Vorlesung

Sicherheit: Fragen und Lösungsansätze

im Wintersemester 2012 / 2013

Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

Teil 6: Case Studies: PGP and Kerberos

v. 04.12.12

Themen der Vorlesung

Part I: Challenges and Basic Approaches

1) Interests, Requirements, Challenges, and Vulnerabilities
2) Key Ideas and Combined Techniques

Part II: Control and Monitoring

3) Fundamentals of Control and Monitoring
4) Case Study: UNIX

Part III: Cryptography

5) Fundamentals of Cryptography
6) **Case Studies: PGP and Kerberos**
7) Symmetric Encryption
8) Asymmetric Encryption and Digital Signatures with RSA
9) Some Further Cryptographic Protocols

Part IV: Access Control

10) Discretionary Access Control and Privileges
11) Mandatory Access Control and Security Levels

Part V: Security Architecture

12) Layered Design Including Certificates and Credentials
13) Intrusion Detection and Reaction

Pretty Good Privacy (PGP)

- supports participants of a distributed computing system in autonomously enforcing their security interests (*confidentiality*, *integrity* as *detection of modification*, *authenticity*, *non-repudiation*)
- provides a user-friendly interface to *encryption* and *authentication* (*digital signatures*) to be employed
  - explicitly by means of a simple command language
  - transparently embedded into some appropriate application software
- may serve
  - to protect *files* on a local computer
  - to ensure *end-to-end security* in a global environment
- assists participants with the necessary *key management*, including assessment
  - of claims that a public key belongs to a specific partner
  - of the *trust* in the respective issuers of such claims

Basic blocks

- *symmetric encryption* by a *block cipher* (IDEA, Triple-DES, AES, ...), extended into a *stream cipher* with *cipher block chaining* (*CBC*) mode:
  applied to
  - plaintexts (files to be stored or messages to be sent)
  - private asymmetric (decryption or signature) keys

- *asymmetric encryption* (RSA, ElGamal, ...) within *hybrid encryption*:
  applied to secret session keys for symmetric encryption

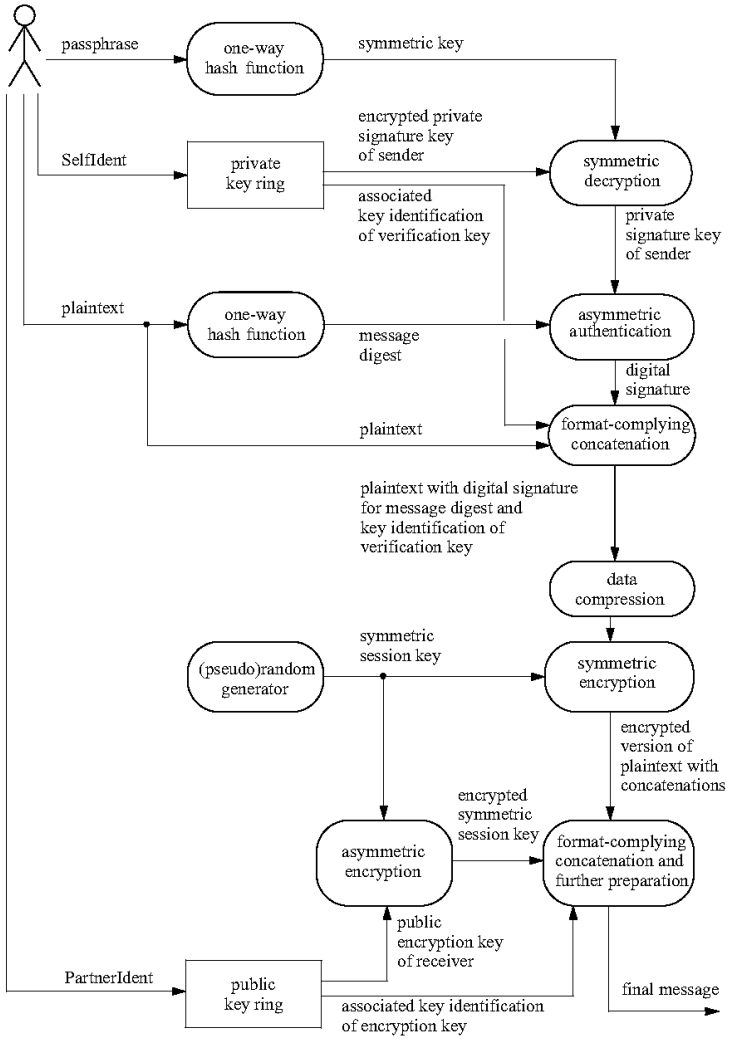
- *authentication* by *digital signatures* (RSA, ElGamal, ...)

- *one-way hash function* (MD5, ...):
  to generate
  - a message digest from an original message
  - a symmetric key from a passphrase

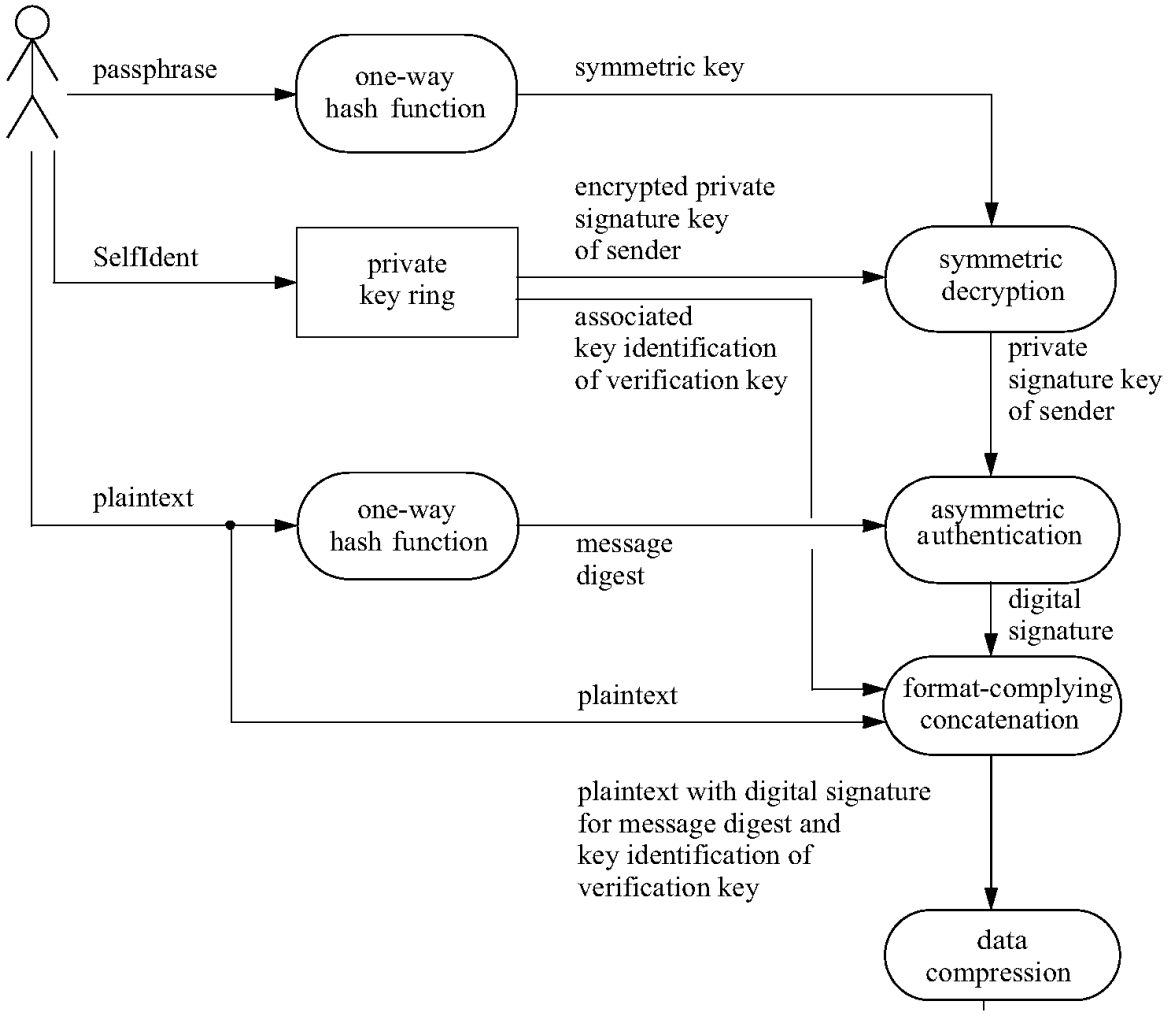
Basic blocks

- *random generator* or *pseudorandom generator*:
  for generating symmetric session keys
- *data compression*:
  for reducing the redundancy of plaintexts
- *passphrases*:
  for generating symmetric keys
  - to protect private asymmetric (decryption or signature) keys
  - for secure end-to-end connections
  - to protect the user's own files
- *key management* by means of a *private key ring* and a *public key ring*:
  - for storing the user's own private asymmetric keys
  - for storing, assessing and selecting the public asymmetric keys of the user's partners

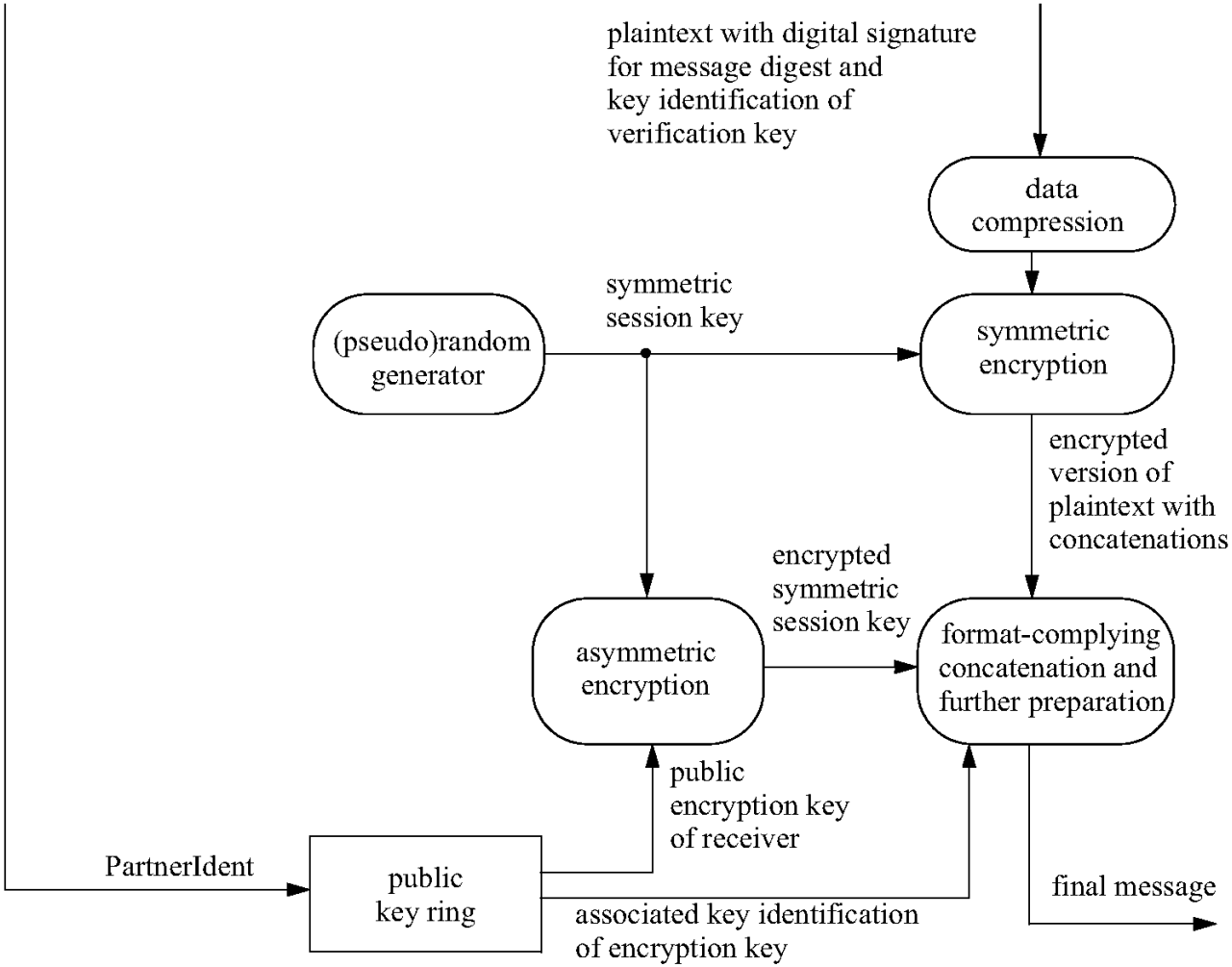
Conceptual design of secure message transmission



Secure message transmission: preparations



Secure message transmission: encryption and finalization



PGP parameters

- *SelfIdent*,
  denoting the participant acting as a sender

- *passphrase*,
  as an exhibit for a proof of authenticity of the sender

- *PartnerIdent*,
  denoting the intended receiver

- *plaintext*,
  to be communicated from the sender to the receiver

Key management

- only the keys for the asymmetric mechanisms are stored persistently
- a secret key for any symmetric mechanism employed is
  - generated or recovered only when it is actually needed
  - afterwards immediately destroyed

Using a symmetric secret key for securing an asymmetric private key

- authentication is strongly needed
  (owner is distinguished among all other participants):
  - authentication by demanding a passphrase
  - from which the secret key is directly derived by a one-way hash function

- the secret key is never stored persistently
  but is always dynamically regenerated whenever it is required

- the task of keeping secret information is reduced to
the burden of handling the passphrases,
  and thus is mainly shifted to the users of PGP in diminished form

Using a symmetric secret key as a session key for the hybrid method

- the symmetric secret key is generated on the fly
  by a (pseudo)random generator,

  used only once for encrypting content data
  by means of the block cipher employed,

  and then itself asymmetrically encrypted for later use
  when the content data must be recovered

- on the side of the participant acting as the encryptor,
there is no need to keep the secret key

- on the side of the participant acting as the later decryptor,
the secret key is held in encrypted form:
  when the non-encrypted form of the secret key is recovered,
  the first case applies, since authentication is strongly needed

Private key ring

- this ring contains the user's own key pairs, each of which consists of
  - a *private signature key* and
  - the matching *public verification key*

  or
  - a *private decryption key* and
  - the matching *public encryption key*

- each private key is stored in encrypted form

- each private key is stored together with
  - a *timestamp*
  - a derived *key identification* for referencing the key pair
  - an *identification* of the owner
  - some further administrative data

- the access to a private key is secured by a passphrase that the owner selected when he issued the PGP command to generate and store a key pair

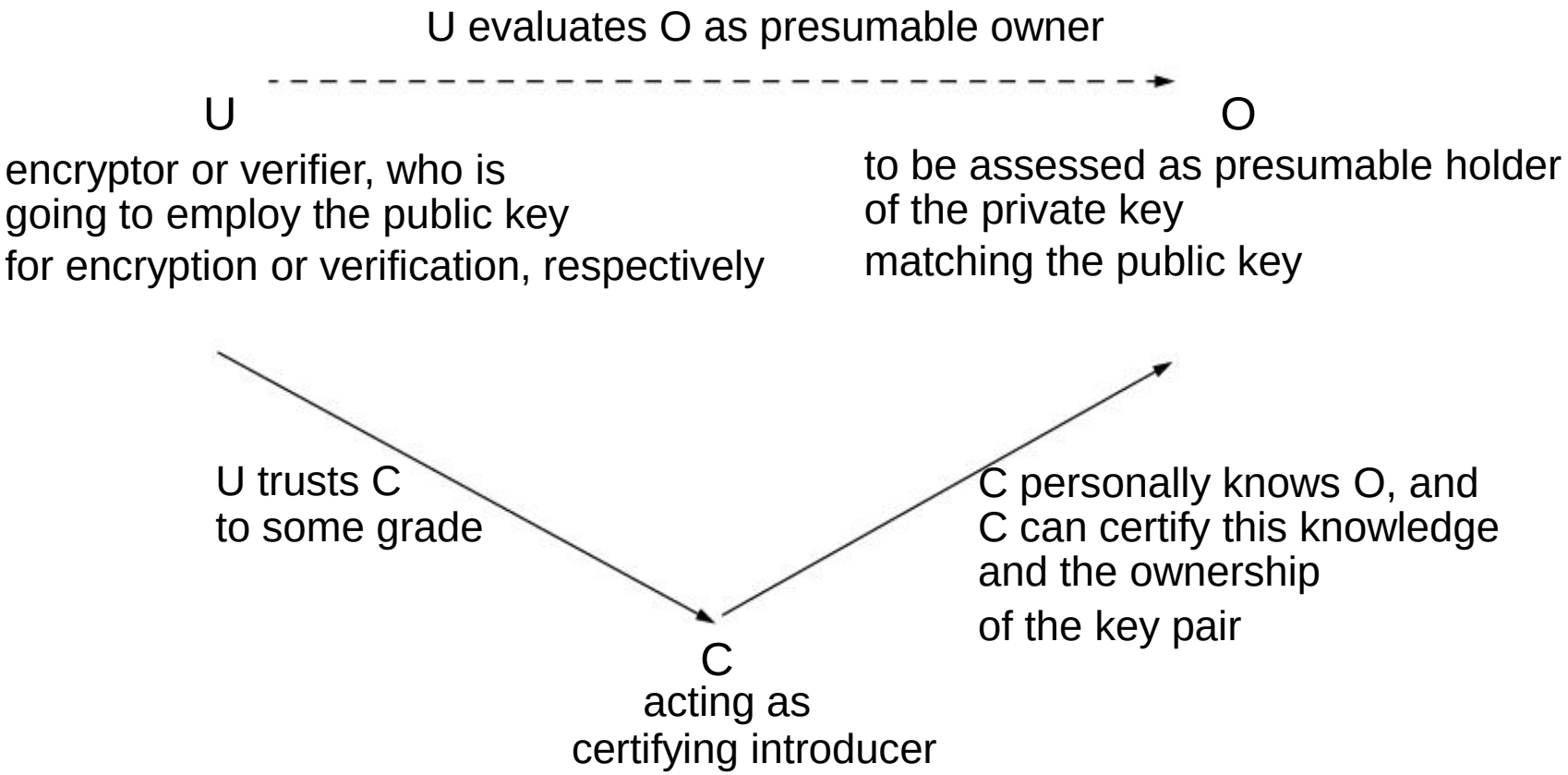
Public key ring

- this ring contains the
  - *public verification keys* and
  - *public encryption keys*

  of the owner's communication partners

- a key is complemented by
  - a *timestamp*
  - a derived *key identification*
  - an *identification* of the partner
  - further administrative data
  - some further entries to be used to assess the public key

Assessment of public keys



Two basic relationships

- one participant *C(ertifier) personally knows* another participant *O(wner)*

such that *C* can certify that a public key *k* belongs to *O*:

the participant *O* is the legitimate owner of the pertinent key pair and thus the actual holder of the matching private key;

the participant *C* (perceived as the *introducer* of *O*) confirms such an ownership by issuing and digitally signing a *key certificate*, also known as an *identity certificate*, basically consisting of

- an identification *OIdent*
- the public key *k* together with the pertinent digital signature

- one participant *U(ser)*, willing to encrypt or to verify a message,

may *trust* another participant *C(ertifier)* to various degrees to issue correct key certificates;

PGP suggests four *trust grades* (more sophisticated grades could be used):
unknown, *untrusted*, *marginally_trusted*,*completely_trusted*

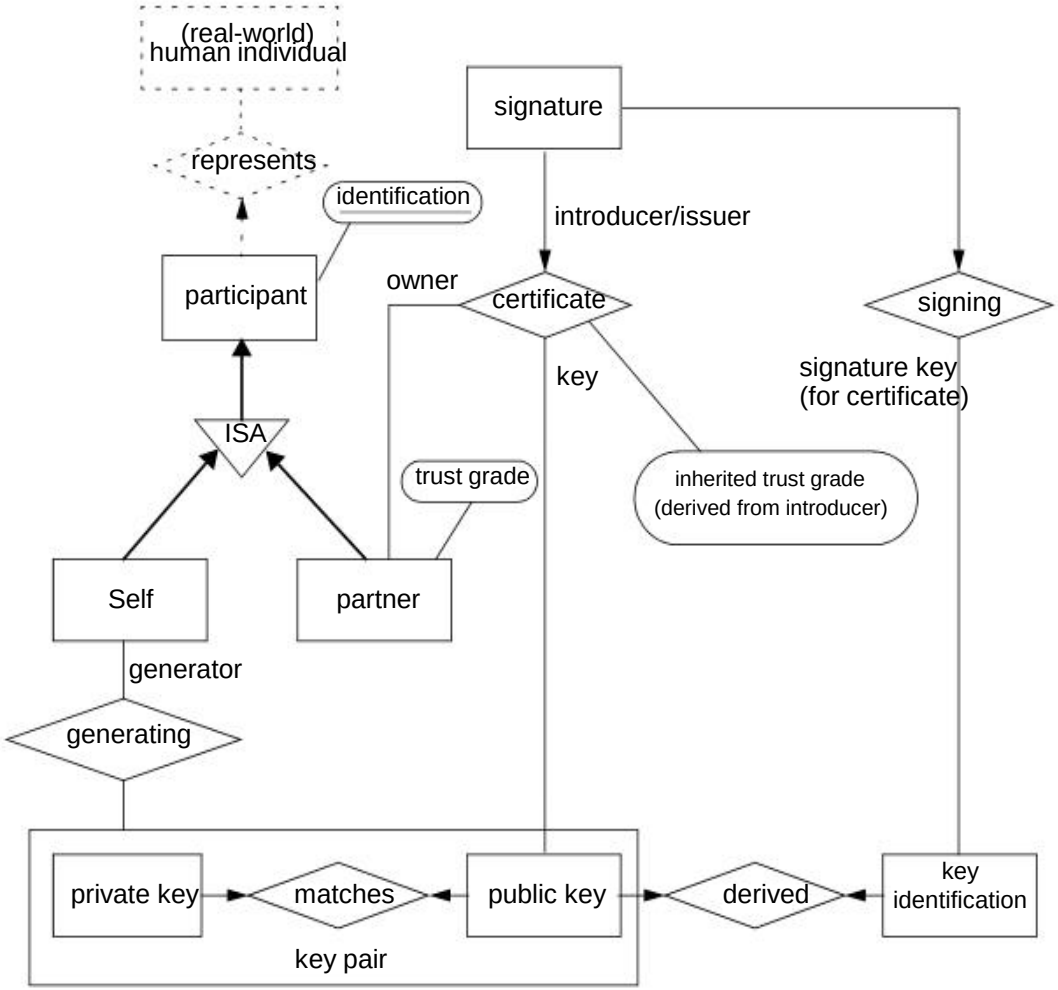
A derived relationship

- the participant *U(ser) evaluates* another participant *O(wner)*
as the presumable owner of a public key *k*,

  on the basis of successfully verifying
  the digital signature of a key certificate
  of the form (*OIdent* , *k*)*signature* ,
  issued and digitally signed by some introducer *C(ertifier)*

- the grade of the evaluation of *O* is derived
  from the grade of the trust in the introducer *C*

Participants, asymmetric keys, signatures and their relationships



Kerberos

- supports participants,
  may be unknown to each other before interacting,
  who are acting in a *distributed* computing system,
  - as a (functional) server
  - as a client

- enables servers to specify and enforce a *security policy*
that describes the permissions of potential clients

- initializes and maintains secure *end-to-end connections*
  that achieve mutual *authenticity* and enforce *confidentiality*

- proposes the use of a *trusted third party*, known as a Kerberos server,
to dynamically act as a *mediator* on a request from of a client,
on the basis of statically agreed relationships
  between the participants and the Kerberos server

Overall security achievements and trust

- participants assign *trust* to the Kerberos server:
  - each of the participants and the Kerberos server have to initially exchange a *secret* (*key*) for enabling symmetric authentication
  - a server has to permanently delegate the granting of permissions to the Kerberos server
  - however, within Kerberos, permission granting is degenerated to allow accesses whenever proper authentication has been achieved

Basic blocks

- *symmetric encryption*,
  - for evaluating the authenticity of messages
    on the basis of the possession of a secret symmetric *key*
  - for enforcing the confidentiality and integrity of messages

- *passwords*,
  used as substitutes for the secret symmetric key
  agreed between a particular participant and the Kerberos server

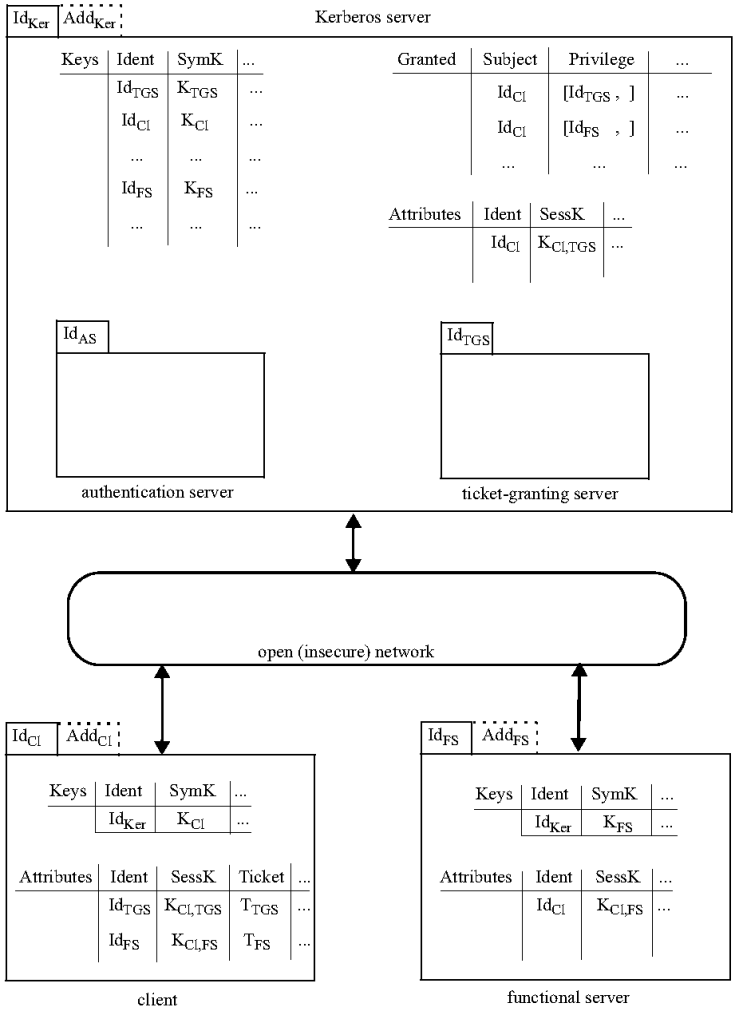
- *one-way hash function*
  for dynamically regenerating a key from the substituting password

- *random generator*
  to generate symmetric *session keys*,
  to be used for a secure *end-to-end connection*
  during a client-server interaction

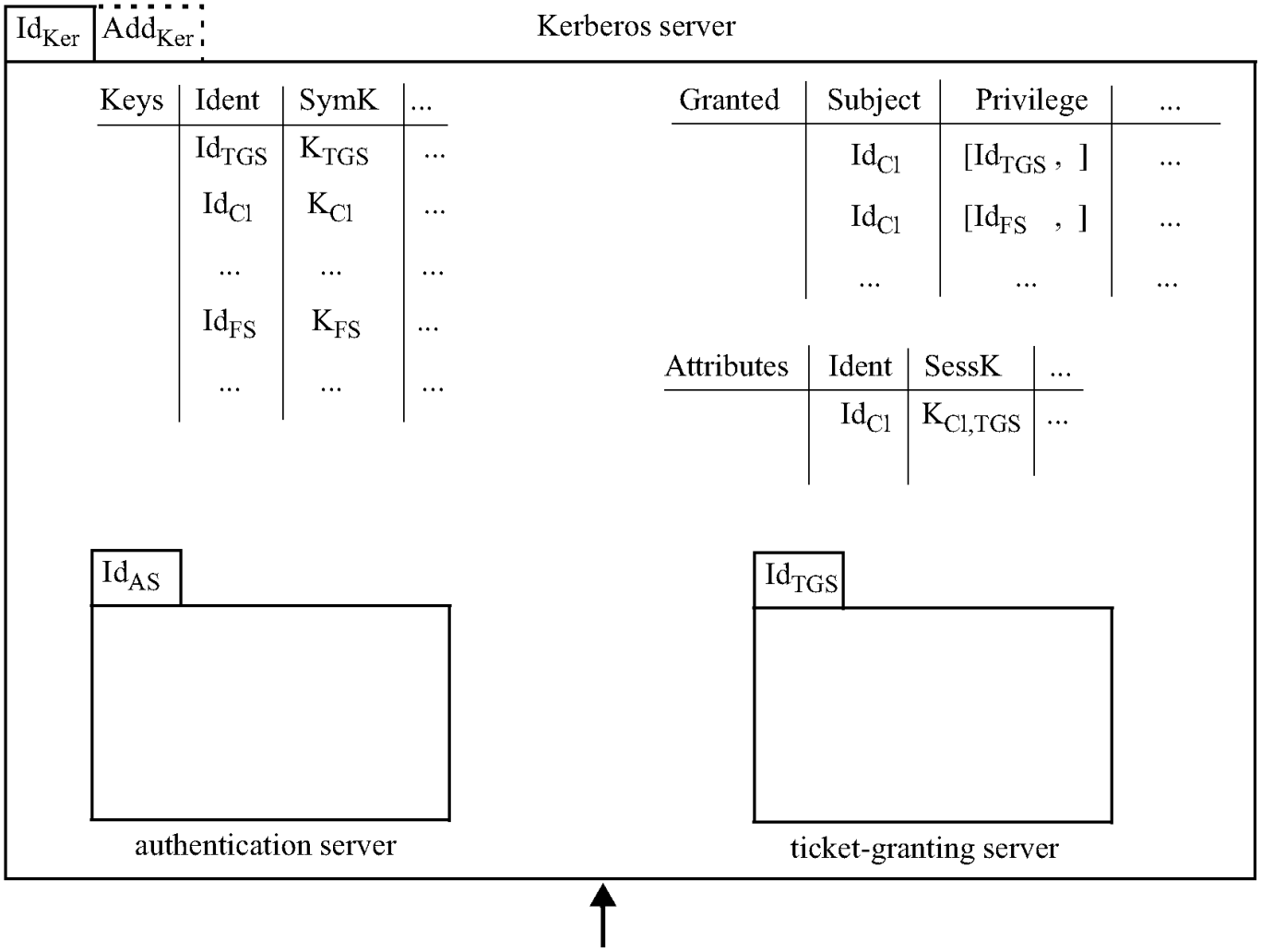
Basic blocks

- *timestamps*,
  used as indications of the *freshness* of messages
- *nonces* (random bit strings),
  used as *challenges* to be included in responses
- *tickets*,
  used as a special kind of *credential* that
  - encode *privileges* granted to a client as a grantee
  - are shown to a server as a (self-protecting) controlled object
- *validity* specifications for tickets
- *access decisions*,
  taken by a server on the basis of shown tickets
- *delegation*
  of the issuing of tickets by the Kerberos server on behalf of a server

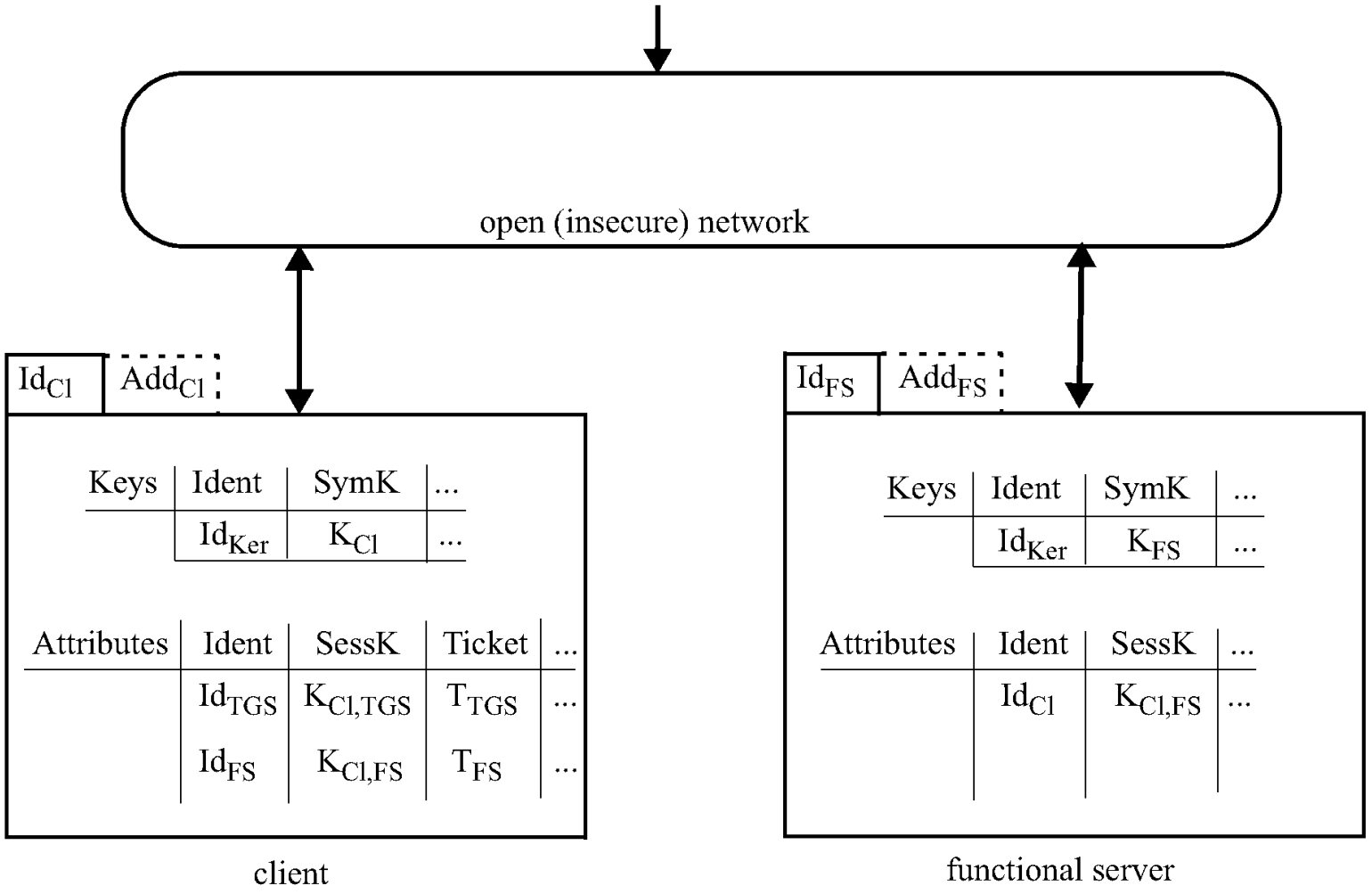
Conceptual design: structures



Structure of a Kerberos server



Structures of a client and a functional server



Names, identifiers, addresses and keys

- Kerberos server
  - *AS* *authentication server*
  - *TGS* *ticket-granting server*

- participant *P* (client *Cl*, Kerberos server *Ker* with components *AS* and *TGS*)
  - Id_P unique identifier
  - Add_P network address
  - K_P secret symmetric key for a symmetric encryption method

Kerberos server

- *Keys* local table with:
  - *Ident(ifier)* column for identifier *IdP*
  - *Sym(metric)K(ey)* column for key *KP* of each registered participant *P*
  - ... columns for further administrative data
- *Granted* local table with columns
  - *Subject*
  - *Privilege*

  to represent the *permissions* of clients to access services:
  - (Subject: Id_{Cl}, Privilege: [Id_{FS} ,]):
    the participant identified by *IdCl* is permitted, as a client,
    to access the services offered by the functional server identified by *IdFS*
  - (Subject: Id_{Cl} , Privilege: [Id_{TGS} ,]):
    the participant identified by *IdCl* is permitted, as a client,
    to access the service of the ticket-granting server,
    which is identified by *IdTGS* and is a component of the Kerberos server

A client

- *Keys* local table referring to the identifier Id_{Ker} of the Kerberos server

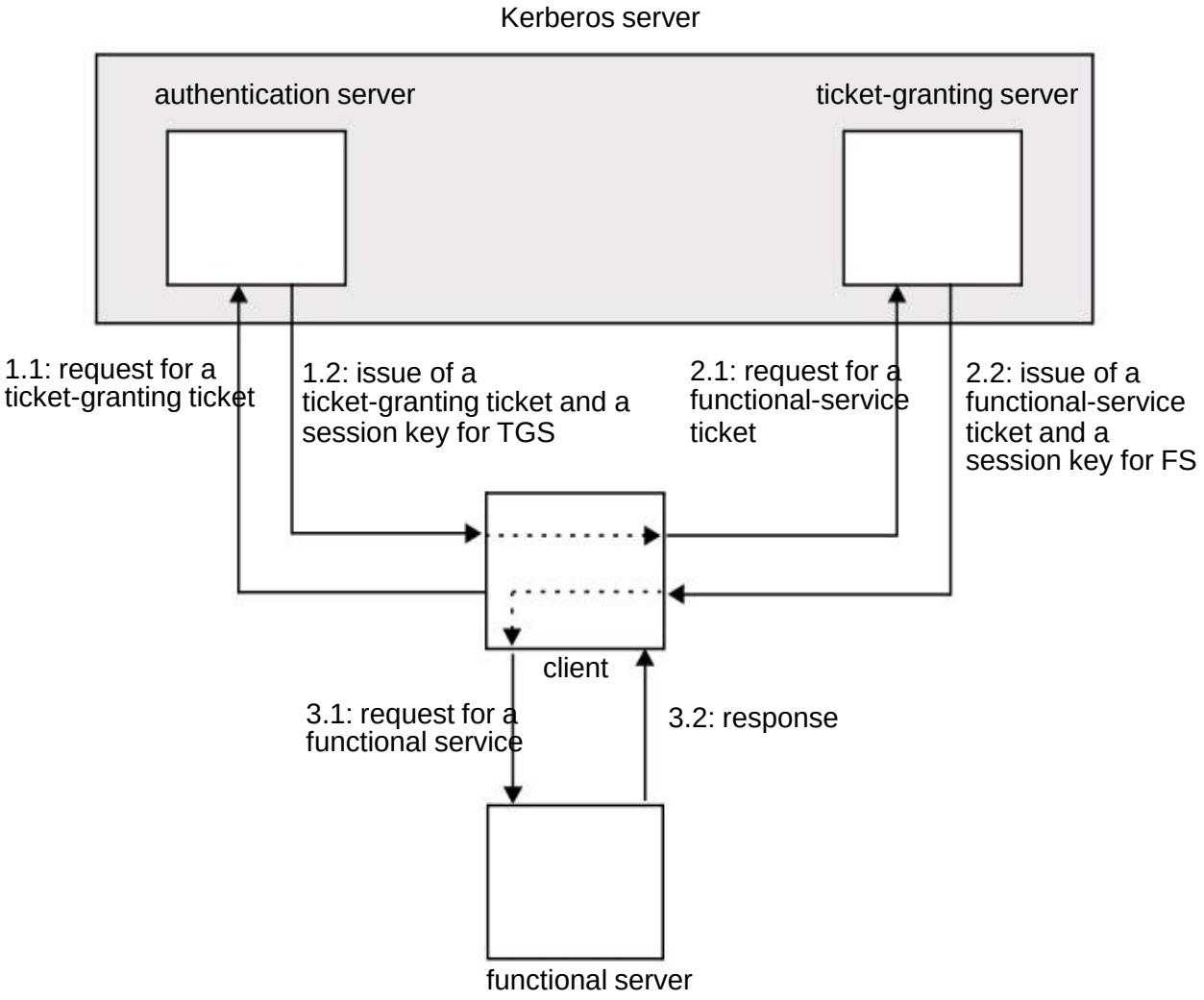
- however, for a human individual acting as a client, the secret symmetric key is *not* permanently stored:

  instead, the individual can choose a secret *password*, from which the symmetric key can be repeatedly computed by use of a *one-way hash function*

Rounds of the Kerberos protocol

- each round is initialized by a client and has two messages
- **first round,**
  executed once per client session (can be integrated within a login procedure):
  to authenticate the client for the later process of
  obtaining and exploiting a reusable *ticket*
  that expresses a *privilege* for a service
- **second round,**
  performed once for each functional server
  that is contacted during a client session:
  to actually grant the privilege to the client
- **third round,**
  repeatedly called for each actual *service invocation*:
  to exploit the granted privilege

Messages between a client, a Kerberos server and a functional server



Rough meanings of the six different Kerberos messages

- 1.1: a client requests a ticket-granting ticket from the authentication server
- 1.2: the authentication server issues a ticket-granting ticket for the client, together with a session key for a secure end-to-end connection between the client and the ticket-granting server
- 2.1: a client requests a functional-service ticket from the ticket-granting server
- 2.2: the ticket-granting server issues a functional-service ticket for the client, together with a session key for a secure end-to-end connection between the client and the functional server
- 3.1: a client requests a service invocation from the functional server
- 3.2: the functional server responds to the client

Simplified message 1.1

the client *Cl*

- requests a ticket-granting ticket from the authentication server *AS*, to be shown to the ticket-granting server *TGS*

- adds the wanted validity specification $Validity_1$

- includes a nonce $Nonce_1$

$$Id_{Cl}\ ,\ Id_{TGS}\ ,\ Validity_1\ ,\ Nonce_1$$

Simplified message 1.2

the authentication server *AS*

- issues a ticket-granting ticket $Ticket_{TGS}$ to the client *Cl*,

to be shown to the ticket-granting server *TGS*

- attaches
  - a session key $K_{Cl,TGS}$ for a secure end-to-end connection between the client *Cl* and the ticket-granting server *TGS*
  - the wanted $Validity_1$
  - the received $Nonce_1$

  where the attachments are encrypted with the client's secret key K_{Cl}

$$Id_{Cl} , Ticket_{TGS} , Enc(K_{Cl} , [K_{Cl,TGS} , Validity_1 , Nonce_1 , Id_{TGS}])$$

Ticket-granting ticket

the ticket-granting ticket $Ticket_{TGS}$ contains

- the session key $K_{Cl,TGS}$ for a secure end-to-end connection between the client *Cl* and the ticket-granting server *TGS*
- the client's identifier Id_{Cl}
- the client's network address Add_{Cl}
- the wanted $Validity_1$

and is encrypted with the ticket-granting server's secret key K_{TGS}

$$Ticket_{TGS} = Enc(K_{TGS} , [K_{Cl,TGS} , Id_{Cl} , Add_{Cl} , Validity_1])$$

Simplified message 2.1

showing the ticket $Ticket_{TGS}$, **the client *Cl***

- requests a functional-service ticket from the ticket-granting server *TGS*, to be shown to the functional server *FS*
- adds the wanted validity specification $Validity_2$
- includes a nonce $Nonce_2$
- attaches
  - an authentificator $Auth_{Cl,TGS}$ that encrypts the client's identifier Id_{Cl}
  - a timestamp TS_3

  where the authentificator is encrypted with the session key $K_{Cl,TGS}$
  (which is made available to the ticket-granting server by the ticket $Ticket_{TGS}$)

$$Id_{FS} , Validity_2 , Nonce_2 , Ticket_{TGS} , Auth_{Cl,TGS}$$

where

$$Auth_{Cl,TGS} = Enc(K_{Cl,TGS} , [Id_{Cl}, TS_3])$$

Simplified message 2.2

the ticket-granting server

- issues a functional-service ticket $Ticket_{FS}$ to the client *Cl*,

to be shown to the functional server *FS*

- attaches
  - a session key $K_{Cl,FS}$ for a secure end-to-end connection between the client *Cl* and the functional server *FS*
  - the wanted $Validity_2$
  - the received $Nonce_2$

  where the attachments are encrypted with the session key $K_{Cl,TGS}$

$$Id_{Cl} , Ticket_{FS} , Enc(K_{Cl,TGS} , [K_{Cl,FS} , Validity_2 , Nonce_2 , Id_{FS}])$$

Functional-service ticket

the functional-service ticket $Ticket_{FS}$ contains

- the session key $K_{Cl,FS}$ for a secure end-to-end connection between the client *Cl* and the functional server *FS*
- the client's identifier Id_{Cl}
- the client's network address Add_{Cl}
- the wanted $Validity_2$

and is encrypted with the functional server's secret key K_{FS}

$$Ticket_{FS} = Enc(K_{FS} , [K_{Cl,FS} , Id_{Cl} , Add_{Cl} , Validity_2])$$

Simplified message 3.1

showing the ticket $Ticket_{FS}$, **the client *Cl***

- requests a service invocation from the functional server *FS*

- includes
  - an authentificator $Auth_{Cl,FS}$
    that encrypts the client's identifier Id_{Cl}
  - atimestamp TS_4

  where the authentificator is encrypted with the session key $K_{Cl,FS}$
  (which is made available to the functional server by the ticket $Ticket_{FS}$)

$$Ticket_{FS} , Auth_{Cl,FS}$$

where

$$Auth_{Cl,FS} = Enc (K_{Cl,FS}, [Id_{Cl},TS_4])$$

Simplified message 3.2

the functional server *FS*

- responds to the client
  by sending back the received timestamp TS_4,

  encrypted with the session key $K_{Cl,FS}$

$$Enc(K_{Cl,FS}, TS_4)$$