Note: Khi làm việc trên Word, sử dụng những mẫu chữ là Heading 1, 2, … và Normal để dễ chỉnh sửa tự động về sau hơn.

Contents

[Contents 2](#_Toc161230325)

[I. Remote User-Authentication Principles 3](#_Toc161230326)

[II. Remote User-authentication using Symmetric encryption 4](#_Toc161230327)

[1. Mutual Authentication 4](#_Toc161230328)

[2. One-Way Authentication 7](#_Toc161230329)

[III. Kerberos 9](#_Toc161230330)

[IV. Remote User-Authentication using Asymmetric Encryption 10](#_Toc161230331)

[V. Real Life Applications 11](#_Toc161230332)

[VI. References 12](#_Toc161230333)

# Remote User-Authentication Principles:

## Introduction:

In the realm of computer security, user authentication is the critical building block, essential for establishing defenses and enforcing user accountability. User authentication is the procedure for verifying identities within a system, as defined by RFC 4949. This involves a two-step process:

* **Identification:** where a user presents an identifier
* **Verification:** where the user provides corroborative authentication information.

# Example: User Identification and Verification:

Taking the user Alice Toklas, with identifier ABTOKLAS, as an example: her unique identifier and corresponding password would be securely stored within the system she accesses. While her identifier may be public, allowing for functionalities like email, her password remains confidential, ensuring that her activities and permissions on the system are secured and auditable.

## NIST Framework for User Authentication:

The National Institute of Standards and Technology (NIST) has set forth a comprehensive guideline, the SP 800-63-2 (Electronic Authentication Guideline August 2013), which delineates a robust framework for electronic use authentication. This framework is a cornerstone for instilling trust in electronic identities and is pivotal for determining the authenticity of individuals engaging with information systems.

## Registration and Credential Issuance:

The NIST framework starts with the registration process, where individuals, referred to as applicants, engage with a Registration Authority (RA). The RA plays a crucial role in verifying and vouching for the identity of an applicant before they become subscribers to a Credential Service Provider (CSP). The CSP then issues a credential to the subscriber, which is a pivotal element of the authentication process.

## Credential as a Key Element

Credentials, according to NIST, are authoritative bindings of an identity to a token that a subscriber possesses. This token can take various forms, such as an encrypted password or an encryption key. These credentials and tokens form the basis for the user's subsequent authentication and are critical for maintaining the integrity of the authentication process.

## Authentication Procedure

Post-registration, authentication involves the subscriber presenting the token for validation during login attempts or access requests. This interaction primarily occurs between the subscriber, now the claimant, and the system performing the authentication, known as the verifier. Upon successful verification, the claimant is authenticated, and the verifier relays an authenticated assertion to the Relying Party (RP), which then decides on the access or authorization level based on this assertion.

## E-Authentication Model

NIST's e-authentication model encompasses a networked environment where authentication—and subsequent authorization—might occur over the Internet or local networks. This wide applicability underscores the framework's versatility in addressing various scenarios and technological contexts.

## Security Considerations

The framework also addresses security considerations, such as the safeguarding of credentials and the prevention of unauthorized access through the use of electronic credentials. It emphasizes the need for security measures that align with the sensitivity of the transactions and the potential risks involved.

## Federated Identity and Cross-Domain Authentication

NIST's guidelines also extend to federated identity management, which enables a user to utilize a single authentication credential across multiple domains, thereby simplifying the authentication process for users and reducing the administrative burden on organizations.

## Continuous Evolution

It is important to note that while the NIST framework provides a foundational model, actual implemented systems might be more complex, tailored to specific organizational needs, and subject to continuous updates in response to evolving technological landscapes and security threats.

## Methods of Authentication:

User authentication can be conducted through knowledge (e.g., passwords), possession (e.g., smart cards), inherent traits (e.g., fingerprints), or behavior (e.g., voice recognition). Each method presents its unique set of challenges, from the risk of theft to administrative burdens.

## Mutual Authentication and Its Importance:

Mutual authentication protocols are paramount, especially for secure communication and session key exchanges. They must tackle issues related to confidentiality and timeliness to counter threats like **replay attacks**, which could compromise the system.

## Replay attacks and Mitigation Strategies:

Replay attacks are malicious activities where a valid message is captured and resent by an adversary. Strategies to combat these include the use of sequence numbers, timestamps, and challenge/response mechanisms, each suitable for different application contexts and with their own set of trade-offs.

Example of replay attack:

* **Simple Replay:** An attacker captures a message and retransmits it without alteration at a later time.
* **Timestamped Replay Within Valid Window:** A message with a valid timestamp is replayed during its window of validity, potentially leading to unauthorized access if not detected.
* **Suppressed Original Message:** Similar to the previous example, but the attacker prevents the original message from reaching its destination, making the replayed message appear to be the legitimate one.
* **Backward Replay Without Modification:** If symmetric encryption is used, an attacker might replay a message back to sender, exploiting the symmetric nature of the keys.

## One-way Authentication for Email:

Encryption gains prominence in one-way authentication for email communications, enabling messages to remain confidential and authenticated while transiting through mail-handling protocols like SMTP or X.400.

# Remote User-authentication using Symmetric encryption

## Mutual Authentication

Mutual authentication, as the name implies, involves both parties proving their identities to each other. This ensures that not only is the client confident in the server's identity, but the server is also confident in the client's identity. Mutual authentication is crucial in scenarios where trust needs to be established in both directions.

An example of mutual authentication is seen in some secure communication protocols like Kerberos. In Kerberos, the client proves its identity to the Key Distribution Center (KDC) and receives a ticket. When the client presents this ticket to the server, the server can verify the ticket's validity with the help of the KDC, thus authenticating the client. Meanwhile, the server also proves its identity to the client by showing that it can decrypt the information sent by the KDC, which could only be done if the server is legitimate.

**Key Distribution Center (KDC)**:

* + - Each party in the network shares a secret key, known as a master key, with the KDC. The KDC is responsible for generating keys to be used for a short time over a connection between two parties, known as session keys, and for distributing those keys using the master keys to protect the distribution.
    - [Needham–Schroeder protocol](https://en.wikipedia.org/wiki/Needham–Schroeder_protocol#See_also) [NEED78] for secret key distribution:
      * The protocol can be summarized as follows:
        1. f () is a generic function that modifies the values of nonce (N)

Diagram of a diagram of a key distribution center

Description automatically generated

A and B are identities of Alice and Bob respectively, Secret keys and are shared between A and the KDC and B and the KDC, respectively. The purpose of the protocol is to distribute securely a session key to A and B. Firstly A send a request to KDC include: A’s ID, B’s ID and a nonce, In step 2 A receive and an encrypted message from KDC that only B can decrypt, A then send this message to B in step 3. In step 4 B acknowledges and step 5 assures A know and assures B that this is a fresh message by the uses of nonce . The purpose of steps 4 and 5 is to prevent a certain type of replay attack. In particular, if an opponent is able to capture the message in step 3 and replay it, this might in some fashion disrupt operations at B.

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. 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.

* + - Denning [DENN81, DENN82] proposes to overcome this weakness by a modification to the Needham/Schroeder protocol that includes the addition of a timestamp to steps 2 and 3:
      * Her proposal assumes that the master keys, and , are secure, and it consists of the following steps:
      * T is a timestamp that assures A and B that the session key has only just been generated. Thus, both A and B know that the key distribution is a fresh exchange. A and B can verify timeliness by checking that where is the estimated normal discrepancy between the KDC’s clock and the local clock (at A or B) and is the expected network delay time. Each node can set its clock against some standard reference source. Because the timestamp T is encrypted using the secure master keys, an opponent, even with knowledge of an old session key, cannot succeed because a replay of step 3 will be detected by B as untimely.
      * The Denning protocol seems to provide an increased degree of security compared to the Needham/Schroeder protocol. However, a new concern is raised: namely, that this new scheme requires reliance on clocks that are synchronized throughout the network. [GONG92] points out a risk involved. The risk is based on the fact that the distributed clocks can become unsynchronized as a result of sabotage on or faults in the clocks or the synchronization mechanism. The problem occurs when a sender’s clock is ahead of the intended recipient’s clock. In this case, an opponent can intercept a message from the sender and replay it later when the timestamp in the message becomes current at the recipient’s site. This replay could cause unexpected results. Gong refers to such attacks as **suppress-replay attacks**.
      * One way to counter suppress-replay attacks is to enforce the requirement that parties regularly check their clocks against the KDC’s clock. The other alternative, which avoids the need for clock synchronization, is to rely on handshaking protocols using nonces. This latter alternative is not vulnerable to a suppress-replay attack, because the nonces the recipient will choose in the future are unpredictable to the sender.
    - In [KEHN92], an attempt is made to respond to the concerns about suppressing replay attacks and at the same time fix the problems in the Needham/Schroeder protocol.
      * The attempt was still inconsistent but there was a improved new strategy presented in [NEUM93a]:
      * Let us follow this exchange step by step:
        1. A initiate the authentication exchange by generate a nonce, , then sent to B in plaintext.
        2. B request KDC a session key by sending KDC: B’s ID and nonce (). B also send an encrypted block using the key B share between B and the KDC () which include A’s ID, Nonce and B’s timestamp. This block is used to instruct the KDC to issue credentials to A and specify A’s information and a suggested credentials expiration time.
        3. A receives B’s nonce and two block from KDC, one A can decrypt using the key A share with KDC and the other block A can’t decrypt. The block A can decrypt includes: to verify that B received A initial message, to ensure that this is not a replay, Session key , and time limit for on its use ().
        4. A then send B the block A can’t decrypt which work like a ticket to B alongside B’s nonce which encrypted with session key. B can then decrypt that block and receive session key, A’s ID to confirm sender. B can the use session key to decrypt the other block to recover B’s nonce then B can confirm that this is not a replay message.
      * This protocol provides an effective, secure means for A and B to establish a session with a secure session key. Furthermore, the protocol leaves A in possession of a key that can be used for subsequent authentication to B, avoiding the need to contact the authentication server repeatedly.
      * 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:
      * When B receives the message in step 1, it verifies that the ticket has not expired. The newly generated nonces and assure each party that there is no replay attack.
      * In all the foregoing, the time specified in is a time relative to B’s clock. Thus, this timestamp does not require synchronized clocks, because B checks only self-generated timestamps.

Some authentication protocols that support mutual authentication:

* + - Kerberos: Supports mutual authentication. Both the client and the server verify each other's identity using tickets provided by the Key Distribution Center (KDC). Kerberos was used in large networks of companies such as AWS, Google Cloud, and Microsoft Azure to facilitate SSO.
    - NT LAN Manager (NTLM): Provides mutual authentication. Both the client and the server verify each other's identity through a challenge-response mechanism, where the client proves its knowledge of the password through a calculation without sending the password to the server. SharePoint, a web-based collaborative platform that is part of the Microsoft Office suite, employs NTLM for authentication in environments where more modern protocols like OAuth are not configured. This is particularly common in intranet scenarios within organizations.
    - Challenge Handshake Authentication Protocol (CHAP): Used by Point-to-Point Protocol (PPP) servers to validate the identity of remote clients. CHAP periodically verifies the identity of the client by using a three-way handshake, where the server sends a challenge to the client, the client responds with a value calculated using a one-way hash function, and the server checks this response against its own calculation of the expected hash value. Internet Service Providers (ISPs) like AT&T and Verizon have used CHAP in the past for dial-up and DSL services to authenticate subscribers' Internet connections.
    - Secure Remote Password (SRP): A password-authenticated key exchange protocol that allows the establishment of a secure communication channel without transmitting the password over the network. It's designed to work around the vulnerabilities of transmitting passwords over insecure networks. 1Password, a popular password management service, implements a version of the Secure Remote Password protocol to authenticate users to their servers securely. This ensures that even during the login process, users' passwords are never transmitted or at risk of interception, enhancing the security of user data.

## One-Way Authentication

* + In one-way authentication, only one party proves its identity to the other, while the identity of the second party remains unverified. This is commonly seen in scenarios where a client needs to prove its identity to a server to gain access to resources, but the server does not need to prove its identity to the client.
  + A typical example of one-way authentication is when you log into a website. You, as the user, provide your username and password to the server to prove who you are. However, unless a separate mechanism is used (like certificates in SSL/TLS), the server does not prove its identity to you in the same way.
  + The Key Distribution Center (KDC) strategy for single-side authentication using symmetric encryption simplifies secure communications by managing the distribution of secret keys within a network. This approach is particularly useful in environments where multiple clients need to securely authenticate various services or servers.
    - With some refinement, the KDC strategy illustrated above is a candidate for encrypted electronic mail. Because we wish to avoid requiring that the recipient (B) be online at the same time as the sender (A), steps 4 and 5 must be eliminated. For a message with content M, the sequence is as follows:
    - This approach guarantees that only the intended recipient of a message will be able to read it. It also provides a level of authentication that the sender is A. As specified, the protocol does not protect against replays. Some measure of defense could be provided by including a timestamp with the message. However, because of the potential delays in the email process, such timestamps may have limited usefulness.
  + Some authentication protocols that support one-way authentication:
    - Password Authentication Protocol (PAP): A simple, and less secure, authentication protocol where passwords are sent over the network as plain text. It's often used in some contexts where security is not a major concern or is provided by other means.
    - LAN Manager (LM) Hash: An outdated and insecure authentication protocol used by Microsoft for network LAN Manager sessions. It uses a hashed version of the user's password for authentication, which has been known to be susceptible to various attacks and is generally not recommended for use.

# Kerberos

Kerberos4, developed under Project Athena at MIT, addresses critical security concerns in open distributed environments. Here, users at various workstations access services on network-distributed servers, raising the challenge of restricting access to authorized Individuals and authenticating service requests. The inherent trust issue with workstations, which may not accurately identify users to network services, loads to several potential threats:

* **Impersonation at the Workstation Level:** An individual could access a workstation and masquerade as another user, thus gaining unauthorized access to resources or data.
* **Network Address Manipulation:** By altering a workstation’s network address, an attacker could make requests appear to originate from a different, potentially Impersonated, workstation, misleading the authentication process.
* **Eavesdropping ang replay attack:** Intercepting communication between the user and the server allows an attacker not only to gain unauthorized access through replayed messages but also to disrupt normal operations by injecting or altering messages.

Each of these scenarios poses a significant threat to the security of network services, allowing unauthorized access to sensitive data and services. Kerberos v4 was designed to mitigate these vulnerabilities by providing a robust authentication framework that does not rely on the security of the workstation or the integrity of network addresses but rather on secure, encrypted authentication exchanges that safeguard against eavesdropping and replay attacks.

Two version 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 2121).

## Motivation

In a world where technology is ever-evolving, the safeguarding of user information and resources has become increasingly important. Traditionally, users operating on standalone personal computers without network connections could rely on physical security measures to protect their data. Conversely, users connected to centralized time-sharing systems depended on the operating system's security protocols, which identified users at login and enforced access controls based on those identities.

However, the landscape of today's computing environment is vastly different. It is characterized by a distributed architecture, where users interact with a network of both centralized and distributed servers through their personal workstations. This setup necessitates a reevaluation of security approaches, leading to three primary strategies:

* Trust in the client workstation to accurately verify the identity of its users and depend on servers to implement access controls based on these identities.
* Allow client systems to authenticate to servers, assuming the client accurately knows the user’s identity.
* Mandate that users authenticate themselves for every service t hey access and require servers to authenticate themselves to users as well.

In more contained environments, where a single organization controls all systems, the first or second approach might be sufficient. However, in open environments with connections to external networks, the third strategy becomes crucial to ensure the security of use data and services at the server level. This is where Kerberos comes into play.

Kerberos, developed as part of MIT’s Project Athena, is designed to cater to the third security approach. It operates under a distributed client/server model, utilizing one or more Kerberos servers to facilitate an authentication service. This service is built on the foundation of mutual trust between clients and servers, with Kerberos acting as the trusted intermediary.

Outlined in the initial report on Kerberos are several key requirements for this authentication service:

* **Security:** It should be virtually impossible for anyone eavesdropping on the network to gather enough information to impersonate a user. Kerberos aims to be the fortified link in the network security chain.
* **Reliability:** Given that many services depend on Kerberos for access control, its availability is synonymous with the availability of these services. Therefore, Kerberos must be highly reliable, supported by a distributed architecture to ensure continuous service.
* **Transparency:** The ideal scenario is one in which users are unaware of t he ongoing authentication processes, apart from the necessity to input their password.
* **Scalability:** The system needs to accommodate a growing number of clients and servers without faltering, suggesting a need for a modular and distributed architecture.

Kerberos embodies the concept of a third-party authentication service, utilizing a protocol inspired by Needham and Schroeder’s earlier work. This setup posits Kerberos as a secure, reliable, transparent, and scalable solution for modern network security challenges, ensuring that authentication processes are both robust and user-friendly.

## Kerberos version 4

**A simple authentication dialogue:**

Kerberos version 4 introduced an advanced authentication protocol designed to secure network communications. To understand its complexities, let’s simplify the authentication process into a straightforward dialogue, illustrating how Kerberos version 4 ensures secure and authenticated communication between a client and a server within an unsecured network environment.

Initial setup Kerberos version 4 authentication:

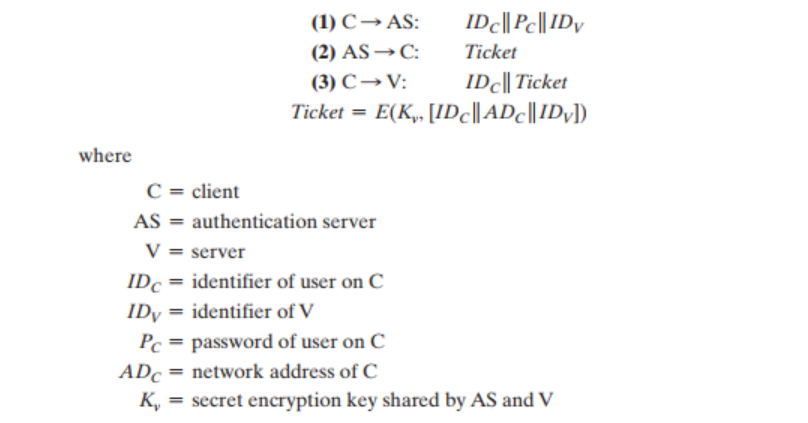
* **Participants:** A user (client ‘C’), an authentication server (‘AS’), and a desired service server (‘V’)
* **Scenario:** The user wants to access services provided by server ‘V’, but both ‘C’ and ‘V’ need assurance of each other’s identity to prevent unauthorized access.

The Dialogue:

* **Client’s Request:** ‘C’ initiates the process by requesting access to ‘V’. ‘C’ sends a massage to ‘AS’ including its own identifier (‘IDC’), its password (‘PC’), and the identifier of ‘V’ (‘IDV’). This message essentially says, “I am ‘C’, and I want to access ‘V’.”
* **AS’s Response:** ‘AS’ verifies ‘C’s credentials (using ‘IDC’ and ‘PC’) against its database. If the credentials are valid ad ‘C’ is allowed to access ‘V’, ‘AS’ creates a “ticket” for ‘C’ to use. This ticket is encrypted with a secret key that only ‘AS’ and ‘V’ share (‘Kv’), ensuring that only ‘V’ can read it. The ticket includes ‘C’’s identifier, ‘C’’s network address (‘ADC’), and ‘V’’s identifier.
* **Client’s Access Request to Server ‘V’:** Armed with the ticket, ‘C’ now sends a new message to ‘V’. This message includes ‘C’’s identifier and the ticket received from ‘AS’.
* **Server ‘V’’s Verification:** Upon receiving the message, ‘V’ decrypts the ticket using ‘Kv’ and verifies that the information within matches the incoming request. If the decrypted ticket’s identifiers align with the request’s identifiers and the ticket originates from the correct network address, ‘V’ is assured of ‘C’’s authenticity and grants access.

Key Elements:

* **Encryption:** The ticket’s encryption wit ‘Kv’ ensures that only ‘V’ can decrypt it, safeguarding against unauthorized access and ticket forgery.
* **Unique Secret Key:** The use of a unique secret key (‘Kv’) shared between ‘AS’ and ‘V’ ensures secure communication, as only these two entities can encrypt and decrypt messages they exchange.
* **Ticket Contents:** The ticket includes identifiers (‘IDC’, ‘IDV’) and the network address (‘ADC’), ensuring the ticket is explicitly tied to ‘C’’s session and intended access to ‘V’.
* **Security Against Threats:** Including the network address (‘ADC’) in the ticket combats potential threats like an attacker capturing the ticket and attempting access from a different network location. Since the ticket validates the network origin, such an attack is thwarted.



## A more secure authentication dialogue:

To enhance the security measures introduced by Kerberos Version 4, let’s explore an advanced scenario designed to tackle additional vulnerabilities, particularly focusing on reducing password exposures ad the potential for replay attacks. This more secure authentication dialogue introduces a new component, the Ticket-Granting Server (TGS), to further safeguard user credentials and streamline the authentication process for multiple service accesses during a user session.

Initial setup:

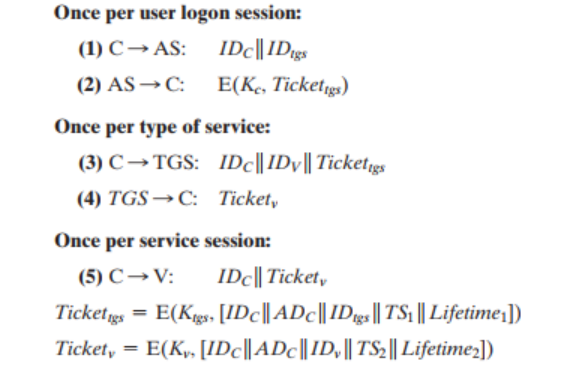
* **Participants:** A user (client ‘C’), an Authentication Server (‘AS’), a Ticket-Granting Server (‘TGS’), and a target service server (‘V’).
* **Goal:** To securely authenticate ‘C’ for access to ‘V’ while minimizing password exposure and improving the ticketing process’s efficiency.

Enhanced Authentication Process:

* Initial Ticket Request to ‘AS’:
  + ‘C’ sends a message to ‘AS’, requesting a ticket to access ‘TGS’. The message includes ‘C’’s identifier (‘IDC’) and the identifier of ‘TGS’ (‘IDTGS’), but notably, it does not include ‘C’’s password.
  + ‘AS’ validates ‘C’ using its identifier and, if authenticated, issues a ticket-granting ticket (‘TicketTGS’). This ticket is encrypted with a key (‘Kc’) derived from ‘C’’s password, ensuring only ‘C’ can use it.
* Request for Service Ticket from ‘TGS’:
  + With ‘TicketTGS’ in hand, ‘C’ requests a service ticket for ‘V’ from ‘TGS’. ‘C’ sends ‘TicketTGS’, along with a new message containing ‘C’’s identifier and ‘V’’s identifier (‘IDV’).
  + ‘TGS’ decrypts ‘TicketTGS’ to verify ‘C’’s credentials. If the veification is successful, ‘TGS’ issues a service ticket (‘TicketV’) for ‘V’, encrypted with a secret key shared between ‘TGS’ and ‘V’ (‘Kv).
* Access Request to Service Server ‘V’:
  + ‘C’ then presents ‘TicketV’ to ‘V’ to request access. This ticket proves that ‘C’ has been authenticated by ‘TGS’ and is authorized to access ‘V’.
  + ‘V’ decrypts ‘TicketV’ using ‘Kv’ and grants access to ‘C’ if the ticket is valid.

Key Security Enhancements

* **Reduced Password Exposure:** By only requiring ‘C’’s password for the initial ticket request to ‘AS’ and not for subsequent service requests, the protocol significantly reduces the exposure of ‘C’’s password over the network.
* **Ticket-Granting Ticket (‘TicketTGS'):** This ticket allows ‘C’ to obtain service tickets for various services without re-entering the password or directly contacting 'AS' again, streamlining the authentication process and enhancing security.
* **Encrypted Service Tickets:** Service tickets issued by ‘TGS’ are encrypted, ensuring that only the target service server (‘V’) can decrypt and validate them. This mechanism prevents unauthorized access and ticket forgery.
* **Dual-Role of ‘TGS’:** ‘TGS’ acts as a secure intermediary that manages ticket issuance for various services, reducing the load on ‘AS’ and improving the overall efficiency of the authentication process.



**The version 4 Authentication Dialogue**

Initial Authentication Request:

* User to Authentication Server (AS):
  + The process begins with the client (‘C’) sending a request to the Authentication SERVER (‘AS’) for access to a specific service. This request includes the client’s identifier (‘IDC’), the identifier of the Ticket-Granting Server (‘TGS’), and a timestamp (‘TS1’).
  + The purpose here is for ‘C’ to obtain a Ticket-Granting Ticket (‘TGT’) that will later be used to request service-specific tickets from the ‘TGS’.

AS Response with TGT

* ‘AS’’s Encrypted Response to Client:
  + The ‘AS’ validates ‘C’'s request and returns two crucial pieces of information: a ‘TGT’ and a session key (‘Kc’,’tgs’), both encrypted with a key derived from ‘C’'s password (‘Kc’). This ensures that only the legitimate client can decrypt and use the ‘TGT’.
  + The ‘TGT’ is specifically meant for the ‘TGS’ and is encrypted with a secret key known only to the ‘AS’ and the ‘TGS’ (‘Ktgs’).

Requesting Service Ticket from TGS

* Client to Ticket-Granting Server (‘TGS’):
  + Armed with the ‘TGT’, ‘C’ now requests a service ticket for the target service server (‘V’) from the ‘TGS’. This request includes the ‘TGT’ (proof of ‘C’'s authentication) and an authenticator, which contains ‘C’'s ID and a timestamp, encrypted with the session key (‘Kc’,’tgs’).
  + This step authenticates ‘C’ to the ‘TGS’ without sending sensitive information in plaintext and requests a ticket for ‘V’.

TGS Responds with Service Ticket

* ‘TGS’’s encrypted response to Client:
  + The ‘TGS’ decrypts the ‘TGT’, verifies the request, and issues a service ticket (‘TicketV’) for accessing ‘V’. This ticket includes ‘C’'s ‘ID’, ‘C’'s network address, and a timestamp, encrypted with a service-specific key (‘Kv’) known only to ‘TGS’ and ‘V’.
  + Alongside ‘TicketV’, ‘TGS’ sends ‘C’ a session key for communication with ‘V’ (‘Kc’,’v’), encrypted with ‘Kc’,’tgs’.

Accessing the Service on Server V

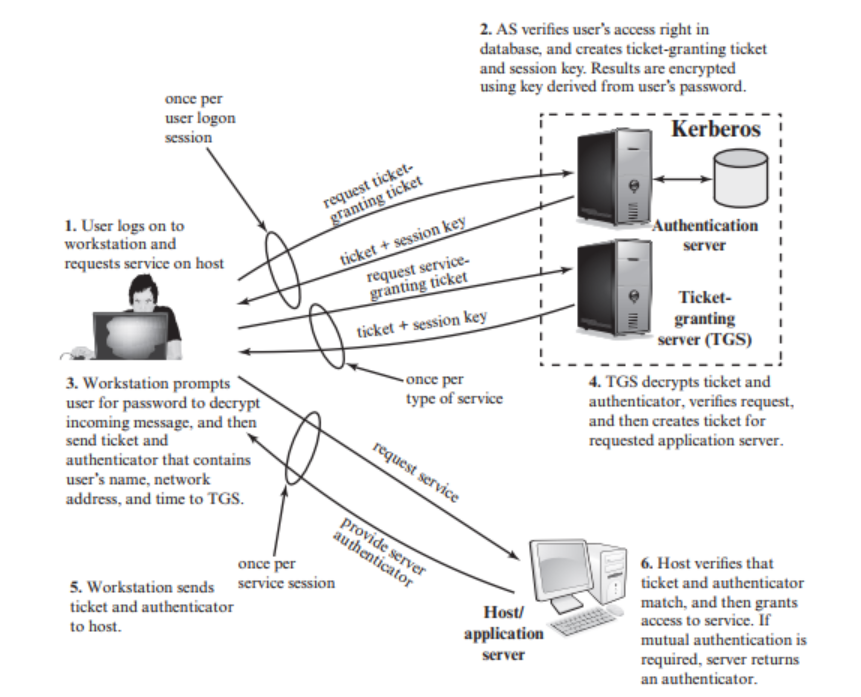
* Client Requests Access to Server ‘V’:
  + ‘C’ approaches ‘V’ with ‘TicketV’ and a new authenticator, proving ‘C’'s identity and authorization to access ‘V’. This new authenticator is encrypted with ‘Kc’,’v’.
  + ‘V’ decrypts ‘TicketV’ with ‘Kv’, validates the request against the authenticator, and grants access if the validation is successful.

Mutual Authentication (Optional)

* Optional Step for Mutual Authentication:
  + If required, ‘V’ can authenticate itself back to ‘C’ by sending a confirmation message encrypted with ‘Kc’,’v’. This step assures ‘C’ of ‘V’'s legitimacy.
  + This mutual authentication process establishes trust in both directions, ensuring both parties are who they claim to be.

Key Security Features

* Encrypted Communications: All sensitive information is encrypted, ensuring that authentication details cannot be intercepted and misused by attackers.
* Session Keys: Unique session keys for each interaction prevent replay attacks and ensure secure communication between ‘C’ and the ‘TGS’, as well as between ‘C’ and ‘V’.
* Timestamps: The use of timestamps prevents replay attacks by ensuring the freshness of each request and response.



**Kerberos realms and multiple Kerber:**

Definition of a Kerberos Realm

* A Kerberos realm is a set of managed nodes under a single administrative domain that share the same Kerberos database. The realm name is typically structured as an uppercase domain name and is used to identify the network or administrative unit within which authentication is managed. Each principal (a user or service) within a realm is uniquely identified and managed by the Kerberos server.

Components of a Kerberos Realm

* Kerberos Server (KDC): The Key Distribution Center (KDC) is central to a realm, performing the roles of both the Authentication Server (AS) and the Ticket-Granting Server (TGS). It holds the secret keys (or passwords) of all principals in the realm.
* Principals: These are entities (users, services, or machines) recognized by the Kerberos system, each identified by a unique principal name.
* Kerberos Database: This contains the secret keys of all principals and other relevant data needed for authentication.

Multiple Kerberos Realms

* In a networked world, the need often arises for users in one Kerberos realm to access services in another realm. To facilitate this, Kerberos supports inter-realm authentication, allowing users to prove their identities across different realms securely. This is particularly important in environments where networks or organizations need to allow controlled access to their resources to external users.

Inter-Realm Authentication

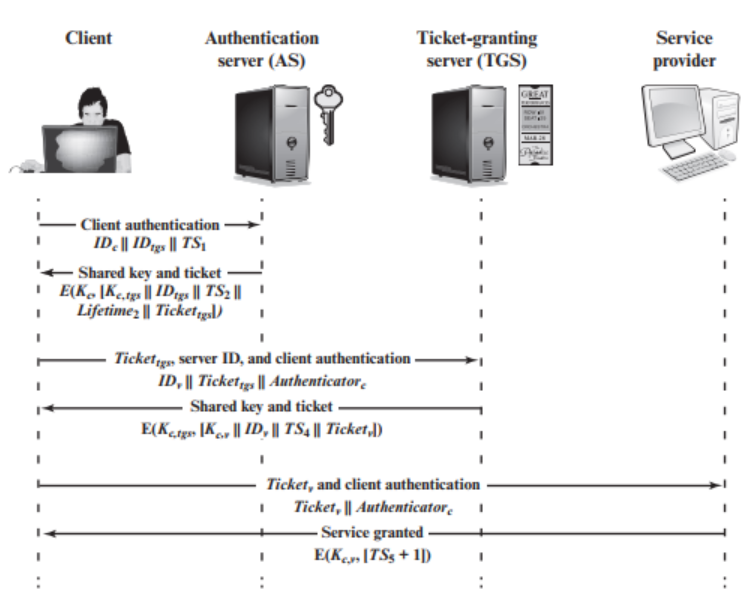
* To establish trust between two realms, the administrators of each realm must agree to share a secret key, similar to how a principal in a realm shares a secret key with the KDC. This shared key is used to encrypt and verify tickets across realms, allowing principals from one realm to authenticate themselves to services in another realm.

Steps for Cross-Realm Authentication

* Initial Authentication: A user in the originating realm authenticates to their local KDC and receives a Ticket-Granting Ticket (TGT) for their realm.
* Cross-Realm TGT Request: The user then requests a ticket for the TGS of the target realm, using the local TGT.
* Accessing Services in the Target Realm: With the cross-realm TGT, the user can now request service tickets from the target realm's TGS for services within that realm.

Challenges with Multiple Realms

* Key Management: Managing the shared keys between realms can be complex, especially as the number of interconnected realms increases.
* Trust Relationships: Establishing and maintaining trust relationships between realms requires careful coordination and security practices.
* Scalability: As the number of realms and cross-realm relationships grows, so does the complexity of managing these relationships.



## Kerberos Version 5

Key Enhancements in Kerberos Version 5

* Encryption System Independence:
  + Version 4 Limitation: Reliance on the DES encryption algorithm, raising concerns due to export restrictions and doubts about its strength.
  + Version 5 Improvement: Introduces encryption-type identifiers, allowing for the use of various encryption techniques. Encryption keys are also tagged with type and length, providing the flexibility to use different algorithms and adapt to future cryptographic needs.
* Flexible Network Address Types
  + Version 4 Limitation: Dependence on Internet Protocol (IP) addresses, excluding other network address types like ISO.
  + Version 5 Improvement: Network addresses are tagged with type and length, accommodating any network address type, enhancing interoperability across different network protocols.
* Standardized Message Byte Ordering
  + Version 4 Limitation: Inconsistent byte ordering, relying on sender-defined conventions.
  + Version 5 Improvement: Utilizes Abstract Syntax Notation One (ASN.1) and Basic Encoding Rules (BER) for defining message structures, ensuring unambiguous byte ordering.
* Extended Ticket Lifetime
  + Version 4 Limitation: Encoded lifetime values limited to 21 hours, insufficient for long-running applications.
  + Version 5 Improvement: Supports explicit start and end times for tickets, allowing for arbitrary lifetimes tailored to application requirements.
* Authentication Forwarding
  + Version 4 Limitation: Inability to forward credentials to another host for subsequent use.
  + Version 5 Improvement: Enables credential forwarding, allowing a client to access a server, which in turn can access another server on the client's behalf.
* Improved Interrealm Authentication
  + Version 4 Limitation: Requires a quadratic number of Kerberos-to-Kerberos relationships for N realms.
  + Version 5 Improvement: Introduces mechanisms that require fewer interrealm relationships, simplifying interrealm authentication.

Technical Enhancements and Protocol Revisions

* **Elimination of Double Encryption**: Simplifies ticket issuance by removing unnecessary double encryption of tickets, reducing computational overhead.
* **Standard CBC Mode Encryption with Integrity Mechanisms**: Moves away from the nonstandard PCBC mode to standard CBC mode, enhancing security and integrity of communications.
* **Session and Subsession Keys**: Addresses the risk of replay attacks by enabling the negotiation of subsession keys for one-time connections, further securing session-specific communications.
* **Preauthentication Mechanisms**: Introduces preauthentication to mitigate password attacks, making unauthorized access more difficult.

Authentication Dialogue and Flags

* Kerberos Version 5 also refines the authentication dialogue, adding options, timestamps, and nonce values to assure message freshness and prevent replay attacks. It introduces a comprehensive set of ticket flags, such as INITIAL, RENEWABLE, and FORWARDABLE, expanding the protocol's functionality to support scenarios like ticket renewal, postdated tickets, and secure credential forwarding.

Interoperability and Scalability

* With these improvements, Kerberos Version 5 significantly enhances interoperability among different realms and network environments, offering scalable solutions for secure authentication across varied infrastructures. It lays a solid foundation for future extensions and adaptations, ensuring Kerberos remains a vital part of secure network authentication frameworks.

# Remote User-Authentication using Asymmetric Encryption

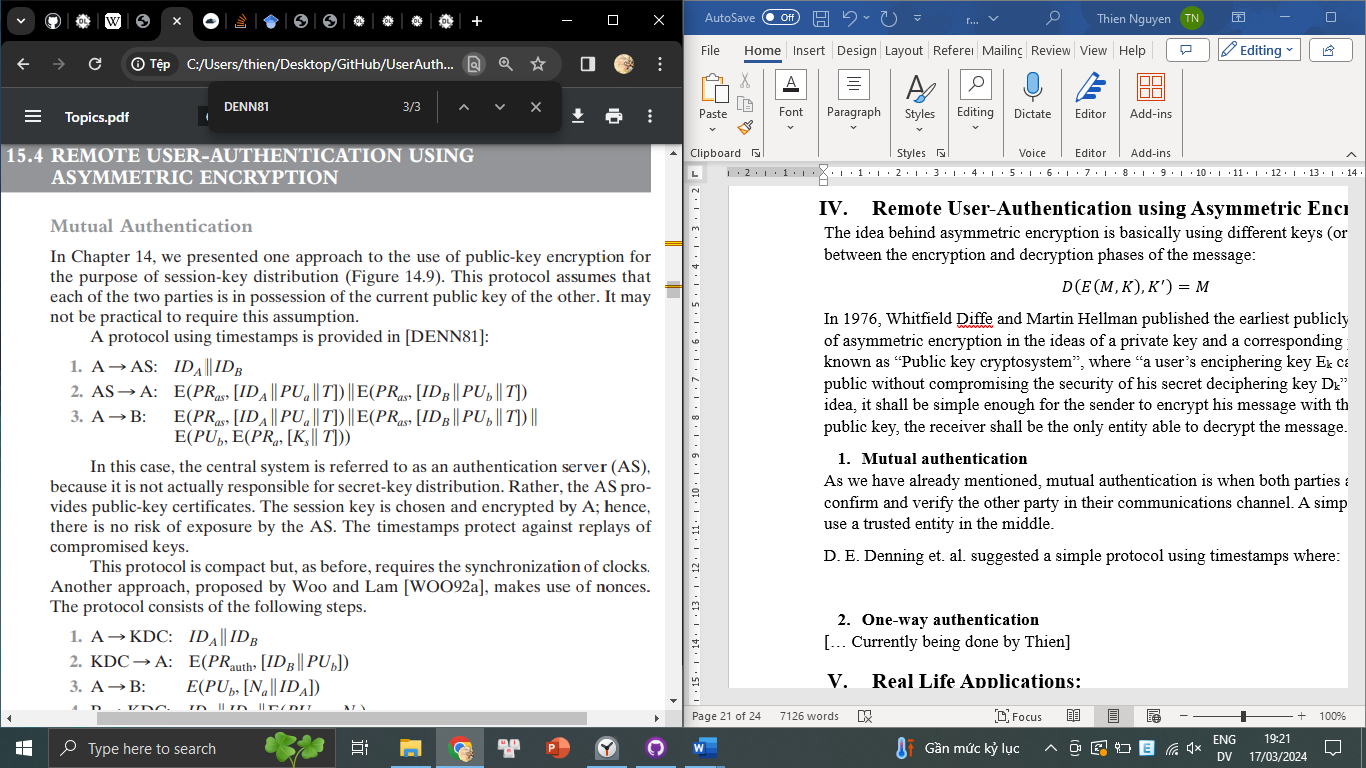
The idea behind asymmetric encryption is basically using different keys (or sets of keys) between the encryption and decryption phases of the message:

In 1976, Whitfield Diffe and Martin Hellman published the earliest publicly known work of asymmetric encryption in the ideas of a private key and a corresponding public key [1] known as “Public key cryptosystem”, where “a user’s enciphering key Ek can be made public without compromising the security of his secret deciphering key Dk”. With that idea, it shall be simple enough for the sender to encrypt his message with the receiver’s public key, the receiver shall be the only entity able to decrypt the message.

## Mutual authentication

As we have already mentioned, mutual authentication is when both parties are able to confirm and verify the other party in their communications channel. A simple idea is to use a trusted entity in the middle.

D. E. Denning et. al., in 1981, suggested a simple protocol using timestamps and certifications where:

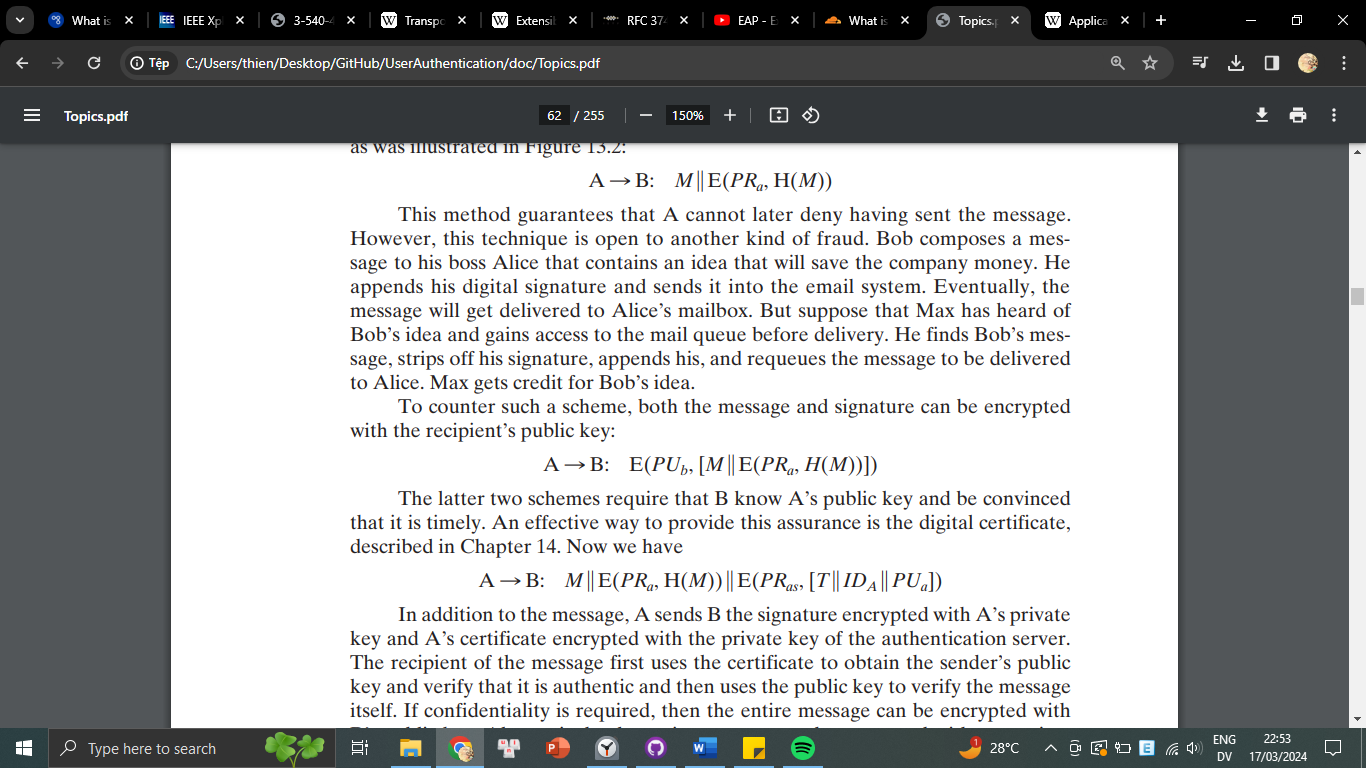


…we use a trusted entity in the middle called AS (Authentication Server) between two users A and B.

A gets to AS to request a connection to B and validation from AS as well. AS after confirming A is a valid user and the request from A is valid, it shall give A the certificates of users A and B, which A sends to B so B could confirm A through that A has recently been confirmed by trusted AS (through timestamp T and the certificates themselves). After that, B could trust to decrypt the final message from A using B’s private key and A’s public key to get the session key Ks for future exchange. It shall be noted as well that the certificates from the AS are encrypted with its own private key to avoid those certificates being edited.

## One-way authentication

This is the much simpler than mutual authentication, since only the receiver has to verify if the message comes from an intended sender. One can go with a simple idea below:



… where A combines the message and its own “signature”, then encrypts it with B’s public key. This way, A cannot deny having sent the message, and B is the only one able to decrypt with its private key.

In cases A needs to be validated where some people have restricted access in sending messages to B, B could implement an AS in order to require A to get a certificate to AS before sending to B.



# Real Life Applications:

In this part, we will simulate how the Kerberos v4 work in simple way to show its effects and effectiveness. As we know about the Kerberos v4 in part2. Kerberos v4, an early version of the Kerberos protocol developed at MIT as part of Project Athena, aimed to provide a secure method for authenticating service requests between trusted hosts across an unsecured network. It was designed to address the issue of secure authentication in a network environment where packets could be intercepted, replayed, or tampered with.

Now, we will describe how we do for simulate Kerberos v4 in C++ using sockets to sending and receiving data. We split the Kerberos operation into 3 parts and build in 8 entities (classes) includes: tgtReq, asReply, TGT (Ticket Granting Ticket), UA (User Authentication), tgsReq, tgsRes, serviceTicket, serviceAuthen.

## CLIENT-KEY DISTRIBUTION-PART1

**- Step 1:** User sends to the Authentication Server an Ticket-Granting Ticket request, it includes 4 attributes that are user’s ID, service ID, user Port and the requested lifetime for TGT. Server receives the request, then it checks the secret key based on the user’s id and decrypt the message. After that, Server generates an AS reply with 4 attributes (Ticket Granting Server’s ID, a TGS session key a life time same as TGT lifetime), along with a TGT (user’s ID, TGS’s ID, Timestamp, user’s port, a life time for TGT, and a TGS Session Key same as with the key which is provided in AS reply). Server encrypt data then send it back to User.

**- Step 2:** After receiving data from Authentication Server, user encrypts data of AS reply with its client secret key to get TGS session key. With the key which is got from AS reply, user generates and encrypts an User Authenticator message includes user’s id, a timestamp. At this moment, user also generates and TGS request, this message includes (service’s id and requested lifetime for ticket). Then, those messages will be sent to Ticket Granting Server.

## CLIENT-KEY DISTRIBUTION-PART2

It starts by looking at the service’s ID contained in the unencrypted message and checks to see if the server ID in the message is in the list of services in the KDC. If the service is in the list, TGS grabs a copy of the service secret key. Back to the messages that the TGS received from the user, service use the TGS secret key (which is use to encrypt by AS in step 2 – part 1) to decrypt the Ticket Granting Ticket. Contain in TGT is a TGS session key, and now service will use this key to decrypt the User Authenticator data. Now that both the TGT and the Authenticator message have been decrypted, the TGS start to validate the data contained in the TGS first, make sure that the user’s ID in the TGT and the Authenticator are matched, then TGS compares the timestamps, and compares the IP address in the TGT to the IP address of user in User Authenticator. Checking that the Ticket Granting Ticket has not expired if everything copasetic then the TGS performs one more important step. The TGS maintains a cache of a recently received authenticators from users, TGS will check its cache to ensure that the Authenticator it just received is not ready on the cache, this check provides replay protection if the Authenticator is not already in the cache then the TGS will add it. Now, server moves on to creating its own messages to the user. The first message contains the service ID of the service which is the user wants to access the timestamp of the message and the lifetime of the message. The second message is the service ticket, it contains the user’s id, the service’s id which the user wants to access the timestamp of the message the user’s IP address ad the lifetime of the service ticket. Server then generates a random symmetric service session key and adds the symmetric key to both messages. Final step before sending the messages is encrypting them. The first message is encrypted with the TGS session key and the service ticket is encrypted with the service secret key. These two messages are now sent to User.

## Client - Service

**- Step 1:** Now, the user has already received the TGS session key from the authentication server, so that user decrypts the message with its TGS session key, the User can now read the contents of this message and importantly now has a copy of the service session key. The user creates a new Authenticator message containing the user’s ID and a timestamp and encrypts this new Authenticator with the service session key. Note, the user cannot decrypt the service ticket as it was encrypted with the services secret key, so the user simply forwards the service ticket along with newly created Authenticator message to the Service. It’s now the services turned to its work. First, Service also encrypts the Service Ticket with its secret key, this gives the service access to the service session key which the service uses to decrypt the User Authenticator message. Both the service ticket and Authenticator have been decrypted now. The service starts to validate the data contained within. The service first makes sure the sure ID in the service ticket and the Authenticator are matched, compares the timestamps, compares the IP address in the service ticket to the IP address of the user it received the messages from, checks to see that the service ticket has not expired. Then, service checks its cache, just like the TGS, the service maintains a cache of recently received authenticators from users, the service will check its cache to ensure the Authenticator its just received is not already in the cache. This check again provides replayer to protection if the User Authenticator is not already in the cache and the service will added. Now, service needs to create its own Authenticator message which it will send to the user. Similar to User Authenticator message, the Service Authenticator includes the service’s id and timestamp, this Authenticator is encrypted with the service session key and this message is sent to the User.

**- Step 2:** The User now needs to encrypt the message with symmetric service session key, therefore, it will decrypt with the same symmetric key and recall the user was sent this service session key by the TGS. The user now verify that the service name contained within the Authenticator is the service which is the user wants to talk.

# References

<https://en.wikipedia.org/wiki/Needham–Schroeder_protocol>

<https://en.wikipedia.org/wiki/Kerberos_(protocol)>

<https://en.wikipedia.org/wiki/Authentication_protocol>

<https://en.wikipedia.org/wiki/Mutual_authentication>

<https://en.wikipedia.org/wiki/Kerberos_(protocol)>

<https://en.wikipedia.org/wiki/Challenge-Handshake_Authentication_Protocol>

<https://en.wikipedia.org/wiki/NTLM>

<https://en.wikipedia.org/wiki/Secure_Remote_Password_protocol>

<https://secgroup.dais.unive.it/teaching/cryptography/strong-authentication/#:~:text=We%20discuss%20strong%20authentication%20protocols,a%20message%20encrypted%20under%20K>.

<https://www.brainkart.com/article/Remote-User-Authentication-Using-Symmetric-Encryption_8474/#:~:text=Secret%20keys%20Ka%20and%20Kb,Ks%20to%20A%20and%20B>.

<https://www.net.in.tum.de/pub/netsec2016/10_cryptoprot.pdf>

[1] https://ee.stanford.edu/%7Ehellman/publications/24.pdf