Unit-5: Authentication

- Authentication is the process of recognizing a user's identity. It is the mechanism of associating an incoming request with a set of identifying credentials. The credentials provided are compared to those on a file in a database of the authorized user's information on a local operating system or within an authentication server.
- Authentication is the process of determining whether a user (or other entity) should be allowed access to a system.
- Only Authenticated users are allowed access to system resources
- Note that authentication is a binary decision— access is granted or it is not—while authorization is all about a more fine grained set of restrictions on access to various system resources
    - ❖ **Authentication:** Are you who you say you are?
    - ❖ **Authorization:** Are you allowed to do that?

Authentication System

— Technique that provides access control for systems by checking to see if a user's credentials match the credentials in a database of authorized users or in a data authentication server.

— Authentication is any process by which a system verifies the identity of a user who wishes to access it

Three Factors in Authentication System:

a. Something you know
    — Passwords/Secret key
b. Something you have
    — Secure tokens/smart card/ ATM card
c. Something you are
    — Biometrics (eg: fingerprint)

Password-Based Authentication

— A password is a string of alphabets, numbers and special characters, which is supposed to be known only to the entity (usually person) that is being authenticated

— Prompt for user ID and Password
— User enters user ID and Password
— User ID and Password Validation
— Authentication Result
— Inform user accordingly

- Passwords are often stored as hash value of original password

Dictionary Attack

— A type of brute force attack where an intruder attempts to crack a password-protected security system with a "dictionary list" of common words and phrases used by businesses and individuals

— A dictionary attack is a password attack that attempts to determine a password by trying words from a predefined list, or dictionary, of likely passwords.

— Dictionary attacks often succeed because many people have a tendency to choose short passwords that are ordinary words or common passwords, or simple variants obtained, for example, by appending a digit or punctuation character. Dictionary attacks are relatively easy to defeat, e.g. by using a passphrase or otherwise choosing a password that is not a simple variant of a word found in any dictionary or listing of commonly used passwords.

Online vs offline password (dictionary) attacks

Online Attack	**Offline Attack**
• Attacker has access to regular login interface • Attack is attempts to guess password using normal UI.	• Attacker has access to hashed passwords. • Attack is to guess words, compute hash value of that words and then compare with hashed passwords.

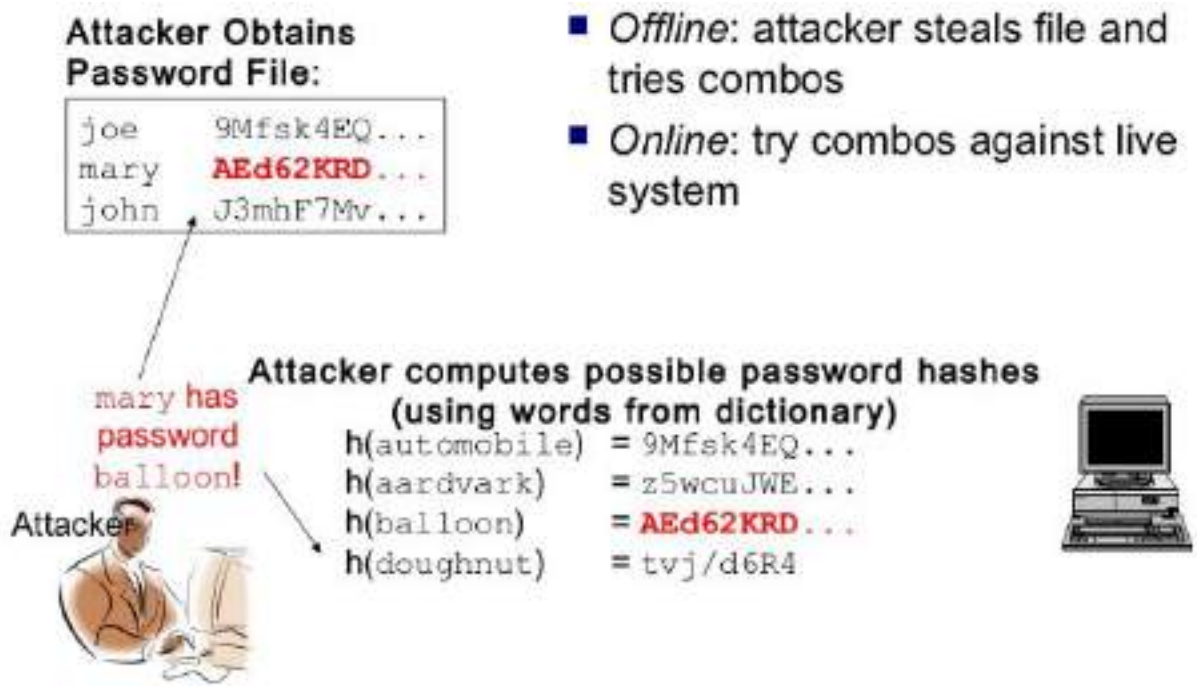
Online Dictionary Attack

- Online password attacks are the traditional type of attacks you can expect against a web application, exposed SSH terminal, or really any login interface. An online password attack consists of trying a large number of username/password combinations against the login portal in hopes of guessing the correct password.
  - Online password attacks are limited in two key ways. First, they are limited by the speed of the network. Each username/password combination has to be sent over the network to the authentication server and then the server responds accordingly. This time it takes for this back and forth transmission depends widely on the speed of the application server and the speed of the network, but a typical password attack can only get around 3 – 5 login attempts per second.
  - The second way online password attacks are limited is that they are extremely noisy. When we are attempting 5 logins every second for an average password dictionary (around 10,000 passwords), this is likely going to be flagged by almost any type of logging and alerting mechanism. Additionally, most applications are protected with account lockouts. When a password is guessed incorrectly a certain number of times in a row, it may lock out the targeted account, block the attacker's IP address, or both.

Offline Dictionary Attack

- The difference between offline and online password attacks is that… offline password attacks are offline. Great, but what does that mean? How could a password attack be offline? Well in some cases, an attacker can get a hash of your password that they can take offline and try to crack it.
- A hash is just a one-way form of encryption. When your computer saves your password, it doesn't (or shouldn't) save your password in clear-text. Instead, it hashes your password and saves that. So, for example, if your password is Password123 your computer will store: 42f749ade7f9e195bf475f37a44cafcb. This way if anyone is able to read the memory of your computer, they won't be able to know what your password is.
- Now when you login to your computer, the computer takes what you put in the password prompt, computes a hash, and compares that hash with the one it stored when you set your

password. If the passwords match, you are granted access. **An offline password attack will take this hash offline and try to find the clear-text value that computes to that hash. To do this, an attacker will use a computer (or a beefed up computer) to take passwords, compute the hash, and compare them very quickly. This will be performed over and over again until a match is found.**



- In an offline password attack, the attacker is never actually attempting to login to the application server. This means it is invisible to the security team and logs. This also means that common protections such as account lockouts will not work. This is because the attacker is going to take it offline, find the password, and then only one correct attempt will be registered by the application.
- Another major difference between offline and online password attacks is speed. While online password attacks are limited by the speed of the network, offline password attacks are limited only by the speed of the computer the attacker is using to crack them

Best practices to defend against dictionary and brute-force attacks

Using a strong, uncommon password will make an attacker's job more difficult, but not impossible.

Some tricks to prevent such attacks are as follows:

Slow down repeated logins: This is the simplest countermeasure available. An end user is unlikely to notice a 0.1 second delay while logging in, but that delay would accumulate quickly for an attacker, especially if they cannot parallelize their attempts.

Force captchas after multiple failed logins: While a user could have simply forgotten which password they used for the account, this will help slow down an attacker significantly. This is a great deterrent method as for modern captchas are difficult to defeat with computers. Many captchas need manual inputs in order to be solved.

Lock accounts: Even better, a system can be configured to lock an account after a specified number of attempted logins. Many websites will trigger additional protections for accounts with repeated bad password attempts. In the extreme case, for example, an iPhone will self-destruct (wipe all data) after 10 tries.

Refresh passwords: Modern systems typically require users to cycle passwords regularly. Some corporate environments require users to change passwords every 90 days, or maybe even every 30 days. The rationale behind this is that an attacker who is attempting a brute-force attack against a complex password would need weeks to succeed. If the password changes during that time frame, the attacker will need to start over. However, as many users would confess, these strict password requirements can backfire, with users choosing weaker, sequential passwords ('longhorns2018,' 'longhorns2019,' and so on). An attacker would quickly try incrementing the password.

Challenge Response System

— Challenge Response Authentication Mechanism (CRAM) is the most often used way to authenticate actions. They are a group of protocols in which one side presents a challenge (to be answered) and the other side must present a correct answer (to be checked/validated) to the challenge in order to get authenticated.

Following Protocol is a **challenge-and-response protocol**. In it, we assume that Alice is identifying herself to Bob, and their common secret key is denoted by K. (Bob can also identify himself to Alice, by interchanging the roles of Alice and Bob in the scheme.) In this scheme, Bob sends a **challenge** to Alice, and then Alice sends Bob her **response**.

1. Bob chooses a random challenge, denoted by r, which he sends to Alice.
2. Alice computes her response

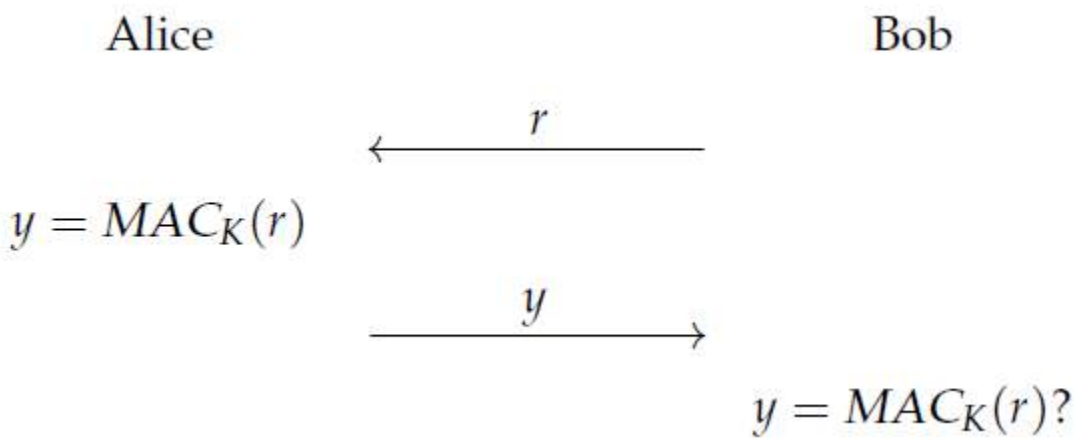
$$y = MAC_K(r)$$

and she sends y to Bob.

3. Bob computes

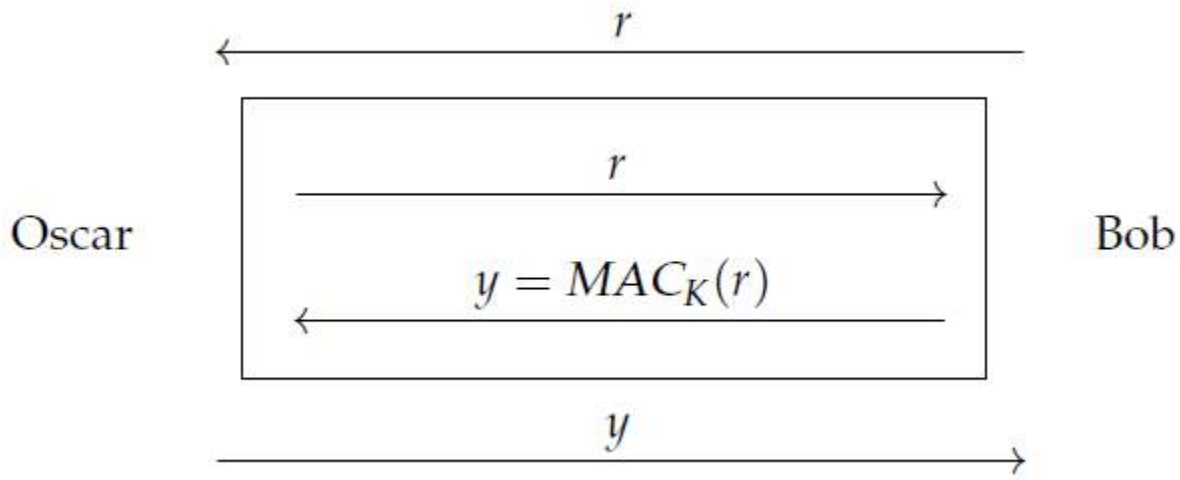
$$y' = MAC_K(r).$$

If $y' = y$, then Bob "accepts"; otherwise, Bob "rejects."



But this protocol is insecure (because of parallel session attack)

In **parallel session attack** Oscar impersonates Alice. The attack is depicted in following figure



Within the first session (in which it is supposed that Oscar is impersonating Alice to Bob), Oscar initiates a second session in which he asks Bob to identify himself. This second session is boxed in above figure.

In this second session, Oscar gives Bob the same challenge that he received from Bob in the first session.

Once he receives Bob's response, Oscar resumes the first session, in which he relays Bob's response back to him. Thus Oscar is able to successfully complete the first session

Following is **the secure challenge response protocol**

1. Bob chooses a random challenge, r, which he sends to Alic
2. Alice computes
$$y = MAC_K(ID(Alice) \parallel r)$$
and sends y to Bob.
3. Bob computes
$$y' = MAC_K(ID(Alice) \parallel r).$$
If $y' = y$, then Bob "accepts"; otherwise, Bob "rejects."

Note: A scheme in which Alice and Bob are both proving their identities to each other is called **mutual authentication** or **mutual identification**. Both participants are required to "accept" if a session of the scheme is to be considered a successfully completed session.

Biometric System

(Fingerprint, face recognition, eye retina and iris recognition etc)

- Biometrics represent the "something you are" method of authentication
    - "you are your key"
- There are many different types of biometrics, including such long-established methods as fingerprints.
    - Recently, biometrics based on speech recognition, gait (walking) recognition, and even a digital doggie (odor recognition) have been developed.
- Biometrics are currently a very active topic for research

In the information security arena, biometrics are seen as a more secure alternative to passwords. For biometrics to be a practical replacement for passwords, cheap and reliable

systems are needed. Today, usable biometric systems exist, including laptops/smartphones using thumbprint authentication, palm print systems for secure entry into restricted facilities, the use of fingerprints to unlock car doors, and so on. But given the potential of biometrics—and the well-known weaknesses of password-based authentication—it's perhaps surprising that biometrics are not more widely used.

An ideal biometric would satisfy all of the following:

Universal — A biometric should apply to virtually everyone. In reality, no biometric applies to everyone. For example, a small percentage of people do not have readable fingerprints.

Distinguishing — A biometric should distinguish with virtual certainty. In reality, we can't hope for 100% certainty, although, in theory, some methods can distinguish with very low error rates.

Permanent — Ideally, the physical characteristic being measured should never change.

In practice, it's sufficient if the characteristic remains stable over a reasonably long period of time.

Collectable — The physical characteristic should be easy to collect without any potential to cause harm to the subject. In practice, collectability often depends heavily on whether the subject is cooperative or not.

Reliable, robust, and user-friendly — These are just some of the additional real-world considerations for a practical biometric system. Some biometrics that have shown promise in laboratory conditions have subsequently failed to deliver similar performance in practice.

Two phases to a biometric system

There are two phases to a biometric system which are:

1. Enrollment phase
2. Recognition phase

- First, there is an **enrollment** phase, where subjects have their biometric information gathered and entered into a database.
    - Typically, during this phase very careful measurement of the pertinent physical information is required. Since this is one-time work (per subject), it's acceptable if the process is slow and multiple measurements are required. In some fielded systems, enrollment has proven to be a weak point since it may be difficult to obtain results that are comparable to those obtained under laboratory conditions.
- The second phase in a biometric system is the **recognition** phase. This occurs when the biometric detection system is used in practice to determine whether (for the authentication problem) to authenticate the user or not.
    - This phase must be quick, simple, and accurate.

Some Biometric Authentication Techniques

Retinal scanner: This is a device that examines the tiny blood vessels in the back of your eye. The layout is as distinctive as a fingerprint and apparently easier to read.

Fingerprint readers: This would seem an obvious technology since fingerprints have been used as a method of identification for many years.

Face recognition: Looking at a digitized picture of a person, a computer can measure facial dimensions and do a good job of recognizing people.

Iris scanner: Like a retinal scanner, this maps the distinctive layout of the iris of your eye..

Handprint readers: They measure the dimensions of the hand: finger length, width, and so on.

Voiceprints. It turns out that it's possible to do a frequency spectrum analysis of someone's voice and get identification.

Keystroke timing. The exact way in which people type is quite distinctive, and experiments have been done with identification based on the way people type.

Signatures: These are a classic human form of authentication, and there are human experts quite adept at determining whether two signatures were produced by the same person.

The Needham-Schroeder Scheme

- One of the first session key distribution schemes is the Needham-Schroeder SKDS (session key distribution schemes), which was proposed in 1978.

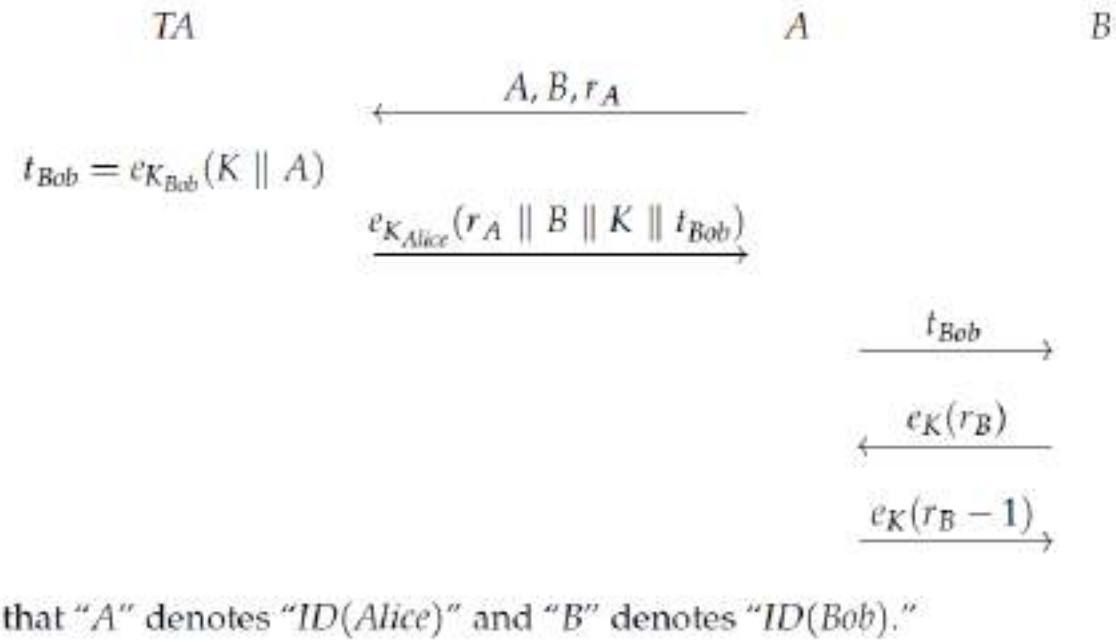


Figure: Needham-Schroeder Scheme

The Steps in this scheme are as follows:

1. Alice chooses a random number, r_A. Alice sends $ID(Alice)$, $ID(Bob)$, and r_A to the TA.
2. The TA chooses a random session key, K. Then it computes

$$t_{Bob} = e_{K_{Bob}}(K \parallel ID(Alice))$$

(which is called a ***ticket to Bob***) and

$$y_1 = e_{K_{Alice}}(r_A \parallel ID(Bob) \parallel K \parallel t_{Bob}),$$

and it sends y_1 to Alice.

3. Alice decrypts y_1 using her key K_{Alice}, obtaining K and t_{Bob}. Then Alice sends t_{Bob} to Bob.
4. Bob decrypts t_{Bob} using his key K_{Bob}, obtaining K. Then, Bob chooses a random number r_B and computes $y_2 = e_K(r_B)$. Bob sends y_2 to Alice.
5. Alice decrypts y_2 using the session key K, obtaining r_B. Then Alice computes $y_3 = e_K(r_B - 1)$ and she sends y_3 to Bob.

In flow 1, Alice asks the TA for a session key to communicate with Bob. At this point, Bob might not even be aware of Alice's request.

The TA transmits the encrypted session key to Alice in flow 2, and Alice sends an encrypted session key to Bob in flow 3.

Thus flows 1–3 of Needham-Schroeder comprise the session key distribution: the session key K is encrypted using the secret keys of Alice and Bob and it is distributed to both of them.

The purpose of flows 4 and 5 is to convince Bob that Alice actually possesses the session key K. This is accomplished by having Alice use the new session key to encrypt the challenge r_B - 1; the process is called key confirmation (from Alice to Bob).

- There are some validity checks required in the Needham-Schroeder SKDS, where the term validity check refers to verifying that decrypted data has the correct format and contains expected information. (Note that there are no message authentication codes being used in the Needham-Schroeder SKDS.) These validity checks are as follows:

1. When Alice decrypts y_1, she checks to see that the plaintext $d_{K_{Alice}}(y_1)$ has the form

$$d_{K_{Alice}}(y_1) = r_A \parallel ID(Bob) \parallel K \parallel t_{Bob}$$

for some K and t_{Bob}. If this above condition holds, then Alice "accepts"; otherwise, Alice "rejects" and aborts the session.

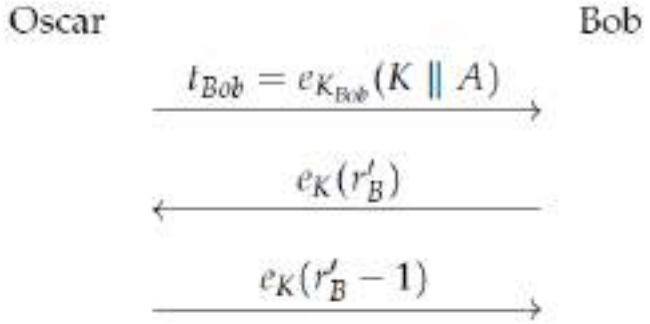
2. When Bob decrypts y_3, he checks to see that the plaintext

$$d_K(y_3) = r_B - 1.$$

If this condition holds, then Bob "accepts"; otherwise, Bob "rejects."

The Denning-Sacco Attack on the NS Scheme

- In 1981, Denning and Sacco discovered a replay attack on the Needham- Schroeder SKDS.
  - — Suppose Oscar records a session, say S, of the Needham-Schroeder SKDS scheme between Alice and Bob, and somehow he obtains the session key, K, for the session S. (this attack model is called a "**known session key attack.**") Then Oscar can initiate a new session, say S', of the Needham-Schroeder SKDS with Bob, starting with the third flow of the session S', by sending the previously used ticket, t_{Bob}, to Bob:



Notice that when Bob replies with $e_K(r'_B)$, Oscar can decrypt this using the known key K, subtract 1, and then encrypt the result. The value $e_K(r'_B - 1)$ is sent to Bob in the last flow of the session S'. Bob will decrypt this and "accept."

Let's consider the consequences of this attack. At the end of the session S' between Oscar and Bob, Bob thinks he has a "new" session key, K, shared with Alice (this is because ID(Alice) occurs in the ticket t_{Bob}). This key K is known to Oscar, but it may not be known to Alice, because Alice might have thrown away the key K after the previous session with Bob, namely S, terminated. Hence, there are two ways in which Bob is deceived by this attack:

1. The key K that is distributed in the session S' is not known to Bob's intended peer, Alice.
2. The key K for the session S' is known to someone other than Bob's intended peer (namely, it is known to Oscar).

Kerberos

- Kerberos is a computer-network authentication protocol that works on the basis of tickets to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner. The protocol was named after the character Kerberos (or Cerberus) from Greek mythology, the ferocious three-headed guard dog of Hades. Its designers aimed it primarily at a client–server model and it provides mutual authentication—both the user and the server verify each other's identity. Kerberos protocol messages are protected against eavesdropping and replay attacks.
- Kerberos requires a trusted third party, and may use public-key cryptography during certain phases of authentication

Kerberos comprises a popular series of schemes for session key distribution that were developed at MIT in the late 1980s and early 1990s.

A simplified treatment of version five of the scheme is discussed here.

This is presented as follows:

1. Alice chooses a random number, r_A. Alice sends $ID(Alice)$, $ID(Bob)$, and r_A to the TA.

2. The TA chooses a random session key K and a validity period (or lifetime), L. Then it computes a ticket to Bob,

$$t_{Bob} = e_{K_{Bob}}(K \parallel ID(Alice) \parallel L),$$

and

$$y_1 = e_{K_{Alice}}(r_A \parallel ID(Bob) \parallel K \parallel L).$$

The TA sends t_{Bob} and y_1 to Alice.

3. Alice decrypts y_1 using her key K_{Alice}, obtaining K. Then Alice determines the current time, *time*, and she computes

$$y_2 = e_K(ID(Alice) \parallel time).$$

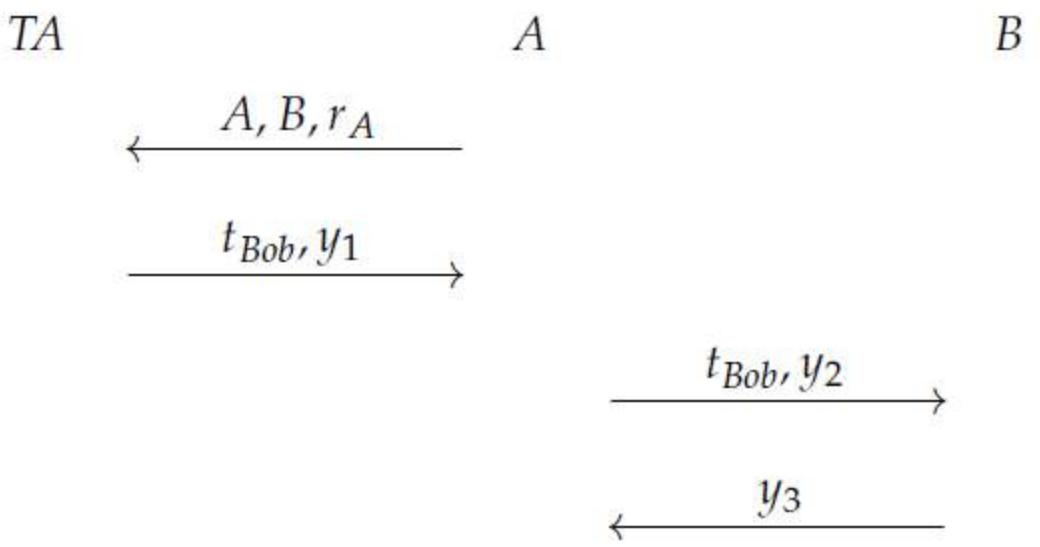
Finally, Alice sends t_{Bob} and y_2 to Bob.

4. Bob decrypts t_{Bob} using his key K_{Bob}, obtaining K. He also decrypts y_2 using the key K, obtaining *time*. Then, Bob computes

$$y_3 = e_K(time + 1).$$

Finally, Bob sends y_3 to Alice.

Following diagram depicts the four flows in a session of the scheme :



where

$$
\begin{aligned}
A &= ID(Alice), \\
B &= ID(Bob), \\
t_{Bob} &= e_{K_{Bob}}(K \parallel A \parallel L), \\
y_1 &= e_{K_{Alice}}(r_A \parallel B \parallel K \parallel L) \\
y_2 &= e_K(A \parallel time), \quad \text{and} \\
y_3 &= e_K(time + 1).
\end{aligned}
$$

- As was the case with Needham-Schroeder, there are certain validity checks required in Kerberos. These are as follows:

1. When Alice decrypts y_1, she checks to see that the plaintext $d_{K_{Alice}}(y_1)$ has the form

$$d_{K_{Alice}}(y_1) = r_A \parallel ID(Bob) \parallel K \parallel L,$$

for some K and L. If this condition does not hold, then Alice "rejects" and aborts the current session.

2. When Bob decrypts y_2 and t_{Bob}, he checks to see that the plaintext $d_K(y_2)$ has the form

$$d_K(y_2) = ID(Alice) \parallel time$$

and the plaintext $d_{K_{Bob}}(t_{Bob})$ has the form

$$d_{K_{Bob}}(t_{Bob}) = K \parallel ID(Alice) \parallel L,$$

where $ID(Alice)$ is the same in both plaintexts and $time \leq L$. If these conditions hold, then Bob "accepts"; otherwise, Bob "rejects."

3. When Alice decrypts y_3, she checks that $d_K(y_3) = time + 1$. If this condition holds, then Alice "accepts"; otherwise, Alice "rejects."

- When a request for a session key is sent by Alice to the TA, the TA will generate a new random session key K.

  As well, the TA will specify the lifetime, L, during which K will be valid. That is, the session key K is to be regarded as a valid key until time L. All this information is encrypted before it is transmitted to Alice.

- Alice can use her secret key to decrypt y_1, and thus obtain K and L. She will verify that the current time is within the lifetime of the key and that y_1 contains Alice's random challenge, rA. She can also verify that y_1 contains ID(Bob), where Bob is Alice's intended peer. These checks prevent Oscar from replaying an "old" y_1, which might have been transmitted by the TA in a previous session.
- Next, Alice will relay t_{Bob} to Bob. As well, Alice will use the new session key K to encrypt the current time (denoted by time) and ID(Alice). Then she sends the resulting ciphertext y_2 to Bob.

- When Bob receives t_{Bob} and y_2 from Alice, he decrypts t_{Bob} to obtain K, L, and ID(Alice). Then he uses the new session key K to decrypt y_2 and he verifies that ID(Alice), as decrypted from t_{Bob} and y_2, are the same. This assures Bob that the session key encrypted within t_{Bob} is the same key that was used to encrypt y_2. He should also check that time <= L to verify that the key K has not expired.
- Finally, Bob encrypts the value time+1 using the new session key K and sends the result back to Alice. When Alice receives this message, y_3, she decrypts it using K and verifies that the result is time + 1. This assures Alice that the session key K has been successfully transmitted to Bob, since K is needed in order to produce the message y_3

Note: The purpose of the lifetime L is to prevent an active adversary from storing "old" messages for retransmission at a later time, as was done in the Denning- Sacco attack on the Needham-Schroeder SKDS. One of the drawbacks of Kerberos is that all the users in the network should have synchronized clocks, since the current time is used to determine if a given session key K is valid. In practice, it is very difficult to provide perfect synchronization, so some amount of variation in times must be allowed.