273.8	76.6	39.0

Future Work

The scheme presented has distinct advantages over traditional authentication and security models. We are focusing our future work on different scenarios relating to security and authentication problems, keeping in view the bandwidth-constrained media and computation limited devices.

Now we are going to implement our work by utilizing Bluetooth devices. To show actual transactions we are using PHP and MySql. Our work will mainly focus on upper layers (e.g. RFCOMM layer, SDP, L2CAP etc.) of Bluetooth by using open source code of Bluetooth device driver in Linux.

Conclusion

There is no any single authentication model which ensures authentication in every environment. While for a group meeting in a small conference room the Password-based key exchange will work perfectly, for a network defined by hierarchy of trust relationships, the Resurrecting-Duckling policy is an ideal alternative. For guaranteed secure transaction certainly the choice is to use public key system involving the hassle of group certificates.

In this paper we have presented new approaches for peer-to-peer authentication in mobile ad hoc networks. We have constructed our schema on previous work by Anderson, Stajano and others, and presented to perform pre-authentication over location-limited channels. Our schema is public key infrastructure less and resolves naming problem that plagues traditional authentication systems.

At present we have tried to implement ECC in SSL and have compared its results. In future, we will use block ciphers, other signatures and hash algorithms and expand arithmetic operations to integers with different representations of polynomials (different bases) and coordinate systems.

References

L. Zhou and Z..J. Haas. Securing Ad Hoc Networks. IEEE Network Magazine, vol. 13, no. 6, 1999.

William Stallings. Network and Internet work Security Principles and practice.

J.-P. Hubaux, L. Butty´an, and S. ¡ Capkun. The quest for security in mobile ad hoc networks. In Proceedings of the 2nd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), October 2001.

F. Stajano and R. Anderson. The Resurrecting Duckling: Security Issues for Ad hoc Wireless Networks. The 7th International Workshop on Security Protocols, LNCS 1796, Springer-Verlag, 1999.

F. Stajano. The Resurrecting Duckling-what next? The 8th International Workshop on Security Protocols, LNCS 2133, Springler-Verlag, 2000.

R. Anderson and M. Kuhn. Tamper resistance - a cautionary note. 2nd USENIX Workshop on Electronic Commerce, 1996.

N. Asokan and P. Ginzboorg. Key Agreement in Ad-hoc Networks. Computer Communications 23, 2000.

Steven M. Bellovin and Michael Merrit. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In proceedings of the IEEE Symposium on Research in Security and Privacy, May 1992.

Michael Steiner, Gene Tsudik, and Michael Waidner. Private communication. Unpublished work described in slides of presentation made at the ACM CCS conference, March 1996.

Klaus Becker and Uta Wille. Communication complexity of group key distribution. In 5th ACM Conference on Computer and Communications Security, pages 1-6.

Y. Desmedt and Y. Frankel. Threshold cryptosystems. Advances in Cryptology -Crypto '89, LNCS 435, Springer-Verlag, 1990.

R. Anderson and M. Kuhn. Tamper resistance - a cautionary note. 2nd USENIX Workshop on Electronic Commerce, 1996.
R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. Advances in Cryptology - Eurocrypt '96, LNCS 1070, Springer-Verlag, 1996.
G. Horn and B. Preneel. Authentication and Payment in Future Mobile Systems. Computer Security - ESORICS '98, LNCS 1485, Springer-Verlag, 1998.
The official Bluetooth SIG website. www.bluetooth.com.
Preetida Vinayakray jani. Security within Ad hoc Networks, Position paper, PAMPAS Workshop, September 16/17 2002, London.
D. Balfanz, D. K. Smetters, P. Stewart and H. Chi Wong, "Talking To Strangers: Authentication in Ad-Hoc Wireless Networks", Internet Society, Conference Proceeding of NDSS Conference 2002.
T. Dierks and C. Allen. *The TLS Protocol Version 1.0.* IETF Network Working Group, the Internet Society, January 1999. RFC 2246.
A. Frier, P. Karlton and P. Kocher, \The SSL3.0 Protocol Version 3.0", see http://home.netscape.com/eng/ssl3/.
Miller, V.: Uses of elliptic curves in cryptography. In Lecture Notes in Computer Science 218 Advances in Crytology - CRYPTO '85, pages 417-426, Springer-Verlag, Berlin, 1986.
Koblitz, N.: Elliptic curve cryptosystems. Mathematics of Computation, 48:203-209, 1987.
J. Lopez and R. Dahab, "An Overview of Elliptic Curve Cryptography," Institute of Computing, State University of Campinas (Brazil: 2000) Available from http://citeseer.nj.nec.com/
N. Koblitz. A Course in Number Theory and Cryptography. New York, NY: Springer-Verlag, Second edition, 1994.
The OpenSSL Project, see http://www.openssl.org/.

Bibliography & References

Bibliography and References

Research Papers

[1] Charles E. Perkins, Ad hoc Networking, 2001, Addison-Wesley, London; ISBN: 0-201-30976-9.

[2]Yongguang Zhang, Wenke Lee, Intrusion Detection in ad hoc Networks.

[3] J.-P. Hubaux, L. Butty´an, and S. ¡ Capkun. The quest for security in mobile ad hoc networks. In Proceedings of the 2nd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), October 2001.

[4] William Stallings. Network and Internet work Security Principles and practice.

[5] F. Stajano and R. Anderson. The Resurrecting Duckling: Security Issues for Ad hoc Wireless Networks. The 7th International Workshop on Security Protocols, LNCS 1796, Springer-Verlag, 1999.

[6] F. Stajano. The Resurrecting Duckling–what next? The 8th International Workshop on Security Protocols, LNCS 2133, Springler-Verlag, 2000.

[7] Konrad Lorenz. Er redete mit dem Vieh, den V¨ogeln und den Fischen (King Solomon's ring). Borotha-Schoeler, Wien, 1949.

[8] B.Warneke, M. Last, B. Leibowitz, and K.S.J. Pister. Smart Dust: Communicating with a Cubic-Millimeter Computer. Computer Magazine, IEEE, Jan. 2001.

[9] R. Anderson and M. Kuhn. Tamper resistance – a cautionary note. 2nd USENIX Workshop on Electronic Commerce, 1996.

[10] N. Asokan and P. Ginzboorg. Key Agreement in Ad-hoc Networks. Computer Communications 23, 2000.

[11] Steven M. Bellovin and Michael Merrit. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In proceedings of the IEEE Symposium on Research in Security and Privacy, May 1992.

[12] Michael Steiner, Gene Tsudik, and Michael Waidner. Private communication. Unpublished work described in slides of presentation made at the ACM CCS conference, March 1996.

[13] Klaus Becker and Uta Wille. Communication complexity of group key distribution. In 5th ACM Conference on Computer and Communications Security, pages 1-6.

[14] L. Zhou and Z.J. Haas. Securing Ad Hoc Networks. IEEE Network Magazine, vol. 13, no. 6, 1999.

[15] Y. Desmedt and Y. Frankel. Threshold cryptosystems. Advances in Cryptology –Crypto '89, LNCS 435, Springer-Verlag, 1990.

[16] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. Advances in Cryptology – Eurocrypt '96, LNCS 1070, Springer-Verlag, 1996.

[17] G. Horn and B. Preneel. Authentication and Payment in Future Mobile Systems. Computer Security - ESORICS '98, LNCS 1485, Springer-Verlag, 1998.

[18] The official Bluetooth SIG website. www.bluetooth.com.

[19] T. Dierks and C. Allen. *The TLS Protocol Version 1.0.* IETF Network Working Group, the Internet Society, January 1999. RFC 2246.

[20] J. Lopez and R. Dahab, "An Overview of Elliptic Curve Cryptography," Institute of Computing, State University of Campinas (Brazil: 2000) Available from **http://citeseer.nj.nec.com/**

[21] M. Rosing, Implementing Elliptic Curve Cryptography, (Greenwich: Manning Publications Co.,1999).

[22] D. Balfanz, D. K. Smetters, P. Stewart and H. Chi Wong, "Talking To Strangers: Authentication in Ad-Hoc Wireless Networks", Internet Society, Conference Proceeding of NDSS Conference 2002.

[23] V. Miller. Uses of elliptic curves in cryptography. In H. C. Williams, editor, Advances in Cryptology — CRYPTO 85, Proceedings, Lecture Notes in Computer Science, No. 218, pages 417–426. New York, NY: Springer-Verlag, 1985.

[24] N. Koblitz. Elliptic curve cryptosystems. Mathematics of Computation, 48(177):203–209, January 1987.

[25] A. J. Menezes. Elliptic Curve Public Key Cryptosystems. Boston, MA: Kluwer Academic Publishers, 1993.

[26] N. Koblitz. A Course in Number Theory and Cryptography. New York, NY: Springer-Verlag, Second edition, 1994.

[27] The OpenSSL Project, see http://www.openssl.org/

Books

[1] Ad Hoc Networking by Charles E.Perkins and Elizabeth M.Royer Pages 173-219.

[2] Software Engineering A Practitioner's Approach (Fourth Edition) by Roger S. Pressman

1992, McGraw-Hill Companies Inc.

[3] Network and Internetwork Security by William Stallings.