The University of Sydney
School of Computer Science
Dr Suranga Seneviratne
Senior Lecturer - Security

THE UNIVERSITY OF SYDNEY

Authentication & Key Distribution

Recommended Reading

Cryptography and Network Security (7^{th} Edition - William Stallings)

- **Chapter 14** - Key Management and Distribution
- **Chapter 15** - User Authentication

Security Engineering (3^{rd} Edition - Ross Anderson)

- **Chapter 4** - Protocols

So far, we discussed how we can use encryption to keep our data and communications confidential. For example, symmetric key cryptography (e.g., AES) and asymmetric key cryptography (e.g., RSA). The commonality is that they all require a key to encrypt information. However, one last factor is missing: how do individuals or entities obtain the keys they need for secure communication?

For example, in symmetric key encryption we assumed that both parties have access to a shared key. In public key encryption, we assumed that the sender knows the receiver's public key. But how does this work in practice? How can we make sure that these assumptions are realistic? This is the key distribution problem.

In the discussions that follow, we always assume the Dolev-Yao model for our attacker. Formerly defining an attacker's capability is called *threat modelling*. Recall that in the cryptography overview, we discussed the types of attackers, defining four varying capabilities of the attacker (e.g., has access to cipher text only, can query and get some cipher text decrypted).

Dolev-Yao Model

The Dolev-Yao model assumes that the attacker has **full control** over the network, with capabilities such as carrying messages, eavesdropping on any communication, and the ability to delay, delete, modify, replay, and inject messages. However, the attacker's ability is **constrained by cryptography**. For example, under this model, the attacker cannot alter an integrity-protected message without the receiver noticing, nor can they decrypt encrypted messages. Intuitively, this represents the type of attacker we typically envision in the context of secure electronic communication.

1 The Problem of Key Distribution

As mentioned earlier, the key distribution problem in security refers to the challenge of securely sharing cryptographic keys among parties involved in communication. In a secure communication system, encryption keys must be distributed to the communicating parties in such a way that unauthorized individuals cannot intercept or compromise these keys. The problem is particularly complex in large networks or open environments where secure communication channels may not be readily available. Boyd's Theorem governs the challenges of key distribution.

Boyd's Theorem

Boyd's theorem says, "Assuming the absence of a secure channel, two entities cannot establish an authenticated session without the existence of an entity that can mediate between the two and which both parties trust and have a secure channel with". In short, Alice and Bob cannot securely establish keys between them if they do not already have existing, established keys. Thus, how do we create a "safe channel" for them to share their keys? The only way is 'introduction' via a third party, with which both Alice and Bob have established keys already.

Why is Diffie-Hellman not enough? Diffie-Hellman key exchange is secure **if the attacker is passive**. In our attack scenario with an **active attacker**, we need to protect the Diffie-Hellman key exchange with some form of origin authentication. For instance: A Message Authentication Code (MAC) requires a previously exchanged symmetric key; a digital signature requires a known public key and correct mapping between the entity/origin and the key. This implies that Diffie-Hellman can still be utilized, but it necessitates additional protections. Importantly, Diffie-Hellman provides **forward secrecy**, meaning that once the exchange is securely completed, a secure communication channel is established. Forward secrecy **protects past sessions against future compromises of keys or passwords**. i.e., The session keys used to encrypt and decrypt information change frequently and automatically. This ongoing process ensures that even if the most recent key or the long-term key is breached, a minimal amount of sensitive data is exposed.

Forward Secrecy/Perfect Forward Secrecy

Forward Secrecy (FS)/Perfect Forward Secrecy (PFS) is a property of secure communication protocols that ensures that the compromise of long-term keys does not compromise the confidentiality of past session keys. In simpler terms, even if an attacker gains access to a server's private key or other long-term credentials, they should not be able to decrypt past communications. This is achieved by generating unique session keys for each communication session, which are independent of the long-term keys.

Forward Secrecy (FS) and Perfect Forward Secrecy (PFS) are used interchangeably and mean the same thing.

Ephemeral Diffie-Hellman (DHE) or **Ephemeral Elliptic-Curve Diffie-Hellman (ECDHE)** refer to the use of Diffie-Hellman key exchange to establish ephemeral keys - meaning that for each session, a new, temporary key is generated. This key is used exclusively for that session and is discarded afterwards. Because each session uses a unique and independent key, even if the long-term private key used to authenticate the session (such as a key used in a digital signature) is later compromised, the attacker cannot recover the session keys or decrypt past communications. This property is what ensures forward secrecy, as it protects the confidentiality of past sessions even if long-term credentials are compromised in the future.

Options for key Establishment/Distribution

- **Without a third party**: Secure physical shipment of keys is indeed a method, but it's not practical. Thus, it is only done for high-value communication - e.g., with embassies, some devie parings with QR codes.
- **With a third party**: In symmetric case, Key Distribution Centers (KDCs) facilitate the exchange. In asymmetric case, Public Key Infrastructures (PKI) play the crucial role.

2 Symmetric Key Distribution (Using Symmetric Encryption)

For symmetric encryption to work, the two parties 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. Following Byod's theory, this means that you need a trusted third party.

2.1 Key Hierarchy

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 (Figure 1). 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 a transport layer 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. Master keys can be distributed in some non-cryptographic way, such as physical delivery. For example, in the case of the KDC we will discuss later, the keys may have been physically copied by your IT department when they installed and configured your laptop.

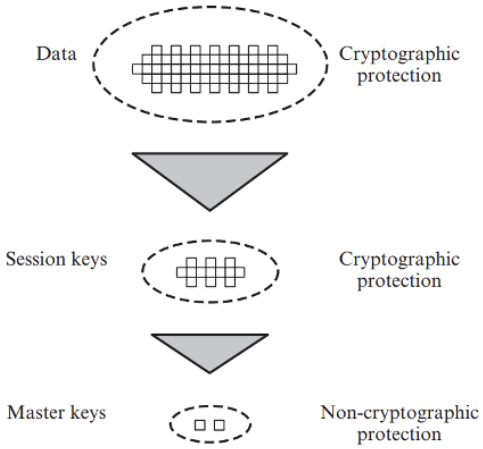

Figure 1: The Use of a Key Hierarchy
Source: Cryptography and Network Security (Seventh Edition - William Stallings)

2.2 Key Distribution Centers (KDCs)

KDCs are among the **most important technologies** we use. Even new technologies like OpenID and OAuth fall into this category. KDCs are the basis for the famous Kerberos protocol, which is utilized across numerous operating systems, including Windows, Linux, UNIX, macOS, and even AIX

and Solaris.

A KDC generates and distributes session keys. They are always online and have long-term keys established with their members. These keys can be pre-configured by admins or derived from a strong password in practice.

How a KDC works - Entity A sends a request to the KDC for a symmetric key to be used as a session key for communication with B. The KDC generates a symmetric session key and then encrypts this session key with the master key it shares with A, sending it to A. Similarly, the KDC encrypts the session key with the master key it shares with B and sends it to B. Alternatively, the KDC can send both encrypted key values to A, who then forwards the session key, encrypted with the master key shared by the KDC and B, to B.

2.3 A Key Distribution Scenario

The above idea is further illustrated in Figure 2. The scenario assumes that each user shares a **unique master key** with the key distribution center (KDC).

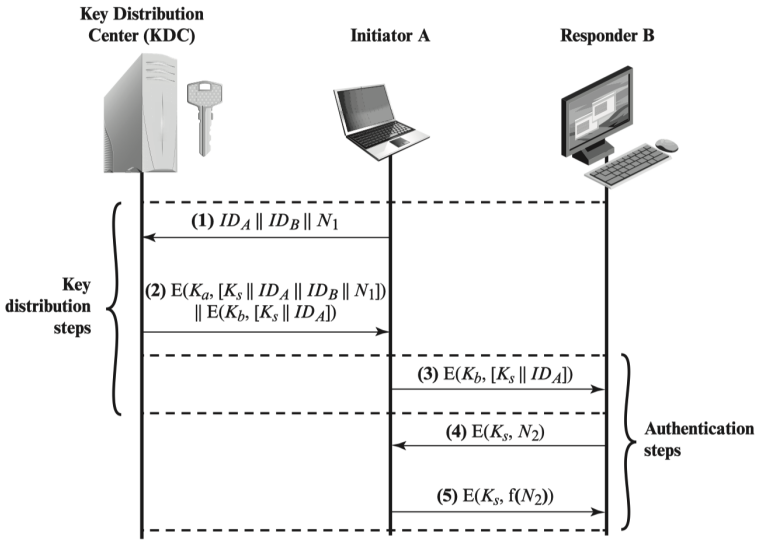

Figure 2: Key Distribution Scenario
Source: Cryptography and Network Security (Seventh Edition - William Stallings)

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, K_a , 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 and also ensures the freshness.

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.

3. 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.

3 Symmetric Key Distribution (Using Asymmetric Encryption)

Because of the inefficiency of public-key cryptosystems, they are almost never used for the direct encryption of sizable blocks of data, but are limited to relatively small blocks. One of the most important uses of a public-key cryptosystem is to encrypt secret keys for distribution.

3.1 Person-In-The-Middle Attack

Any attempt to distribute public keys naively can result in a Person-in-the-Middle attack, as illustrated in Figure 3.

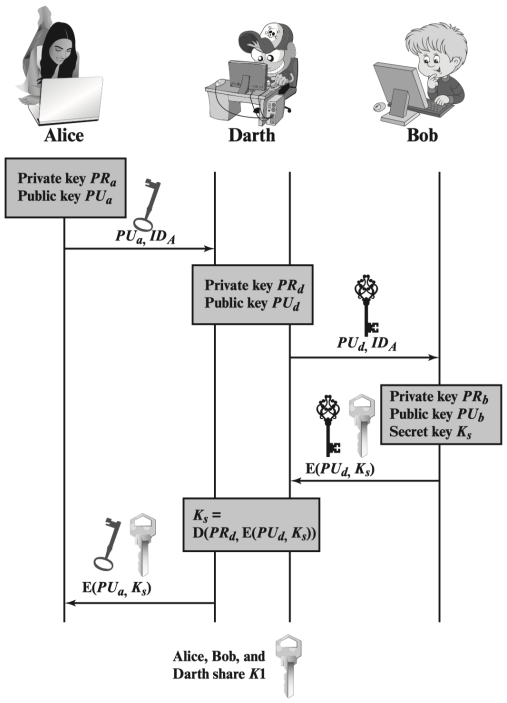

Figure 3: Person-in-the-Middle Attack
Source: Cryptography and Network Security (Seventh Edition - William Stallings)

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 $\mathrm{E}(PU_d, K_s)$.
4. D intercepts the message and learns K_s by computing $\mathrm{D}(PR_d, \mathrm{E}(PU_d, K_s))$.
5. D transmits $\mathrm{E}(PU_a, K_s)$ 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.

3.2 Symmetric Key Distribution with Confidentiality and Authentication

The solution for PITM attack is to provide protection against both active and passive attacks – this scenario – we are using nonce. We begin at a point when it is assumed that A and B have exchanged public keys by one of the schemes described subsequently in this chapter. Then the following steps occur (Figure 4).

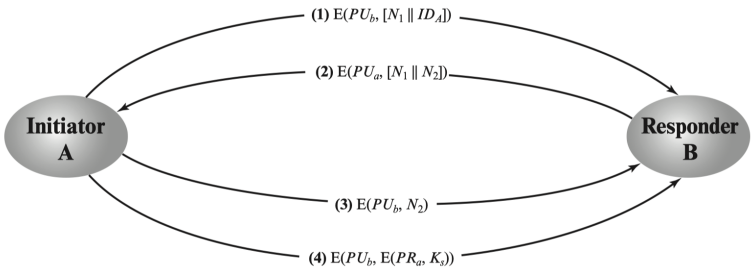

Figure 4: Public-Key Distribution of Secret Keys
Source: Cryptography and Network Security (Seventh Edition - William Stallings)

1. A uses B's public key 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.
2. 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 (1), the presence of N_1 in message (2) assures A that the correspondent is B.
3. A returns N_2, encrypted using B's public key, to assure B that its correspondent is A.
4. A selects a secret key K_s and sends $M = \text{E}(PU_b, \text{E}(PR_a, K_s))$ to B. Encryption of this message with B's public key ensures that only B can read it; encryption with A's private key ensures that only A could have sent it.
5. B computes $\text{D}(PU_a, \text{D}(PR_b, M))$ to recover the secret key.

The result is that this scheme ensures both confidentiality and authentication in the exchange of a secret key.

A Hybrid Scheme Another way to use public-key encryption to distribute secret keys is a hybrid approach in use on IBM mainframes. This scheme retains the use of a key distribution center (KDC) that shares a secret master key with each user and distributes secret session keys encrypted with the master key. A public-key scheme is used to distribute the master keys. The addition of a public-key layer provides a secure, efficient means of distributing master keys. This is an advantage in a configuration in which a single KDC serves a widely distributed set of users.

4 Public-Key Distribution and Certificates

4.1 Public-Key Distribution Scheme

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**: 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 their public key to any other participant or broadcast the key to the community at large. For example, because of the growing popularity of PGP (pretty good privacy), 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.

- **Publicly available directory**: 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 need the authority, which maintains a directory with a name, public key entry for each participant. 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 based on the Needham-Schroeder protocol. Public-key authority scheme 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.
- **Public-key certificates (Best option)**: 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. The central idea is using **certificates** used in Public Key Infrastructures (PKIs).

4.2 Certificate Creation

In essence, a certificate consists of a public key (PU), an identifier (ID) of the key owner, and the whole block signed by a trusted third party. Typically, the third party is a certificate authority (CA), such as a government agency or a financial institution, that is trusted by the user community.

A user can present their 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.

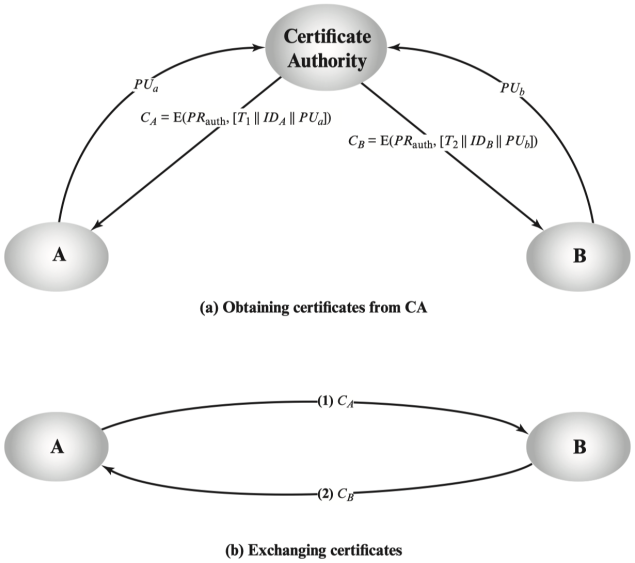

Figure 5: Exchange of Public-Key Certificates
Source: Cryptography and Network Security (Seventh Edition - William Stallings)

A certificate scheme is illustrated in Figure 5. 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 $C_A = \mathrm{E}(PR_{\mathrm{auth}}, [T \| ID_A \| PU_a])$, 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: $\mathrm{D}(PU_{\mathrm{auth}}, C_A) = \mathrm{D}(PU_{\mathrm{auth}}, \mathrm{E}(PR_{\mathrm{auth}}, [T \| ID_A \| PU_a])) = (T \| ID_A \| PU_a)$. 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 is used to verify whether the certificate is still valid.

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.

4.3 X.509 Certificate

Definition of a certificate A certificate is a cryptographic binding between an identifier and a public key that is to be associated with that identifier. An issuer creates the binding and takes the role of a trusted third party (TTP), in accordance with Boyd's theorem.

X.509 One scheme has become universally accepted for formatting public-key certificates: the X.509 standard. X.509 certificates are used in most network security applications, including IP security,

transport layer security (TLS), and S/MIME. 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. Figure 6 illustrates the overall X.509 scheme for generation of a public-key certificate.

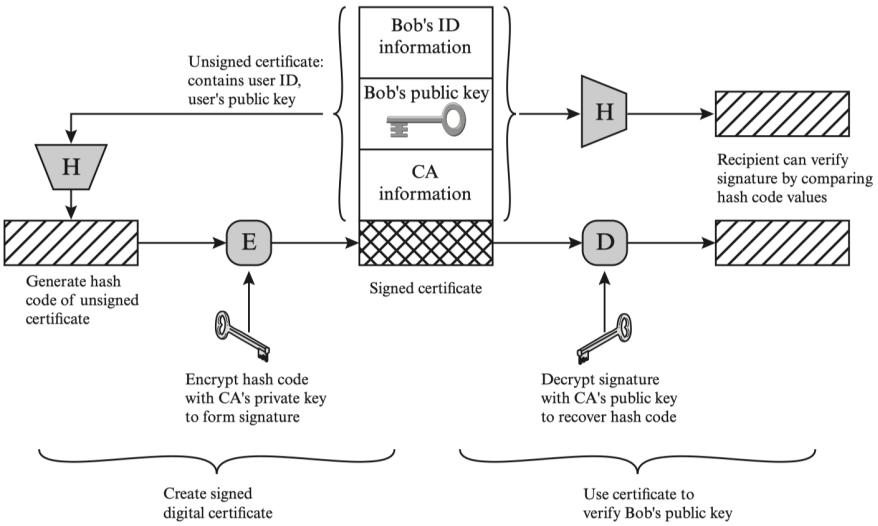

Figure 6: X.509 Public-Key Certificate Use
Source: Cryptography and Network Security (Seventh Edition - William Stallings)

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. The current version of X.509 does not dictate a specific digital signature algorithm.

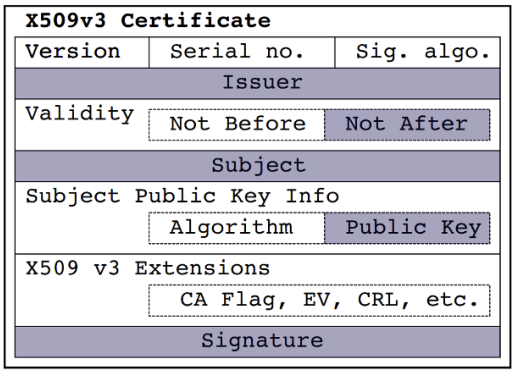

Figure 7: Example of X.509 Certificate Version 3

5 Public Key Infrastructures (PKIs)

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.

Let's say Bob wants to securely encrypt messages for Alice. To do this, Bob needs to know Alice's public key. Similarly, Alice needs Bob's public key to verify any digital signatures from him. This problem is addressed by using Public Key Infrastructures (PKI).

PKI revolves around the use of certificates. As mentioned before, a certificate binds an entity's identity to their public key, and a trusted authority digitally signs this binding to ensure its validity.

However, in practice, Bob may not have a certificate that Alice can verify directly. Instead, Bob presents a **certificate chain**, which allows Alice to trace the trust back to a common trusted entity.

For example, imagine there is a certificate authority (CA) I_1 that issues a certificate to I_2, a subordinate CA. I_2 then issues a certificate to Bob, represented as X. In this case, the chain of trust is: $I_1 \rightarrow I_2 \rightarrow X$. Each arrow represents a certificate issued by the previous entity. Alice can verify Bob's certificate X by following this chain of trust from X, through I_2, up to I_1, which she trusts.

This process is known as **certificate chaining**, where each link in the chain establishes trust, and the top-most certificate authority (CA) is trusted by all parties involved.

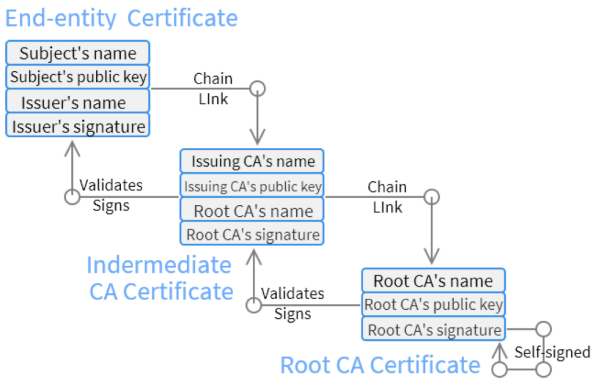

Figure 8: An Example Certification Chain

5.1 Identity Verification

You would need some formal proof to verify your identity, raising the question of which verification process is universally acceptable. The issuance process embeds a critical step. Recall that the issuer, or Certificate Authority (CA), is a trusted third party, such as a government agency or financial institution, that the user community trusts. Before issuing a certificate, the issuer must ensure that they are issuing it to the correct identity and that the identity indeed holds the particular key.

There are industry-agreed steps that must be carried out, which vary depending on purpose of certificate, and its value. Baseline verification, often used for domains like 'suranga.me', involves proof of ownership where the CA requests the domain operator to place a specific file on the web server or to be able to receive an email under that domain name. It's noteworthy that this method relies on technologies that might be insecure for identity verification. On the other hand, extended verification involves the use of legal documents. Though this is a more expensive approach, it is still implemented due to its thoroughness in confirming an entity's legitimacy.

5.2 Hierarchical PKIs

In all variants of hierarchical PKIs, the issuer is called 'Certificate Authority' (CA). CAs are responsible for issuing certificates. The naive idea is that we can have one global issuer trusted by everyone shown in Figure 9. However, this is not that practical. What if there are too many requests?

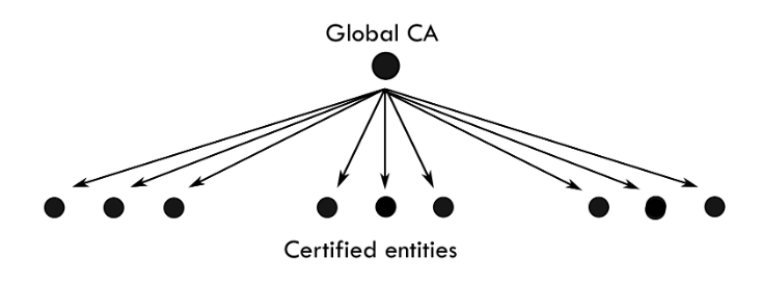

Figure 9: Centralized Certification Authority Model

There are other reasons. First, determining which global authority is deemed trustworthy for the role poses a significant challenge. Moreover, the verification steps must be universally agreed upon, complicating the process further. The issue of namespace being global, such as distinguishing between individuals with common names like "John Smith," necessitates unique global identifiers akin to passports. Therefore, relying on a single certificate authority is impractical.

5.3 Pragmatic 'Solution' to the Problem of Global CA

The solution involves allowing multiple CAs and accepting them equally. However, this introduces a new challenge: how to establish user trust in these CAs. The approach is to include the public keys of 'trusted' CAs with the operating system and/or browser. Consequently, users don't have the option to select which CAs they trust, as the nature of the verification process is determined by agreements between vendors and CAs.

5.4 Root Stores

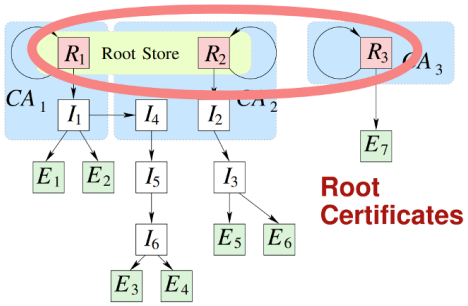

Figure 10: Trusted CAs are configured as trusted in 'root stores'

This certificate shown in Figure 10 is called the root certificate store. For example, your browser – Firefox or Chrome - will have a setup of root certificates called the root certificate store.

Root stores hold certificates from **trusted CAs**, which are considered 'trusted' to issue certificates to legitimate entities. Applications utilizing X.509 certificates must have a root store. Operating systems such as Windows, Apple, and Linux all have their own root stores, as do browsers.

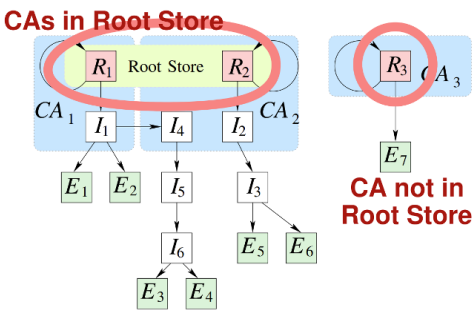

Figure 11: Not all CAs are in root stores

Figure 12 shows the famous cert data message. The reason why you received this is that the cert is not signed by any trusted cert in your root store. i.e., in this case, R_3 is not in your root certificate store, and therefore, you can't verify any certificates whose chain of trust ends with R_3.

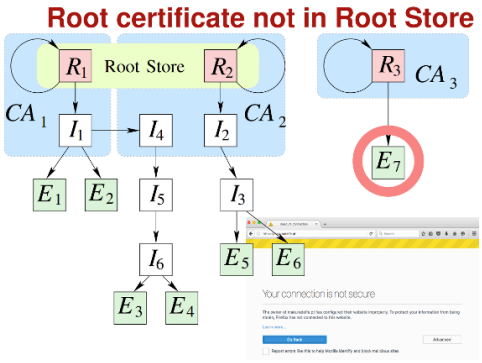

Figure 12: One source of WWW errors

5.5 Intermediate Certificates

Certificates signed by the root certificate are called intermediate certificates (Figure 13). Most websites typically possess certificates that are signed by these intermediate certificates, not directly by the root certificate itself.

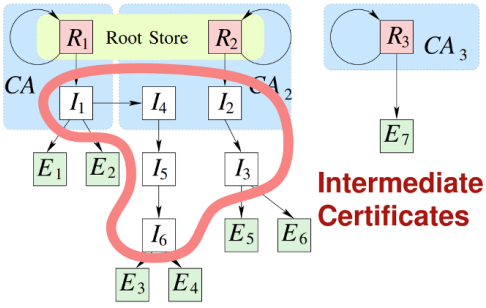

Figure 13: Intermediate Certificates

Intermediate certificates serve primarily for security purposes and are part of a certificate chain, positioned between the root certificate and the end-entity certificate. They are not root certificates themselves, nor are they the final certificates used by websites (end-entity certificates).

There are two primary reasons for using intermediate certificates:

1. To delegate signing authority to a another organization, creating what is known as a sub-CA.
2. To protect the main root certificate: The intermediate certificate is operated by the same organization as the root. The root certificate's private key can be kept offline in a secure location to reduce the risk of compromise. Online day-to-day operations, such as issuing new certificates, are conducted using the private key of the intermediate certificate, which mitigates the risk if the intermediate certificate's key is compromised.

5.6 PKI X.509 model

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 standard defines certificate format, certificate revocation lists (CRLs) and hierarchical PKI with certificate chains. The Internet Engineering Task Force (IETF) Public Key Infrastructure X.509 (PKIX) working group has proposed the PKI X.509 (called PKIX) model.

5.7 Certificate Revocation

It's important to be cautious with certificates. Note that a certificate is only statement that binding between identity and key was valid at time of issuing. There is always the possibility that the key could become compromised later. To ensure that the binding remains valid at the time of verification, it is necessary to perform a revocation check. This process typically involves checking a Certificate Revocation List (CRL) or using the Online Certificate Status Protocol (OCSP) to confirm that the certificate has not been revoked and is still trustworthy for use.

Certificate Revocation Lists (CRLs) CRL is a list of certificates that are considered revoked. CRL should be issued and updated and maintained by every CA. Certificates are identified by serial number and every CRL must be timestamped and signed. Technically, a browser (client) should download CRL (and update it after the given time), and lookup a host certificate every time it connects to a server. However, this is too slow and not practical.

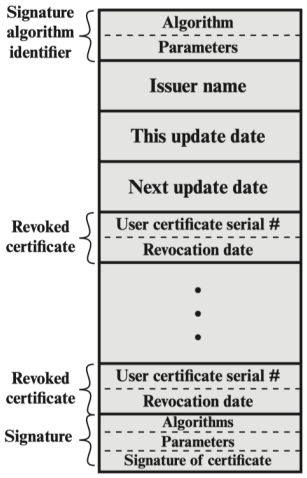

Figure 14: Certificate Revocation List

Online Certificate Status Protocol (OCSP) The current method for checking the revocation status of a certificate includes using the Online Certificate Status Protocol (OCSP). OCSP allows live

revocation checks over the network, which is a query-response model. The CRL is hosted server-side rather than on the local host. Query = lookup of a certificate in a server-side CRL-like data structure. Response contains cert status that must be signed.

5.8 Problem of X.509

There have been numerous known cases where tricksters have managed to deceive CAs into issuing certificates for domains they did not own by presenting a forged identity. Additionally, there have been instances where CAs themselves were compromised.

Empirical research over the past decade has revealed that the deployment of X.509 certificates provides adequate security predominantly for top, high-value websites. The research also points out that there are very lax certification practices in place, as well as careless operational practices, such as including incorrect domain names in certificates. In response to these challenges, many helper technologies have been developed since.

6 Authentication

Authentication is indeed one of the key security goals and serves as a preliminary step before access control can be enforced, when discussing users and privileges. To interact with the filesystem, a user must first log in; this login process is a form of authentication. Successful authentication then leads to authorization, which grants the user permission to access certain resources or perform specific actions. What authentication actually is?

Definition by Menezes et al. Based on a definition by Menezes et al. "*Authentication is the process whereby one party is assured (through the acquisition of corroborative evidence) of the identity of a second party involved in a protocol, and that the second has actually participated.*" Note the elements needed to ascertain the identity: Corroborative evidence; Process between (at least) two parties; Involvement and participation of the second party

6.1 Corroborative Evidence

Corroborative evidence means the factors that are unique to an entity. We classify them into 3 categories:

1. **Possession**: Something the entity/user has. For example, Physical key, phone to send SMS messages to. Possession sounds like a good way to authenticate an entity or user, however, the common problem is the loss of the physical device. For instance, when using a phone for authentication, the device is typically identified by its phone number. However, SMS messages used for authentication can be rerouted due to vulnerabilities in the global signaling network, SS7, which handles SMS among other services. These vulnerabilities mean that simply possessing a device may not provide sufficient assurance to authenticate an entity or user reliably.

2. **Inherence**: Something the user is. For instance, Biometrics: fingerprints, iris scan, face and voice recognition. People once believed that biometrics were secure because, generally, biometric data cannot be changed. However, biometric data can be easily inferred or duplicated. For example, it is possible to capture a fingerprint using a standard camera, an attack method that has been successfully demonstrated (Figure 15). Biometric authentication methods are theoretically more susceptible to forgery through recordings, as technology evolves to better capture and reproduce biometric data. The major risk with biometrics is that unlike a password or a phone number, you cannot simply change your fingerprint or other biometric identifiers if they

are compromised. This permanence makes biometrics a potentially vulnerable authentication method if the data is intercepted or replicated.

Figure 15: Example for capturing a fingerprint

3. **Knowledge**: Something the user knows. Such as Passwords, security questions (mother's maiden name, etc.) These can be used stand-alone or in combination. We already discussed the usability and strength of passwords and databases of cracked passwords exist. The security questions used for account recovery are often very standard and predictable, offering limited options that can be easily guessed. Common questions like "What is your mother's maiden name?" can be vulnerable to attacks, especially in certain cultures where common surnames like Smith, Chang, Kim, or Schmidt may apply. Similarly, answers to questions like "What was your first car?" could be guessed with popular choices such as Golf, Yaris, or Corolla. Additionally, social networks can provide a rich source of personal information that can be used to answer these security questions, further compromising the effectiveness of this security measure.

6.2 Multi-factor Authentication

Requiring multiple factors for authentication enhances security by combining information from different categories, such as knowledge (something you know) and inherence (something you are). A specific example of this is two-factor authentication (2FA), which typically involves a password combined with a code sent via text to a phone. This setup significantly increases security, requiring a highly motivated attacker and typically thwarting broad, non-targeted attacks.

However, the choice of factors should be carefully considered based on the context and specific security needs. For example, while possession (something you have, like a phone) is a strong factor due to its physical nature, there's also the risk of loss. In scenarios where a user might lose their phone, this could compromise their ability to authenticate. Therefore, it's crucial to select authentication factors that balance security with practical considerations like user behavior and potential risks.

However, we are still implementing multi-factor authentication (MFA), but of course, with proper setting and implementation. 2FA (Two-Factor Authentication) has become standard today for high-value sites such as webmail providers and social networks.

MFA often includes mechanisms that check for plausibility. For example, if a login request comes from an unusual country, one that the user has never logged in from before, or if there are multiple failed login attempts, or if the IP address is from a range known to be the origin of other attacks, then MFA might be triggered or additional authentication factors may be required. For instance, the user might need to enter their password, receive a code on their phone, and get an access token sent via email.

Additionally, 2FA must follow the principle of psychological acceptability, ensuring that the authentication process does not overly burden or frustrate the user, which is crucial for maintaining user compliance and overall security integrity.

Multi-factor authentication (MFA) has several limitations:

- MFA places a bigger onus on the user. Remember psychological acceptability! Attacker can trick user to send their OTP and so on, he/she can bypass multi-factor authentication. Generally, MFA is more readily accepted in high-value, high-risk scenarios.
- MFA with factors from the same category is not guaranteed stronger. For instance, admin asks for two passwords (which may both be weak) – this leads to no change at all to the security.
- Social engineering can still easily overcome multiple factors. Tactics like phone scams can still trick users into disclosing crucial security information, such as passwords and codes.

6.3 One-Time Passwords

Strengthening the authentication process can be effectively achieved using One-Time Passwords (OTPs), which are valid for only one login session. This method prevents attackers from reusing captured passwords. Examples of systems using dynamic OTPs include Google Authenticator, Microsoft OTP, and Yubikey.

There are two types of OTPs:

1. Static OTP: An indexed list of OTPs is given to user
2. Dynamic OTP: Follows on a challenge-response principle where the OTP generated depends on the specific challenge presented

OTP serves as a form of possession and can be done as MFA. For instance, when making a high-value purchase, a bank might require an OTP before processing the payment. This OTP might be generated by a device the user possesses or in response to a challenge code provided by a website, ensuring the OTP is context-specific and secure.

6.4 Console Login vs. Network Login

Console login or network login is an example of a **process between (at least) two parties**. The principle of participation is guaranteed by the requirement that someone or something must physically input the password, typically via a keyboard. This interaction guarantees active participation in the authentication process. However, it's still possible to construct a device that attempts to crack UNIX passwords by plugging it into a USB port. To counteract such threats, UNIX systems implement a security measure that introduces a slight delay between password attempts, helping to defend against rapid, automated password-guessing attacks.

6.5 Authentication Protocols

Network communication heavily relies on protocols, which dictate how layers interact with each other. Previously, we applied cryptography primarily to single messages. However, for effective authentication over a network, a more robust approach is necessary, involving a sequence of steps. This sequence forms a protocol, and when cryptographic protection is incorporated, it becomes a cryptographic protocol. Creating secure cryptographic protocols is challenging. Even with flawless cryptographic operations, vulnerabilities can still exist due to loopholes in protocol design. For example, an attacker could intercept a password and reuse it in subsequent requests, known as a replay attack, where even an encrypted password could be replayed.

6.6 Person-in-the-Middle

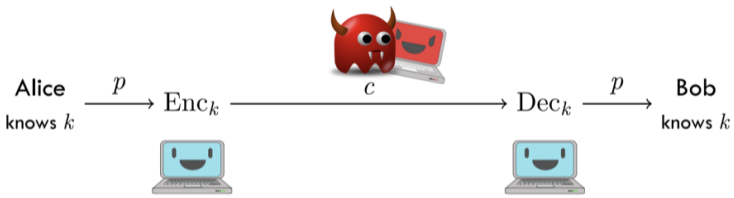

Figure 16: MITM Attack

Alice tries to send password p to Bob, encrypted with k. Attacker can just record $c = Enc_k(p)$ and replay it later. Bob does not actually know if Alice has participated. Neither MACs nor signatures help since they can be replayed as well. We must somehow bind the transmission of the password to the **current communication**.

To find somehow bind the transmission of the password to the current communication, we need to design or **build a protocols**. We first construct cryptographic protocols from primitives:

- Cryptographically secure random numbers, which is essential ingredient in all that follows
- Cryptographic hash functions give us MACs (by mixing in shared secret)
- Public-key cryptography gives us key exchange over certain untrusted channels, confidentiality (typically achieved through hybrid cryptography, where symmetric keys are encrypted with a public key.), and origin authentication and data integrity
- Symmetric-key cryptography gives us confidentiality (secures actual data transfers by encrypting the data), and origin authentication and data integrity achieved using MACs

7 Authentication and Key Establishment

7.1 Constructing an Authentication Protocol

Alice (A) wants to authenticate to Bob (B). A must prove to B possession or knowledge of something that no one but A can have. We will assume they have a shared, secret key $k_{A,B}$. Alice will try to send an encrypted password $c = Enc_k(p)$. Returning to our broken example. C is Clare – attacker here, who can perform replay attack. $A \to B : c, MAC_{k_{A,B}}(c)$ - A even adds a MAC. It's not secure yet. $C \to B : c, MAC_{k_{A,B}}(c)$.

Challenge-Response with Nonces In the following, let m denote an encrypted and integrity-protected message. A nonce is a 'random number used once'. We play $A \to B : N_A$, $B \to A : \{N_A, N_B\}$, $A \to B : \{N_A, N_B + 1\}$. B has assurance now that A has participated and reacted to precisely the second message. Because she was able to read N_B (the challenge) and increment it (response). Note: nonces can be replaced with timestamps in some cases, but we do not show this here.

7.2 AKE: Authentication and Key Establishment

Most protocols combine authentication and key establishment, typically through one of two methods: either by creating a password and securely encrypting and sending it, or by utilizing a Diffie-Hellman key exchange.

There are critical engineering principles to ensure robust security: Always generate 'session keys' from your actual, master keys.; Use the session keys for one communication 'session'; then delete them; Never reuse session keys; Both parties should be involved in the key creation.

The idea of a **simple AKE** is to use long-term asymmetric keys to protect short-term symmetric keys. In the simplest hybrid crypto approach, Alice generates symmetric key k:

$$A \to B : Enc_{k_e,B}(k), Sig_A(k), \text{ use } k \text{ for further encryption}$$

Better schema for hybrid crypto:

$$\begin{gathered} A \to B : Enc_{k_e,B}(k_A), Sig_A(k_A) \\ B \to A : Enc_{k_e,A}(k_B), Sig_B(k_B) \\ \text{Then use } k = f(k_A || k_B) \text{ for encryption} \end{gathered}$$

Now both sides are involved in key creation. Even if one chooses a poor key half by accident, the scheme remains secure. Variants of this are in use, but are being phased out.

7.3 AKE with Diffie-Hellman

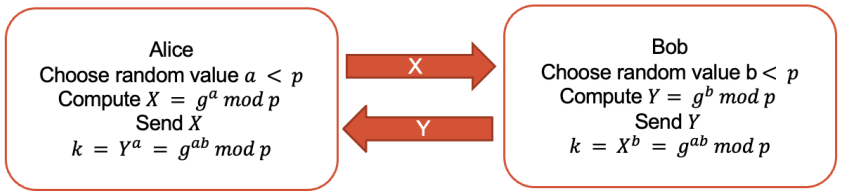

Figure 17: Diffie-Hellman Recap

In practice, we sign Diffie-Hellman values (Figure 18). Alice and Bob did all the steps the same, except that when they send X and Y, they also send them with $sig_A(x)$ and $sign_B(y)$. This protects against an active attacker trying to modify them. Alice and Bob use their (long-term secret) private keys for that e.g. RSA keys.

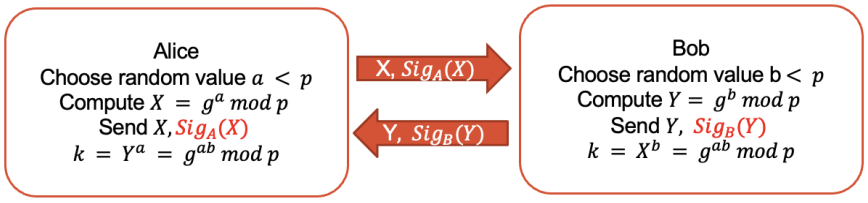

Figure 18: Signing Diffie-Hellman Values

Forward Secrecy The scheme shown in Figure 18 is called forward secrecy. Even if an attacker obtains the long-term signing keys, messages encrypted with session keys will remain secure. When attacker learns one pair (a_i, b_i), all messages protected with $(a_j, b_j), i \neq j$ are still secure! Using Diffie-Hellman every time we run a protocol gives us fresh session keys every time ('ephemeral' keys). But must choose fresh a, b every time and delete old ones!

7.4 Modern APIs

In cryptography and protocols, there are too many ways to make mistakes, ranging from the use of inadequate key lengths and poor random value generation to improper padding or block modes, along with a myriad of other pitfalls. Modern APIs limit developers' choices by defaulting to safe options and offering only those. What modern APIs cannot prevent are semantic flaws in the protocol. This is why we use standardized protocols that have been thoroughly vetted, such as the Kerberos protocol.

8 Kerberos

8.1 Needham-Schroeder protocol

The basic idea of Needham-Schroeder protocol involves an intermediary 'Authentication Server' that facilitates communication between two parties, such as Alice and Bob. This protocol is designed to establish a session key between two users over an insecure network. There are several variants of this protocol, with Kerberos being one of the most well-known examples. The version we will focus on specifically utilizes symmetric keys. Essentially, the Needham-Schroeder protocol serves as a method for constructing a Key Distribution Centre (KDC), which is crucial for managing the distribution of keys in a secure manner.

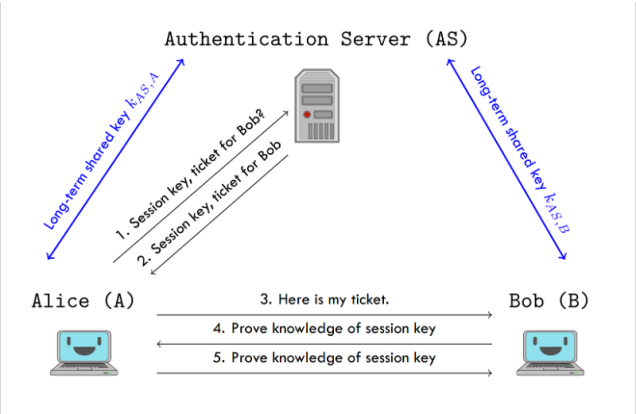

Figure 19: Needham-Schroeder Protocol Basic Idea

In Figure 20, initiate; nonce identifies the session:

$$A \rightarrow AS : ID_A, ID_B, N_1$$

AS creates session key K_s; encrypts with shared with Bob ('ticket'):

$$AS \rightarrow A : E(K_a[K_s||ID_A||ID_B||N_1])||E(K_b[K_s||ID_A])$$

Forward the ticket:

$$A \rightarrow B : E(K_b[Ks||ID_A])$$

Prove knowledge of session key (freshness!): Forward the ticket:

$$B \rightarrow A : E(K_s, N_2)$$

Prove knowledge of session key and nonce, f is a generic function that modifies the value of nonce:

$$A \rightarrow B : E(K_s, f(N_2))$$

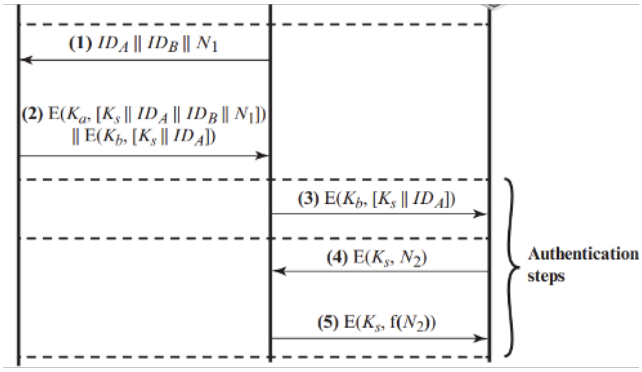

Figure 20: Needham-Schroeder Protocol

8.2 Kerberos protocol

Kerberos is a widely-used protocol for authentication and access control. Instead of implementing complex authentication protocols on each server, Kerberos centralizes this process with an authentication server. This server's primary function is to authenticate users to servers and vice versa, which is similar to the function of the Needham-Schroeder protocol.

Kerberos is built on Needham-Schroeder, and has remarkable features. It uses a ticket concept that allows users to access services securely. Users first authenticate with a password and then request access to a server. To enhance security, these tickets are assigned a lifetime, enforced by timestamps, after which reauthentication is necessary. Additionally, Kerberos has the capability to federate several administrative domains, making it versatile for use across different organizational structures.

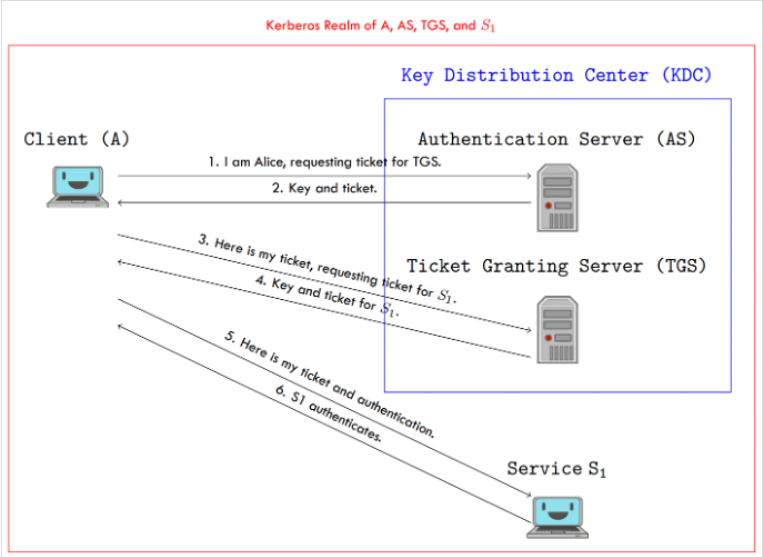

Figure 21: Kerberos Overview

Figure 21 shows an overview of Kerberos. To address the issue of ticket-granting tickets being captured

and misused before their expiration, a strategy is employed where the Authentication Server (AS) provides both the client and the Ticket Granting Server (TGS) with a secret piece of information securely. This method ensures that even if a ticket is stolen, the thief cannot use it without also having access to the secret information. The client then verifies its identity to the TGS by revealing this secret, doing so in a secure manner. A practical and efficient approach to managing this secret information is to use an encryption key, known as a session key in the context of Kerberos. This session key serves as both the secret and the means of secure communication, effectively safeguarding the integrity of the authentication process.

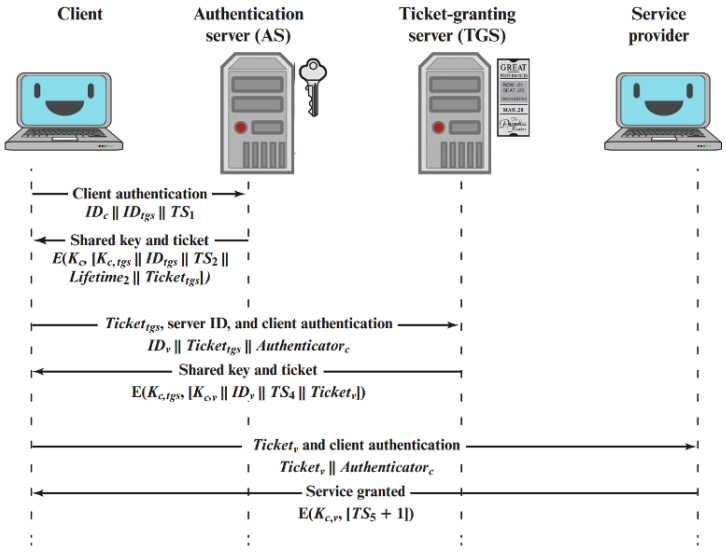

Figure 22: Kerberos Exchanges Among the Parties

Figure 22 shows authentication service exchange:

1. Client requests ticket-granting ticket:

$$Client(C) \rightarrow AS : ID_c||ID_{tgs}||TS_1$$

ID_c: client identity
ID_{tgs}L: tells AS that user requests access to TGS
TS_1: allows AS to verify that client's clock is synchronized with that of AS

2. AS returns ticket-granting ticket:

$$AS \rightarrow C : E(K_c, [K_{c,tgs}||ID_{tgs}||TS_2||Lifetime_2||Ticket_{tgs}])$$

K_c: encryption is based on user's password, enabling AS and client to verify password, and protecting contents of message
$K_{c,tgs}$: copy of session key accessible to client created by AS to permit secure exchange between client and TGS without requiring them to share a permanent key
ID_{tgs}: confirms that this ticket is for the TGS
TS_2: informs client of time this ticket was issued
$Lifetime_2$: informs client of the lifetime of this ticket
$Ticket_{tgs}$: ticket to be used by client to access TGS

3. Client requests service-granting ticket:

$$C \rightarrow TGS : ID_v||Ticket_{tgs}||Authenticator_c$$

ID_v: tells TGS that user requests access to server V
$Ticket_{tgs}$: assures TGS that this user has been authenticated by AS
$Authenticator_c$: generated by client to validate ticket

4. TGS returns service-granting ticket:

$$TGS \rightarrow C : E(K_{c,tgs}, [K_{c,v}||ID_v||TS_4||Ticket_v])$$

$K_{c,tgs}$: key shared only by client and TGS to protect content of message
$K_{c,v}$: session key for client and server V
ID_v: confirm ticket is for server V
$Ticket_v$: ticket to access server V

5. Client requests service:

$$C \rightarrow S_v : Ticket_v||Authenticator_c$$

$Ticket_v$: assures server that this user has been authenticated by AS

6. Optional authentication of server to client, service granted:

$$S_v \rightarrow C : E(K_{c,v}, [TS_5 + 1])$$

$K_{c,v}$: assures client that this message is from V
$TS_5 + 1$: assures C that this is not a replay of an old reply

9 Practice Quiz

Question 1: ..... refers to the means of delivering a key to two parties who wish to exchange data without allowing others to see the key.

a) Manual key delivery
b) Key distribution technique
c) Session key delivery
d) Confidential key distribution

Explanation*: This is the textbook definition of key distribution. We discussed there are two ways to do key distribution: manual or the use of a trusted third party. The session key established happens after the master key has been established. "Confidential key distribution" doesn't refer to anything meaningful.*

Question 2: The principal objective for developing a ..... acquisition of public keys.
a) KTC
b) CRL
c) PKI

d) KDC

Explanation*: The machinery we have to distribute public keys is called PKI or the Public Key Infrastructure.*

Question 3: Digital signature certification is needed by an independent authority because;

a) The private key claimed by a sender may not be actually theirs
b) It is safe
c) It gives confidence to a business
d) The authority checks and assures customers that the public key indeed belongs to the business which claims its ownership.

Explanation*: The idea of the certificates is that there is this trusted third party who can vouch for a given public key, that they have verified the owner's identity, and that the given public is indeed their publish key (by providing the has). Therefore, the correct answer is D. B and C are vague and lack technical backing. A is incorrect because we never share private keys.*

Question 4: Kerberos consists of a/an .....

a) Authorization Server
a) Client Server
a) Authentication Server
a) Mail Server

Explanation*: We discussed two key elements of the Kerberos protocol. Authentication server and the Ticket Granting Server. Based on the given answers, the correct answer is "C. Authentication Server."*

Question 5: Kerberos is based on the Needham-Schroeder Protocol. TRUE/FALSE.

a) TRUE
b) FALSE

Explanation*: Kerberos is based on the basic Needham-Schroeder Protocol with more advanced features like using timestamps, separating the authentication function from the ticket-granting function etc. Therefore, the correct answer is TRUE.*

Question 6: The Kerberos protocol protects against which of the following attacks?

a) Dictionary Attacks
b) Replay Attacks
c) Denial of Service Attacks
d) Person in the Middle Attacks

Explanation*: Kerberos is an authenticated key exchange protocol that uses a KDC (Key Distribution Center). Therefore, the primary attack it tries to mitigate is the person-in-the-middle attack.*

Question 7: Consider the below two statements.

i) Perfect Forward Secrecy is a feature of specific key agreement protocols that assures that session keys

will not be compromised even if long-term secrets used in the session key exchange are compromised.

ii) Signed Diffe-Hellman Key exchange can provide Perfect Forward Secrecy.

Which of the below correctly represents the TRUE/FALSE nature of the two statements?

a) FALSE, FALSE
b) FALSE, TRUE
c) TRUE, FALSE
d) TRUE, TRUE

Explanation*: Both statements are true. The first is the textbook definition of Perfect Forward Secrecy. The second is true because we can use DH-key exchange to establish session keys. So, even if private and public key pairs were compromised, the attackers can't recover session keys.*

Question 8: ..... is an integer value unique within the issuing CA that is unambiguously associated with this certificate.

a) Signature identifier
b) Version
c) Serial number
d) Issuer unique identifier

Explanation*: We discussed this during the format of the X.509 certificate and the CRL part. A serial number uniquely identifies a certificate within a certificate authority.*

Question 9: The authenticator that is used as a possession factor is a .....

a) Fingerprint
b) Token
c) PIN
d) Secret question answer

Explanation*: The fingerprint is inherence. PIN and secret question answers are knowledge. Token refers to possession, something like a hardware token (e.g. a Yubikey).*

Question 10: ..... 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.

a) Identification
b) Message authentication
c) Verification
d) User authentication

Explanation*: This is not something we directly discussed this week. I included it to differentiate message authentication from user authentication. The description describes message authentication.*