SwissExchange
TNA Security Concept

- Draft Version -

Jung-Uh Yang
Microsoft Consulting Services
Strategic Enterprise Services

INHALTSVERZEICHNIS

1. INHALT

Eine wichtige Rolle im Rahmen der SwissExchange TNA-Lösung (Trading New Architecture) nimmt die Sicherheitsbetrachtung ein. Aus diesem Grunde geht folgendes Dokument auf die Sicherheitsaspekte dieser Lösung ein. Im Vordergrund steht die Integration von unterschiedlichen Sicherheitsmechanismen, die im Falle von Windows 2000 betriebssystemseitig zur Verfügung stehen.

Ausgehend von einer Gesamtbetrachtung der Lösung, wird auf die einzelnen Teilszenarien detailliert eingegangen.

Das TNA-Sicherheitskonzept beruht auf der Grundlage wohlbekannter Securityprinzipien wie

- Vertraulichkeit
- Datenintegrität
- Authentifikation
- Authorisierung

Die Einbindung in die bestehende SwissExchange Security Infrastruktur und ein Ausblick in Rahmen von Windows 2000 PKI (Public Key Infrastructure) sind ebenso Gegenstand dieser Untersuchung.

Die Security Entities der TNA-Lösung sind unten aufgeführt.

- Exchange System, **ES**, stellt das Back-End System der TNA Lösung dar und basiert auf Open VMS. Die Authentifikation gegen ES erfolgt z. Z. auf Basis von Kerberos V4. Die Kommunikation zum Trading System erfolgt über ein Gateway.
- Trading System, **TS**, beinhalten die Server Applikationen und basieren auf UNIX oder zukünftig auf Windows 2000. TS stellt wiederum in der Gesamtbetrachtung einen Client zum ES Back-End System dar.
- Trading System User Application, **TNA-GUI**, ist das Front-End oder Client zum TS. Implemtiert ist die TNA-GUI auf UNIX und NT4 Plattformen. Die Kommunikation zwischen TS und TNA-GUI wird über Lokale Netze (LAN) oder auch WAN Verbindungen realisiert, ebenso denkbar ist eine Kommunikation über gesicherte Dial-Up Verbindungen und über öffenliche Netze z. B. das INTERNET.

2. GESAMTBETRACHTUNG

Unter der Sicherheitsbetrachtung fällt die gesamte Kommunikationskette ausgehend vom TS-Client → TS → ES unter Berücksichtigung oben erwähnter Security Prinzipien, unterschiedlichen Kommunikationsinfrastrukturen und unterschiedlicher TS Szenarien. Dabei liegt der Fokus der Untersuchung bei der Kommunikation zwischen TS-Client ←→ TS.
Folgende TS Szenarien werden untersucht:

- SMTS (Single Member Trading System) – SWX und Member Bank
- MMTS (Multi-Member Trading System) – SWX, Member Bank und Sub-Member Banken

Single Member TS
TS Clients im LAN

2.1. Single Member TS - TS Clients im LAN

Das erste Szenario in Abbildung 2.1. beschreibt eine TNA-Lösung basierend auf einem Single Member TS Konzept. Hierbei wird das ES System von der SwissExchange gehosted. Alle weiteren Elemente von TNA verbleiben im Einzugsbereich der Member Bank. Die Anbindung des Gateways zum ES System erfolgt über einen WAN Link und DECnet kommt als Kommunikationsprotokoll auf der Verbindungsstrecke zum tragen. Die Protokollumsetzung von DECnet ←→ auf IP wird durch das Gateway ausgeführt. TS und TNA GUI sind auf Seiten der Member Bank über ein LAN miteinander eingebunden.

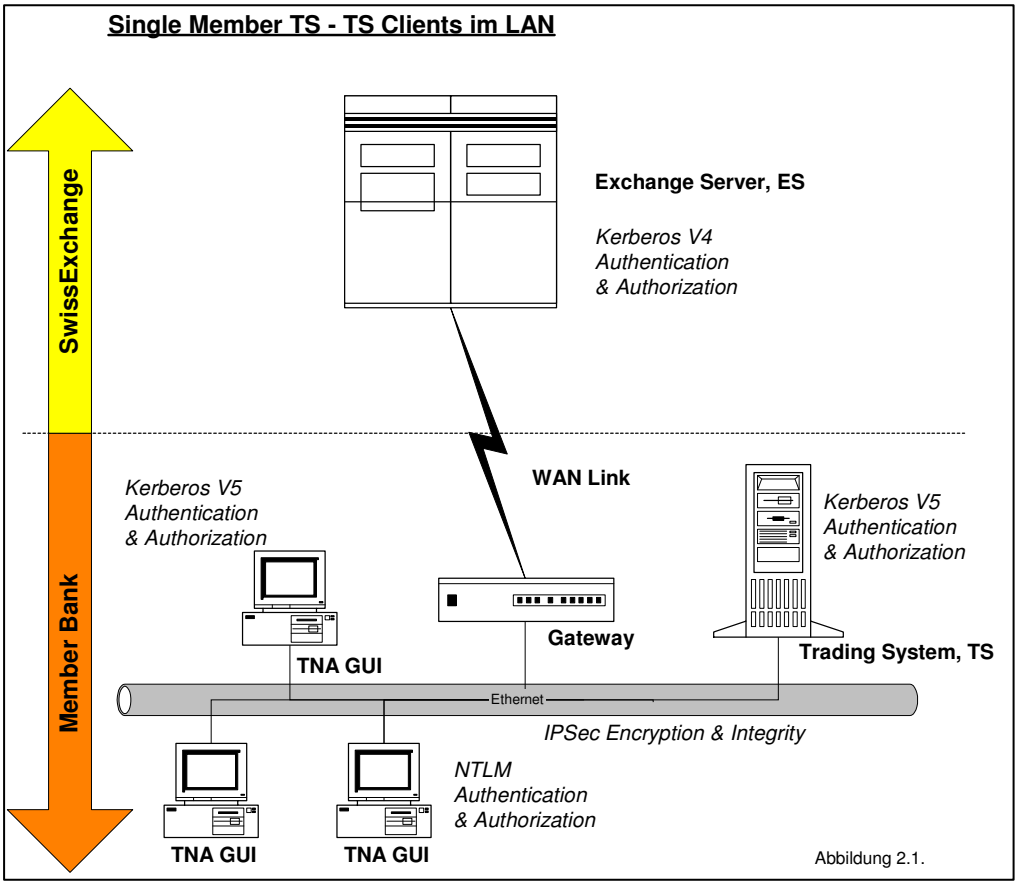


Abbildung 2.1.

Authentifikation[1]: Die Anmeldung des TS Systems zum ES basiert auf Kerberos V4 und erfolgt über die Member ID der Member Bank. Die Anmeldung der TNA Clients gegen das TS System wird über die User ID ausgeführt. Eine starker Authentifikationsmechanismus, Kerberos V5, wird im Fall von Windows 2000 als TS Plattform direkt unterstützt, sofern die TNA Clients auch auch Kerberos V5 unterstützen. Zur Zeit ist der TNA Client auf UNIX- und NT- Plattformen verfügbar, wobei eine Kerberos V5 Authentifikation noch nicht implementiert wurde. Durch die Abwärtskompatibilität von Windows 2000 erfolgt im Falle eines NT-TNA GUIs ein Fall-Back auf die gesicherte NTLM Authentifikation (NT Challenge Response Verfahren). Einschränkungen hinsichtlich NTLM zu Kerberos bestehen in der gegenseitigen Authentifikation (mutual Authentication). In NTLM Fall wird nur der Client gegenüber dem Server authentifiziert.
Eine Migration der TNA-GUI auf Windows 2000 Plattformen ist aus der Security Perspektive anzuraten um einen homogenen Anmeldungsprozess zu ermöglichen.

Vertraulichkeit und Integrität: Die für die Kerberos Authentifikation notwendigen Informationen werden verschlüsselt zwischen den Security Entities ausgetauscht. Damit ist die Einhaltung der Vertraulichkeitsbedingung erfüllt. Die NT Challenge Response Anmeldung erfolgt ebenso vertraulich, da der Anmeldeprozess über Verschlüsselungsverfahren verstärkt werden kann.
Den eigentlichen Datenverkehr[2] zwischen TNA-GUI und TS System im LAN, wie *Order Entries* oder *Broadcast Data*, kann über entsprechende Maßnahmen geschützt verlaufen. Mit IPSec existiert ein standardisiertes Verfahren um auf IP-Ebene den Datenverkehr zu verschlüsseln und zu authentifizieren. D. h. die Vertraulichkeit und Integrität der ausgetauschten Informationen wird mit IPSec erfüllt. Voraussetzung für IPSec ist, dass entsprechende Security Entities IPSec als Protokoll unterstützen müssen. IPSec ist ein integraler Bestandteil des Windows 2000 Betriebssystems und werden sowohl server- als auch clientseitig unterstützt.

[1] Im Sprachgebrauch der SwissExchange unterscheidet man zwischen einer technischen und businessorientierten Anmeldung. Bei der technischen Anmeldung werden Credentials zum Authentification Service weitergeleitet und authentifiziert. Wohingegen eine businessorientierte Anmeldung, technisch gesehen ein zusätzliches Attribut der Credentials darstellt, die für die Geschäftsanforderungen der SwissExchange herangezogen werden. So stellt die Anmeldung gegen ES mit der Member ID eine technische Anmeldung über Kerberos dar, im Gegensatz zu Sub-Member ID, Trader ID oder User ID, über die eine hohe Granularität hinsichtlich der erlaubten oder verweigerten Business Prozesse für jeden Teilnehmer erreicht wird.
2 Der Datenaustausch zwischen TS und ES muß über gegeignete Mechanismen im DECNet Umfeld realisiert werden oder auf Basis von möglichen Schutzmechanismen im WAN Umfeld. Dieser Aspekt ist nicht Fokus des Security Konzeptes.

Authorisierung[3]: Die Authorisierung, Rechtevergabe auf Resourcen und Ausführungrechte für die TNA Appliaktion selbst, sind zum einem über das Betriebssystem und zum anderen durch die Businesslogik der Applikation mit abgedeckt.
Im Windows 2000 TS System wird Zugriff auf Resourcen über das Active Directory gesteuert, wie z. B. den Aufbau von IPSec Verbindungen. Die Rechtevergabe auf TNA-Ebene wird durch den Einsatz von MTS Security Roles realisiert. Mit MTS Security Roles und *deklarativer Security* kann die Ausführung von COM Komponenten beschränkt werden, ist höhere Granularität erforderlich, so ist der Einsatz von *programmatischer security* zu empfehlen.

[3] Die Authorisierung auf dem ES System ist nicht Bestandteil dieser Untersuchung. Authorisierungsmechanismen sind in der Applikationslogik im ES verankert

2.2. Single Member TS - Remote TS Clients

In Abbildung 2.2. ist eine Variante des Single Member TS Konzept dargestellt. Wie im ersten Konzept schon dargestellt, wird das ES System von der SwissExchange gehosted. Alle weiteren Entities sind im Einzugsbereich der Member Bank angesiedelt. Der Hauptunterschied liegt in der Anbindung von TNA-GUIs zum TS System. Die TNA-GUIs sind hierbei über eine WAN Strecke oder über ein öffentliches Netz (INTERNET) mit dem Trading System der Member Bank verbunden.

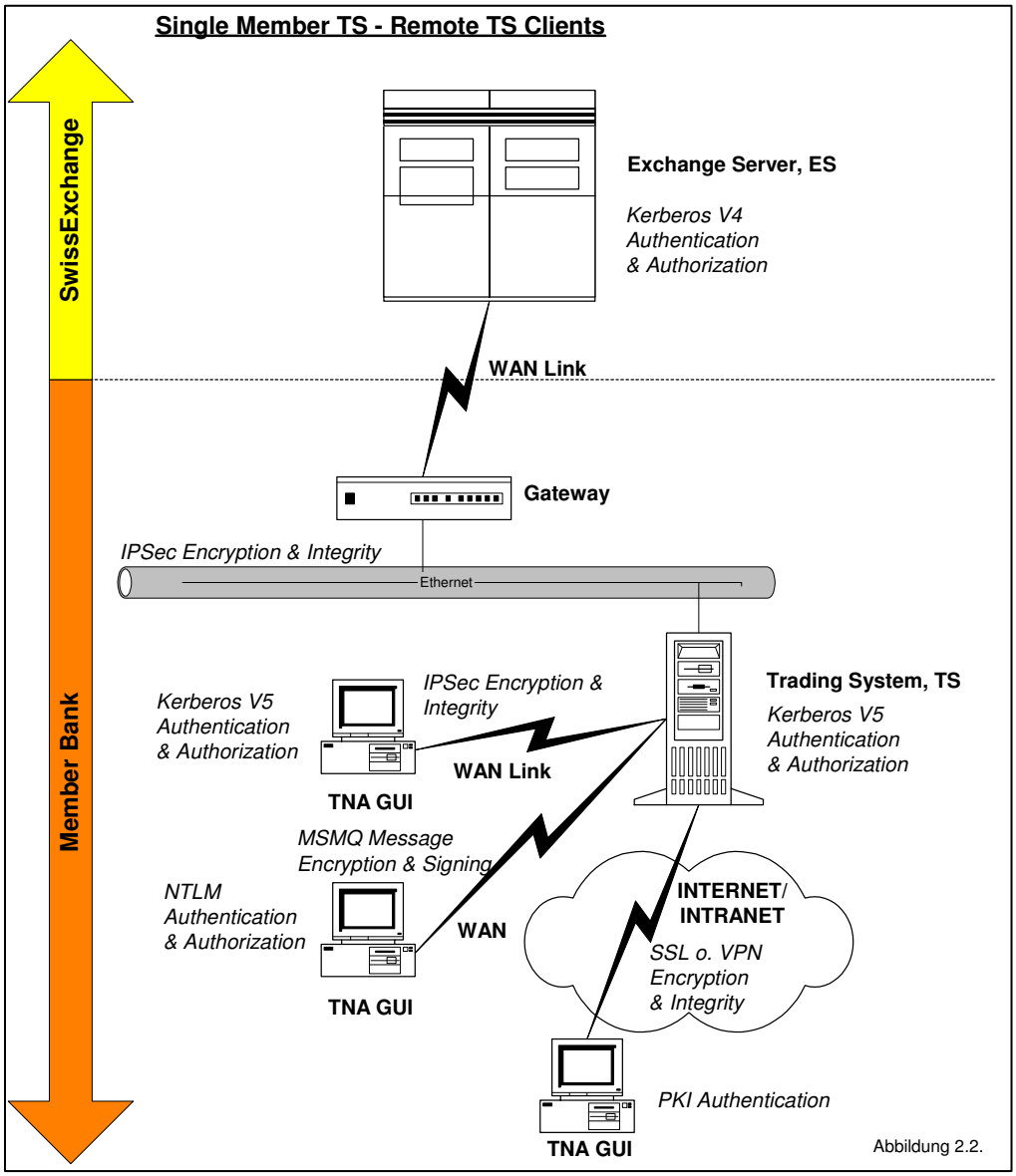


Abbildung 2.2.

Authentifikation: Die Authentifikation von TS zum ES wurde im ersten Konzept erläutert. Die Hauptunterschiede liegen im konkreten Fall bei den variablen Anmeldeldeszenarien von TNA Clients zum Single Member TS. Neben Kerberos basierenden Authentifikationsmechanismen ergibt sich die Einsatzmöglichkeit von zertifikatsabhängigen Anmeldeprozessen (PKI

Authentication). Hierbei findet eine gegenseitige Anmeldung von TS ←→ TNA GUI über X.509 Zertifikaten statt. Dieser Mechanismus ist im Internetumfeld etabliert. Voraussetzung hierfür ist eine gegenseitige Vertrauensstellung zu einer Third Party Instanz (Certificate Authority) oder kurz Public Key Infrastruktur-PKI. Der Aufbau einer PKI kann mit Windows 2000 Umgebungen realisiert werden.

Vertraulichkeit und Integrität: Der Unterschied zum ersten Konzept ist wiederum im Kommunikationsverkehr zwischen TNA Client und TS angesiedelt. Wie schon erwähnt, werden die notwendigen Authentication Informationen (Kerberos Tickets) verschlüsselt zwischen TNA GUI und TS ausgetauscht. Bei einer PKI Anmeldung wird die Vertraulichkeitsbedingung durch den Austausch von öffentlichen Zertifikaten und Verschlüsselung bzw. Entschlüsselung von Anmeldeinformationen durch den öffentlichen bzw. den korrespondierenden privaten Schlüssel erfüllt.
Zur Absicherung des Datenverkehrs kann IPSec herangezogen werden, oder auf Message Ebene im Falle von MSMQ, die einzelnen transaktionsorientierten Messages verschlüsselt und digital signiert werden. Im Internet Szenario stehen standardisierte Verschlüsselungsalgorithmen wie SSL (Secured Sockets Layer) oder TLS (Transprot Layer Security) zur Verfügung. Diese Protokolle bieten die Möglichkeit einer Verschlüsselung und Verifikation der Datenintegrität. Weitere Möglichkeiten exisitieren im Einsatz von VPNs (Virtual Private Networks) auf Basis von PPTP (Point-To-Point Tunneling Protocol) oder L2TP+IPSec (Layer 2 Tunneling Protocol). Eine Kombination der oben beschriebenen Verfahren ist ebenso denkbar, wie den Einsatz von VPN Lösungen und zusätzlich die Verschlüsselung und Signatur von MQ-Messages.

Authorisierung: Die Authorisierung unterscheidet sich nicht vom Konzeption 2.1. Ebenso muß betriebssystemseitig und in der Applikationslogik die Authorisierung adressiert werden.

Multi Member TS
Hosted by SwissExchange

2.3. Multi Member TS - Hosted by SwissExchange

Im Single Member TS Szenario wird allen TS Clients der Zugriff auf alle Informationen gewährt. Im Gegensatz dazu stehen in Multi Member TS Umgebungen bestimmte nur für den Sub-Member spezifizierte Informationen zur Verfügung. D. h. in der Regel handelt es sich nur um ein Subset von Informationen des Single Member, auf die der Sub-Member Zugriff hat.
Organisatorisch stellt die Multi-Member Lösung ein Trading System der SwissExchange dar, auf denen unterschiedliche Member Banken zugreifen oder alternativ das Trading System der Member Bank auf denen Sub-Member Banken derselben Member Bank Zugriff haben.

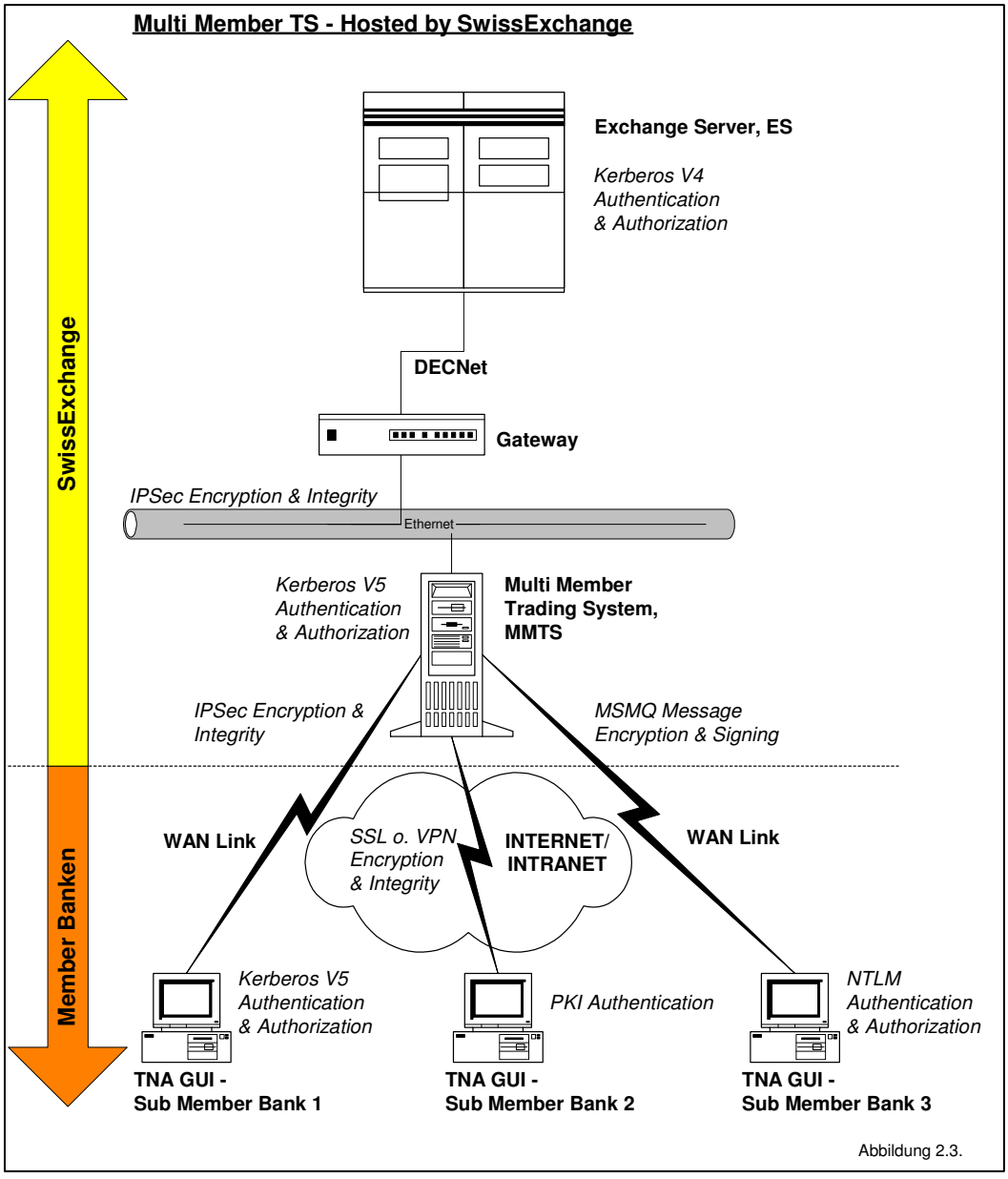


Abbildung 2.3.

Abbildung 2.3. ist die MMTS Lösung für den Zugriff unterschiedlicher Member Banken dargestellt. Die SwissExchange übernimmt das Hosting für das MMTS. Unterschiedliche Member

Banken greifen über die TS Clients auf das MMTS System der SwissExchange zu. Die Back-End Systeme, wie Gateway und ES, werden ebenfalls von der SwissExchange bereitgestellt und befinden sich in der SwissExchange LAN Umgebung.
Die TS Clients werden von den Member Banken betrieben. Die Anbindung an das MMTS System kann je nach bereitgestellter Kommunikationsinfrastruktur sich von den Member Banken untereinander unterscheiden. Eine TS Anbindung zum SwissExchange MMTS ist über ein WAN realisiert, aber auch zukünftige Anbindungen über das INTERNET sind denkbare Alternativen.

Authentifikation: Die Anmeldung des TS Systems zum ES basiert auf Kerberos V4 und erfolgt über die Member ID der Member Bank. Im Vergleich zur Single Member TS wird zusätzlich eine Sub-Member ID als Attribut mitgegeben um im nachfolgenden Athorisierungsprozess, spezifische Rechte und Informationen den unterschiedlichen Memberbanken bereit-zustellen.
Die Authentifikation der TS Clients gegen das MMTS der SwissExchange erfolgt analog wie im Kapitel 2.2. schon beschrieben. Auch hier können unterschiedliche Anmeldemechanismen zum tragen kommen, wie NTLM bei TS Clients auf Basis von NT und Windows 2000, PKI gestüzte Anmeldung bei MS und UNIX Clients und Kerberos V5 bei Windows 2000 und UNIX[4] Clients, wenn das Trading System auf Windows 2000 beruht.

Vertraulichkeit und Integrität: Wie in den voher erwähnten Szenarien findet auch hier der Authentifikationsverkehr verschlüsselt statt. Der Datenverkehr zwischen ES und TS wird verschlüsselt durchgeführt, wobei die Banken nur für sie relevante Informationen entschlüsseln können, da mit Hilfe der Sub-Member ID verschiedene Member Banken voneinander unterschieden werden können.
Für die Absicherung des TS-Client ←→ TS Datenverkehrs greifen die gleichen Mechanismen, wie sie im Kapitel 2.2. beschrieben wurden.

Authorisierung: Die Authorisierung unterscheidet sich nicht von Konzeption 2.1. Ebenso muß betriebssystemseitig und in der Businesslogik der Applikation die Authorisierung adressiert werden. Der Hauptunterschied besteht darin, dass in diesem Szenario unterschiedliche Member Banken auf das Trading System zugreifen. Durch die Vorgabe von Sub-Member ID, Trader ID und User ID wird die damit erforderliche Granularität der Authorisierung entsprechend angepasst.

[4] Kerberos Interoperabilität besteht zwischen MS Kerberos und den MIT Kerberos V5. Kerberos Client- und Serversoftware stehen als Freeware zum Download für unterschiedliche UNIX Plattformen bereit.

Multi Member TS
Hosted by Member Bank

2.4. Multi Member TS - Hosted by Member Bank

Abbildung 2.4. stellt eine weitere Variante des Mutli Member TS Konzeptes dar. Die SwissExchange betreibt nur das ES System. Aus SwissExchange Sicht besteht für dieses Szenario eine hohe Konformität zu den Single Member TS Szenarien in den Kapiteln 2.1. & 2.2.

Das Hosting für das MMTS und den Gateway wird von der (Main-) Member Bank bereitgestellt. Sub-Member Banken, die der eigentlichen Member Bank zugeordnet sind, aber nicht die gleiche Organisation darstellen, greifen mit ihren TS Clients auf das MMTS der Member Bank zu.

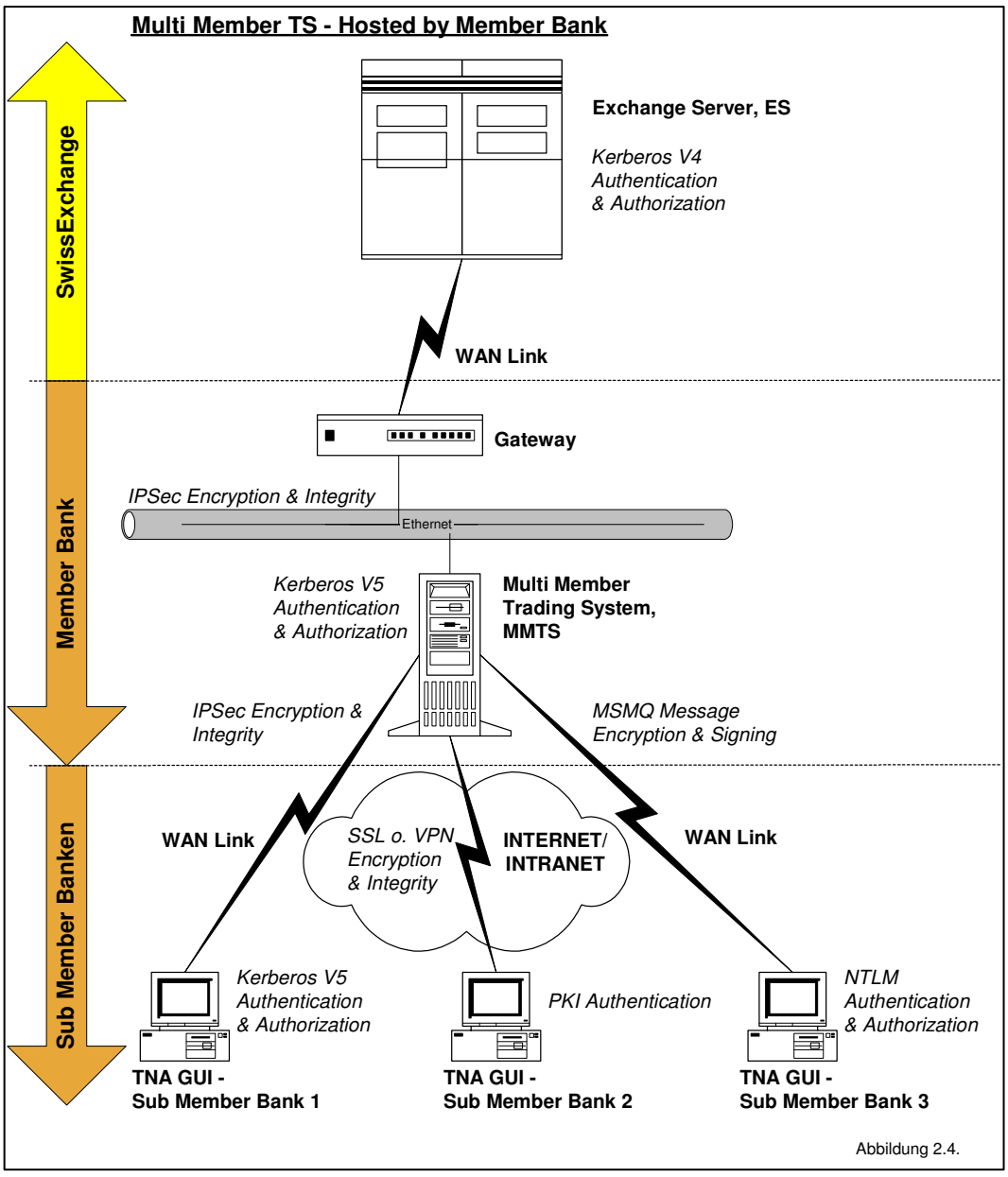


Abbildung 2.4.

Authentifikation: Der Authentifikationsprozess erfolgt analog, wie im Kapitel 2.3. schon beschrieben. Dies gilt sowohl für die Anmeldung von TS ←→ ES, als auch zwischen TS-Client←→ TS.

Vertraulichkeit und Integrität: Wie in den voher erwähnten Szenarien findet auch hier der Authentifikationsverkehr verschlüsselt statt. Der Datenverkehr zwischen ES und TS wird verschlüsselt durchgeführt, wobei die Sub-Member Banken nur für sie relevante Informationen entschlüsseln können, da mit Hilfe der Sub-Member IDs Sub-Member Banken voneinander unterschieden werden können.
Für die Absicherung des TS-Client ←→ TS Datenverkehrs greifen die gleichen Mechanismen, wie sie im Kapitel 2.2. beschrieben wurden.

Authorisierung: Die Authorisierung unterscheidet sich nicht von Konzeption 2.1. Ebenso muß betriebssystemseitig und in der Businesslogik der Applikation die Authorisierung adressiert werden. Der Hauptunterschied besteht darin, dass in diesem Szenario unterschiedliche Sub-Member Banken auf das Trading System zugreifen. Durch die Vorgabe von Sub-Member ID, Trader ID und User ID wird die damit erforderliche Granularität der Authorisierung entsprechend angepasst.

TS-CLIENT ←→ TS - AUTHENTICATION

3. TS-CLIENT ←→ TS – AUTHENTICATION

Beginn jeder Trading Session ist die Authentifikation. Aus Security Sicht stellt die Authentifikation das wichtigste Element im Security Framework dar. Ohne genaustens zu wissen welcher Trader im TS-Client/TS Szenario oder welche Bank im TS/ES Szenario miteinander kommunizieren, ist ein Einsatz von Verschlüsselungstechnologien oder einer nachfoglenden Rechtevergabe (Authorisierung, → wer soll denn nun authorisiert werden, wenn ich nicht weiß wer auf der anderen Seite ist?) nicht allzu sinnvoll. Aus besagtem Grunde wird ein großer Teil der Untersuchung diesem Thema gewidmet. Im Einzelnen handelt es sich um die unten aufgeführten Authentifikationsmechanismen, die von Windows 2000 Trading System direkt unterstützt werden:

- NTLM Authentication
- Kerberos Authentication
- PKI Authentication mit Zertifikaten (SSL)

TS-Client ←→ TS NTLM Authentication

3.1. TS-Client ←→ TS - NTLM Authentication

Sofern keine Portierung von Windows NT4 auf Windows 2000 auf der TS-Clientseite (TNA-GUI) erfolgt, werden die Trader über NTLM Challenge Response Verfahren gegen das Trading System (Windows 2000) authentifiziert. Vorteilhaft erweist sich, dass in diesem Umfeld kein Active Directory Service implementiert werden muß. Allerdings beschränkt sich die Anmeldung nur auf NT basierende TS-Clients. Demzufolge steht UNIX basierende TS-Clients dieser Mechanismus nicht zur Verfügung. Im Vergleich zu Kerberos unterstützt NTLM keine gegenseitige Authentifikation (mutual authentication), sondern beschränkt sich auf die Client Authentifikation.

Abbildung 3.0. veranschaulicht das NT Challenge Response Verfahren. Aus Security Sicht ist es dringend anzuraten Service Pack 4[5] auf den TS-Client zu installieren. Damit wird sichergestellt, dass eine Authentifikation mittels NTLM[6] so sicher wie möglich ausgeführt wird. Hierbei wird der gesamte Handshaking Prozess zwischen den Entities verschlüsselt durchgeführt.

[5] NTLMv2 ist verfügbar ab SP4 und erlaubt ein zusätzlich verschlüsseltes Handshaking (Challenge Response Verfahren) und das Ausschalten der Lan Manager Authentifikation.

[6] NTLM besteht im eigentlichen Sinne aus 2 unterschiedlichen Authentifikationsverfahren. Zum einen die NT Authentifikation zwischen NT Systemen, und zum anderen die LAN MANAGER Authentifikation zwischen NT und Win9.x//Win3.x um Abwärtskompatibilität zu gewährleisten. Dabei entsprechen, die im LM Umfeld eingesetzten Mechanismen, nicht mehr den heutigen Anforderungen. So wird bei der LM-Anmeldung nicht zwischen Upper-Case und Lower-Case Password Eingaben unterschieden. Außerdem werden veraltete Cryptoverfahren zur Bildung des Password-Hashs herangezogen, die zusammen mit den modernen NT-Hash in der SAM gespeichert werden. D.h. es sind immer zwei Passworte für einen User in der SAM abgelegt**. In einer homogenen NT bzw. W2K Umgebung ist dringenst anzuraten, die LM Authentifikation auszuschalten.**
Ebenso sollte die SAM Userdatenbank mit SYSKEY (ab Service Pack 3) verschlüsselt werden, um sich vor sogenannten Hacker/Admin Tools wie „L0phtCrack, -Dictionary Attacks“ zu schützen.

Windows NT speichert die UserID als SID (Security Identifier) und das dazugehörige Password in der SAM (Security Account Manager) Datenbank. Dabei enthält die SAM nicht das eigentliche Password, anstatt dessen ein 32-Byte langes kryptographischen Hashwert. Der Anmeldeprozess (NT Challenge Response) zwischen TS-Client und Trading System erfolgt in mehreren Schritten, die unten aufgeführt sind:

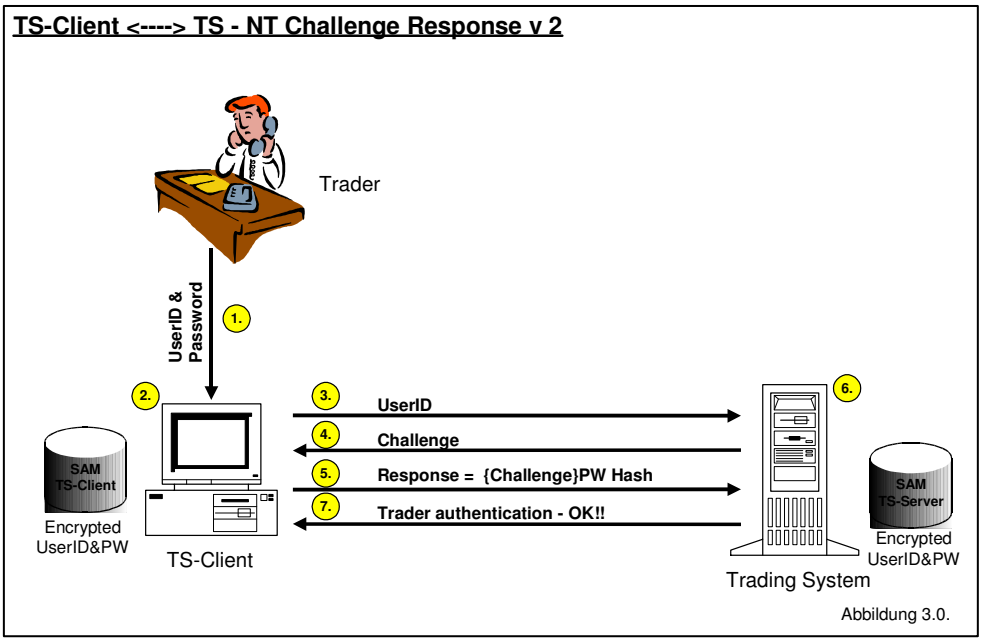


Abbildung 3.0.

1.) Der Trader gibt auf der TNA-GUI seine UserID und Password ein.
2.) Der TS-Client generiert aus dem eingegebenen Password einen kryptographischen Hashwert.
3.) Der TS-Client sendet die UserID zum Trading System
4.) Der TS erzeugt eine 16 Byte lange Zufallszahl, die auch Nonce oder Challenge genannt wird und versendet diese an den TS-Client.
5.) Der TS-Client verschlüsselt den Challenge mit den Hashwert der UserID und schickt diese als Response zurück an den TS-Server.
6.) Der TS-Server sucht das korrespondierende Password zu dem zuvor erhaltenen UserID in der eigenen SAM. Danach verschlüsselt der TS-Server mit dem Hash seine eigene Challenge. Dieses wird mit der Response vom TS-Client verglichen.
7.) Stimmen TS-Client Response mit dem lokalen Ergebnis, das der TS-Server generiert hat, überein, so ist der Trader gegenüber dem TS-Server erfolgreich authentifiziert. Als Bestätigung wird dem TS-Client die erfolgreiche Authentifikation mitgeteilt.

3.2. TS-Client ←→ TS - Kerberos Authentication

Das folgende Kapitel beschäftigt sich mit der Kerberos Authentifikation der TNA-Lösung. Kerberos V5 wird vom Windows 2000 System *Abbildung 3.1.* als primärer Authentication Provider unterstützt. Jeder Windows 2000 Domain Controller ist ein Kerberos Authentication Server oder auch Key Distribution Center. Bedauerlicherweise wird Kerberos nicht von NT Plattformen unterstützt. Beim Anmelden einer NT4 TNA-GUI gegen einen Windows 2000 Server findet daher ein Fall-Back auf den NT üblichen Authentication Provider NTLM Challenge Resonse statt. Daher ist eine Migration der TS-Clients von NT auf Windows 2000 zu empfehlen. Im UNIX Umfeld stehen Kerberos v5 Clients zum Download bereit. Interoperabilität zwischen MS Kerberos und MIT v5 Kerberos ist vorgesehen und dokumentiert im MS Kerberos Interop Whitepaper.

TS-Client <----> TS - Kerberos V5 Authentication

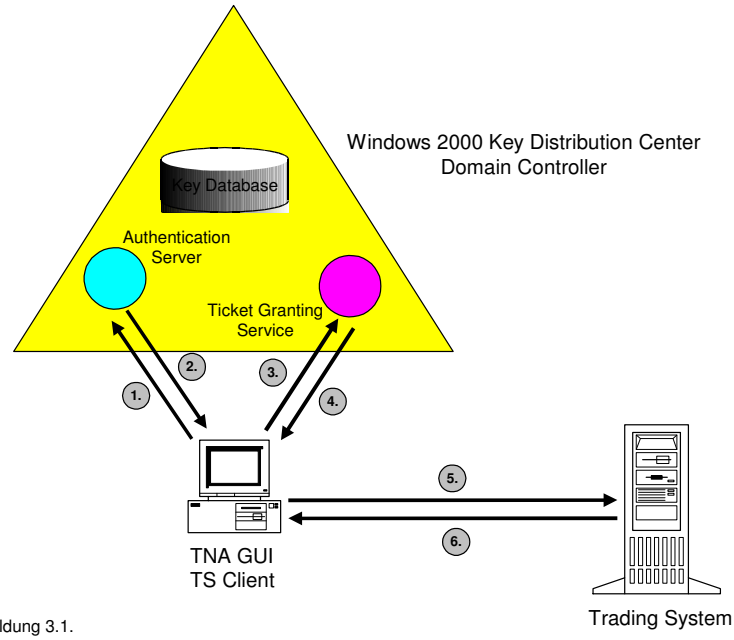


Abbildung 3.1.

Der TS-Client **1.** fordert zuerst ein TGS Ticket an. Als Antwort sendet der Kerberos Authentication Service dem TS-Client **2.** ein TGS Ticket (**Ticket Granting Ticket**) und einen TGS Session Key. Im nächsten Schritt fordert der TS-Client **3.** vom Ticket Granting Server ein Ticket für den Trading Server (**Service Ticket**) an. Die Antwort des TGS **4.** beinhaltet das Trading Server Ticket und den Trading Server Session Key. Das Trading Server Ticket und seinen **Authenticator** leitet der TS-Client **5.** weiter an den Trading Server. Auf Basis der vorliegenden Informationen kann sich der TS-Client gegenüber dem Trading Server authentifizieren und dem TS-Client die angeforderten Dienste zur

Verfügung stellen. Optional kann auch der Trading Server gegenüber dem TS-Client **6.** authentifiziert werden.

Für den gesicherten Austausch von Session Keys werden **Kerberos Tickets** herangezogen. Neben den Session-Key beinhaltet das Ticket auch Informationen über den Client, sowie Server- bzw. TGS-Namen, Zeitstempel und Lebensdauer. Bei Ablauf der Lebensdauer können Tickets erneuert werden und sind in diesem Rahmen mehrfach benutzbar. Die begrenzte Lebensdauer von Tickets erhöht die Sicherheit, falls ein Ticket gestohlen wird. Da jedes Ticket über Timestamps ausgewiesen wird, ist eine zeitliche Synchronisation von Client und Serversystemen ein sehr wichtiger Faktor. Bei abweichenden Zeiten ist wird eine Authentifikation vom Kerberos System verweigert. ***Kerberos Tickets werden immer verschlüsselt versendet.***

Der **Kerberos Authenticator** dient zum Nachweis der Identiät einer Instanz (Client), die das Kerberos Ticket einem Verifier (Server) vorlegt und beinhaltet den Clientnamen, Clientadresse sowie einen Zeitstempel. Der Authenticator kann nur von der Instanz (Client) angefertigt werden, die im Besitz des Session Keys ist. ***Ebenso wie Tickets werden Kerberos Authenticator nur verschlüsselt übertragen.***

K_{TS}	**= Long Term Key des Trading Servers**
K_{TGS}	**= Long Term Key des Ticket Granting Servers**
K_{UserID}	**= Long Term Key des Traders oder Users**
$SK_{UserID/TGS}$	**= Session Key für die Session zw. Trader und TGS**
$SK_{UserID/TS}$	**= Session Key für die Session zw. Trader und Trading Server**
$T_{UserID/TGS}$	**= Kerberos Ticket vom Trader → TGS**
$T_{UserID/TS}$	**= Kerberos Ticket von Bob → Trading Server**
$A_{UserID/TGS}$	**= Kerberos Authentficator vom Trader → TGS**
$A_{UserID/TS}$	**= Kerberos Authentficator vom Trader → Trading Server**

Der Trader in *Abbildung 3.2.* gibt seinen Benutzernamen "UserID" auf seiner Workstation (TS-Client) ein **1a.** Damit identifiziert sich der Trader lokal auf der Workstation unter seiner eingegebenen UserID. Der TS-Client leitet **1b.** den Benutzernamen "UserID" und einen Request für eine TGS-Session an den KDC.

TS-Client <----> TS - Lokale Anmeldung & Ticket Granting Ticket

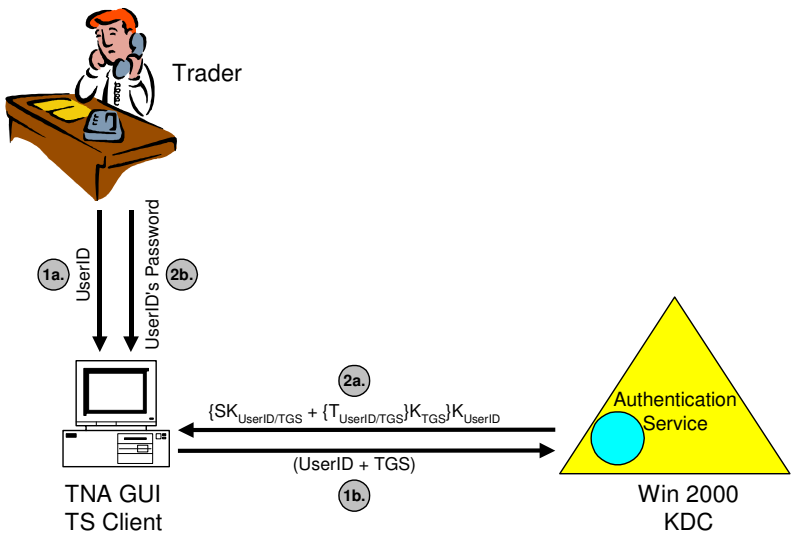


Abbildung 3.2.

Der KDC in **2a**[7]**.** erzeugt daraufhin einen Session Key: $\mathbf{SK_{UserID/TGS}}$, der nur dem Trader und dem Ticket Granting Server bekannt ist. Damit der Session Key $\mathbf{SK_{UserID/TGS}}$ verschlüsselt, sowohl an den Trader und auch an den Ticket Granting Server über das Netz ausgeliefert werden kann, wird ein Ticket für den TGS erzeugt, das den Session Key beinhaltet: $\mathbf{T_{UserID/TGS} = (TGS, UserID, UserID.NetAddr, timestamp, lifetime, SK_{UserID/TGS})}$.
Das Ticket wird verschlüsselt mit den Long Term Key des Ticket Granting Servers: $\mathbf{\{T_{UserID/TGS}\}K_{TGS}}$. Dadurch ist es nur dem TGS (auch nicht der Trader) möglich das Ticket zu lesen. Der KDC sendet den Session Key $\mathbf{SK_{UserID/TGS}}$ und das Ticket $\mathbf{\{T_{UserID/TGS}\}K_{TGS}}$ verschlüsselt mit dem Trader's Long Term Key $\mathbf{K_{UserID}}$ zurück an die Workstation:
$\mathbf{\{ SK_{UserID/TGS} , \{T_{UserID/TGS}\}K_{TGS} \}K_{UserID}}$
Der TS-Client generiert lokal aus dem UserID Password den Long Term Key $\mathbf{K_{UserID}}$ für den Trader in **2b**. Dadurch kann der TS-Client die vom Kerberos Server erhaltene Botschaft entschlüsseln.

Als Resultat besitzt der TS-Client:
a.) den Session Key für TGS-Session: $\mathbf{SK_{UserID/TGS}}$
b.) das Ticket Granting Ticket für die TGS: $\mathbf{\{T_{UserID/TGS}\}K_{TGS}}$

[7] die Klammern {…} um die Einträge bedeuten, daß diese verschlüsselt werden mit den nachfolgenden Schlüssel.

Abbildung 3.3. zeigt die Anforderung eines Trading Server Tickets vom TGS. Tickets für eine Client/Server Session, werden ausschließlich vom TGS ausgegeben. Der Request vom TS-Client für ein Server Ticket in **3.** enthält den Trading Servernamen **TS**, das TGT $\mathbf{\{T_{UserID/TGS}\}K_{TGS}}$ und den Authenticator für den Trader an die TGS

$\mathbf{A_{UserID/TGS} = (UserID, UserID.NetAddr, timestamp)}$,

der mit den gemeinsamen Session Key vom Trader und TGS verschlüsselt wird. $\mathbf{\{ A_{UserID/TGS} \}SK_{UserID/TGS}}$.

TS-Client <----> TS - Trading System Service Ticket

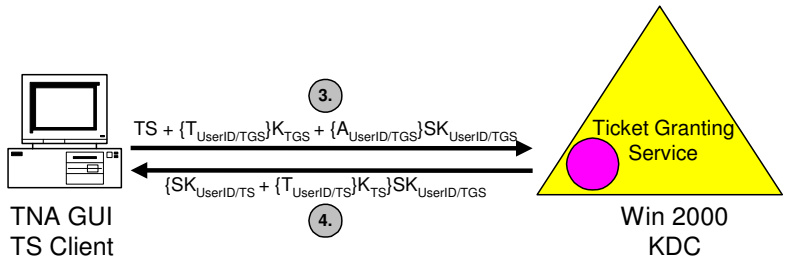


Abbildung 3.3.

Der Ticket Granting Server entschlüsselt zuerst das erhaltene Ticket Granting Ticket $\mathbf{\{T_{UserID/TGS}\}K_{TGS}}$ mit seinem Long Term Key $\mathbf{K_{TGS}}$. Das TGT enthält auch den Session Key $\mathbf{SK_{UserID/TGS}}$ für die Session Trader/TGS. Die Überprüfung des Authenticators dient der Feststellung ob tatsächlich der Trader den Request für eine TS-Client/Trading Server Session gesendet hat. Kann nun der TGS den Authenticator mit dem Session Key $\mathbf{SK_{UserID/TGS}}$ entschlüsseln, so stammt der Request tatsächlich vom Trader. Nur der Trader konnte diesen Request ausstellen, da nur er die Möglichkeit besitzt diesen Session Key $\mathbf{SK_{UserID/TGS}}$ zur Verschlüsselung seines Authentificators heranzuziehen; denn der Session Key $\mathbf{SK_{UserID/TGS}}$, enthalten in der verschlüsselten Botschaft des Kerberos Servers, konnte nur über über den Long Term Key $\mathbf{K_{UserID}}$ des Traders ausgelesen werden. Damit ist der Trader gegenüber dem Ticket Granting Server authentifiziert.

Der TGS generiert einen Session Key $\mathbf{SK_{UserID/TS}}$ für die TS-Client/Trading Server Session. Desweiteren generiert der TGS ein Trading Server Ticket.

$\mathbf{T_{UserID/TS} = (TS, UserID, UserID.NetAddr, timestamp, lifetime, SK_{UserID/TS})}$.

Das TS Ticket wird verschlüsselt mit den Long Term Key des Servers $\mathbf{\{T_{UserID/TS}\}K_{TS}}$. Der TGS sendet den Session Key $\mathbf{SK_{UserID/TS}}$ und das verschlüsselte Ticket $\mathbf{\{T_{UserID/TS}\}K_{TS}}$ verschlüsselt mit den gemeinsamen Session Key $\mathbf{SK_{UserID/TGS}}$ zurück zum TS-Client in **4.**

TS-Client entschlüsselt mit seinem vorliegenden Session Key $\mathbf{SK_{UserID/TGS}}$ die Antwort des TGS.

Der TS-Client ist nun im Besitz von:

a.) den Session Key für TGS-Session: $\mathbf{SK_{UserID/TGS}}$
b.) das Ticket Granting Ticket: $\mathbf{\{T_{UserID/TGS}\}K_{TGS}}$
c.) den Session Key für TS-Session: $\mathbf{SK_{UserID/TS}}$
d.) das TS Session Ticket: $\mathbf{\{T_{UserID/TS}\}K_{TS}}$

TS-Client <----> TS - Authentication Client & Server (optional)

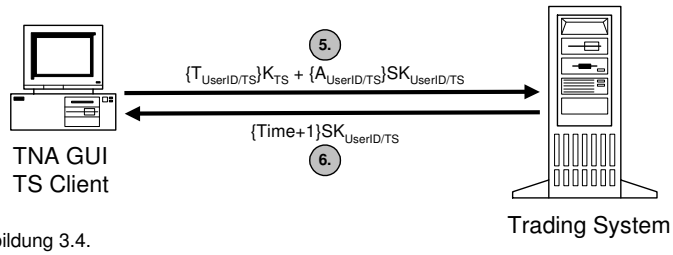


Abbildung 3.4.

In *Abbildung 3.4.* wird die Authentifikation des TS-Clients gegenüber dem Trading Server und die (optionale) Authentifikation des Trading Servers gegenüber dem TS-Client dargestellt. TS-Client sendet für die Authentifikation das TS-Ticket $\mathbf{\{T_{UserID/TS}\}K_{TS}}$ und den verschlüsselten Authenticator $\mathbf{\{A_{UserID/TS}\}SK_{UserID/TS}}$ zum Server in **5.** Der Trading Server entschlüsselt das erhaltene TS Session Ticket $\mathbf{\{T_{UserID/TS}\}K_{TS}}$ mit seinem Long Term Key $\mathbf{K_{TS}}$. Das TS Ticket enthält auch den Session Key $\mathbf{SK_{UserID/TS}}$. Wenn nun der Trading Server den verschlüsselten Authenticator $\mathbf{\{A_{UserID/TS}\}SK_{UserID/TS}}$ mit den gemeinsamen Session Key $\mathbf{SK_{UserID/TS}}$ entschlüsseln kann, so stammt der Authenticator tatsächlich vom TS-Client. Damit ist der TS-Client (Trader) gegenüber TS authentifiziert.

Optional wird auch eine Authentifizierung des Trading Servers gegenüber dem TS-Client ausgeführt in **6.**, indem der TS aus dem TS Session Ticket $\mathbf{T_{UserID/TS}}$ den Zeitstempel verschlüsselt mit den Session Key $\mathbf{SK_{UserID/TS}}$ an den TS-Client schickt. Der TS-Client entschlüsselt den Zeitstempel mit seinem vorliegenden Session Key. Da als einziges nur der TS über den Long Term Key $\mathbf{K_{TS}}$ verfügt, um das vom TS-Client zuvor versendete TS-Ticket $\mathbf{\{T_{UserID/TS}\}K_{TS}}$ zu entschlüsseln und die im Ticket enthaltenen Komponenten $\mathbf{\{Timestamp\}SK_{UserID/TS}}$ an den TS-Client zurückzusenden, kann es sich tatsächlich nur um diesen Server handeln. Damit ist auch der Trading Server gegenüber dem TS-Client authentifiziert.

TS-Client ←→ TS SSL Authentication

3.4. TS-Client ←→ TS - SSL Authentication

Ein alternatives Authentifikationsverfahren stellt die PKI basierende SSL Authentication dar. SSL baut auf Public Key Mechanismen auf. Als großer Vorteil gegenüber der Kerberos und NTLM Authentifikation erweist sich die Cross Plattform Unterstützung, als auch die im Protokoll verankerte Datenverschlüsselung. D. h. nicht nur MS-Plattformen, sondern auch UNIX Umgebungen können durch SSL authentifiziert werden. Nachteilig wirkt sich der beschränkte Einsatzbereich einer SSL Authentication aus, da nur spezifische Applikationen (z. B. Web-Browser) SSL direkt unterstützen, ebenso eine Trusted Third Party (CA) und Zertifikate notwendige Voraussetzung ist. Interessant ist daher das Verfahren für einen zukünftigen Ansatz der TNA-Lösung, die zusätzlich als Web-Anwendung realisiert wird.

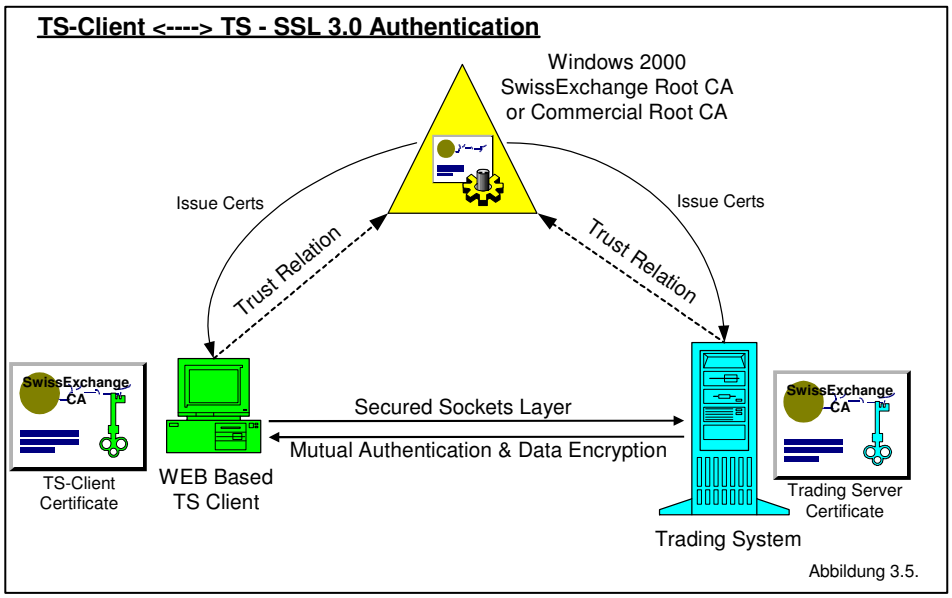


Abbildung 3.5.

In *Abblidung 3.5.* ist das SSL-Verfahren dargestellt. Sowohl der TS-Client als auch das Trading System müssen der gleichen CA[8] vertrauen, die das Serverzertifikat ausgestellt hat. Der Aufbau eines Trusts zur SwissExchange CA erfolgt mit der Akzeptanz von **Self Signed Certificates** dieser CA, die zusätzlich den Public Key der SwissExchange CA beinhaltet.

[8] Der Einsatz einer kommerziellen CA wie VeriSign hat den grossen Vorteil, daß die Trust Relationship zu den MS Produkten schon existiert. Wohingegen der Einsatz und Aufbau einer eigenen SwissExchange CA die Kontrolle für die Ausgabe von Zertifikaten, Revocation etc. im Einflußbereich der SwissExchange verbleibt. Auch sprechen Kostenfaktoren für den Aufbau einer eigenen CA.

Abbildung 3.6. verdeutlicht die Authentifikation des Trading Servers[9] gegenüber dem TS-Client. Im ersten Schritt **1.** (**Client Hello**) sendet der TS-Client eine Challenge Phrase (z. B. einen Text “abcdefg123”) an den TS-Server. Außerdem eine Liste von Cipher Algorithmen (RSA, DES, RC2/4, IDEA, MD5 etc.), die der Client für die Kommunikation nutzen möchte.

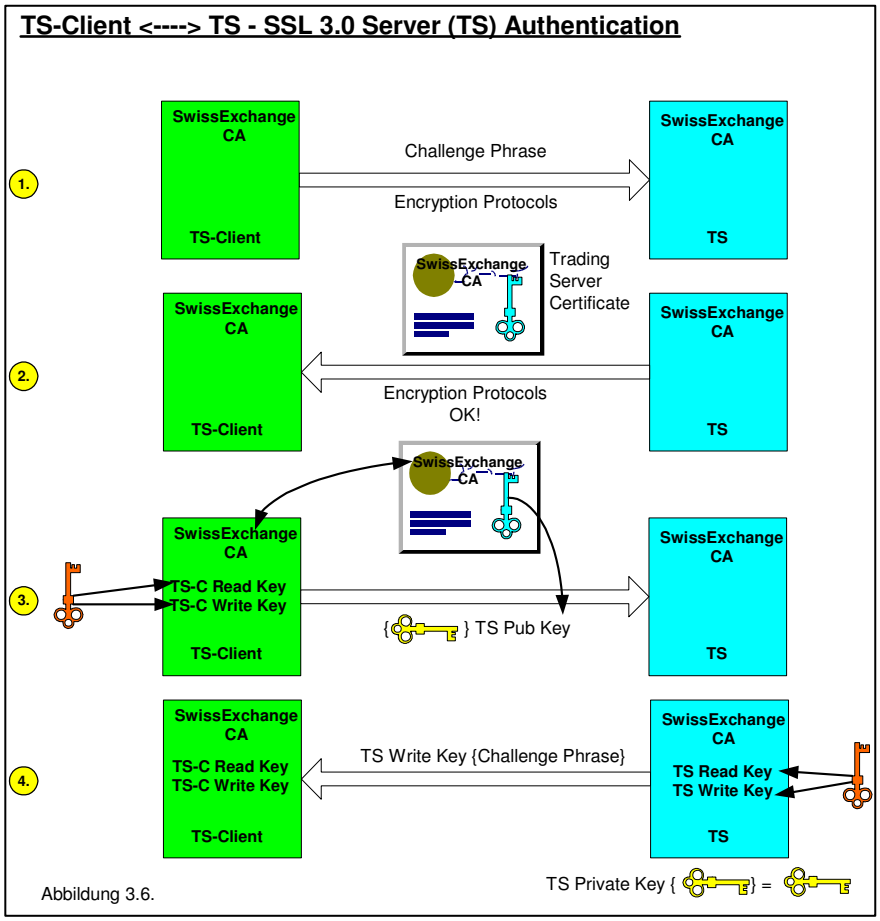


Abbildung 3.6.

Im zweiten Schritt **2.** (**Server Hello**) beantwortet der TS-Server ob die gewählten Cipher Algorithmen auch vom Server unterstützt werden. Desweiteren wird ein Serverzertifikat an den TS-Client weitergeleitet. Dieses Zertifikat beinhaltet den Public Key des TS-Servers und ist (Zertfikat+TS-Server Public Key) signiert mit den Privat Key der austellenden SwissExchange CA.

In Schritt **3.** wird das Zertifikat auf TS-Client Seiten validiert. Mit den Public-Key der SwissExchange CA wird die Signatur des Zertifikats überprüft. Ist die Signatur korrekt, so wurde das Serverzertifikat tatsächlich von der SwissExchange CA ausgestellt und die Inhalte (TS-Server Public Key) des Zertifikats wurden auch auf den Transport nicht verändert.

[9] Bei der SSL Authentifikation ist die Reihenfolge der authentifizierten Entities genau umgekehrt. Im Kerberos oder NTLM Fall wird zuerst der Client auhtentifiziert, wohingegen in SSL der Server authentifiziert wird. Dies liegt in der Natur des INTERNETS, wo man davon ausgeht, dass i. d. R. Web Server anonyme Entities darstellen.

Mit der Antwort von unterstützten Ciphern serverseitig wird auf der TS-Client Seite ein Zufallsschlüssel, oder **Master Session Key**, mit entsprechender Länge generiert. Dieser Master Key dient nicht zur Verschlüsselung von Daten. Vielmehr dient der Schlüssel als Basis für die Generierung von einem **symmetrischen Schlüsselpaar**, einem TS-Client Read Key und einem TS-Client Write Key. Das Schlüsselpaar wird zur eigentlichen Datenverschlüsselung herangezogen, wobei der TS-Client Read Key für den ankommenden und der TS-Client Write Key für den abgehenden Verkehr genutzt wird, was eine höhere Vertraulichkeit gewährleistet. Als Antwort sendet der TS-Client seinen Master Session Key, der verschlüsselt ist mit dem TS-Server Public Key, an den Server.

Im vierten Schritt **4.** (**Server Verify**) entschlüsselt der TS-Server mit seinen Private Key den Master Session Key. Wiederum dient der Master Session Key als Basis für die Erzeugung eines symmetrischen Schlüsselpaars. Diesmal ein TS-Server Read Key und ein TS-Server Write Key. Da das TS-Server Schlüsselpaar und das TS-Client Schlüsselpaar vom gleichen Master Session Key abgeleitet werden, korrespondieren beide Schlüsselpaare. Dies bedeutet der TS-Server Read Key entspricht dem TS-Client Write Key und der TS-Server Write Key entspricht dem TS-Client Read Key. Die im Vorfeld erhaltene Challenge Phrase “abcdefg123” wird mit den TS-Server Write Key verschlüsselt und an den TS-Client gesendet. Der TS-Client entschlüsselt die Challenge Phrase mit den TS-Client Read Key und vergleicht die vom TS-Server erhaltene Phrase mit der zuvor abgesendeten Phrase. Stimmen beide Phrasen überein ist der **Server gegenüber dem Client eindeutig authentifiziert** und der SSL Channel zwischen TS-Client/TS-Server ist aufgebaut. Ab diesem Zeitpunkt verläuft der Datentransfer verschlüsselt ab. Im Fall der SwissExchange ist aber auch die Client Authentication[10] notwendig, welches nachträglich ausgeführt wird.

Nach erfolgter TS-Server Authentifikation (s. *Abbildung 3.7.*), sendet nun der TS-Server **5.** eine Challenge Phrase#2 “abcd123Servertext”, verschlüsselt mit dem TS-Server Write Key, an den TS-Client.

Im nächsten Schritt **6.** wird auf der TS-Client Seite die Phrase#2 mit den TS-Client Read Key wieder entschlüsselt. Der Client setzt nun seine Antwort (Response Phrase) aus unterschiedlichen Komponenten zusammen. Zuerst wird ein Hash (Message Digest)

[10] Vorbedingung für die Client-Authentifikation ist ein Clientzertifikat, das vergleichbar eines Serverzertifikates von einer Trusted CA dem Client zur Verfügung gestellt wird. Wie schon erwähnt erfolgt Client-Authentifikation erfolgt immer nach der Server-Authentifikation. D. h. eine ausschließliche Client-Authentifikation wird in SSL nicht unterstützt. Die Einstellung ob eine SSL Verbindung nur mit Server-Authentifikation oder zusätzlich mit Client-Authentifikation aufgebaut werden soll, wird im WEB Server eingestellt.

gebildet aus der Phrase#2 und dem zuvor erhaltenen TS-Serverzertifikat. Der Hash wird danach mit dem TS-Client Private Key verschlüsselt. Das Resultat ist eine digitale TS-Client Signatur von der Phrase und dem Serverzertifikat. Die Signatur und das TS-Client Zertifikat werden anschließend mit den TS-Client Write Key verschlüsselt. Das Ergebnis daraus, die Response Phrase, wird nun an den TS-Server in Schritt **7.** versendet.

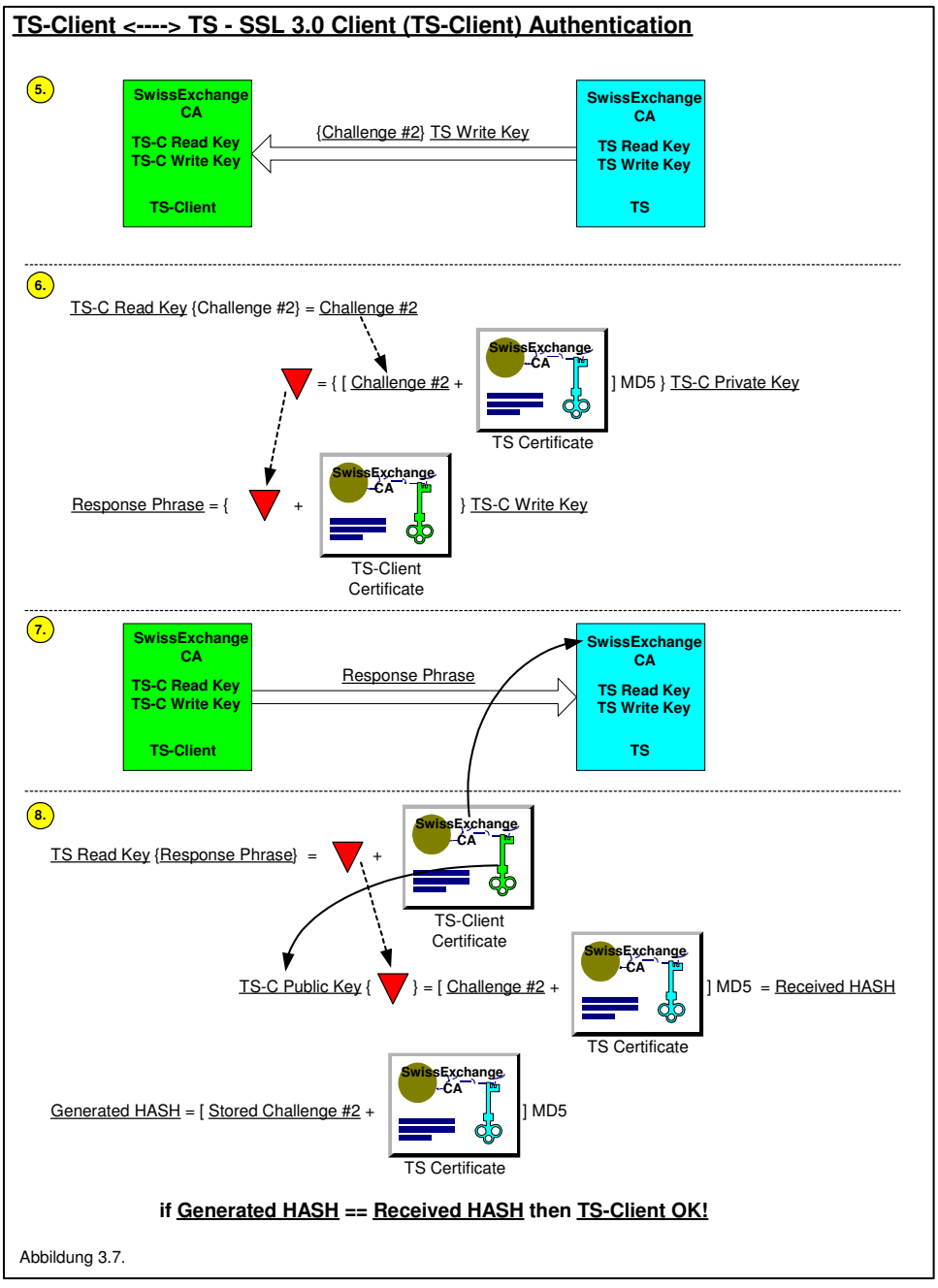


Abbildung 3.7.

Im letzten Schritt **8.** wird die Response Phrase mit den TS-Server Read Key entschlüsselt. Die digitale TS Client Signatur und das TS-Client Zertifikat liegen dem Server nun vor. Das TS Client Zertifikat wird vom Server validiert. Zum einem wird mit dem Public Key der SwissExchange CA die Signatur des TS-Client Zertifikats samt Inhalt überprüft, zum anderen wird mit den TS

Client Public Key die TS-Client Signatur entschlüsselt was als Resultat zu einem Hash (Received Hash) führt. Nun generiert der TS-Server lokal einen Hash (generated Hash) aus dem gespeicherten Challenge #2 Phrase und seinem TS-Server Zertifikat. Stimmen Received Hash und Generated Hash überein, so ist der TS-Client gegenüber dem TS-Server eindeutig authentifiziert.

4. TS-CLIENT ←→ TS VERSCHLÜSSELUNG & DATENINTEGRITÄT

In diesem Abschnitt liegt der Fokus auf den Schutzmechanismen für den Datenverkehr, der zwischen den Entities TS-Client und TS ausgetauscht wird. Die nachfolgend beschriebenen Security Verfahren schützen nicht nur den Datenverkehr im LAN Umfeld, sondern auch Remote Anbindungen die über ein WAN oder auch zukünftig über ein Intranet oder Internet realisiert werden. Folgende Mechanismen werden näher untersucht:

- IPSec
- VPNs und SSL
- MSMQ

TS-Client ←→ TS IPSec Encryption & Integrity

4.1. TS-Client ←→ TS – IPSec Encryption & Integrity

IPSec ist ein von der IETF definierter Standard für die sicherere verschlüsselte Übetragung von Daten auf der IP Ebene. Es ist ein Kozept für End-to-End Security, das für Anwendungen transparent ist, da die Security auf der Ebene des Transportprotokolls stattfindet. Daher ist IPSec für die TNA-Lösung ein wichtiger Ansatz um den Datenverkehr zwischen TS-Client und Trading System zu schützen. Die wichtigsten Funktionen beinhalten:

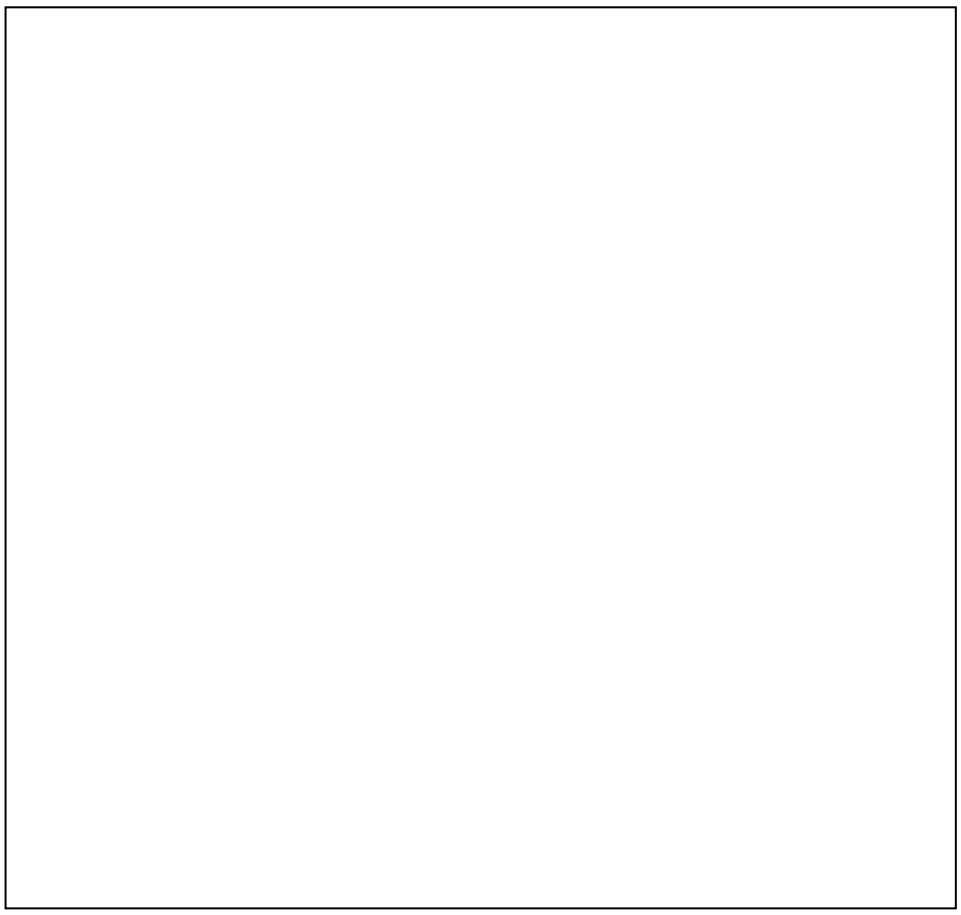
- Authentifikation der Kommunikationspartner (Maschinen Authentifikation)
- Integrität des Datenverkehrs[11]
- Verschlüsselung der übertragenen Information

Für die Verschlüsselung sind unterschiedliche Verfahren nutzbar. IPSec basiert auf Public-Key und symetrischen Session Key Mechanismen. Über die Public Keys werden die Session Keys gesichert ausgetauscht (vergleichbar mit dem SSL-Verfahren), wobei die Session Key die eigentliche Datenverschlüsselung durchführen.

Ein denkbares Szenario ist in *Abbildung 4.1.* dargestellt. In einem Single Member TS Umfeld haben die TS-Clients einen direkten Zugriff auf das TS System über das LAN. Voraussetzung einer IPSec Kommunikation ist, dass beide Entities (TS-Client und TS) IPSec unterstützen müssen. Im Fall der NT TS-Clients ist keine direkte Unterstützung von IPSec möglich, wohingegen IPSec vom Windows 2000 TS betriebssystemseitig unterstützt wird. Wiederum spricht es für eine Portierung der TNA-GUI auf Windows 2000. Alternativ kann über Third Party Produkte für NT IPSec verfügbar gemacht werden – ebenso zutreffend für UNIX Systeme ohne IPSec Unterstüzung. Ein weiterer Ansatz ist die

[11] Zur Verschlüsselung und Datenintegrität stehen IPSec-Modi, wie AH (Authentication Header) oder ESP (Encrypted System Payload) zur Verfügung. Verschlüsselungsstärken 40-Bit und höher werden durch unterschiedliche Cipheralgorithmen (DES, Triple-DES, RC...) erreicht. Die Authentifikation der IPSec-Endpoints kann über Kerberos, PKI (Zertifikate) und Shared Secrets erfolgen. Für den Fall einer PKI Authentifikation ist u. U. der Aufbau einer PKI erforderlich.

IPSec Kommunikation ausgeführt durch Router Systeme. Moderne Router Systeme, wie CISCO Router, sind IPSec fähig. Allerdings stellen die IPSec Endpoints Router ←→ TS dar, wobei der Verkehr zwischen TS-Client ←→ Router unverschlüsselt durchgeführt wird.



Unter der Annahme eines Windows 2000 TS-Clients wird die Verwendung von IPSec über Security Policies festgelegt. Diese werden im Active Directory definiert und an den lokalen Policy Agent des TS-Clients bzw. TS übergeben. Die Security Policies legen fest, mit welchem Security Level TS-Client und TS kommunizieren soll. Diese Informationen werden an den ISAKMP/Oakley[12]-Dienst und den IPSec Driver übergeben. Über ISAKMP/Oakley wird das zu verwendete Security Protokoll definiert. ISAKMP (Internet Security Association and Key Management Protocol) definiert auf welche Art und Weise die Security zwischen den beiden Systemen aufgebaut wird. Oakley ist ein Protokoll zuständig für die Festlegung, der für die Kommunikationssitzung benötigten Keys. Im Anschluß an den Verbindungsaufbau unter Verwendung von ISAKMP/Oakley erfolgt die eigentliche Kommunikation über den IPSec Driver. Dieser verschlüsselt und signiert die IP Pakete mit dem von ISAKMP/Oakley festgelegten Key.

[12] Eine weitere Bezeichnung für diesen Dienst ist IKE, Internet Key Exchange

4.2. TS-Client ←→ TS – Virtual Private Networks / SSL

Virtual Private Networks kurz VPNs unterscheiden sich grundsätzlich nicht zu SSL Anbindungen. Beide Verfahren bauen einen virtuellen Tunnel auf, in denen die Daten gesichert zwischen zwei Systemen übetragen werden. Die Unterschiede liegen auf welchem Layer die Tunnel aufgebaut werden. Im Falle von SSL findet der Tunnelaufbau im Application Layer statt, wohingegen im VPN Fall der Tunnelaufbau im Frame-Layer unterhalb der Netzwork Layers angesiedelt ist.
Beispielhaft ist das SSL- und VPN-Verfahren für das Multi-Member Szenario in *Abbildung 4.2. und 4.3.* dargestellt.

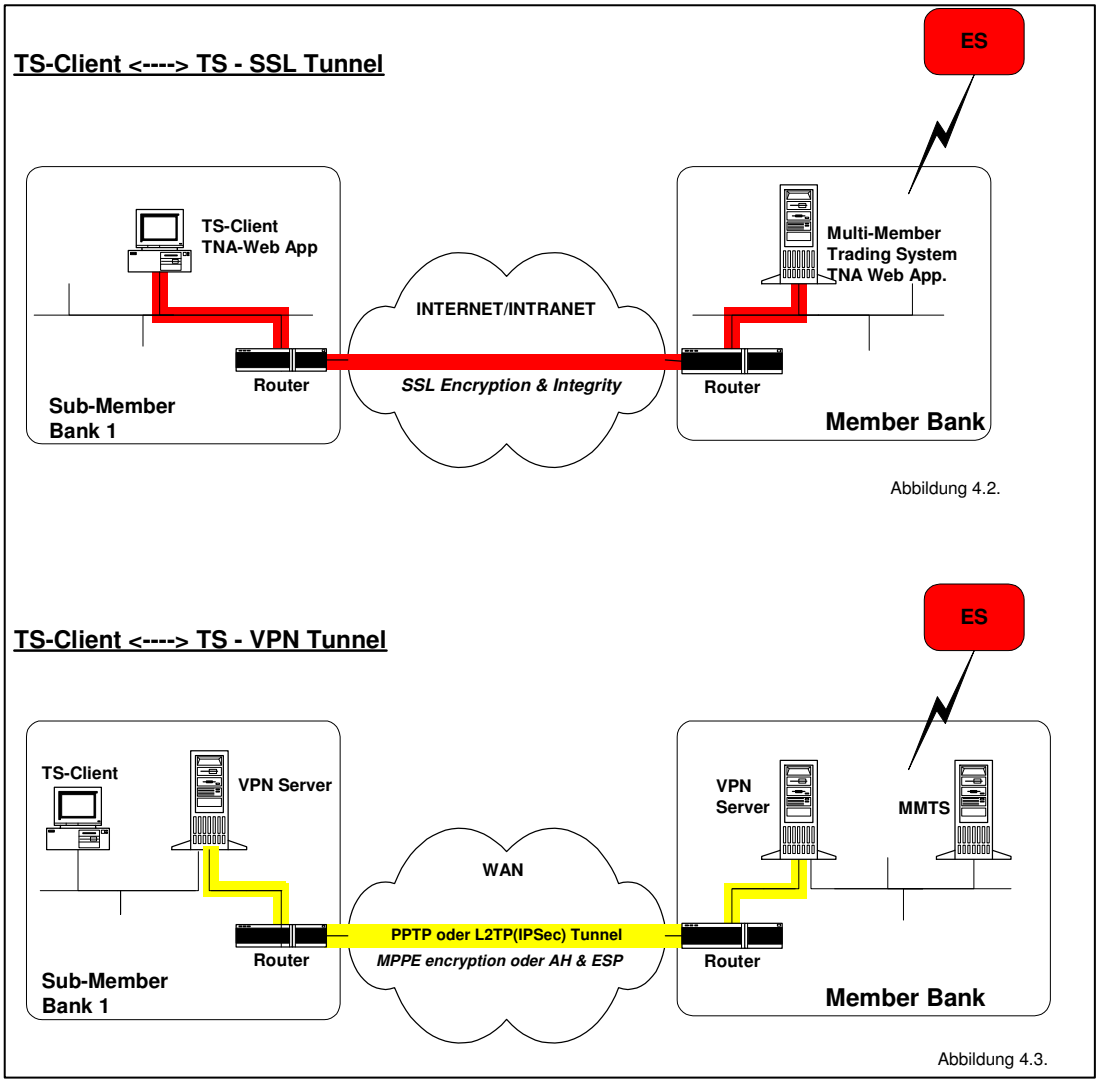


Abbildung 4.2.

Abbildung 4.3.

SSL-Tunnel:

Da SSL im Application Layer aufsetzt ist es notwendig für die TNA-Lösung diesem Umstand Rechnung zu tragen. Im Fall einer WEB basierten TNA Umsetzung kommt die SSL Verschlüsselung zum tragen. Der SSL Authentifikationsmechanismus wurde schon im Kapitel 3.4. eingehendst besprochen. Die Verschlüsselung des

Datenverkehrs setzt nach erfolgreicher Authentifikation ein. Es können unterschiedliche Verschlüsselungsstärken genutzt werden. Ausgehend von einer 40-Bit Datenverschlüsselung, hin zu einer Schlüssellänge von 128-Bit beim Einsatz von SGC[13] Serverzertifikaten, wenn kein High Encryption Pack vorliegt. Die Übertragung im SSL-Verkehr unterscheidet sich von der gewohnten Kommunikation. Anstatt HTTP auf Port 80 erfolgt die gesicherte Kommunikation über HTTPS auf Port 443. Diesem Umstand sollte beim Web-Design Rechnung getragen werden, wenn auf geschützte Web-Seiten über Links referenziert wird.

VPN Tunnel: Den VPN Tunnelaufbau führen zwei Systeme[14] (VPN-Server) aus. Die jeweiligen Tunnel Endpoints stehen in der Sub-Member Bank bzw. Member Bank. Der gesamte Datenverkehr wird verschlüsselt und authentifiziert (Data Integrity) zwischen diesen beiden Endpoints ausgetauscht. VPN Tunnel unterscheiden sich in der Wahl des Tunneling Protokolls. Windows 2000 unterstützt hierbei PPTP (Point-To-Point-Tunneling Protocol) und L2TP mit IPSec (Layer 2 Tunneling Protocol).
Die Vorteile beim PPTP Tunnel liegen in der einfachen Implementierung und der supporteten Endpoints (W2K, NT, Win9.x). Wohingegen mit L2TP&IPSec noch zusätzlich Integritätsanforderungen des Datenverkehrs abgedeckt werden und CISCO Router Systeme auch LT2P unterstützen. Allerdings ist bei L2TP die Implementierung aufwendiger und beschränkt sich auf Windows 2000 Endpoints.
In **PPTP** erfolgt die Authentifikation über MS-CHAP V2 (Microsoft Challenge Response Authentication Protocol – vergleichbar mit der NTLM Anmeldung). Nach erfolgter Authentifizierung wird ein PPP Tunnel aufgebaut. Der Tunnel bietet neben einer Verschlüsselung von PPP-Paketen auf Basis von MPPE (Microsoft Point to Point Encryption – 40/56-Bit oder 128-Bit – RC4 Algorithmus), auch Encapsulation des intern im LAN eingesetzten Protokolls an. Dabei spielt es keine Rolle ob IP, IPX oder NetBUI für die Kommunikation im LAN verwendet wird.
L2TP ist ausschließlich ein Tunneling Protokoll und verfügt insofern über keinen eigenen Securitymechanismus. Die Security wird durch die Kombination mit IPSec bereitgestellt. Da IPSec auf dem IP Protokoll beruht, muß im internen LAN IP gefahren werden, Es werden somit keine weiteren Netzwerkprotokolle außer IP unterstützt.

[13] Obwohl die Exportrestriktion für starke Verschlüsselung mittlerweile aufgehoben wurde, verfügen eine Vielzahl der heutigen Browsersysteme - außerhalb US und Kanada - nur über eine limitierte Verschlüsselungsstärke von 40-Bit, da die meisten Systeme z. Z. noch nicht mit High Encryption Packs gepatched worden sind. Die Microsoft Export-Browser (ab IE 3.02) und Netscape Export-Browser verfügen jedoch die Möglichkeit auf starke Verschlüsselung (128-Bit) transparent umzuschalten, wenn der Internet Server über ein spezielles Zertifikat verfügt. Voraussetzung ist die Beantragung eines SGC-Zertifikates (Global Server Ids).
[14] Bei hohen Datendurchsätzen ist der Einsatz einer Crypto Karte auf den VPN-Servern zu empfehlen, die den gesamten Verschlüsselungsprozeß ausführt und die CPU bei der Arbeit entlastet. Weiterhin kann die Skalierbarkeit und Ausfallsicherheit mit IP-Load Balancing (Bestandteil von Windows 2000) adressiert werden.

4.3. TS-Client ←→ TS – MSMQ Security

VPN, SSL und IPSec Mechanismen arbeiten Session-orientiert, wohingegen MSMQ-Security Mechanismen, ähnlich wie e-mail Security, Item-orientiert arbeiten. Session- und Item orientierte Security Technologien arbeiten auf unterschiedlichem Wege.

Für **Session-orientierte** Verfahren wird ein virtueller Tunnel zwischen den authentifizierten Entities aufgebaut, worin der Informationsfluß geschützt abläuft. Hinsichtlich der Granularität im Datenverkehr (z. B. eine gesicherte Antwort auf die Fragestellung zu erhalten: Wer hat Was versendet?) ist ein Session-orientiertes Verfahren nur bedingt einsetzbar.

Item-orientierte Verfahren, wie im Falle von MSMQ-Security, wo jede einzelne Message verschlüsselt und authentifiziert (Message Integrity) wird, adressiert die Problemstellung.

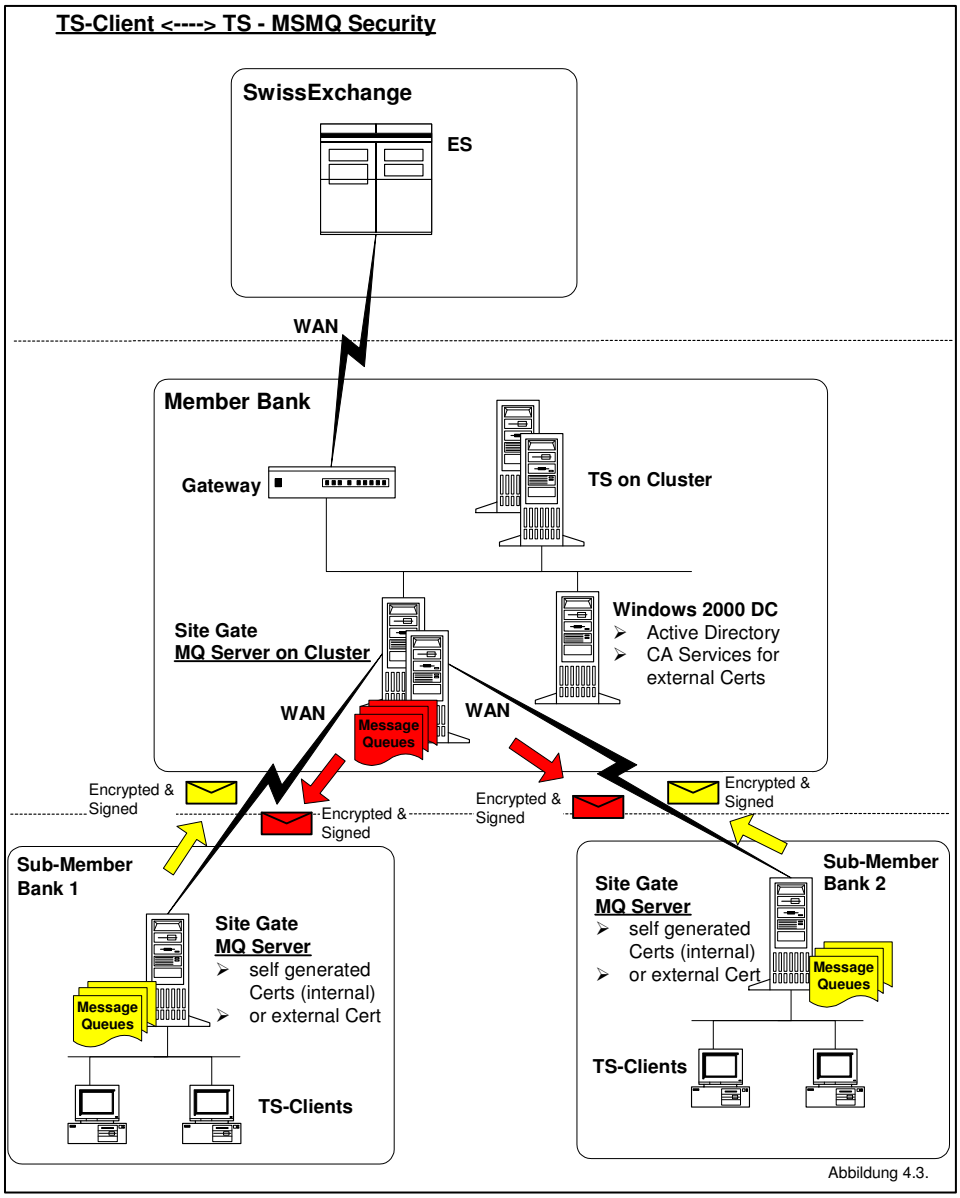


Abbildung 4.3. zeigt einen gesicherten Austausch von MQ-Messages zwischen zwei MQ Server Systemen. Im MMTS WAN

Szenario kommunizieren, TS-Client und TS, die Sub-Member Banken mit der Member Bank, mit Hilfe einer MSMQ fähigen TNA-Lösung, über Messages miteinander. Die Messages laufen lokal auf den definierten Queues auf; beim Transport (z. B. WAN) wird jede einzelne Message verschlüsselt (Message encryption) und digital signiert (Message authentication). Ähnlich wie bei SSL, sind Zertifikate die notwendige Voraussetzung von MSMQ.

Der Aufbau[15] einer MSMQ Security Infrastruktur wird durch automatische Generierung von Zertifikaten beim User Logon erreicht – Interne Zertifikate; im heterogenen Umfeld (z. B. NT & W2K Umgebungen) müssen die Zertifikate über ein CA Service bereitgestellt werden – Externes Zertifikat. Zertifikate intern als auch extern müssen im Directory Service publiziert werden, damit die Zertzifikate und notwendigen Public Keys für jeden verfügbar sind.

Ebenso wie User (Trader) ihre Messages authentifizieren, so müssen sich die MSMQ Server, also Maschinen, sich gegenseitig authentifizieren. Hintergrund ist, daß nur die dafür bestimmten MQ Systeme sich Messages austauschen sollen und nicht ohne Weiteres eine andere Maschine (Fake MQ Server) an der Kommunikation teilhaben soll. Diese Einrichtung bietet noch einen zusätzlichen Level an Security für MSMQ.
Für den Fall, daß die MSMQ Entities auf Windows 2000 basieren, erfolgt eine gegenseitige Authentifikation über Kerberos. In einem mixed Environment (NT 4 & Windows 2000) erfolgt die Authentifikation des MQ Clients mittels NTLM. Wie schon im Kapitel NTLM erwähnt, unterstützt dieses Verfahren keine gegenseitige (mutual) Authentifikation. Abhilfe schafft hierbei ein Server Zerifikat, mit dem sich der MQ Server gegenüber den Client[16] per PKI Mechanismen authentifiziert. Unten aufgeführt unterschiedliche Authentifikationsverfahren für Maschinen in unterschiedlicher Konstellation.

client	server	Server authentication method
Message Queuing client running on Windows 2000	Message Queuing server running on Windows 2000	Kerberos V5
Message Queuing client on Windows 2000	MSMQ 1.0 controller server running on Windows NT 4.0	Windows NTLM (using a server certificate)
MSMQ 1.0 client running on Windows NT 4.0, Windows 95, or Windows 98	Message Queuing server on Windows 2000	Windows NTLM (using a server certificate)

[15] Nach erfolgter Installation von MSMQ Services wird die Konfiguration der Security Infrastruktur mit dem MQ Manager durchgeführt. Das Queue Management und Einstellung von den Betriebsmodi findet man im MMC Tool **Computer Management** unter Services und Apps. Die Zertifikatsverwaltung von User/Client Zertifikaten und Server Zertifikaten, sowie Key & Certificate Renewal werden durch den **MSMQ Wizard** aus dem Control Panel administriert.
[16] Mit Client ist hierbei die aktuelle Rolle gemeint, die der MQ Server einnimmt

Message Encryption:

1. Der MQSender generiert einen Session Key, Mit dem Session Key wird die Message (CipherText Message) verschlüsselt
2. Der MQ Sender sucht im Directory nach dem Zertifikat des MQ Empfänger, holt sich diesen und verschlüsselt mit dem darin enthaltenen Public Key des MQ Empfängers seinen Session Key
3. Der MQ Sender legt die CipherText Message und den als Attachment beigefügten verschlüsselten Session Key in seiner lokalen Message Queue ab

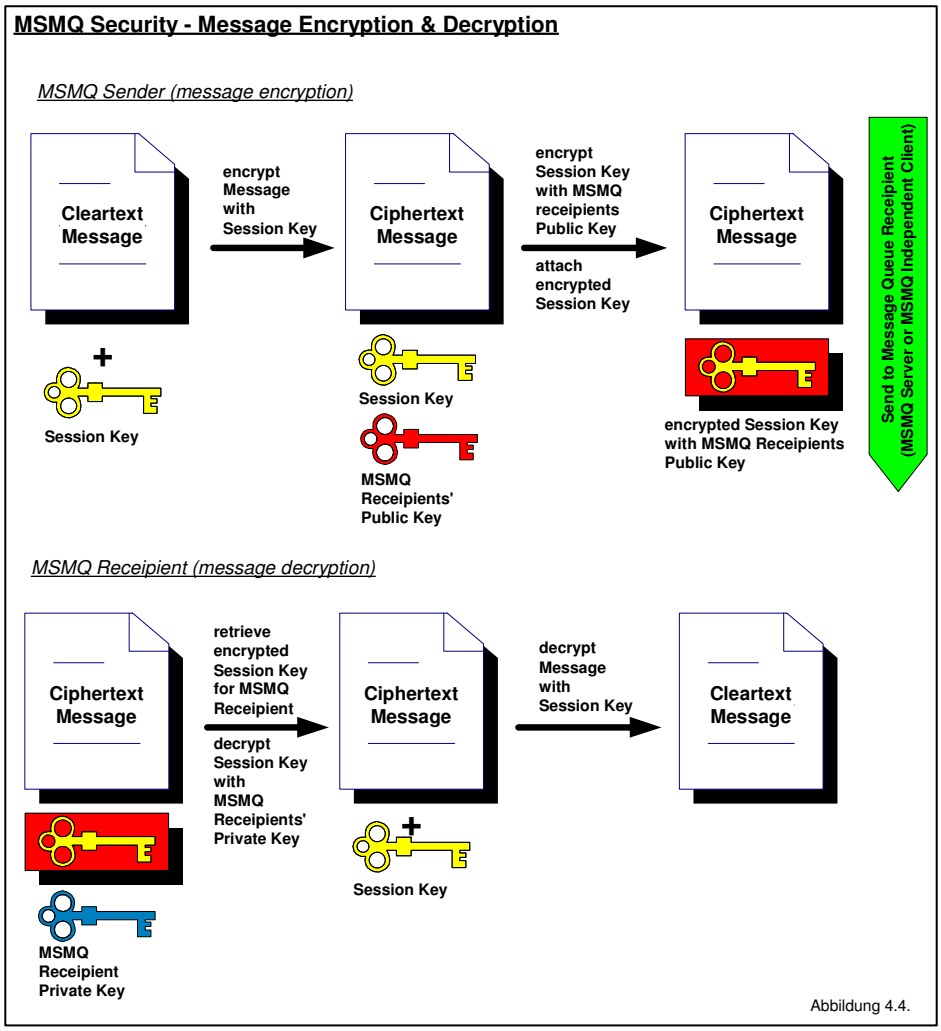


Abbildung 4.4.

4. Die Ciphertext Message wird weitergeleitet und landet in der lokalen Queue des Empfängers. Der MQ Empfänger kann mit seinem korrespondierenden Private Key den verschlüsselten Session Key entschlüsseln
5. Mit den „im Cleartext verfügbaren“ Session Key wird nun die Ciphertext Message entschlüsselt

Message Authentication:

1. Der MQ Sender hashed seine Message mit einem Hashing Algorithmus (MD4). Das Hash-Ergebnis ist der Digest
2. Der Digest wird mit seinem Private Key verschlüsselt. Das Ergebnis aus dieser Operation ist die digitale Signatur
3. Message und Signatur als Attachement werden in der lokalen Message Queue abgelegt

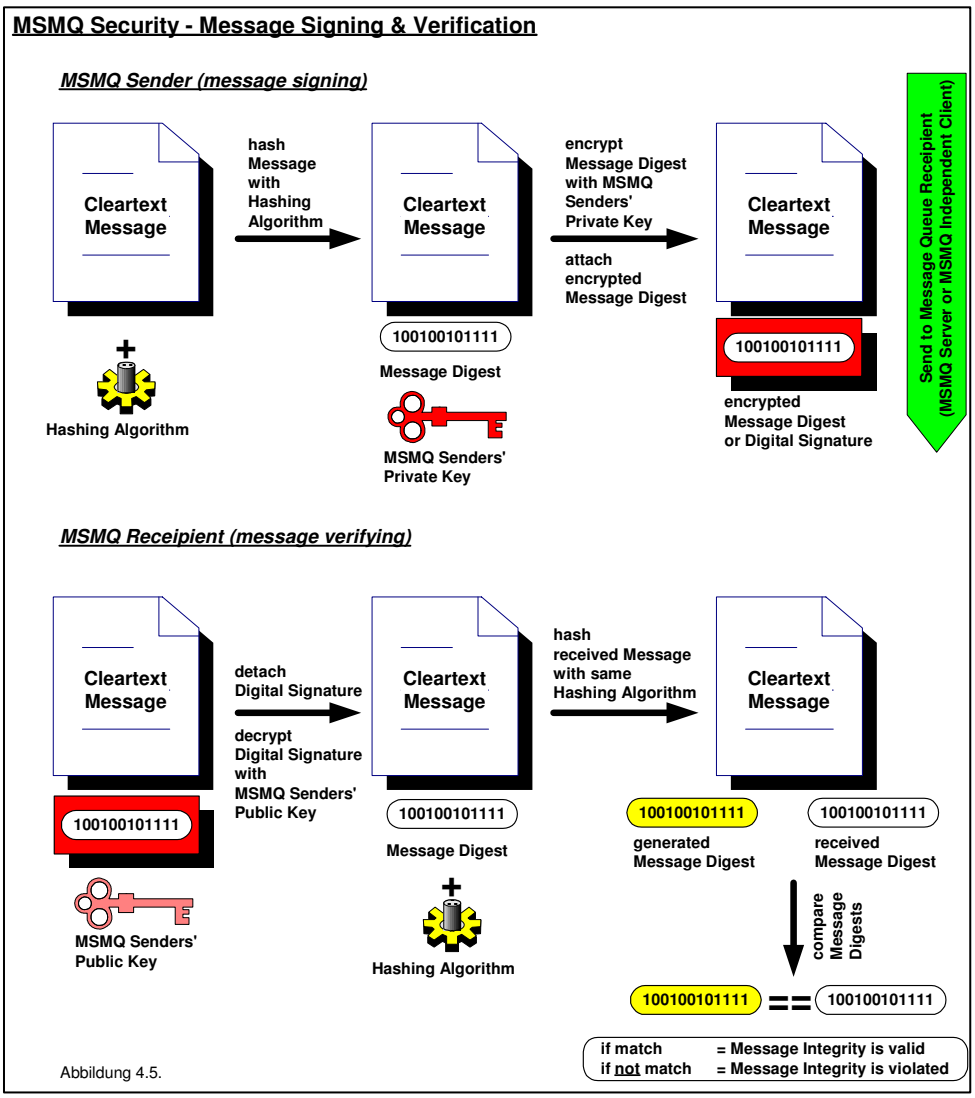


Abbildung 4.5.

4. Message und Signatur laufen in der lokalen Queue des MQ Empfängers auf. Zur Verifikation der digitalen Signatur wird zuerst mit dem Public Key des MQ Senders (aus dem Zertifikat im Directory) die Signatur entschlüsselt.
5. Resultierend aus der Operation ist der Digest der Message. Nun wird lokal der gleiche Hashing Algorithmus aus der empfangenen Message angewandt. Ergebnis ist ein lokal generierter Hashwert oder Digest
6. Lokal generierter Digest und empfangener Digest werden miteinander verglichen. Stimmen diese überein, so ist die *Message authentisch und wurde auf dem Transport nicht verändert*.

TNA-GUI ←→ TS TNA AUTHORISIERUNG

5. TS-CLIENT ←→ TS TNA - AUTHORISIERUNG

Die spezifischen Anforderungen, die ib der SwissExchange TNA-Lösung existieren, können allein durch die infrastrukturelle Security nicht abgedeckt werden. Eine typische Anforderung stellt der Trader dar, der eine Liste von bestimmten Securities handeln kann, aber bestehende Trades aus dem Order Book nicht löschen darf, etc.

Mit MTS Role-Based Security kann die oben genannte Problematik auf unterschiedlichen Wege adressiert werden. Zu einem kann deklarativ einem MTS-Package eine Rolle (Trader, Main Trader, etc.) zugewiesen werden, diesen Rollen stehen wiederum unterschiedliche Ausführungrechte zur Verfügung. Z. B. „Order erlaubt“ oder „Order verweigert“. Diese Rollen werden dann existierenden Windows 2000 Benutzerkonten zugewiesen. Die Einstellung der deklarativen MTS Security wird im MTS-Explorer vorgenommen. Doch reicht diese Granularität hinsichtlich von unterschiedlich handelbaren Securities in den meisten Fällen nicht aus. Eine höher Granularität erreicht man durch den programmatischen Ansatz. Hierbei findet die Überprüfung im Code selbst statt. (IsecurityCallContext → IsCallerInRole → IsSecurityEnabled). D. h. die Rechtevergabe findet in der eigentlichen Businesslogik der Anwendung statt. Näheres dazu siehe Anhang – *7.4. MTS Security*.

Ein weiterer Problempunkt besteht für DCOM Anwendungen, wenn diese über FireWalls miteinander kommunizieren. DCOM allokiert dynamisch TCP Ports für den Kommunikationsaufbau, dies wiederum steht im Gegensatz zur Konfiguration bestehender Firewall Umgebungen. Firewalls legen in der Regel nur spezifische und wohlbekannte Ports (z. B. http port 80 oder ftp port 21) frei auf denen die Kommunikation stattfindet. Ein Lösungsansatz ist im *Anhang 7.2. Using Distributed COM with Firewalls und 7.3. COM Internet Services* dokumentiert.

6. PUBLIC KEY INFRASTRUCTURE

Die Durchgängigkeit einer zentralen Authentifikation (Single Sign-On) und Authorisierung kann zukünftig mit Einsatz einer Public Key Infrastruktur[17] erfüllt werden, wie man aus der Darstellung in Abbildung 6.1. entnehmen kann.

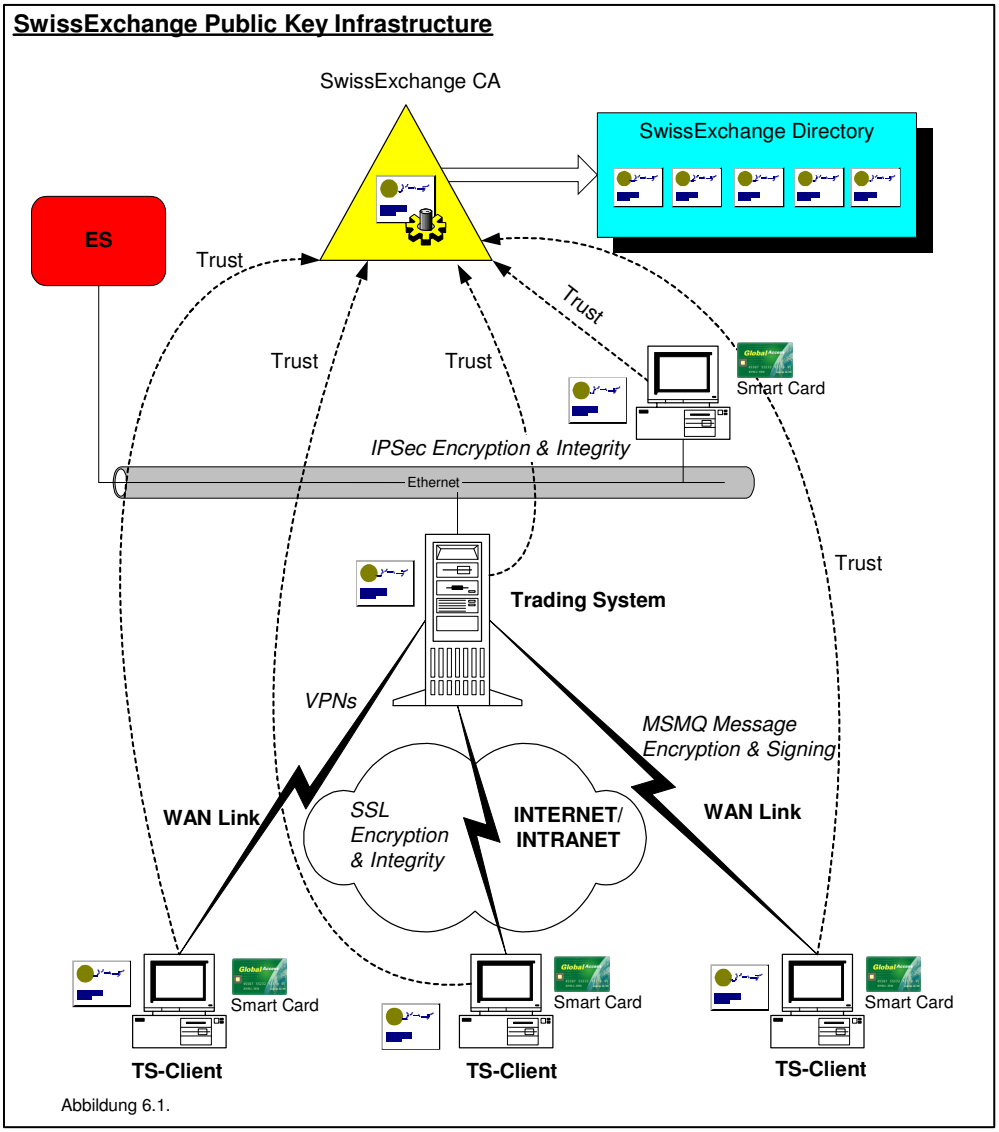


Abbildung 6.1.

Das Windows 2000 Trading System ist vorbereitet für den Einsatz von Zertifikaten und ebenso auch Windows 2000 TS-Clients. Anstatt der Anmeldung durch UserIDs und PWs kann eine gesicherte Authentifikation mittels Zertifikaten erfolgen. In sicherheitskritischen Umgebungen ist der Einsatz von Smart Cards zu empfehlen. Eine Vielzahl, der in den Kapiteln erwähnten MS-Technologien (IPSec, SSL, MSMQ), unterstützen den Einsatz einer PKI Umgebung.

[17] Windows 2000 beinhaltet alle notwendigen Komponenten für den Aufbau einer Public Key Infrastruktur.

Der Einsatzbereich einer PKI-Lösung beschränkt sich nicht nur auf die TNA Umgebung, sondern stellen nur einen kleinen Ausschnitt der denkbaren Szenarien dar. Szenarien wie den Einsatz von Secured Mail Systemen (S/MIME), Code Sigining Mechanismen (Authenticode) basieren ebenso auf der Grundlage einer PKI Umgebung.

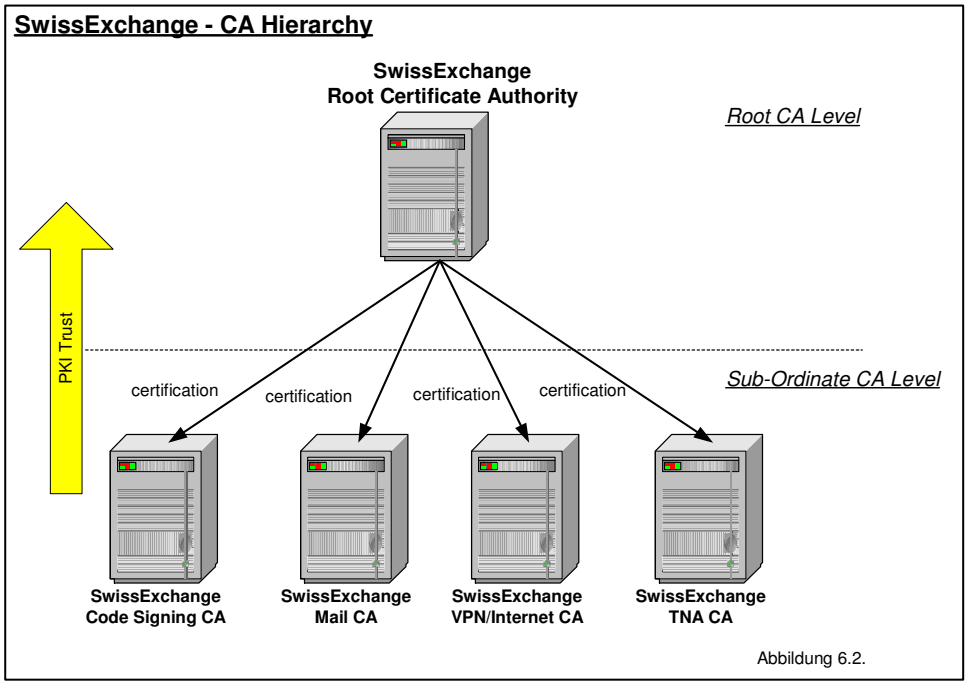


Abbildung 6.2.

Grundlage einer PKI ist die CA Infrastruktur. Aus organisatorischen und technischen Erwägungen als auch Gründen der Ausfallsicherheit werden CAs in einer Hierarchy innerhalb einer Organisation aufgebaut. *Abbildung 6.2.* verdeutlicht den Zusammenhang.

Eine zentrale SwissExchange Root CA zertifiziert die darunterliegenden Issuing CAs. Technisch gesehen werden den Issuing CAs das Root Zertifikat der Root CA auf den Issuing CAs registriert. Damit wird eine Hierarchy über einen PKI Trust zwischen Root CA und Issuing CA etabliert. Die eigentliche Ausstellung der Zertifikate erfolgt dann in aller Regel über die ISSUING CAs. Im Diagramm sind die ISSUING CAs je nach Art der Anwendung aufgeteilt, die für die unterschiedlichsten Anwendungen die entsprechenden Zertifkate an den Benutzer oder auch an Maschinen ausstellen.

7. ANHANG

7.1. DCOM Security

Designing a distributed application poses several challenges to the developer. One of the most difficult design issues is that of security: Who can access which objects? Which operations is an object allowed to perform? How can administrators manage secure access to objects? How secure does the content of a message need to be as it travels over the network?

Mechanisms to deal with security-related design issues have been built into DCOM from the ground up. DCOM provides an extensible and customizable security framework upon which developers can build when designing applications.

Different platforms use different security providers, and many platforms even support multiple security providers for different usage scenarios or for interoperability with other platforms. DCOM and RPC are designed in such a way that they can simultaneously accommodate multiple security providers.

All these security providers provide a means of identifying a security principal (typically a user account), a means of authenticating a security principal (typically through a password or private key), and a central authority that manages security principals and their keys. If a client wants to access a secured resource, it passes its security identity and some form of authenticating data to the resource and the resource asks the security provider to authenticate the client. Security providers typically use low-level custom protocols to interact with clients and protected resources.

Security Policies

DCOM distinguishes between four fundamental aspects of security:

Access security. Which security principals are allowed to call an object?

Launch security. Which security principals are allowed to create a new object in a new process?

Identity. What is the security principal of the object itself?

Connection policy. Integrity—can messages be altered? Privacy—can messages be intercepted by others? Authentication—can the object find out or even assume the identity of the caller?

Protecting the object: access security

The most obvious security requirement on distributed applications is the need to protect objects against unauthorized access. Sometimes only authorized users are supposed to be able to connect to an object. In other cases, non-authenticated or unauthorized users might be allowed to connect to an object, but must be limited to certain areas of functionality. Current implementations of DCOM provide declarative access control on a per-process level. Existing components can be securely integrated into a distributed application by simply configuring their security policy as appropriate. New components can be developed without explicit security awareness, yet still run as part of a completely secure distributed application.

If an application requires more flexibility, objects can programmatically perform arbitrary validations, be it on a per-object basis, per-method basis, or even per-method parameter basis. Objects might also want to perform different actions depending on who the caller is, what specific access rights the caller has, or to which user group the caller belongs.

Protecting the server machine: launch security

Another related requirement on a distributed infrastructure is to maintain control over who can create objects. Since all COM objects on a machine are potentially accessible via DCOM, it is critical to prevent unauthorized users from creating instances of these objects. This protection has to be performed without any programmatic involvement of the object itself, since the mere act of launching the server process could be considered a security breach and would open the server to denial-of-service attacks.

The COM libraries thus perform special security validations on object activation. If a new instance of an object is to be created, COM validates that the caller has sufficient privileges to perform this operation. The privilege information is configured in the registry, external to the object.

Controlling the object: security identity

Another aspect of distributed security is that of controlling the objects themselves. Since an object performs operations on behalf of arbitrary callers, it is often necessary to limit the capabilities of the object itself. One obvious approach is that of making the object assume the identity of the caller. Whatever action the object performs—a file access, network access, registry access, and so on—is limited by the caller's privileges. This approach works well for objects that are used exclusively by one caller since the security identity is established once at object creation time. (For more information, see explanation of "Run as Activator" in "Fundamentals: Windows NT security infrastructure" later in this section.) The approach can also be used for shared objects if the object performs an explicit action on each method call. (For more information, see explanation of "Impersonation" in "Programmatic Security" later in this section.)

However, for applications with large number of users, the approach of making the object assume the identity of the caller can impose problems. All resources that are potentially used by an object need to be configured to have exactly the right set of privileges. If the privileges are too restrictive, some operations on the object will fail. If the privileges are too generous (i.e., there is write access to some files where only read access is required), security violations might be possible if the object is not well behaved. Although managing access can be simplified by using user groups, it is often simpler to have the object itself run under a dedicated security identity, independent of the security identity of the current caller.

Other applications may not even be able to determine the security identity of the caller. Many Internet applications, for example, do not assign a dedicated user account for every user. Any user can use the application and yet the objects still need to be secure when accessing

resources. Again, assigning objects a security identity of their own makes this kind of application manageable in terms of security.

Protecting the data: connection policy

As the “wire” between callers and objects becomes longer, the possibility of data that is being transported as part of method invocations being altered or intercepted by third parties increases. DCOM gives both callers and objects a range of choices to determine how the data on the connection is to be secured. The overhead in terms of machine and network resources tends to grow with the level of security. DCOM, therefore, lets applications dynamically choose the level of security they require.

Physical data integrity is usually guaranteed by the low-level network transport. If a network error alters the data, the transport automatically detects this and retransmits the data. However, for secure distributed applications, data integrity really means the ability to determine whether the data actually originated from a legitimate caller and whether it has been logically altered by anyone. The process of authenticating the caller can be relatively expensive, depending on the security provider and the security protocol it implements. DCOM lets applications choose whether and how often this authentication occurs (see the next section for details.)

DCOM currently offers two fundamental choices with regard to data protection: integrity and privacy. Clients or objects can request that data be transferred with additional information that ensures data integrity. If any portion of the data is altered on its way between the client and the object, DCOM will detect this and automatically reject the call. Data integrity implies that each and every data packet contains authentication information.

However, data integrity does not mean no one can intercept and read the data being transferred. Clients or objects can request that all data be encrypted (See the explanation of packet privacy in the next section). Encryption implies an integrity check as well as per-packet authentication information. Since both privacy and integrity require authentication, the mechanisms for specifying privacy and authentication are unified into a single enumeration of authentication levels, which are described in the next section.

Identifying the caller: authentication

The above mechanisms for access check, launch permission check, and data protection require some mechanism for determining the security identity of the client. This client authentication is performed by one of the security providers, which returns unique session tokens that are used for ongoing authentication once the initial connection has been established. The initial authentication often requires multiple roundtrips between caller and object. The NTLM security provider, for example, authenticates by challenging the caller: the security provider knows the password of the user (more precisely an MD4 hash of the password). It encrypts a randomly generated block of data using the MD4 hash of the password and sends it back to the client (the challenge). The client then decrypts the data block and returns it to the server. If the client also

knows the correct password, the decryption is successful and the server knows that the client is “authentic.” The NTLM security provider then generates a unique access token, which it returns to the client for future use. For future authentication, the client can simply pass in the token, and the NTLM security provider does not perform the extra roundtrips for the “challenge/response” protocol.

DCOM uses the access token to speed up security checks on calls. However, to avoid the additional overhead of passing the access token on each and every call, DCOM by default only requires authentication when the initial connection between two machines is established (RPC_C_AUTHN_LEVEL_CONNECT). It then caches the access token on the server side and uses it automatically whenever it detects a call from the same client.

For many applications this level of authentication is a good compromise between performance and security. However, some applications may require additional authentication on each and every call. Often, certain methods in an object are more sensitive than others. An online shopping mall might only require authentication on connection establishment as long as the client is only calling methods for “browsing” the shopping mall. But when the client actually orders merchandise and passes in credit card information or other sensitive information, the object might require calls to be individually authenticated.

Depending on the transport used and the size of the method data to be transmitted, a method invocation can actually require multiple data packets on the network. DCOM lets applications choose whether only the first packet of a method invocation contains authentication information (RPC_C_AUTHN_LEVEL_CALL) or whether each packet should be individually authenticated (RPC_C_AUTHN_LEVEL_PKT).

As discussed in the previous section, authentication, integrity, and privacy are tightly related. For this reason, DCOM defines a single set of constants that conveys the level of authentication and privacy.

7.2. Using Distributed COM with Firewalls

Introduction

This article is intended to provide the reader with information on how to configure the Distributed Component Object Model (DCOM) to work through a firewall. It assumes that the reader is familiar with the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP), DCOM, and understands the basics of firewall technology. Please note that the port restriction procedures outlined below only work on Microsoft® Windows NT®; Microsoft Windows® 95 does not include this functionality at this time (don't worry, it isn't necessary in most cases). Also note that much of the discussion in this article is also relevant for remote procedure call (RPC) applications which use dynamic ports. This makes sense considering that the features mentioned below are really implemented in the RPC subsystem that DCOM leverages to communicate across the network.

Unlike most Internet applications which have fixed TCP and/or UDP ports, DCOM dynamically assigns—at run time—one TCP port and one UDP port to each executable process serving DCOM objects on a computer. That is, even if a process is hosting 2,000 clients and 50,000 objects, any client wishing to communicate with objects owned by it will always connect to the same TCP or UDP port. Clients discover the port associated with a particular object by connecting to and using the services provided by DCOM's Service Control Manager (SCM). The SCM always operates at a fixed network port on every computer; in the Internet case this is always port 135 for both TCP and UDP. The SCM offers several RPC-based (not DCOM/ORPC-based) services which handle operations like: "create a new COM class object for me and tell me what TCP and UDP port it is on" or "I have an interface pointer, tell me where I need to go to actually use it", and so forth. A much more technical explanation of this process and of the DCOM wire protocol in general is documented in the Distributed Component Object Model Protocol—DCOM/1.0.

DCOM's dynamic port allocation feature offers great flexibility in that programmers and administrators alike are free from the burden of having to configure (or hard code) applications to specific ports, free from resolving conflicts between multiple applications attempting to use the same port(s), and so on. Unfortunately, because DCOM (by default) is free to use any port between 1024 and 65535 when it dynamically selects a port for an application, it is rather "firewall unfriendly" out of the box. Configuring your firewall to leave such a wide range of ports open would present a serious security hole. Microsoft's developers realized this and have implemented a feature that allows you to restrict the range of ports that DCOM will use to assign to applications.

You should be aware that callbacks in DCOM are *not* handled on the same connection that is used for client/server method calls. When a server makes a callback to a client, it creates a new connection to the client and sends method calls over that separate channel. In other words, DCOM treats callbacks just like any other client/server method call, except that your "client" is really a server and the "server" is really a

client. In some circumstances (this is *very* rare), you may need to configure port restrictions on your clients if your firewall restricts what ports machines on the inside can connect to on the outside.
Also note that if you want to use callbacks through a firewall, you must use TCP. The reason for this is that when the server makes a call to the client, the source port will not be within the range below and thus when the client sends a reply to the server's source port, it will not be able to penetrate the firewall. This is not a problem with TCP because most firewalls keep track of TCP connections and permit bidirectional traffic on connections, regardless of the source port, as long as they are opened from a machine on the inside.
One last thing before I continue, the client *must* be able to reach the server by its actual IP address. You cannot use DCOM through firewalls that do address translation (i.e. where a client connects to virtual address 198.252.145.1 that the firewall maps transparently to the server's actual address of, say, 192.100.81.101). This is because DCOM stores raw IP addresses in the interface marshaling packets and if the client cannot connect to the address specified in the packet, it won't work.

Configuring DCOM to Use TCP Only
Here is a rundown of the transport protocols that DCOM will use depending on the client/server platform:

Platform	**Protocol**
Microsoft Windows NT 4.0 <-> Windows NT 4.0	UDP
Microsoft Windows 2000 <-> Any	TCP (*)
Microsoft Windows 95 or Windows 98 <-> Any	TCP
DCOM for UNIX <-> Any	TCP

***** When a Windows NT 4.0 client attempts to connect to a DCOM server running anything but Windows NT 4.0, there will be a 30-45 second delay while it attempts to connect via the UDP protocol. Windows 95 and Windows 98, Windows 2000, and DCOM for UNIX (including Microsoft's DCOM for UNIX product and Software AG's EntireX product) always use the TCP protocol.
The Windows NT 4.0 implementation of DCOM uses UDP wherever it can because benchmarks show that it can outperform TCP in certain scenarios. Unfortunately, it is very difficult to make UDP applications work through firewalls without exposing your network to unnecessary risks. Before continuing, be sure you make TCP the default protocol on all Windows NT servers by moving the "NCACN_IP_TCP" value to the top of the list in the *DCOM Protocols* named value of the **HKEY_LOCAL_MACHINE\Software\Microsoft\Rpc** registry key using regedt32.exe. To eliminate a 30-45 second delay when connecting to TCP only servers, the same change should also be made on Windows NT 4.0 clients. Note that this delay only occurs if you haven't connected

to the target server in a while (RPC's connection caching features retain knowledge about the server so that multiple connections in a short period of time are resolved instantaneously).
The Windows 2000 implementation of DCOM will likely default to TCP because it is not possible to implement the SSL and SNEGO security protocols over UDP.

Restricting the Range of TCP Ports
There are several registry settings that control the DCOM port restriction functionality. All of the named values listed below are located under the **HKEY_LOCAL_MACHINE\Software\Microsoft\Rpc\Internet** registry key (which you must create). Remember that you only need to do this on the server machine. Clients will automatically pick up the right port numbers when they connect to the SCM on the server machine.
Note You must use regedt32.exe to configure these settings, regedit.exe does not currently support the REG_MULTI_SZ type required by the *Ports* named value entry. Also, you must reboot your machine any time you make changes to any of the following registry settings in order for them to take effect.

Name	Type	Value	Description
Ports	REG_MULTI_SZ	Specify one port range per line. Example: 3000-4000 5141	One or more port ranges. The options below determine the meaning of this named value.
PortsInternetAvailable	REG_SZ	"Y" (don't include quotes)	Always set this to "Y".
UseInternetPorts	REG_SZ	"Y" or "N" (don't include quotes)	If this value is set to "Y", then the *Ports* named value indicates which ports should be used for DCOM applications. If this value is set to "N", then the *Ports* named value indicates which ports should NOT be used for DCOM applications.

Restricting the Internet Accessibility to Specific Applications
An even greater level of security is afforded if you configure your system so that DCOM applications have to explicitly ask to be accessible through the Internet-accessible port range. To set this up:
Change the *UseInternetPorts* named value above from "Y" to "N" so that DCOM object servers (and RPC servers) aren't reachable via the Internet-accessible ports automatically. Reboot your machine.
Insert the following piece of code before **CoInitializeEx()** into every DCOM object server application that should be accessible through the Internet-accessible port range:

```
RPC_POLICY rp;
```

```
rp.Length = sizeof (RPC_POLICY);
rp.EndpointFlags = RPC_C_USE_INTERNET_PORT;
rp.NICFlags = RPC_C_BIND_TO_ALL_NICS;
hr = RpcServerUseAllProtseqsEx
(RPC_C_PROTSEQ_MAX_REQS_DEFAULT, NULL, &rp);
```

Add "**rpcrt4.lib**" to the link command line in your makefile.

Configuring Your Firewall

The firewall between your server and the Internet should be configured as follows:

Deny all incoming traffic from the Internet to your server.

Permit incoming traffic from all clients to TCP port 135 (and UDP port 135, if necessary) on your server.

Permit incoming traffic from all clients to the TCP ports (and UDP ports, if necessary) on your server in the *Ports* range(s) specified above.

If you are using callbacks, permit incoming traffic on all ports where the TCP connection was initiated by your server.

7.3. COM Internet Services

Introduction

COM Internet Services (CIS) introduces support for a new DCOM transport protocol known as *Tunneling TCP* that allows DCOM to operate over TCP port 80. This allows a client and a server to communicate in the presence of most proxy servers and firewalls, thereby enabling a new class of COM-based Internet scenarios.

In addition to the new DCOM protocol, CIS also provides a new type of simple moniker—the OBJREF moniker—that facilitates the use of COM in Internet scenarios. The OBJREF moniker represents a reference to a running object and has a display name that can, for example, be embedded in an HTML page and bound by an ActiveX® control or client applet.

This article explains what the COM Internet Services are, how they work, and how to configure computers running Microsoft® Windows® to use the services. The article assumes the reader has a basic understanding of distributed COM and the DCOM configuration tool.

A New DCOM Transport Protocol

In many Internet situations, the network connectivity between a client and a server is subject to a number of restrictions. For example:

A proxy server that filters outbound network traffic may gate the client connection to the Internet. This is often the case for applications running in a corporate environment, but it can also apply to applications run by a user connecting to the Internet through an ISP.

A firewall often controls incoming Internet traffic, defining what combinations of network ports, packets and protocols are accepted to protect the server (or client) network environment.

In practice, the net effect of such restrictions is that a client and a server will probably have a very narrow set of protocol and port combinations available to carry out a conversation. Since DCOM dynamically selects network ports in a range (1024 – 65535) on which Internet-to-intranet network traffic is typically not allowed, it is not possible to reliably use the existing DCOM transport protocols over the Internet (although they are perfectly suitable for intranets). Moreover, firewalls are often setup to restrict access to port 135, which DCOM depends on for a variety of services[18].

The Tunneling TCP protocol introduces a special handshake at the beginning of each DCOM connection that allows it to pass through most firewalls and proxies. After this handshake, the wire protocol is simply DCOM over TCP. Aside from the caveats listed later in this article, this means:

The protocol is transparent to both client and server. Neither the client code nor the server code need to be modified to use CIS.

All of the DCOM over TCP protocol services are available—including DCOM security and lifetime management (that is, “pinging”) services.

[18] For a discussion of firewall issues with DCOM over TCP applicable to some firewall scenarios the reader is referred to Michael Nelson’s article “Using Distributed COM with Firewalls” (http://www.microsoft.com/com/wpaper/dcomfw.asp).

Limitations of the Tunneling TCP Protocol

The Tunneling TCP protocol is subject to the following limitations:

It requires that Internet Information Server version 4.0 or higher be installed on the server-side machine hosting CIS-accessible COM objects, since part of CIS functionality is implemented using an ISAPI filter.

Since Tunneling TCP consists of non-HTTP traffic after the initial handshake, CIS requires that proxy servers and firewalls permit such traffic over a port opened to HTTP.

Note that because of these limitations, in practice CIS does not support callbacks. This means, for example, that your applications cannot perform notifications using the connection point or advise sink mechanisms. However, if the CIS client can function as a CIS server and is configured as discussed later in this article, nothing prevents the client from receiving DCOM calls—including callbacks.

Protocol Overview

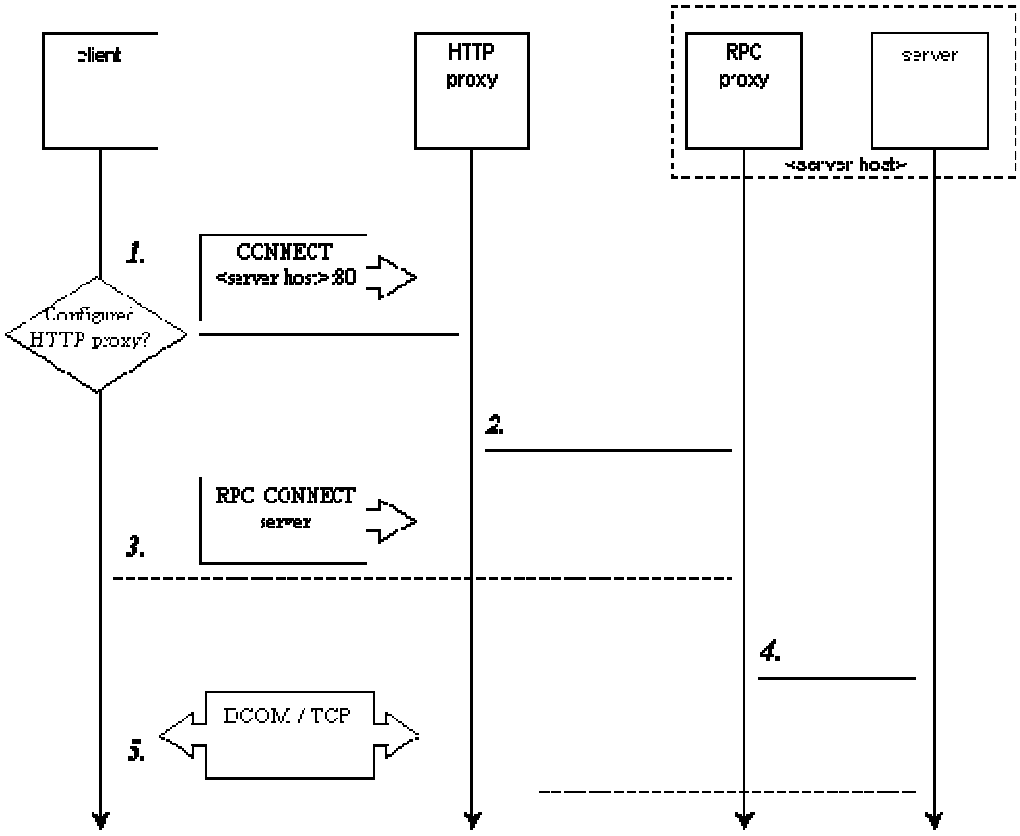


Figure 1: The Tunneling TCP Protocol

If the client configuration indicates that HTTP traffic to the server should be routed through a proxy, the client DCOM run-time environment establishes a TCP/IP connection to that proxy. It then sends the HTTP CONNECT method to the proxy requesting connection to port 80 on <server host>.

The proxy establishes a TCP/IP connection with <server host>. Note that this assumes that the proxy is configured to enable the HTTP CONNECT method on the port connected by the client. This port configuration on the proxy is sometimes referred to as “enabling SSL tunneling”.

If the client configuration does not use a proxy, the DCOM run-time environment establishes a TCP/IP connection to port 80 on <server host>. After this step, whether or not the client uses a proxy, the client has a connection to port 80 on <server host> (perhaps mediated by a proxy). The client now sends the RPC_CONNECT command to the server requesting connection to the DCOM server on <server host>[19].
In response to the RPC_CONNECT, the server RPC run-time environment (implemented in part by an ISAPI filter/extension pair) establishes a local connection to the DCOM server[19].
Client and server have now established a mediated TCP/IP connection and engage in a DCOM over TCP conversation.

7.4. MTS Security Roles

MTS provides a distributed security implementation for component-based applications. MTS uses Windows NT security to authenticate users, but provides its own options for authorization.Transaction Server provides two complementary models called declarative and programmatic security. Declarative security is automatic and is specified when components are added into a package and doesn't require developers to do any programming. Administrators declare which users and groups of users have access to the package using the MTS Explorer. This enables deployment-specific security, even for when used with prebuilt components purchased from third parties.
In contrast, programmatic security lets developers build custom access controls directly into their component by explicitly using roles. For example, roles in a banking application could be called teller and manager, and components could check to make sure that the current user has been given manager privileges before performing certain operations. Administrators associate users with roles using the MTS Explorer. Most important, components themselves have no embedded knowledge of specific users and don't have to make explicit calls to the Windows NT Security environment. This dramatically improves the ability of a component to be both secure and reusable in a broad sense.

[19] This may include the initial additional step of contacting the DCOM Service Control Manager on <server host> to resolve the DCOM server endpoint.

7.5. Security Support Provider Interface

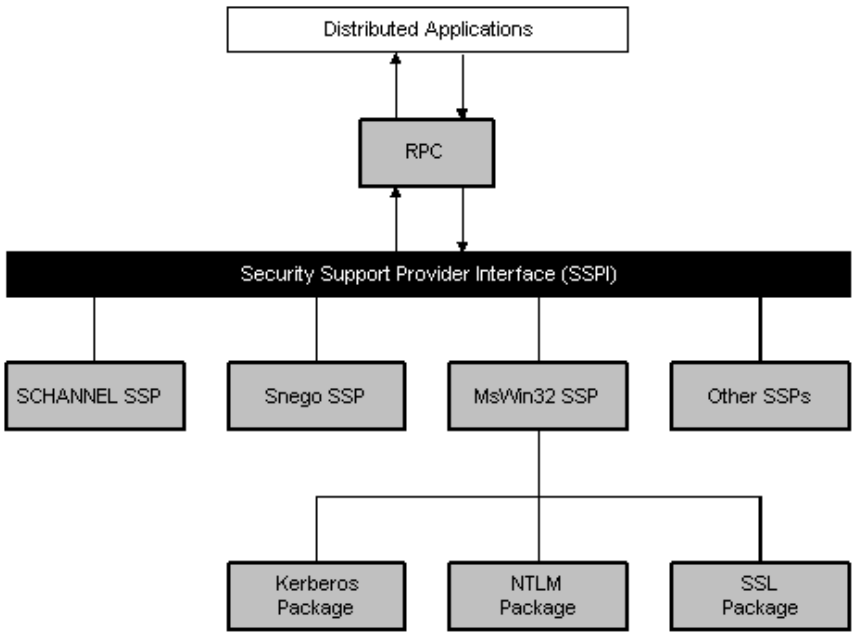
SSPI Architectural Overview

SSPI is a software interface. Distributed programming libraries such as RPC can use it for authenticated communications. One or more software modules provide the actual authentication capabilities. Each module, called a security support provider (SSP), is implemented as a dynamic link library (DLL). An SSP provides one or more security packages.

A variety of SSPs and packages are available. For instance, Windows NT and Windows 2000 ship with the NTLM security package. Beginning with Windows 2000, Microsoft also provides the Microsoft Kerberos protocol security package. In addition, you may choose to install the Secure Socket Layer (SSL) security package. These security packages are implemented by the Microsoft® Win32® SSP, which is implemented in Secur32.dll. You may also choose to install any other SSPI-compatible SSP.

Using SSPI ensures that no matter which SSP you select, your application accesses the authentication features in a uniform manner. This capability provides your application greater independence from the implementation of the network than was available in the past.

The architecture of SSPI is illustrated in the following diagram.



The preceding illustration shows that distributed applications communicate through the RPC interface. The RPC software in turn, accesses the authentication features of an SSP through the SSPI.

SSPI/Kerberos Interoperability with GSSAPI

Care must be taken when using the Kerberos SSP if interoperability with GSSAPI is a requirement. The following coding conventions allow interoperability with GSSAPI-based applications:

- Windows 2000 compatible names
- Authentication
- Message Integrity and Privacy

Windows 2000 Compatible Names

GSSAPI functions use a name format known as *gss_nt_service_name* as specified in the RFC. Note in the GSSAPI name, 'nt' does not stand for 'NT.' *sample@host.dom.com* is an example of a name that can be used in a GSSAPI based application. Windows 2000 does not recognize the *gss_nt_service_name* format, and the full service principal name, for example *sample/host.dom.com@REALM*, must be used.

Authentication

Authentication is usually handled when a connection is first set up between a client and server. In this example, the client is using SSPI and the server is using GSSAPI.

In the SSPI client:

1. Get outbound credentials using **AcquireCredentialsHandle**.
2. Create service name with **gss_import_name()** and get inbound credentials using **gss_aquire_cred**.
3. Get authentication token to send to server using **InitializeSecurityContext**.
4. Send token to server.

In the GSSAPI server:

1. Parse message from client to extract security token. Use **gss_accept_sec_context** passing token as argument.
2. Parse message from server to extract security token. Pass to **InitializeSecurityContext**.
3. Send response token to client. **gss_accept_sec_context** can return a token to send back to the client.
4. If continue needed, send token to server; otherwise connection is completed
5. If continue needed, wait for next token from client; otherwise connection is completed

Message Integrity and Privacy

Most GSSAPI-based applications use **GSS_Wrap** to sign a message before sending it. Conversely, **GSS_Unwrap** verifies the signature. **GSS_Wrap** is a new part of the API (v2) and is now widely used and specified in Internet standards that describe using the GSSAPI for adding security to protocols. Earlier, the GSS **SignMessage** and **SealMessage** functions were used for message integrity and privacy. **GSS_Wrap** and

GSS_Unwrap are used for both integrity and privacy with the use of privacy controlled by the value of the "conf_flag" argument.

If a GSSAPI-based protocol is specified to use **gss_get_mic** and **gss_verify_mic**, the correct SSPI functions would be **MakeSignature**[20] and **VerifySignature**. Be aware that **MakeSignature** and **VerifySignature** will not interoperate with **gss_wrap** with conf_flag set to zero or with **gss_unwrap**. The same is true for mixing **EncryptMessage** set for signature only and **gss_verify_mic**.

In the SSPI client:

```
// Need 3 descriptors - 2 for the SSP and one to hold the
application data
in_buf_desc.cBuffers = 3;
in_buf_desc.pBuffers = wrap_bufs;
in_buf_desc.ulVersion = SECBUFFER_VERSION;
wrap_bufs[0].cbBuffer = sizes.cbSecurityTrailer;
wrap_bufs[0].BufferType = SECBUFFER_TOKEN;
wrap_bufs[0].pvBuffer = malloc(sizes.cbSecurityTrailer);
// This buffer holds the application data
wrap_bufs[1].BufferType = SECBUFFER_DATA;
wrap_bufs[1].cbBuffer = in_buf.cbBuffer;
wrap_bufs[1].pvBuffer = malloc(wrap_bufs[1].cbBuffer);
memcpy(wrap_bufs[1].pvBuffer, in_buf.pvBuffer, in_buf.cbBuffer);
wrap_bufs[2].BufferType = SECBUFFER_PADDING;
wrap_bufs[2].cbBuffer = sizes.cbBlockSize;
wrap_bufs[2].pvBuffer = malloc(wrap_bufs[2].cbBuffer);
maj_stat = EncryptMessage(&context,
SignOnly ? KERB_WRAP_NO_ENCRYPT : 0,
&in_buf_desc, 0);
// Send message to server
```

In the GSSAPI server:

```
// Received message is in recv_buf
maj_stat = gss_unwrap(&min_stat, context, &recv_buf, &msg_buf,
&conf_state, (gss_qop_t *) NULL);
(void) gss_release_buffer(&min_stat, &recv_buf);
// Original message is in msg_buf
```

The SSPI equivalent to **GSS_Unwrap** is **DecryptMessage**. Here is a code sample that shows how to use **DecryptMessage** to decrypt data that was encrypted by **GSS_Wrap.**

In the GSSAPI server:

```
// Seal the message
send_buf.value = msg;
send_buf.length = msglen;
```

[20] **Note** Do not use the **MakeSignature** or **VerifySignature** functions when **GSS_Wrap** and **GSS_Unwrap** are called for.
The SSPI equivalent to **GSS_Wrap** is **EncryptMessage** for both integrity and privacy.
The following code sample shows using **EncryptMessage** to sign data that will be verified by **GSS_Unwrap**.

```
// If encrypt_flag = 1, privacy; encrypt_flag = 0, integrity
maj_stat = gss_wrap(&min_stat, context, encrypt_flag,
GSS_C_QOP_DEFAULT,
&send_buf, &state, &msg_buf);
// The message to send is in msg_buf
```