GTE **Laboratories**

Security Functions in the GTE Management Information Protocol

Shai Halevi
(summer MTS)

Project Team

Lazar Koyfman
Jeff Getchius
Bhaskar Purimetla

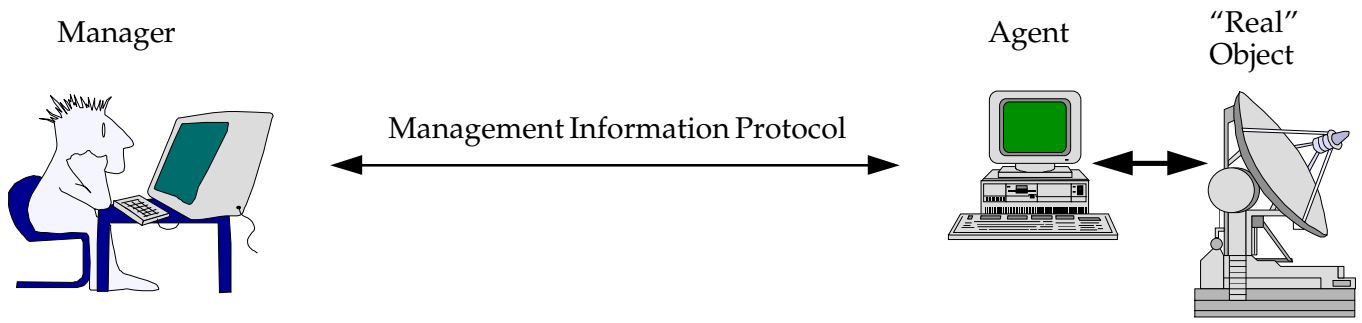
GTE **Laboratories**

Outline

- Background
  - Service and Network Management
  - GMIP (GTE Management Information Protocol)
  - Security Issues, cryptography
- Design
  - Scope of this work
  - Design Choices
  - Design overview / Protocol description
- Implementation
  - Mapping to the OSI model
  - Cryptographic algorithms for encryption, authentication, signatures
  - Flexibility / Extensibility
  - Inter-operability issues
- Future work

Service/Network Management

- A "real" object is represented by a *managed object* in an *agent*
- A *manager* manages objects by interacting with their agents
- The collection of the managed objects in an agent domain is the Management Information Base (*MIB*)
- The manager and agent communicate by means of aManagement Information Protocol



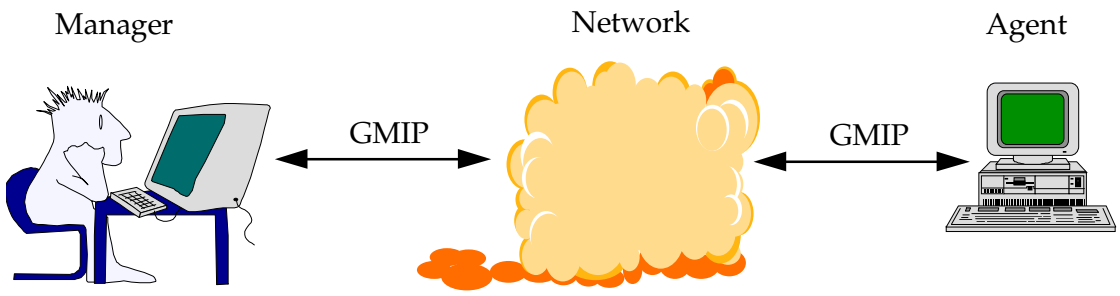
GMIP

- Common Management Information Protocol (CMIP)
  - A manager can retrieve the state of a managed object by issuing a GET message to its agent
  - The state can be changed by issuing a SET message
  - Operations on / by objects can be triggered by issuing an ACTION message
  - An agent can send notification to the manager using an EVENT message
- GMIP extends CMIP by supporting transactions
  - Many operations can be bound together and executed as one “atomic” operation

Security Issues

The problem: How to prevent unauthorized entities from accessing and modifying the MIB ?

Security model: Two "relatively secure" entities, communicating over an insecure network



Security Issues (cont.)

Threats

- Disclosure of sensitive information
- Unauthorized modification of network and service data
- Disruption or denial of service

Attacks

- Passive eavesdropping
- Replay
- Delaying messages
- Cut & paste
- Session interleaving
- Clogging

Cryptography

Symmetric cryptography

- *Message digest (hashing)* :
  - A function which produces a fixed size output on variable-size messages
  - It is hard to find two messages which yields the same output
- *Symmetric encryption / decryption* :
  - The parties share a secret key
  - They use it to encrypt / decrypt messages between them

Cryptography (cont.)

Public-key cryptography : Every party has a private-key and a corresponding public-key

- The public-key is known to everyone
- Only this party knows the private-key

- *Digital signatures:*
  - You need to know the private-key to sign a message
  - It is enough to know the public-key to verify a signature on a message
- *Public-key encryption / decryption*:*
  - It is enough to know the public-key to encrypt a message
  - You need to know the private-key to decrypt it

* We do not use public-key encryption in this design

Design Goals

- Scope of this work
  - Only addresses the protocol itself
  - No infrastructure (key distribution)
  - No local issues (access control, audit trail)
- Security services
  - *Origin authentication* - the sender is really who it claims to be
  - *Data integrity* - the data received is really what was sent by the sender
  - *Timeliness & sequencing* - the data received is "fresh" and in the right order
  - *Confidentiality* - only the intended receiver can read the data
  - *Availability* (as much as possible) - bogus messages can't abort a "good session"
- Security should not have a major effect on the performance
- Flexibility
  - It should be easy to add / remove cryptographic primitives
  - It should be easy to add / remove other security mechanisms

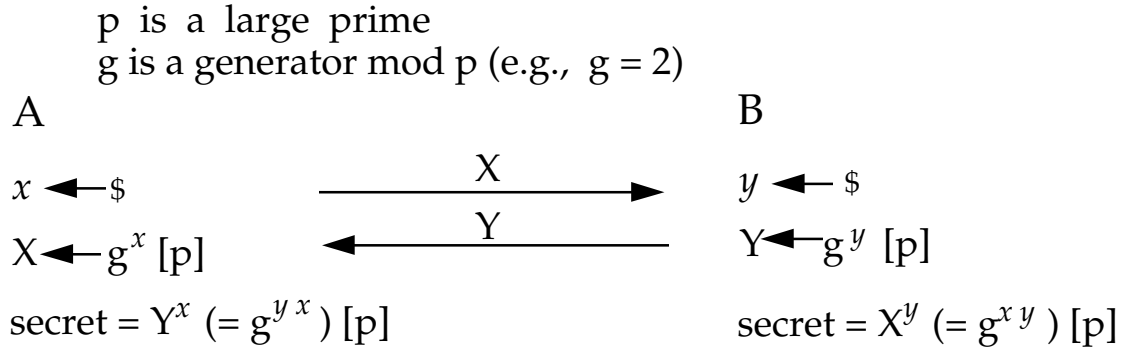
Design Choices

- Security-design for a connection-oriented protocol
  - Creating a security-context in session start-up, using it throughout the session
  - Using interaction to guarantee “freshness” (no time-stamps)
- Use of public-key cryptography
  - The parties should know each other’s public key
  - We do not need a secure on-line key-distribution-center

  but
  - Public-key algorithms are computationally expensive. We use them only during session start-up, in order to establish shared secret keys
- Providing an efficient re-key negotiation scheme
  - It may be risky to use the same key for all the messages in a very long session.
  - Choosing a new key can be much more efficient than setting up a new session.

Design Choices (cont.)

- Using the Diffie-Hellman key-exchange scheme



- We only use private / public keys to authenticate the DH-values
- The secret does not depend on long-term private keys
  - If the long-term keys are later compromised, it has no effect on the secrecy of the secret

Design Overview

The security design consists of three parts

- Session start-up
  - The parties authenticate each other and generate shared secret keys
- Security transformations
  - The parties use the shared secret keys to authenticate and/or encrypt messages during the session
- Re-key negotiation
  - The parties can negotiate new share secret keys whenever any of them feels that the current keys are no longer secure

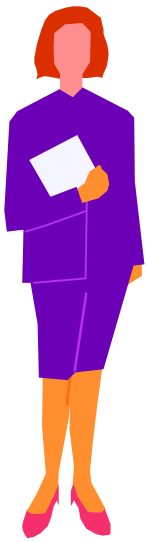
GTE **Laboratories**

Session start up (1)

Initialization request

Initiator (Alice)

Responder (Bob)



- I am Alice
- I want to start a session with Bob
- This is my public-key and a certificate for it
- Here is my public DH-value
- This is another random-value for use with the keys

- Signed - Alice

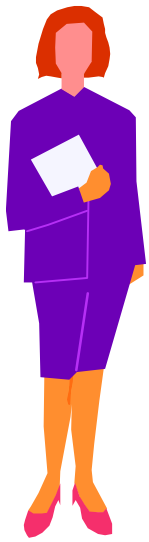
1) Bob verifies the certificate
2) Then he verifies the signature using the given public key
3) if everything goes well, he stores the DH value and the random value

GTE **Laboratories**

Session start up (2)

Initialization response

Initiator (Alice) Responder (Bob)



- I am Bob
- I am willing to start a session with Alice
- This is my public-key and a certificate for it
- Here is my public DH-value
- I have seen Alice's random-value
- Here is my own random-value for use with the keys

- Signed - Bob

1) Alice verifies the certificate
2) Then she verifies the signature using the given public key
3) if everything goes well, she stores Bob's DH value and his random value

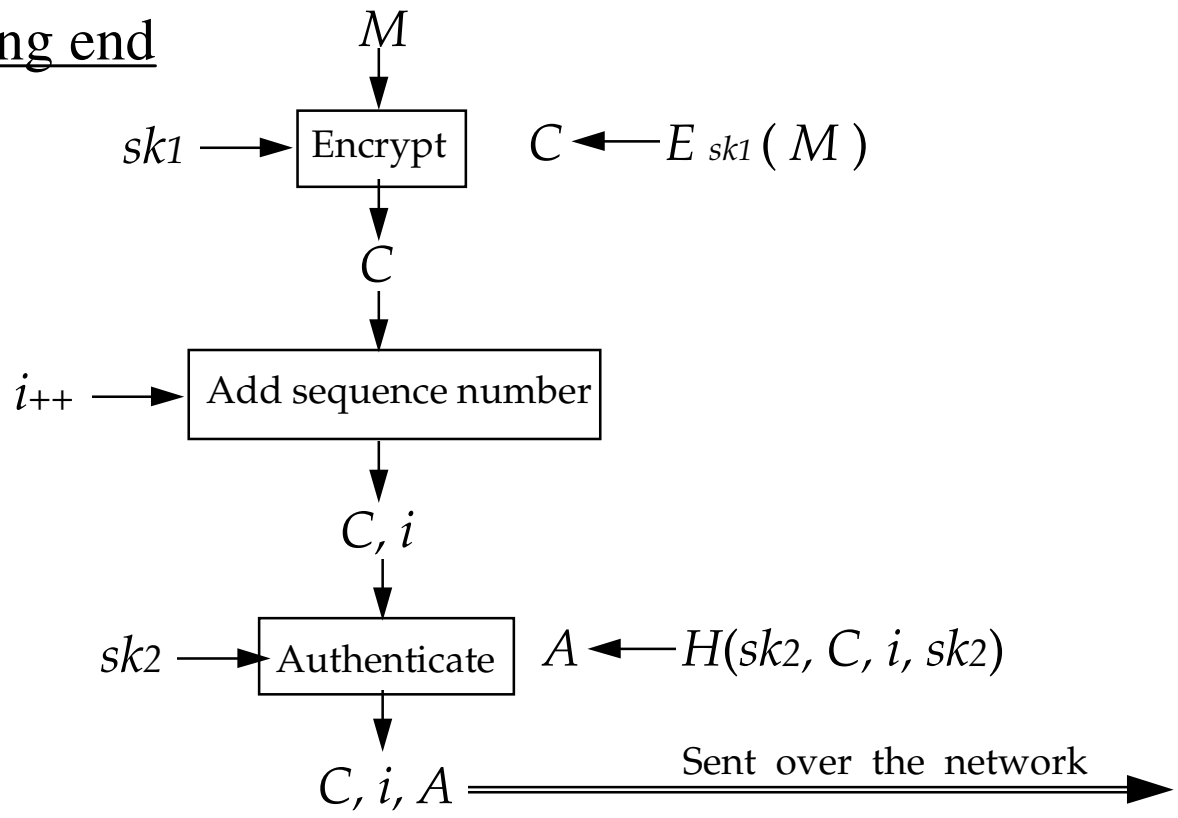
Session start up (3)

- Using the DH-values, Alice and Bob compute a shared secret (which we call the *master key*)
- Using the master key and the two random values, they compute the session keys

- Then, Alice sends a message to Bob and authenticate it with the session keys
- Bob is willing to send messages to Alice only after he sees Alice's first message
  - Otherwise, the "adversary" could replaye old init-request messages from Alice and cause Bob to send messages to her, even though Alice is not there at all
  - Even in this case, the adversary will not be able to figure out the session key

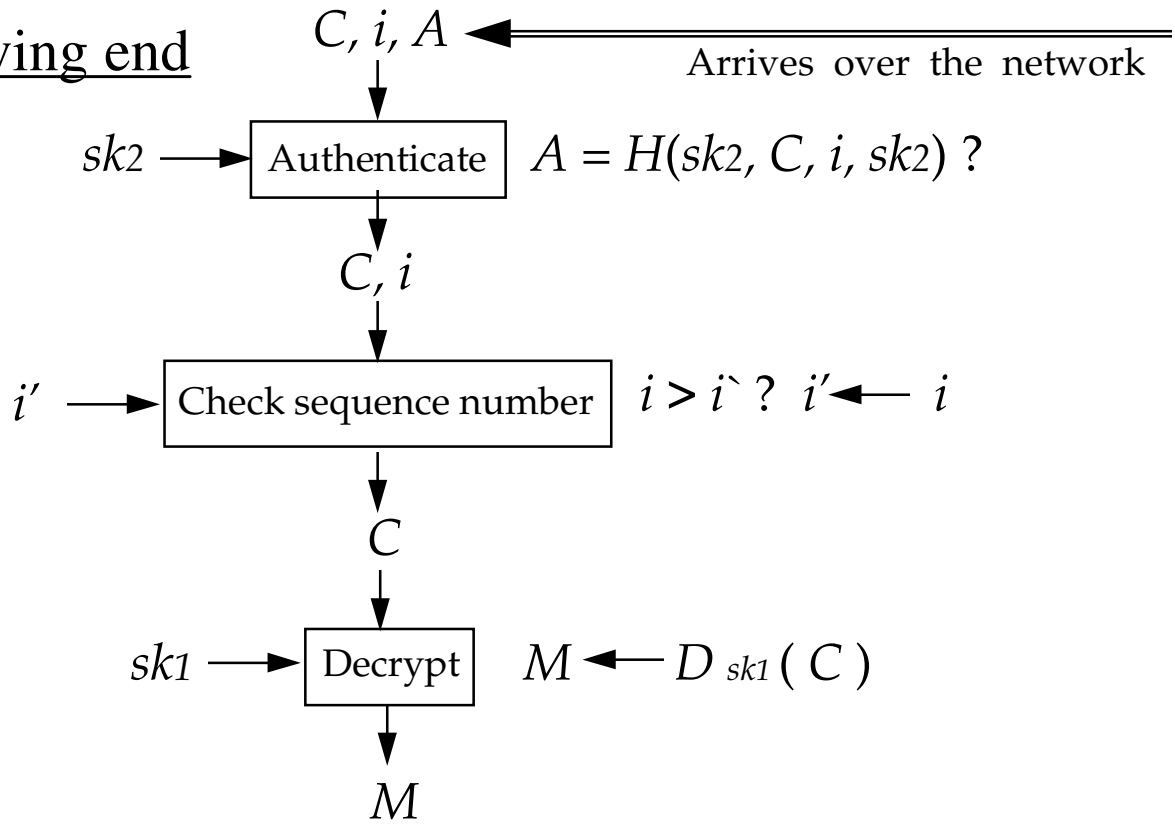
Security Transformations (1)

The sending end

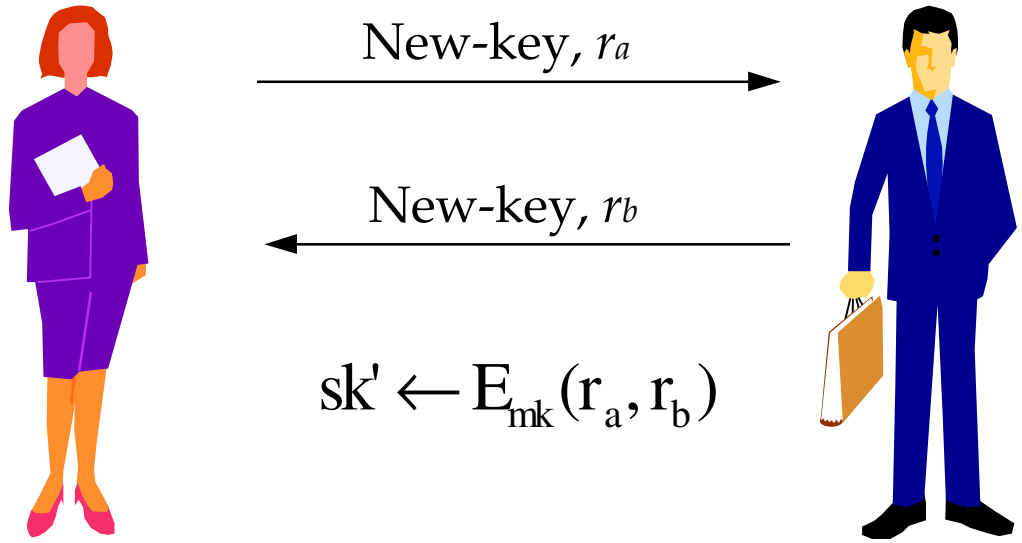


Security Transformations (2)

The receiving end



Re-key negotiation



The previous session key is kept around for a while

Implementation

- Session start-up negotiation
  - Handled by the application layer. Controlled by the application using GMIP
  - Uses the ACSE messages to pass the information
  - Also includes negotiation of the cryptographic algorithms
  - Requires encoding of the message in a distinguish syntax (DER), for signature generation / verification
  - Security transformations are *not* applied to the the initial messages
- Security Transformations
  - Handled by the presentation layer
  - Performed on the transfer syntax, do not require distinguish syntax
- Re-key negotiation
  - Is triggered automatically by the security-transformations when the current session-key reaches the middle of its *time-to-live*
  - Can also be started directly by the application

GTE **Laboratories**

Cryptographic Algorithms

- Symmetric Encryption: DES, Triple-DES, IDEA, RC5
- Digital Signatures: RSA, DSA, Micali's algorithm
- Message Digest: MD5, SHA

- Most of the cryptographic algorithms are patented and require licensing
- Most of the code is publicly available (*RSAREF2*, *PGP*, other)

Flexibility / Extensibility

Two levels of extensibility:

- Dynamically adding / removing cryptographic algorithms
  - not implemented yet
- Hooks for installing / removing other security mechanisms (e.g., Kerberos, Session re-connect)
  - An application can register functions which implement a security mechanism. Then it can specify this mechanism in the INIT message and GMIP will use it
  - The application can also register call-back functions which are used by the security mechanism to ask the application for parameters (e.g., keys)
  - A function is provided for registering the GMIP security mechanism
  - If an application is not interested in security mechanisms, it can register and use the *nil-mechanism.* This results in using plain GMIP with no security

Inter-Operability

- Most of the security area is not standardized yet
- We tried to conform to standards where we could find them
  - The session start-up uses the ACSE authentication functional unit
  - We use the X.509 syntax for certificates
- Our design was influenced by the new IP-security standard proposals
  - In particular, our session start-up protocol was influenced by variants of the proposed Photuris key-exchange protocol [Karns 95]
- We hope that parts of our approach will eventually be incorporated in security standards
  - In particular, the use of authenticated Diffie-Hellman key-exchange, and the master-key/session-keys approach

Future work

P l e n t y

- Enhance the GMIP security mechanism
  - Implement dynamic addition / removal of cryptographic primitives, add support for more primitives, get better / faster code for primitives
  - Improve initial negotiation for primitives
  - Get performance measurements, optimize performance vs. security level
  - Work on standardization
  - Prove correctness of the protocol*
- Explore other mechanisms
- Design /implement access control policies and audit trail
- Design/implement key distribution system(s)