A HIERARCHICAL TRUSTED THIRD-PARTY SYSTEM FOR SECURE PEER-TO-PEER TRANSACTIONS

A Written Report

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Computer Science

by

Khoi Vu Nguyen

April 2007

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Mark Stamp

Dr. Robert Chun

Dr. Sami Khuri

APPROVED FOR THE UNIVERSITY

ABSTRACT

A HIERARCHICAL TRUSTED THIRD-PARTY SYSTEM FOR SECURE PEER-TO-PEER TRANSACTIONS

by Khoi Vu Nguyen

A peer-to-peer (P2P) network is a distributed network of peer computers loosely connected through the Internet. Transactions in a P2P network are often conducted on a no-security basis. Moreover, peer anonymity is often highly desirable, which makes security even more difficult to achieve. In most cases, a peer executes a transaction solely based on the faith that the other peer plays by the rules.

Here we propose a hierarchical Trusted Third-Party (TTP) system that facilitates secure transactions between peers in an existing P2P network. This system is designed to provide mutual authentication by using public key cryptography for peers to authenticate the TTP system and by using symmetric key cryptography for the TTP system to authenticate peers. After logging into the system, two peers can obtain a shared secret key from the TTP system to form a secure channel over which all transactions are encrypted using the secret key. The TTP system is designed to operate as an independent entity that peers can choose to join independently of their P2P network and can remain anonymous among each other.

In addition, a reputation scheme, in which peers rate each other, is employed in the TTP system. This self-policing system provides a relative measure of trust among peers so that a peer can decide whether to allow a transaction based on another peer's rating. The anonymity of peers in P2P systems creates many difficulties for establishing an accurate rating system. However, we believe this is still achievable to a degree.

ACKNOWLEGEMENTS

I would like to thank my thesis adviser, Dr. Mark Stamp, for introducing me to this fascinating field and guiding me through this thesis. Without his insightful feedback, this thesis would not have succeeded. I would like to express my thanks to my committee members, Dr. Robert Chun and Dr. Sami Khuri, for valuable comments.

This paper would not have been possible without the valuable help of my brother, Nguyen Nguyen. I also want to thank my family for all of their support and encouragement as I completed this thesis, especially my lovely wife, Silvie Phan.

Contents

Figures

1 Background

1.1 Problem Statement and Motivation

A P2P network consists of loosely connected peers and decentralized resources. Often, there is also a desire for some degree of anonymity. Because of the desire for anonymity, security is routinely omitted. When most P2P networks attempt to provide security features, they become too restrictive to enable network growth. Therefore, P2P networks in use today usually have minimal or no security features for providing authentication of peers and confidentiality of transactions. Most P2P networks resolve security issues by limiting the types of operations that peers can perform on network resources. While this approach works well in most cases, it limits the potential capabilities of P2P networking to predominantly file sharing.

P2P networks present many security concerns to an organization's network since it opens a presumably secure network to various forms of attacks such as viruses, worms, data theft, intellectual property loss, network bandwidth clogging, backdoor access, and so on. By decentralizing resources, P2P networks compromise an organization’s security policies and potentially place the organization’s network at risk.

However, P2P networking can also be used to enhance and expand network-computing capabilities when certain security measures are put in place. One such security requirement is the ability to communicate safely and securely between peers. In some cases it is also important that peers can communicate securely without having to reveal their identities to each other. If this problem was addressed successfully, additional P2P networking possibilities could be explored, including the ability for peers to share files securely with certain peers on the network, the ability for peers to lend unused CPU cycles to certain peers securely, the ability for peers to securely send sensitive information to other peers over an open P2P network, and so on. Therefore, the potential contributions of our TTP system in P2P networks are limitless.

1.2 Requirements and Goals

The followings are the requirements and goals of our TTP system.

- Mutual authentication between peers and the TTP system
- Scalability of the TTP system, allowing future growth
- Independence of the TTP system from the underlying P2P network
- High accuracy and fault-resilient peer ratings
- Plug-in connectivity to existing P2P networks
- Use of stateless TTP servers
- Peer anonymity achieved at the authentication level

1.3 Secure Socket Layer Protocol

The Secure Socket Layer (SSL) protocol was developed by Netscape to address the need for secure transactions over the Internet [11]. SSL uses public key cryptography to provide authentication, integrity, and confidentiality. In practice, it is used primarily when clients need to authenticate a server, although it can also provide mutual authentication.

The basic idea of an SSL session is illustrated in Figure 1 below [8]. In this scenario, a server named Bob has a pair of keys, one public and one private. The public key, which is signed by a trusted Certificate Authority, is distributed widely and the private key is kept secret. In this protocol, a client named Alice sends a request for a secure transaction with the server along with her supported cipher list and a nonce R_A. The server replies with a certificate to the client along with his chosen cipher and another nonce R_B. Next, the client sends her generated "pre-master secret" S, encrypted with the server's public key, along with a hash of messages encrypted with the key $K = h(S, R_A, R_B)$, which is used to verify the correctness of the preceding messages. Last, the server replies with a similar hash, which the client can decrypt to authenticate the server by verifying that the server had received the previous messages correctly and was able to obtain the secret S. At this point, both the server and the client have established a shared symmetric key K that they can use to encrypt subsequent messages. Note that in this protocol the client authenticates the server but the server does not authenticate the client.

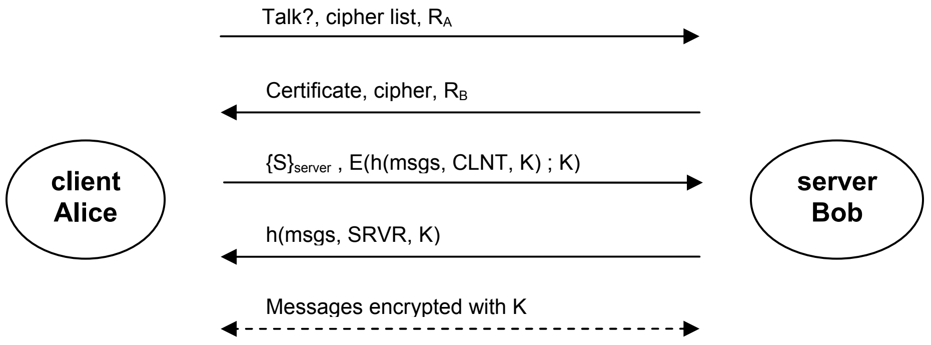

Talk?, cipher list, R_A
Certificate, cipher, R_B
$\{S\}_{server}$, E(h(msgs, CLNT, K) ; K)
h(msgs, SRVR, K)
Messages encrypted with K
client Alice
server Bob

Figure 1. Simplified SSL [8]

After an SSL session is established, the client can open multiple subsequent SSL connections using the symmetric key K, avoiding expensive public key operations.

1.4 Kerberos Authentication Protocol

Kerberos is a network authentication protocol developed at MIT [6]. It is based on symmetric key cryptography and a trusted third-party model. The Kerberos protocol uses a trusted third-party entity called the Key Distribution Center (KDC), which consists of two parts: an Authentication Server (AS) and a Ticket Granting Server (TGS). Kerberos is based on the use of "tickets" to authenticate users [8]. The KDC has a secret master key, K_{KDC}, which is used to encrypt the user's Ticket Granting Ticket (TGT) [8]. This special ticket is used to obtain ordinary tickets that are used for accessing network resources [6].

After a user named Alice logs in with her password, the key K_A = hash(Alice's password) is computed. Alice is then granted a Ticket Granting Ticket, TGT = E("Alice", S_A; K_{KDC}), by the TGS, which she can use to obtain other tickets for accessing network resources. Here, S_A is the session key that KDC assigns to Alice, and K_{KDC} is the KDC's secret key. Figure 2 illustrates a scenario in which user Alice uses her TGT to obtain access to a network resource named Bob.

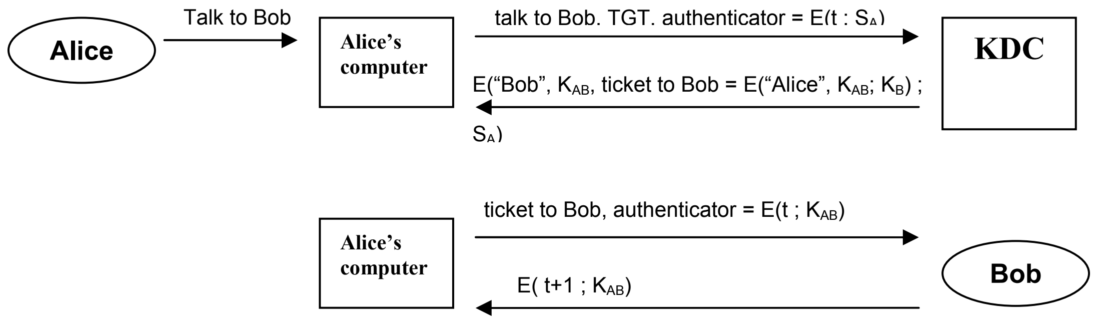

Alice
Talk to Bob
Alice's computer
talk to Bob. TGT. authenticator = E(t : S_A)
KDC
E("Bob", K_AB, ticket to Bob = E("Alice", K_AB; K_B) ; S_A)
Alice's computer
ticket to Bob, authenticator = E(t ; K_AB)
Bob
E(t+1 ; K_AB)

Figure 2. Using Kerberos Tickets [8]

Using her TGT and an authenticator, Alice issues a request to the KDC for access to Bob. The KDC replies with a "ticket to Bob," which only Bob can decrypt, and a shared session key, K_{AB}, that Alice and Bob use to encrypt their session transactions. Using this "ticket to Bob," Alice can communicate securely with Bob by using the shared session key K_{AB}.

2 Introduction

A P2P network is a distributed network of peer computers connected loosely through the Internet. Desirable characteristics of a P2P network are:

- There are many peers
- Resources are distributed
- Peers are anonymous in the P2P network
- Each peer is both a server and client
- Each peer has roughly equal functionality

One of the main characteristics of a P2P network is the ability for peers to remain anonymous in the P2P network. This usually means that a peer's representation must not contain any direct identifiable information[1]. Although this characteristic provides tremendous flexibility in a P2P network, it places a heavy burden on security implementations to maintain anonymity.

[1] A peer in a P2P network is highly, but not completely, anonymous because it still has an "address" [18].

In general, the number of peers in a P2P network can be very large. Therefore, it is essential that any security scheme must be efficient and easily scalable as the number of peers grows. Since P2P networks are based on decentralized resources, public key cryptography is a natural choice. However, it is impossible to implement public key cryptography directly on peers in a P2P network because each peer would have to maintain a pair of public-private keys, which would require the existence of a Certificate Authority to authenticate and publish the public keys to all other peers. Expecting public-key certification for every peer is impractical because of the high economic cost associated with doing so. On the other hand, symmetric key cryptography would require each pair of peers to share a symmetric key, requiring in $N(N-1) \approx N^2$ total keys for a network of N peers [6]. Obviously, this will not scale well as the number of peers grows.

Considering these issues, a third-party model is a logical solution. A Trusted Third Party (TTP) system resides independently from the P2P network and shares one unique symmetric key with each peer in the network, for a total of N symmetric keys. Using a symmetric key cryptography TTP system, peers can authenticate each other with the help of the TTP system. Peers can also establish secure channels with other peers in the P2P network by requesting the TTP system to generate symmetric keys that only the communicating peers know, while the peers remain anonymous to each other. However, mutual authentication is not achieved using this scheme alone since peers cannot authenticate the TTP system.

In light of these factors, a hybrid scheme emerges as the best candidate for the TTP system. In such a scheme, symmetric key cryptography is used along with limited public key cryptography. Public key cryptography in the form of the Secure Sockets Layer Protocol (SSL) [11], is used to authenticate the TTP system. After a secure SSL session or connection has been established between a peer and the TTP system, a symmetric key cryptography system implementing the Kerberos Authentication Protocol [6] is used to authenticate peers and to generate tickets. These two combined protocols provide mutual authentication while improving system efficiency since only a symmetric key is needed for every peer and only one public-key pair is needed for the entire system.

It is natural to use SSL to authenticate a TTP system because a TTP system represents a single entity that all peers must share and trust. On the other hand, it makes sense to use Kerberos authentication for the peer-to-peer interactions because it does not require public key certificates for all peers, but yet it scales well, due to the use of a TTP. Furthermore, a Kerberized TTP system can help peers to establish secure communication channels between them by generating symmetric keys stored in encrypted tickets that can only be decrypted by the communicating peers. These two security systems work hand-in-hand to provide mutual authentication, taking advantage of the superior performance of Kerberos's symmetric key cryptography and the superior convenience of SSL's public key cryptography.

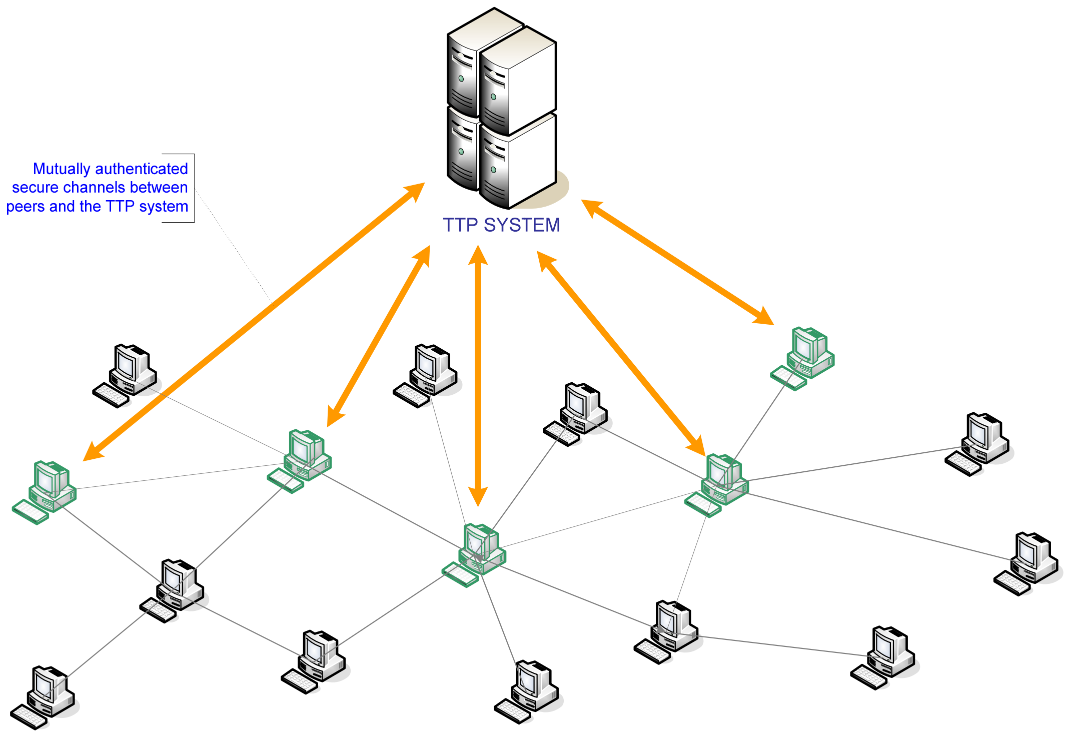

Mutually authenticated secure channels between peers and the TTP system
TTP SYSTEM

Figure 3. The TTP system in P2P Network

Using this TTP system, peers can communicate safely and securely with each other. However, an important question remains: how can a peer determine whether he should trust another peer before establishing a secure channel with that peer? This problem is addressed by incorporating a reputation-based scheme in which peers rate one another after every transaction. Peers' reputations are based on their past behavior. Therefore, they theoretically should behave well to earn reputations that will allow them to be trusted by others. Peer ratings are not stored in the peers, but are stored in the TTP system along with other peer data so that the integrity of the ratings can be guaranteed. Peer ratings are time-stamped and signed with the TTP system's SSL private key before being issued to the peer in a Kerberized ticket along with other data.

3 System Architecture

3.1 Assumptions

The design of the system assumes the following.

- The number of peers in the underlying P2P network is large.
- The public key of the TTP system is published to all peers in a certificate signed by a trusted Certificate Authority.
- Peers and TTP servers are all capable of establishing SSL sessions and connections.

- Peers and TTP servers are all capable of performing symmetric key cryptography, such as AES, DES, 3DES, RC4, and so on.

3.2 Overview

The TTP system is a single entity that all peers in the P2P network can see and trust. It implements the Kerberos Authentication Protocol, a proven network-based authentication system [6]. The TTP system itself is analogous to the KDC in the Kerberos authentication protocol and hence it uses a Kerberized ticket scheme to authenticate peers and to establish confidentiality in transactions between peers. Any peer in the P2P network who wants to participate needs to register with the TTP system to form a shared symmetric key, K_{peer} = hash(peer password), with the TTP system. This secret key is shared only between the peer and the TTP system and is used to encrypt and decrypt messages between them.

The TTP system has a public/private key pair that is used to establish an SSL session with peers. Before a peer starts communicating with the TTP system, it uses the TTP system's certificate, $Cert_{TTP,}$ which contains the TTP system's public key and is signed by a Certificate Authority, to start up an SSL session as described in Figure 1. The peer uses this public key SSL protocol to authenticate the TTP system server. All subsequent messages between the peer and the TTP system are encrypted with the symmetric session key, K_{SSL}, which is generated by the SSL session. The peer can use K_{SSL} to open cheaper subsequent SSL connections later. After an SSL session or connection has been established, the TTP system's Kerberos ticket protocol takes over for the rest of the authentication. Thus, all Kerberized tickets and messages between the TTP system and the peer are wrapped within a secure SSL session or connection.

The Kerberized ticket scheme provides many advantages to the TTP system. It allows the TTP system to remain stateless because peer session information need not be stored on the TTP server but instead can be stored in the peer's TGT; the total number of messages is minimized by using timestamp in tickets[2]; and only N symmetric keys are needed for N peers in the P2P network.

The scalability of the TTP system is an important concern since the number of peers can grow to millions in a P2P network. Therefore, the TTP system is designed internally as a hierarchy of TTP servers consisting of Local TTP servers (LTTP) at the bottom level of the TTP tree and Intermediate TTP servers (ITTP) at the upper levels of the TTP tree. Each LTTP manages a region of P2P network and each ITTP in turn manages LTTPs and other ITTPs. Only LTTPs are exposed to the outside world while ITTPs do not have any links to the outside world. This hierarchical TTP design makes it easy for the TTP system to accommodate new peers as the network grows. When the capacities of all LTTPs are reached, the TTP system only needs to add a new LTTP server to the existing TTP server tree and assign new peers to it. This process is entirely

[2] Timestamp in tickets are set by the TTP system's servers, which clocks are synchronized with each other within the TTP system, as required in Kerberos.

transparent to peers. Furthermore, peers assigned to an LTTP server can be moved to another LTTP server if the LTTP server is shut down for maintenance or repair.

$\mathbf{K_{ITTP/LTTP}}$	"master"/KDC symmetric key of ITTP or LTTP server.
$\mathbf{K_A}$	private key of peer Alice, K_A = hash(Alice's password).
$\mathbf{S_A}$	symmetric session key of peer Alice and her LTTP server.
$\mathbf{TGT_A}$	Ticket Granting Ticket of user Alice issued by her LTTP server.
$\mathbf{\{ M \}_{TTP}}$	encrypted with TTP system's SSL public key.
$\mathbf{[M]_{TTP}}$	signed with TTP system's SSL private key.
$\mathbf{Cert_{TTP}}$	TTP system's SSL certificate issued by a Certificate Authority.
$\mathbf{E(m_1, m_2,\ldots; K)}$	encrypt data with key K using symmetric cryptography.
$\mathbf{D(m_1, m_2,\ldots; K)}$	decrypt data with key K using symmetric cryptography.
$\mathbf{K_{AB}}$	secret symmetric key of peer Alice and Bob.

Table 1. Notation

3.3 Hierarchy of TTP Servers

The TTP system consists of a hierarchy of TTP servers in which LTTP servers reside at the bottom and ITTP servers reside at the upper levels. The hierarchical architecture of TTP servers addresses the scalability issue to better accommodate future growth of the network. This model allows the TTP system to dynamically expand with the growth of the P2P network by adding more LTTPs and ITTPs as needed. Moreover, it permits flexibility in balancing the workload of the LTTP servers.

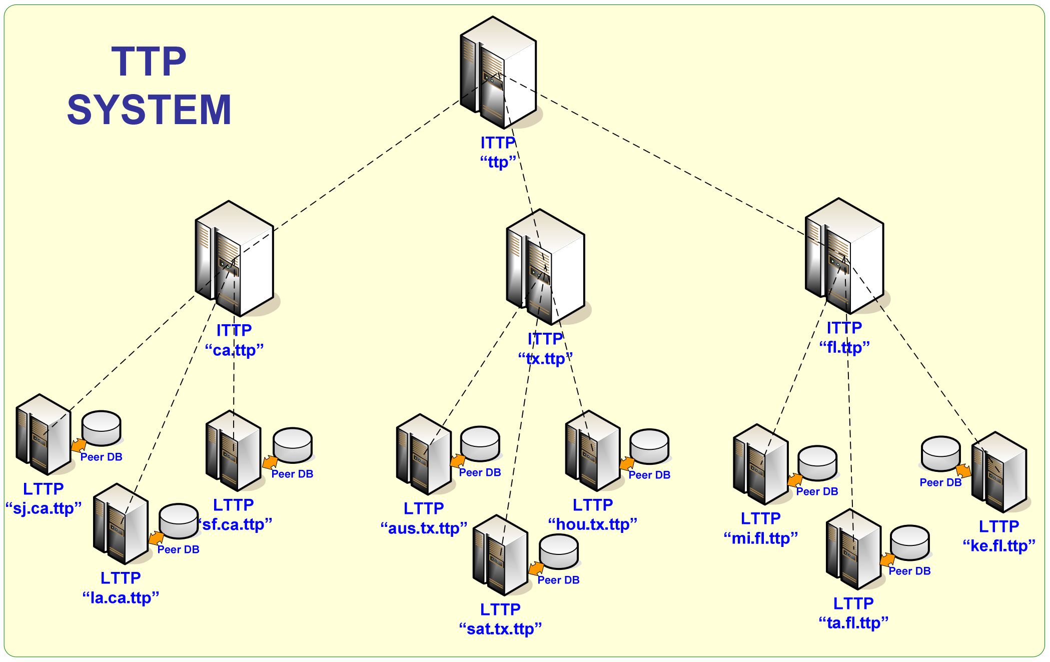


Figure 4. Hierarchy of TTP Servers

The TTP server naming follows the Domain Name standard: names are restricted to ASCII characters [a – z], [A – Z], digits [0 – 9], and the hyphen. Each TTP server name is a combination of its parent TTP server name and its own name separated by '.' characters. Each server name must be unique among its sibling servers of the same parent server. Although this naming scheme reveals little information about the TTP system structure, it does not make the system less secure because only LTTP servers are exposed to the outside world while all ITTP servers are isolated from it. However, any other naming standard for TTP servers could be used in the actual implementation if this was a concern. The X.500 naming standard, which does not use a hierarchical naming format, is one alternative [17].

All servers in the TTP system share a public/private key pair used in establishing an SSL session and connection with peers. A certificate that includes the public key issued by a third-party Certificate Authority is published to all peers. Using this certificate, a peer is able to authenticate its assigned LTTP server and open an SSL client session to this server. After an SSL session or connection has been established, subsequent communication between a peer and the LTTP server is done through Kerberos ticket-based messages within this secure SSL channel by encrypting all Kerberized messages with the SSL session symmetric key as in Figure 1. The private key on the other hand is used to sign peer ratings to protect its integrity when it is sent to peers.

Although all Kerberos messages between the LTTP server and peers are encrypted with an SSL session key, the communication is not necessarily more secure because multiple layers of encryption do not mean better security automatically. The security of this authentication system relies primarily on the ticket-based Kerberos authentication method. The SSL protocol is used merely for authenticating the LTTP server. It serves no further purpose once the Kerberos authentication takes over the remainder of the peer authentication.

If for any reason a peer's user identification and password are compromised, an attacker could use the stolen identity to establish an SSL session with the TTP system and obtain needed Kerberos tickets for use without the knowledge of the TTP system. Of course, this problem exists in many authentication systems. There are, however, ways to limit the chance of such a breach of security, such as requiring peers to change their passwords periodically[3].

3.3.1 LTTP Server

An LTTP server is analogous to a KDC in the Kerberos system because it serves as both authentication server and ticket granting server. Each LTTP owns a "master" KDC key, $\mathbf{K_{LTTP}}$, which is used to encrypt all TGT tickets sent to peers. This "master" key is shared with its immediate parent ITTP server and used to encrypt all internal messages between them. It would also be possible to use a separate symmetric key for

[3] Passwords are regarded as a relatively insecure method of access control because of many inherent weaknesses [8]; however, passwords are still widely used because they are cheap. In fact, they are cost-free.

communications between LTTP and ITTP servers, but that would cause an increase in the workload of LTTP servers. This approach is not employed in the system by default but is left as an option in the implementation.

Each LTTP server manages a group of assigned peers. When a peer joins in the TTP system, he is assigned to a particular LTTP server, which becomes the sole server that he communicates with in all TTP system operations from that point onward. The LTTP server can be chosen randomly or based on the geographical region of the peer. A random assignment is used by default because it helps to balance the number of peers among all LTTP servers.

Each LTTP server owns a database that it uses to store its peers' data. After a peer registers with the LTTP server using a unique name and password, a symmetric key, $\mathbf{K_A}$, which is the hash of the peer's password, is shared between the LTTP server and the peer. A peer profile including peer name, hashed password, and rating, along with other data, is stored in the LTTP database. The peer's rating is stored in the LTTP for integrity purposes and only sent to the peer at login time.

Since LTTP servers are the only ones in the system open to the outside world, they are the most vulnerable to attack. Using a separate peer database for each LTTP server minimizes the damage of a compromised LTTP server. If that does happen, only the compromised LTTP server needs to be disconnected from the P2P network and removed from the TTP hierarchy. Its peer profiles can be recovered from its database and transferred to a new LTTP server or to other existing LTTP servers. The new LTTP server to which peers are assigned automatically updates the symmetric keys of the affected peers, by requiring the peers to change their passwords.

In contrast to the Kerberos authentication system, cross-LTTP communication is disallowed in the TTP system. All cross-LTTP communication must go through a mutual ITTP ancestor. For example, in Figure 4 messages between LTTP servers "sj.ca.ttp" and "la.ca.ttp" must be done through their mutual ITTP parent "ca.ttp," and messages between LTTP servers "sf.ca.ttp" and "mi.fl.ttp" must be done through their mutual ITTP server "ttp." This restriction is enforced by the following requirements.

- All messages between TTP servers must be encrypted with their shared symmetric key.
- An TTP server can only send and receive internal messages to and from its direct parent or child TTP server.

These restrictions produce a clearer and simpler stream of internal communications among servers and, more importantly, better security for the entire system. By preventing LTTP servers from communicating directly with each other, the system reduces the chance of having a compromised LTTP server cause another LTTP server to be compromised. To break into multiple LTTP servers using a compromised LTTP server, the attacker must break into the parent ITTP server. This is presumably more difficult because all ITTP servers are disconnected from the outside world and only communicate with their child LTTP servers for a very limited set of operations.

3.3.2 ITTP Server

An ITTP server manages other TTP servers including ITTP servers and LTTP servers. Its jobs include monitoring its child TTP servers and relaying communications between LTTP servers. The later job is essential to communications between LTTP servers as described above.

Similarly, an ITTP server owns a "master" KDC key, $\mathbf{K_{ITTP}}$, which it shares with its immediate parent server. The root ITTP server is the only exception—it does not have a "master" key since it has no parent. An ITTP server uses this symmetric key to encrypt and decrypt messages to and from its parent server. It also shares a number of symmetric keys with its immediate child servers and uses those keys to encrypt and decrypt messages to and from its child servers.

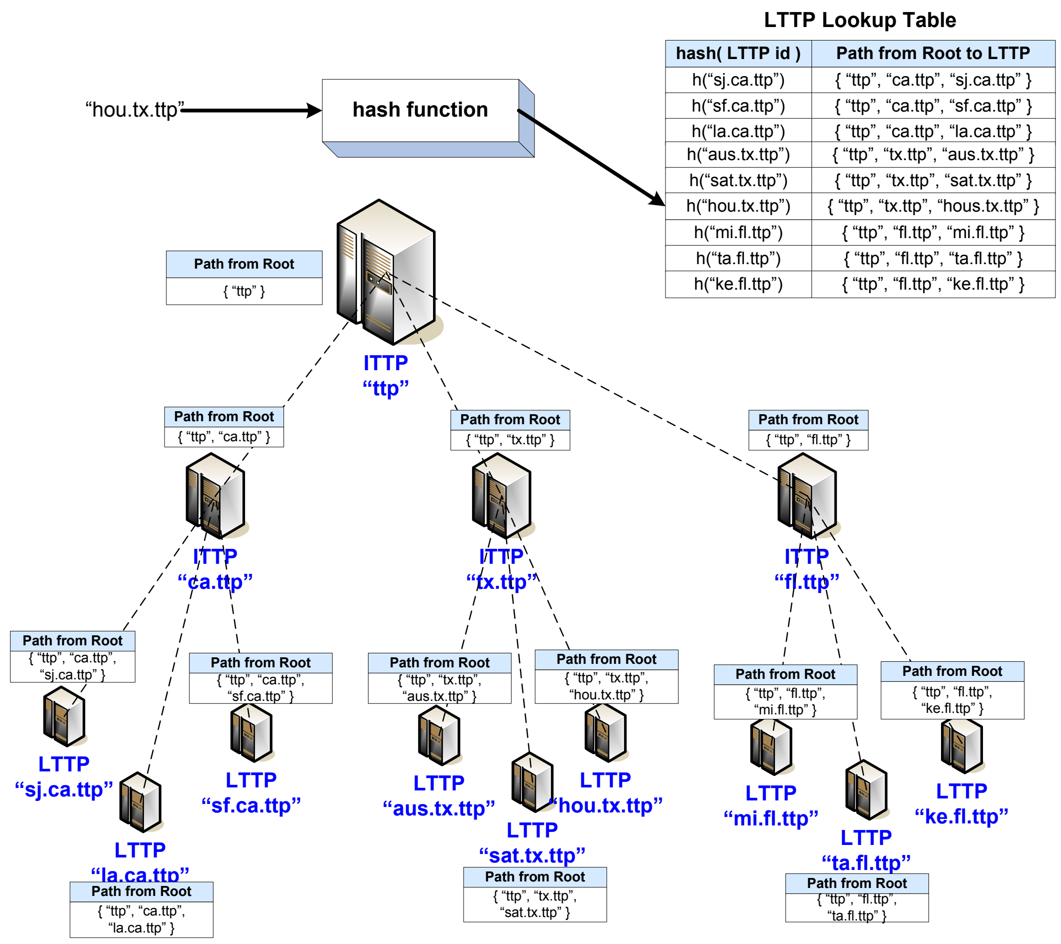
As discussed above, cross-LTTP communications are not allowed. Therefore, inter-LTTP messages must be relayed by ITTP servers that lie in the path from one LTTP server to the other. For example, in Figure 4 messages between the LTTP servers "sj.ca.ttp" and "la.ca.ttp" must be relayed through their mutual ITTP parent "ca.ttp"; messages between LTTP servers "sf.ca.ttp" and "mi.fl.ttp" must be relayed through ITTP servers "ca.ttp," "ttp," and "fl.ttp."

To efficiently relay inter-LTTP messages, each TTP server maintains the path from the root server to itself, which is encoded by the concatenation of its own name with its parent server path. All LTTP servers share one LTTP lookup table that has two columns: the hash of the LTTP server name and its path from the root server as described in Figure 5. Whenever a new LTTP server is added to the hierarchy, a new row is added to the table. Here are the steps to find the path from one LTTP server to another[4]:

- Get the paths from root to the two LTTP servers in the lookup table by taking the hash of the two LTTP names, which hash to the appropriate row indices containing the paths.
- Find the mutual ancestor server from these paths.

The path between them is the path from the first LTTP server to the mutual ancestor server to the second LTTP server. For example, the paths from the root server to LTTP servers "sf.ca.ttp" and "mi.fl.ttp" are {"ttp," "ca.ttp," "sf.ca.ttp"} and {"ttp," "fl.ttp," "mi.fl.ca.ttp"} respectively. Hence, their mutual ancestor server is server "ttp," and the path from "sf.ca.ttp" to "mi.fl.ttp" is {"sf.ca.ttp," "ca.ttp," "ttp," "fl.ttp," "mi.fl.ca.ttp"}.

[4] This path-finding method is a generic method that works on any naming standard. If the Domain Name standard is used for server names, then the path can be found directly from the names of the two LTTP servers.



LTTP Lookup Table

hash(LTTP id)	Path from Root to LTTP
h(“sj.ca.ttp”)	{ “ttp”, “ca.ttp”, “sj.ca.ttp” }
h(“sf.ca.ttp”)	{ “ttp”, “ca.ttp”, “sf.ca.ttp” }
h(“la.ca.ttp”)	{ “ttp”, “ca.ttp”, “la.ca.ttp” }
h(“aus.tx.ttp”)	{ “ttp”, “tx.ttp”, “aus.tx.ttp” }
h(“sat.tx.ttp”)	{ “ttp”, “tx.ttp”, “sat.tx.ttp” }
h(“hou.tx.ttp”)	{ “ttp”, “tx.ttp”, “hous.tx.ttp” }
h(“mi.fl.ttp”)	{ “ttp”, “fl.ttp”, “mi.fl.ttp” }
h(“ta.fl.ttp”)	{ “ttp”, “fl.ttp”, “ta.fl.ttp” }
h(“ke.fl.ttp”)	{ “ttp”, “fl.ttp”, “ke.fl.ttp” }

Figure 5. TTP Server Paths and the LTTP Lookup Table

Relaying messages from one LTTP server to another is easily accomplished at every TTP server through the following steps.

- Find the path from one LTTP server to the other as described earlier.
- Starting from the first LTTP server, remove it from the path and relay this message along with the shortened path to the next server in the path, which should be either the immediate parent server or the immediate child server.
- Recursively repeat the above step until the path is empty. The designation LTTP server is the last server in the list.

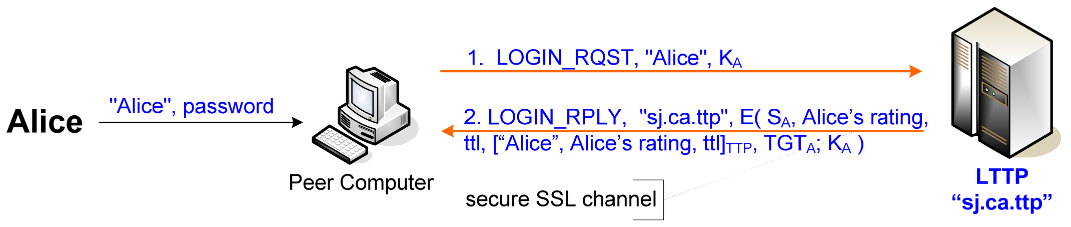
It is essential that an ITTP server be excluded from all connections to the outside world. It should only be inter-connected with its immediate parent server and its immediate child servers. This restriction is vital to the security of the whole system since these ITTP servers are supposed to be hidden and protected behind LTTP servers.

3.4 Peer Authentication

The TTP system is designed as an independent trusted-third-party model so that peers need only to join when they want secure transactions. Peers can operate in a P2P network with insecure transactions as usual without having to do anything with the TTP system. When a peer is ready to join, he can register with the system using a unique name and password. The TTP system reserves a default LTTP server for peer registration. This reserved LTTP server can determine which LTTP server this peer should be assigned to according to the peer's geographical location, organization, the current workload of LTTP servers, or a random choice. Once an LTTP assignment is made, the registration LTTP server requests the assigned LTTP server to create the peer profile and reply to the peer with the following data upon success.

- The assigned LTTP server name and IP address.
- The peer's hashed password, $\boldsymbol{K_{peer}}$ = *hash(peer password)*, which is his private symmetric key for use with the assigned LTTP server.
- The hash function type used in hashing the password.

Note that all communications between peer and TTP servers are always carried out within the secure SSL channel that the peer has established with the TTP system. The peer computer only stores the hash function type and LTTP server data, which are used to hash the peer password and communicate with the LTTP server in the login process.



Alice
"Alice", password
Peer Computer
1. LOGIN_RQST, "Alice", K_A
2. LOGIN_RPLY, "sj.ca.ttp", E(S_A, Alice's rating, ttl, ["Alice", Alice's rating, ttl]$_{TTP}$, TGT_A; K_A)
secure SSL channel
LTTP
"sj.ca.ttp"

Figure 6. Alice logs in to the TTP system

When peer Alice desires a secure transaction, she logs in to her LTTP server using her user name and password as in Figure 6. Her peer computer uses the predefined hash function to compute the hashed password, K_{Alice}, to be sent to the server. It then authenticates and establishes an SSL session with the LTTP server and sends the login request with Alice's name and hashed password to the LTTP server. When receiving the LTTP server's encrypted reply, the peer computer uses K_{Alice} to decrypt the message, which contains the following.

- A session key, $\boldsymbol{S_{Alice}}$, generated randomly by the LTTP server.
- A Ticket Granting Ticket, $\boldsymbol{TGT_{Alice}}$ = *{ LTTP name, E("Alice," S_{Alice}, time-to-live; K_{LTTP}) }*, used to obtain other tickets.
- The signed and time-stamped peer rating, ***["Alice", Alice's rating, time-to-live]$_{TTP}$***.

These data are saved on the peer computer for use within the lifetime of Alice's TGT. TTP authentication system benefits from this Kerberos ticket-based scheme because it does not have to store the peer's session data while they are logged in. All session data are stored in the peer TGT itself, encrypted with the LTTP master key, and can be decrypted only by the LTTP server. This is an advantage in a P2P network since the number of peers may be very large. Storing all peers' data on the TTP servers would be inefficient and perhaps even impossible. These data are distributed among peers in a secure manner instead.

This login mechanism is vulnerable to one potential attack method. Trudy, an attacker, may try to guess Alice's password by logging in with Alice's user name and different passwords multiple times until succeeding. This attack is present in almost all authentication systems. Several methods are used in LTTP servers to prevent such an attack.

- Limit the number of logins to three. After three failed attempts, the peer account is suspended for some time. This, however, creates a window for Denial of Service attack.
- Delay the login reply from LTTP server for a short time. Three seconds is used in LTTP servers by default. This slows down an attack dramatically.

3.5 Establish Secure Channels between Peers

One of the main purposes of this system is to provide a mechanism for establishing secure transactions between peers. The method employed is based on the Kerberos ticket-based scheme. Peers use their TGTs to request a symmetric key, K_{AB}, which is issued only to the two involved peers. The symmetric key generated by the TTP system is sent to the requesting peer. In addition, a Kerberized ticket including the symmetric key encrypted with the other peer's session key is sent to the requesting peer who must then send it to the other peer. Therefore, only the two involved peers can get the secret key K_{AB}, which is used to form a secure channel between the two peers in which all messages are encrypted using K_{AB}.

Figure 7 shows a sequence of messages between two peers and between a peer and its LTTP server for obtaining a secret key, K_{AB}. This scenario assumes that both peers Alice and Bob have logged in to their respective LTTP servers and obtained TGT_{Alice} and TGT_{Bob}. First, peer Alice requests peer Bob to establish a secure channel that they can use for secure transactions. In her request message, Alice sends to Bob her time-stamped rating signed by the TTP system along with her TGT_{Alice}. If Bob does not trust Alice based on her rating, he can deny the request, and the interaction terminates. If Bob decides to trust Alice, he opens an SSL connection with his LTTP server and sends his TGT_{Bob} and Alice's TGT_{Alice} along with an encrypted timestamp[5] requesting the LTTP server for a shared secret key. If Alice belongs to the same LTTP server as Bob, then this server can decrypts peer Alice's TGT_{Alice} to verify her identity. Otherwise, Bob's LTTP server finds Alice's LTTP server, as described above in section 3.3.2, to request verification of Alice. When all peers are verified, Bob's LTTP server generates a random symmetric key, K_{AB}, enclosed in the following message and ticket.

- *"Message to Bob"*

[5] Timestamps in messages between peers or between a peer and the TTP servers are used to prevent replay attack.

- *"Ticket to Alice"* encrypted with S_{Alice}

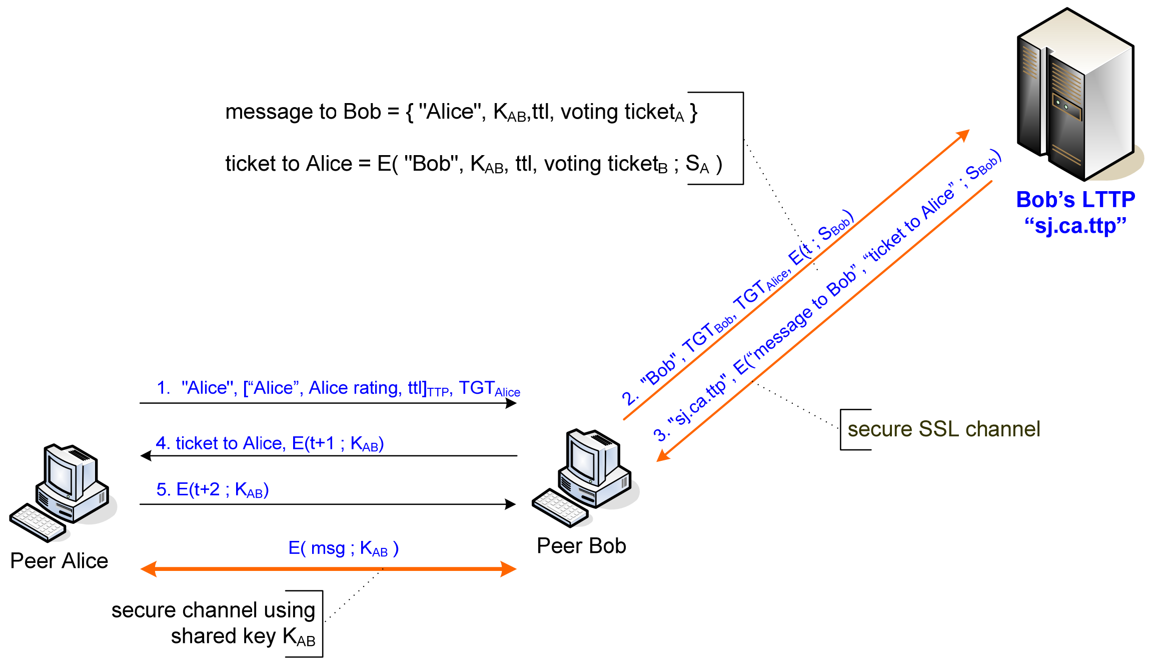

message to Bob = { "Alice", K_{AB},ttl, voting ticket$_A$ }
ticket to Alice = E("Bob", K_{AB}, ttl, voting ticket$_B$; S_A)
Bob's LTTP
"sj.ca.ttp"
2. "Bob", TGT$_{Bob}$, TGT$_{Alice}$, E(t ; S_{Bob})
3. "sj.ca.ttp", E("message to Bob", "ticket to Alice" ; S_{Bob})
secure SSL channel
1. "Alice", ["Alice", Alice rating, ttl]$_{TTP}$, TGT$_{Alice}$
4. ticket to Alice, E(t+1 ; K_{AB})
5. E(t+2 ; K_{AB})
Peer Alice
Peer Bob
E(msg ; K_{AB})
secure channel using
shared key K_{AB}

Figure 7. Establishing a Secure Channel Between Peer Alice and Peer Bob

Bob opens his message to get K_{AB} and a voting ticket used to vote on Alice later. He sends to Alice her ticket along with the current time incremented by 1, encrypted with K_{AB}. Alice decrypts the ticket to get K_{AB} and a voting ticket used to vote on Bob later. After she uses K_{AB} to verify Bob's timestamp, she sends back an acknowledgement of the time incremented by 2 encrypted with K_{AB} which Bob can also verify using K_{AB}. Timestamps are used for mutual authentication between peers. The purpose of incrementing timestamps here is to prevent replay attack. After this, they can start using the secure channel by encrypting all messages with the secret key K_{AB}. This secret key K_{AB} is valid for a limited time, Time-To-Live (TTL), specified by Bob's LTTP server.

Note that the two voting tickets generated by the two peers' LTTP server(s) are encrypted with their respective master key(s) and issued only to the two peers involved in the request. One is sent directly to Bob for voting on Alice. The other is contained in the "ticket to Alice" sent by Bob to Alice for voting on Bob. After their transactions are completed, the peers use these tickets to vote on each other by sending the tickets along with their votes to their respective LTTP servers. Only the correct LTTP server can decrypt the voting ticket and use it to verify the vote authenticity. If sent to the wrong LTTP, the vote will be discarded.

Messages 1, 4, and 5 are sent over an insecure channel between peers Alice and Bob. However, this does not make their communication less secure because these messages are all encrypted with either Alice's private key K_A or the shared secret key K_{AB} that only peers Alice and Bob can decrypt.

Timestamps are used throughout in messages between peers and between peers and the TTP system to avoid replay attack and reduce the number of messages needed. This approach presents a drawback since time becomes a security-critical parameter and requires clock synchronization among all involved parties [8]. There are two options to overcome this problem.

- Synchronize the peer software's clock with TTP server's clock when the peer logs in and use this clock to produce timestamps in messages instead of peer computer clock. TTP servers' clocks are internally synchronized with each other so that LTTP servers' peer software clocks are also synchronized indirectly with each other. This is relatively easy to accomplish since TTP servers are closely coupled. A small clock skew of one minute is accepted in the system by default since peer software's clocks can deteriorate over long periods.

- Instead of synchronizing clocks, TTP servers memorize all timestamps received within a certain clock skew close to the current time and reject repeated timestamps [8]. This method, however, places much more workload on the TTP system than the previous approach.

The TTP system implements the first approach by default. However, the decision of choosing either approach is left to the actual implementation of the TTP system.

4 Peer Rating Scheme

Trust is a key factor in reputation-based systems. It is particularly important in a P2P network because peers remain anonymous to each other; a peer's reputation is essentially the only information that anyone can use to determine whether he can be trusted.

A reputation-based rating system is integrated into the TTP system. A peer rating is kept in the LTTP server and issued to the peer when he logs in, signed by the TTP system's SSL private key. To prove his trustworthiness, a peer sends his signed rating to other peers, who can decrypt it using the TTP system's public key. When the TTP system creates a secret key, K_{AB}, for two peers, it includes two voting tickets issued to the involved peers. Peers use these tickets to rate each other after completing their transactions. The followings are the rules that the TTP system uses in its peer reputation scheme.

- A rating scale of -5.0 to 5.0 is used.
- Ratings for each peer are kept in the peer's LTTP server and are signed and issued to the peer when he logs in.
- A peer rating is updated when another peer uses a voting ticket to vote on the peer.
- A peer can only vote on a particular peer once per voting ticket.

In the TTP system, as in any reputation system, a peer rating is built on his past reputation and is affected by the consequences of his interactions with other peers [4]. However, the reputation system in the TTP system does not compute peer ratings by merely averaging the summation of all votes on the peer. Using that method alone is a flawed approach. For example, a peer that has ten +5 votes in the last year but remains inactive in the current year and a peer that has one hundred +5 votes in the recent month both have the same +5 rating. This is obviously flawed because the second peer should be more trusted than the first peer.

The rating system in the TTP system therefore is designed to encourage actively good behavior in peers and discourage bad behavior or passive peers. In other words, a peer must actively maintain his good behavior in order to maintain a high rating. To better represent a peer rating in the system, a "weighted" rating formula is used to compute peer ratings from a number of factors, including direct rating, objective rating, subjective rating, and fading ratio:

$$peer\ rating = (\alpha * direct\_rating + \beta * obj\_rating + \gamma * subj\_rating) * fading\_ratio \quad (1)$$

, where α*,* β*, and* γ *are the percentages and* $\alpha + \beta + \gamma = 1$*.*

This rating is computed and given to the peer only when he logs in to the system. This approach enhances system efficiency because the system does not have to compute the value repeatedly. The percentages α, β, and γ are chosen to be 70%, 15%, and 15% respectively by default. However, these percentages should be fine-tuned in the actual implementation to create a suitable outcome.

4.1 Direct rating

The direct rating is the average of other peers' votes on a peer. Every time another peer votes on this peer, his direct rating is recalculated and updated in the LTTP server. This rating source should be the biggest percentage in the overall rating formula because it represents the most important factor, the votes/opinions of other peers in the P2P network.

However, using direct rating as a major percentage in the formula could also be the pitfall of this rating system. A group of peers in a P2P network can cooperate in bringing up their ratings by giving each other high votes. It is difficult to design a rating-tampering resistance system without increasing the workload of the system. Thus, peer reputation scheme in the TTP system is designed to compromise between rating-tampering resistance and system performance as follows.

- For a cluster of peers with the same IP address' net_id, take only an average of these peers' votes to prevent vote tampering of a clique of dummy or cooperating peers. This essentially collapses these votes from the same net_id to one single vote [2].
- Select a set of random votes from all votes in computing a peer's direct rating. LTTP servers randomly determine which votes to accept according to a predefined

probability. By default, 70% of votes are accepted. However, this should only be employed when there exists a large number of votes on a peer.

4.2 Objective rating

Peer rating is affected by a number of objective factors.

- Responsiveness: the time from when the voting ticket was issued to when this vote was received. The longer the duration, the less effect the vote has on the peer rating.
- Liveliness: how active the peer is in the system, as measured by the total number of secret channels this peer has owned and the total number of votes this peer has voted. The more he has, the better objective rating he possesses.
- Credibility: take into account the voter's own reputation on his vote. For example, if Alice's rating is +4, then her vote on Bob has only 90% of the effect of a perfect vote. In other words, we do not trust a voter more than other peers trust him.

These factors are based on objective information that LTTP servers collect and store in their databases. They do not overburden TTP servers and have minimal effect on system performance. These are the objective factors used in the TTP system by default. However, any other objective factors can be added in the actual implementation.

4.3 Subjective rating

Peer rating is also affected by a number of subjective factors that are based on context-aware tests [1].

- Suspicion detection: A peer’s rating history plays an important role in determining this subjective rating. For example, if a peer rating is 0/neutral for months but suddenly surges up to +5 in a few days, then his subjective rating should be low since this activity is suspicious.
- System behavior: This is determined by peer interactions with TTP servers. This score is low if the peer fails to log in many times or has tried to use invalid Kerberized tickets frequently in the last few days.

To determine peer subjective ratings, the system must monitor the history of peer behavior, which increases the workload of the system. Therefore, a balanced number of subjective tests must be chosen in the system to determine peer subjective ratings without affecting system performance too much. The impact on system performance of each subjective test must be carefully considered. The TTP system chooses to include the above two tests by default. However, any new test can be added to the actual implementation of the system.

4.4 Reputation Fading

The fading of reputation reflects a social norm in which a person's trustworthiness decreases slowly as time passes. A person's reputation is not the same today as a year ago if he remains inactive the whole time. In other word, he must always work actively to maintain his good reputation; otherwise, his reputation fades. On the other hand, if he has had a bad reputation in the past, that aspect of his reputation also fades over time.

In the TTP system, this fading factor is implemented by discounting the peer rating by a small percentage periodically [1]. This prevents peers from capitalizing on past good behavior and provides the possibility of redemption from previous bad behavior [1]. A peer rating slowly fades back to the "neutral" state (rating = 0.0); each day it is multiplied by a fading factor of 0.975, for a 0.025% daily discount as in formula 1. This fading factor has the effect of decreasing a +5.0 rating or increasing a -5.0 rating back to a neutral/0.0 rating over a 200-day period.

Many algorithms for fading factors can be incorporated into the system to improve the outcome. One is to have different fading ratios for positive and negative ratings so that it takes longer to fade back to the neutral state from a negative rating than a positive one. Another is to increase or decrease the fading factor according to the peer's behavior over time. For example, for second-chance misbehavior, a peer can get back his reputation slowly by good behavior over time. However, following any future misbehavior reflected in negative votes of other peers, his reputation will fade even faster. This fading method is very effective for the case in which a peer intentionally maintains good behavior to get a good rating over a long period and then uses it for bad behavior over a short period before discontinuing the account. Even though these algorithms are not implemented in the TTP system by default due to their load on TTP servers, they can be incorporated in an actual implementation.

5 Security and Performance Analysis

5.1 System Architecture and Protocol Issues

Although the TTP authentication system is based on the proven security protocols SSL and Kerberos, as in any other authentication system it has a number of outstanding issues. Most of them are inherited directly from the SSL and Kerberos architectures and have been discussed in many publications. The following are some of the more prominent issues with the TTP system.

- **Single Point of Failure**

Since the TTP system is based on a trusted third-party model, it rests on a single entity that is critical to the availability of the whole system. The system requires continuous availability of its central servers. If an LTTP server is down, no peer associated with this server can log in or use its services. Solutions for this issue follow.

- Keep a backup/mirror LTTP server for each LTTP and switch to the backup server automatically when it goes down.
- Delegate the LTTP server's peers to another LTTP server in the TTP server hierarchy. However, this should be implemented carefully so that it does not offset the workload balance among the servers.

- **Bottleneck Effect**

Since all inter-LTTP communications are relayed through ITTP servers along the path from one LTTP server to the other, the system contains a potential bottleneck that may slow down its performance when there is a large number of such communications. If the TTP hierarchical tree depth is d, the average number of involved ITTP servers in inter-LTTP communication is (d – 1)/2. Thus, to maintain system performance, the hierarchy tree depth should be minimized so that the number of ITTP servers in inter-LTTP communication is small.

- **Replay Attack**

As in the Kerberos authentication system, a timestamp is used in TTP system messages to cut down on the number of messages required [8]. If the TTP system is implemented exactly akin to Kerberos servers, so that the clocks of peer computers and TTP servers are not synchronized, there must be an allowance for time skew between peer message timestamps and the TTP server's clock. This means that any message with a timestamp close to the allowable time skew of the server's clock is accepted. This opens up a small window for a replay attack using a stolen live authenticator.

The TTP system limits this replay attack by requiring the peer software's clock to synchronize with the LTTP server's clock whenever the peer logs in, as described in section 3.5. However, since even the peer software's clock may be out of sync with the server's over a long period, a time skew of one minute is still used by default. Thus, the chance of such an attack is small but still exists. Another approach is to remember timestamps within the clock skew and refuse to accept duplicate timestamps. This eliminates the replay attack, but puts an additional burden on the TTP servers. Therefore, it is a tradeoff.

- **Password Guessing Attack**

The TTP system is vulnerable to a password-guessing attack because most users do not use "strong" passwords. This issue exists in almost all authentication systems that use a password for login authentication. A peer's password could be discovered by a dictionary attack, in which the password is forward-searched by trying words from a dictionary of commonly used passwords [8]. To prevent this attack, the system can lock down the account after a certain number of failed attempts, often three. However, the question remains: how long should the account be locked down? If the lock-down time is too short, it will have minimal affect on the attack. But if it is too long, it opens up a window for a Denial of Service attack in which the attacker can intentionally use an incorrect password to deny service to a legitimate peer [8].

- **Login Spoofing**

A peer computer uses the TTP client software to handle all peer operations with the system. Because personal computers and workstations are in general more vulnerable to attacks, it is very likely that this TTP client software can be hacked to record peer passwords whenever he enters a password to the peer computer [10].

A standard countermeasure used in the TTP login design is to use a challenge/response dialog instead of a hashed password [10]. This method requires the LTTP server and the peer computer to share a predefined secret key, K_s. When receiving a peer login request, an LTTP server first sends a random number, R, to the peer computer, which then uses the predefined secret key to encrypt the received random number, $E(R; K_s)$, and sends it back to the server. If the server's encrypted value matches the peer computer's, then the server allows the peer to be logged in. The Diffie-Hellman key exchange protocol can be used to establish this symmetric secret key, K_s, between the LTTP server and the peer computer [8].

Another countermeasure is to use a smartcard for authentication of peer identity. This method is attractive because it implements relatively better security than passwords. A smartcard's embedded chip usually implements a cryptographic algorithm. If used in combination with biometrics such as fingerprints, smartcards can provide two- or three-factor authentication [16]. However, it may not be practical at this time to require all peers in a P2P network to have a smartcard due to its high cost.

- **Peer Data Storage**

Peer symmetric keys and other information are stored in a database associated with the LTTP server assigned to the peer. It is tempting to store these keys on the LTTP server or to keep this database on the same server computer. This architecture may be convenient but not a good idea since these peers' information will be unavailable when the LTTP server is down. Therefore, it is better to keep the peer database on a separate database server connected to the LTTP server through a dedicated line. However, this could be a weak link if the database is not on the LTTP server but is connected through a multiple-purpose network that may create a security risk if it also connects with the outside world. The following methods can be used to avoid this security risk.

  - Each LTTP server is associated with a separate peer database server through a dedicated secure link that is not used for any other purpose. In case the server goes down, a new LTTP server can easily replace it by connecting to its database server, or its peers' profiles can be transferred to other LTTP servers' databases.
  - All messages between the database server and an LTTP must be encrypted with a predefined symmetric key.

    - An LTTP server downloads peer information from this database only to its "volatile"/RAM memory as needed [10].

- **Peer Repudiation**

Since an LTTP server does not store the secret key K_{AB} given to the two peers to establish a secure channel, peers can repudiate this transaction later, asserting that it has never taken place. This is a problem or a feature depending on the perspective taken. It is a desirable feature for peers who want to maintain anonymity as well as transaction secrecy with others in the P2P network.

5.2 Rating Scheme Issues

Rating of peers is based mainly on other peers' opinions. The rating scheme relies on a P2P peer community to sort out over time the proper peer ratings. This scheme is not perfect and presents several issues, as do most reputation systems.

- **Pseudospoofing**

A malicious peer may exploit anonymity in the system by using multiple phony identities [2]. Such a peer often turns to a new identity after earning a bad reputation with the old one. However, creating a clique of phony identities using the same IP address is avoided in the TTP system because votes are taken only from the average of all votes coming from the same net_id as described in section 4.1.

- **Shilling**

This attack method is well known in auction-based protocols. A malicious bidder uses multiples identities to push up the bidding price of his auctioned product [2]. It is different from pseudospoofing in that multiple identities, called shills, are created from different IP addresses [2], or a group of legitimate peers cooperates in manipulating the system. In the TTP system, shilling can be used by a group of cooperating peers to push up their ratings over time. A number of mechanisms in the TTP system are employed to deal with this issue. They include selecting a set of random votes from all votes, as described in section 4.1, and using reputation fading to fade away the peer's rating over time, as described in section 4.4.

6 Software Simulation

6.1 Software Design

In order to demonstrate the TTP authentication system's correctness and effectiveness, a software simulation of the TTP system in a P2P network has been developed. The simulation comprises two main software modules: a module simulating the TTP system and a module simulating a P2P network of peers. These two modules run independently and communicate with each other through secure SSL connections. This simulation design allows us to monitor several aspects of the system: the authentication process between peers and the TTP system, the Kerberized ticket scheme between peers and the TTP system, and the rating scheme. Each module has a Graphical User Interface to allow us adjust the simulation data, control the simulation process, and run particular simulation tests.

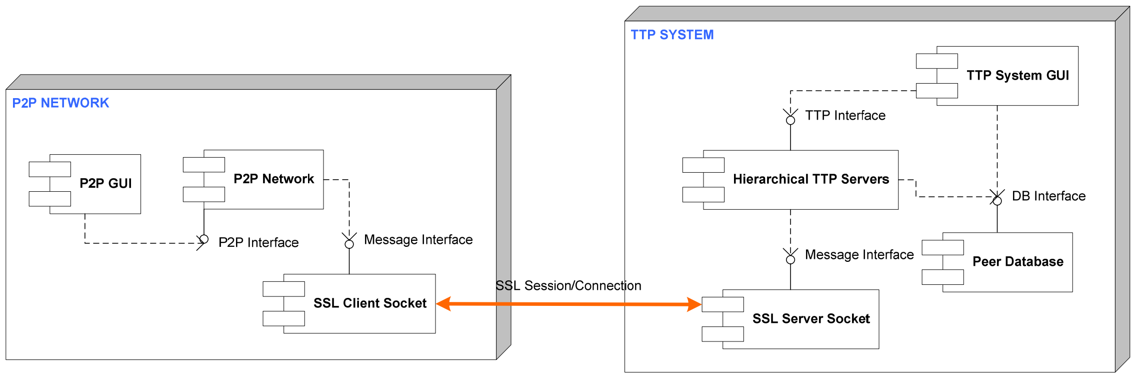


Figure 8. Software Simulation Architecture

In Figure 8 above, the simulation module on the right, which simulates the TTP system, includes four components: a Hierarchy of Kerberized TTP Servers, a Peer Database, an SSL Server Socket, and a Graphical User Interface. The first component represents a hierarchy of TTP servers, comprising LTTP servers on the bottom level and ITTP servers on the upper levels. Each LTTP server manages a number of peers from the simulated P2P network, and each ITTP server in turn manages LTTPs and other ITTPs. As in the Kerberos KDC server, the LTTP server issues Kerberized tickets to peers to be used for subsequent operations. The Peer Database component provides an interface to a database storing peer data and system data. TTP servers use this interface to access and query the database regarding peer data. A single database that stores all peer data in this simulation is shared among all LTTP servers instead of having a separate database for each LTTP server. This configuration was deemed sufficient for the simulation. The SSL Server Socket component simulates an SSL server socket for receiving requests from peers to establish secure SSL sessions/connections with the system.

The P2P Network module, which represents the entire P2P network of peers, includes three components: a P2P Network of peers, an SSL Client Socket, and a Graphical User Interface. The P2P Network component manages all peers in the network, which includes creating, deleting, and changing peer data. The SSL Client Socket component in the P2P network module simulates an SSL client socket issuing requests to establish secure SSL sessions/connections with the system.

6.2 Software Implementation

6.2.1 SSL Server/Client Socket Implementation

SSL server and client sockets handle secure communications between a peer and its LTTP server using the SSL public key protocol. Its implementation is based on Java Secure Socket Extension (JSSE) technology, in which Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols are designed to authenticate and protect data transfer across networks [12].

A Java SSL server socket and client socket framework can be implemented using a variety of public key cryptographies. In our simulation, RSA with an MD5-signature algorithm is chosen to run internally in the public key cryptography of SSL server and client sockets. The Java key and certificate management tool, keytool, is used in managing public keys and certificates by creating one key store that stores a public-key pair on the TTP system side and one key store that stores a self-signed certificate on the P2P network side. A pair of public/private keys and a SunX.509 self-signed certificate are generated by the tool on the TTP system side with a 512-bit-RSA key size [12]. The certificate containing the RSA public key is imported into the P2P network's key store and used in opening an SSL secure channel with the LTTP server. Note that a Certificate Authority, usually needed for signing the certificate, is intentionally omitted in this public key protocol because it is deemed unnecessary for this software simulation.

6.2.2 Peer Database

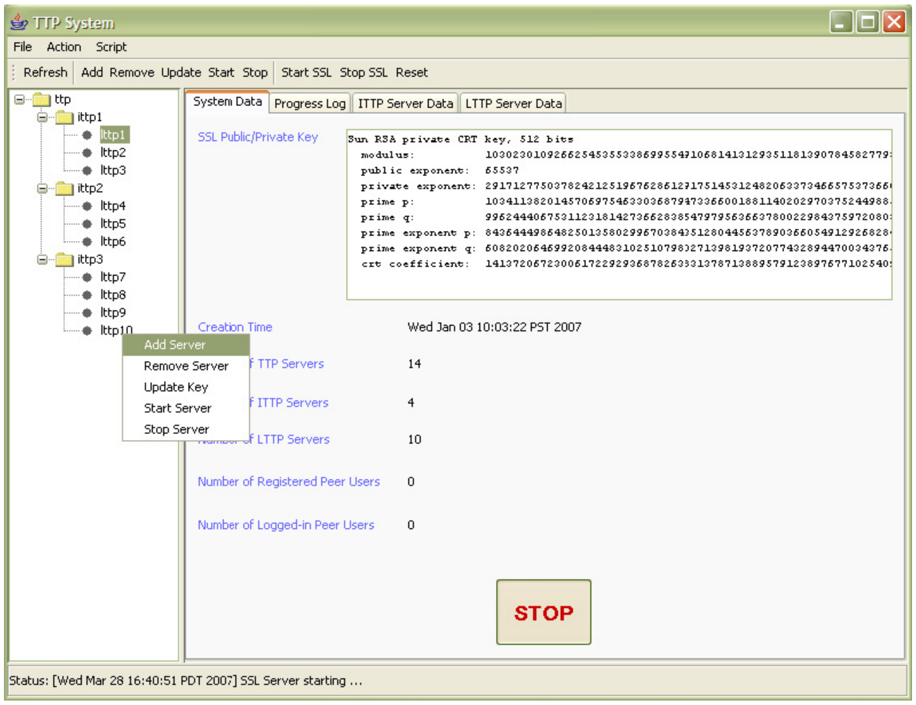
According to the TTP architecture, each LTTP server is associated with a database that stores its assigned peers' data. To simplify the software simulation, only a single database is used to store all peers' data shared among all LTTP servers. A MySQL database is chosen to implement the database in this simulation [13]. Two database tables, one storing TTP server data and one storing peer data, are created in the database. LTTP servers access these tables to keep a few essential peer data such as peer name, hashed password, rating, and so on.

The Java Database Connectivity (JDBC) framework is used to integrate Java software code with database SQL code [14]. A JDBC technology-based driver is imported into the software project to provide connectivity with the MySQL database. When the TTP system module boots up, it first loads the driver using Java Driver Manager and then establishes a connection with the database using a JDBC API. Once the database connection is opened, LTTP servers can begin to access the database's tables using a set of JDBC APIs.

6.2.3 TTP System Implementation

A tree data structure is used to represent the hierarchy of TTP servers. Bottom nodes represent LTTP servers and upper-level nodes represent ITTP servers. Each LTTP server is implemented similar to a KDC server in the Kerberos authentication system. Tickets are generated in response to peer requests using a set of Kerberos APIs and are encrypted using the LTTP server's "master" key, K_{LTTP}, the peer's session key, S_{Alice}, or the peer's private key, K_{Alice}.

The 3DES algorithm is the symmetric key cipher used in encrypting and decrypting Kerberized tickets with 168-bit symmetric key size. This symmetric key size, however, is actually 24 bytes in Java Cryptography Extension (JCE) with the last three bytes duplicating the first three bytes [15]. The 24-byte 3DES symmetric keys are generated for TTP servers' "master" keys and peers' session keys.



SSL Public/Private Key

```
Sun RSA private CRT key, 512 bits
  modulus:            103023010926625453553386995547106814131293511813907845827779
  public exponent:    65537
  private exponent:   29171277503782421251967628612917514531248206337346657537366
  prime p:            10341138201457069754633036879473366001881140202970375244988
  prime q:            99624440675311231814273662838547979563663780022984375972080
  prime exponent p:   84364449864825013580299670384351280445637890366054912926828
  prime exponent q:   60820206469920844483102510798327139819372077432894470034376
  crt coefficient:    14137206723006172292936878263313787138895791238976771025405
```

Creation Time Wed Jan 03 10:03:22 PST 2007

f TTP Servers 14

f ITTP Servers 4

f LTTP Servers 10

Number of Registered Peer Users 0

Number of Logged-in Peer Users 0

Status: [Wed Mar 28 16:40:51 PDT 2007] SSL Server starting ...

Figure 9. The TTP system GUI

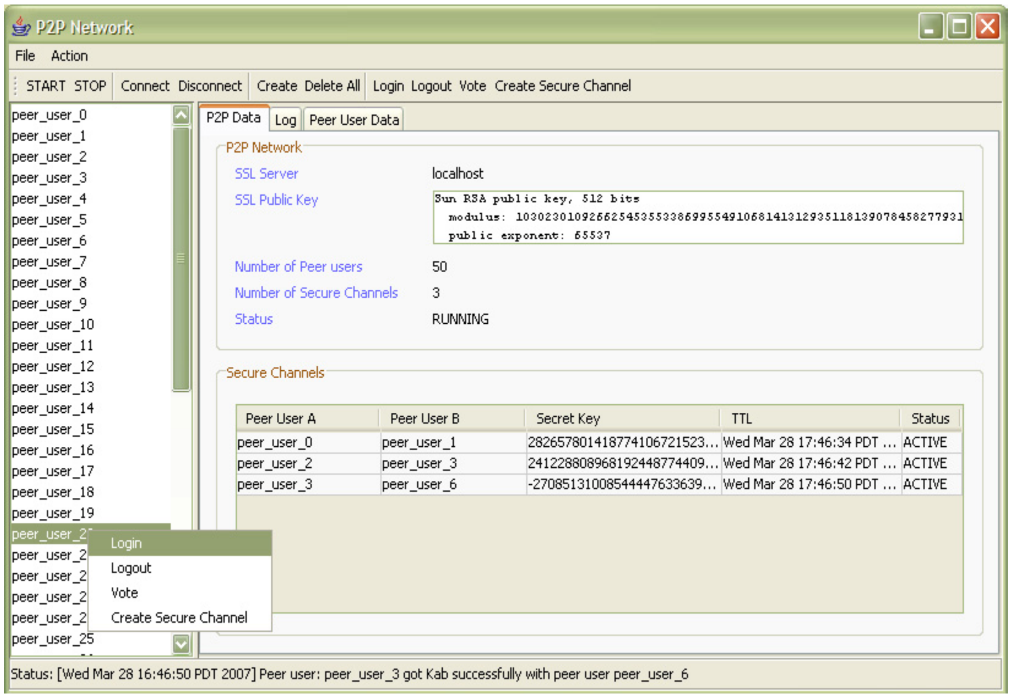
The TTP system module includes a Graphical User Interface (GUI) to monitor and control the system. The GUI allows us to control and operate the TTP server hierarchy, including such actions as adding new servers, removing servers, running XML scripts to create multiple servers, and so on. The GUI displays all system data and TTP server data, including the "master" key, SSL public and private keys, and peer-hashed keys, in real time to help us debug and understand TTP system interaction with peers.

6.2.4 P2P Network Implementation

Instead of simulating each peer separately, a network of P2P peers is simulated as a whole to simplify the simulation. The simulated P2P network contains a list of peers, each of which can individually initiate interactions with the TTP system or with each other. Every time a new peer is created in the P2P module, he will attempt to register to the TTP system. When receiving a registration request, the TTP system module assigns him to a random LTTP server with which he will communicate exclusively from this point on.

The peer password is hashed by a SHA-1 hash function from the Java Cryptography Extension (JCE) in both the TTP system and the P2P network modules [15]. This 20-byte hashed password is sent to the LTTP server in the peer login process.

For verification, the LTTP server compares the received hashed password with its stored hashed password, which was hashed previously by the same SHA-1 function.



SSL Server	localhost
SSL Public Key	Sun RSA public key, 512 bits modulus: 1030230109266254535533869955491068141312935118139078458277931 public exponent: 65537
Number of Peer users	50
Number of Secure Channels	3
Status	RUNNING

Peer User A	Peer User B	Secret Key	TTL	Status
peer_user_0	peer_user_1	282657801418774106721523...	Wed Mar 28 17:46:34 PDT ...	ACTIVE
peer_user_2	peer_user_3	241228808968192448774409...	Wed Mar 28 17:46:42 PDT ...	ACTIVE
peer_user_3	peer_user_6	-27085131008544447633639...	Wed Mar 28 17:46:50 PDT ...	ACTIVE

Status: [Wed Mar 28 16:46:50 PDT 2007] Peer user: peer_user_3 got Kab successfully with peer user peer_user_6

Figure 10. The P2P Network GUI

A Graphical User Interface (GUI) is created for this P2P network to monitor and control peer interactions in the TTP authentication system. The GUI displays in real time P2P network data and peer data including SSL public key, peer hashed key, and existing secure channels between peers. It allows us to run a number of tests on peers including logging in, logging out, voting, creating a secure channel with another peer, and so on.

6.3 Simulation Results

Using the two GUIs to simulate the TTP system and P2P network of peers, we were able to observe the interactions of peers within the TTP system. A number of testing scenarios for peer login/logout, creating secure channels between peers, and voting on peers was successfully carried out. They demonstrated the success of the following objectives of the TTP system.

- The TTP server hierarchy can be expanded and shrunken as needed.
- Peers are able to authenticate and establish secure connections with LTTP servers using the SSL public key protocol.
- LTTP servers are able to authenticate peers using the Kerberos ticket-based authentication protocol.
- Peers are able to login and logout of the TTP system as desired.
- Two peers are able to create a secure channel by requesting the TTP system to generate a secret key, K_{AB}.

- Peers are able to vote on each other in a reputation-based scheme integrated into the TTP system.
- Peers remain anonymous to each other during the test runs.

7 Conclusions

In this paper, an authentication framework is proposed for giving peers in a P2P network the ability to establish secure transactions with each other with the help of an independent trusted third-party authentication system. In this system, peers use public key cryptography, the SSL protocol, to authenticate the TTP system. After logging in, the remainder of the authentication process is accomplished through a Kerberized ticket-based scheme. To help peers determine other peer's integrity, a reputation-based scheme is integrated into the system that lets peers rate each other based on completed transactions.

This TTP authentication system provides several advantages to peers in a P2P network. The system can be plugged into any existing P2P network; it is scalable to accommodate network growth; it provides secure transactions for peers; it provides accurate rating scheme for peers; and most importantly, it accomplishes all of these goals without compromising peer anonymity.

References

[1] Sonja Buchegger and Jean-Yves Le Boudec. A Robust Reputation System for P2P and Mobile Ad-hoc Networks. *In Proceedings of the 2nd Workshop on the Economics of Peer-to-Peer Systems*, 2004.
[2] Ernesto Damiani, De Capitani di Vimercati, Stefano Paraboschi. A Reputation Based Approach for Choosing Reliable Resources in PeertoPeer Networks. *In Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002.
[3] Ali Aydin Selcuk, Ersin Uzun, Mark Resat Pariente. A Reputation-Based Trust Management System for P2P Networks. *Technical Report BU-CE-0402*, Department of Computer Engineering, Bilkent University, 2004.
[4] Li Xiong and Ling Liu. PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities. *IEEE Transactions on Knowledge and Data Engineering (TKDE), Special Issue on Peer-to-Peer Based Data Management*, 2004.
[5] Li Xiong and Ling Liu. Reputation and Trust. Idea Group Inc., 2005.
[6] John Kohl and B. Clifford Neuman. The Kerberos Network Authentication Service (V5). *RFC 1510,* 1993, http://www.ietf.org/rfc/rfc1510.txt
[7] Marvin A. Sirbu and John Chung-I Chuang. Distributed Authentication in Kerberos Using Public Key Cryptography. Information Networking Institute's Net Bill, Carnegie Mellon University, 2001.
[8] Mark Stamp. *Information Security: Principles and Practice*. Wiley-Interscience, 2005.

[9] Kaufman, C., R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World.* Prentice Hall PTR, 2nd Edition, 2002.
[10] Steve M. Bellovin and Michael Merritt. Limitations of the Kerberos Authentication System. *In USENIX Conference Proceedings,* 1991.
[11] K. E. B. Hickman. Secure Socket Library. *Netscape Communication Corp.,* 1995.
[12] Sun Microsystem. Java Secure Socket Extension Reference Guide. http://java.sun.com/j2se/1.5.0/docs/guide/security/jsse/JSSERefGuide.html
[13] MySQL AB. MySQL 5.0 Reference Manual. http://dev.mysql.com/doc/refman/5.0/en
[14] Sun Microsystem. Java Database Connection Reference Guide. http://java.sun.com/j2se/1.5.0/docs/guide/jdbc
[15] Sun Microsystem. Java Cryptography Extension Reference Guide. http://java.sun.com/j2se/1.5.0/docs/guide/security/jce/JCERefGuide.html
[16] Wikipedia. Smart Card. http://en.wikipedia.org/wiki/Smart_card
[17] D W Chadwick. Understanding X.500 – The Directory. http://sec.cs.kent.ac.uk/x500book/
[18] Wikipedia. Anonymous P2P. http://en.wikipedia.org/wiki/Anonymous_P2P

Appendix

A.1 Message Specification

This appendix provides the specification of messages used in the TTP system. All messages are constructed with a message identification number followed by a number of encrypted or unencrypted data:

{ message id, data 1, data 2, … }, where message id is a 4-byte integer.

This appendix uses the following notation:

- Peer Alice is a peer named Alice that is often the requester.
- Peer Bob is a peer named Bob that is often the replier.

A.1.1 Messages between Peers

A.1.1.1 To Establish a Secure Channel

COM_RQST	:	peer Alice → peer Bob
COM_DENY_RPLY	:	peer Alice ← peer Bob
COM_ACCEPT_RPLY	:	peer Alice ← peer Bob
COM_ACCEPT_ACK	:	peer Alice → peer Bob
COM_ERR	:	peer Alice ←→ peer Bob

These messages between two peers are used in establishing a secure channel. As in figure below, peer Alice sends an invitation to peer Bob for establishing a secure channel so that they can secretly use to transfer their transactions. After peer Bob determines to accept her invitation, he requests his LTTP server to generate a secret symmetric key used for encrypting messages in the secure channel. The LTTP server replies to Bob with the generated key and a "ticket to Alice" that also includes the generated key which Bob then sends to Alice. After both peers acknowledge each other on sharing this key, they can start using the secure channel by encrypting all messages with the shared key K_{AB}.

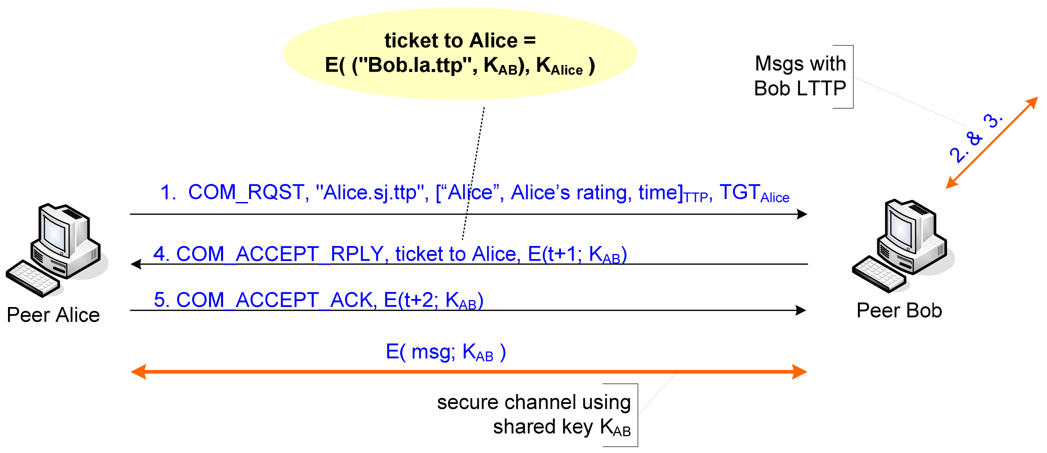

ticket to Alice = E(("Bob.la.ttp", K_AB), K_Alice)
Msgs with Bob LTTP
2. & 3.
1. COM_RQST, "Alice.sj.ttp", ["Alice", Alice's rating, time]TTP, TGT Alice
4. COM_ACCEPT_RPLY, ticket to Alice, E(t+1; K_AB)
5. COM_ACCEPT_ACK, E(t+2; K_AB)
Peer Alice
Peer Bob
E(msg; K_AB)
secure channel using shared key K_AB

Figure 11. Peer Alice Establishes Secure Channel with Peer Bob

- **COM_ RQST** : peer Alice → peer Bob
  Peer Alice sends a request to peer Bob for establishing a secure channel.

  Message Format:
  { COM_RQST, peer name, [peer name, peer's rating, timestamp]$_{TTP}$, TGT$_{peer}$ }

peer name	:	string such as "alice.sj.ttp"
[peer name, peer's rating, timestamp]$_{TTP}$	:	latest peer rating signed by TTP system
TGT$_{peer}$	:	peer's TGT issued by its LTTP

  Example:
  { COM_RQST, "alice.sj.ttp", ["Alice", Alice's rating, t]$_{TTP}$, TGT$_{Alice}$ }

- **COM_DENY_RPLY** : peer Alice ← peer Bob
  Peer Bob denies this request for establishing a secure channel due to his decision on Alice's rating, his server's capacity, and so on.

  Message Format:
  { COM_DENY_RPLY }

- **COM_ACCEPT_RPLY** : peer Alice ← peer Bob
  Peer Bob replies his acceptance of Alice's request after requesting his LTTP to generate a shared symmetric key K_{AB}.

  Message Format:
  { COM_ACCEPT_RPLY, ticket to Alice, E(t; K_{AB}) }

  Ticket to Alice = E("bob.la.ttp", K_{AB}; K_{alice})

E(t+1; K_{AB})	:	Bob's challenge

  Example:

{ COM_ACCEPT_RPLY, E(“bob.la.ttp”, K_{AB}; K_{alice}), E(t; K_{AB}) }

- **COM_ACCEPT_ACK** : peer Alice → peer Bob
  Peer Alice responses to Bob’s challenge.

  Message Format:
  { COM_ACCEPT_ACK, E(t+1; K_{AB}) }

  E(t+2; K_{AB}) : reply to Bob’s challenge

  Example:
  { COM_ACCEPT_ACK, E(t+1; K_{AB}) }

- **COM_ERR** : peer Alice ← peer Bob
  Peer Bob reports an error that could be caused by invalid data in Alice message, overloading in Bob’s capacity, and so on.

  Message Format:
  { COM_ERR, error number }

NOTE:
Message 2 and 3 from peer Bob to his LTTP server is specified in section A.1.2.4.

A.1.1.2 To Get a Peer Rating

RATING_RQST	:	peer Alice → peer Bob
RATING_DENY	:	peer Alice ← peer Bob
RATING_RPLY	:	peer Alice ← peer Bob
RATING_ERR	:	peer Alice ← peer Bob

A Peer can request for other peers’ ratings so that he can determine how much trustworthy the other peer is. As in figure below, peer Alice requests for Bob’s rating:

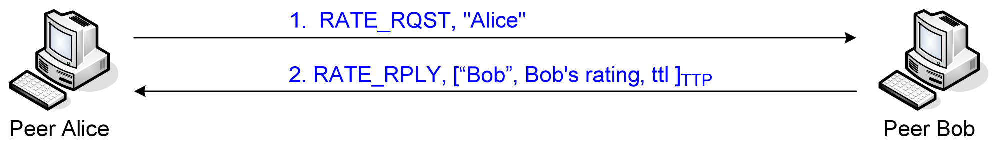


Figure 12. Peer Alice Requests Peer Bob’s Rating

- **RATING_RQST** : peer Alice → peer Bob
  Peer Alice requests for Bob’s rating.

  Message Format:
  { RATING_RQST, peer name }

Peer name : name such as “alice.sj.ttp”

- **RATING_DENY** : peer Alice ← peer Bob
  Peer Bob denies this request due to reasons such as Bob doesn’t want to send his rating to Alice, Bob’s computer is too busy, and so on.

  Message Format:
  { RATING_DENY }

- **RATING_RPLY** : peer Alice ← peer Bob
  Peer Bob replies to the request with his rating.

  Message Format:
  { RATING_RPLY, [peer name, peer’s rating, ttl]$_{TTP}$ }

  [peer name, peer’s rating, ttl]$_{TTP}$
  : signed and time-stamped rating issued to peer by the TTP system

- **RATING_ERR** : peer Alice ← peer Bob
  Error happens in peer Bob due to condition such as computer overloading, invalid request, and so on.

  Message Format:
  { RATING_ERROR, error number }

A.1.2 Messages between Peers and LTTPs

The main difference in messages between peers and LTTPs is that all messages are wrapped within an SSL session/connection. In other words, peer must establish an SSL session with the LTTP server first before sending these messages.

In addition, peer can only communicate with the assigned LTTP. Vice versa, the LTTP server can only communicate with its registered peers.

A.1.2.1 To Register with the TTP System

REG_RQST	
REG_RPLY	(ok)
REG_ERR	(TTP system is busy, error in registration info, etc)

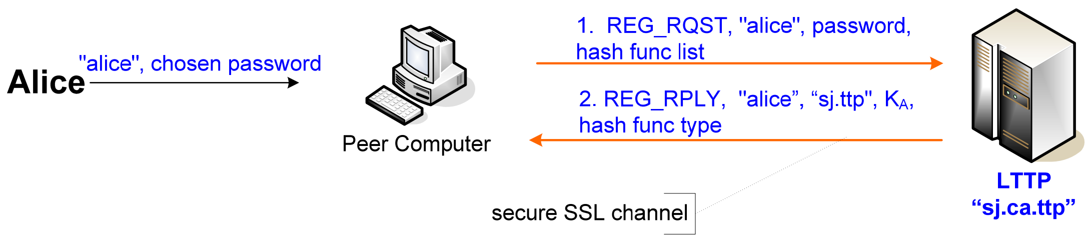

Alice
"alice", chosen password
Peer Computer
1. REG_RQST, "alice", password, hash func list
2. REG_RPLY, "alice", "sj.ttp", K_A, hash func type
secure SSL channel
LTTP
"sj.ca.ttp"

Figure 13. Peer Alice Registers with TTP System

Peer needs to register with the TTP system when he wants to join. In the registration process, peer sends information to a designated LTTP server that the TTP system reserves for peer registration. Upon obtaining peer registration information, this LTTP server will determine the actual LTTP server that the peer will be assigned to.

Peer software does not store the user password but instead the hashed password, K_A, along with the hash function type.

- **REG_RQST** : peer Alice → TTP
  Peer Alice requests to register with the TTP system.

  Message Format:
  { REG_RQST, peer name, chosen password, hash function list }

Peer name – 128B	:	peer name
Chosen password – 32B	:	password chosen by the peer
Hash function list – 128B	:	list of hash function types

- **REG_RPLY** : peer Alice ← TTP
  LTTP server replies with the peer name, the hashed password, and its chosen hash function type after creating a unique profile for the peer in its database. Peer software stores this hash function type for use in the peer login.

  Message Format:
  { REG_RPLY, peer name, LTTP name, LTTP IP address, hash(password), hash function type }

Peer name - 128B	:	peer name
LTTP name - 128B	:	assigned LTTP name
LTTP IP address – 4B	:	IP address of the LTTP server
K_A = *hash(password) – 20B*	:	peer secret key, K_A created by hash function SHA-1
Hash function type – 8B	:	hash function type

- **REG_ERR** : peer Alice ← TTP
  LTTP replies with an error due to conditions such as peer name already existed, invalid password, server overloaded, and so on.

Message Format:

{ REG_ERROR, error number }

A.1.2.2 To Login to the TTP System

LOGIN_RQST	
LOGIN_RPLY	(ok, TGT is sent to peer also)
LOGIN_ERR	(wrong password, etc)

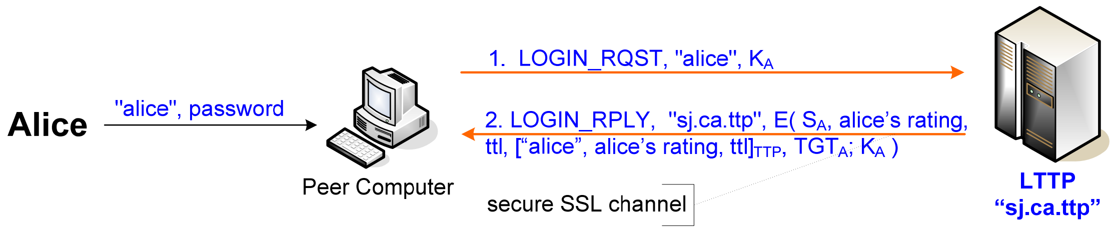

Alice
"alice", password
Peer Computer
1. LOGIN_RQST, "alice", K_A
2. LOGIN_RPLY, "sj.ca.ttp", E(S_A, alice's rating, ttl, ["alice", alice's rating, ttl]$_{TTP}$, TGT_A; K_A)
secure SSL channel
LTTP
"sj.ca.ttp"

Figure 14. Peer Alice Logs into the LTTP Server

When Alice logs in, her peer computer first establishes an SSL session and sends her name and her hashed password to the LTTP server. When receiving the reply, peer uses her hashed password, K_{peer} = *hash(peer password)*, to decrypt the message and obtain the followings from the LTTP server:

- A session key, S_A , generated randomly by the LTTP server
- A Ticket Granting Ticket, TGT_{peer} = *(LTTP name, E(peer name,* S_{peer}*, ttl;* K_{LTTP}*))*
- Signed and time-stamped rating, *[peer name, peer's rating, ttl]*$_{TTP}$

These data are saved on the peer computer for use within the maximum lifetime of the login session.

- **LOGIN_RQST** : peer Alice → TTP

  Peer Alice requests to login to the LTTP server.

  Message Format:

  { LOGIN_RQST, peer name, K_{peer} *}*

Peer name - 128B :	peer name
K_{peer} *= hash(password) – 20B*:	peer symmetric key, created by hash function SHA-1

- **LOGIN_RPLY** : peer Alice ← TTP

  The LTTP server accepts the peer login and replies with the associated data encrypted with the peer symmetric key.

  Message Format:

{ LOGIN_RPLY, LTTP name, E(S_A, peer rating, TTL , [peer name, peer's rating, TTL]$_{TTP}$, TGT_A; K_A) }

LTTP name – 128B	:	LTTP name such as "sj.ttp"
S_A - 24B	:	peer session key generated randomly by the LTTP
peer rating – 8B	:	peer rating in range of [-5, 5]
TTL – 8B	:	time-to-live of the session
[peer name, peer's rating, TTL]$_{TTP}$	:	time-stamped and signed peer rating
TGT_A – 296B	:	peer Ticket Granting Ticket
K_A – 20B	:	peer symmetric key, created by hash function SHA-1

- **LOGIN_ERR** : peer Alice ← TTP
  LTTP replies with an error due to conditions such as server busy, server overloaded, and so on.

  Message Format:
  { LOGIN_ERROR, error number }

NOTE:
_ The purposes of peer login are to get a TGT, a session key, and the latest peer rating.
_ Session key, S_A, in the issued TGT is stored on the peer with an expiration time but not stored on the LTTP server. The session's time-to-live represents a maximum window in which the peer can operate in. When the peer logs out, his session key along with other temporary data are destroyed in the peer computer.
_ If peer wants to get the latest rating or a new session key, he needs to re-login to the system.

A.1.2.3 To Logout of the TTP System

LOGOUT_RQST	
LOGOUT_ACK	(ok)
LOGOUT_ERR	(error)

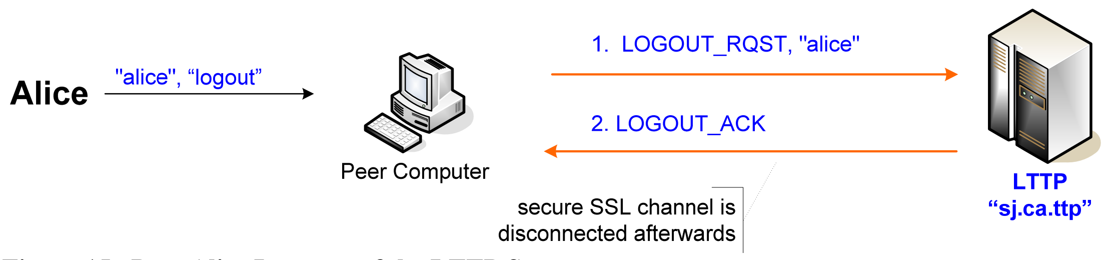


Figure 15. Peer Alice Logs out of the LTTP Server

Peer Alice requests to logout of the TTP system. When receiving the LTTP server's acknowledgement, peer software shuts down the SSL connection with TTP system.

- **LOGOUT_RQST** : peer Alice → TTP
  Peer Alice requests to logout of the LTTP server.

  Message Format:
  { LOGIN_RQST, peer name, K_{peer} }

Peer name - 128B	:	peer name
K_{peer} *= hash(password) – 20B*	:	peer symmetric key, created by hash function SHA-1

- **LOGOUT_RPLY** : peer Alice ← TTP
  The LTTP acknowledges the peer logout.

  Message Format:
  { LOGOUT_ACK }

- **LOGOUT_ERR** : peer Alice ← TTP
  LTTP replies with an error due to conditions such as server busy, server overloaded, and so on.

  Message Format:
  { LOGOUT_ERROR, error number }

A.1.2.4 To Request for a Secret Symmetric Key K_{AB}

The system assumes that both peer Alice and peer Bob have logged in to and received TGTs from the TTP system.

KEY_RQST	
KEY_RPLY	(ok)
KEY_ERR	(invalid peer, wrong authentication, etc)

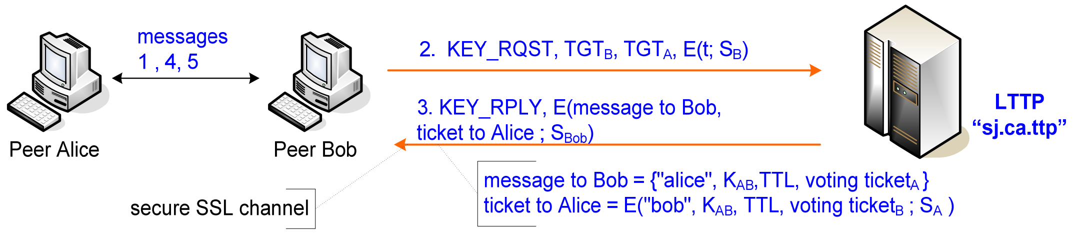

messages
1 , 4, 5
Peer Alice
Peer Bob
2. KEY_RQST, TGT_B, TGT_A, E(t; S_B)
3. KEY_RPLY, E(message to Bob, ticket to Alice ; S_Bob)
LTTP
"sj.ca.ttp"
secure SSL channel
message to Bob = {"alice", K_AB, TTL, voting ticket_A }
ticket to Alice = E("bob", K_AB, TTL, voting ticket_B ; S_A)

Figure 16. Requesting a Secret Key K_{AB} with the LTTP Server

This is the continuation of part A.1.1.1. Peer Bob requests the TTP system to verify peer Alice identity and generate a shared secret key for both peers. The TTP system decrypts peer Alice's TGT_A to verify her identity and generates a random key K_{AB} for establishing the secure channel between peer Alice and peer Bob.

In addition, two voting tickets generated by peers' LTTP server(s) are issued to the two peers. One is sent directly to peer Bob for voting on peer Alice. The other one is kept inside the "ticket to Alice" which is sent to peer Alice for voting on peer Bob. Peers can use these tickets to vote on each other after the transaction is completed.

- **KEY_RQST** : peer Bob → TTP

  Peer Bob requests the TTP system to verify peer Alice identity and generate a shared secret key for both peers.

  Message Format:

  { KEY_RQST, TGT_{peerB}, TGT_{peerA} , E(time; S_{peerB}) }

TGT_{peerB} – 296B	:	the requesting peer's Ticket Granting Ticket
TGT_{peerA} – 296B	:	the other peer's Ticket Granting Ticket
Time – 8B	:	the timestamp in second to prevent relay attack
S_{peerB} – 24B	:	the requesting peer session key

- **KEY_RPLY** : peer Bob ← TTP

  The TTP system verifies peer Alice and replies with the secret key K_{AB}. Two voting tickets are also generated by peers' LTTP(s) for voting on the peer after the transaction is completed.

  Message Format:

  { KEY_RPLY, message to peer B, ticket to peer A }

  message to peer B = { peer A name, K_{AB} , TTL, voting $ticket_{peerA}$ }
  ticket to peer A = E(peer B name, K_{AB} , TTL, voting $ticket_{peerB}$; S_{peerA})
  voting $ticket_{peerA}$ = (peer A's LTTP name, E(peer A name, expiration; K_{LTTPA}))
  voting $ticket_{peerB}$ = (peer B's LTTP name, E(peer B name, expiration; K_{LTTPB}))

Peer A name – 128B	:	peer A name
K_{AB} - 24B	:	shared secret key between peer A and peer B
TTL – 8B	:	time-to-live of K_{AB}
Message to peer B – 440B	:	message sent to peer B
Ticket to peer A – 440B	:	ticket sent to peer A
voting $ticket_{peerA}$ – 272B	:	ticket used to vote on peer A
voting $ticket_{peerB}$ – 272B	:	ticket used to vote on peer B

- **KEY_ERR** : peer Bob ← TTP

LTTP replies with an error due to conditions such as server busy, server overloaded, and so on.

Message Format:

{ KEY_ERROR, error number }

A.1.2.5 To Vote on a Peer

VOTE_RQST	
VOTE_ACK	(ok)
VOTE_ERR	(wrong voting key, wrong vote value, etc)

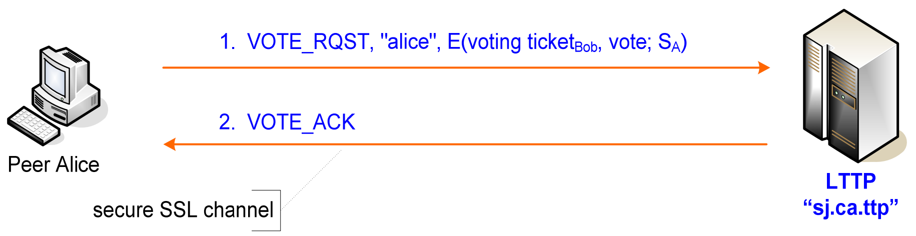

1. VOTE_RQST, "alice", E(voting ticket$_{Bob}$, vote; S$_A$)
2. VOTE_ACK
Peer Alice
secure SSL channel
LTTP
"sj.ca.ttp"

Figure 17. Peer Alice Sends a Vote to the LTTP Server

Peers use the voting ticket obtained in the previous transaction to vote on each other. Only the voted peer's LTTP server can decrypt the ticket to verify the validity of the voting peer's vote and update the peer rating according to the vote.

- **VOTE_RQST** : peer Alice → LTTP
  Peer uses the voting ticket to vote on another peer.

  Message Format:

  { VOTE_RQST, peer A name, E(voting ticket$_{peerB}$, vote; S$_{peerA}$) }

  voting ticket$_{peerB}$ = { LTTP B name, E(peer B name, expiration; K$_{LTTPB}$) }

Peer A name – 128B	:	voting peer name
voting ticket$_{peerB}$ - 272B	:	ticket used to vote on peer B.
vote – 8B	:	a vote of type double in range [-5.0 , +5.0]
S$_{peerA}$ – 24B	:	peer A's symmetric session key

- **VOTE_ACK** : peer Alice ← LTTP
  LTTP server acknowledges the peer's vote after successfully updating the other peer's rating.

  Message Format:

  { VOTE_ACK }

- **VOTE_ERR** : peer Alice ← LTTP

LTTP server replies with an error due to conditions such as invalid vote, server busy, server overloaded, and so on.

Message Format:

{ VOTE_ERROR, error number }

A.1.3 Messages within the TTP System

Each (L)TTP server can only communicate with its immediate parent TTP server. No cross-level communication between LTTPs is allowed. This must be accomplished through the TTP hierarchy. All communication messages between (L)TTP servers to their parent TTP server are encrypted with the (L)TTP's symmetric key.

A.1.3.1 To Relay a Peer Verification

PEER_CHK_RQST	
PEER_CHK_RPLY	(ok)
PEER_CHK_ERR	(fail to verify)

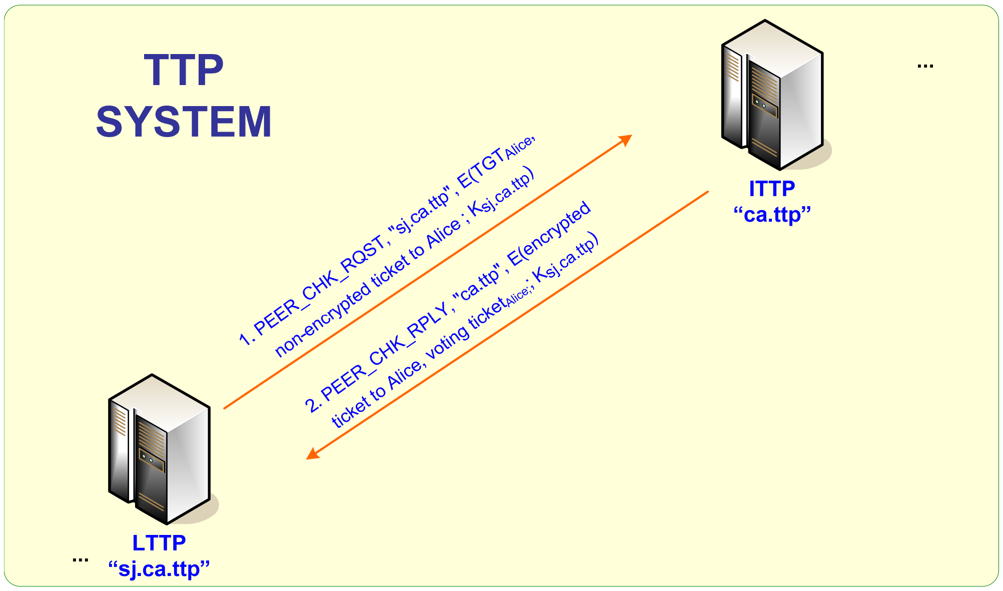

TTP SYSTEM
...
ITTP "ca.ttp"
1. PEER_CHK_RQST, "sj.ca.ttp", E(TGT Alice, non-encrypted ticket to Alice ; Ksj.ca.ttp)
2. PEER_CHK_RPLY, "ca.ttp", E(encrypted ticket to Alice, voting ticket Alice;; Ksj.ca.ttp)
...
LTTP "sj.ca.ttp"

Figure 18. Relaying a Peer Verification

This is the continuation of part A.1.2.4. When the TTP system receives a symmetric key request from a peer for establishing secure channel with another peer, the TTP system needs to verify involved peers and to generate a secret key for them. However, only the peer's assigned LTTP server can verify his identity. Therefore if involved peers are from different LTTP servers, the receiving LTTP server must hierarchically go through the TTP tree structure to request the other peer's LTTP server to verify the peer identity.

This process also involves requesting the other peer's LTTP server to generate a "ticket to peer" encrypted with the peer's symmetric key and a voting ticket on the peer encrypted with the LTTP "master" key.

- **PEER_CHK_RQST** : (L)TTP $\rightarrow$ parent/child (L)TTP
  (L)TTP relays the request to its parent TTP or to one of its child (L)TTP in the TTP hierarchy that lies in the path to the peer's LTTP for verifying a peer and generating the needed tickets.

  Message Format:
  { PEER_CHK_RQST, (L)TTP A name, E(TGT$_{peerB}$, non-encrypted ticket to peer B; K$_{(L)TTPA}$) }

  (L)TTP A name : LTTP name of the requesting peer
  TGT$_{peerB}$: TGT of peer needed to be verified
  non-encrypted ticket to peer B = (peer A name, K$_{AB}$, voting ticket$_{peerA}$)
  : ticket to peer B that is needed to be encrypted with peer B's symmetric key
  K$_{(L)TTPA}$: (L)TTP A's symmetric key with its parent/child (L)TTP

- **PEER_CHK_RPLY** : (L)TTP $\leftarrow$ parent/child (L)TTP
  Parent/child (L)TTP replies or relays the reply with the needed tickets after the peer has been successfully verified.

  Message Format:
  { PEER_CHK_RPLY, (L)TTP B name, E(encrypted ticket to peer B, voting ticket$_{peerB}$; K$_{(L)TTPA}$) }

  (L)TTP B name : the replying LTTP name
  encrypted ticket to peer B = E(peer A name, K$_{AB}$, voting ticket$_{peerB}$; K$_{LTTPB}$)
  : ticket to send to peer B encrypted with its LTTP key
  voting ticket$_{peerB}$ = E(peer B name, expiration; K$_{LTTPB}$)
  : ticket used to vote on peer B
  K$_{(L)TTPA}$: (L)TTP A's symmetric key with its parent/child (L)TTP

- **PEER_CHK_ERR** : (L)TTP $\leftarrow$ parent/child (L)TTP
  (L)TTP relays an error due to condition such as invalid peer, (L)TTP down, etc.

  Message Format:
  { PEER_CHK_ERROR, error number }

A.1.3.2 To Relay a Peer Vote to His LTTP Server

PEER_VOTE_RQST
PEER_VOTE_ACK (ok)
PEER_VOTE_ERR

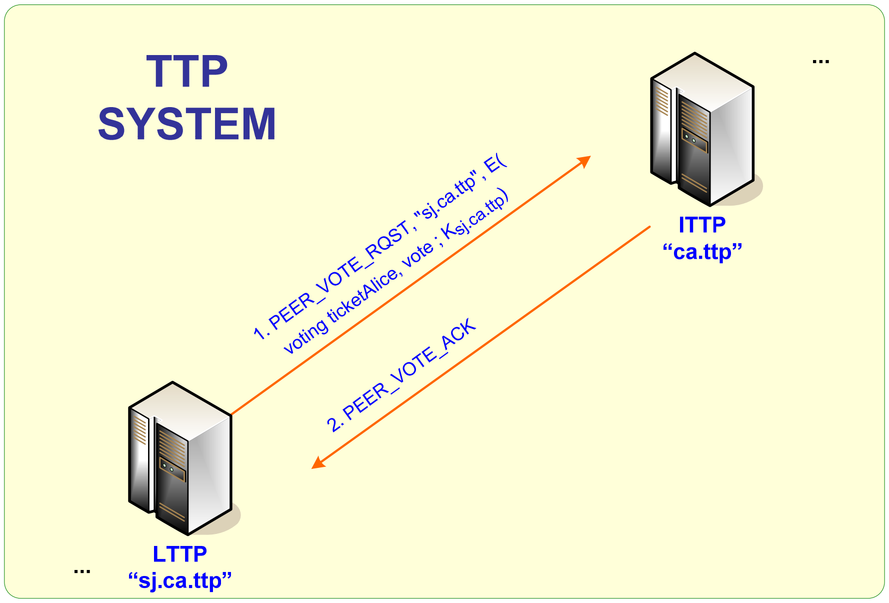

TTP SYSTEM
...
ITTP
"ca.ttp"
1. PEER_VOTE_RQST, "sj.ca.ttp", E(voting ticketAlice, vote ; Ksj.ca.ttp)
2. PEER_VOTE_ACK
LTTP
"sj.ca.ttp"
...

Figure 19. Relaying a Peer Vote to his LTTP Server

This is the continuation of partA.1.2.5. When TTP system receives a vote from a peer on another peer, the TTP system verifies the voting ticket and updates the voted peer's rating. However, only the voted peer's LTTP can decrypt this ticket to verify. Therefore if involved peers are from different LTTPs, the receiving LTTP must hierarchically go through the TTP tree structure to request the other peer's LTTP to process this vote.

- **PEER_VOTE_RQST** : (L)TTP → parent/child (L)TTP
  (L)TTP relays the vote to its parent TTP or to one of its child (L)TTP that lies in the path to the voted peer's LTTP.

  Message Format:
  { PEER_VOTE_RQST, (L)TTP A name, E(voting ticket$_{peerB}$, vote; K$_{(L)TTPA}$) }

  (L)TTP A name : the requesting LTTP name
  voting ticket$_{peerB}$ = (LTTPB name, E(peer B name, expiration; K$_{LTTPB}$))
  : ticket used to vote on peer B
  vote : the vote
  K$_{(L)TTPA}$: (L)TTP A's symmetric key with its parent/child (L)TTP

- **PEER_VOTE_ACK** : (L)TTP ← parent/child (L)TTP

(L)TTP relays the vote acknowledgement from the voted peer's LTTP back to the original requesting LTTP through its parent TTP or one of its child (L)TTP in the path.

Message Format:
{ PEER_VOTE_ACK }

- **PEER_VOTE_ERR** : (L)TTP ← parent/child (L)TTP
  (L)TTP relays an error on the vote due to conditions such as invalid voting ticket, (L)TTP down, and so on.

  Message Format:
  { PEER_VOTE_ERROR, error number }

A.1.3.3 To Update Symmetric Key with Its Parent TTP

SKEY_UPD_RQST	
SKEY_UPD_ACK	(ok)
SKEY_UPD_ERR	(wrong TTP id, etc)

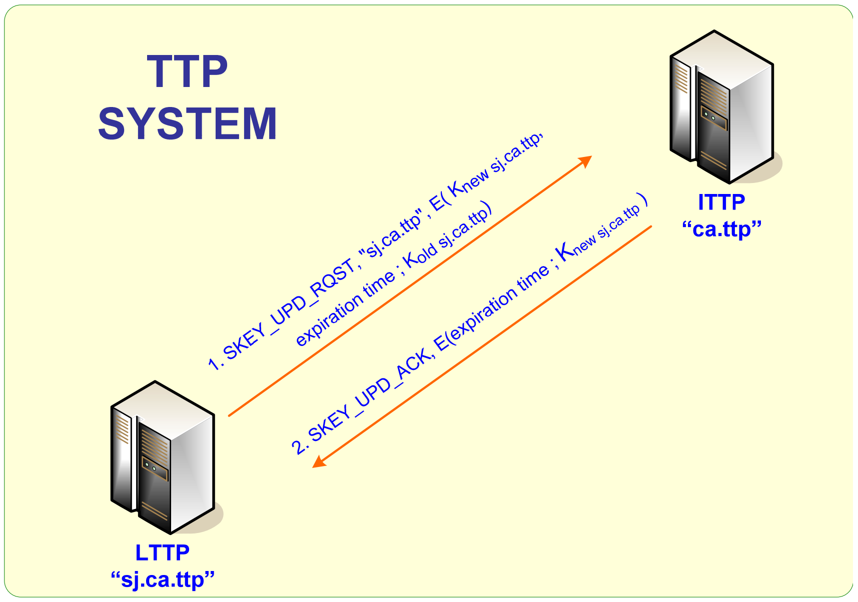

TTP SYSTEM
ITTP
"ca.ttp"
1. SKEY_UPD_RQST, "sj.ca.ttp", E(Knew sj.ca.ttp, expiration time ; Kold sj.ca.ttp)
2. SKEY_UPD_ACK, E(expiration time ; Knew sj.ca.ttp)
LTTP
"sj.ca.ttp"

Figure 20. Updating Symmetric Key with Its Parent Server

(L)TTPs change their symmetric key $K_{(L)TTP}$ shared with the parent TTPs and update the new key to the parent TTPs periodically. To strengthen the system security, lower-level (L)TTPs change their symmetric keys more often than the upper-level TTPs.

- **SKEY_UPD_RQST** : (L)TTP → parent TTP
  The (L)TTP sends a new symmetric key and its expiration time to its parent TTP.

Message Format:

{ SKEY_UPD_RQST, (L)TTP name, E($K_{new\ (L)TTP}$, expiration time; $K_{old\ (L)TTP}$) }

(L)TTP name	:	the child LTTP name
$K_{new\ (L)TTP}$	:	the new (L)TTP's symmetric key
expiration time:		the new key's expiration time until the next update
$K_{old\ (L)TTP}$	:	the old (L)TTP's symmetric key

- **SKEY_UPD_ACK** : (L)TTP ← parent TTP

  The parent TTP sends an acknowledgement of receiving the new symmetric key back to the (L)TTP.

  Message Format:

  { SKEY_UPD_ACK, E(expiration time; $K_{new\ (L)TTP}$) }

expiration time :		new key's expiration time until the next update
$K_{new\ (L)TTP}$	:	new (L)TTP's symmetric key

- **SKEY_UPD_ERR** : (L)TTP ← parent TTP

  Parent TTP reports an error on the key update due to conditions such as invalid symmetric key, invalid expiration time, TTP busy, and so on.

  Message Format:

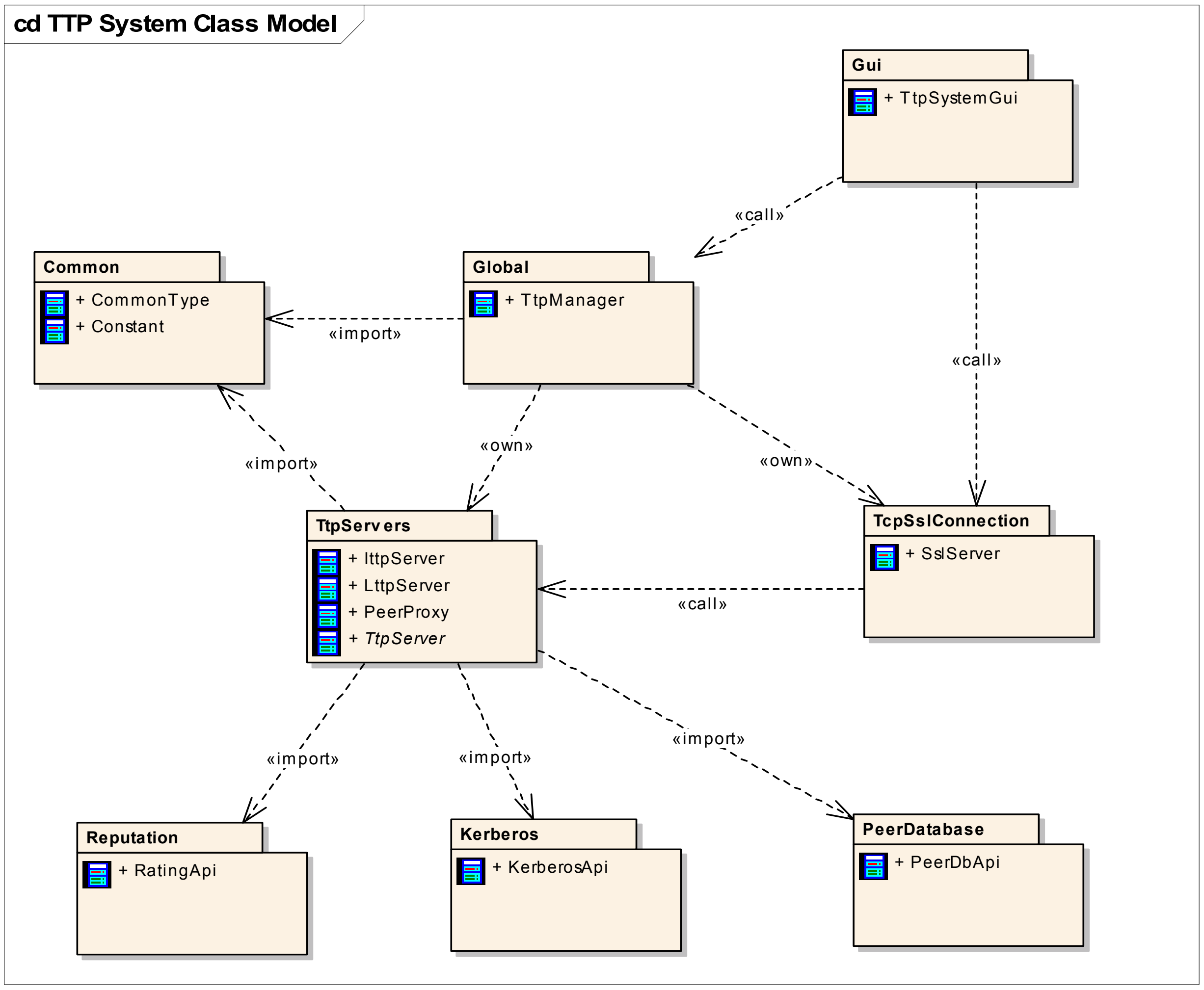
  { SKEY_UPD_ERROR, error number }

A.2 *The Software Simulation's UML Diagrams*

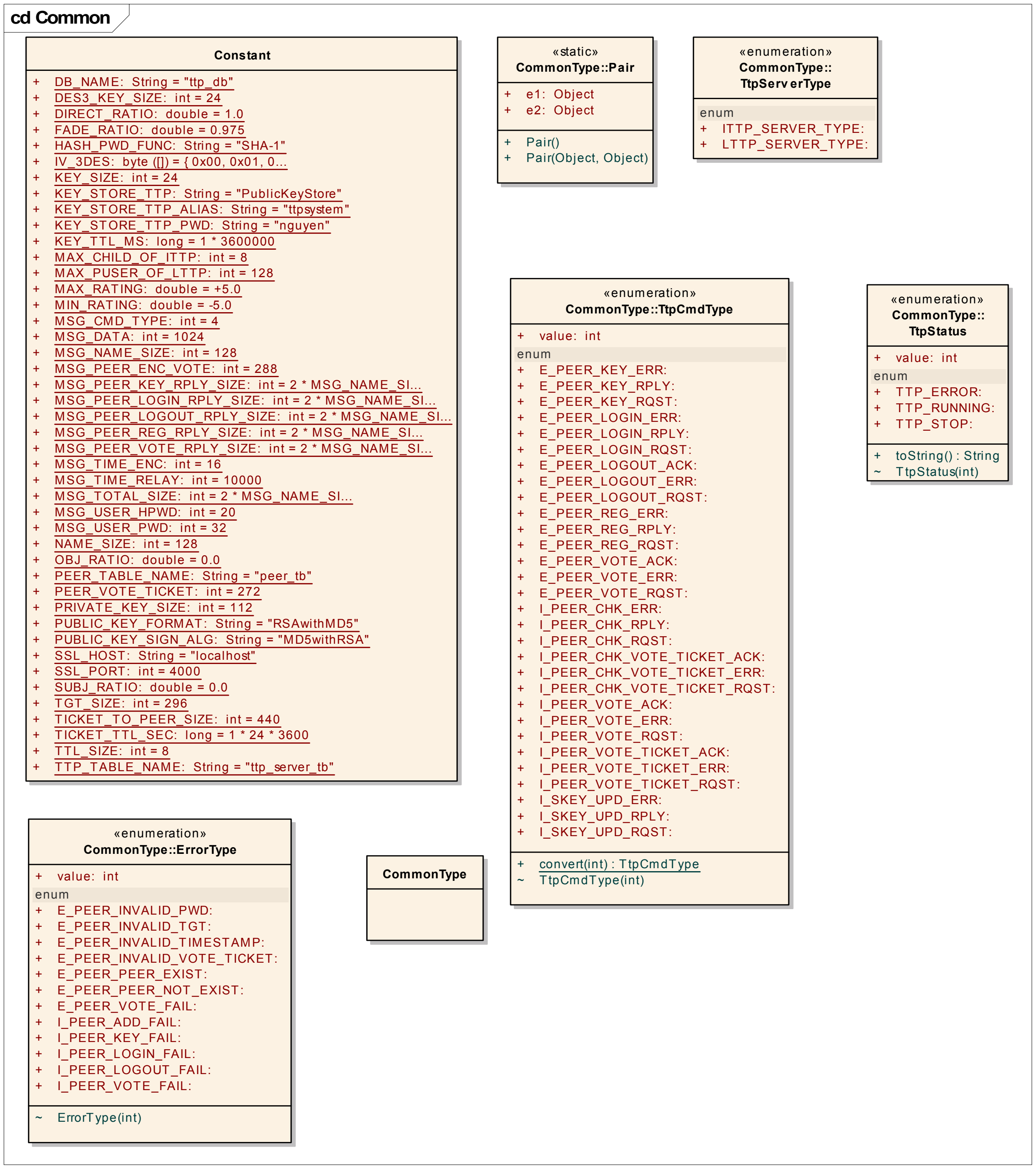
This appendix provides the UML diagrams of the software simulation.

A.2.1 The TTP System's UML Diagrams

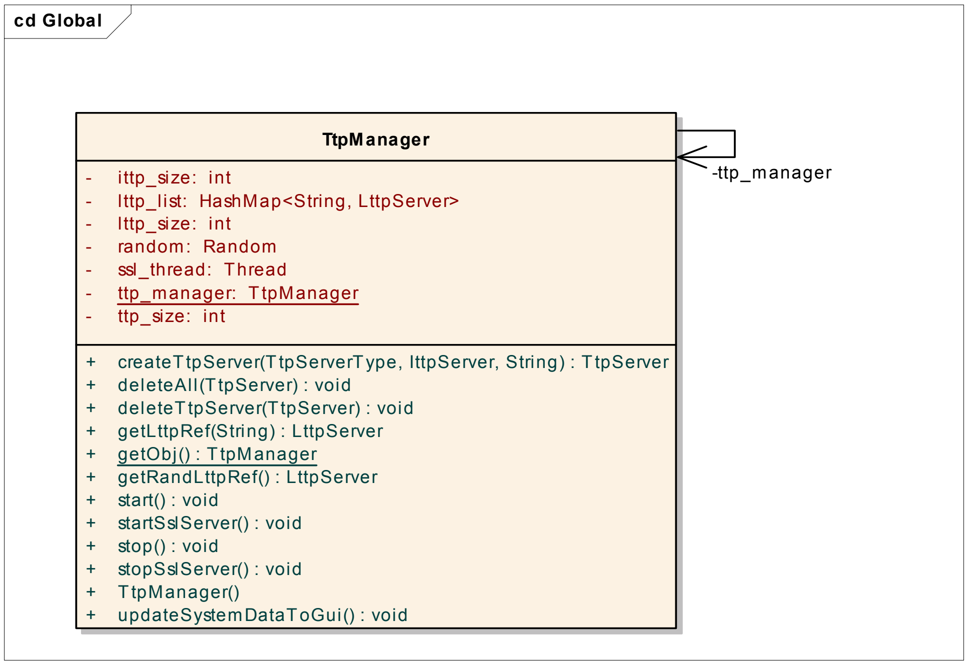
- **The TTP System Package Model**



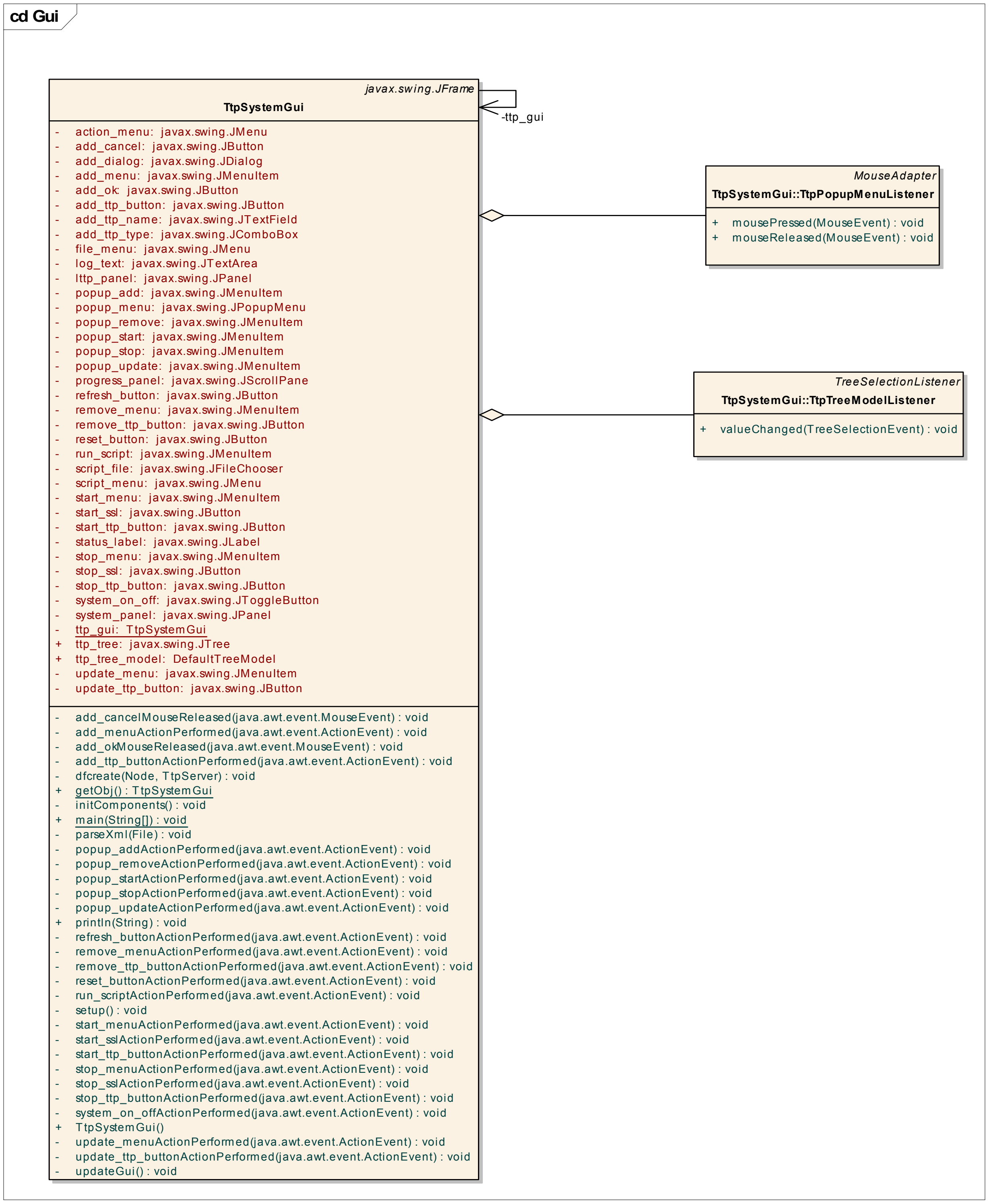
▪ Package Common


cd Common
Constant
+ DB_NAME: String = "ttp_db"
+ DES3_KEY_SIZE: int = 24
+ DIRECT_RATIO: double = 1.0
+ FADE_RATIO: double = 0.975
+ HASH_PWD_FUNC: String = "SHA-1"
+ IV_3DES: byte ([]) = { 0x00, 0x01, 0...
+ KEY_SIZE: int = 24
+ KEY_STORE_TTP: String = "PublicKeyStore"
+ KEY_STORE_TTP_ALIAS: String = "ttpsystem"
+ KEY_STORE_TTP_PWD: String = "nguyen"
+ KEY_TTL_MS: long = 1 * 3600000
+ MAX_CHILD_OF_ITTP: int = 8
+ MAX_PUSER_OF_LTTP: int = 128
+ MAX_RATING: double = +5.0
+ MIN_RATING: double = -5.0
+ MSG_CMD_TYPE: int = 4
+ MSG_DATA: int = 1024
+ MSG_NAME_SIZE: int = 128
+ MSG_PEER_ENC_VOTE: int = 288
+ MSG_PEER_KEY_RPLY_SIZE: int = 2 * MSG_NAME_SI...
+ MSG_PEER_LOGIN_RPLY_SIZE: int = 2 * MSG_NAME_SI...
+ MSG_PEER_LOGOUT_RPLY_SIZE: int = 2 * MSG_NAME_SI...
+ MSG_PEER_REG_RPLY_SIZE: int = 2 * MSG_NAME_SI...
+ MSG_PEER_VOTE_RPLY_SIZE: int = 2 * MSG_NAME_SI...
+ MSG_TIME_ENC: int = 16
+ MSG_TIME_RELAY: int = 10000
+ MSG_TOTAL_SIZE: int = 2 * MSG_NAME_SI...
+ MSG_USER_HPWD: int = 20
+ MSG_USER_PWD: int = 32
+ NAME_SIZE: int = 128
+ OBJ_RATIO: double = 0.0
+ PEER_TABLE_NAME: String = "peer_tb"
+ PEER_VOTE_TICKET: int = 272
+ PRIVATE_KEY_SIZE: int = 112
+ PUBLIC_KEY_FORMAT: String = "RSAwithMD5"
+ PUBLIC_KEY_SIGN_ALG: String = "MD5withRSA"
+ SSL_HOST: String = "localhost"
+ SSL_PORT: int = 4000
+ SUBJ_RATIO: double = 0.0
+ TGT_SIZE: int = 296
+ TICKET_TO_PEER_SIZE: int = 440
+ TICKET_TTL_SEC: long = 1 * 24 * 3600
+ TTL_SIZE: int = 8
+ TTP_TABLE_NAME: String = "ttp_server_tb"
«static»
CommonType::Pair
+ e1: Object
+ e2: Object
+ Pair()
+ Pair(Object, Object)
«enumeration»
CommonType::
TtpServerType
enum
+ ITTP_SERVER_TYPE:
+ LTTP_SERVER_TYPE:
«enumeration»
CommonType::TtpCmdType
+ value: int
enum
+ E_PEER_KEY_ERR:
+ E_PEER_KEY_RPLY:
+ E_PEER_KEY_RQST:
+ E_PEER_LOGIN_ERR:
+ E_PEER_LOGIN_RPLY:
+ E_PEER_LOGIN_RQST:
+ E_PEER_LOGOUT_ACK:
+ E_PEER_LOGOUT_ERR:
+ E_PEER_LOGOUT_RQST:
+ E_PEER_REG_ERR:
+ E_PEER_REG_RPLY:
+ E_PEER_REG_RQST:
+ E_PEER_VOTE_ACK:
+ E_PEER_VOTE_ERR:
+ E_PEER_VOTE_RQST:
+ I_PEER_CHK_ERR:
+ I_PEER_CHK_RPLY:
+ I_PEER_CHK_RQST:
+ I_PEER_CHK_VOTE_TICKET_ACK:
+ I_PEER_CHK_VOTE_TICKET_ERR:
+ I_PEER_CHK_VOTE_TICKET_RQST:
+ I_PEER_VOTE_ACK:
+ I_PEER_VOTE_ERR:
+ I_PEER_VOTE_RQST:
+ I_PEER_VOTE_TICKET_ACK:
+ I_PEER_VOTE_TICKET_ERR:
+ I_PEER_VOTE_TICKET_RQST:
+ I_SKEY_UPD_ERR:
+ I_SKEY_UPD_RPLY:
+ I_SKEY_UPD_RQST:
+ convert(int) : TtpCmdType
~ TtpCmdType(int)
«enumeration»
CommonType::
TtpStatus
+ value: int
enum
+ TTP_ERROR:
+ TTP_RUNNING:
+ TTP_STOP:
+ toString() : String
~ TtpStatus(int)
«enumeration»
CommonType::ErrorType
+ value: int
enum
+ E_PEER_INVALID_PWD:
+ E_PEER_INVALID_TGT:
+ E_PEER_INVALID_TIMESTAMP:
+ E_PEER_INVALID_VOTE_TICKET:
+ E_PEER_PEER_EXIST:
+ E_PEER_PEER_NOT_EXIST:
+ E_PEER_VOTE_FAIL:
+ I_PEER_ADD_FAIL:
+ I_PEER_KEY_FAIL:
+ I_PEER_LOGIN_FAIL:
+ I_PEER_LOGOUT_FAIL:
+ I_PEER_VOTE_FAIL:
~ ErrorType(int)
CommonType

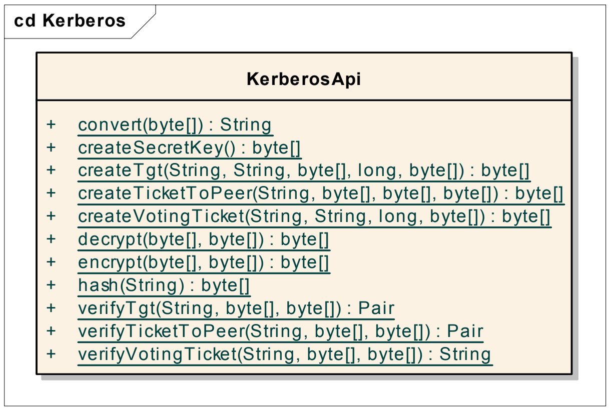
- **Class TtpManager**


cd Global
TtpManager
- ittp_size: int
- lttp_list: HashMap<String, LttpServer>
- lttp_size: int
- random: Random
- ssl_thread: Thread
- ttp_manager: TtpManager
- ttp_size: int
+ createTtpServer(TtpServerType, IttpServer, String) : TtpServer
+ deleteAll(TtpServer) : void
+ deleteTtpServer(TtpServer) : void
+ getLttpRef(String) : LttpServer
+ getObj() : TtpManager
+ getRandLttpRef() : LttpServer
+ start() : void
+ startSslServer() : void
+ stop() : void
+ stopSslServer() : void
+ TtpManager()
+ updateSystemDataToGui() : void
-ttp_manager

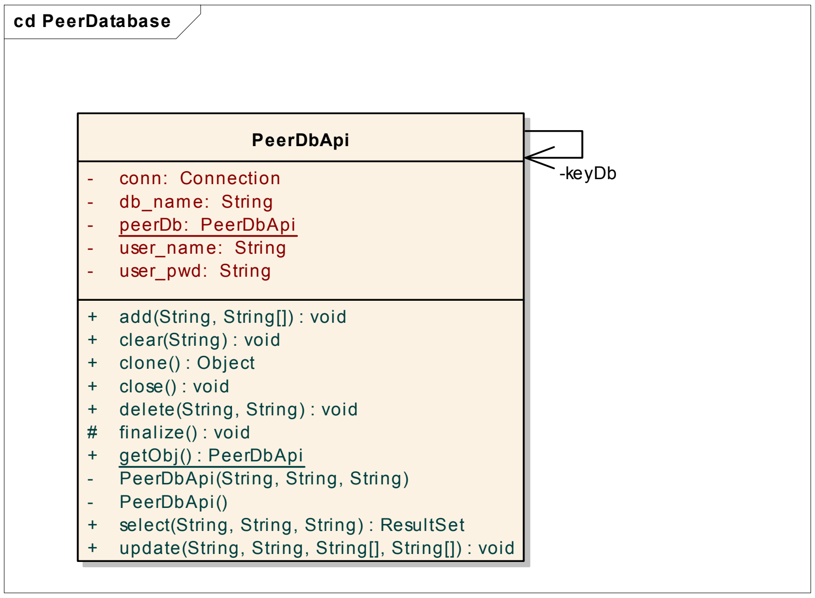
- **Class TtpSystemGui**


cd Gui
javax.swing.JFrame
TtpSystemGui
-ttp_gui
- action_menu: javax.swing.JMenu
- add_cancel: javax.swing.JButton
- add_dialog: javax.swing.JDialog
- add_menu: javax.swing.JMenuItem
- add_ok: javax.swing.JButton
- add_ttp_button: javax.swing.JButton
- add_ttp_name: javax.swing.JTextField
- add_ttp_type: javax.swing.JComboBox
- file_menu: javax.swing.JMenu
- log_text: javax.swing.JTextArea
- lttp_panel: javax.swing.JPanel
- popup_add: javax.swing.JMenuItem
- popup_menu: javax.swing.JPopupMenu
- popup_remove: javax.swing.JMenuItem
- popup_start: javax.swing.JMenuItem
- popup_stop: javax.swing.JMenuItem
- popup_update: javax.swing.JMenuItem
- progress_panel: javax.swing.JScrollPane
- refresh_button: javax.swing.JButton
- remove_menu: javax.swing.JMenuItem
- remove_ttp_button: javax.swing.JButton
- reset_button: javax.swing.JButton
- run_script: javax.swing.JMenuItem
- script_file: javax.swing.JFileChooser
- script_menu: javax.swing.JMenu
- start_menu: javax.swing.JMenuItem
- start_ssl: javax.swing.JButton
- start_ttp_button: javax.swing.JButton
- status_label: javax.swing.JLabel
- stop_menu: javax.swing.JMenuItem
- stop_ssl: javax.swing.JButton
- stop_ttp_button: javax.swing.JButton
- system_on_off: javax.swing.JToggleButton
- system_panel: javax.swing.JPanel
- ttp_gui: TtpSystemGui
+ ttp_tree: javax.swing.JTree
+ ttp_tree_model: DefaultTreeModel
- update_menu: javax.swing.JMenuItem
- update_ttp_button: javax.swing.JButton
- add_cancelMouseReleased(java.awt.event.MouseEvent) : void
- add_menuActionPerformed(java.awt.event.ActionEvent) : void
- add_okMouseReleased(java.awt.event.MouseEvent) : void
- add_ttp_buttonActionPerformed(java.awt.event.ActionEvent) : void
- dfcreate(Node, TtpServer) : void
+ getObj() : TtpSystemGui
- initComponents() : void
+ main(String[]) : void
- parseXml(File) : void
- popup_addActionPerformed(java.awt.event.ActionEvent) : void
- popup_removeActionPerformed(java.awt.event.ActionEvent) : void
- popup_startActionPerformed(java.awt.event.ActionEvent) : void
- popup_stopActionPerformed(java.awt.event.ActionEvent) : void
- popup_updateActionPerformed(java.awt.event.ActionEvent) : void
+ println(String) : void
- refresh_buttonActionPerformed(java.awt.event.ActionEvent) : void
- remove_menuActionPerformed(java.awt.event.ActionEvent) : void
- remove_ttp_buttonActionPerformed(java.awt.event.ActionEvent) : void
- reset_buttonActionPerformed(java.awt.event.ActionEvent) : void
- run_scriptActionPerformed(java.awt.event.ActionEvent) : void
- setup() : void
- start_menuActionPerformed(java.awt.event.ActionEvent) : void
- start_sslActionPerformed(java.awt.event.ActionEvent) : void
- start_ttp_buttonActionPerformed(java.awt.event.ActionEvent) : void
- stop_menuActionPerformed(java.awt.event.ActionEvent) : void
- stop_sslActionPerformed(java.awt.event.ActionEvent) : void
- stop_ttp_buttonActionPerformed(java.awt.event.ActionEvent) : void
- system_on_offActionPerformed(java.awt.event.ActionEvent) : void
+ TtpSystemGui()
- update_menuActionPerformed(java.awt.event.ActionEvent) : void
- update_ttp_buttonActionPerformed(java.awt.event.ActionEvent) : void
- updateGui() : void
MouseAdapter
TtpSystemGui::TtpPopupMenuListener
+ mousePressed(MouseEvent) : void
+ mouseReleased(MouseEvent) : void
TreeSelectionListener
TtpSystemGui::TtpTreeModelListener
+ valueChanged(TreeSelectionEvent) : void

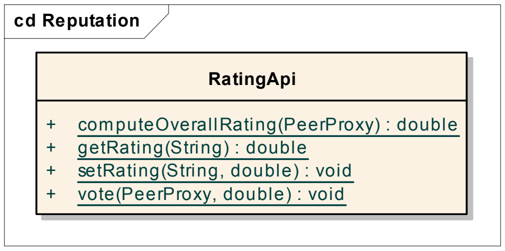
- **Class Kerberos**


cd Kerberos
KerberosApi
+ convert(byte[]) : String
+ createSecretKey() : byte[]
+ createTgt(String, String, byte[], long, byte[]) : byte[]
+ createTicketToPeer(String, byte[], byte[], byte[]) : byte[]
+ createVotingTicket(String, String, long, byte[]) : byte[]
+ decrypt(byte[], byte[]) : byte[]
+ encrypt(byte[], byte[]) : byte[]
+ hash(String) : byte[]
+ verifyTgt(String, byte[], byte[]) : Pair
+ verifyTicketToPeer(String, byte[], byte[]) : Pair
+ verifyVotingTicket(String, byte[], byte[]) : String

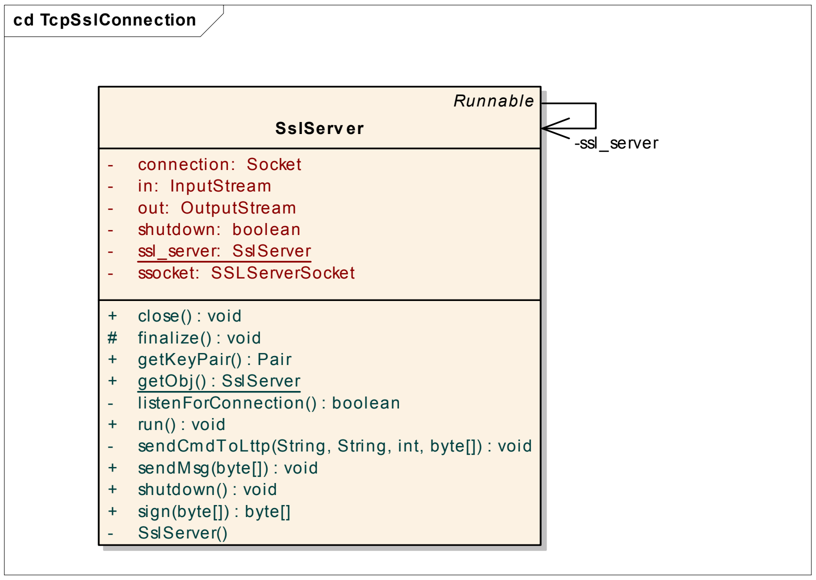
- **Class PeerDbApi**



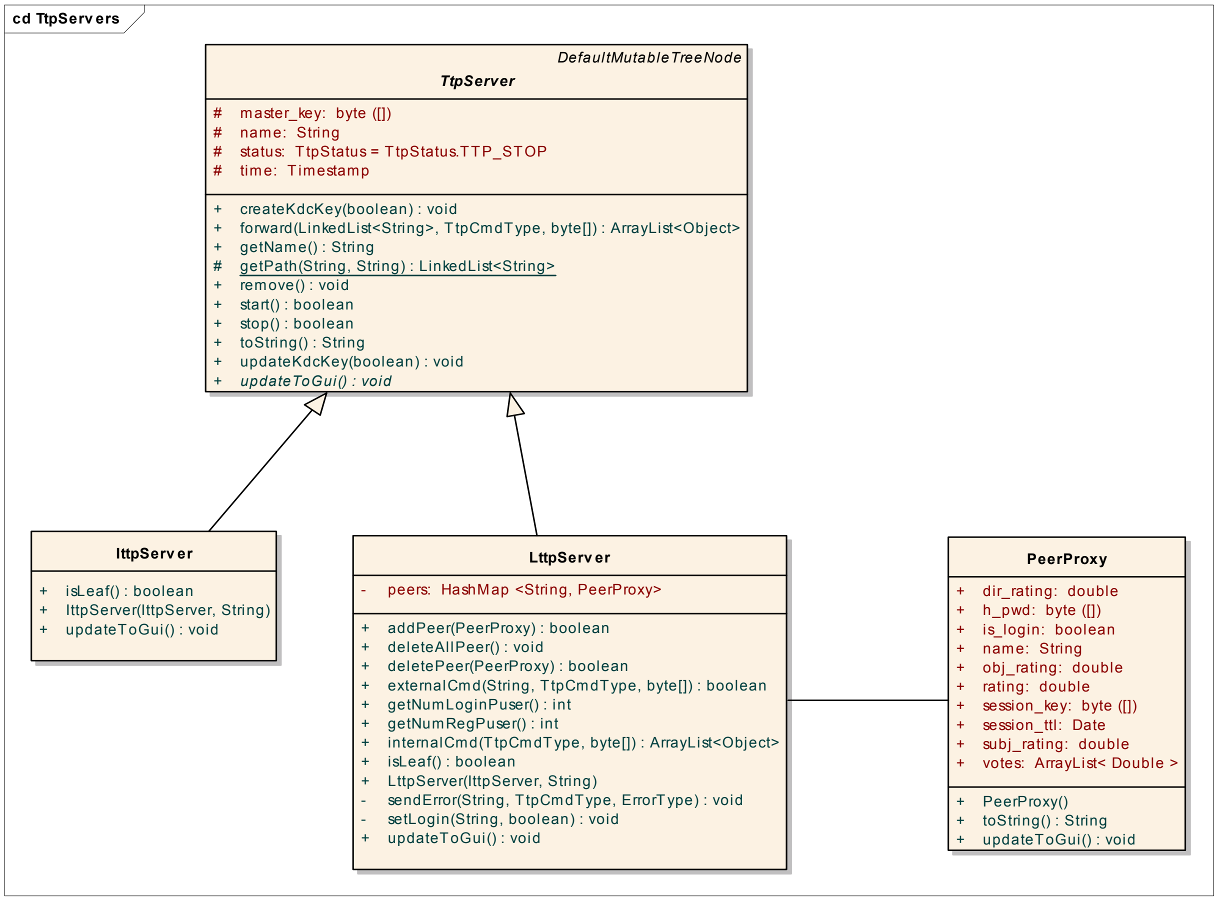
- **Class RatingApi**


cd Reputation
RatingApi
+ computeOverallRating(PeerProxy) : double
+ getRating(String) : double
+ setRating(String, double) : void
+ vote(PeerProxy, double) : void

▪ Class SslServer

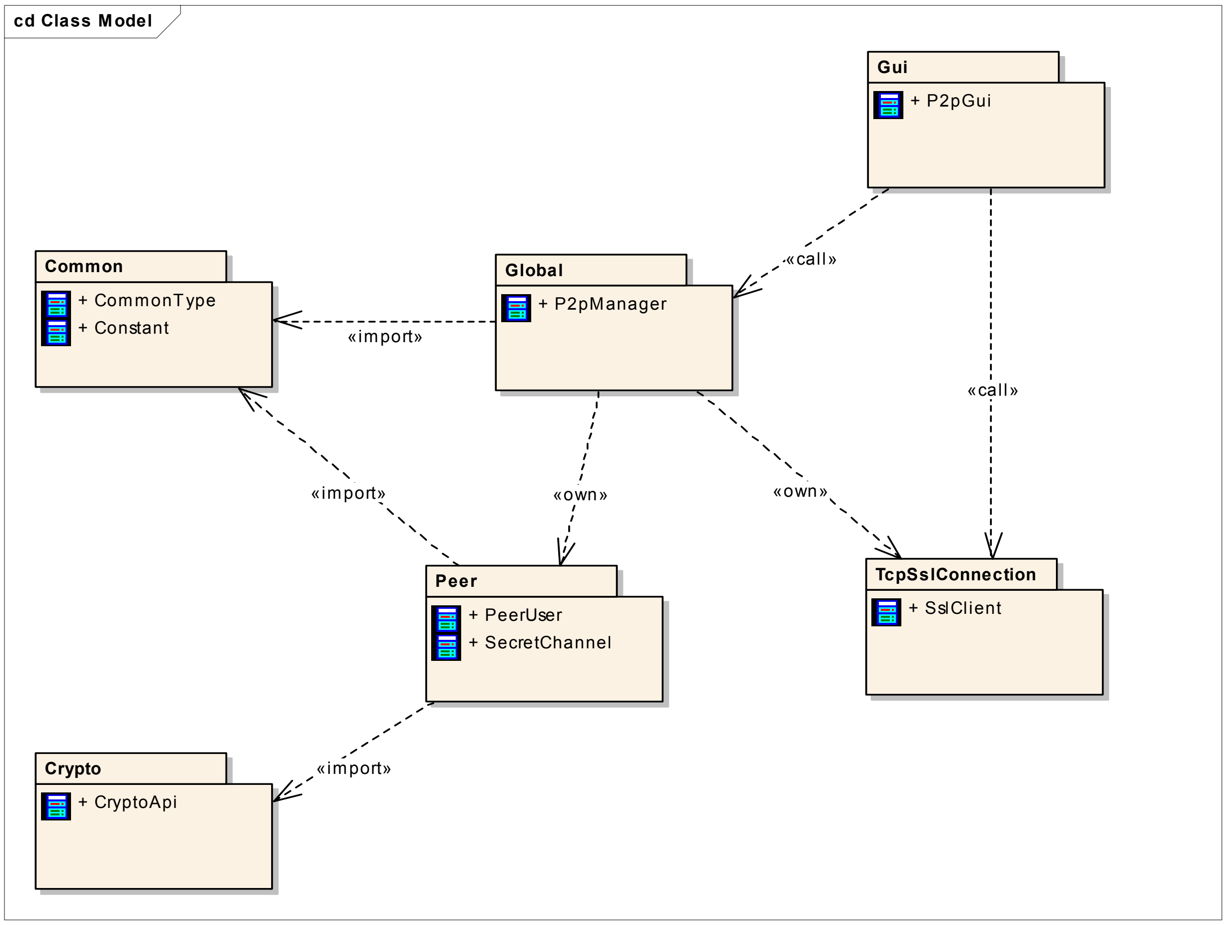


▪ Package TtpServers

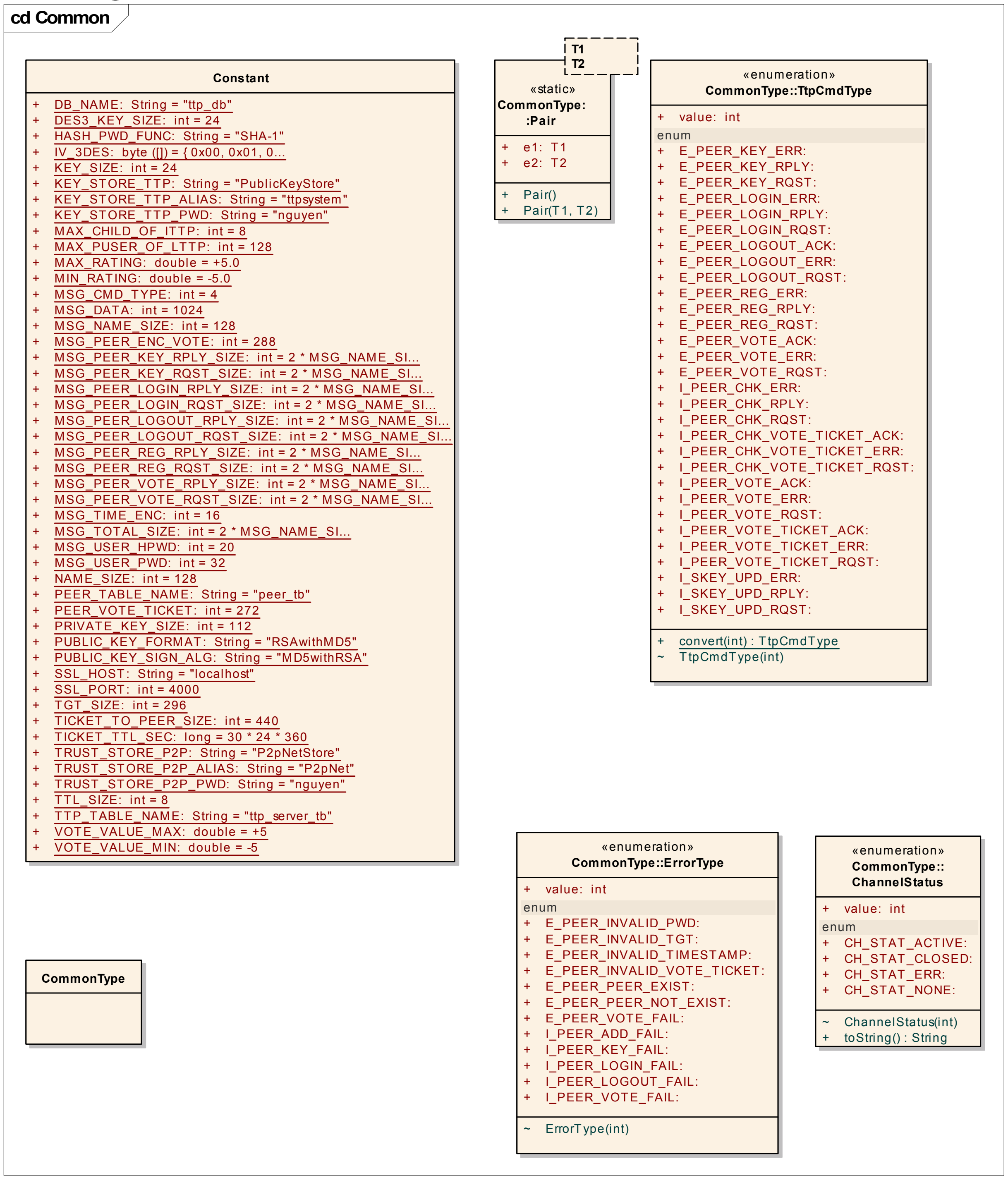

cd TtpServers
DefaultMutableTreeNode
TtpServer
# master_key: byte ([])
# name: String
# status: TtpStatus = TtpStatus.TTP_STOP
# time: Timestamp
+ createKdcKey(boolean) : void
+ forward(LinkedList<String>, TtpCmdType, byte[]) : ArrayList<Object>
+ getName() : String
# getPath(String, String) : LinkedList<String>
+ remove() : void
+ start() : boolean
+ stop() : boolean
+ toString() : String
+ updateKdcKey(boolean) : void
+ updateToGui() : void
IttpServer
+ isLeaf() : boolean
+ IttpServer(IttpServer, String)
+ updateToGui() : void
LttpServer
- peers: HashMap <String, PeerProxy>
+ addPeer(PeerProxy) : boolean
+ deleteAllPeer() : void
+ deletePeer(PeerProxy) : boolean
+ externalCmd(String, TtpCmdType, byte[]) : boolean
+ getNumLoginPuser() : int
+ getNumRegPuser() : int
+ internalCmd(TtpCmdType, byte[]) : ArrayList<Object>
+ isLeaf() : boolean
+ LttpServer(IttpServer, String)
- sendError(String, TtpCmdType, ErrorType) : void
- setLogin(String, boolean) : void
+ updateToGui() : void
PeerProxy
+ dir_rating: double
+ h_pwd: byte ([])
+ is_login: boolean
+ name: String
+ obj_rating: double
+ rating: double
+ session_key: byte ([])
+ session_ttl: Date
+ subj_rating: double
+ votes: ArrayList< Double >
+ PeerProxy()
+ toString() : String
+ updateToGui() : void

A.2.2 The P2P Network's UML Diagrams

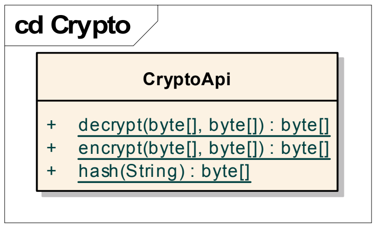
- **The P2P Network Package Model**



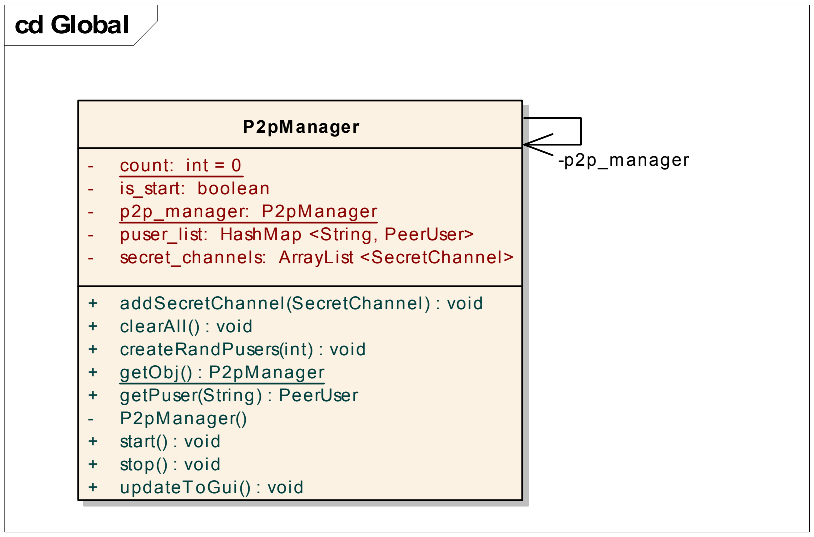
▪ **Package Common**


cd Common
Constant
+ DB_NAME: String = "ttp_db"
+ DES3_KEY_SIZE: int = 24
+ HASH_PWD_FUNC: String = "SHA-1"
+ IV_3DES: byte ([]) = { 0x00, 0x01, 0...
+ KEY_SIZE: int = 24
+ KEY_STORE_TTP: String = "PublicKeyStore"
+ KEY_STORE_TTP_ALIAS: String = "ttpsystem"
+ KEY_STORE_TTP_PWD: String = "nguyen"
+ MAX_CHILD_OF_ITTP: int = 8
+ MAX_PUSER_OF_LTTP: int = 128
+ MAX_RATING: double = +5.0
+ MIN_RATING: double = -5.0
+ MSG_CMD_TYPE: int = 4
+ MSG_DATA: int = 1024
+ MSG_NAME_SIZE: int = 128
+ MSG_PEER_ENC_VOTE: int = 288
+ MSG_PEER_KEY_RPLY_SIZE: int = 2 * MSG_NAME_SI...
+ MSG_PEER_KEY_RQST_SIZE: int = 2 * MSG_NAME_SI...
+ MSG_PEER_LOGIN_RPLY_SIZE: int = 2 * MSG_NAME_SI...
+ MSG_PEER_LOGIN_RQST_SIZE: int = 2 * MSG_NAME_SI...
+ MSG_PEER_LOGOUT_RPLY_SIZE: int = 2 * MSG_NAME_SI...
+ MSG_PEER_LOGOUT_RQST_SIZE: int = 2 * MSG_NAME_SI...
+ MSG_PEER_REG_RPLY_SIZE: int = 2 * MSG_NAME_SI...
+ MSG_PEER_REG_RQST_SIZE: int = 2 * MSG_NAME_SI...
+ MSG_PEER_VOTE_RPLY_SIZE: int = 2 * MSG_NAME_SI...
+ MSG_PEER_VOTE_RQST_SIZE: int = 2 * MSG_NAME_SI...
+ MSG_TIME_ENC: int = 16
+ MSG_TOTAL_SIZE: int = 2 * MSG_NAME_SI...
+ MSG_USER_HPWD: int = 20
+ MSG_USER_PWD: int = 32
+ NAME_SIZE: int = 128
+ PEER_TABLE_NAME: String = "peer_tb"
+ PEER_VOTE_TICKET: int = 272
+ PRIVATE_KEY_SIZE: int = 112
+ PUBLIC_KEY_FORMAT: String = "RSAwithMD5"
+ PUBLIC_KEY_SIGN_ALG: String = "MD5withRSA"
+ SSL_HOST: String = "localhost"
+ SSL_PORT: int = 4000
+ TGT_SIZE: int = 296
+ TICKET_TO_PEER_SIZE: int = 440
+ TICKET_TTL_SEC: long = 30 * 24 * 360
+ TRUST_STORE_P2P: String = "P2pNetStore"
+ TRUST_STORE_P2P_ALIAS: String = "P2pNet"
+ TRUST_STORE_P2P_PWD: String = "nguyen"
+ TTL_SIZE: int = 8
+ TTP_TABLE_NAME: String = "ttp_server_tb"
+ VOTE_VALUE_MAX: double = +5
+ VOTE_VALUE_MIN: double = -5
T1
T2
«static»
CommonType:
:Pair
+ e1: T1
+ e2: T2
+ Pair()
+ Pair(T1, T2)
«enumeration»
CommonType::TtpCmdType
+ value: int
enum
+ E_PEER_KEY_ERR:
+ E_PEER_KEY_RPLY:
+ E_PEER_KEY_RQST:
+ E_PEER_LOGIN_ERR:
+ E_PEER_LOGIN_RPLY:
+ E_PEER_LOGIN_RQST:
+ E_PEER_LOGOUT_ACK:
+ E_PEER_LOGOUT_ERR:
+ E_PEER_LOGOUT_RQST:
+ E_PEER_REG_ERR:
+ E_PEER_REG_RPLY:
+ E_PEER_REG_RQST:
+ E_PEER_VOTE_ACK:
+ E_PEER_VOTE_ERR:
+ E_PEER_VOTE_RQST:
+ I_PEER_CHK_ERR:
+ I_PEER_CHK_RPLY:
+ I_PEER_CHK_RQST:
+ I_PEER_CHK_VOTE_TICKET_ACK:
+ I_PEER_CHK_VOTE_TICKET_ERR:
+ I_PEER_CHK_VOTE_TICKET_RQST:
+ I_PEER_VOTE_ACK:
+ I_PEER_VOTE_ERR:
+ I_PEER_VOTE_RQST:
+ I_PEER_VOTE_TICKET_ACK:
+ I_PEER_VOTE_TICKET_ERR:
+ I_PEER_VOTE_TICKET_RQST:
+ I_SKEY_UPD_ERR:
+ I_SKEY_UPD_RPLY:
+ I_SKEY_UPD_RQST:
+ convert(int) : TtpCmdType
~ TtpCmdType(int)
«enumeration»
CommonType::ErrorType
+ value: int
enum
+ E_PEER_INVALID_PWD:
+ E_PEER_INVALID_TGT:
+ E_PEER_INVALID_TIMESTAMP:
+ E_PEER_INVALID_VOTE_TICKET:
+ E_PEER_PEER_EXIST:
+ E_PEER_PEER_NOT_EXIST:
+ E_PEER_VOTE_FAIL:
+ I_PEER_ADD_FAIL:
+ I_PEER_KEY_FAIL:
+ I_PEER_LOGIN_FAIL:
+ I_PEER_LOGOUT_FAIL:
+ I_PEER_VOTE_FAIL:
~ ErrorType(int)
«enumeration»
CommonType::
ChannelStatus
+ value: int
enum
+ CH_STAT_ACTIVE:
+ CH_STAT_CLOSED:
+ CH_STAT_ERR:
+ CH_STAT_NONE:
~ ChannelStatus(int)
+ toString() : String
CommonType

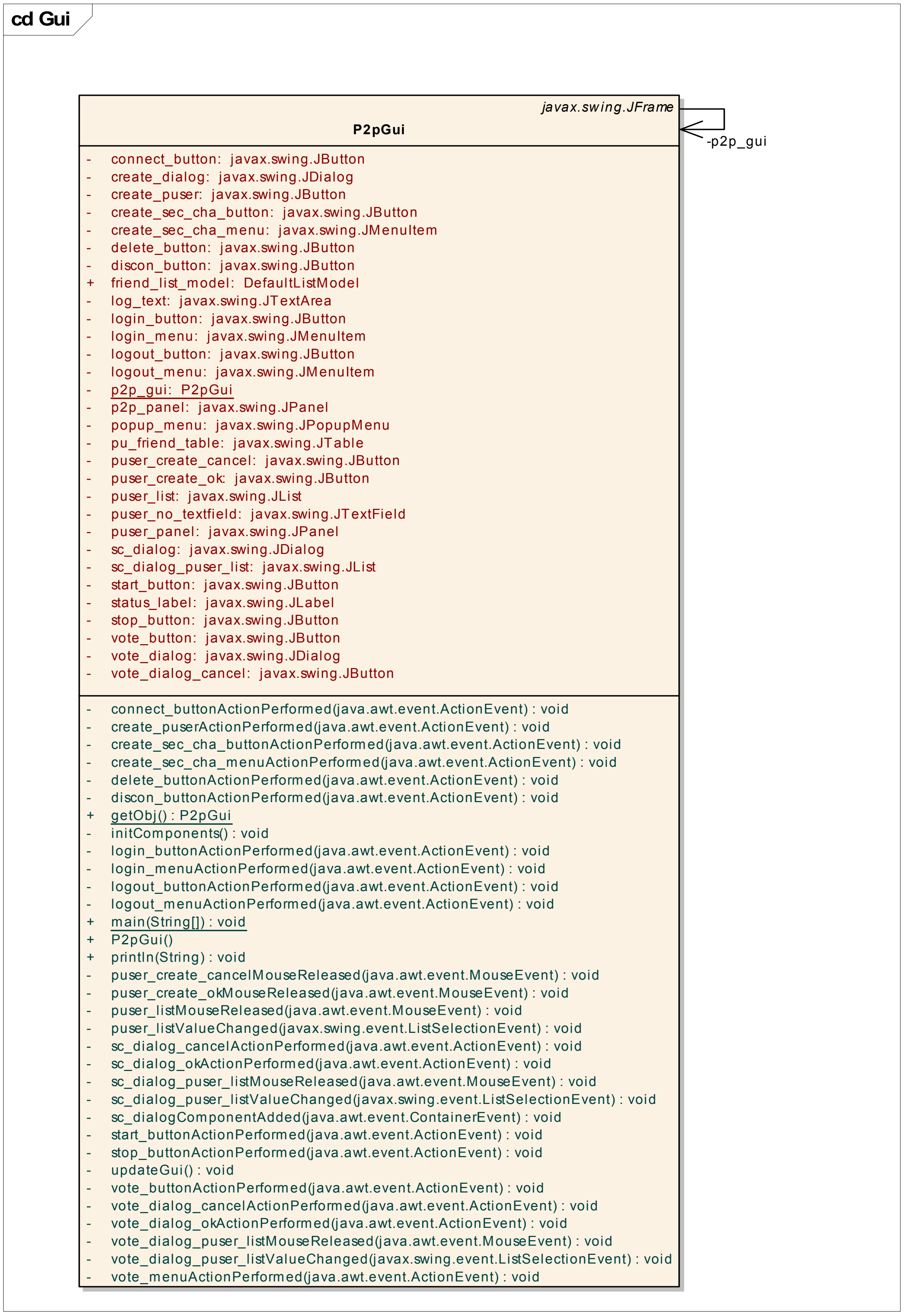
- **Class CryptoApi**


cd Crypto
CryptoApi
+ decrypt(byte[], byte[]) : byte[]
+ encrypt(byte[], byte[]) : byte[]
+ hash(String) : byte[]

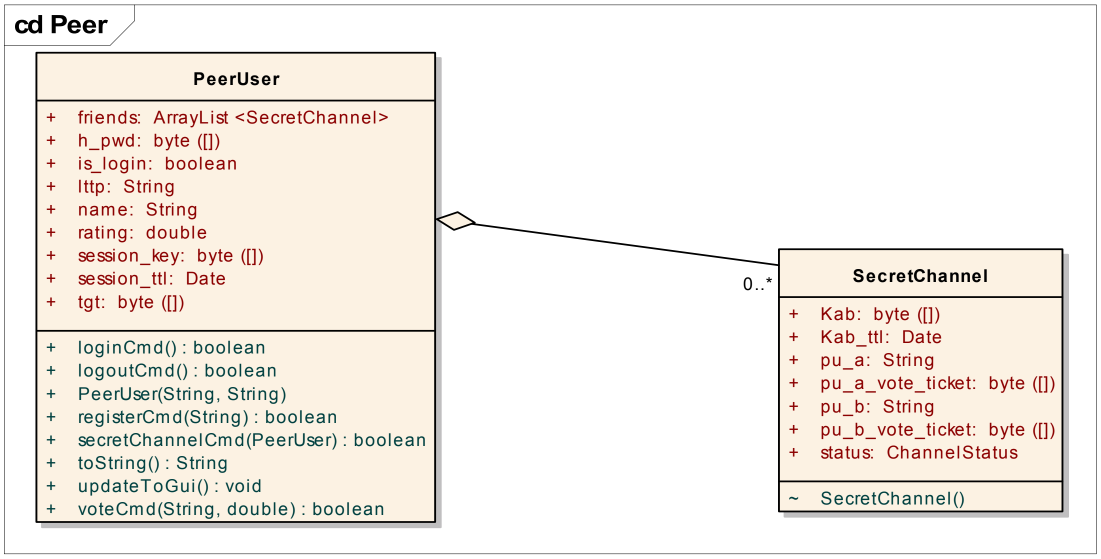
- **Class P2pManager**


cd Global
P2pManager
- count: int = 0
- is_start: boolean
- p2p_manager: P2pManager
- puser_list: HashMap <String, PeerUser>
- secret_channels: ArrayList <SecretChannel>
+ addSecretChannel(SecretChannel) : void
+ clearAll() : void
+ createRandPusers(int) : void
+ getObj() : P2pManager
+ getPuser(String) : PeerUser
- P2pManager()
+ start() : void
+ stop() : void
+ updateToGui() : void
-p2p_manager

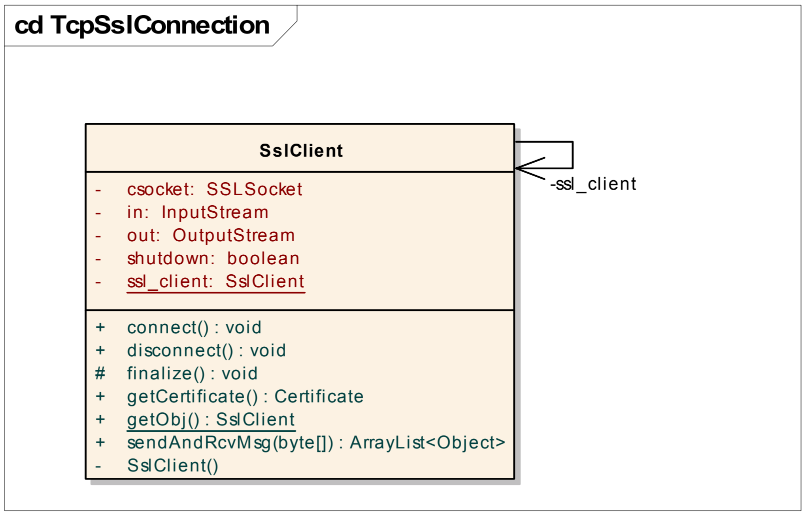
- **Class P2pGui**


cd Gui
javax.swing.JFrame
P2pGui
-p2p_gui
- connect_button: javax.swing.JButton
- create_dialog: javax.swing.JDialog
- create_puser: javax.swing.JButton
- create_sec_cha_button: javax.swing.JButton
- create_sec_cha_menu: javax.swing.JMenuItem
- delete_button: javax.swing.JButton
- discon_button: javax.swing.JButton
+ friend_list_model: DefaultListModel
- log_text: javax.swing.JTextArea
- login_button: javax.swing.JButton
- login_menu: javax.swing.JMenuItem
- logout_button: javax.swing.JButton
- logout_menu: javax.swing.JMenuItem
- p2p_gui: P2pGui
- p2p_panel: javax.swing.JPanel
- popup_menu: javax.swing.JPopupMenu
- pu_friend_table: javax.swing.JTable
- puser_create_cancel: javax.swing.JButton
- puser_create_ok: javax.swing.JButton
- puser_list: javax.swing.JList
- puser_no_textfield: javax.swing.JTextField
- puser_panel: javax.swing.JPanel
- sc_dialog: javax.swing.JDialog
- sc_dialog_puser_list: javax.swing.JList
- start_button: javax.swing.JButton
- status_label: javax.swing.JLabel
- stop_button: javax.swing.JButton
- vote_button: javax.swing.JButton
- vote_dialog: javax.swing.JDialog
- vote_dialog_cancel: javax.swing.JButton
- connect_buttonActionPerformed(java.awt.event.ActionEvent) : void
- create_puserActionPerformed(java.awt.event.ActionEvent) : void
- create_sec_cha_buttonActionPerformed(java.awt.event.ActionEvent) : void
- create_sec_cha_menuActionPerformed(java.awt.event.ActionEvent) : void
- delete_buttonActionPerformed(java.awt.event.ActionEvent) : void
- discon_buttonActionPerformed(java.awt.event.ActionEvent) : void
+ getObj() : P2pGui
- initComponents() : void
- login_buttonActionPerformed(java.awt.event.ActionEvent) : void
- login_menuActionPerformed(java.awt.event.ActionEvent) : void
- logout_buttonActionPerformed(java.awt.event.ActionEvent) : void
- logout_menuActionPerformed(java.awt.event.ActionEvent) : void
+ main(String[]) : void
+ P2pGui()
+ println(String) : void
- puser_create_cancelMouseReleased(java.awt.event.MouseEvent) : void
- puser_create_okMouseReleased(java.awt.event.MouseEvent) : void
- puser_listMouseReleased(java.awt.event.MouseEvent) : void
- puser_listValueChanged(javax.swing.event.ListSelectionEvent) : void
- sc_dialog_cancelActionPerformed(java.awt.event.ActionEvent) : void
- sc_dialog_okActionPerformed(java.awt.event.ActionEvent) : void
- sc_dialog_puser_listMouseReleased(java.awt.event.MouseEvent) : void
- sc_dialog_puser_listValueChanged(javax.swing.event.ListSelectionEvent) : void
- sc_dialogComponentAdded(java.awt.event.ContainerEvent) : void
- start_buttonActionPerformed(java.awt.event.ActionEvent) : void
- stop_buttonActionPerformed(java.awt.event.ActionEvent) : void
- updateGui() : void
- vote_buttonActionPerformed(java.awt.event.ActionEvent) : void
- vote_dialog_cancelActionPerformed(java.awt.event.ActionEvent) : void
- vote_dialog_okActionPerformed(java.awt.event.ActionEvent) : void
- vote_dialog_puser_listMouseReleased(java.awt.event.MouseEvent) : void
- vote_dialog_puser_listValueChanged(javax.swing.event.ListSelectionEvent) : void
- vote_menuActionPerformed(java.awt.event.ActionEvent) : void

- **Package Peer**



- **Class SslClient**



cd TcpSslConnection
SslClient
- csocket: SSLSocket
- in: InputStream
- out: OutputStream
- shutdown: boolean
- ssl_client: SslClient
+ connect() : void
+ disconnect() : void
# finalize() : void
+ getCertificate() : Certificate
+ getObj() : SslClient
+ sendAndRcvMsg(byte[]) : ArrayList<Object>
- SslClient()
-ssl_client