Datasäkerhetsmetoder föreläsning 7

Nyckelhantering

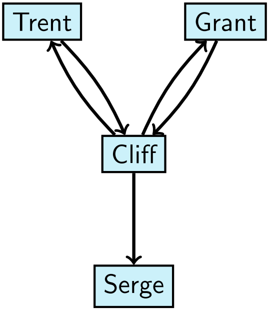
Jan-Åke Larsson

LINKÖPING UNIVERSITY

Cryptography

- A security *tool*, not a general solution
- Cryptography usually converts a communication security problem into a key management problem
- So now you must take care of the key security problem, which becomes a problem of computer security

Key management



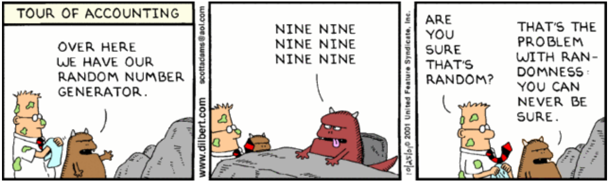
The problem is to

- generate
- distribute
- store
- use
- revoke

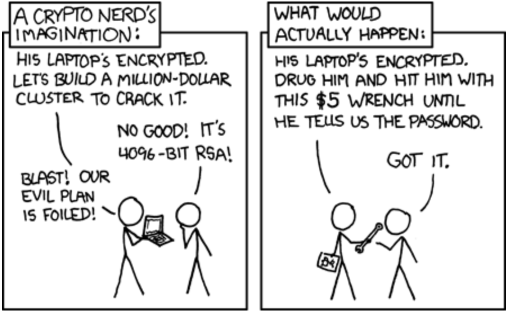
the key in a secure way

Key generation

- The key size decides how many different keys you can have, the search space for exhaustive key search
- If keys are not chosen at random, the attacker can first try more likely keys
- If all bit combinations are not used, security is given by the number of possible keys, not the size in bits
- If keys are generated from a known random seed, the size of that seed decides the security



Key length



A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

Key length

Table 7.1: Minimum symmetric key-size in bits for various attackers

Attacker	Budget	Hardware	Min security	(1996)
"Hacker"	0	PC	53	45
	$< \$400$	PC(s)/FPGA	58	50
	0	"Malware"	73	
Small organization	$10k	PC(s)/FPGA	64	55
Medium organization	$300k	FPGA/ASIC	68	60
Large organization	$10M	FPGA/ASIC	78	70
Intelligence agency	$300M	ASIC	84	75

From "ECRYPT II Yearly Report on Algorithms and Keysizes (2009-2010)"

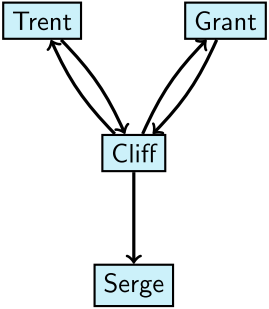
Key length

Table 7.1: Minimum symmetric key-size in bits for various attackers

Attacker	Budget	Hardware	Min security	(1996)
"Hacker"	0	PC	58	45
	$< \$400$	PC(s)/FPGA	63	50
	0	"Malware"	77	
Small organization	$10k	PC(s)/FPGA	69	55
Medium organization	$300k	FPGA/ASIC	69	60
Large organization	$10M	FPGA/ASIC	78	70
Intelligence agency	$300M	ASIC	84	75

From "ECRYPT II Yearly Report on Algorithms and Keysizes (2011-2012)"

Key establishment and authentication



- Once upon a time, protocols establishing a session key was called authentication protocols
- This is no longer the case
- Kerberos (to the left) is known mainly as an authentication protocol
- The end result is an authorization ticket that contains a “session key”

Key Management

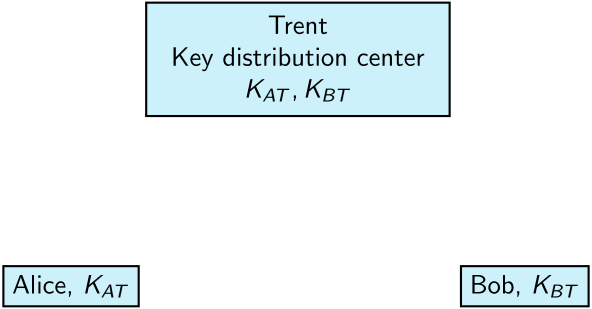
- The first key in a new connection or association is *always* delivered via a courier

- Once you have a key, you can use that to send new keys

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can exchange a key via Trent (provided they both trust Trent)

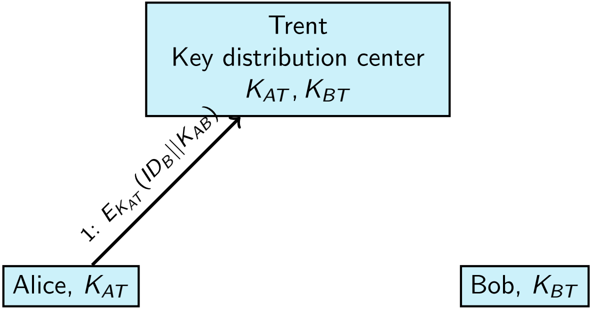
Key distribution center

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can exchange a key via Trent (provided they both trust Trent)



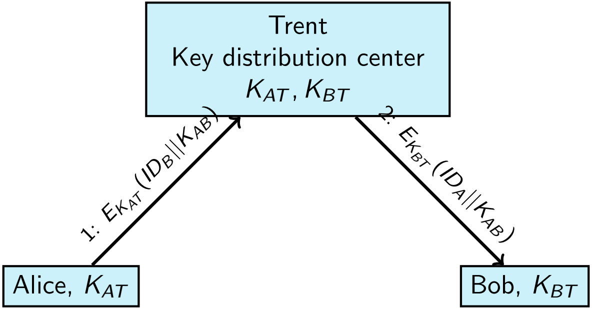
Key distribution center

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can exchange a key via Trent (provided they both trust Trent)



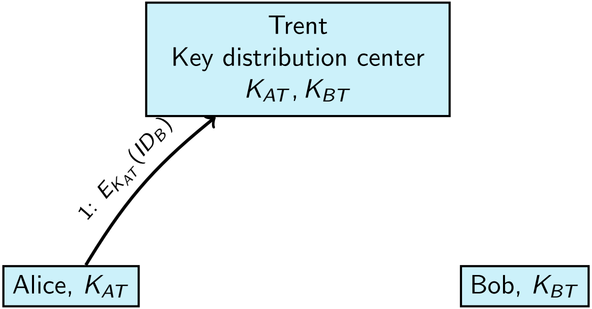
Key distribution center

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can exchange a key via Trent (provided they both trust Trent)



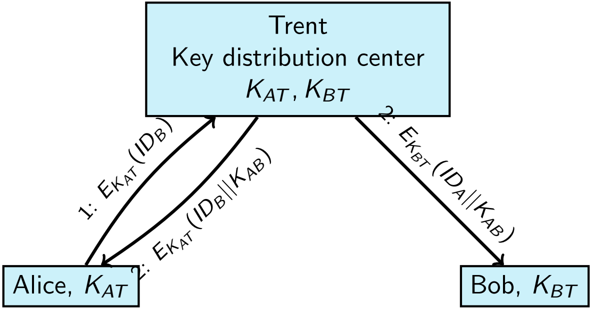
Key distribution center, key server

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can receive a key from Trent (provided they both trust Trent)



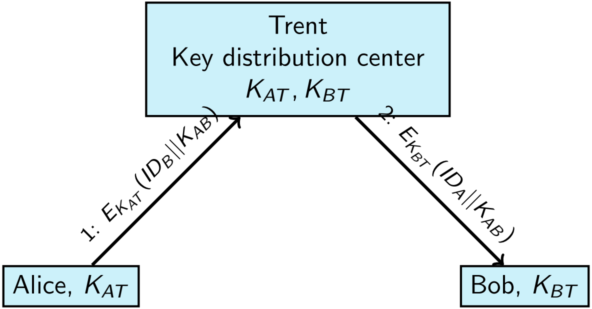
Key distribution center, key server

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can receive a key from Trent (provided they both trust Trent)



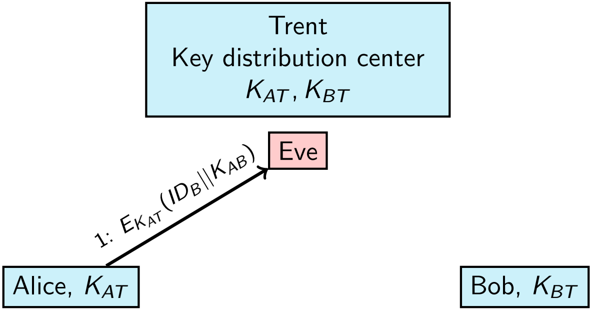
Key distribution center

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can exchange a key via Trent (provided they both trust Trent)



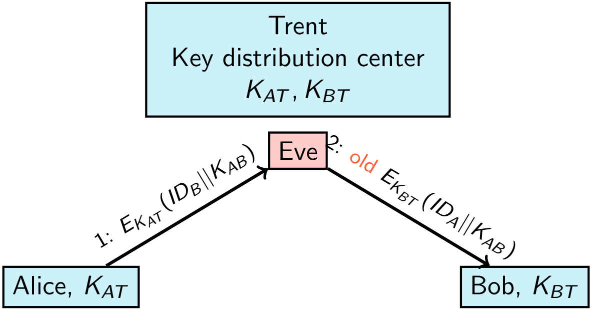
Key distribution center, replay attacks

- But perhaps Eve has broken a previously used key, and intercepts Alice's request



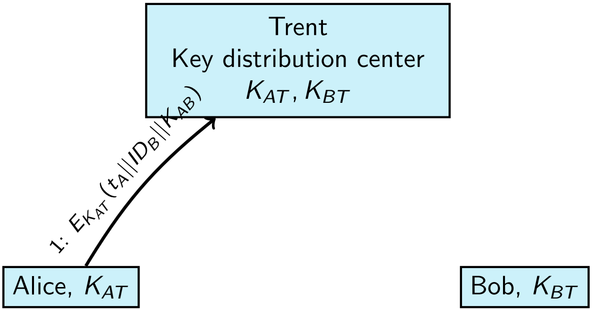
Key distribution center, replay attacks

- But perhaps Eve has broken a previously used key, and intercepts Alice's request
- Then she can fool Bob into communicating with her



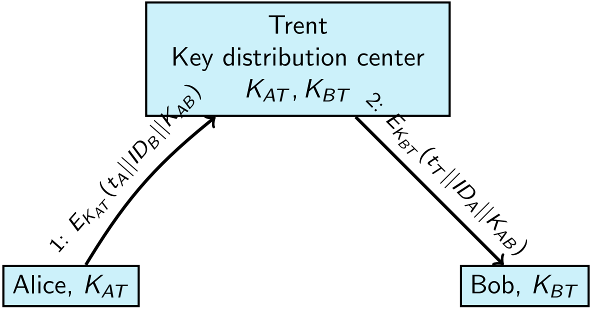
Key distribution center, wide-mouthed frog

- Alice and Trent add time stamps to prohibit the attack



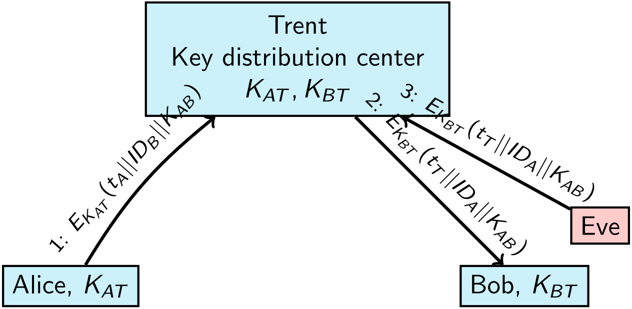
Key distribution center, wide-mouthed frog

- Alice and Trent add time stamps to prohibit the attack



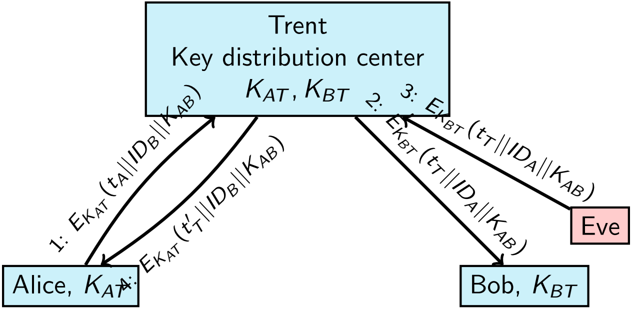
Key distribution center, wide-mouthed frog

- Alice and Trent add time stamps to prohibit the attack
- But now, Eve can pretend to be Bob and make a request to Trent



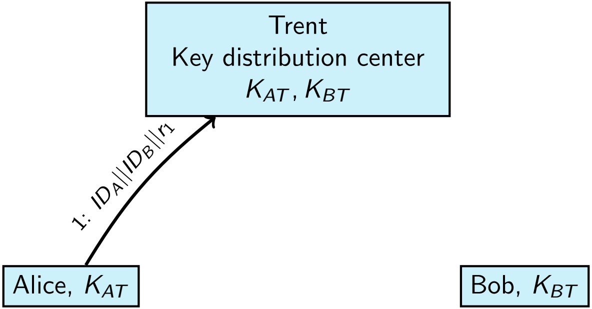
Key distribution center, wide-mouthed frog

- Alice and Trent add time stamps to prohibit the attack
- But now, Eve can pretend to be Bob and make a request to Trent, who will forward the key to Alice



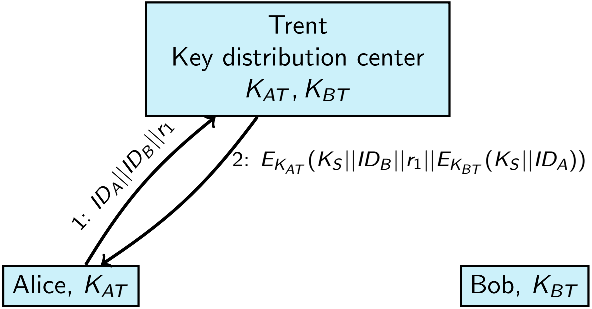
Key distribution center, Needham-Schroeder key agreement

- Another variation is to use nonces to prohibit the replay attack



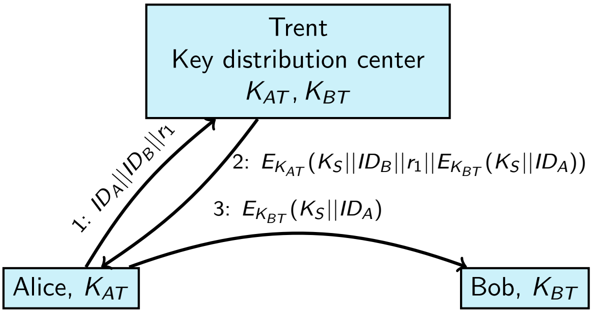
Key distribution center, Needham-Schroeder key agreement

- Another variation is to use nonces to prohibit the replay attack



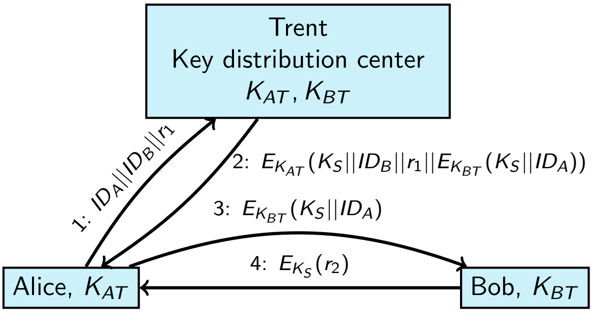
Key distribution center, Needham-Schroeder key agreement

- Another variation is to use nonces to prohibit the replay attack



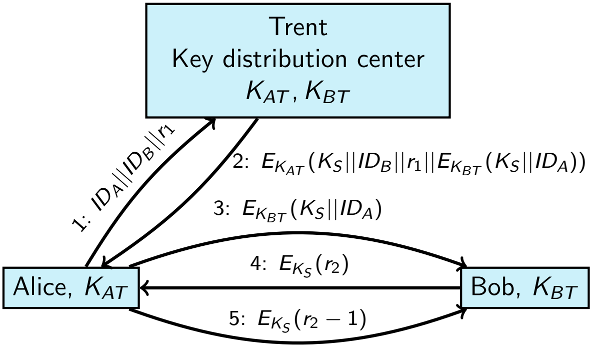
Key distribution center, Needham-Schroeder key agreement

- Another variation is to use nonces to prohibit the replay attack



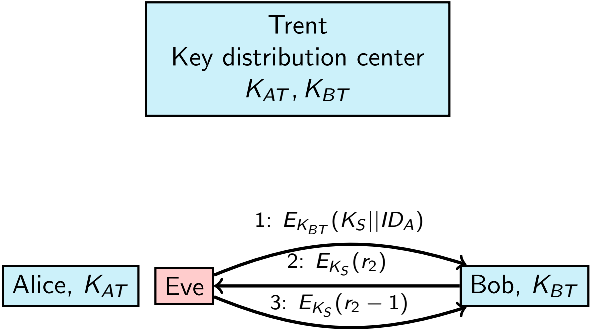
Key distribution center, Needham-Schroeder key agreement

- Another variation is to use nonces to prohibit the replay attack

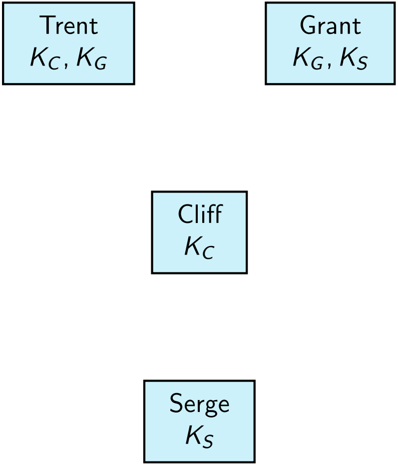


Key distribution center, Needham-Schroeder key agreement

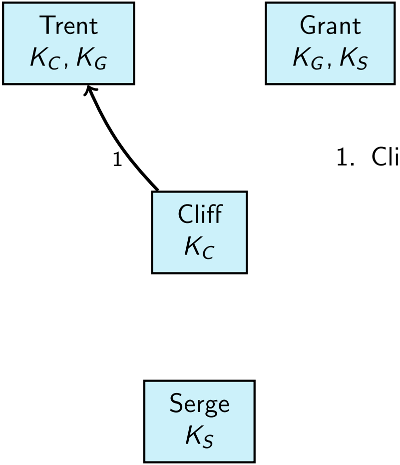
- Another variation is to use nonces to prohibit the replay attack
- If Eve ever breaks one session key, she can get Bob to reuse it



Kerberos

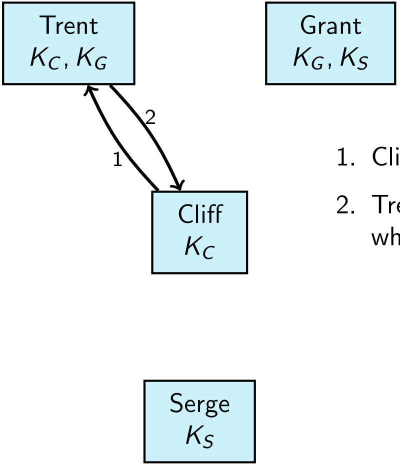


Kerberos



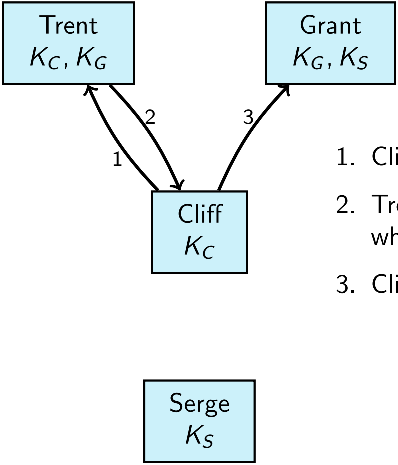
1. Cliff sends Trent $ID_C || ID_G$

Kerberos



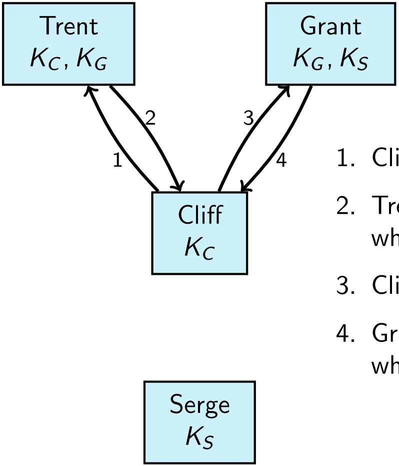
1. Cliff sends Trent $ID_C||ID_G$
2. Trent responds width $E_{K_C}(K_{CG})||TGT$
   where $TGT = ID_G||E_{K_G}(ID_C||t_1||K_{GC})$

Kerberos



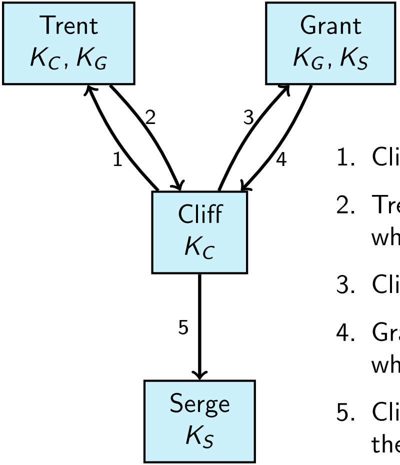
1. Cliff sends Trent $ID_C || ID_G$
2. Trent responds width $E_{K_C}(K_{CG}) || TGT$
   where $TGT = ID_G || E_{K_G}(ID_C || t_1 || K_{GC})$
3. Cliff sends Grant $E_{K_{CG}}(ID_C || t_2) || TGT$

Kerberos



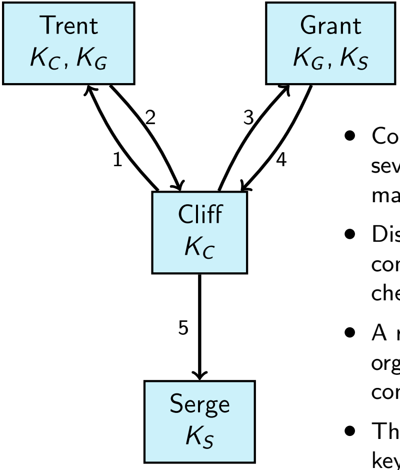
1. Cliff sends Trent $ID_C || ID_G$
2. Trent responds width $E_{K_C}(K_{CG}) || TGT$
   where $TGT = ID_G || E_{K_G}(ID_C || t_1 || K_{GC})$
3. Cliff sends Grant $E_{K_{CG}}(ID_C || t_2) || TGT$
4. Grant responds with $E_{K_{CG}}(K_{CS}) || ST$
   where $ST = E_{K_S}(ID_C || t_3 || t_{\text{expir.}} || K_{CS})$

Kerberos



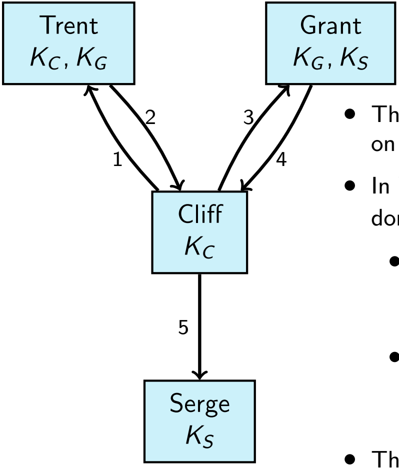
1. Cliff sends Trent $ID_C || ID_G$
2. Trent responds width $E_{K_C}(K_{CG}) || TGT$ where $TGT = ID_G || E_{K_G}(ID_C || t_1 || K_{GC})$
3. Cliff sends Grant $E_{K_{CG}}(ID_C || t_2) || TGT$
4. Grant responds with $E_{K_{CG}}(K_{CS}) || ST$ where $ST = E_{K_S}(ID_C || t_3 || t_{\text{expir.}} || K_{CS})$
5. Cliff sends Serge $E_{K_{CS}}(ID_C || t_4)$ and can then use Serge's services

Kerberos realms



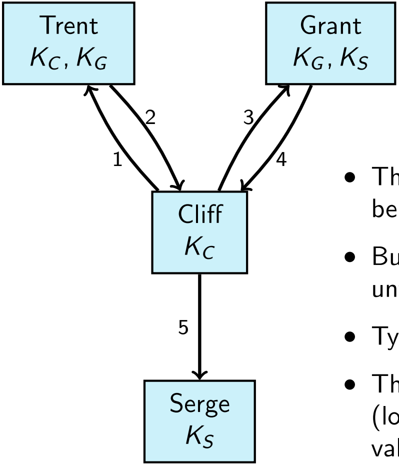
- Contains one authentication server (KAS), several authorization servers (TGS), and many services
- Distributed system, with centralized access control, a single security policy that is easy to check, and change
- A realm often corresponds to a single organization, and several realms can be connected
- This often is controlled by trust (shared keys), but also other considerations like contractual agreements

Controlled invocation in distributed systems



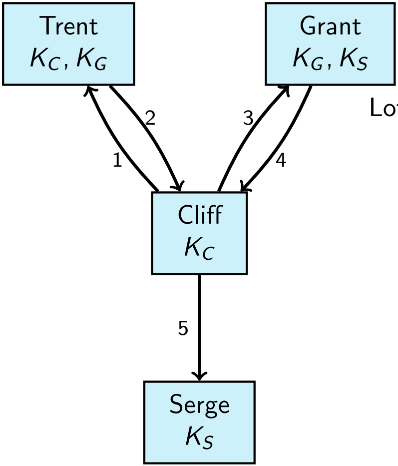
- The remote program (subject) needs to act on behalf of the user (principal)
- In Windows AD ($\sim$Kerberos), this can be done in two ways
  - "Proxy tickets" that are limited in the access rights, e.g., to one file for printing it
  - "Forwarded TGTs" that can be used to apply for new tickets on behalf of the user
- The latter is like lending out your password for the duration of the ticket

Revocation in Kerberos



- The access rights of the principal needs to be revoked at the TGS
- But issued tickets continue to be valid until they expire (TOCTTOU)
- Typically, KAS tickets is vaild for a day
- There is a tradeoff between convenience (long validity) and fast revocation (short validity)

Kerberos, more comments

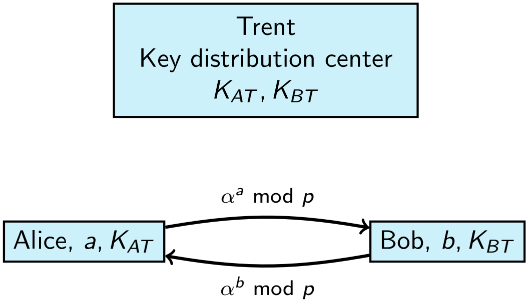


Lots of technical details:

- Clock sync
- Timestamp skew window
- Online servers (Availability)
- Trusting servers
- Password security
- Client machine security
- DOSing the KAS
- ...

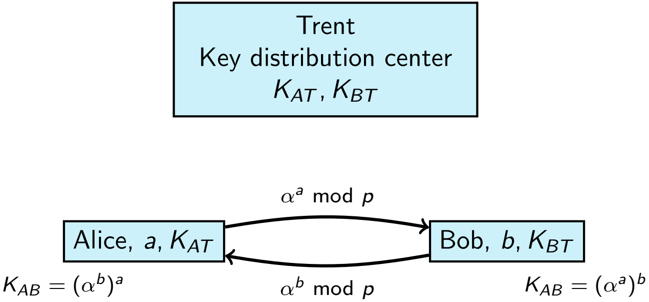
Public key distribution, Diffie-Hellmann

- Diffie-Hellman key exchange is a way to share key
- Alice and Bob create secrets a and b
- They send $\alpha^a \bmod p$ and $\alpha^b \bmod p$ to each other



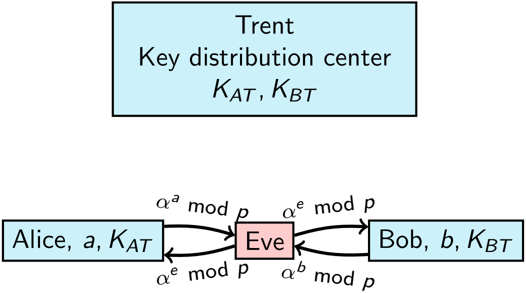
Public key distribution, Diffie-Hellmann

- Diffie-Hellman key exchange is a way to share key
- Alice and Bob create secrets a and b
- They send $\alpha^a \bmod p$ and $\alpha^b \bmod p$ to each other
- Both calculate $K_{AB} = (\alpha^a)^b = (\alpha^b)^a \bmod p$



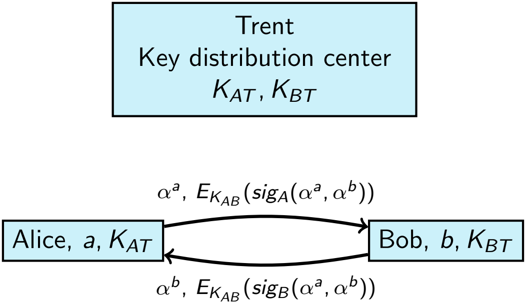
Public key distribution, Diffie-Hellmann

- Diffie-Hellman key exchange is a way to share key
- However, Eve can do an “intruder-in-the-middle”



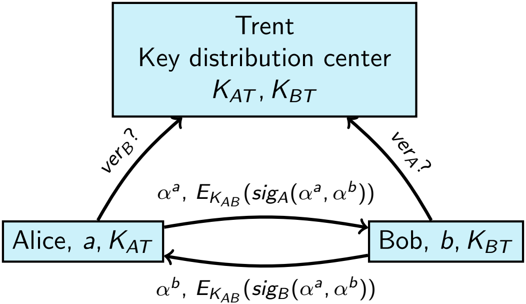
Public key distribution, Station-To-Station (STS) protocol

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can use Trent to verify that they exchange key with the right person



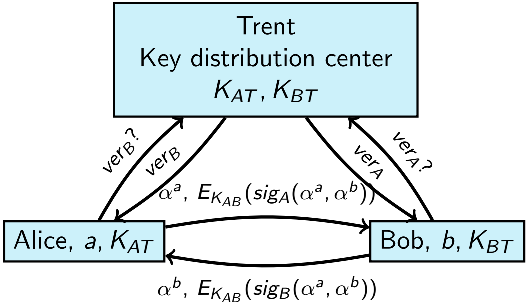
Public key distribution, Station-To-Station (STS) protocol

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can use Trent to verify that they exchange key with the right person



Public key distribution, Station-To-Station (STS) protocol

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can use Trent to verify that they exchange key with the right person



Public key distribution

- Public key distribution uses a Public Key Infrastructure (PKI)

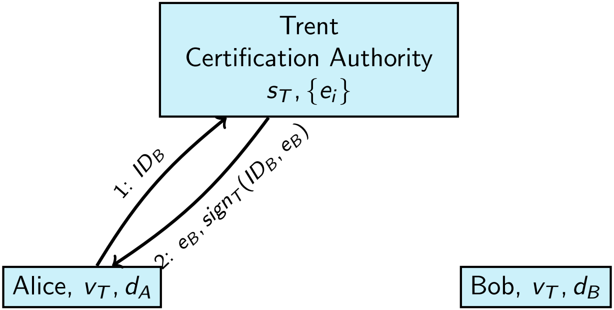
Trent
Certification Authority
$s_T, \{e_i\}$

Alice, v_T, d_A

Bob, v_T, d_B

Public key distribution, using Certification Authorities

- Public key distribution uses a Public Key Infrastructure (PKI)
- Alice sends a request to a Certification Authority (CA) who responds with a certificate, ensuring that Alice uses the correct key to communicate with Bob



Public key distribution, using X.509 certificates

- The CAs often are commercial companies, that are assumed to be trustworthy
- Many arrange to have the root certificate packaged with IE, Mozilla, Opera,. . .
- They issue certificates for a fee
- They often use Registration Authorities (RA) as sub-CA for efficiency reasons
- This creates a “certificate chain”

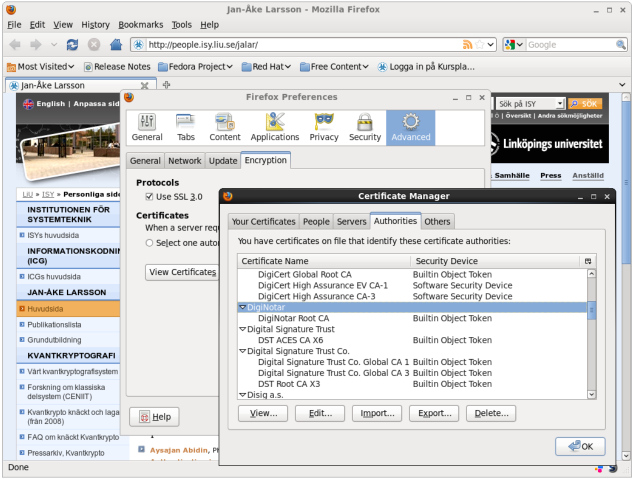
The content of a X.509 certificate

Version (v3)
Serial Number
Algorithm ID
Issuer
Validity Period
Subject Name
Subject Public Key Info (Algorithm, Public Key)
Issuer Unique Identifier (optional)
Subject Unique Identifier (optional)
Extensions (optional)
Certificate Signature Algorithm
Certificate Signature

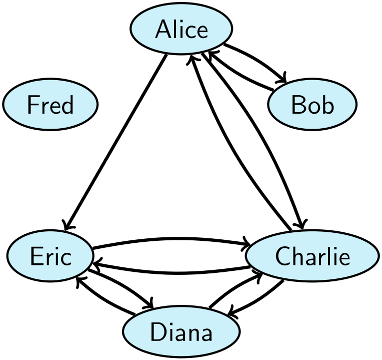
Revocation

- Certificate Revocation Lists distributed at regular intervals is the proposed solution in X.509
- On-line checks are better, but can be expensive
- Short-lived certificates are an alternative, but needs frequent certificate changes
- And the CAs themselves are not the best examples of trustworthy organizations

Public key distribution, X.509 (PKIX) certificates in your browser

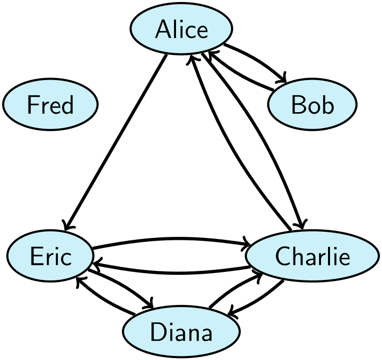


Public key distribution, using web of trust



- No central CA
- Users sign each other's public key (hashes)
- This creates a "web of trust"

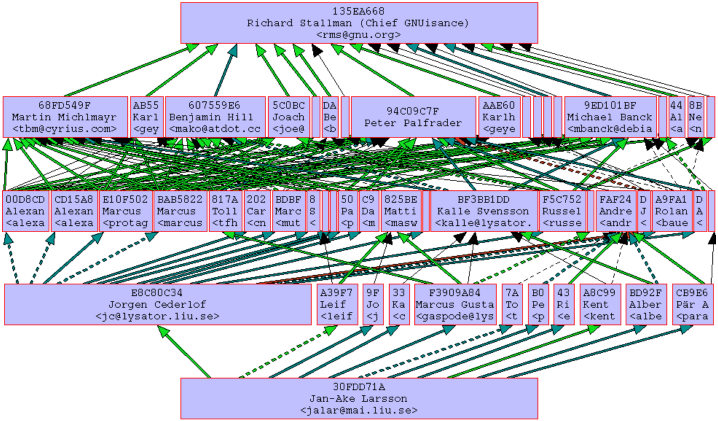
Public key distribution, using web of trust (PGP and GPG)



- No central CA
- Users sign each other's public key (hashes)
- This creates a "web of trust"

- Each user keeps a keyring with the keys (s)he has signed
- The secret key is stored on a secret keyring, on h{er,is} computer
- The public key(s) and their signatures are uploaded to key servers

Public key distribution, a web-of-trust path



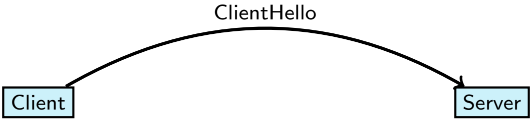
http://www.lysator.liu.se/~jc/wotsap/

Secure Sockets Layer (SSL); Transport Layer Security (TLS)

- This is a client-server handshake procedure to establish key
- The server (but not the client) is authenticated (by its certificate)

Secure Sockets Layer (SSL); Transport Layer Security (TLS)

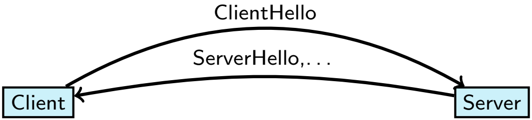
ClientHello: highest TLS protocol version, random number, suggested public key systems + symmetric key systems + hash functions + compression algorithms



Secure Sockets Layer (SSL); Transport Layer Security (TLS)

ClientHello: highest TLS protocol version, random number, suggested public key systems + symmetric key systems + hash functions + compression algorithms

ServerHello, Certificate, ServerHelloDone: chosen protocol version, a (different) random number, system choices, public key

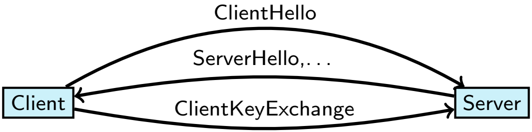


Secure Sockets Layer (SSL); Transport Layer Security (TLS)

ClientHello: highest TLS protocol version, random number, suggested public key systems + symmetric key systems + hash functions + compression algorithms

ServerHello, Certificate, ServerHelloDone: chosen protocol version, a (different) random number, system choices, public key

ClientKeyExchange: PreMasterSecret, encrypted with the server's public key



Secure Sockets Layer (SSL); Transport Layer Security (TLS)

ClientHello: highest TLS protocol version, random number, suggested public key systems + symmetric key systems + hash functions + compression algorithms

ServerHello, Certificate, ServerHelloDone: chosen protocol version, a (different) random number, system choices, public key

ClientKeyExchange: PreMasterSecret, encrypted with the server's public key

(Master secret): creation of master secret using a pseudorandom function, with the PreMasterSecret as seed

(Session keys): session keys are created using the master secret, different keys for the two directions of communication

Secure Sockets Layer (SSL); Transport Layer Security (TLS)

ClientHello: highest TLS protocol version, random number, suggested public key systems + symmetric key systems + hash functions + compression algorithms

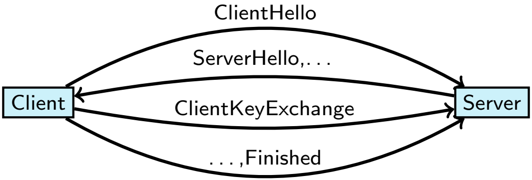
ServerHello, Certificate, ServerHelloDone: chosen protocol version, a (different) random number, system choices, public key

ClientKeyExchange: PreMasterSecret, encrypted with the server's public key

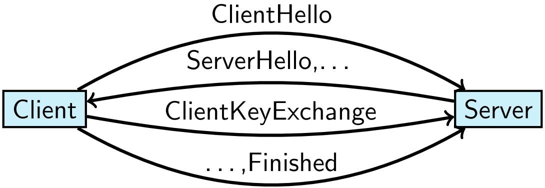
(Master secret): creation of master secret using a pseudorandom function, with the PreMasterSecret as seed

(Session keys): session keys are created using the master secret, different keys for the two directions of communication

ChangeCipherSpec, Finished authenticated and encrypted, containing a MAC for the previous handshake messages

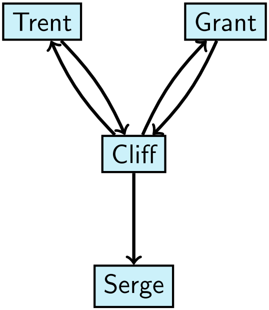


Secure Sockets Layer (SSL) and Transport Layer Security (TLS)



- SSL 1.0 (no public release), 2.0 (1995), 3.0 (1996), originally by Netscape
- TLS 1.0 (1999), TLS 1.1 (2006), TLS 1.2 (2008), and some later changes
- Current problem: TLS 1.0 is fallback if either end does not support higher versions

Key management



The problem is to

- generate
- distribute
- store
- use
- revoke

the key in a secure way