Computer and Network Security (2023-2024)

Part 5: Mutual Trust

Jeroen Famaey
jeroen.famaey@uantwerpen.be

University of Antwerp
Faculty of Science

Symmetric Key Distribution

1. Symmetric private key distribution
2. Key distribution centers (KDCs)
3. Asymmetric private key distribution

University of Antwerp
Faculty of Science

Private key distribution using symmetric encryption

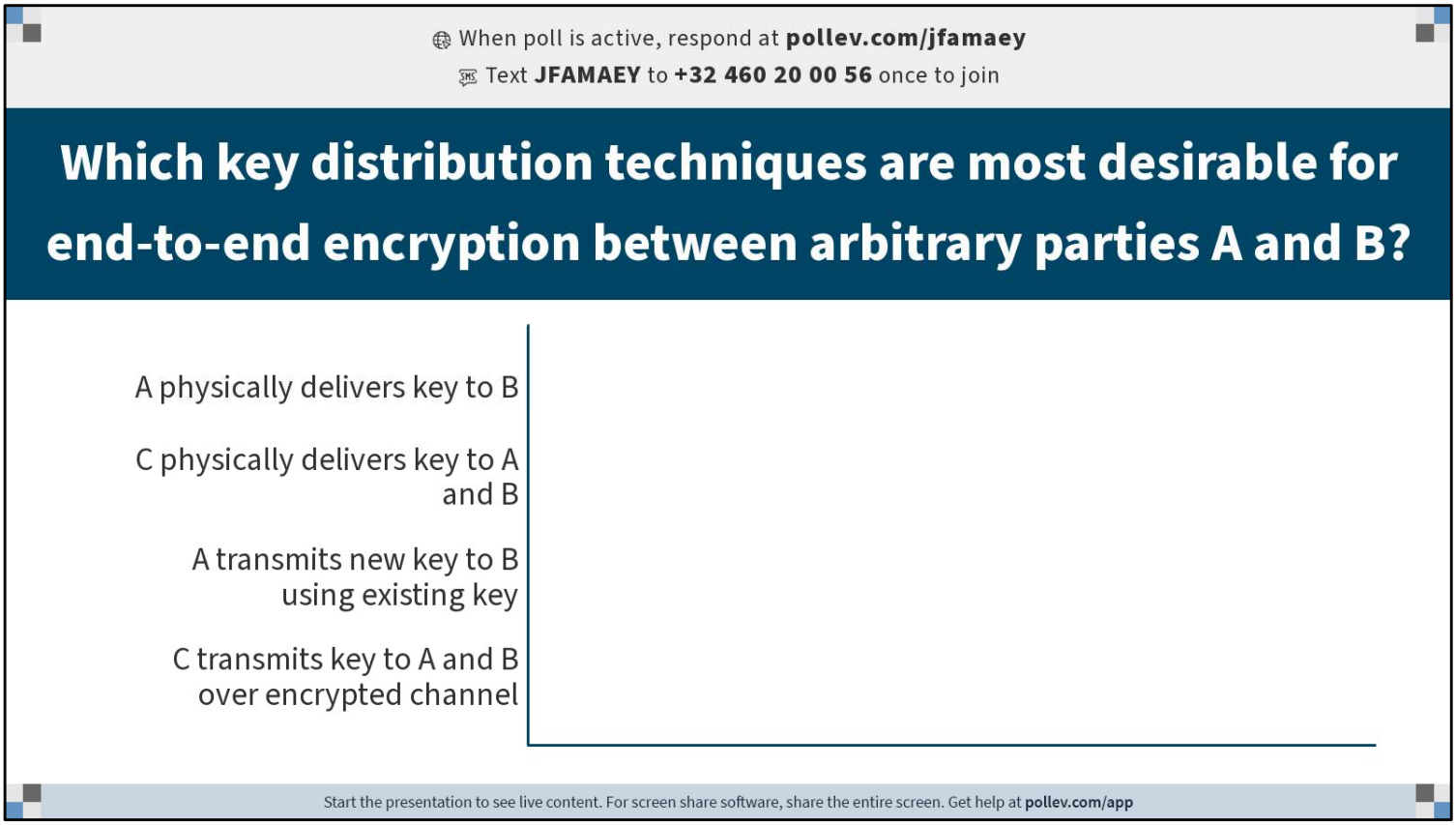
A means to deliver a private encryption key to 2 parties who wish to exchange data without allowing others to see the key

For two parties A and B, different distribution methods exist

- A selects a key and physically delivers it to B
- Third party selects a key and physically delivers it to A and B
- If A and B are already sharing a key, a new key can be encrypted using the old key and transmitted over the network
- If A and B have an encrypted connection to a third-party C, C can deliver a key on the encrypted link to A and B

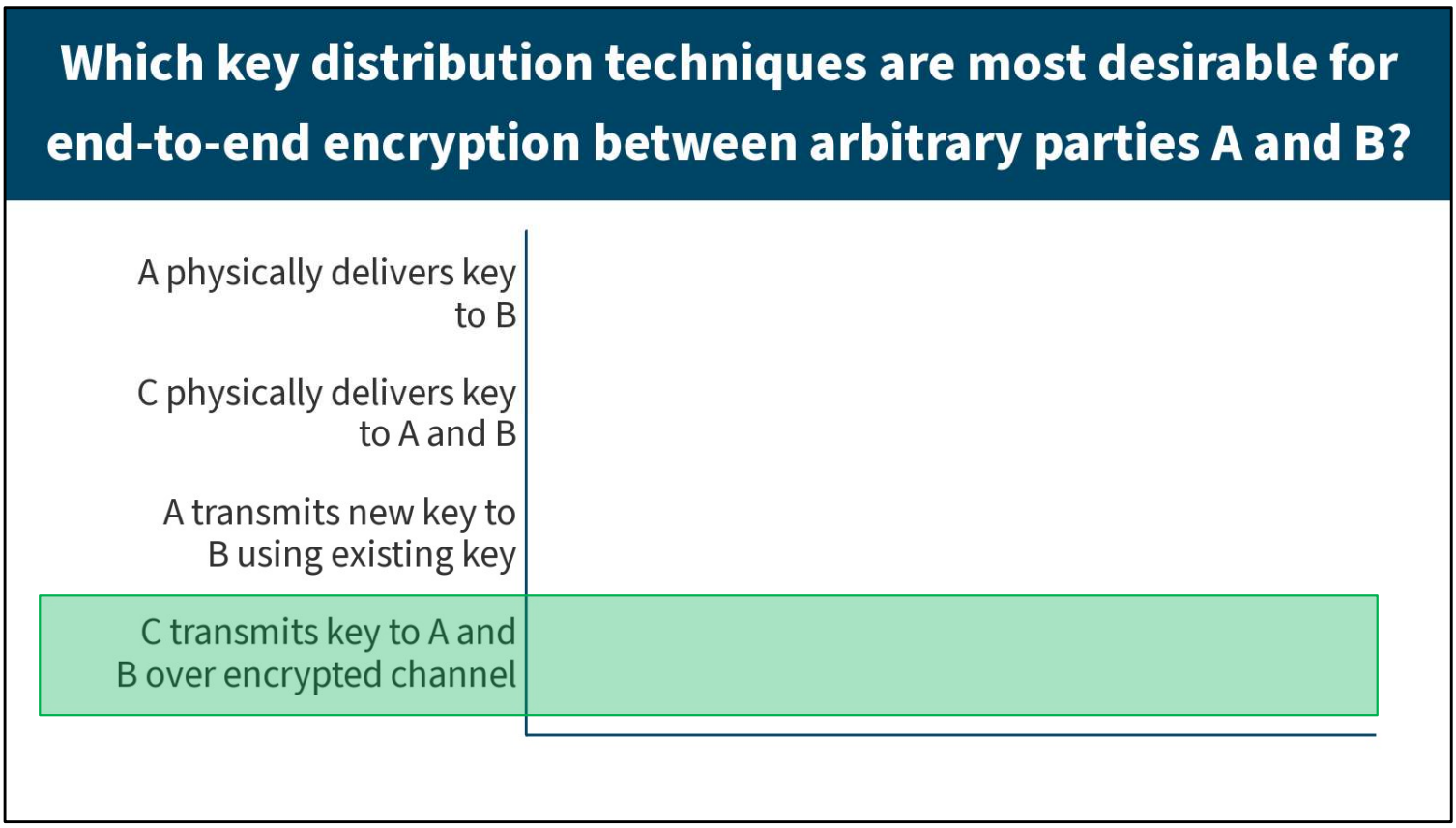
For symmetric encryption to work, the two parties to an exchange must share the same key, and that key must be protected from access by others. Furthermore, frequent key changes are usually desirable to limit the amount of data compromised if an attacker learns the key. Therefore, the strength of any cryptographic system rests with the *key distribution technique*, a term that refers to the means of delivering a key to two parties who wish to exchange data without allowing others to see the key. For two parties A and B, key distribution can be achieved in a number of ways, as follows:

- A can select a key and physically deliver it to B.
- A third party can select the key and physically deliver it to A and B.
- If A and B have previously and recently used a key, one party can transmit the new key to the other, encrypted using the old key.
- If A and B each has an encrypted connection to a third party C, C can deliver a key on the encrypted links to A and B.



Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

Which key distribution techniques are most desirable for end-to-end encryption between arbitrary parties A and B?
https://www.polleverywhere.com/multiple_choice_polls/CPWGlYevz6JXqGc?state=opened&flow=Default&onscreen=persist



Poll Title: Which key distribution techniques are most desirable for end-to-end encryption between arbitrary parties A and B?

Private key distribution using symmetric encryption

A means to deliver a private encryption key to 2 parties who wish to exchange data without allowing others to see the key

For two parties A and B, different distribution methods exist

- A selects a key and physically delivers it to B
- Third party selects a key and physically delivers it to A and B

Out-of-band delivery: infeasible for end-to-end encryption where any pair of hosts may arbitrarily communicate

- If A and B are already sharing a key, a new key can be encrypted using the old key and transmitted over the network
- If A and B have an encrypted connection to a third-party C, C can deliver a key on the encrypted link to A and B

Options 1 and 2 call for manual delivery of a key. For link encryption, this is a reasonable requirement, because each link encryption device is going to be exchanging data only with its partner on the other end of the link. However, for **end-to-end encryption** over a network, manual delivery is awkward. In a distributed system, any given host or terminal may need to engage in exchanges with many other hosts and terminals over time. Thus, each device needs a number of keys supplied dynamically. The problem is especially difficult in a wide-area distributed system.

Private key distribution using symmetric encryption

A means to deliver a private encryption key to 2 parties who wish to exchange data without allowing others to see the key

For two parties A and B, different distribution methods exist

- A selects a key and physically delivers it to B
- Third party selects a key and physically delivers it to A and B
- If A and B are already sharing a key, a new key can be encrypted using the old key and transmitted over the network
- If A and B have an encrypted connection to a third-party C, C can deliver a key on the encrypted link to A and B

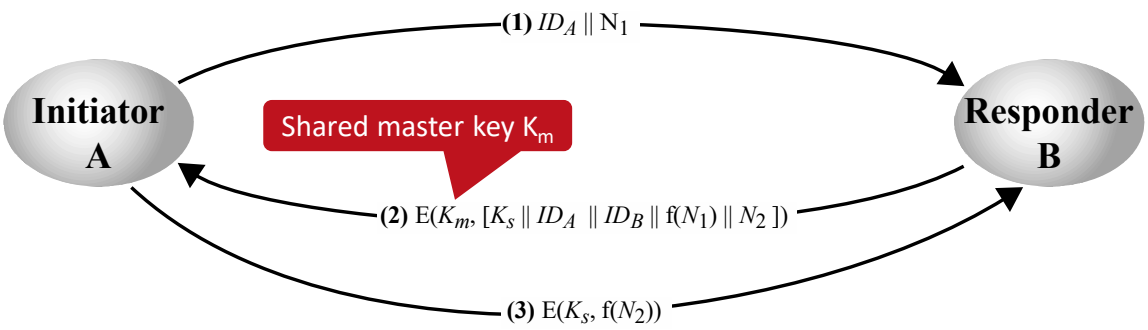
If attacker determines a single key, then all future keys are compromised

Option 3 is a possibility for either link encryption or end-to-end encryption, but if an attacker ever succeeds in gaining access to one key, then all subsequent keys will be revealed. Furthermore, the initial distribution of potentially millions of keys still must be made.

For end-to-end encryption, some variation on option 4 has been widely adopted. In this scheme, a key distribution center is responsible for distributing keys to pairs of users (hosts, processes, applications) as needed. Each user must share a unique key with the key distribution center for purposes of key distribution.

Decentralized key control without third party

- Avoids the need for a central trusted authority (i.e., the Key Distribution Center)
- Only useful is small scale scenarios (e.g., local networks)

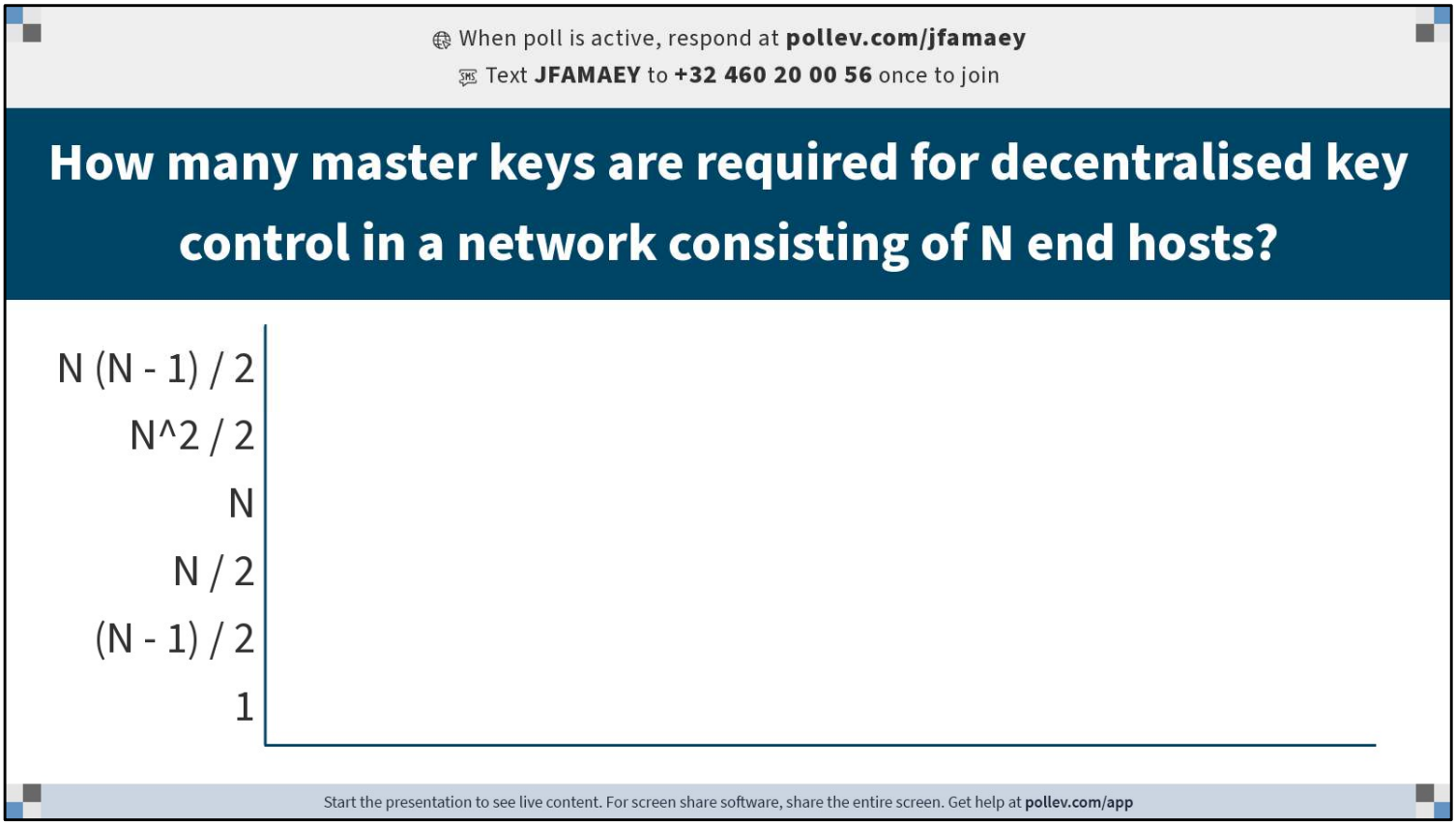


The use of a key distribution center imposes the requirement that the KDC be trusted and be protected from subversion. This requirement can be avoided if key distribution is fully decentralized. Although full decentralization is not practical for larger networks using symmetric encryption only, it may be useful within a local context.

A decentralized approach requires that each end system be able to communicate in a secure manner with all potential partner end systems for purposes of session key distribution. Thus, there may need to be as many as [n (n-1)]/2 master keys for a configuration with n end systems. A session key may be established with the following sequence of steps:

1. A issues a request to B for a session key and includes a nonce, N_1.
2. B responds with a message that is encrypted using the shared master key. The response includes the session key selected by B, an identifier of B, the value f(N_1), and another nonce, N_2.
3. Using the new session key, A returns f(N_2) to B.

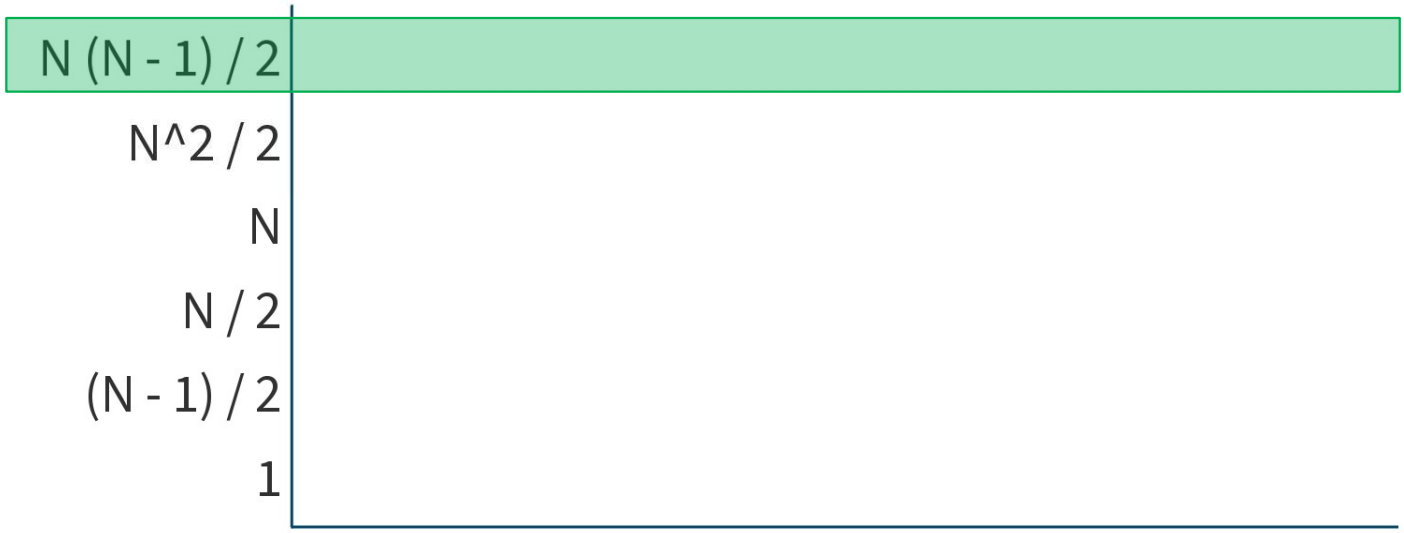
Thus, although each node must maintain at most (n - 1) master keys, as many session keys as required may be generated and used. Because the messages transferred using the master key are short, cryptanalysis is difficult. As before, session keys are used for only a limited time to protect them.



Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

How many master keys are required for decentralised key control in a network consisting of N end hosts?
https://www.polleverywhere.com/multiple_choice_polls/kBH5XKayfgyPHE4?state=opened&flow=Default&onscreen=persist

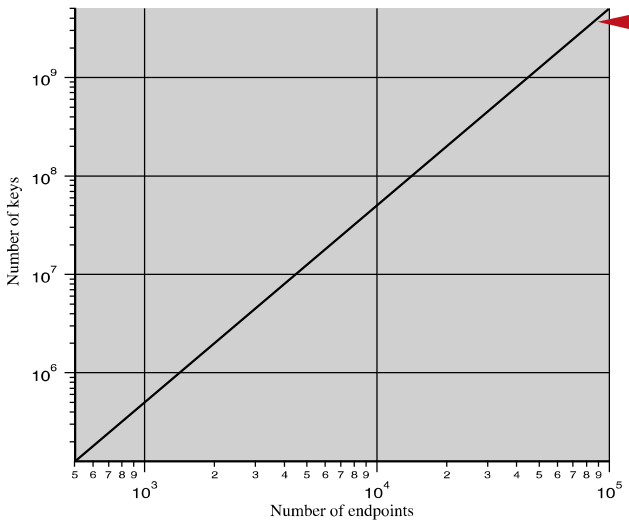
How many master keys are required for decentralised key control in a network consisting of N end hosts?



Poll Title: How many master keys are required for decentralised key control in a network consisting of N end hosts?

The scalability problem of end-to-end encryption

A key is needed for any pair of communicating hosts or applications



For N endpoints, $[N(N-1)]/2$ keys are needed

University of Antwerp
Faculty of Science

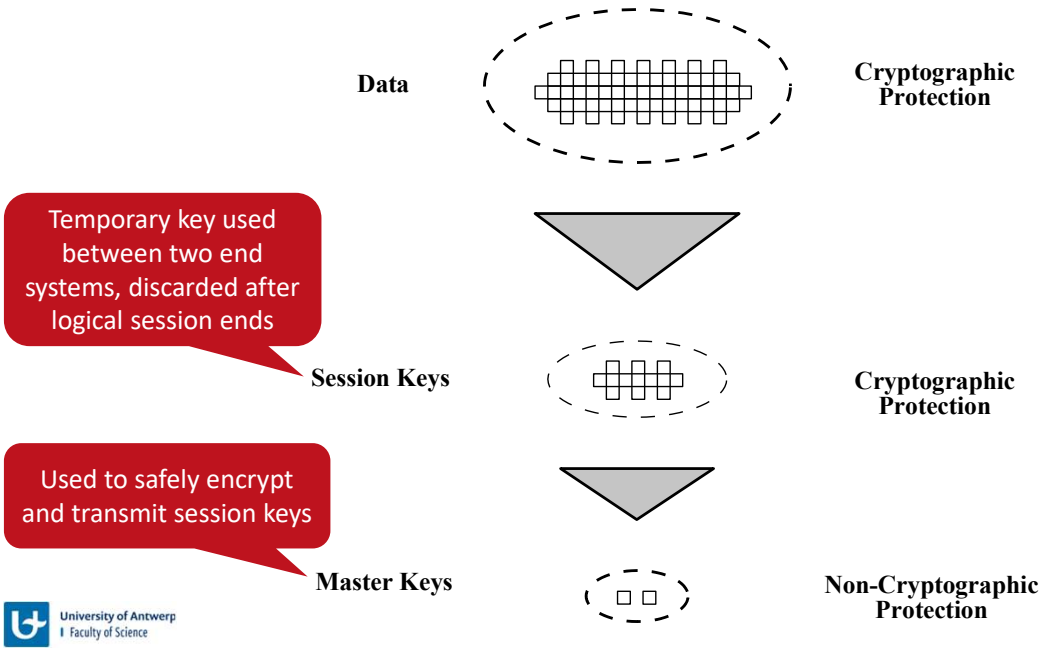
12

The scale of the problem depends on the number of communicating pairs that must be supported. If end-to-end encryption is done at a network or IP level, then a key is needed for each pair of hosts on the network that wish to communicate. Thus, if there are *N* hosts, the number of required keys is [*N*(*N* - 1)]/2. If encryption is done at the application level, then a key is needed for every pair of users or processes that require communication. Thus, a network may have hundreds of hosts but thousands of users and processes. The graph illustrates the magnitude of the key distribution task for end-to-end encryption. A network using node-level encryption with 1000 nodes would conceivably need to distribute as many as half a million keys. If that same network supported 10,000 applications, then as many as 50 million keys may be required for application-level encryption.

1. Symmetric private key distribution
2. Key distribution centers (KDCs)
3. Asymmetric private key distribution

University of Antwerp
Faculty of Science

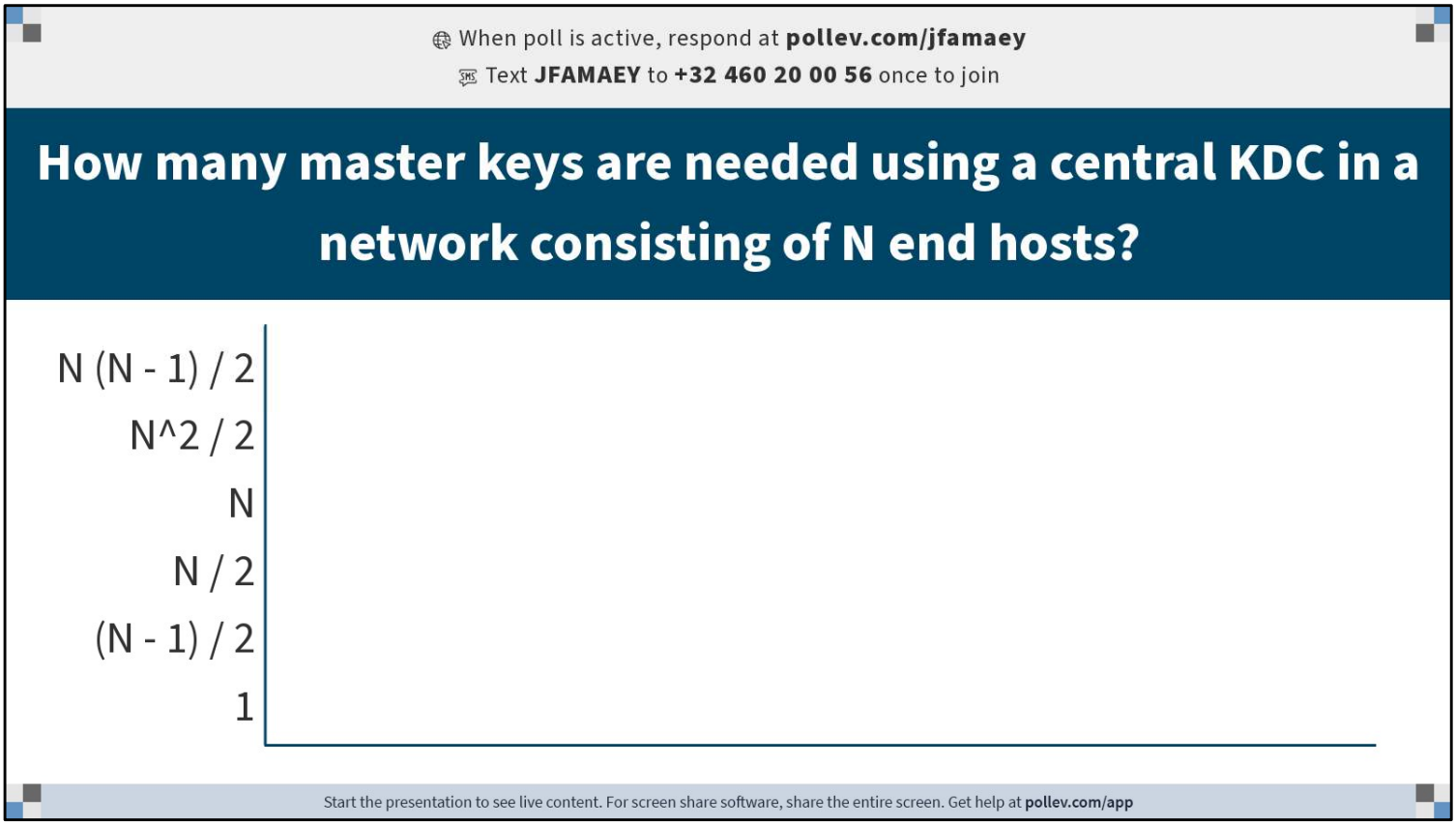
Hierarchical third-party key distribution centers



University of Antwerp
Faculty of Science

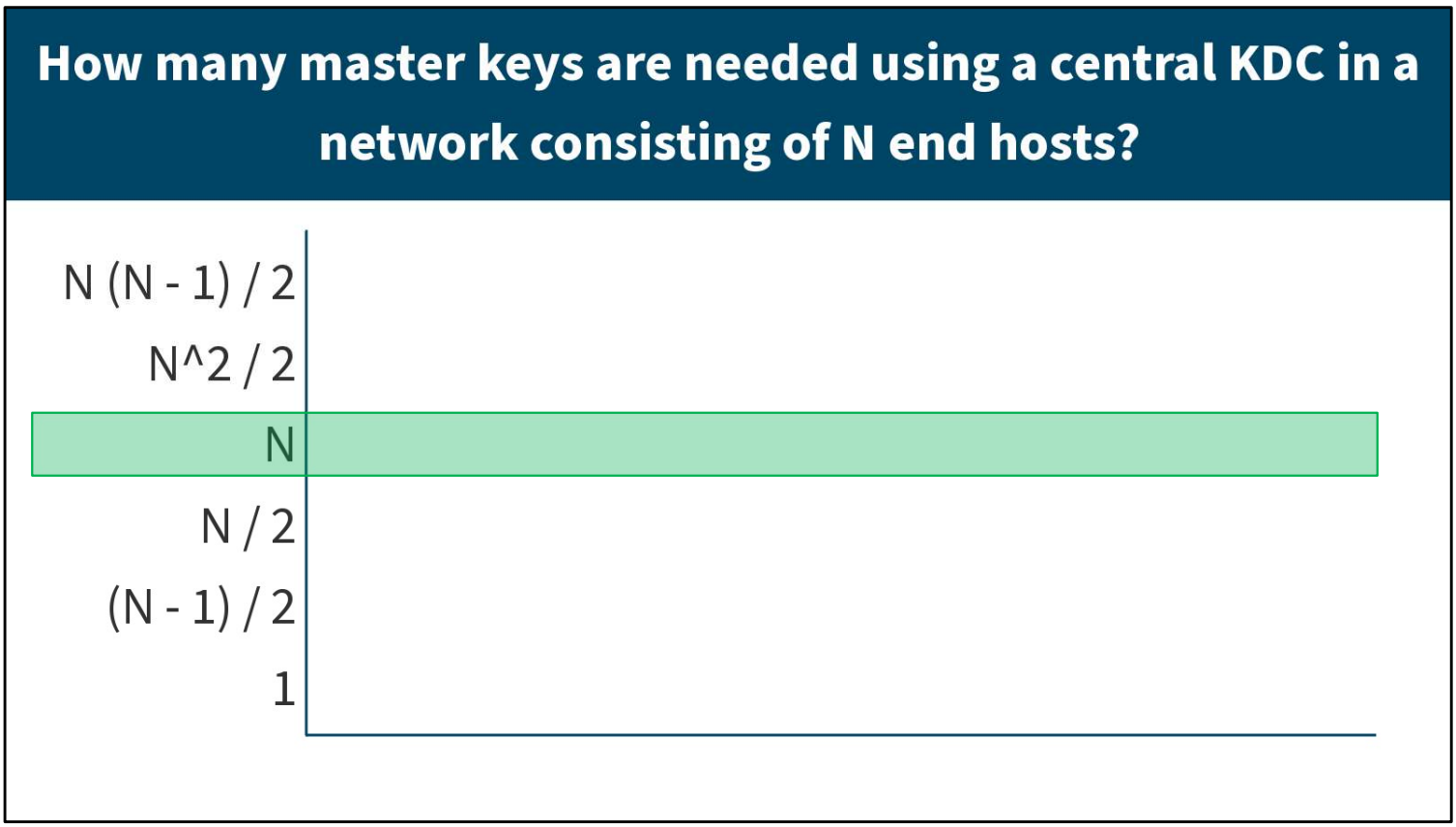
14

The use of a key distribution center is based on the use of a hierarchy of keys. At a minimum, two levels of keys are used. Communication between end systems is encrypted using a temporary key, often referred to as a **session key**. Typically, the session key is used for the duration of a logical connection, such as a frame relay connection or transport connection, and then discarded. Each session key is obtained from the key distribution center over the same networking facilities used for end-user communication. Accordingly, session keys are transmitted in encrypted form, using a **master key** that is shared by the key distribution center and an end system or user.



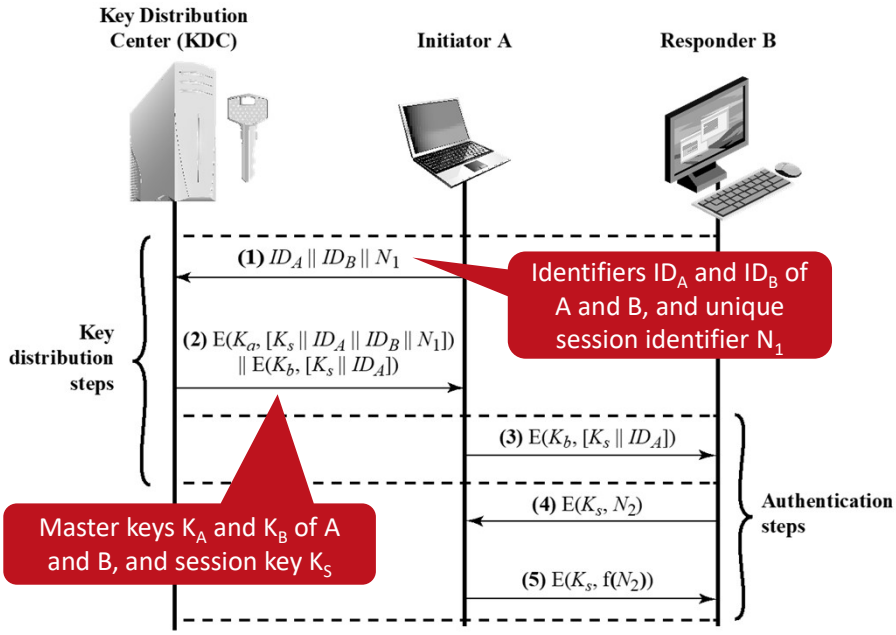
Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

How many master keys are needed using a central KDC in a network consisting of N end hosts?
https://www.polleverywhere.com/multiple_choice_polls/VMr8qjvQzYU6PKJ?state=opened&flow=Default&onscreen=persist



Poll Title: How many master keys are needed using a central KDC in a network consisting of N end hosts?

A Key Distribution Center (KDC) scenario



University of Antwerp | Faculty of Science

17

For each end system or user, there is a unique master key that it shares with the key distribution center. Of course, these master keys must be securely distributed in some fashion. However, the scale of the problem is vastly reduced. If there are *N* entities that wish to communicate in pairs, then, as was mentioned, as many as [*N*(*N* - 1)]/2 session keys are needed at any one time. However, only *N* master keys are required, one for each entity. Thus, master keys can be distributed in some non-cryptographic way, such as physical delivery.

The key distribution concept can be deployed in a number of ways. A typical scenario is illustrated in the figure. The scenario assumes that each user shares a unique master key with the key distribution center (KDC). Let us assume that user A wishes to establish a logical connection with B and requires a one-time session key to protect the data transmitted over the connection. A has a master key, *Ka*, known only to itself and the KDC; similarly, B shares the master key K_b with the KDC. The following steps occur:

1. A issues a request to the KDC for a session key to protect a logical connection to B. The message includes the identity of A and B and a unique identifier, N_1, for this transaction, which we refer to as a nonce. The nonce may be a timestamp, a counter, or a random number; the minimum requirement is that it differs with each request. Also, to prevent masquerade, it should be difficult for an opponent to guess the nonce. Thus, a random number is a good choice for a nonce.
2. The KDC responds with a message encrypted using K_a. Thus, A is the only one who can successfully read the message, and A knows that it originated at the KDC. The message includes two items intended for A:
    - The one-time session key, K_s, to be used for the session

- The original request message, including the nonce, to enable A to match this response with the appropriate request

Thus, A can verify that its original request was not altered before reception by the KDC and, because of the nonce, that this is not a replay of some previous request. In addition, the message includes two items intended for B:

- The one-time session key, K_s, to be used for the session
- An identifier of A (e.g., its network address), ID_A

These last two items are encrypted with K_b (the master key that the KDC shares with B). They are to be sent to B to establish the connection and prove A's identity.

1. A stores the session key for use in the upcoming session and forwards to B the information that originated at the KDC for B, namely, $E(K_b,[K_s \,||\, ID_A])$. Because this information is encrypted with K_b, it is protected from eavesdropping. B now knows the session key (K_s), knows that the other party is A (from ID_A), and knows that the information originated at the KDC (because it is encrypted using K_b).

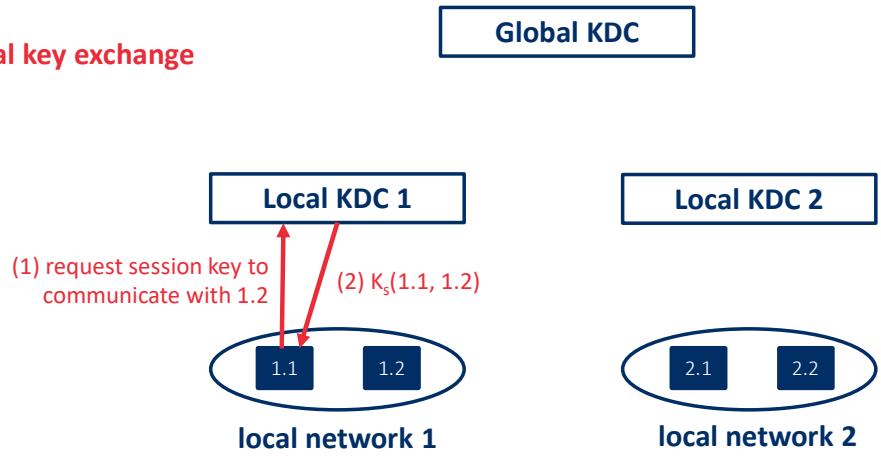
At this point, a session key has been securely delivered to A and B, and they may begin their protected exchange. However, two additional steps are desirable:

4. Using the newly minted session key for encryption, B sends a nonce, N_2, to A.
5. Also, using K_s, A responds with f(N_2), where f is a function that performs some transformation on N_2 (e.g., adding one).

These steps assure B that the original message it received (step 3) was not a replay. Note that the actual key distribution involves only steps 1 through 3, but that steps 4 and 5, as well as step 3, perform an authentication function.

Hierarchical key control

- A hierarchy of KDCs manages different parts of a network
- A local KDC distributes keys within the local network (e.g., a LAN)
- Higher layer KDCs can coordinate key distribution across networks
- **Case 1: Local key exchange**



University of Antwerp | Faculty of Science

18

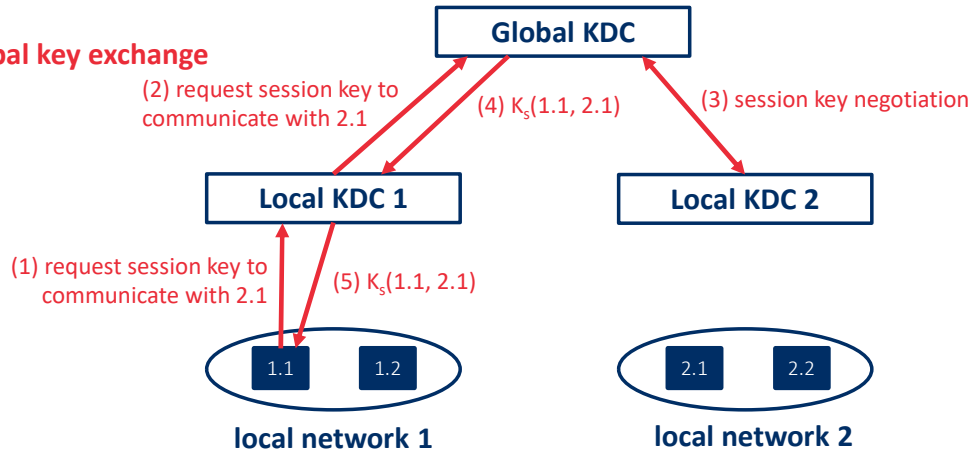
It is not necessary to limit the key distribution function to a single KDC. Indeed, for very large networks, it may not be practical to do so. As an alternative, a hierarchy of KDCs can be established. For example, there can be local KDCs, each responsible for a small domain of the overall internetwork, such as a single LAN or a single building. For communication among entities within the same local domain, the local KDC is responsible for key distribution. If two entities in different domains desire a shared key, then the corresponding local KDCs can communicate through a global KDC. In this case, any one of the three KDCs involved can actually select the key. The hierarchical concept can be extended to three or even more layers, depending on the size of the user population and the geographic scope of the internetwork.

A hierarchical scheme minimizes the effort involved in master key distribution, because most master keys are those shared by a local KDC with its local entities. Furthermore, such a scheme limits the damage of a faulty or subverted KDC to its local area only.

Hierarchical key control

- A hierarchy of KDCs manages different parts of a network
- A local KDC distributes keys within the local network (e.g., a LAN)
- Higher layer KDCs can coordinate key distribution across networks

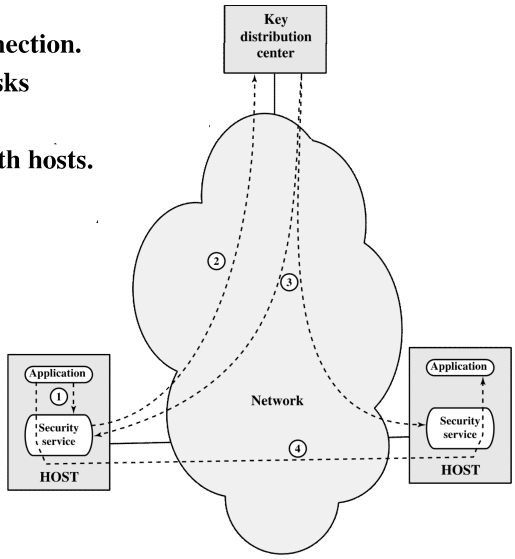
- **Case 2: Global key exchange**



Transparent key control scheme

For providing end-to-end encryption in connection-oriented protocols

1. Host sends packet requesting connection.
2. Security service buffers packet; asks KDC for session key.
3. KDC distributes session key to both hosts.
4. Buffered packet transmitted.



University of Antwerp | Faculty of Science

20

The approach suggested before has many variations, one of which is described on this slide. The scheme is useful for providing end-to-end encryption at a network or transport level in a way that is transparent to the end users. The approach assumes that communication makes use of a connection-oriented end-to-end protocol, such as TCP. The noteworthy element of this approach is a session security module (SSM), which may consist of functionality at one protocol layer, that performs end-to-end encryption and obtains session keys on behalf of its host or terminal.

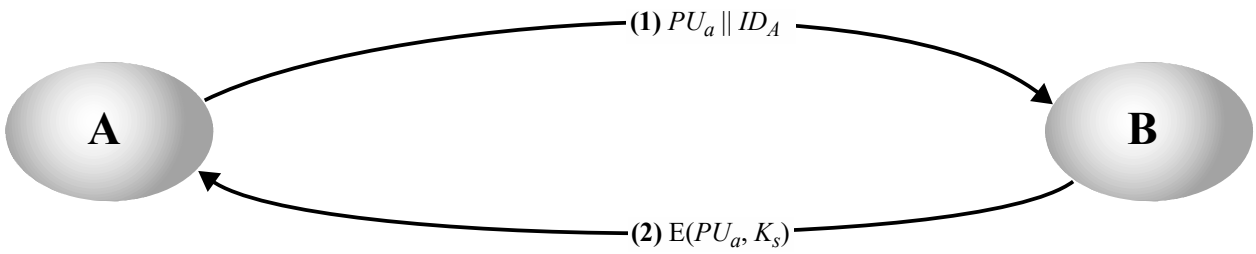
The steps involved in establishing a connection are shown in the figure. When one host wishes to set up a connection to another host, it transmits a connection-request packet (step 1). The SSM saves that packet and applies to the KDC for permission to establish the connection (step 2). The communication between the SSM and the KDC is encrypted using a master key shared only by this SSM and the KDC. If the KDC approves the connection request, it generates the session key and delivers it to the two appropriate SSMs, using a unique permanent key for each SSM (step 3). The requesting SSM can now release the connection request packet, and a connection is set up between the two end systems (step 4). All user data exchanged between the two end systems are encrypted by their respective SSMs using the one-time session key.

The automated key distribution approach provides the flexibility and dynamic characteristics needed to allow a number of terminal users to access a number of hosts and for the hosts to exchange data with each other.

1. Symmetric private key distribution
2. Key distribution centers (KDCs)
3. Asymmetric private key distribution

University of Antwerp
Faculty of Science

Symmetric key distribution using asymmetric encryption



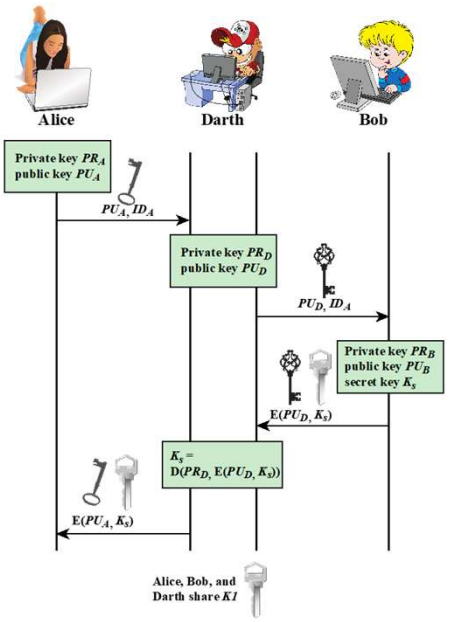
University of Antwerp
Faculty of Science

An extremely simple scheme was put forward by Merkle, as illustrated in the figure. If A wishes to communicate with B, the following procedure is employed:

1. A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message to B, consisting of PU_a and an identifier of A, ID_A.
2. B generates a secret key, K_s, and transmits it to A, which is encrypted with A's public key.
3. A computes $D(PR_a, E(PU_a, K_s))$ to recover the secret key. Because only A can decrypt the message, only A and B will know the identity of K_s.
4. A discards PU_a and PR_a and B discards PU_a.

A and B can now securely communicate using conventional encryption and the session key K_s. At the completion of the exchange, both A and B discard K_s. Despite its simplicity, this is an attractive protocol. No keys exist before the start of the communication and none exist after the completion of communication. Thus, the risk of compromise of the keys is minimal. At the same time, the communication is secure from eavesdropping.

Man-in-the-middle attack



University of Antwerp
Faculty of Science

23

The protocol depicted in on the previous slide is insecure against an adversary who can intercept messages and then either relay the intercepted message or substitute another. Such an attack is known as a **man-in-the-middle attack**. In the present case, if an adversary, D, has control of the intervening communication channel, then D can compromise the communication in the following fashion without being detected:

1. A generates a public/private key pair {PU_a, PR_a} and transmits a message intended for B consisting of PU_a and an identifier of A, ID_A.
2. D intercepts the message, creates its own public/private key pair {PU_d, PR_d} and transmits PU_d || ID_A to B.
3. B generates a secret key, K_s, and transmits E(PU_d, K_s).
4. D intercepts the message and learns K_s by computing D(PR_d, E(PU_d, K_s)).
5. D transmits E(PU_a, K_s) to A.

The result is that both A and B know K_s and are unaware that K_s has also been revealed to D. A and B can now exchange messages using K_s. D no longer actively interferes with the communications channel but simply eavesdrops. Knowing K_s, D can decrypt all messages, and both A and B are unaware of the problem. Thus, this simple protocol is only useful in an environment where the only threat is eavesdropping.

Diffie-Hellman key exchange

- The first published public-key algorithm
- Invented by Diffie and Hellman in 1976
- Enables two users to securely exchange a key (e.g., for use in subsequent symmetric encryption
- It is based on the difficulty to calculate discrete logarithms:
  - The exponent i is the **discrete logarithm** of b for base a mod p
  - $\text{dlog}_{a,p}(b) = i \Leftrightarrow b \equiv a^i (\text{mod } p)$

- The parties share the publically known pair: (q, α)
  - q is a prime number
  - α is an integer and a **primitive root** of q, meaning a unique exponent i exists for every integer b, such that $b \equiv \alpha^i \pmod q$
  - Private keys: a random integers $X_A < q$ and $X_B < q$
  - Public keys: $Y_A = \alpha^{X_A} \bmod q$ and $Y_B = \alpha^{X_B} \bmod q$
  - Shared secret key $K = (Y_B)^{X_A} \bmod q = (Y_A)^{X_B} \bmod q$ can be calculated

The first published public-key algorithm appeared in the seminal paper by Diffie and Hellman that defined public-key cryptography and is generally referred to as Diffie–Hellman key exchange. A number of commercial products employ this key exchange technique.

The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent symmetric encryption of messages. The algorithm itself is limited to the exchange of secret values.

The Diffie–Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms. Briefly, we can define the discrete logarithm in the following way. A **primitive root** of a prime number p is one whose powers modulo p generate all the integers from 1 to p - 1. That is, if a is a primitive root of the prime number p, then the numbers

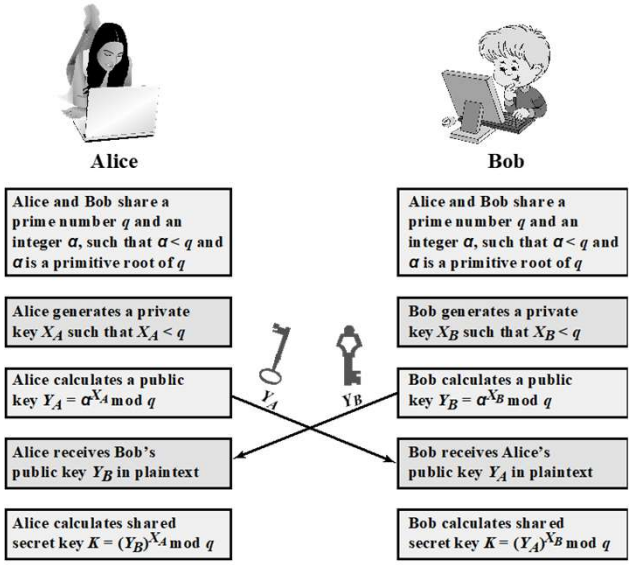
$a \bmod p, a^2 \bmod p, \ldots, a^{p-1} \bmod p$

are distinct and consist of the integers from 1 through p - 1 in some permutation. For any integer b and a primitive root a of prime number p, we can find a unique exponent i such that:

$b \equiv a^i \pmod p$ where $0 <= i <= (p - 1)$

The exponent i is referred to as the **discrete logarithm** of b for the base a, mod p. We express this value as $\text{dlog}_{a,p}(b)$.

Diffie-Hellman key exchange security



- The information that is available to an attacker includes: q, α, Y_A and Y_B
- The private key of user B can be calculated as:
$$X_B = \text{dlog}_{\alpha,q}(Y_B)$$
- The security of Diffie-Hellman lies in the fact that:
  - It is easy to calculate exponentials modulo a prime
  - It is difficult to calculate discrete logarithms

University of Antwerp | Faculty of Science

25

The figure summarizes the Diffie–Hellman key exchange algorithm. For this scheme, there are two publicly known numbers: a prime number q and an integer a that is a primitive root of q. Suppose the users A and B wish to create a shared key.

User A selects a random integer $X_A < q$ and computes $Y_A = a^{XA} \bmod q$. Similarly, user B independently selects a random integer $X_B < q$ and computes $Y_B = a^{XB} \bmod q$. Each side keeps the X value private and makes the Y value available publicly to the other side. Thus, X_A is A's private key and Y_A is A's corresponding public key, and similarly for B. User A computes the key as $K = (Y_B)^{XA} \bmod q$ and user B computes the key as $K = (Y_A)^{XB} \bmod q$. These two calculations produce identical results:

$$\begin{aligned} K &= (Y_B)^{XA} \bmod q \\ &= (a^{XB} \bmod q)^{XA} \bmod q = (a^{XB})^{XA} \bmod q \\ &= a^{XB\,XA} \bmod q \\ &= (a^{XA})^{XB} \bmod q \\ &= (a^{XA} \bmod q)^{XB} \bmod q \\ &= (Y_A)^{XB} \bmod q \end{aligned}$$

The result is that the two sides have exchanged a secret value. Typically, this secret value is used as shared symmetric secret key. Now consider an adversary who can observe the key exchange and wishes to determine the secret key K. Because X_A and X_B are private, an adversary only has the following ingredients to work with: q, a, Y_A, and Y_B. Thus, the adversary is forced to take a discrete logarithm to determine the key. For example, to determine the private key of user B, an adversary must compute

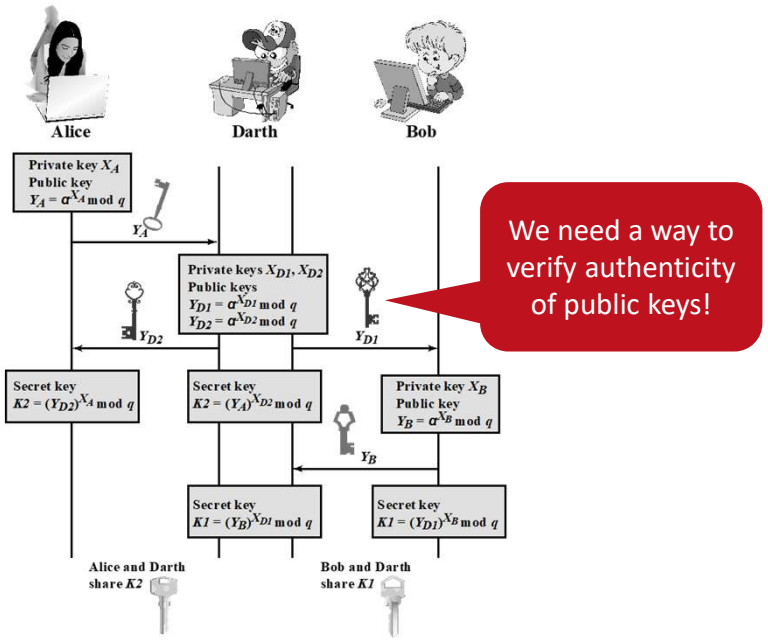
$X_B = \text{dlog}_{a,q}(Y_B)$

The adversary can then calculate the key K in the same manner as user B calculates it. That is, the adversary can calculate K as:

$K = (Y_A)^{XB} \bmod q$

The security of the Diffie–Hellman key exchange lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms. For large primes, the latter task is considered infeasible.

Diffie-Hellman man-in-the-middle attack



University of Antwerp
Faculty of Science

26

The Diffie-Hellman key exchange protocol previously outlined is insecure against a man-in-the-middle attack. Suppose Alice and Bob wish to exchange keys, and Darth is the adversary. The attack proceeds as follows:

1. Darth prepares for the attack by generating two random private keys X_{D1} and X_{D2} and then computing the corresponding public keys Y_{D1} and Y_{D2}.
2. Alice transmits Y_A to Bob.
3. Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates $K_2 = (Y_A)^{XD2} \bmod q$.
4. Bob receives Y_{D1} and calculates $K_1 = (Y_{D1})^{XB} \bmod q$.
5. Bob transmits Y_B to Alice.
6. Darth intercepts Y_B and transmits Y_{D2} to Alice. Darth calculates $K_1 = (Y_B)^{XD1} \bmod q$.
7. Alice receives Y_{D2} and calculates $K_2 = (Y_{D2})^{XA} \bmod q$.

At this point, Bob and Alice think that they share a secret key, but instead Bob and Darth share secret key K_1 and Alice and Darth share secret key K_2. All future communication between Bob and Alice is compromised in the following way:

1. Alice sends an encrypted message M: $E(K_2, M)$.
2. Darth intercepts the encrypted message and decrypts it to recover M.
3. Darth sends Bob $E(K_1, M)$ or $E(K_1, M_2)$, where M_2 is any message. In the first case, Darth simply wants to eavesdrop on the communication without altering it. In the second case, Darth wants to modify the message going to Bob.

The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants. This vulnerability can be overcome with the use of digital signatures and public-key certificates.

Public Key Distribution

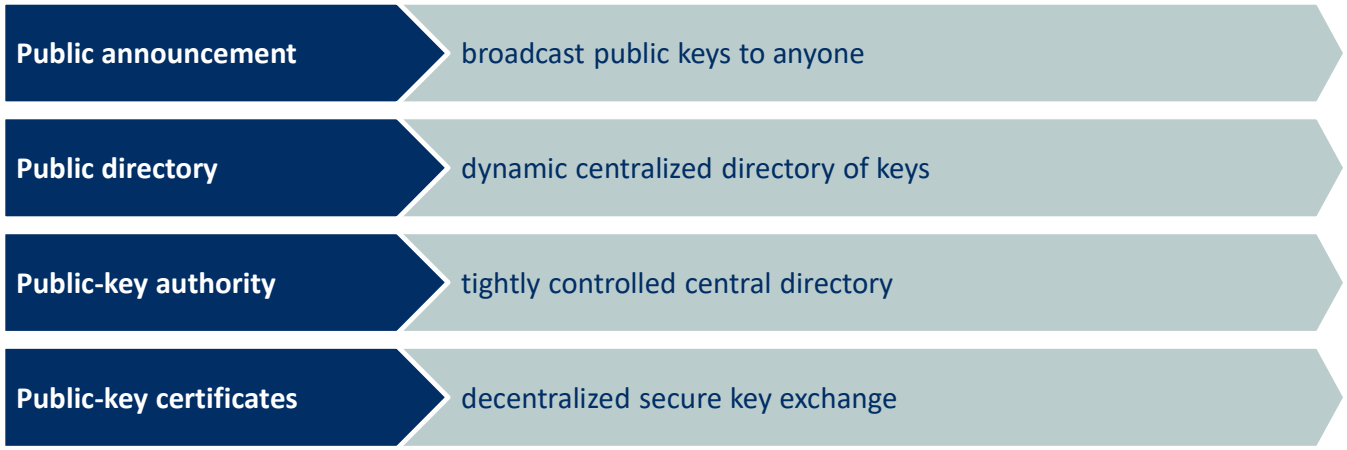
1 Public key distribution methods

2 Public key certificates

University of Antwerp
Faculty of Science

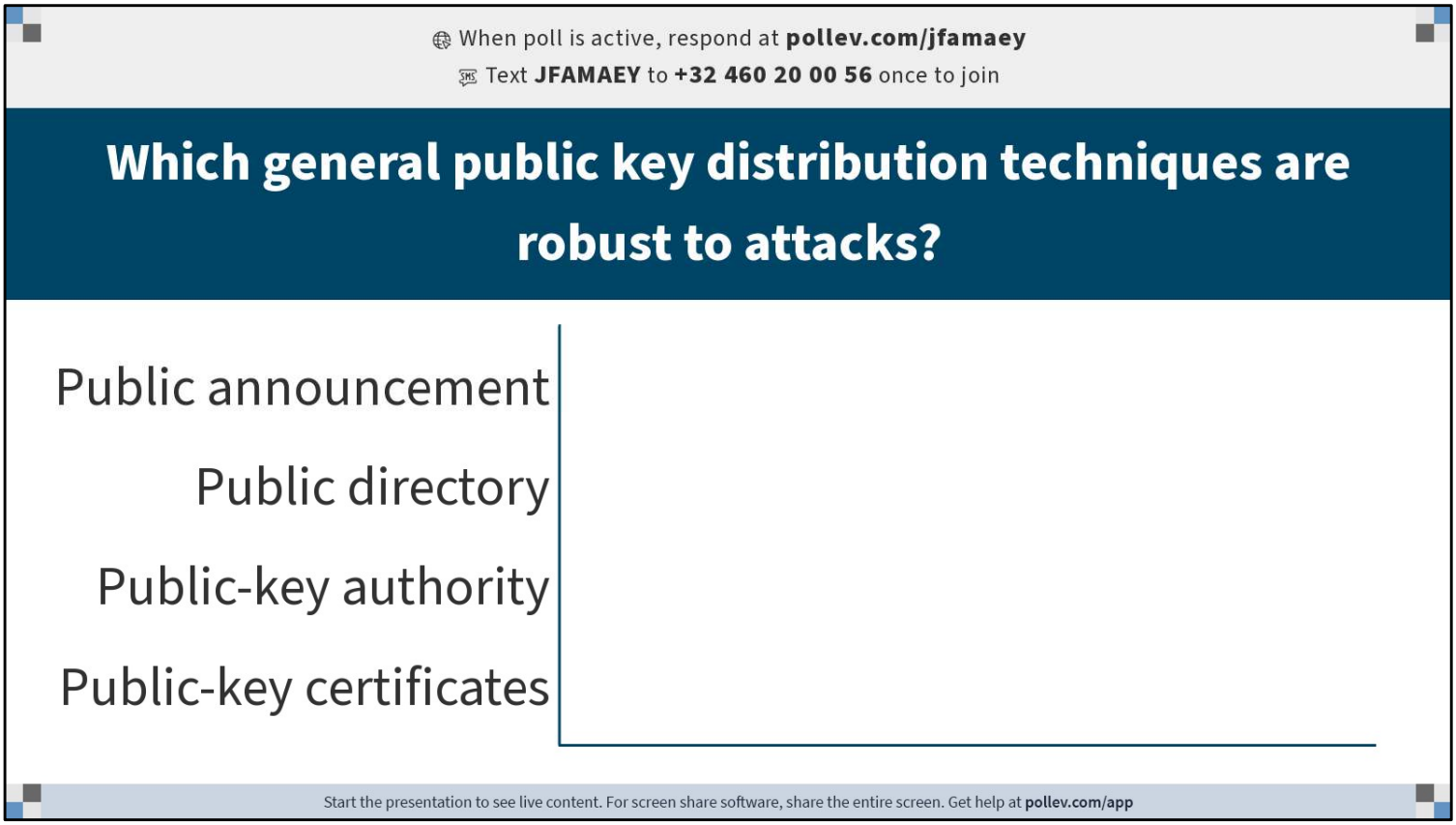
Four general public key distribution techniques

- Or how to securely exchange public keys between communicating parties?
- Public keys can later be used for secret session key distribution



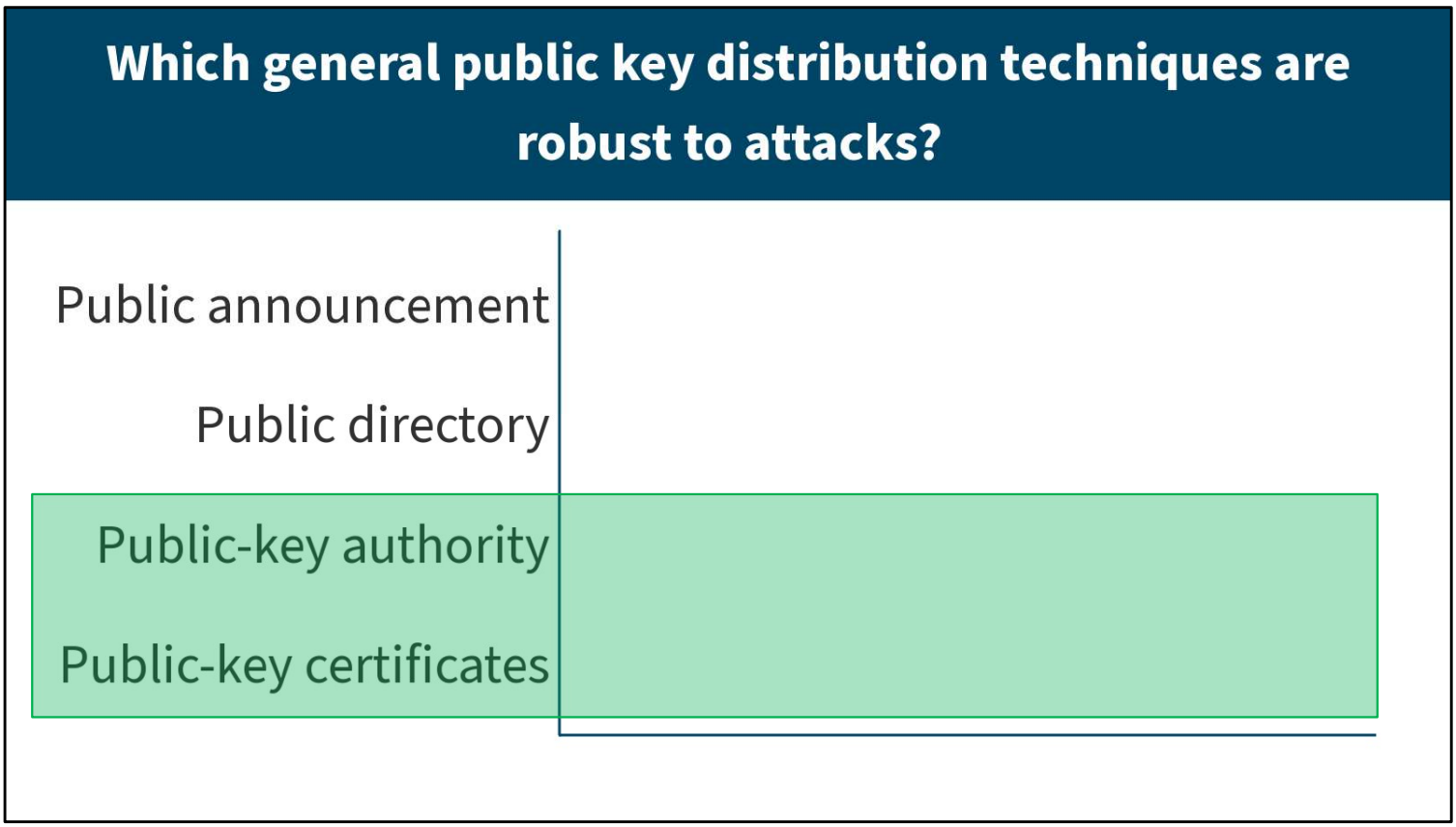
Several techniques have been proposed for the distribution of public keys. Virtually all these proposals can be grouped into the following general schemes:

- Public announcement
- Publicly available directory
- Public-key authority
- Public-key certificates



Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

Which general public key distribution techniques are robust to attacks?
https://www.polleverywhere.com/multiple_choice_polls/7LCNNm0xfDL79is?state=opened&flow=Default&onscreen=persist



Poll Title: Which general public key distribution techniques are robust to attacks?

Public announcement

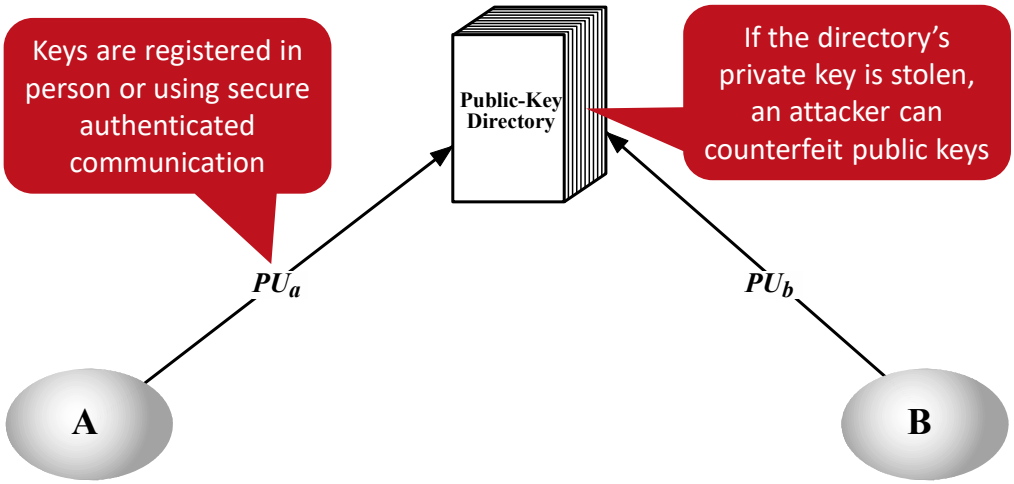
University of Antwerp | Faculty of Science

32

On the face of it, the point of public-key encryption is that the public key is public. Thus, if there is some broadly accepted public-key algorithm, such as RSA, any participant can send his or her public key to any other participant or broadcast the key to the community at large. For example, because of the growing popularity of PGP, which makes use of RSA, many PGP users have adopted the practice of appending their public key to messages that they send to public forums, such as USENET newsgroups and Internet mailing lists.

Although this approach is convenient, it has a major weakness. Anyone can forge such a public announcement. That is, some user could pretend to be user A and send a public key to another participant or broadcast such a public key. Until such time as user A discovers the forgery and alerts other participants, the forger is able to read all encrypted messages intended for A and can use the forged keys for authentication.

Publicly available directory



University of Antwerp | Faculty of Science

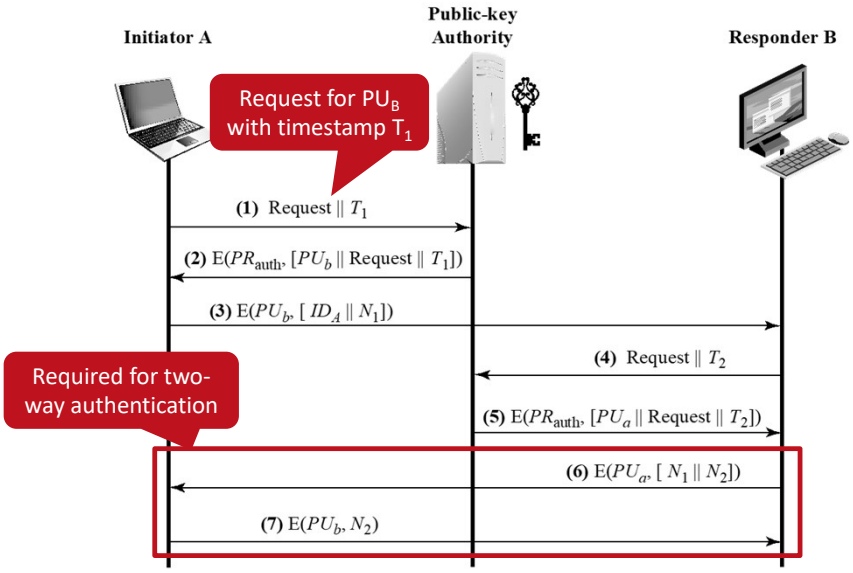
33

A greater degree of security can be achieved by maintaining a publicly available dynamic directory of public keys. Maintenance and distribution of the public directory would have to be the responsibility of some trusted entity or organization. Such a scheme would include the following elements:

1. The authority maintains a directory with a {name, public key} entry for each participant.
2. Each participant registers a public key with the directory authority. Registration would have to be in person or by some form of secure authenticated communication.
3. A participant may replace the existing key with a new one at any time, either because of the desire to replace a public key that has already been used for a large amount of data, or because the corresponding private key has been compromised in some way.
4. Participants could also access the directory electronically. For this purpose, secure, authenticated communication from the authority to the participant is mandatory.

This scheme is clearly more secure than individual public announcements but still has vulnerabilities. If an adversary succeeds in obtaining or computing the private key of the directory authority, the adversary could authoritatively pass out counterfeit public keys and subsequently impersonate any participant and eavesdrop on messages sent to any participant. Another way to achieve the same end is for the adversary to tamper with the records kept by the authority.

Public key authority



Stronger security for public-key distribution can be achieved by providing tighter control over the distribution of public keys from the directory. A typical scenario is above. As before, the scenario assumes that a central authority maintains a dynamic directory of public keys of all participants. In addition, each participant reliably knows a public key for the authority, with only the authority knowing the corresponding private key. The following steps occur:

1. A sends a timestamped message to the public-key authority containing a request for the current public key of B.
2. The authority responds with a message that is encrypted using the authority's private key, PR_{auth}. Thus, A is able to decrypt the message using the authority's public key. Therefore, A is assured that the message originated with the authority. The message includes the following:
    - B's public key, PU_b, which A can use to encrypt messages destined for B.
    - The original request used to enable A to match this response with the corresponding earlier request and to verify that the original request was not altered before reception by the authority.
    - The original timestamp given so A can determine that this is not an old message from the authority containing a key other than B's current public key.
3. A stores B's public key and also uses it to encrypt a message to B containing an identifier of A (ID_A) and a nonce (N_1), which is used to identify this transaction uniquely.
4. (and 5) B retrieves A's public key from the authority in the same manner as A retrieved B's public key.

At this point, public keys have been securely delivered to A and B, and they may begin their protected exchange. However, two additional steps are desirable:

6. B sends a message to A encrypted with PU_a and containing A's nonce (N_1) as well as a new nonce generated by B (N_2). Because only B could have decrypted message (3), the presence of N_1 in message (6) assures A that the correspondent is B.
7. A returns N_2, which is encrypted using B's public key, to assure B that its correspondent is A.

Thus, a total of seven messages are required. However, the initial five messages need be used only infrequently because both A and B can save the other's public key for future use—a technique known as caching. Periodically, a user should request fresh copies of the public keys of its correspondents to ensure currency.

This approach is attractive, yet it has some drawbacks. The public-key authority could be somewhat of a bottleneck in the system, for a user must appeal to the authority for a public key for every other user that it wishes to contact. As before, the directory of names and public keys maintained by the authority is vulnerable to tampering.

1 Public key distribution methods

2 Public key certificates

University of Antwerp
Faculty of Science

Certificates: secure key exchange without an authority

A certificate consists of:

- Public key
- Identifier of the key owner
- Signature of a trusted third party (i.e., certificate authority)

Requirements

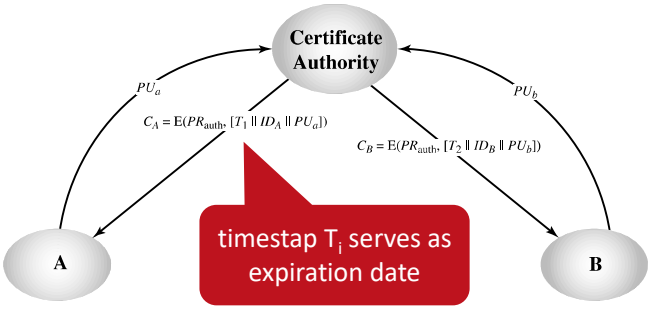
1. Any participant can determine the name and public key of the owner
2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit
3. Only the certificate authority can create and update certificates
4. Any participant can verify the time validity of the certificate

An alternative approach, is to use **certificates** that can be used by participants to exchange keys without contacting a public-key authority, in a way that is as reliable as if the keys were obtained directly from a public-key authority. In essence, a certificate consists of a public key, an identifier of the key owner, and the whole block signed by a trusted third party. Typically, the third party is a certificate authority, such as a government agency or a financial institution, that is trusted by the user community. A user can present his or her public key to the authority in a secure manner and obtain a certificate. The user can then publish the certificate. Anyone needing this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature. A participant can also convey its key information to another by transmitting its certificate. Other participants can verify that the certificate was created by the authority. We can place the following requirements on this scheme:

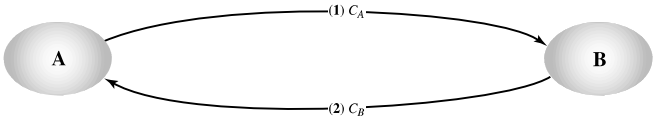
1. Any participant can read a certificate to determine the name and public key of the certificate's owner.
2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.
3. Only the certificate authority can create and update certificates.
4. Any participant can verify the time validity of the certificate.

Certificates: secure key exchange without an authority

Step 1: obtain a certificate for your public key



Step 2: exchange public key certificates



A certificate scheme is illustrated here. Each participant applies to the certificate authority, supplying a public key and requesting a certificate. Application must be in person or by some form of secure authenticated communication. For participant A, the authority provides a certificate of the form:

$$C_A = E(PR_{auth}, [T \,||\, ID_A \,||\, PU_a])$$

where PR_{auth} is the private key used by the authority and T is a timestamp. A may then pass this certificate on to any other participant, who reads and verifies the certificate as follows:

$$D(PU_{auth}, C_A) = D(PU_{auth}, E(PR_{auth}, [T \,||\, ID_A \,||\, PU_a])) = (T \,||\, ID_A \,||\, PU_a)$$

The recipient uses the authority's public key, PU_{auth}, to decrypt the certificate. Because the certificate is readable only using the authority's public key, this verifies that the certificate came from the certificate authority. The elements ID_A and PU_a provide the recipient with the name and public key of the certificate's holder. The timestamp T validates the currency of the certificate.

The timestamp counters the following scenario. A's private key is learned by an adversary. A generates a new private/public key pair and applies to the certificate authority for a new certificate. Meanwhile, the adversary replays the old certificate to B. If B then encrypts messages using the compromised old public key, the adversary can read those messages. In this context, the compromise of a private key is comparable to the loss of a credit card. The owner cancels the credit card number but is at risk until all possible communicants are aware that the old credit card is obsolete. Thus, the

timestamp serves as something like an expiration date. If a certificate is sufficiently old, it is assumed to be expired.

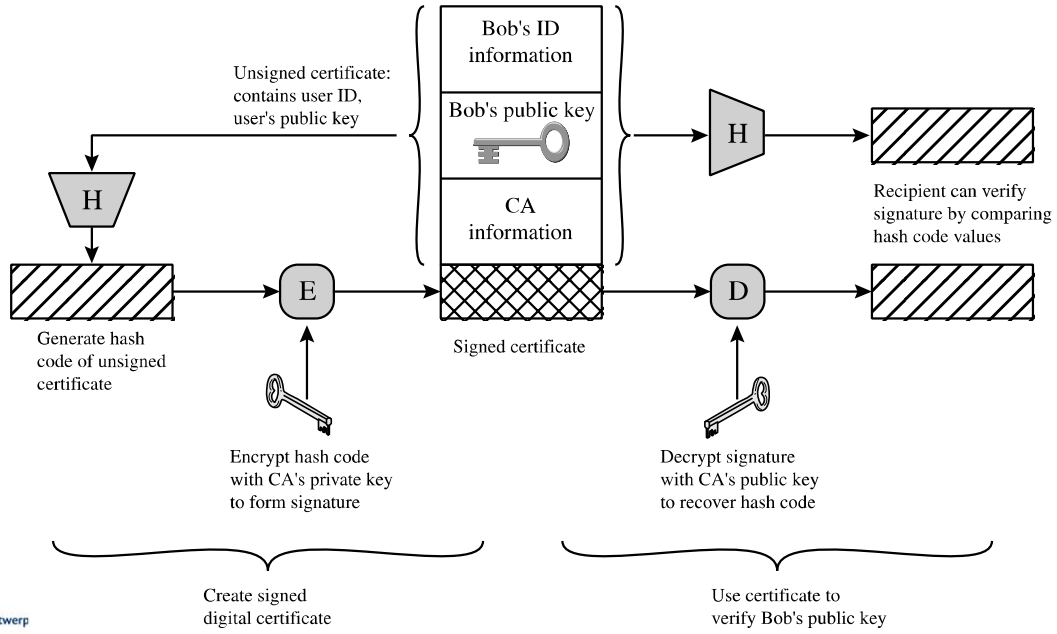
ITU-T X.509 public key certificate recommendation

- Universally accepted format for public-key certificates
- Defines a framework for the provisioning of authentication services
- Used in most network applications: IPSec, TLS, S/MIME, etc.
- 7 different versions are available
  - Version 1 standardized in 1988
  - Version 2 released in 1993 addressed several security concerns
  - Version 3 was the last one to make changes to the certificate format
  - Version 7 was issued in 2012

ITU-T recommendation X.509 is part of the X.500 series of recommendations that define a directory service. The directory is, in effect, a server or distributed set of servers that maintains a database of information about users. The information includes a mapping from username to network address, as well as other attributes and information about the users.

X.509 defines a framework for the provision of authentication services by the X.500 directory to its users. The directory may serve as a repository of public-key certificates. Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority. In addition, X.509 defines alternative authentication protocols based on the use of public-key certificates. X.509 is an important standard because the certificate structure and authentication protocols defined in X.509 are used in a variety of contexts. For example, the X.509 certificate format is used in S/MIME, IP Security, and SSL/TLS. X.509 was initially issued in 1988. The standard was subsequently revised in 1993 to address several security concerns. The standard is currently at version 7, issued in 2012. X.509 is based on the use of public-key cryptography and digital signatures. The standard does not dictate the use of a specific digital signature algorithm nor a specific hash function.

X.509 uses hash-based signatures

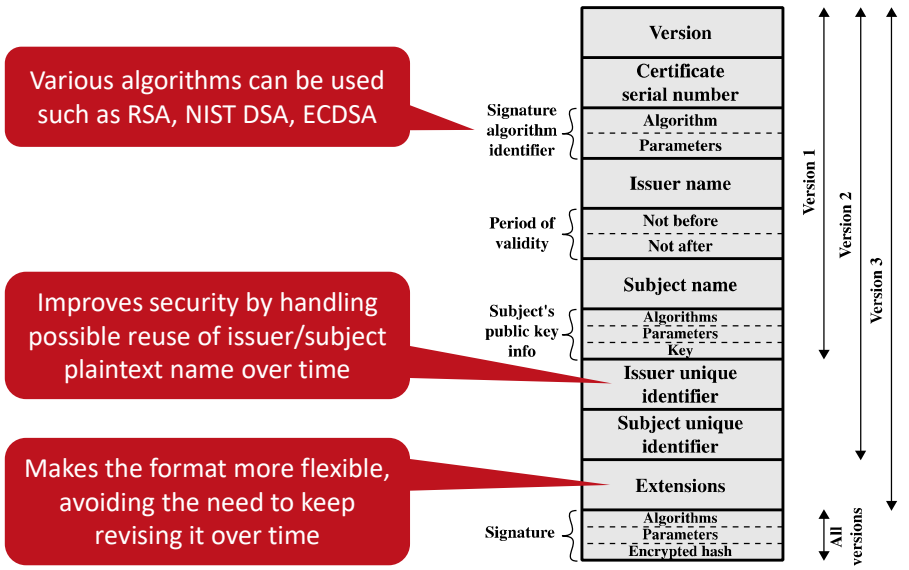


University of Antwerp | Faculty of Science

39

The figure illustrates the overall X.509 scheme for generation of a public-key certificate. The certificate for Bob's public key includes unique identifying information for Bob, Bob's public key, and identifying information about the CA, plus other information as explained subsequently. This information is then signed by computing a hash value of the information and generating a digital signature using the hash value and the CA's private key. X.509 indicates that the signature is formed by encrypting the hash value. This suggests the use of one of the RSA schemes. However, the current version of X.509 does not dictate a specific digital signature algorithm.

X.509 certificate format



The heart of the X.509 scheme is the public-key certificate associated with each user. These user certificates are assumed to be created by some trusted certification authority (CA) and placed in the directory by the CA or by the user. The directory server itself is not responsible for the creation of public keys or for the certification function; it merely provides an easily accessible location for users to obtain certificates.

The figure shows the general format of a certificate, which includes the following elements:

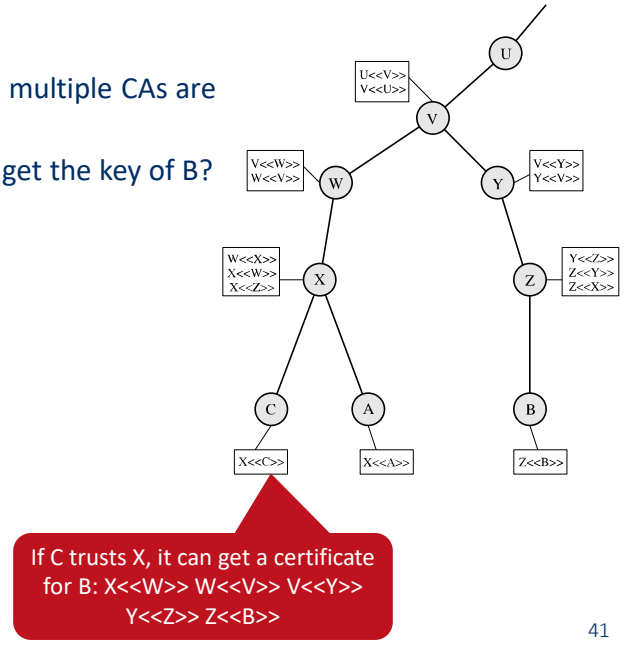
- **Version:** Differentiates among successive versions of the certificate format; the default is version 1. If the *issuer unique identifier* or *subject unique identifier* are present, the value must be version 2. If one or more extensions are present, the version must be version 3. Although the X.509 specification is currently at version 7, no changes have been made to the fields that make up the certificate since version 3.
- **Serial number:** An integer value unique within the issuing CA that is unambiguously associated with this certificate.
- **Signature algorithm identifier:** The algorithm used to sign the certificate together with any associated parameters. Because this information is repeated in the signature field at the end of the certificate, this field has little, if any, utility.
- **Issuer name:** X.500 name of the CA that created and signed this certificate.
- **Period of validity:** Consists of two dates: the first and last on which the certificate is valid.
- **Subject name:** The name of the user to whom this certificate refers. That is, this certificate certifies the public key of the subject who holds the corresponding private key.

- **Subject's public-key information:** The public key of the subject, plus an identifier of the algorithm for which this key is to be used, together with any associated parameters.
- **Issuer unique identifier:** An optional-bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities.
- **Subject unique identifier:** An optional-bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities.
- **Extensions:** A set of one or more extension fields. Extensions were added in version 3 and are discussed later in this section.
- **Signature:** Covers all of the other fields of the certificate. One component of this field is the digital signature applied to the other fields of the certificate. This field includes the signature algorithm identifier.

The unique identifier fields were added in version 2 to handle the possible reuse of subject and/or issuer names over time. These fields are rarely used.

Certificate Authority (CA) organisation

- Y<<X>>: Certificate of user X issued by CA Y
- If the user community is large (e.g., the Internet) then multiple CAs are needed
- User A trusts CA X and user B trusts CA Y. How does A get the key of B?
  - Step 1: X owns Y<<X>> and Y owns X<<Y>>
  - Step 2: A owns X<<A>> and B owns Y<<B>>
  - Step 3: A validates key of B via chain of certificates: X<<Y>> Y<<B>>
- An arbitrarily long chain of CA certificates can be used to validate a key
- Two types of certificates stored by CA X:
  - Forward certificate: Of X generated by other CA
  - Reverse certificate: Of other CA generated by X



If all users subscribe to the same CA, then there is a common trust of that CA. All user certificates can be placed in the directory for access by all users. In addition, a user can transmit his or her certificate directly to other users. In either case, once B is in possession of A's certificate, B has confidence that messages it encrypts with A's public key will be secure from eavesdropping and that messages signed with A's private key are unforgeable.

If there is a large community of users, it may not be practical for all users to subscribe to the same CA. Because it is the CA that signs certificates, each participating user must have a copy of the CA's own public key to verify signatures. This public key must be provided to each user in an absolutely secure (with respect to integrity and authenticity) way so that the user has confidence in the associated certificates. Thus, with many users, it may be more practical for there to be a number of CAs, each of which securely provides its public key to some fraction of the users.

Now suppose that A has obtained a certificate from certification authority X_1 and B has obtained a certificate from CA X_2. If A does not securely know the public key of X_2, then B's certificate, issued by X_2, is useless to A. A can read B's certificate, but A cannot verify the signature. However, if the two CAs have securely exchanged their own public keys, the following procedure will enable A to obtain B's public key.

1. A obtains from the directory the certificate of X_2 signed by X_1. Because A securely knows X_1's public key, A can obtain X_2's public key from its certificate and verify it by means of X_1's signature on the certificate.
2. A then goes back to the directory and obtains the certificate of B signed by X_2.

Because A now has a trusted copy of X_2's public key, A can verify the signature and securely obtain B's public key.

A has used a chain of certificates to obtain B's public key. In the notation of X.509, this chain is expressed as:

$X_1\langle\langle X_2\rangle\rangle$ $X_2\langle\langle B\rangle\rangle$

In the same fashion, B can obtain A's public key with the reverse chain:

$X_2\langle\langle X_1\rangle\rangle$ $X_1\langle\langle A\rangle\rangle$

This scheme need not be limited to a chain of two certificates. An arbitrarily long path of CAs can be followed to produce a chain. A chain with *N* elements would be expressed as

$X_1\langle\langle X_2\rangle\rangle$ $X_2\langle\langle X_3\rangle\rangle$ … $X_N\langle\langle B\rangle\rangle$

In this case, each pair of CAs in the chain (X_i, X_{i+1}) must have created certificates for each other.

All these certificates of CAs by CAs need to appear in the directory, and the user needs to know how they are linked to follow a path to another user's public-key certificate. X.509 suggests that CAs be arranged in a hierarchy so that navigation is straightforward. The figure above is an example of such a hierarchy. The connected circles indicate the hierarchical relationship among the CAs; the associated boxes indicate certificates maintained in the directory for each CA entry. The directory entry for each CA includes two types of certificates:

- **Forward certificates:** Certificates of X generated by other CAs
- **Reverse certificates:** Certificates generated by X that are the certificates of other CAs

In this example, user A can acquire the following certificates from the directory to establish a certification path to B:

X<<W>> W<<V>> V<<Y>> Y<<Z>> Z<<B>>

When A has obtained these certificates, it can unwrap the certification path in sequence to recover a trusted copy of B's public key. Using this public key, A can send encrypted messages to B. If A wishes to receive encrypted messages back from B, or to sign messages sent to B, then B will require A's public key, which can
be obtained from the following certification path:

Z<<Y>> Y<<V>> V<<W>> W<<X>> X<<A>>

B can obtain this set of certificates from the directory, or A can provide them as part of its initial message to B.

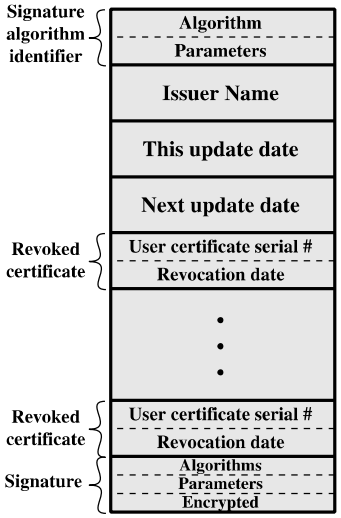
Certificate revocation

It may be needed to **revoke** a certificate that is not expired, if:

- User's private key is possibly compromised
- User is no longer certified by CA
- CA's certificate is possibly compromised

Each CA maintains the **certificate revocation list (CRL)**, with all non-expired but revoked certificates

Certificate revocation list



University of Antwerp | Faculty of Science

42

Each certificate includes a period of validity, much like a credit card. Typically, a new certificate is issued just before the expiration of the old one. In addition, it may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons:

1. The user's private key is assumed to be compromised.
2. The user is no longer certified by this CA. Reasons for this include that the subject's name has changed, the certificate is superseded, or the certificate was not issued in conformance with the CA's policies.
3. The CA's certificate is assumed to be compromised.

Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, including both those issued to users and to other CAs. These lists should also be posted on the directory.

Each certificate revocation list (CRL) posted to the directory is signed by the issuer and includes the issuer's name, the date the list was created, the date the next CRL is scheduled to be issued, and an entry for each revoked certificate. Each entry consists of the serial number of a certificate and revocation date for that certificate. Because serial numbers are unique within a CA, the serial number is sufficient to identify the certificate. When a user receives a certificate in a message, the user must determine whether the certificate has been revoked. The user could check the directory each time a certificate is received. To avoid the delays (and possible costs) associated with directory searches, it is likely that the user would maintain a local cache of certificates and lists of revoked certificates.

X.509 version 3 “Extensions Field”

- X.509 version 3 introduces 3 types of optional extension fields
- Key and policy information
  - Authority key identifier: allows CA to use multiple public keys
  - Subject key identifier: allows user to use multiple public keys
  - Key usage: indicates for which security purposes the key may be used
  - Private-key usage period: may invalidate private key earlier than public key
  - Policy mappings: Indicate equivalence between CA policies
- Certificate subject and issuer attributes
  - Subject alternative name: defines aliases for the user
  - Issuer alternative name: defines aliases for the CA
  - Subject directory attributes: for use with X.500 directories
- Certificate path constraints
  - Basic constraints: indicates if the subject may act as a CA themselves
  - Name constraints: indicates namespace restriction for hierarchical CAs
  - Policy constraints: restrictions on policy mapping on remainder of path

Rather than continue to add fields to a fixed format, standards developers felt that a more flexible approach was needed. Thus, version 3 includes a number of optional extensions that may be added to the version 2 format. Each extension consists of an extension identifier, a criticality indicator, and an extension value. The criticality indicator indicates whether an extension can be safely ignored. If the indicator has a value of TRUE and an implementation does not recognize the extension, it must treat the certificate as invalid. The certificate extensions fall into three main categories: key and policy information, subject and issuer attributes, and certification path constraints:

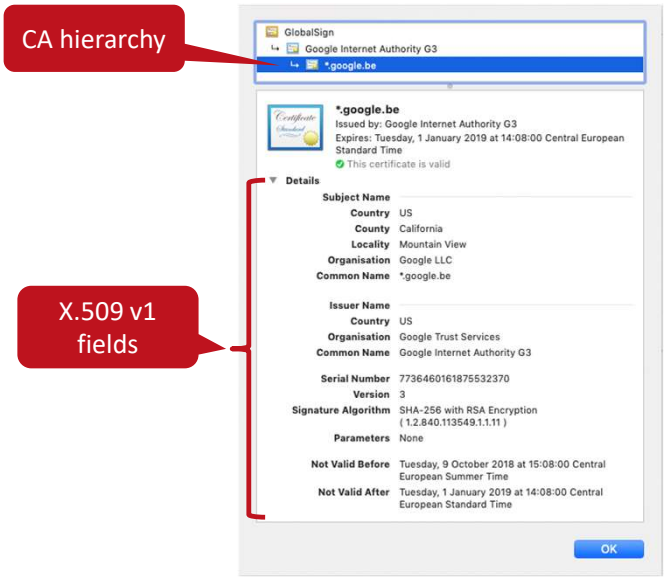
- **Key and policy information:** These extensions convey additional information about the subject and issuer keys, plus indicators of certificate policy. A certificate policy is a named set of rules that indicates the applicability of a certificate to a particular community and/or class of application with common security requirements. For example, a policy might be applicable to the authentication of electronic data interchange (EDI) transactions for the trading of goods within a given price range.
  - Authority key identifier: Identifies the public key to be used to verify the signature on this certificate or CRL. Enables distinct keys of the same CA to be differentiated. One use of this field is to handle CA key pair updating.
  - Subject key identifier: Identifies the public key being certified. Useful for subject key pair updating. Also, a subject may have multiple key pairs

and, correspondingly, different certificates for different purposes (e.g., digital signature and encryption key agreement).

- Key usage: Indicates a restriction imposed as to the purposes for which, and the policies under which, the certified public key may be used. May indicate one or more of the following: digital signature, nonrepudiation, key encryption, data encryption, key agreement, CA signature verification on certificates, CA signature verification on CRLs.
- Private-key usage period: Indicates the period of use of the private key corresponding to the public key. Typically, the private key is used over a different period from the validity of the public key. For example, with digital signature keys, the usage period for the signing private key is typically shorter than that for the verifying public key.
- Certificate policies: Certificates may be used in environments where multiple policies apply. This extension lists policies that the certificate is recognized as supporting, together with optional qualifier information.
- Policy mappings: Used only in certificates for CAs issued by other CAs. Policy mappings allow an issuing CA to indicate that one or more of that issuer's policies can be considered equivalent to another policy used in the subject CA's domain.

- **Certificate subject and issuer attributes:** These extensions support alternative names, in alternative formats, for a certificate subject or certificate issuer and can convey additional information about the certificate subject to increase a certificate user's confidence that the certificate subject is a particular person or entity. For example, information such as postal address, position within a corporation, or picture image may be required.
    - Subject alternative name: Contains one or more alternative names, using any of a variety of forms. This field is important for supporting certain applications, such as electronic mail, EDI, and IPSec, which may employ their own name forms.
    - Issuer alternative name: Contains one or more alternative names, using any of a variety of forms.
    - Subject directory attributes: Conveys any desired X.500 directory attribute values for the subject of this certificate.
- **Certificate path constraints:** These extensions allow constraint specifications to be included in certificates issued for CAs by other CAs. The constraints may restrict the types of certificates that can be issued by the subject CA or that may occur subsequently in a certification chain.
    - Basic constraints: Indicates if the subject may act as a CA. If so, a certification path length constraint may be specified.
    - Name constraints: Indicates a name space within which all subject names in subsequent certificates in a certification path must be located.
    - Policy constraints: Specifies constraints that may require explicit certificate policy identification or inhibit policy mapping for the remainder of the certification path.

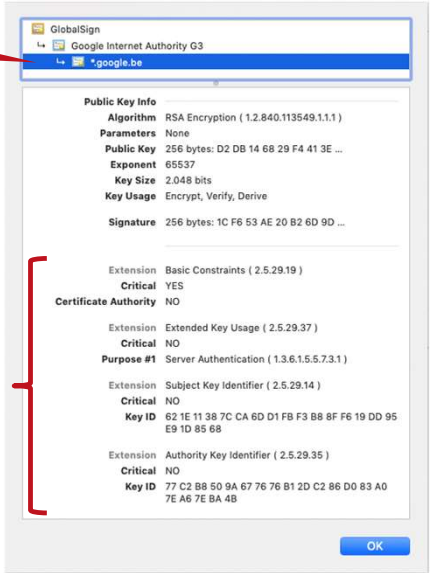
X.509 certificate example



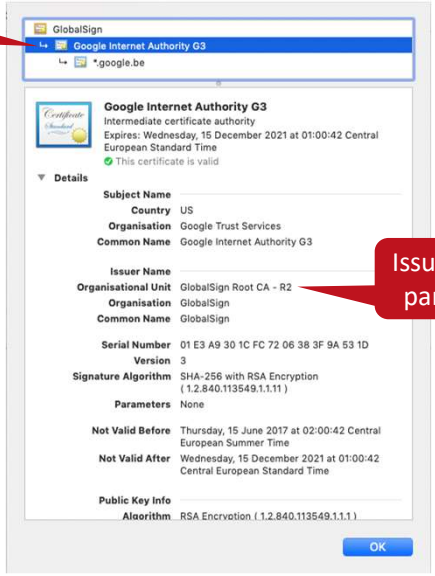
X.509 certificate example

CA hierarchy

X.509 v3 extensions

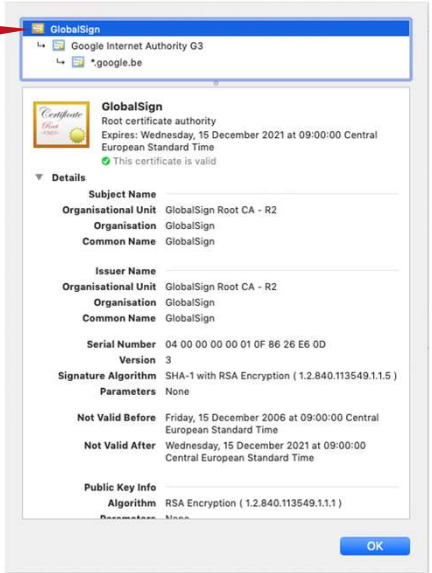


X.509 certificate example

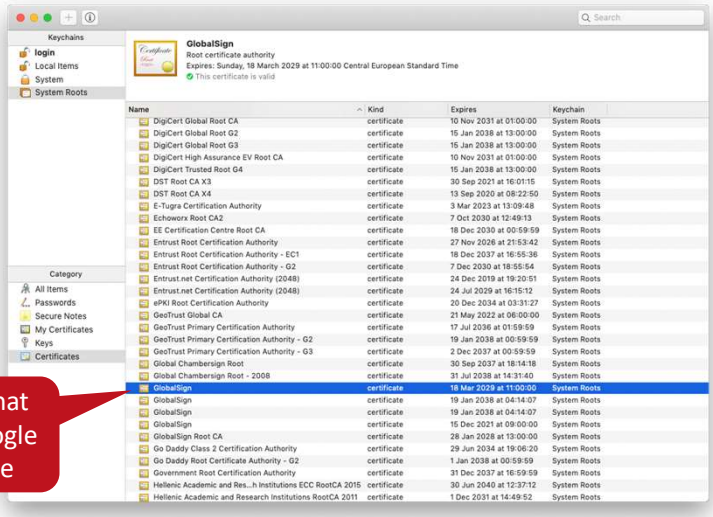


X.509 certificate example

Global Root CA



Operating system keeps list of trusted certificates



Public Key Infrastructure (PKI)

"set of hardware, software, people, policies and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography"

- RFC 4949 (Internet Security Glossary) definition

PKI objective

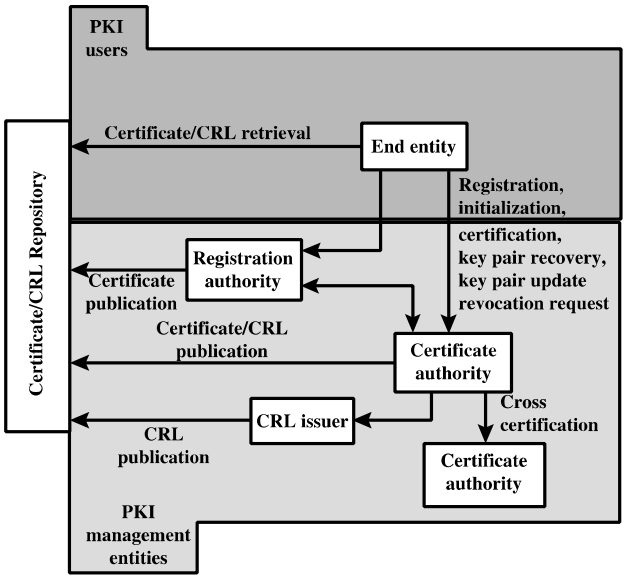
Enable secure, convenient, and efficient acquisition of public keys

Public Key Infrastructure X.509 (PKIX)

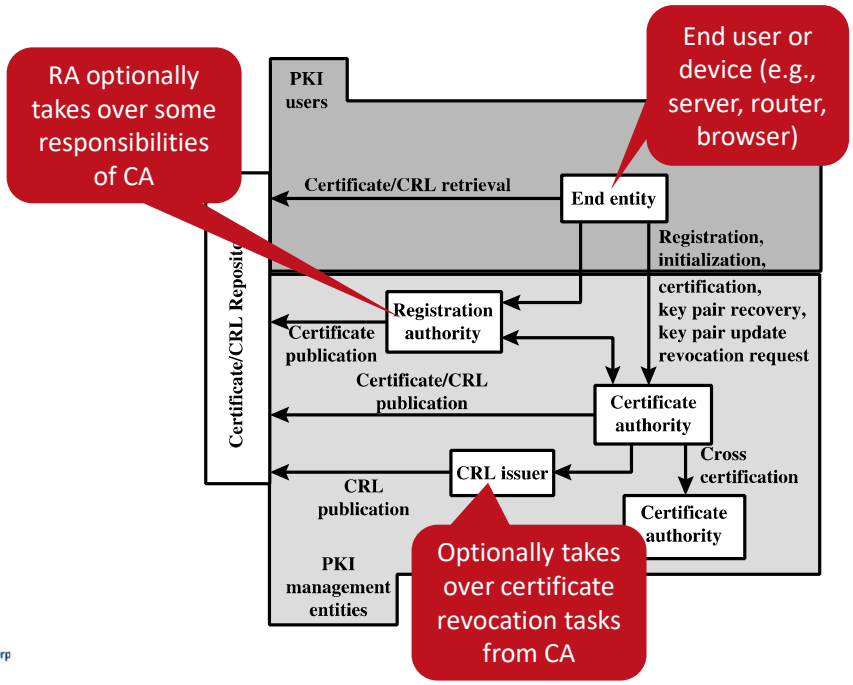
- IETF working group
- Formal, generic model based on X.509 to deploy certificate-based architecture on the Internet

RFC 4949 (*Internet Security Glossary*) defines public-key infrastructure (PKI) as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography. The principal objective for developing a PKI is to enable secure, convenient, and efficient acquisition of public keys. The Internet Engineering Task Force (IETF) Public Key Infrastructure X.509 (PKIX) working group has been the driving force behind setting up a formal (and generic) model based on X.509 that is suitable for deploying a certificate-based architecture on the Internet.

PKIX architecture model



PKIX architecture model



PKIX management functions

- **Management functions provided by one of two protocols:**
  - RFC 2510: certificate management protocols (CMP)
  - RFC 2797: certificate management messages over CMS (CMC)

- **Functions:**
  - Registration: User makes itself known to CA and exchanges shared secret keys
  - Initialization: Initialize the different public and private keys
  - Certification: CA issues a certificate for a user's public key and posts it
  - Key-pair recovery: Recover decryption keys when keying material is not available
  - Key-pair update: Replace keys and certificates over time for extra security
  - Revocation request: Revoke certificate when abnormal behaviour is detected
  - Cross-certification: Two CAs can exchange information and mutual certificates

The PKIX working group has defines two alternative management protocols between PKIX entities that support the management functions listed here. RFC 2510 defines the certificate management protocols (CMP). Within CMP, each of the management functions is explicitly identified by specific protocol exchanges. CMP is designed to be a flexible protocol able to accommodate a variety of technical, operational, and business models. RFC 2797 defines certificate management messages over CMS (CMC), where CMS refers to RFC 2630, cryptographic message syntax. CMC is built on earlier work and is intended to leverage existing implementations. Although all of the PKIX functions are supported, the functions do not all map into specific protocol exchanges.

PKIX identifies a number of management functions that potentially need to be supported by management protocols. These include the following:

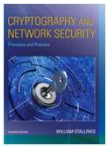
- **Registration:** This is the process whereby a user first makes itself known to a CA (directly or through an RA), prior to that CA issuing a certificate or certificates for that user. Registration begins the process of enrolling in a PKI. Registration usually involves some offline or online procedure for mutual authentication. Typically, the end entity is issued one or more shared secret keys used for subsequent authentication.
- **Initialization:** Before a client system can operate securely, it is necessary to install key materials that have the appropriate relationship with keys stored elsewhere in the infrastructure. For example, the client needs to be securely initialized with the public key and other assured information of the trusted CA(s), to be used in validating certificate paths.
- **Certification:** This is the process in which a CA issues a certificate for a user's public key, returns that certificate to the user's client system, and/or posts that certificate in

a repository.

- **Key-pair recovery:** Key pairs can be used to support digital signature creation and verification, encryption and decryption, or both. When a key pair is used for encryption/decryption, it is important to provide a mechanism to recover the necessary decryption keys when normal access to the keying material is no longer possible, otherwise it will not be possible to recover the encrypted data. Loss of access to the decryption key can result from forgotten passwords/PINs, corrupted disk drives, damage to hardware tokens, and so on. Key pair recovery allows end entities to restore their encryption/decryption key pair from an authorized key backup facility (typically, the CA that issued the end entity's certificate).
- **Key-pair update:** All key pairs need to be updated regularly (i.e., replaced with a new key pair) and new certificates issued. Update is required when the certificate lifetime expires and as a result of certificate revocation.
- **Revocation request:** An authorized person advises a CA of an abnormal situation requiring certificate revocation. Reasons for revocation include private- key compromise, change in affiliation, and name change.
- **Cross certification:** Two CAs exchange information used in establishing a cross-certificate. A cross-certificate is a certificate issued by one CA to another CA that contains a CA signature key used for issuing certificates.

Summary on key distribution

- Symmetric key distribution using symmetric keys
  - Trusted key distribution center (KDC) is responsible for distributing session keys
  - Master key is used to encrypt transmission of session keys
- Symmetric key distribution using asymmetric keys
  - Naïve approach and basic Diffie-Hellman are vulnerable to man-in-the-middle attack
  - Need for secure way to distribute public keys
- Public key distribution
  - Public announcement, public directory, authority, certificates
  - Certificates provide a save way of key distribution without a KDC
- X.509
  - Uses cryptographic hashes and asymmetric encryption to sign and distribute public keys via certificates
  - Uses hierarchy of Certificate Authorities (CAs)
  - Public key infrastructure (PKI) architecture built around X.509



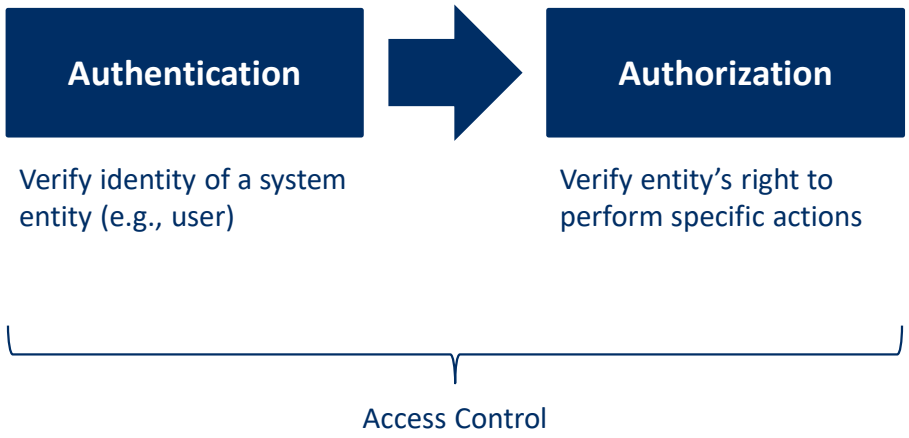
Link with the book

- Chapter 10 (10.1)
- Chapter 14

User Management

University of Antwerp
Faculty of Science

Definitions



University of Antwerp
Faculty of Science

55

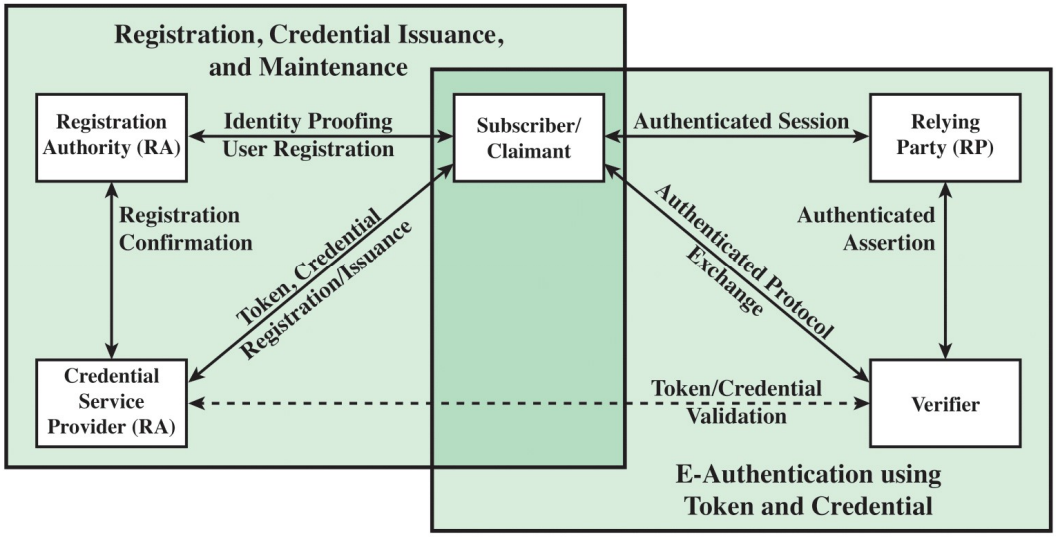
In most computer security contexts, user authentication is the fundamental building block and the primary line of defence. User authentication is the basis for most types of access control and for user accountability. RFC 4949 (*Internet Security Glossary*) defines **user authentication as the process of verifying an identity claimed by or for a system entity**. This process consists of two steps:

- Identification step: Presenting an identifier to the security system. (Identifiers should be assigned carefully, because authenticated identities are the basis for other security services, such as access control service.)
- Verification step: Presenting or generating authentication information that corroborates the binding between the entity and the identifier.

Once the user or system has been authenticated, we use **authorization to decide whether he has the rights** to perform the action he is trying to execute.

Note that user authentication is distinct from message authentication. Message authentication is a procedure that allows communicating parties to verify that the contents of a received message have not been altered and that the source is authentic.

NIST model for user authentication



Source: *NIST SP 800-63-2 E-Authentication Architectural Model*

University of Antwerp | Faculty of Science

56

NIST SP 800-63-2 (*Electronic Authentication Guideline,* August 2013) defines **electronic user authentication as the process of establishing confidence in user identities** that are presented electronically to an information system. Systems can use the authenticated identity to determine if the authenticated individual is authorized to perform particular functions, such as database transactions or access to system resources. SP 800-63-2 defines a general model for user authentication that involves a number of entities and procedures.

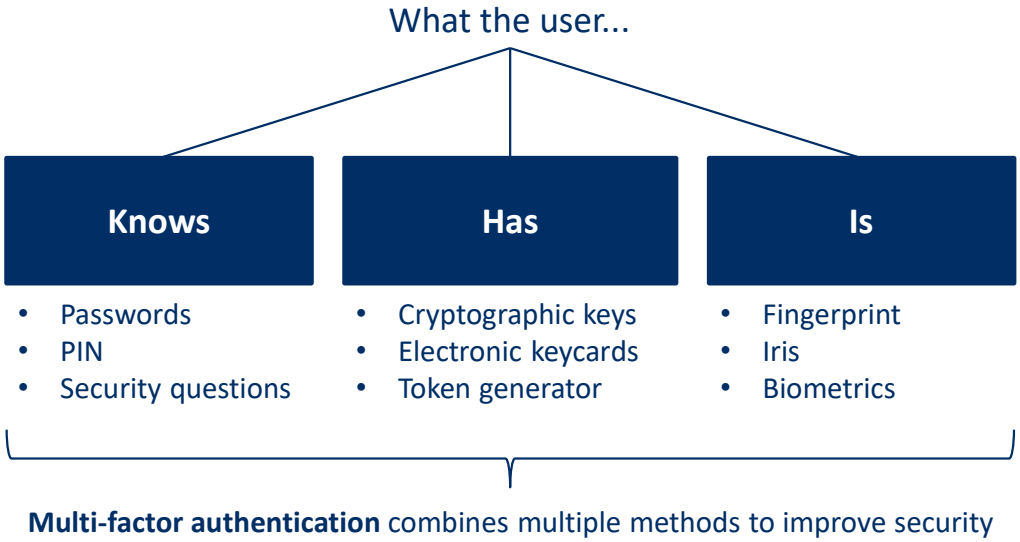
The initial requirement for performing user authentication is that the user must be registered with the system. The following is a typical sequence for registration.

An applicant applies to a **registration authority (RA)** to become a **subscriber** of a **credential service provider (CSP)**. In this model, the RA is a trusted entity that establishes and vouches for the identity of an applicant to a CSP. The CSP then engages in an exchange with the subscriber. Depending on the details of the overall authentication system, the CSP issues some sort of electronic credential to the subscriber. The **credential** is a data structure that authoritatively binds an identity and additional attributes to a token possessed by a subscriber, and can be verified when presented to the verifier in an authentication transaction. The token could be an encryption key or an encrypted password that identifies the subscriber. The token may be issued by the CSP, generated directly by the subscriber, or provided by a third party. The token and credential may be used in subsequent authentication events.

Once a user is registered as a subscriber, the actual authentication process can take

place between the subscriber and one or more systems that perform authentication and, subsequently, authorization. The party to be authenticated is called a **claimant** and the party verifying that identity is called a **verifier**. When a claimant successfully demonstrates possession and control of a token to a verifier through an authentication protocol, the verifier can verify that the claimant is the subscriber named in the corresponding credential. The verifier passes on an assertion about the identity of the subscriber to the **relying party (RP)**. That assertion includes identity information about a subscriber, such as the subscriber name, an identifier assigned at registration, or other subscriber attributes that were verified in the registration process. The RP can use the authenticated information provided by the verifier to make access control or authorization decisions.

Means of authentication



Multi-factor authentication combines multiple methods to improve security

University of Antwerp
Faculty of Science

57

There are four general means of authenticating a user's identity, which can be used alone or in combination:

- **Something the individual knows:** Examples include a password, a personal identification number (PIN), or answers to a prearranged set of questions.
- **Something the individual possesses:** Examples include cryptographic keys, electronic keycards, smart cards, and physical keys. This type of authenticator is referred to as a *token*.
- **Something the individual is (static biometrics):** Examples include recognition by fingerprint, retina, and face.
- **Something the individual does (dynamic biometrics):** Examples include recognition by voice pattern, handwriting characteristics, and typing rhythm.

All of these methods, properly implemented and used, can provide secure user authentication. However, each method has problems. An adversary may be able to guess or steal a password. Similarly, an adversary may be able to forge or steal a token. A user may forget a password or lose a token. Furthermore, there is a significant administrative overhead for managing password and token information on systems and securing such information on systems. With respect to biometric authenticators, there are a variety of problems, including dealing with false positives and false negatives, user acceptance, cost, and convenience. For network-based user authentication, the most important methods involve cryptographic keys and something the individual knows, such as a password.

Mutual authentication

- **Mutual authentication protocols** enable communicating parties to mutually agree on each other's identity while securely exchanging session keys
- Mutual authentication protocols require **confidentiality** and **timeliness**
  - Confidentiality (using encryption) prevents **masquerade attacks** and compromise of session keys
  - Timeliness avoids **replay attacks**
- Methods to ensure timeliness
  - **Sequence numbers**: Generally avoided as each party needs to keep track of last sequence number (overhead!)
  - **Timestamps**: Difficult to keep clocks among participants synchronized in a connection-oriented network setting
  - **Challenge/response** (i.e., nonce): Unsuitable for connection-less transmission due to need for a handshake

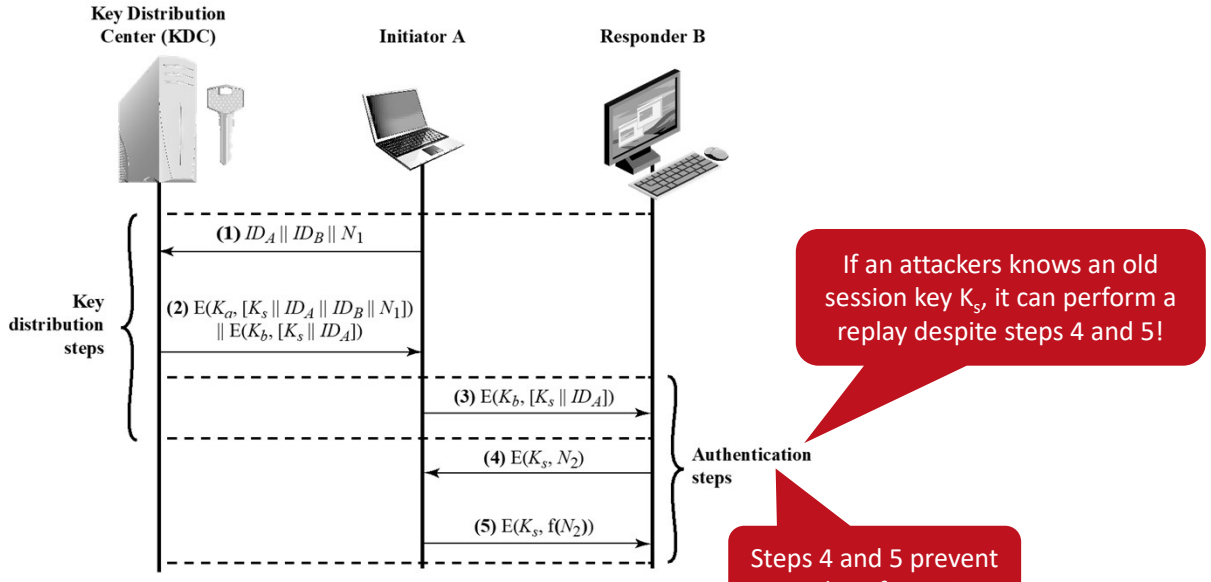
An important application area is that of mutual authentication protocols. Such protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys. Central to the problem of authenticated key exchange are two issues: confidentiality and timeliness. To prevent masquerade and to prevent compromise of session keys, essential identification and session-key information must be communicated in encrypted form. This requires the prior existence of secret or public keys that can be used for this purpose. The second issue, timeliness, is important because of the threat of message replays. Such replays, at worst, could allow an opponent to compromise a session key or successfully impersonate another party. At minimum, a successful replay can disrupt operations by presenting parties with messages that appear genuine but are not.

One approach to coping with replay attacks is to attach a sequence number to each message used in an authentication exchange. A new message is accepted only if its sequence number is in the proper order. The difficulty with this approach is that it requires each party to keep track of the last sequence number for each claimant it has dealt with. Because of this overhead, sequence numbers are generally not used for authentication and key exchange. Instead, one of the following two general approaches is used:

- **Timestamps:** Party A accepts a message as fresh only if the message contains a **timestamp** that, in A's judgment, is close enough to A's knowledge of current time. This approach requires that clocks among the various participants be synchronized.
- **Challenge/response:** Party A, expecting a fresh message from B, first sends B a **nonce** (challenge) and requires that the subsequent message (response) received from B

contain the correct nonce value.

KDC performs key exchange and mutual authentication



University of Antwerp | Faculty of Science

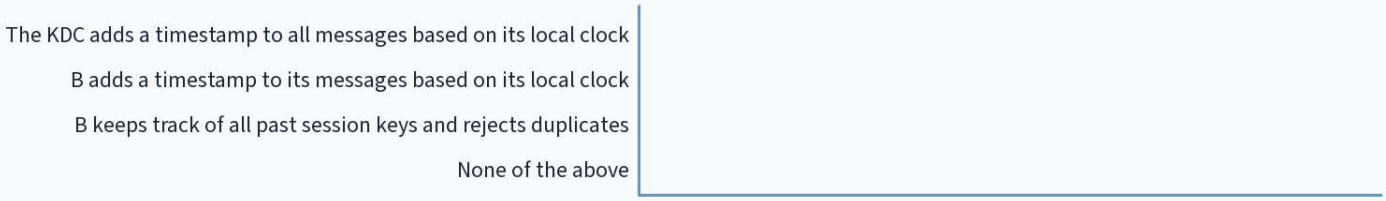
59

Despite the handshake of steps 4 and 5, the protocol is still vulnerable to a form of replay attack. Suppose that an opponent, X, has been able to compromise an old session key. Admittedly, this is a much more unlikely occurrence than that an opponent has simply observed and recorded step 3. Nevertheless, it is a potential security risk. X can impersonate A and trick B into using the old key by simply replaying step 3. Unless B remembers indefinitely all previous session keys used with A, B will be unable to determine that this is a replay. If X can intercept the handshake message in step 4, then it can impersonate A's response in step 5. From this point on, X can send bogus messages to B that appear to B to come from A using an authenticated session key.

When poll is active, respond at **pollev.com/jfamaey**

Text **JFAMAEY** to **+32 460 20 00 56** once to join

What is the most effective method to protect against replaying old compromised session keys in the KDC scenario?



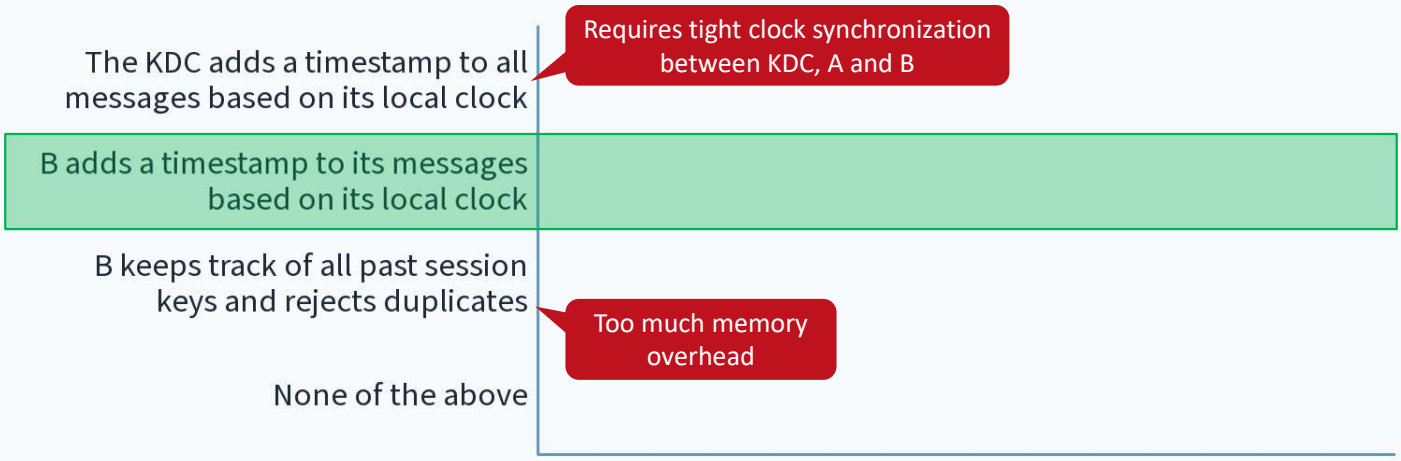
60

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

What is the most effective method to protect against replaying old compromised session keys in the KDC scenario?
https://www.polleverywhere.com/multiple_choice_polls/Ppc1LpqJuzUb6megRWnBT?state=opened&flow=Default&onscreen=persist

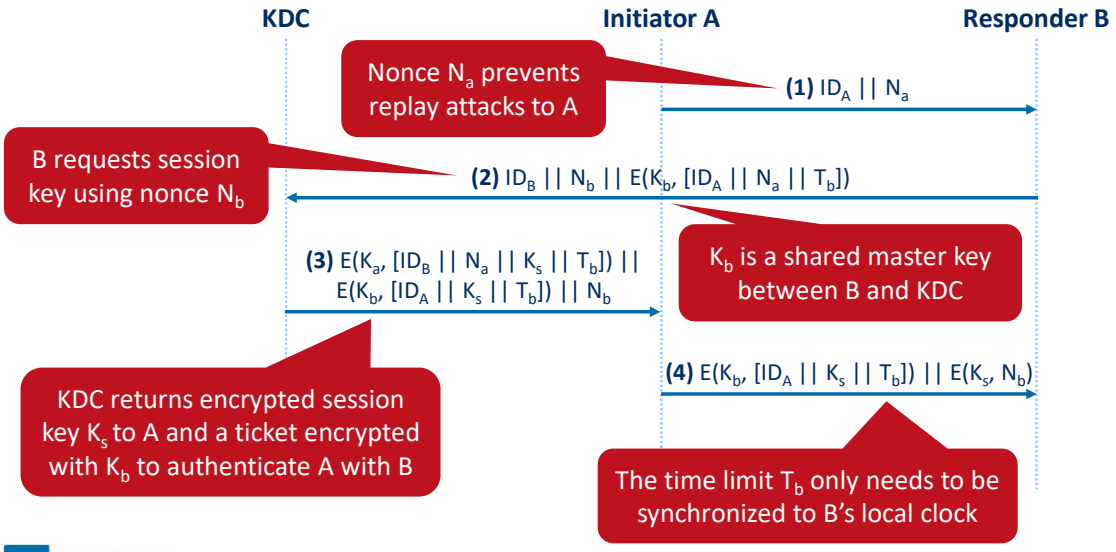
What is the most effective method to protect against replaying old compromised session keys in the KDC scenario?



61

Poll Title: What is the most effective method to protect against replaying old compromised session keys in the KDC scenario?

A secure KDC that avoids replay of compromised session keys



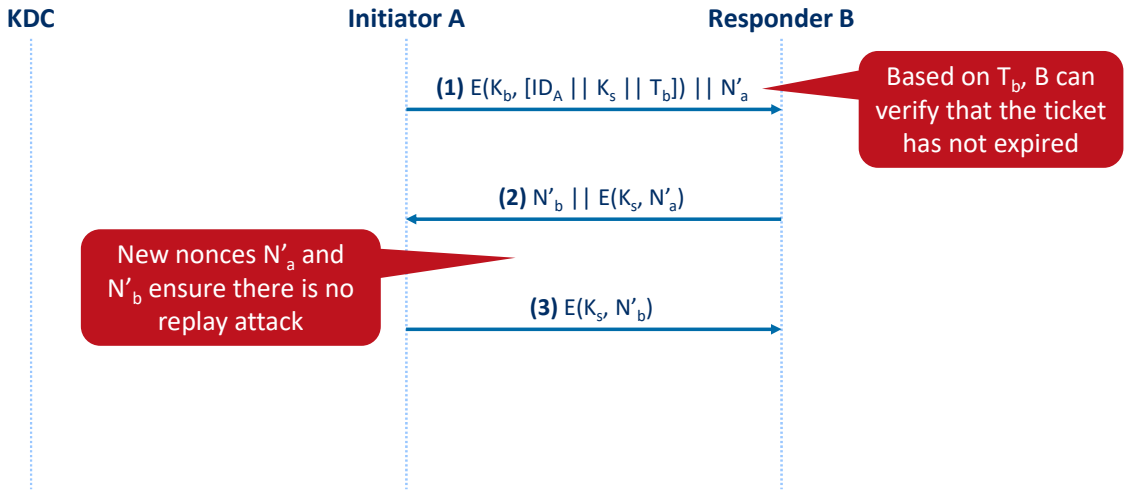
University of Antwerp | Faculty of Science

62

Let us follow this exchange step by step:

1. A initiates the authentication exchange by generating a nonce, N_a, and sending that plus its identifier to B in plaintext. This nonce will be returned to A in an encrypted message that includes the session key, assuring A of its timeliness.
2. B alerts the KDC that a session key is needed. Its message to the KDC includes its identifier and a nonce, N_b. This nonce will be returned to B in an encrypted message that includes the session key, assuring B of its timeliness. B's message to the KDC also includes a block encrypted with the secret key shared by B and the KDC. This block is used to instruct the KDC to issue credentials to A; the block specifies the intended recipient of the credentials, a suggested expiration time for the credentials, and the nonce received from A.
3. The KDC passes on to A B's nonce and a block encrypted with the secret key that B shares with the KDC. The block serves as a "ticket" that can be used by A for subsequent authentications, as will be seen. The KDC also sends to A a block encrypted with the secret key shared by A and the KDC. This block verifies that B has received A's initial message (ID_B) and that this is a timely message and not a replay (N_a), and it provides A with a session key (K_s) and the time limit on its use (T_b).
4. A transmits the ticket to B, together with the B's nonce, the latter encrypted with the session key. The ticket provides B with the secret key that is used to decrypt $E(K_s, N_b)$ to recover the nonce. The fact that B's nonce is encrypted with the session key authenticates that the message came from A and is not a replay.

Establishing a new session without the KDC



Suppose that A and B establish a session using the aforementioned protocol and then conclude that session. Subsequently, but within the time limit established by the protocol, A desires a new session with B. The following protocol ensues:

1. A → B: E(K_b, [ID_A || K_s || T_b]) || N'_a
2. B → A: N'_b || E(K_s, N'_a)
3. A → B: E(K_s, N'_b)

When B receives the message in step 1, it verifies that the ticket has not expired. The newly generated nonces N'_a and N'_b assure each party that there is no replay attack.

In all the foregoing, the time specified in T_b is a time relative to B's clock. Thus, this timestamp does not require synchronized clocks, because B checks only self-generated timestamps.

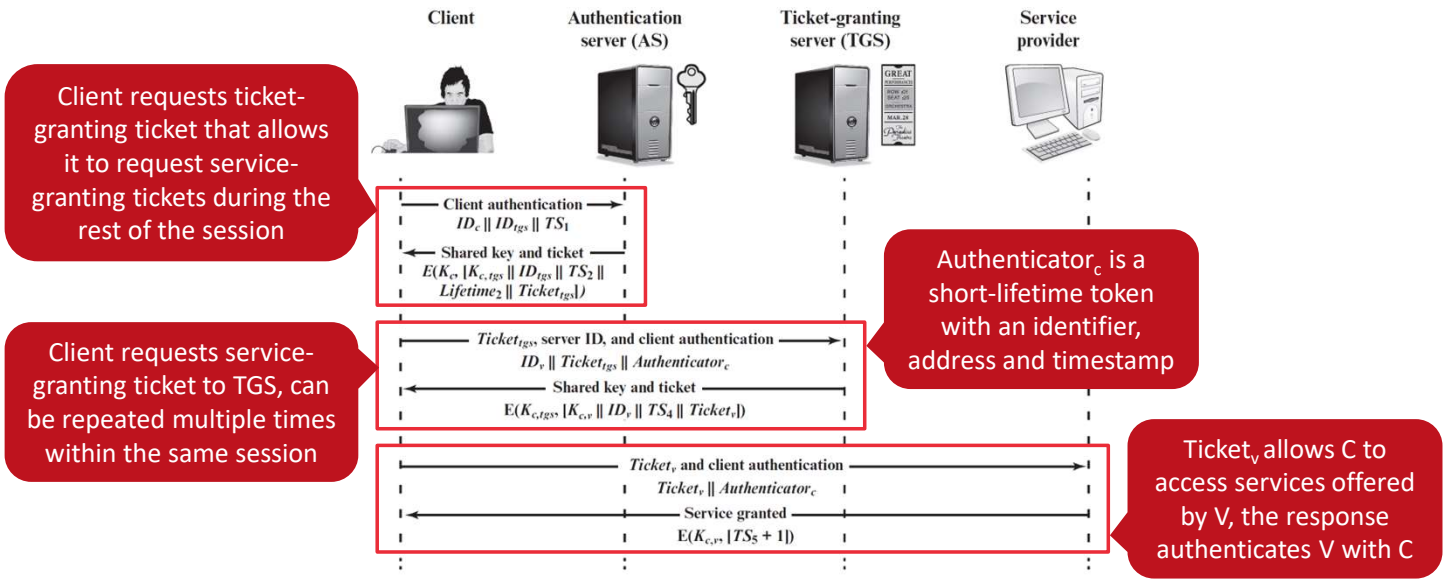
Kerberos – User authentication in distributed environments

- **Goal:** Allow servers in a distributed environment to restrict access to authorized users and authenticate requests for service, where users can make use of different (untrusted) workstations
- Kerberos provides a centralized authentication server that performs mutual authentication solely with symmetric encryption
- Version 4 was published in 1988, and an improved v5 in 1993 (standardized as RFCs 4120 and 4121)
- Mainly used in local area networks, as Kerberos enforces strict synchronization across the participating devices

Kerberos is an authentication service developed as part of Project Athena at MIT. The problem that Kerberos addresses is this: Assume an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network. We would like for servers to be able to restrict access to authorized users and to be able to authenticate requests for service. In this environment, a workstation cannot be trusted to identify its users correctly to network services.

Rather than building in elaborate authentication protocols at each server, Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Unlike most other authentication schemes, Kerberos relies exclusively on symmetric encryption, making no use of public-key encryption. Two versions of Kerberos are in common use. Version 4 implementations still exist. Version 5 corrects some of the security deficiencies of version 4 and has been issued as a proposed Internet Standard (RFC 4120 and RFC 4121).

Authentication in Kerberos version 4



The client sends a message to the AS requesting access to the TGS. The AS responds with a message, encrypted with a key derived from the user's password (K_c), that contains the ticket. The encrypted message also contains a copy of the session key, $K_{c,tgs}$, where the subscripts indicate that this is a session key for C and TGS. Because this session key is inside the message encrypted with K_c, only the user's client can read it. The same session key is included in the ticket ($Ticket_{tgs}$), which can be read only by the TGS. Thus, the session key has been securely delivered to both C and the TGS. Note that several additional pieces of information have been added to this first phase of the dialogue. Message (1) includes a timestamp (TS_1), so that the AS knows that the message is timely. Message (2) includes several elements of the ticket in a form accessible to C. This enables C to confirm that this ticket is for the TGS and to learn its expiration time.

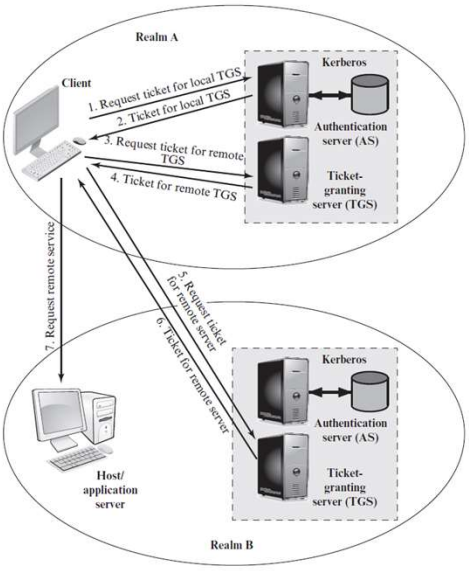
Armed with the ticket and the session key, C is ready to approach the TGS. As before, C sends the TGS a message that includes the ticket plus the ID of the requested service. In addition, C transmits an authenticator, which includes the ID and address of C's user and a timestamp. Unlike the ticket, which is reusable, the authenticator is intended for use only once and has a very short lifetime. The TGS can decrypt the ticket with the key that it shares with the AS. This ticket indicates that user C has been provided with the session key $K_{c,tgs}$. In effect, the ticket says, "Anyone who uses $K_{c,tgs}$ must be C." The TGS uses the session key to decrypt the authenticator. The TGS can then check the name (ID_C) and address (AD_C) from the authenticator with that of the ticket and with the network address of the incoming message. If all match, then the TGS is assured that the sender of the ticket is indeed the ticket's real owner. In effect, the authenticator says, "At time TS_3,

I hereby use $K_{c,tgs}$." Note that the ticket does not prove anyone's identity but is a way to distribute keys securely. It is the authenticator that proves the client's identity. Because the authenticator can be used only once and has a short lifetime, the threat of an opponent stealing both the ticket and the authenticator for presentation later is countered. The reply from the TGS in message (4) follows the form of message (2). The message is encrypted with the session key shared by the TGS and C and includes a session key to be shared between C and the server V ($K_{c,v}$), the ID of V, and the timestamp of the ticket (TS_4). The ticket itself ($Ticket_v$) includes the same session key.

C now has a reusable service-granting ticket for V. When C presents this ticket, as shown in message (5), it also sends an authenticator. The server can decrypt the ticket, recover the session key, and decrypt the authenticator. If mutual authentication is required, the server can reply as shown in message (6). The server returns the value of the timestamp from the authenticator, incremented by 1, and encrypted in the session key. C can decrypt this message to recover the incremented timestamp. Because the message was encrypted by the session key, C is assured that it could have been created only by V. The contents of the message assure C that this is not a replay of an old reply.

Finally, at the conclusion of this process, the client and server share a secret key. This key can be used to encrypt future messages between the two or to exchange a new random session key for that purpose.

Kerberos realms



- Clients can authenticate with servers in different administrative organisations (i.e., realms) using inter-realm authentication
- Requires mutual trust between Kerberos ASs
- Inter-realm authentication steps
  1. C → AS: $ID_c \,||\, ID_{tgs} \,||\, TS_1$
  2. AS → C: $E(K_c, [K_{c,tgs} \,||\, ID_{tgs} \,||\, TS_2 \,||\, Lifetime_2 \,||\, Ticket_{tgs}])$
  3. C → TGS: $ID_{tgsrem} \,||\, Ticket_{tgs} \,||\, Authenticator_c$
  4. TGS → C: $E(K_{c,tgs}, [K_{c,tgsrem} \,||\, ID_{tgsrem} \,||\, TS_4 \,||\, Ticket_{tgsrem}])$
  5. C → TGS_{rem}: $ID_{vrem} \,||\, Ticket_{tgsrem} \,||\, Authenticator_c$
  6. TGS_{rem} → C: $E(K_{c,tgsrem}, [K_{c,vrem} \,||\, ID_{vrem} \,||\, TS_6 \,||\, Ticket_{vrem}])$
  7. C → V_{rem}: $Ticket_{vrem} \,||\, Authenticator_c$

University of Antwerp | Faculty of Science

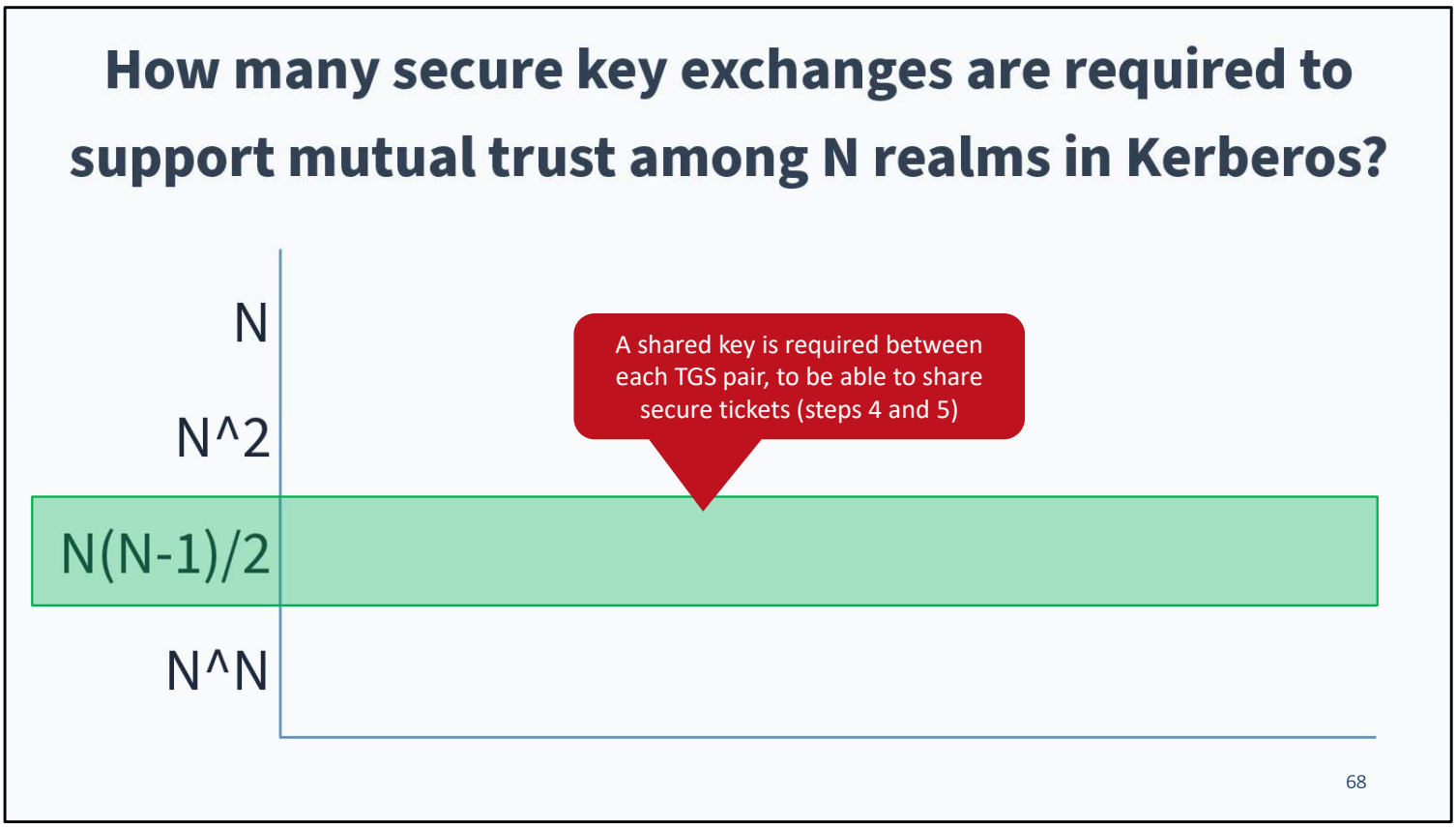
66

A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers is referred to as a Kerberos realm. The concept of realm can be explained as follows. A Kerberos realm is a set of managed nodes that share the same Kerberos database. Networks of clients and servers under different administrative organizations typically constitute different realms. That is, it generally is not practical or does not conform to administrative policy to have users and servers in one administrative domain registered with a Kerberos server elsewhere. However, users in one realm may need access to servers in other realms, and some servers may be willing to provide service to users from other realms, provided that those users are authenticated. Kerberos provides a mechanism for supporting such interrealm authentication. For two realms to support interrealm authentication the Kerberos server in each interoperating realm shares a secret key with the server in the other realm. The two Kerberos servers are registered with each other. The scheme requires that the Kerberos server in one realm trust the Kerberos server in the other realm to authenticate its users. Furthermore, the participating servers in the second realm must also be willing to trust the Kerberos server in the first realm. A user wishing service on a server in another realm needs a ticket for that server. The user's client follows the usual procedures to gain access to the local TGS and then requests a ticket-granting ticket for a remote TGS (TGS in another realm). The client can then apply to the remote TGS for a service-granting ticket for the desired server in the realm of the remote TGS. The ticket presented to the remote server (V_{rem}) indicates the realm in which the user was originally authenticated. The server chooses whether to honor the remote request. One problem presented by the foregoing approach is that it does not scale well to many realms. If there are N realms, then there must be N(N - 1)/2 secure key exchanges so

that each Kerberos realm can interoperate with all other Kerberos realms.

Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

How many secure key exchanges are required to support mutual trust among N realms in Kerberos?
https://www.polleverywhere.com/multiple_choice_polls/uqGHa04TE06lDgBxo0lpP?state=opened&flow=Default&onscreen=persist



Poll Title: How many secure key exchanges are required to support mutual trust among N realms in Kerberos?

Kerberos Version 5 – RFC 4120

Addresses several limitations and technical shortcomings of Version 4

- Environmental shortcomings
  1. Flexibility to use other encryption algorithms besides (the insecure) DES
  2. Flexibility to use other address types besides IPv4
  3. Standardized unambiguous byte ordering based on ASN.1 and Basic Encoding Rules (BER)
  4. Arbitrary maximum ticket lifetime (above 1280 minutes)
  5. Credential forwarding allowing servers to access other servers on behalf of clients
  6. More scalable inter-realm coordination that does not require N^2 relationships among N realms
- Technical deficiencies
  1. Avoid double encryption of tickets to improve computational performance
  2. Replace insecure propagating CBC (PCBC) with standard CBC and an encrypted hash value for confidentiality and integrity
  3. Add subsession keys to encrypt a single connection, rather than reusing the same session key for all connections
  4. Add a pre-authentication step to avoid password guessing attacks based on keys derived from client password

Kerberos version 5 is specified in RFC 4120 and provides a number of improvements over version 4. Version 5 is intended to address the limitations of version 4 in two areas: environmental shortcomings and technical deficiencies. This led to the following environmental shortcomings to be addressed:

1. Encryption system dependence: Version 4 requires the use of DES. Export restriction on DES as well as doubts about the strength of DES were thus of concern. In version 5, ciphertext is tagged with an encryption-type identifier so that any encryption technique may be used. Encryption keys are tagged with a type and a length, allowing the same key to be used in different algorithms and allowing the specification of different variations on a given algorithm.
2. Internet protocol dependence: Version 4 requires the use of Internet Protocol (IP) addresses. Other address types, such as the ISO network address, are not accommodated. Version 5 network addresses are tagged with type and length, allowing any network address type to be used.
3. Message byte ordering: In version 4, the sender of a message employs a byte ordering of its own choosing and tags the message to indicate least significant byte in lowest address or most significant byte in lowest address. This techniques works but does not follow established conventions. In version 5, all message structures are defined using Abstract Syntax Notation One (ASN.1) and Basic Encoding Rules (BER), which provide an unambiguous byte ordering.
4. Ticket lifetime: Lifetime values in version 4 are encoded in an 8-bit quantity in units of five minutes. Thus, the maximum lifetime that can be expressed is 28 * 5 = 1280 minutes (a little over 21 hours). This may be inadequate for some applications (e.g., a long-running simulation that requires valid Kerberos credentials throughout

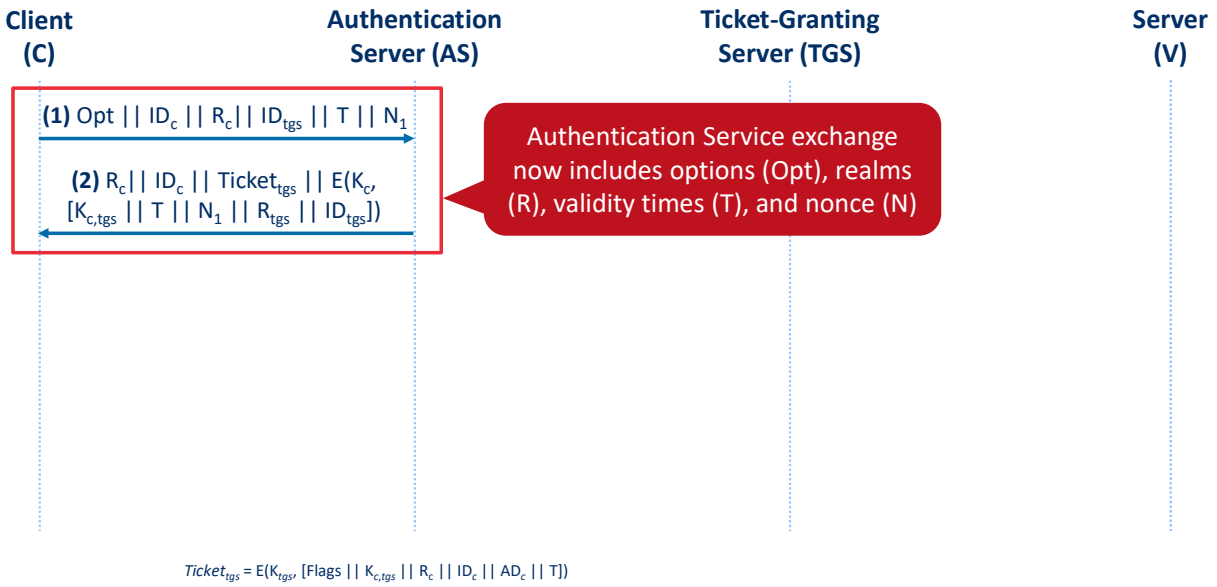
execution). In version 5, tickets include an explicit start time and end time, allowing tickets with arbitrary lifetimes.

5. Authentication forwarding: Version 4 does not allow credentials issued to one client to be forwarded to some other host and used by some other client. This capability would enable a client to access a server and have that server access another server on behalf of the client. For example, a client issues a request to a print server that then accesses the client's file from a file server, using the client's credentials for access. Version 5 provides this capability.
6. Interrealm authentication: In version 4, interoperability among N realms requires on the order of N^2 Kerberos-to-Kerberos relationships, as described earlier. Version 5 supports a method that requires fewer relationships, as described shortly.

Apart from these environmental limitations, there are technical deficiencies in the version 4 protocol itself, that version 5 tries to address:

1. Double encryption: Note in Table 15.1 [messages (2) and (4)] that tickets provided to clients are encrypted twice—once with the secret key of the target server and then again with a secret key known to the client. The second encryption is not necessary and is computationally wasteful.
2. PCBC encryption: Encryption in version 4 makes use of a nonstandard mode of DES known as propagating cipher block chaining (PCBC). It has been demonstrated that this mode is vulnerable to an attack involving the interchange of ciphertext blocks. PCBC was intended to provide an integrity check as part of the encryption operation. Version 5 provides explicit integrity mechanisms, allowing the standard CBC mode to be used for encryption. In particular, a checksum or hash code is attached to the message prior to encryption using CBC.
3. Session keys: Each ticket includes a session key that is used by the client to encrypt the authenticator sent to the service associated with that ticket. In addition, the session key may subsequently be used by the client and the server to protect messages passed during that session. However, because the same ticket may be used repeatedly to gain service from a particular server, there is the risk that an opponent will replay messages from an old session to the client or the server. In version 5, it is possible for a client and server to negotiate a subsession key, which is to be used only for that one connection. A new access by the client would result in the use of a new subsession key.
4. Password attacks: Both versions are vulnerable to a password attack. The message from the AS to the client includes material encrypted with a key based on the client's password. An opponent can capture this message and attempt to decrypt it by trying various passwords. If the result of a test decryption is of the proper form, then the opponent has discovered the client's password and may subsequently use it to gain authentication credentials from Kerberos. Version 5 does provide a mechanism known as preauthentication, which should make password attacks more difficult, but it does not prevent them.

Kerberos version 5 – Authentication service exchange



$Ticket_{tgs} = E(K_{tgs}, [Flags || K_{c,tgs} || R_c || ID_c || AD_c || T])$

University of Antwerp | Faculty of Science

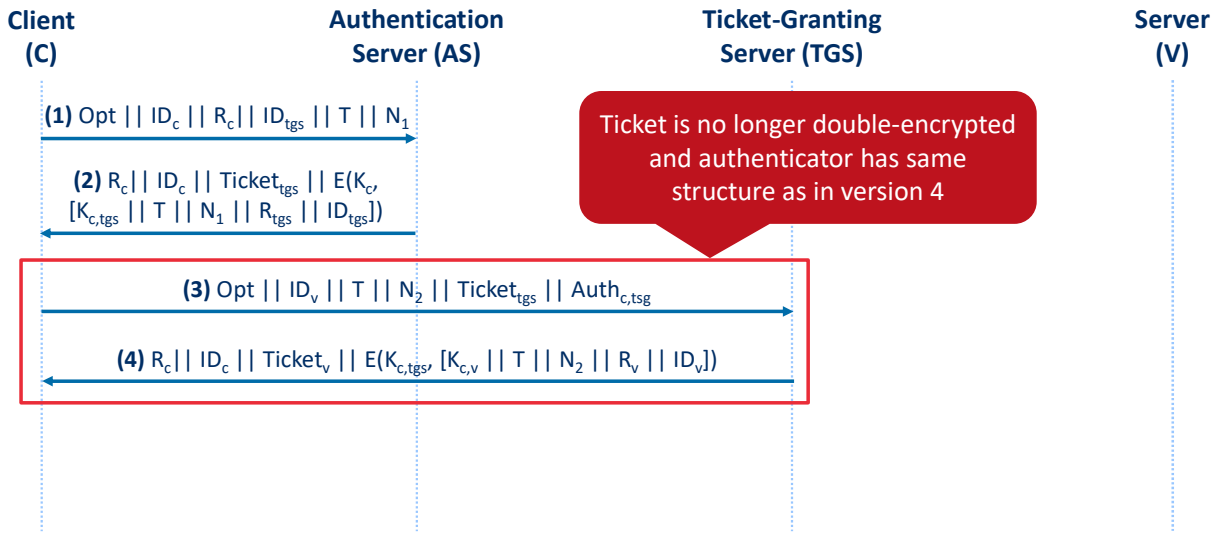
70

Kerberos version 5 is best explained by comparison with version 4. First, consider the authentication service exchange. Message (1) is a client request for a ticket-granting ticket. As before, it includes the ID of the user and the TGS. The following new elements are added:

- **Realm (R)**: Indicates realm of user
- **Options (Opt)**: Used to request that certain flags be set in the returned ticket
- **Times (T)**: Used by the client to request the following time settings in the ticket:
    - from: the desired start time for the requested ticket
    - till: the requested expiration time for the requested ticket
    - rtime: requested renew-till time
- **Nonce (N)**: A random value to be repeated in message (2) to assure that the response is fresh and has not been replayed by an opponent

Message (2) returns a ticket-granting ticket, identifying information for the client, and a block encrypted using the encryption key based on the user's password. This block includes the session key to be used between the client and the TGS, times specified in message (1), the nonce from message (1), and TGS identifying information. The ticket itself includes the session key, identifying information for the client, the requested time values, and flags that reflect the status of this ticket and the requested options. These flags introduce significant new functionality to version 5.

Kerberos version 5 – Ticket-granting service exchange



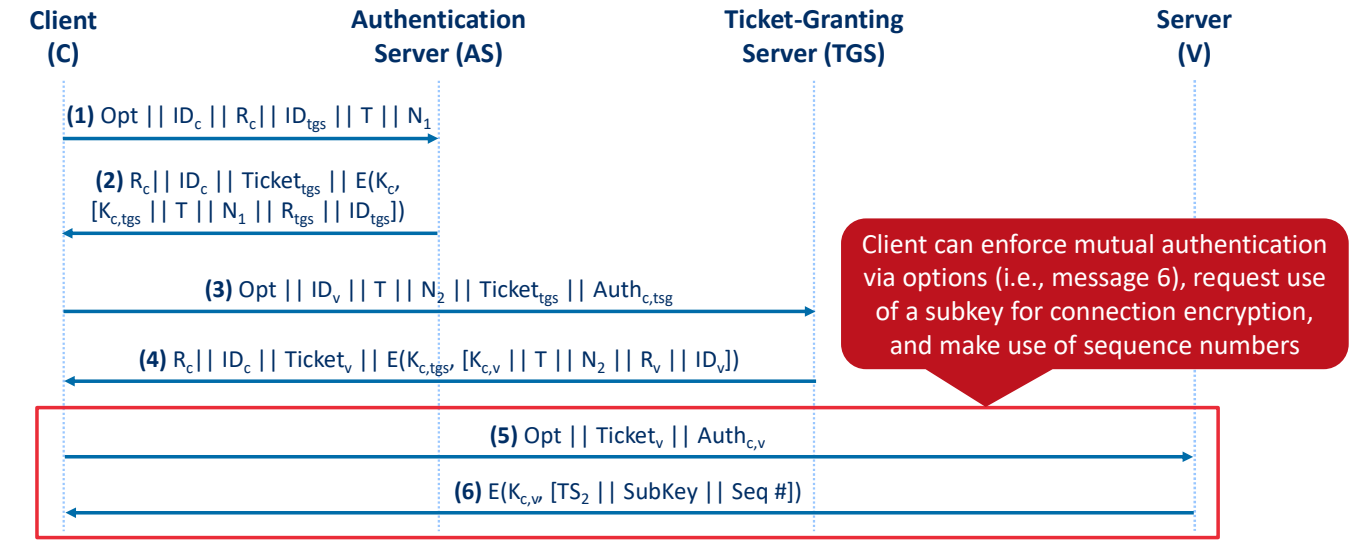
$Ticket_{tgs} = E(K_{tgs}, [Flags || K_{c,tgs} || R_c || ID_c || AD_c || T])$
$Ticket_v = E(K_v, [Flags || K_{c,v} || R_c || ID_c || AD_c || T])$
$Auth_{c,tsg} = E(K_{c,tgs}, [ID_c || R_c || TS_1])$

University of Antwerp | Faculty of Science

71

Let us now compare the ticket-granting service exchange for versions 4 and 5. We see that message (3) for both versions includes an authenticator, a ticket, and the name of the requested service. In addition, version 5 includes requested times and options for the ticket and a nonce—all with functions similar to those of message (1). The authenticator itself is essentially the same as the one used in version 4. Message (4) has the same structure as message (2). It returns a ticket plus information needed by the client, with the information encrypted using the session key now shared by the client and the TGS.

Kerberos version 5 message exchange



$Ticket_{tgs}$ = E(K_{tgs}, [Flags || $K_{c,tgs}$ || R_c || ID_c || AD_c || T])
$Ticket_v$ = E(K_v, [Flags || $K_{c,v}$ || R_c || ID_c || AD_c || T])
$Auth_{c,tsg}$ = E($K_{c,tgs}$, [ID_c || R_c || TS_1])
$Auth_{c,v}$ = E($K_{c,v}$, [ID_c || R_c || TS_2 || SubKey || Seq #])

University of Antwerp | Faculty of Science

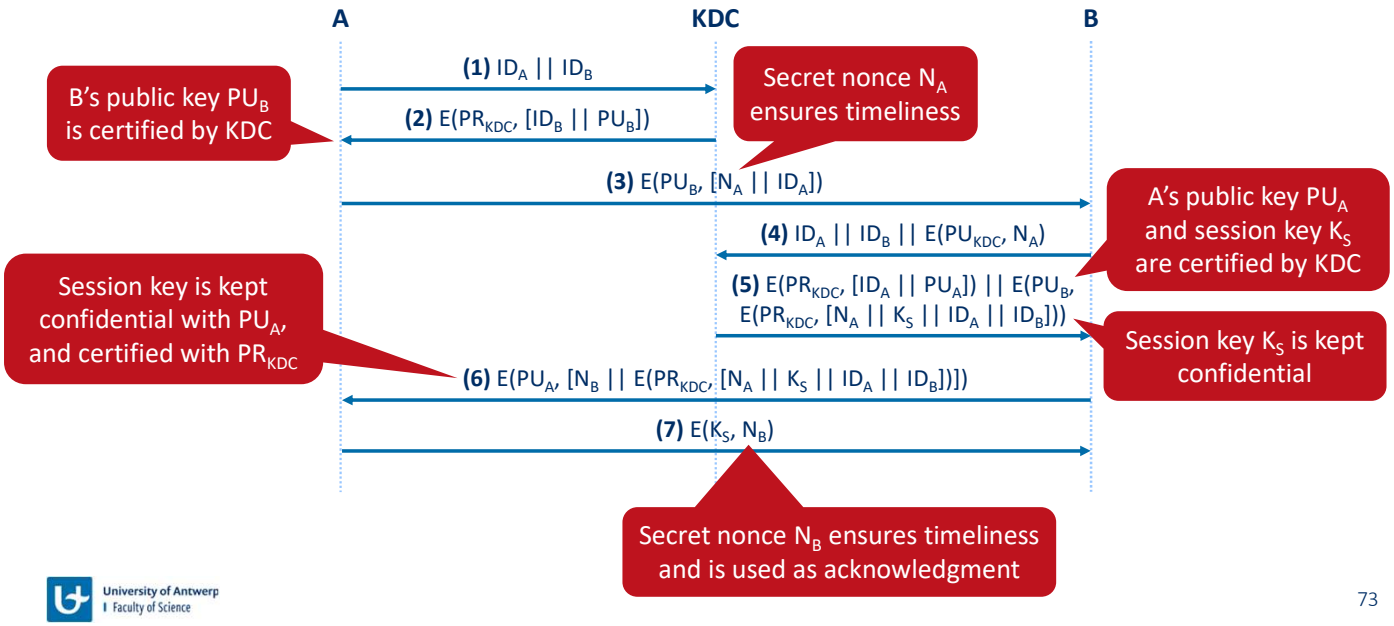
72

Finally, for the client/server authentication exchange, several new features appear in version 5. In message (5), the client may request as an option that mutual authentication is required. The authenticator includes several new fields:

- **Subkey**: The client's choice for an encryption key to be used to protect this specific application session. If this field is omitted, the session key from the ticket ($K_{c,v}$) is used.
- **Sequence number**: An optional field that specifies the starting sequence number to be used by the server for messages sent to the client during this session. Messages may be sequence numbered to detect replays.

If mutual authentication is required, the server responds with message (6). This message includes the timestamp from the authenticator. Note that in version 4, the timestamp was incremented by one. This is not necessary in version 5, because the nature of the format of messages is such that it is not possible for an opponent to create message (6) without knowledge of the appropriate encryption keys. The subkey field, if present, overrides the subkey field, if present, in message (5). The optional sequence number field specifies the starting sequence number to be used by the client.

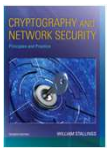
Authentication using public-key certificates



In step 1, A informs the KDC of its intention to establish a secure connection with B. The KDC returns to A a copy of B's public-key certificate (step 2). Using B's public key, A informs B of its desire to communicate and sends a nonce N_A (step 3). In step 4, B asks the KDC for A's public-key certificate and requests a session key; B includes A's nonce so that the KDC can stamp the session key with that nonce. The nonce is protected using the KDC's public key. In step 5, the KDC returns to B a copy of A's public-key certificate, plus the information $\{N_A, K_S, ID_A, ID_B\}$. This information basically says that K_S is a secret key generated by the KDC on behalf of B and tied to N_A; the binding of K_S and N_A will assure A that K_S is fresh. This tuple is encrypted using the KDC's private key to allow B to verify that the tuple is in fact from the KDC. It is also encrypted using B's public key so that no other entity may use the tuple in an attempt to establish a fraudulent connection with A. In step 6, the tuple $\{N_A, K_S, ID_A, ID_B\}$, still encrypted with the KDC's private key, is relayed to A, together with a nonce N_B generated by B. All the foregoing are encrypted using A's public key. A retrieves the session key K_S, uses it to encrypt N_B, and returns it to B. This last message assures B of A's knowledge of the session key.

Summary on user authentication

- **Access control** consists of authentication and authorization
  - Authentication verifies the identity of a system entity
  - Authorization verifies an entity's rights to perform specific actions or access specific resources
- **Mutual authentication** protocols enable communicating parties to agree on each other's identity while securely exchanging secret keys, requiring
  - Confidentiality to avoid masquerade attacks and ensure security of secret keys
  - Timeliness to avoid replay attacks
  - Can be done using symmetric encryption (e.g., Kerberos) or asymmetric encryption (i.e., using public-key certificates)



Link with the book

- Chapter 15 (15.1 – 15.5)

End of Part 5

University of Antwerp
Faculty of Science