Kerberos I - Overview

labs.lares.com/fear-kerberos-pt1

Raúl Redondo March 19, 2024

penetrationtesting

This post, is the first in the series and will aim to provide an overview of the protocol, from its beginnings to the different (ab)use techniques.

Raúl Redondo

Mar 19, 2024 • 13 min read

The three-headed dog is back in business

Kerberos, again

For many years, the Kerberos protocol has been one of the best friends of offensive security professionals. Leveraging this protocol has brought us many wins during assessments and caused many headaches for ‘blue teamers’, who must always be on point and right every time!

That's why, at Lares, we've created a series explicitly relating to the Kerberos protocol and a look at the associated headaches.

Kerberos I - Kerberos Overview (this post).

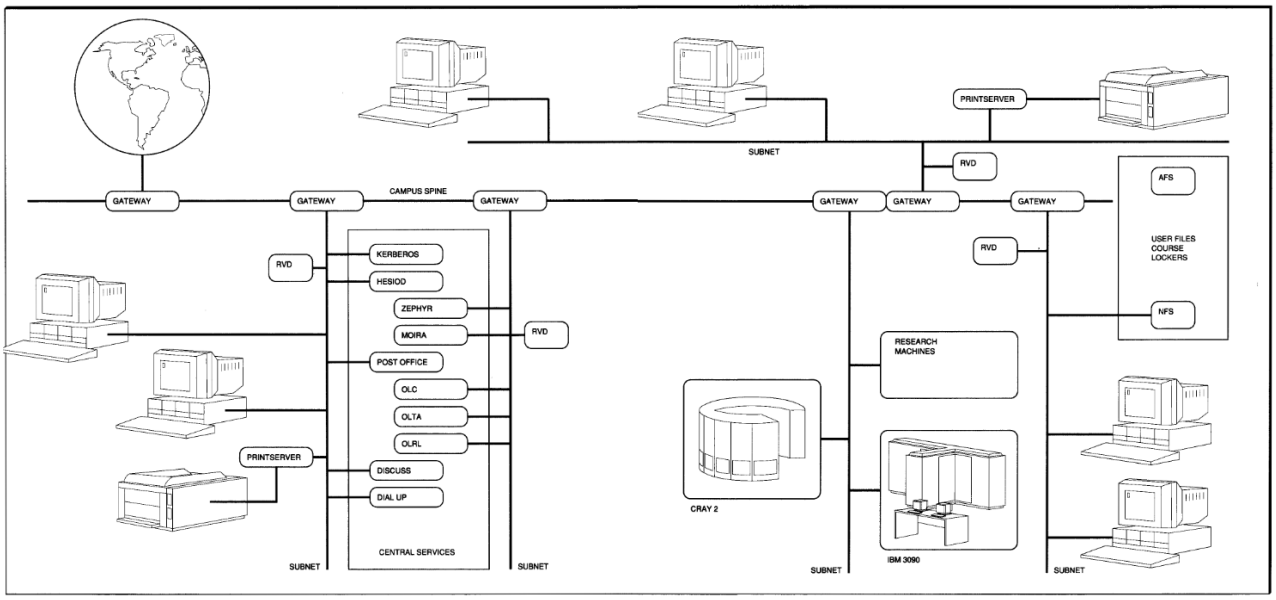
The Kerberos Blog series will include and cover:

With the outline of the series covered, let's get started …

Brief History of Kerberos

The development of the Kerberos protocol began in the **Athena Project** at the Massachusetts Institute of Technology (MIT). This project started in 1983 and aimed to provide cutting-edge computing resources, including networked workstations, high-speed networking, and collaborative software tools, to empower students and researchers.

Below is an example of the Athena computing environment:



research.ibm.com/journal/sj/313/ibmsj3103I.pdf

As shown in the diagram above, the environment consisted of a client-server model of distributed computing using a three-tier architecture.
The environment used, among other features, a name resolution service called Hesiod, instant messaging through the Zephyr service, and Kerberos as a system-wide authentication protocol.

The Kerberos v1 was developed for this project to authenticate service requests between two or more trusted hosts across an untrusted network and **provide a Single Sign-On (SSO) authentication** to these services.

- **Kerberos versions 1** through **3** only operated only within the MIT environment.
- **Kerberos version 4** was released on **January 24, 1989;** however, it was considered insecure due to several cryptographic issues (*its use of DES encryption*).

- **Kerberos v5** was released in **1993 and updated in 2005 to solve the problems of the previous versions. This version added additional encryption types, cross-realm authentication, and** the Generic Security Services Application Program Interface (*GSS-API*).

Kerberos replaced the Windows New Technology LAN Manager (*NTLM*) as the default AUTHENTICATION protocol from Windows Server 2000 onwards; however, Microsoft still supports NTLM for backward compatibility.

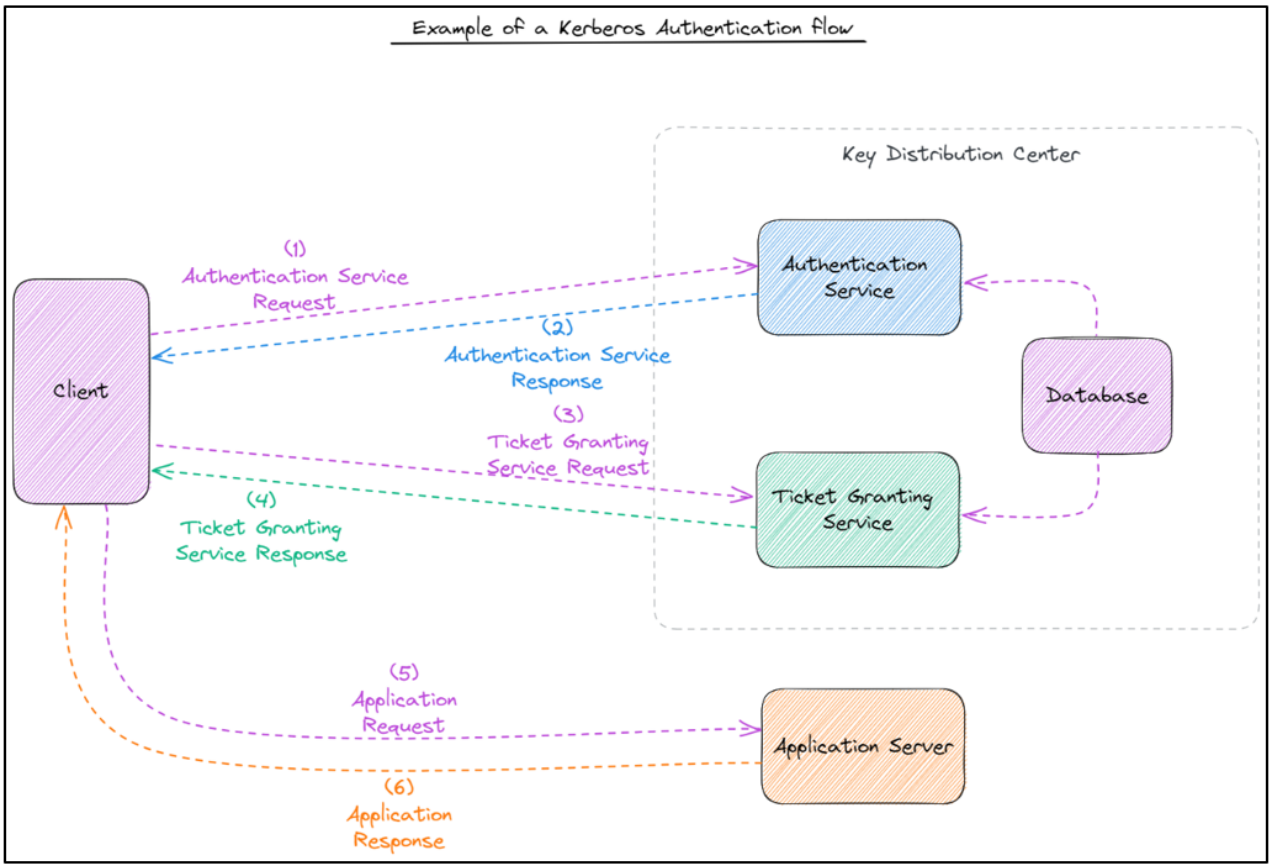
Project Athena Technical Overview video, produced in 1991 with a MIT copyright.

0:00
/1:37
Project Athena Technical Overview video, produced in 1991 with a MIT copyright.

Kerberos 101

The Kerberos authentication protocol outlines how clients (**principals**) engage with a network authentication service and provides a mechanism for mutual authentication between entities before establishing a secure network connection.

Security principals acquire **tickets** from the Kerberos Key Distribution Center (***KDC***) and subsequently present these tickets along with any **authenticators** while establishing a connection. Kerberos tickets serve as the representation of the client's network credentials.
Here’s a basic representation of the Kerberos Authentication flow:



Kerberos authentication flow

Kerberos concepts

Let's walk through some basic concepts in Kerberos:

Realm: a domain or administrative boundary that defines a Kerberos authentication space. Realms are often associated with a specific network or organization. e.g.: *lareslabs.local.*

Principal: unique security entity, such as a user or service identified by a name (**principal name**) within a realm, to which Kerberos can assign a ticket. Principals authenticate themselves to access secured resources using tickets. Some examples of security principals could include a user, a workstation, a server, services (NFS, hosts...).

Principal Names: Unique identifier to each entity within the domain in Active Directory. e.g.:*Elliot.a@Lareslabs.local*

Service Principal Names (SPNs): unique identifier of a service instance. Kerberos authentication uses SPNs to associate a service instance with a service sign-in account. e.g., *CIFS/fs.Lareslabs.local,* MSSQLSvc*/SQL01.Lareslabs.local:1433.*

SPNs are referenced in the servicePrincipalName attribute of the domain machine/user accounts.

Application/Services Servers (AP): A host that provides one or more services over the network.

Key Distribution Center (KDC): Implemented as a domain service and located on a domain controller, its primary function is to securely manage the distribution of encryption keys among authorized principals in a network. KDC provides two services that are started automatically by the domain controller's Local Security Authority (LSA) and run as part of the LSA's process:

- **Authentication Service (AS)**: This service handles the issuing of ticket-granting tickets (TGTs) to connect to the ticket-granting service (TGS) in its domain or other trusted domains. TGTs have an expiration date.
- **Ticket-Granting Service (TGS)**: issues service tickets, allowing authenticated users to access specific network services. Upon presenting a Ticket-Granting Ticket (TGT), the TGS verifies and provides a time-limited service ticket for the requested service.

Tickets: Kerberos tickets are encrypted structures that ensure confidentiality and integrity during transmission. This encryption helps protect sensitive information, and only entities with the appropriate keys (*such as the user, TGS, and target service*) can decrypt and verify the contents of the tickets.

Kerberos tickets are encrypted **using the Kerberos keys derived from the user's password** and additional data.

Note: We will discuss tickets in-depth in the third part of the series, ***Kerberos III— User Impersonation.***

Tickets have an expiration time, although they may be renewable and reusable.

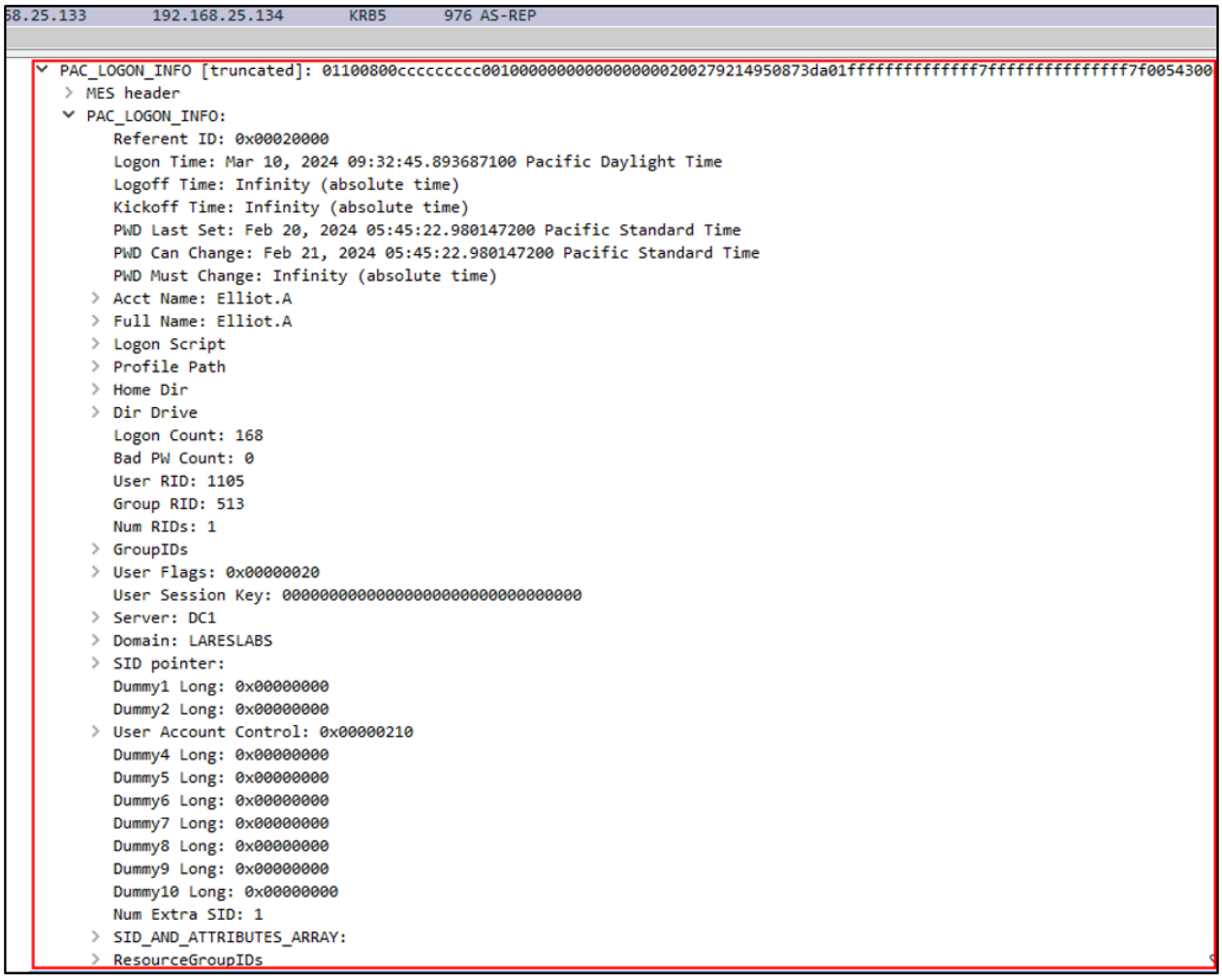
- **Ticket-granting tickets (TGT):** cryptographic structures obtained by a user during the initial authentication process. They are encrypted using the user's long-term secret key and can be used to request service tickets without requiring the user to re-enter their credentials (*allow Single Sign-On*).
- **Service Ticket (ST)**: obtained from the TGS after presenting a valid TGT and specifying the desired service to access. Contains information about the user, the service, and a session key for secure communication.

Authenticators: Data structures created and encrypted by the client with the session keys provided by the Authentication Services (KDC).

Privilege Attribute Certificate (PAC): Kerberos is an **authentication** protocol, not an authorization protocol, this is why this data structure was created, to provide this authorization data for Kerberos extensions.

The PAC is included in nearly every ticket (Authorization Data) that encodes **authorization** information, consisting of group memberships, additional credential information, profile and policy information, and supporting security metadata.

Below is an example of the information that can be found within the PAC, obtained through a network traffic capture:



PAC_LOGON_INFO

When using Kerberos service tickets for server authentication, the Privilege Attribute Certificate (PAC) can be extracted from a user's ticket. This provides the server with information about user privileges without the need to query the domain controller.

Encryption Types

As defined in [***MS-KILE***], The Kerberos V5 protocol supports multiple encryption types, which are the **algorithms for encrypting the tickets or other data**. The Kerberos V5 protocol negotiates which encryption type to use for a particular connection.

Windows **must** support :

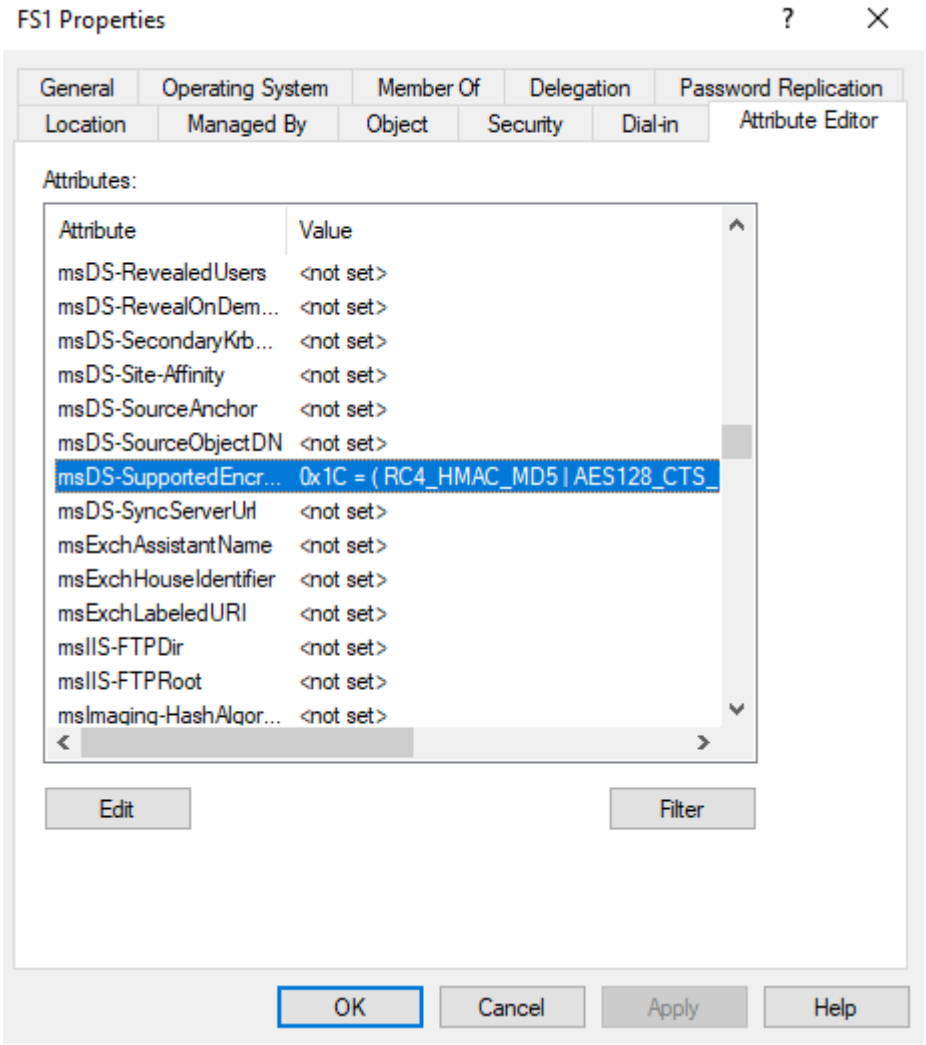
- AES256-CTS-HMAC-SHA1-96 [18]
- AES128-CTS-HMAC-SHA1-96 [17]

And **should** support:

- RC4-HMAC [23]
- DES-CBC-MD5 [3]

- DES-CBC-CRC [1]

The following example shows how to enable the encryption types for a machine account (FS1$) through the msDS-SupportedEncryptionTypes attribute



msDS-SupportedEncryptionTypes attribute

More information on how to set encryption types in Kerberos can be found in the **Microsoft's documentation**.

Wireshark & Kerberos decryption

To better understand what happens within the Kerberos authentication flow, it is possible to provide Wireshark with a key tab file containing Kerberos principal's encryption keys.

To facilitate this, we will need the Kerberos keys of the principals participating in the communication, for example, krbtgt, Elliot.A, and FS1$. To obtain these keys, we can use a number of common tools, such as mimikatz, secretsdump, etc.

```
mimikatz # lsadump::dcsync /domain:lareslabs.local /user:krbtgt
[DC] 'lareslabs.local' will be the domain
[DC] 'DC1.lareslabs.local' will be the DC server
[DC] 'krbtgt' will be the user account
[rpc] Service  : ldap
[rpc] AuthnSvc : GSS_NEGOTIATE (9)

Object RDN           : krbtgt

** SAM ACCOUNT **

SAM Username         : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 2/20/2024 2:21:43 PM
Object Security ID   : S-1-5-21-4164589718-1869447727-3637930069-502
Object Relative ID   : 502

Credentials:
  Hash NTLM: 0b1534c89020c1c44274470a5514f47d
    ntlm- 0: 0b1534c89020c1c44274470a5514f47d
    lm  - 0: d5dbd25f95ecdc27cd735eb2e77e1b2e

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
    Random Value : 83365e8660e82b9a18c23c4753523741

* Primary:Kerberos-Newer-Keys *
    Default Salt : LARESLABS.LOCALkrbtgt
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) : 858d1f9471e97aafc01f3765358e9496a0b8c41856d1e4137b305cbd7537dba8
      aes128_hmac       (4096) : 9ff49197a00e5c89f6af079b7e418c84
      des_cbc_md5       (4096) : c264daa12a2ce99b
```

Mimi dumping krbtgt keys

The next step is to add the keys to the keytab.py script provided by @dirkjanm:

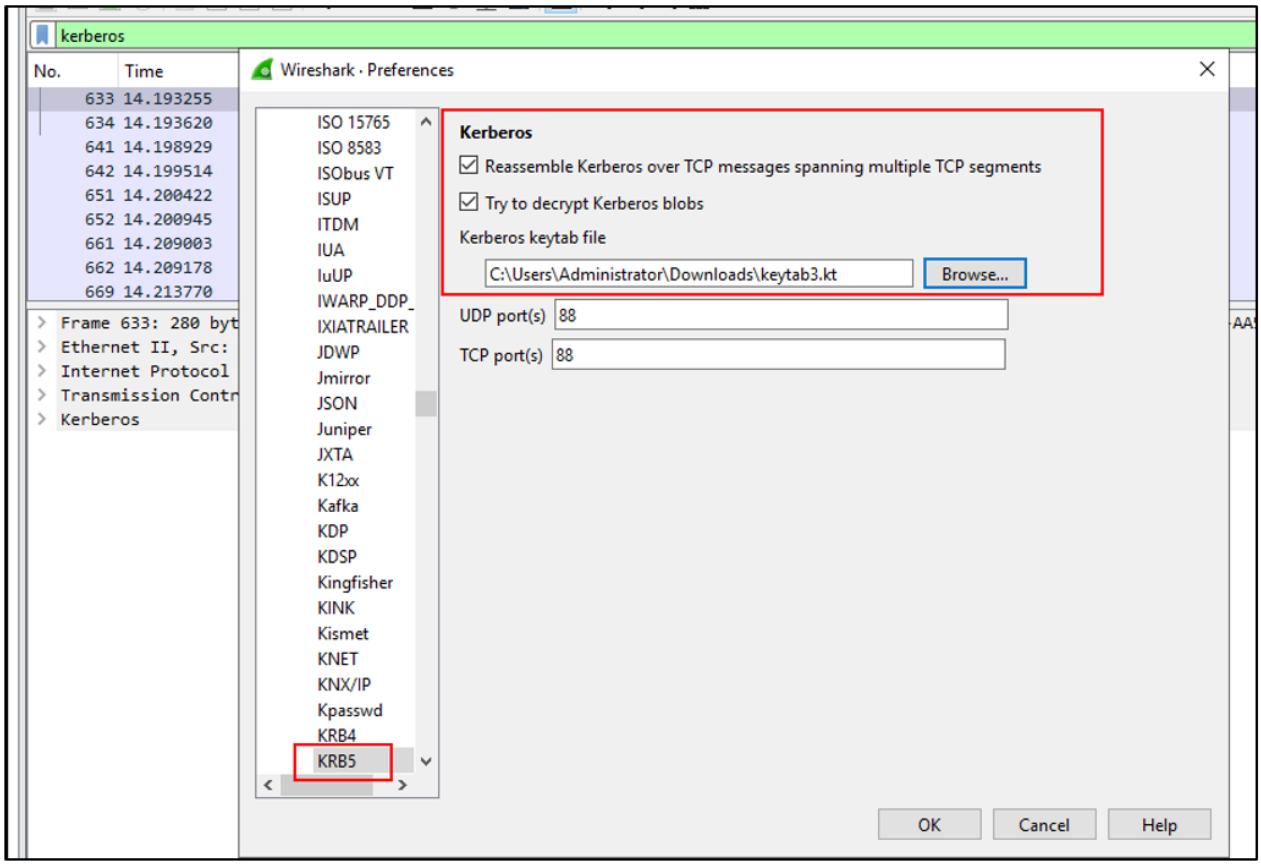
```
# Add your own keys here!
# Keys are tuples in the form (keytype, 'hexencodedkey')
# Common keytypes for Windows:
# 23: RC4
# 18: AES-256
# 17: AES-128
# Wireshark takes any number of keys in the keytab, so feel free to add
# krbtgt keys, service keys, trust keys etc
keys = [
    (23, 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'),
    (18, '858d1f9471e97aafc01f3765358e9496a0b8c41856d1e4137b305cbd7537dba8'),
    (17, '9ff49197a00e5c89f6af079b7e418c84'),
    (18, '663710fa919166efb4a5f0e5c3c50ee9e8674390a37bca3a45c0cf0356c87cb4'),
    (23, 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa')
]

nkt = KeyTab()
nkt.entries = []

for key in keys:
    ktcr = KeyTabContentRest()
    ktcr['keytype'] = key[0]
    ktcr['key'] = binascii.unhexlify(key[1])
    nktcontent = KeyTabContent()
    nktcontent.restfields = ktcr
    # The realm here doesn't matter for wireshark but does of course for a real keytab
    nktcontent['realm'] = b'LARESLABS.LOCAL'
    krbtgt = OctetString()
    krbtgt['value'] = 'krbtgt'
    nktcontent.components = [krbtgt]
    nktentry = KeyTabEntry()
    nktentry['content'] = nktcontent
    nkt.entries.append(nktentry)
```

Keytab.py

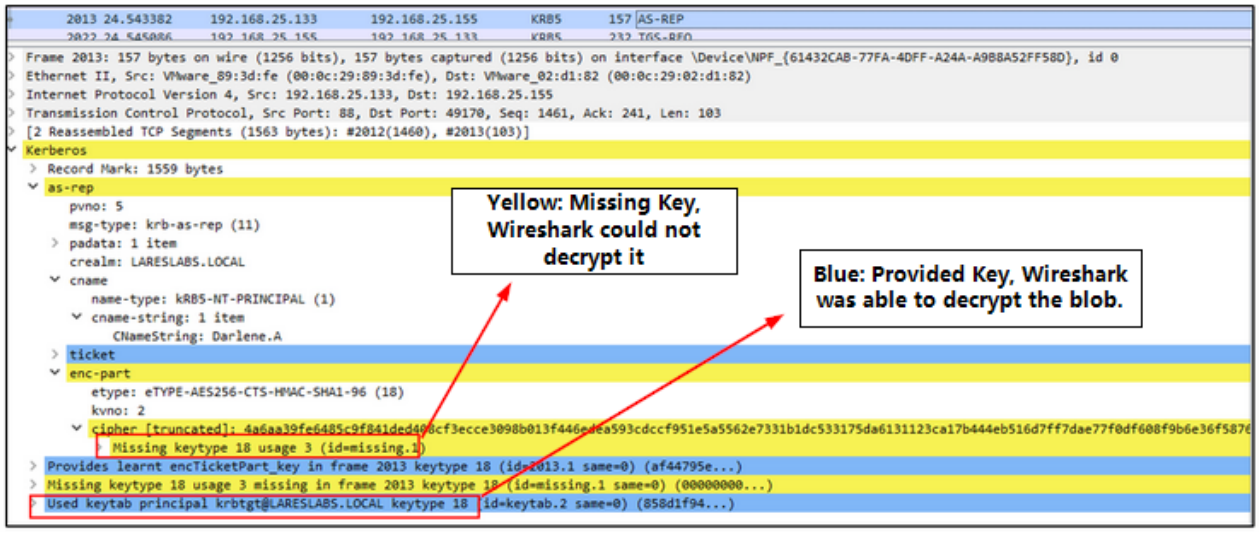
Import the output file from keytab.py to Wireshark (Edit>Preferences>Protocols>Krb5):



Wireshark KRB5 Keytab file

After that, within the Wireshark capture, we should be able to see several things:

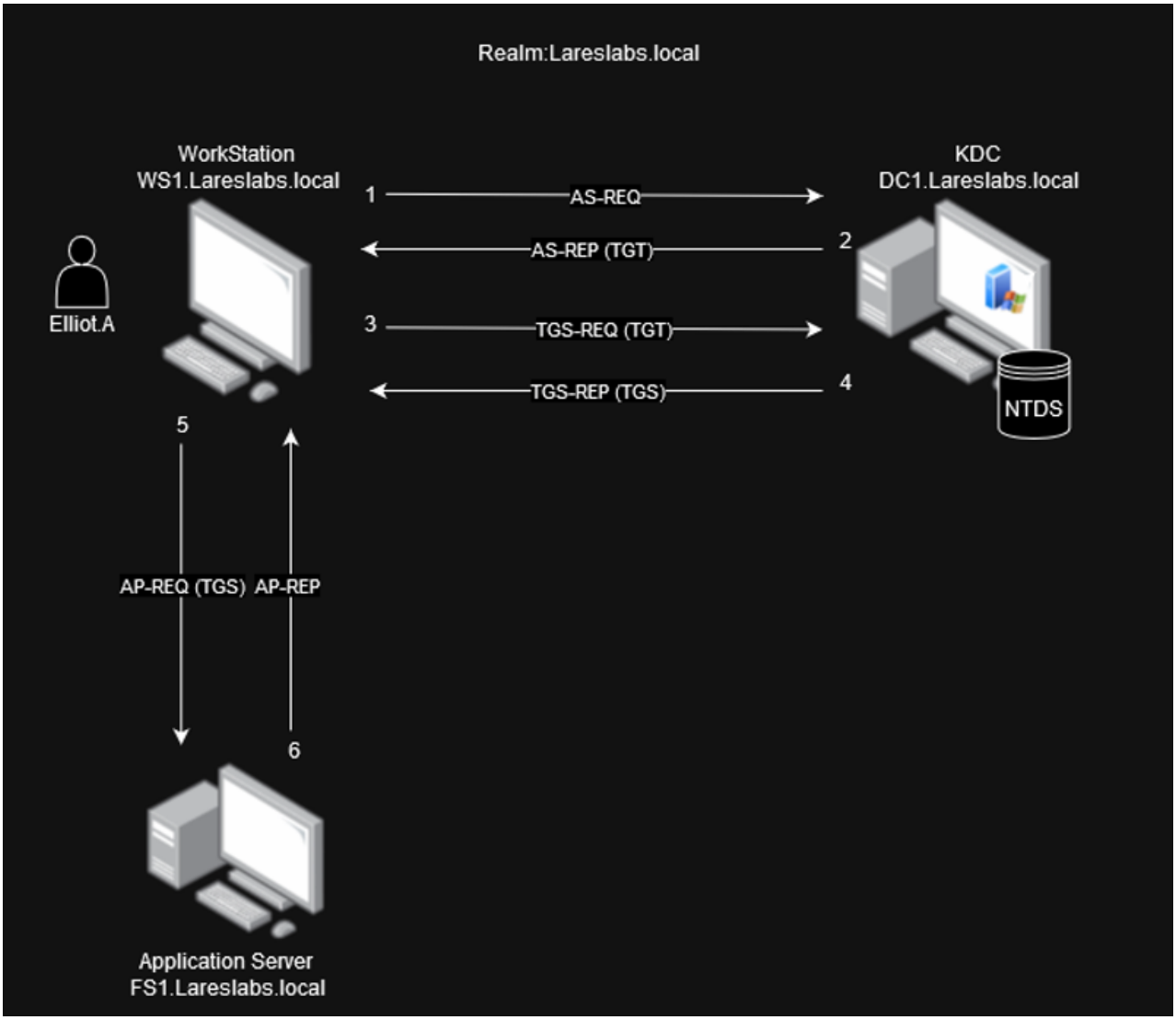
- **Blue highlighted parts,** Wireshark has been able to decrypted information using the Kerberos keys of the different principals.
- **Yellow highlighted parts,** indications of missing key data, resulting the blob not being decrypted.



Wireshark KRB5 Decrypt

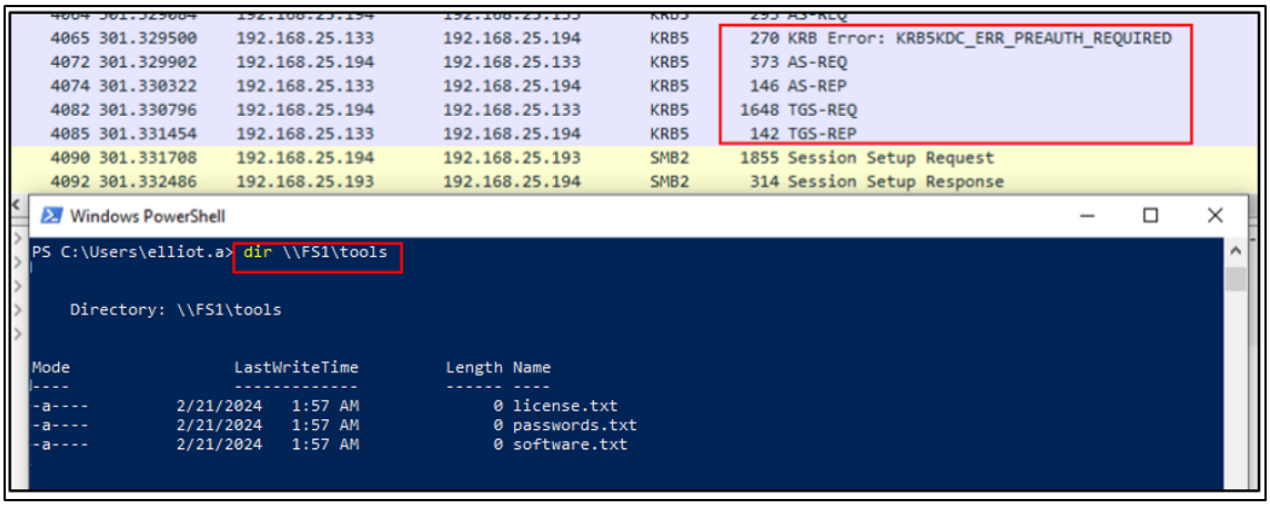
Kerberos Authentication Flow

The image below shows the Kerberos authentication flow, using our example of a client (Elliot.A) requesting access to a shared folder (CIFS service) on the server FS1.Lareslabs.local:



Yes, another Kerberos diagram

In Wireshark, the traffic that is generated when a domain user requests a network resource:



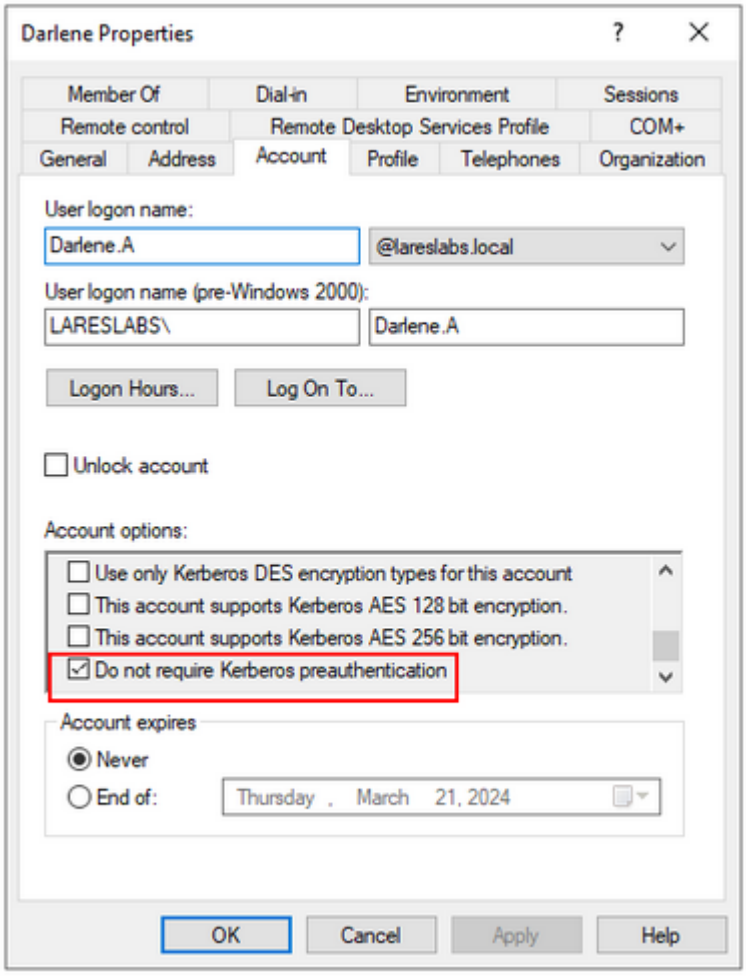
Kerberos auth flow

Kerberos Pre-Authentication

Kerberos supports different types of pre-authentication including password-based and public key cryptography PKINIT. The most common is using a Timestamp, (PA-ENC-TIMESTAMP). This is just the current timestamp encrypted with the client's key.

Pre-authentication in the Kerberos protocol involves the KRB_AS_REQ (Authentication Service Request) and KRB_AS_REP (Authentication Service Reply) messages.

This pre-authentication process is enabled in all accounts by default, but it is possible to disable it with the flag "Do not require pre-authentication":

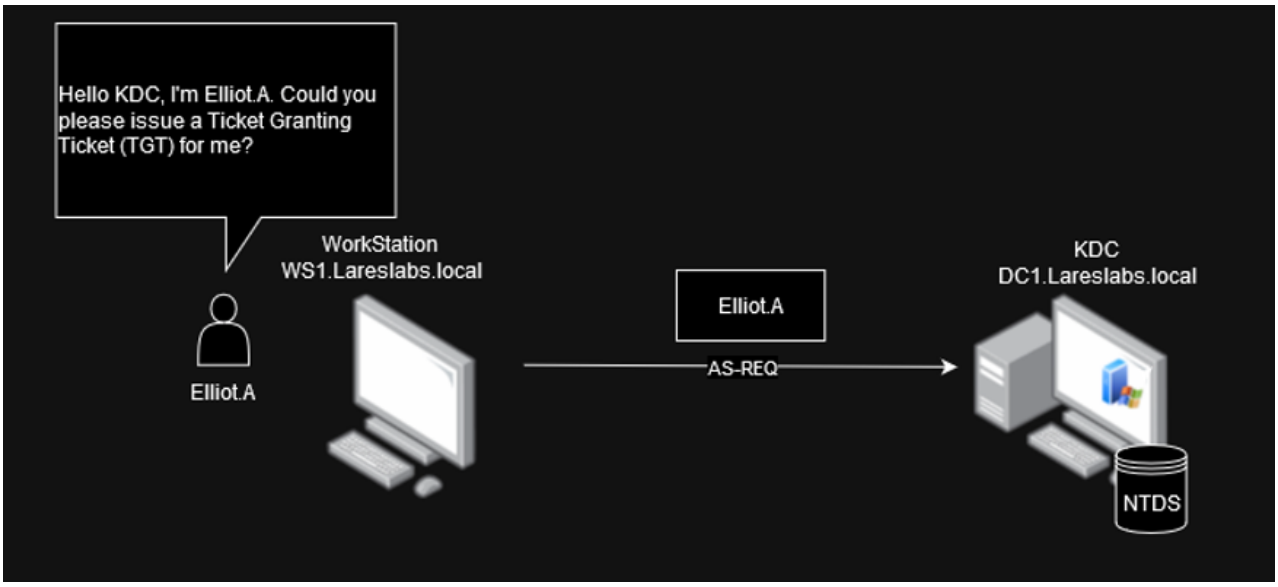


Do not require pre-authentication option

The client will first send the AS-REQ without any pre-authentication data. If the server requires pre-authentication for this principal (**default option)** it will return an error (KRB5KDC_ERR_PREAUTH_REQUIRED) to the client to indicate that pre-authentication is expected.

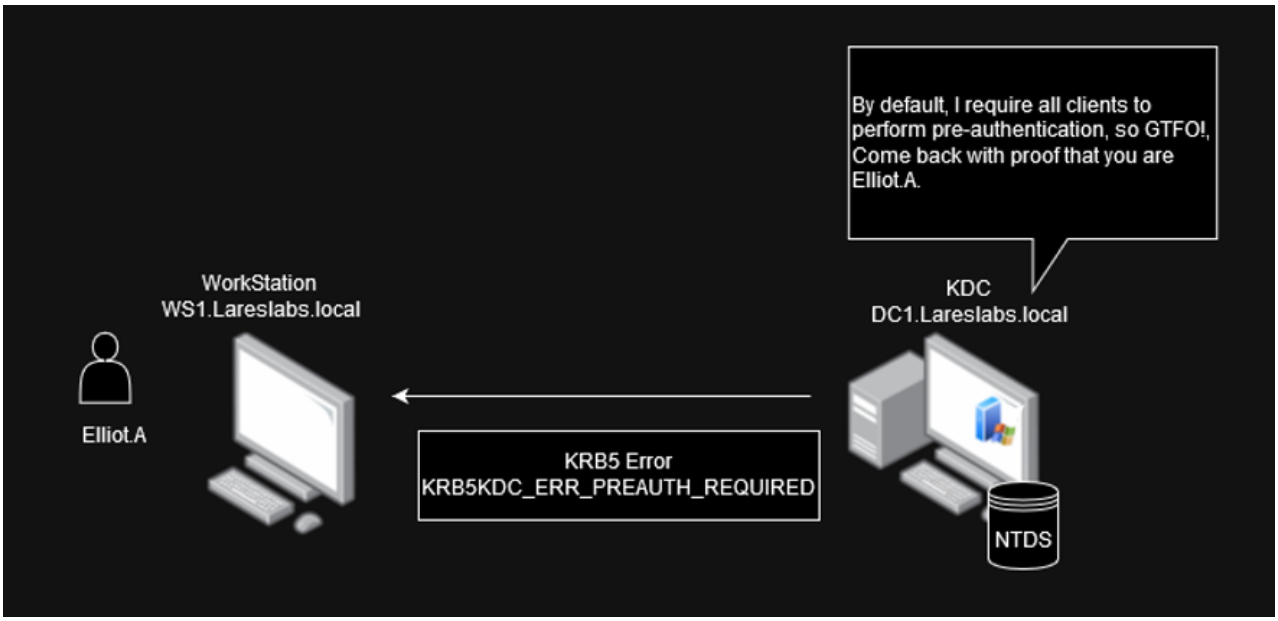
If you find an account with this option enabled, you can obtain an AS-REP message without knowing the client's secret key for that account (ASREProasting). We will discuss this and other 'Roasting' techniques in the second part of the series: Kerberos II—*Credential Access.*

In the following scenario, Elliot.A., a domain user of the Lareslabs.local domain, wants to start the negotiation process with Kerberos. First, they send an AS-REQ message with his username (cname) to the authentication server (KDC):

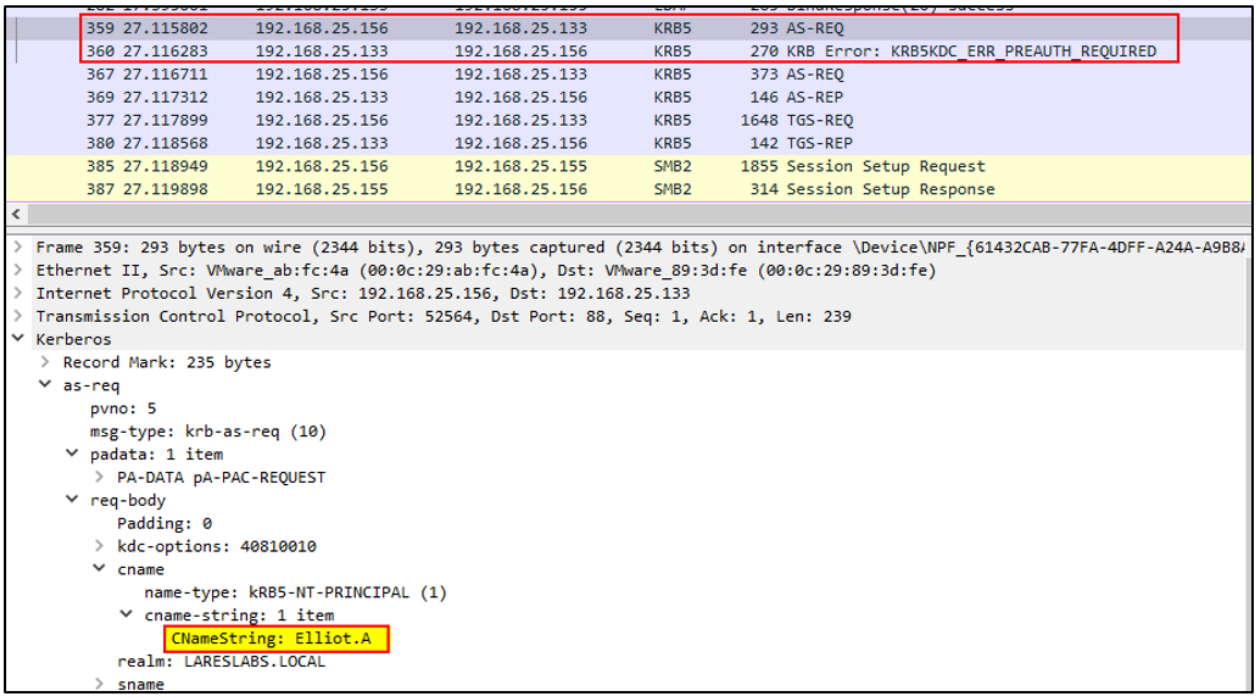


AS-REQ without pre-authentication data

By default, all domain accounts have pre-authentication enabled; as such the KDC will return an error (KRB5KDC_ERR_PREAUTH_REQUIRED):



The following image illustrates an initial AS-REQ request from the client without pre-authentication data, along with the response from the KDC:



Kerberos Pre-authentication flow

It is possible to use these error messages to confirm that a user account exists. We will cover this technique in depth in the post: ***Kerberos II - Credential Access***.

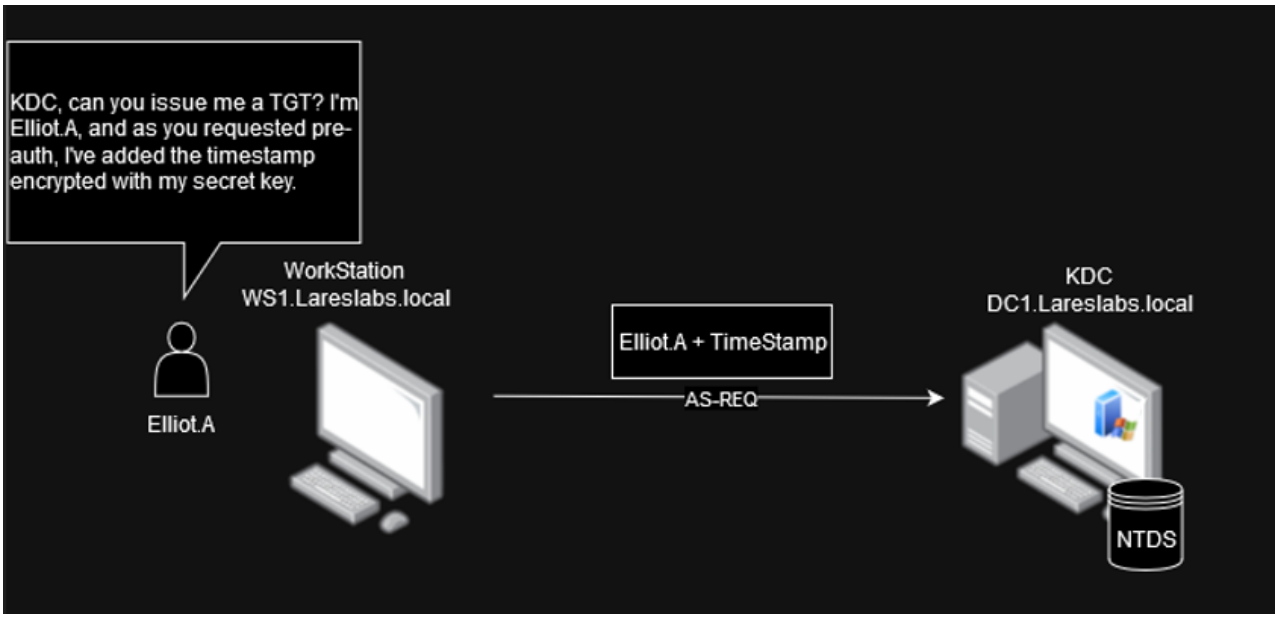
Once the KDC confirms that the client requires pre-authentication, the client must send a second AS-REQ message with more data to authenticate against the KDC.

AS-REQ (Authentication Service Request)

The client requests a TGT from the authentication service (AS) at this stage. To do this, the client must send an AS-REQ message.

This message will include, but are not limited to, the following data:

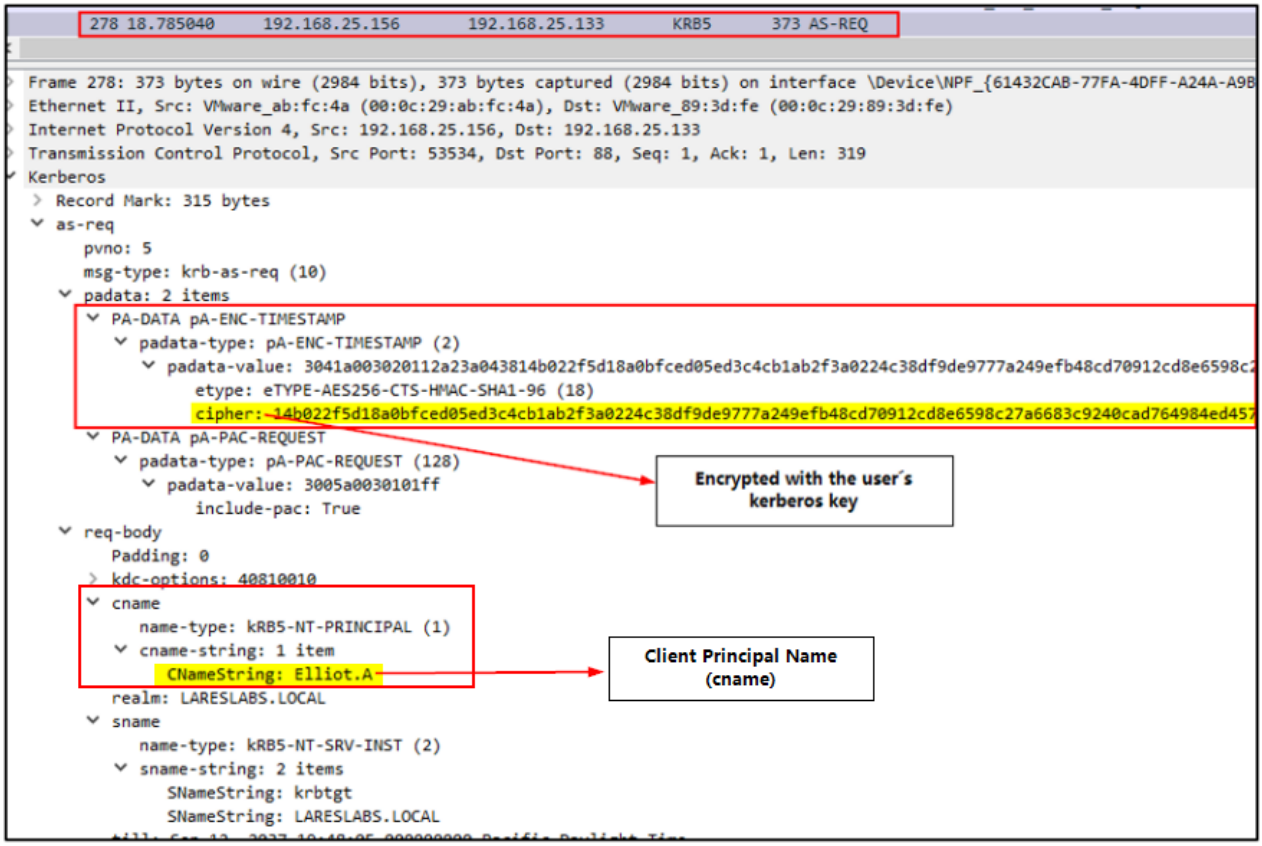
- **Principal Name** (cname):
- **Pre-authentication information** (PA-ENC-TIMESTAMP): Timestamp encrypted with the client's key).



AS-REQ with pre-authentication data

It is essential to know that the encrypted timestamp is only necessary if the user requires pre-authentication, as we have already discussed, is the **default option** unless the user has the DONT_REQ_PREAUTH flag enabled.

The following image below depicts an example of the AS-REQ message captured using Wireshark:

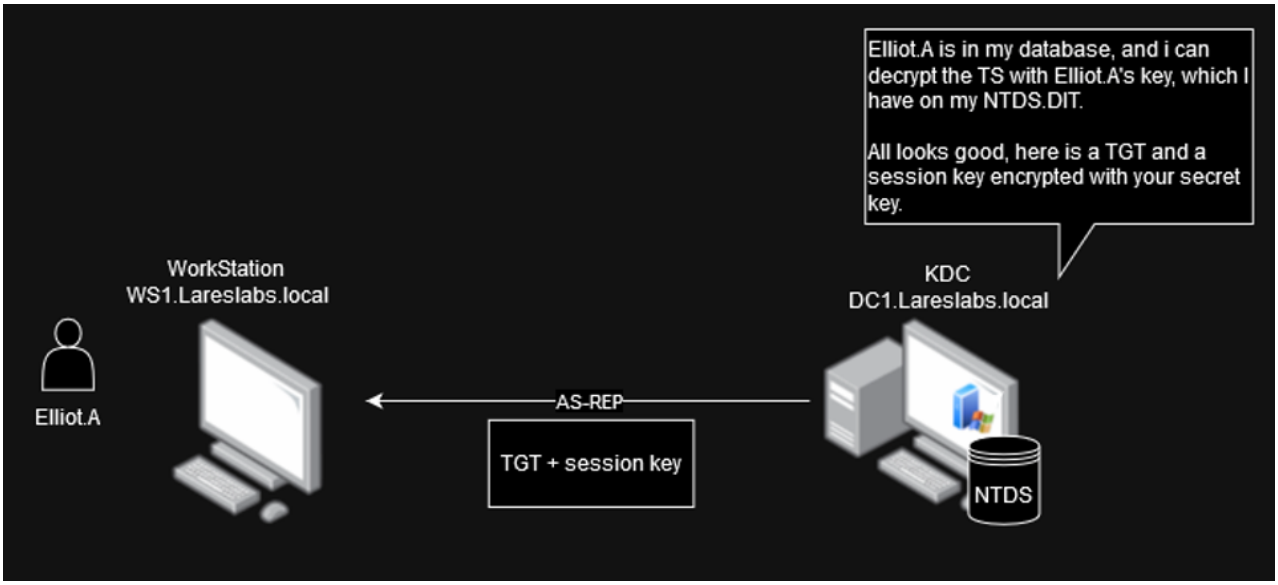


AS-REQ

In the message body, it is possible to find in the padata field, the TimeStamp encrypted with the client's secret key (pA-ENC-TIMESTAMP).

AS-REP (Authentication Service Response)

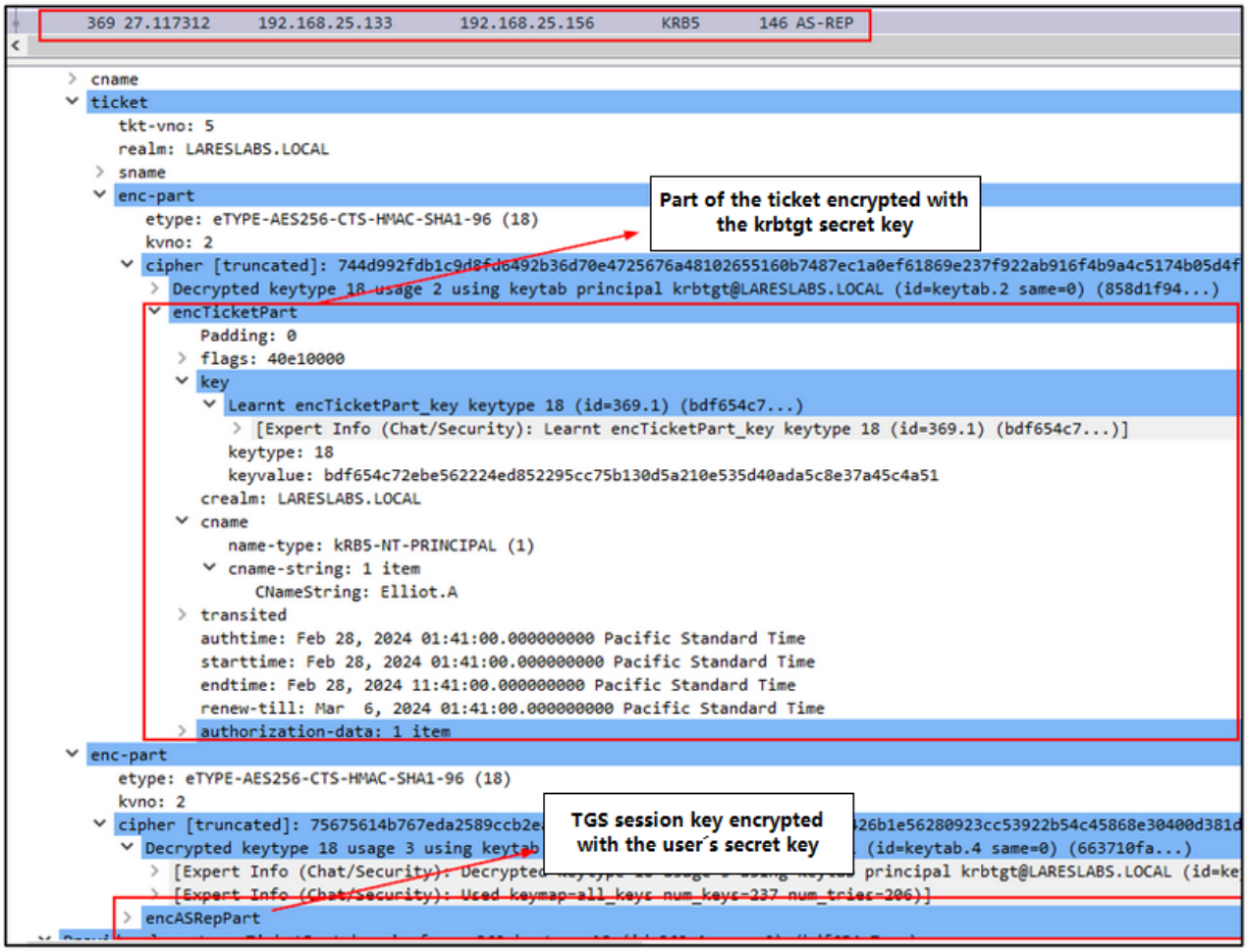
The KDC attempts to decrypt the Timestamp sent by the client in the previous AS-REQ message using the client's secret key. In case of pre-authentication failure, e.g. if the client does not provide a correct password, an error message KDC_ERR_PREAUTH_FAILED will be returned.



AS-REP - KDC response

This message contains, among other data:

- **Ticket Granting Ticket (TGT)**, includes the session key encrypted with the ticket granting service secret key (krbtgt secret key), user name, expiration date, and the PAC (provide SIDs, RIDs, user information, password credentials, used for smart card authentication, S4U data..).
- **Encrypted data:** The session key is encrypted with the user´s secret key.

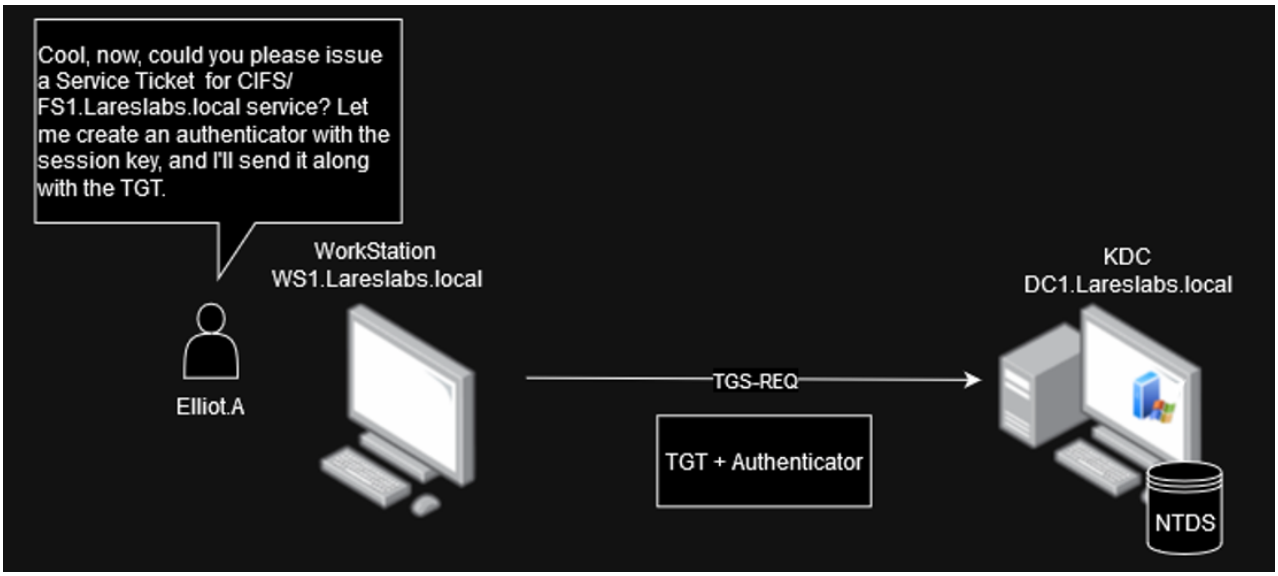


Wireshark AS-REP

TGS-REQ (Ticket-Granting Service Request)

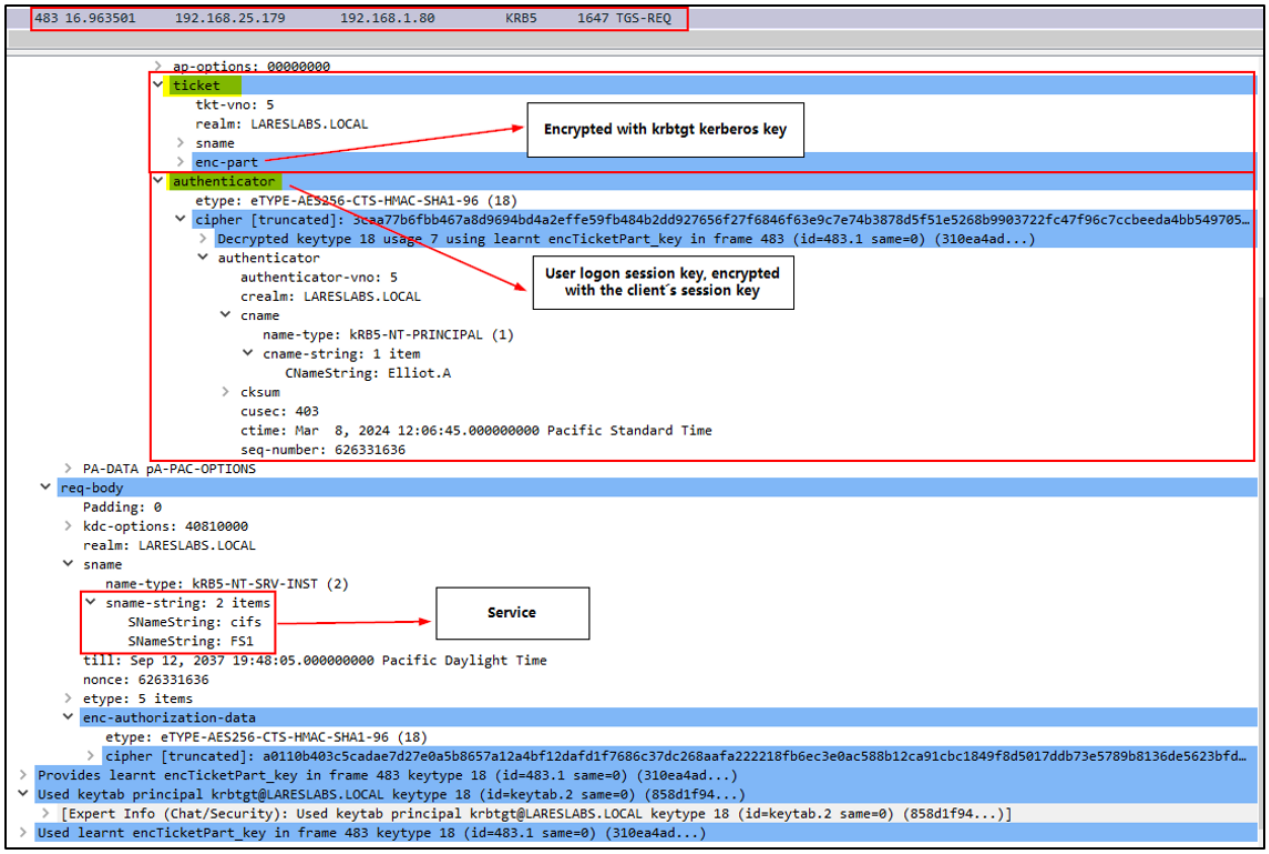
The client sends a TGS-REQ message requesting a service ticket (ST) to access a specific service. The client includes, along with other data:

- **Ticket-Granting Ticket (TGT)** with an "enc-part" encrypted with the krbtgt Kerberos key. It also includes the PAC.
- **Authenticator:** Encrypted with the user's login session key.
- **Service Principal Name (SPN)** of the requested service.



TGS-REQ

The Wireshark capture below identifies that we have found a TGS_REQ message. In the PA-TGS-REQ block, we can see the Ticket Granting Ticket (TGT) and the authenticator sent by the client (Elliot.A). Additionally, in the 're-body' blob, we can identify the service name for which access is being requested (CIFS/FS1):

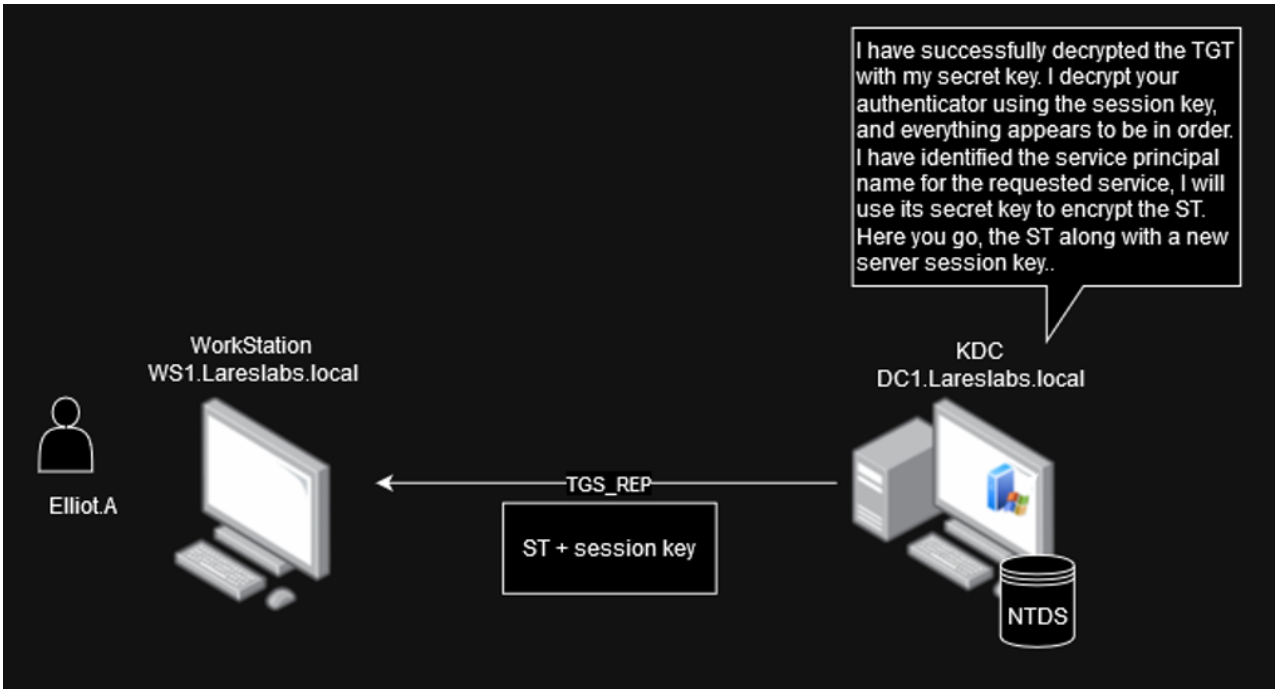


TGS-REQ

TGS-REP (Ticket-Granting Service Response)

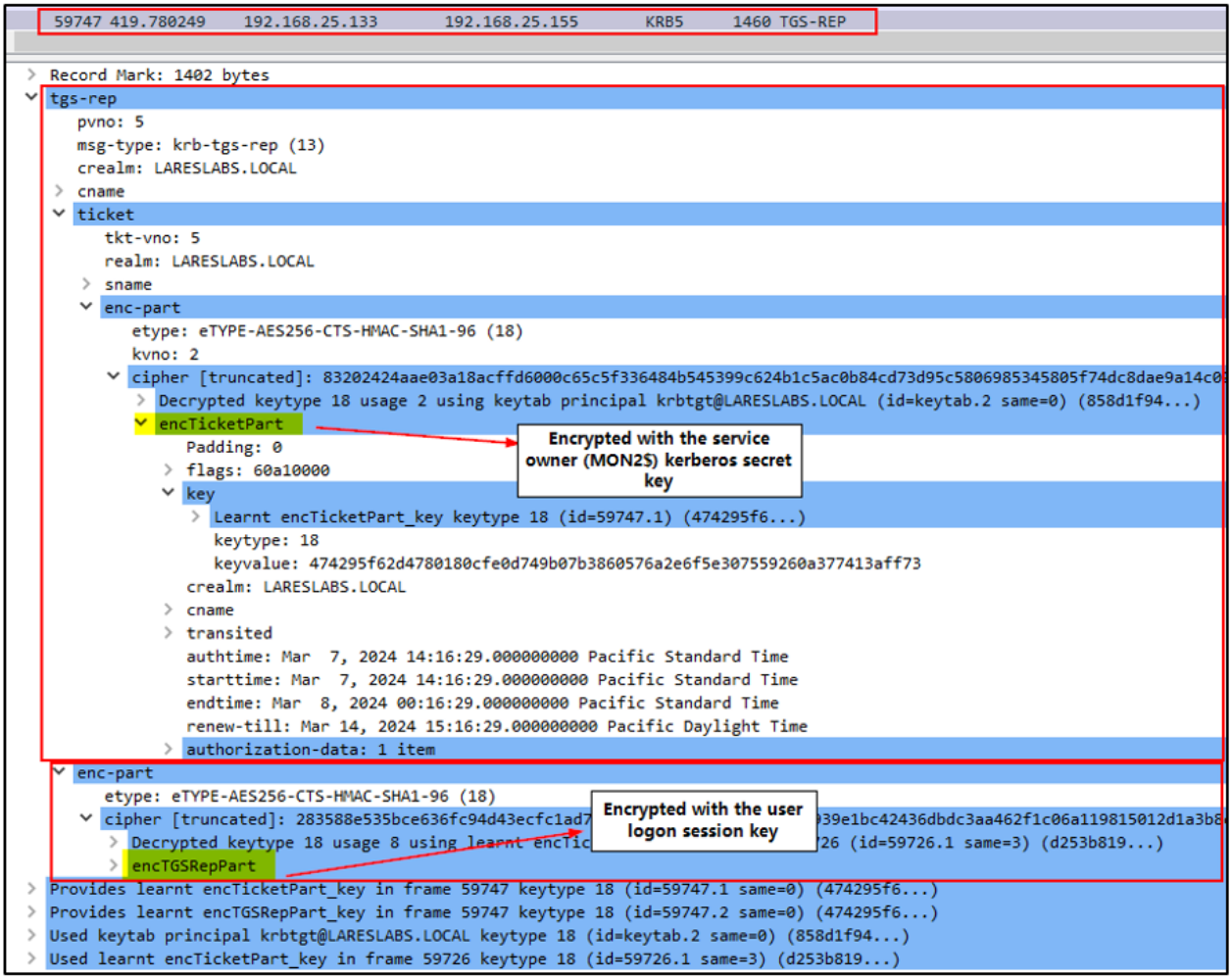
After receiving the TGS-REQ message and verifying the validity of its Ticket-Granting Ticket (TGT), the KDC returns a response through a TGS_REP message with:

- **Service Ticket (TGS)**, encrypted with the service owner (FS1$) Kerberos key.
- **Encrypted data (EncTGSRepPart)** is encrypted with a copy of the service session key and the user logon session key.



TGS-REP

The image below depicts a TGS-REP message, along with key points to note:

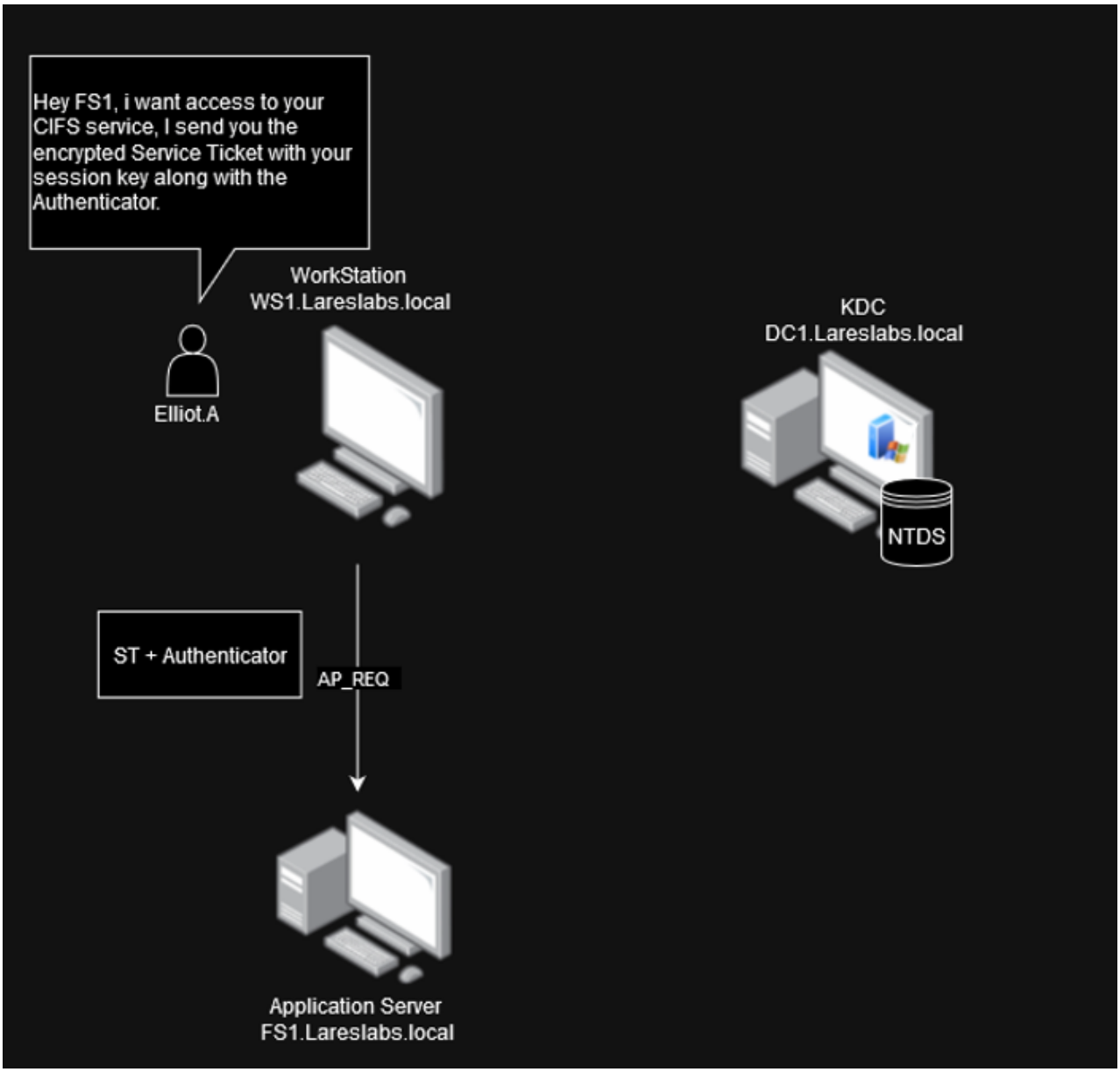


Wireshark TGS-REP

AP-REQ (Application Server Request)

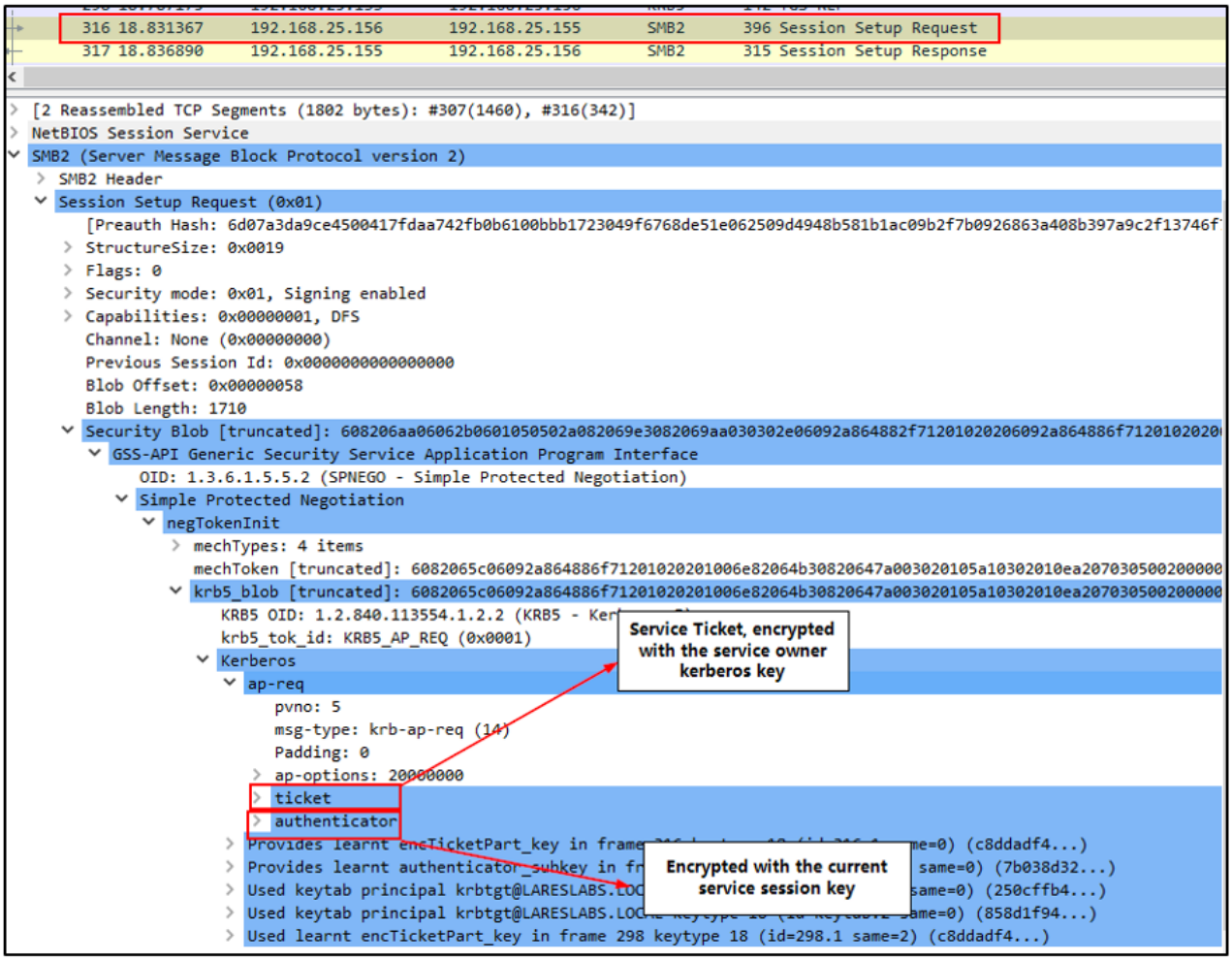
The client forwards a service request message to the intended service, with the following data, among other information:

- **Service Ticket (ST):** Encrypted with the server kerberos key (FS1$).
- **Authenticator**: Timestamp encrypted with the current service key.



AP-REQ

Here is a snippet of the related network traffic:



Wireshark AP-REP

If the PAC validation is enabled, which isn't by default, the AP will verify the PAC against the KDC. Also, if mutual authentication is needed, it will respond to the user with a KRB_AP_REP message.

Optional - KERB_VERIFY_PAC [Need to enable Kerberos PAC validation]:

PAC validation is the procedure of confirming the authenticity of a user's PAC during authentication with a server. The AP will verify the PAC included in the TGS against the KDC. If mutual authentication is required, it will respond with an AP_REP message to the user.

This PAC validation can be enabled in the registry key ValidateKdcPacSignature ("ValidateKdcPacSignature"=dword:00000001), in the path:

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\Kerberos\Parameters].

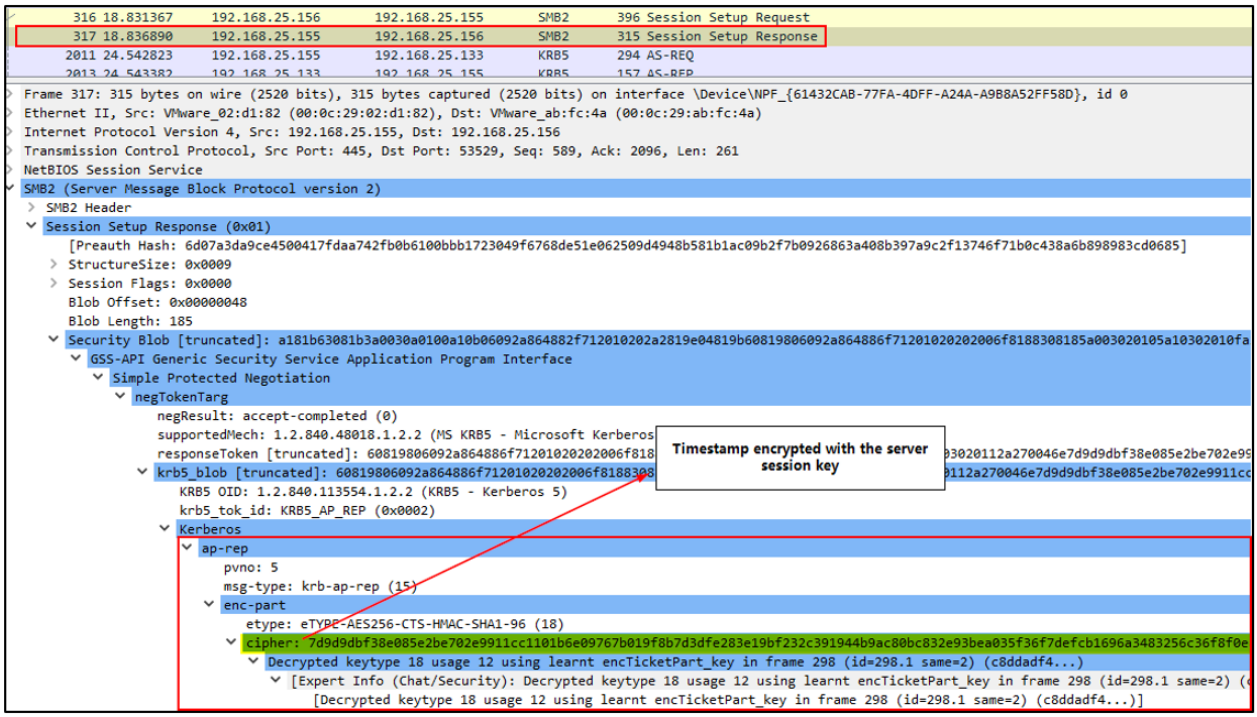
By default, the value of this entry is 0, so Kerberos does not perform PAC validation on a process that runs as a service.

Optional - RPC status code (PAC Verify Response): The domain controller verifies the signature on the response and returns the result to the server. When the method completes, the server operating system MUST examine the return code to determine if

the PAC contents have been altered. The server operating system MUST treat any nonzero return code as a failure.

AP-REP (Application Server Response)

If mutual authentication is required in the AP_REQ request (ISC_REQ_MUTUAL_AUTH request flag), the server must respond with a timestamp encrypted with its own session key:



(OPTIONAL) AP_REP

Once the client receives it and can decrypt it correctly, it will verify that the server is legitimate. If it does not receive a valid AP_REP, the client can confirm that it is a fake/rogue server and stop the communication.

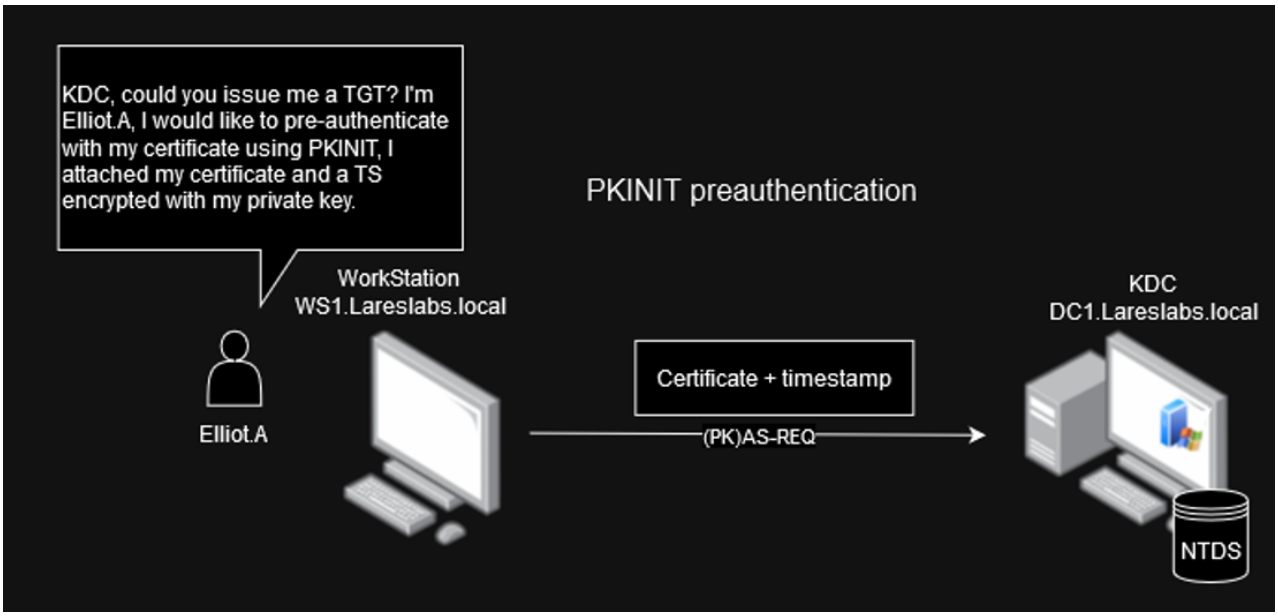
Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)

As described in the MIT documentation, PKINIT is a pre-authentication mechanism for Kerberos 5 that uses X.509 certificates to authenticate the KDC to clients and vice versa. PKINIT can also enable anonymity support, allowing clients to communicate securely with the KDC or with application servers without authenticating as a particular client principal.

PKINIT makes use of the following new pre-authentication types: PA_PK_AS_REQ and PA_PK_AS_REP.

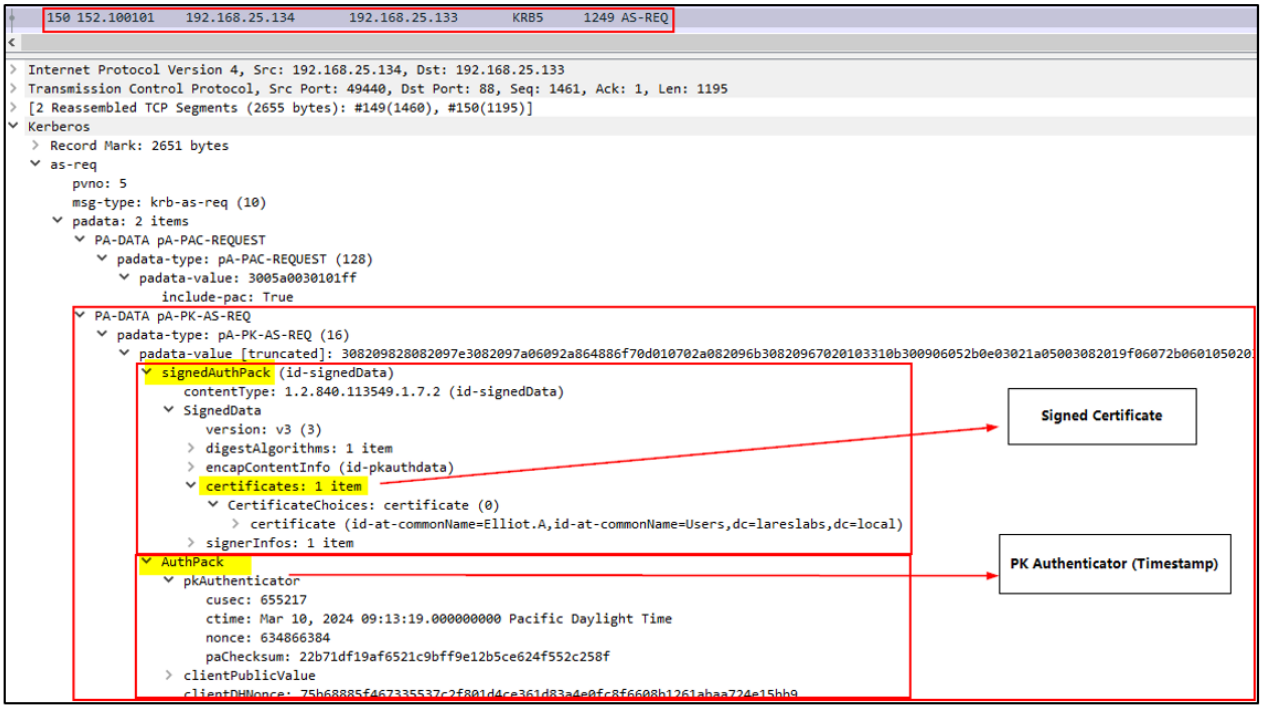
In the context of using PKINIT, the communication flow would be as follows:

The client (Elliot.A) uses its X.509 certificate (signed by the Certificate Authority) and an authenticator (timestamp encrypted with the client´s private key):



PKINIT pre-authentication

Below is the PK-AS-REQ request in Wireshark, including the certificate and timestamp sent by the client:



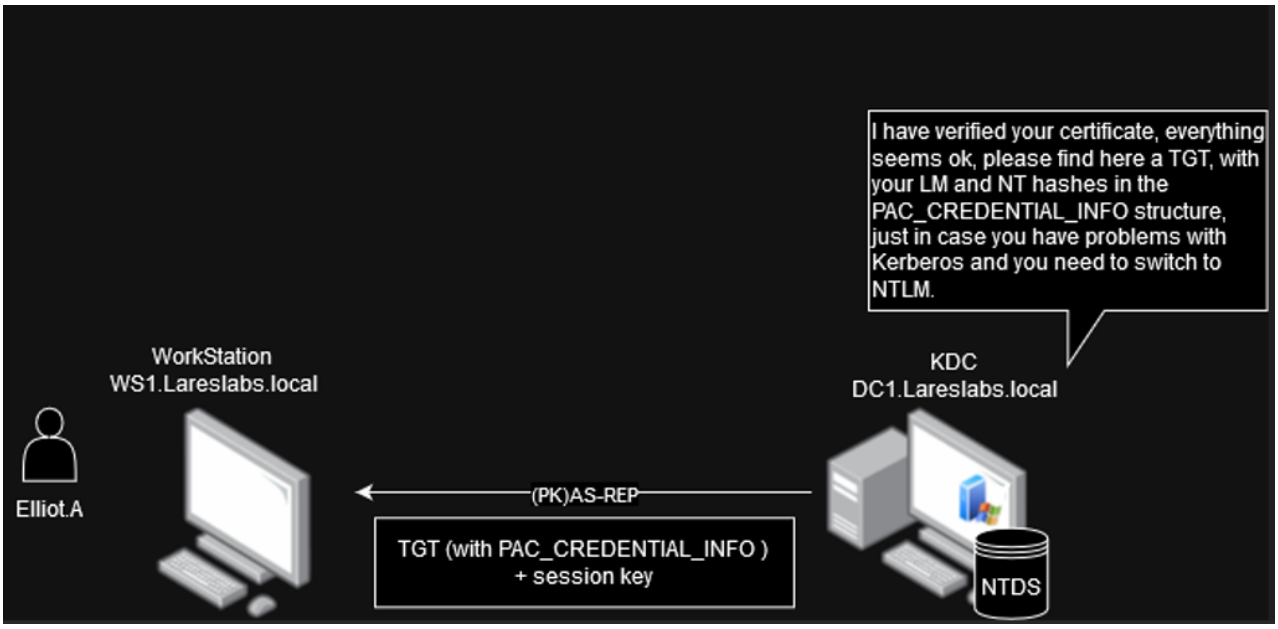
PK-AS-REQ

The authentication server (AS) verifies the certificate, signature and client binding.

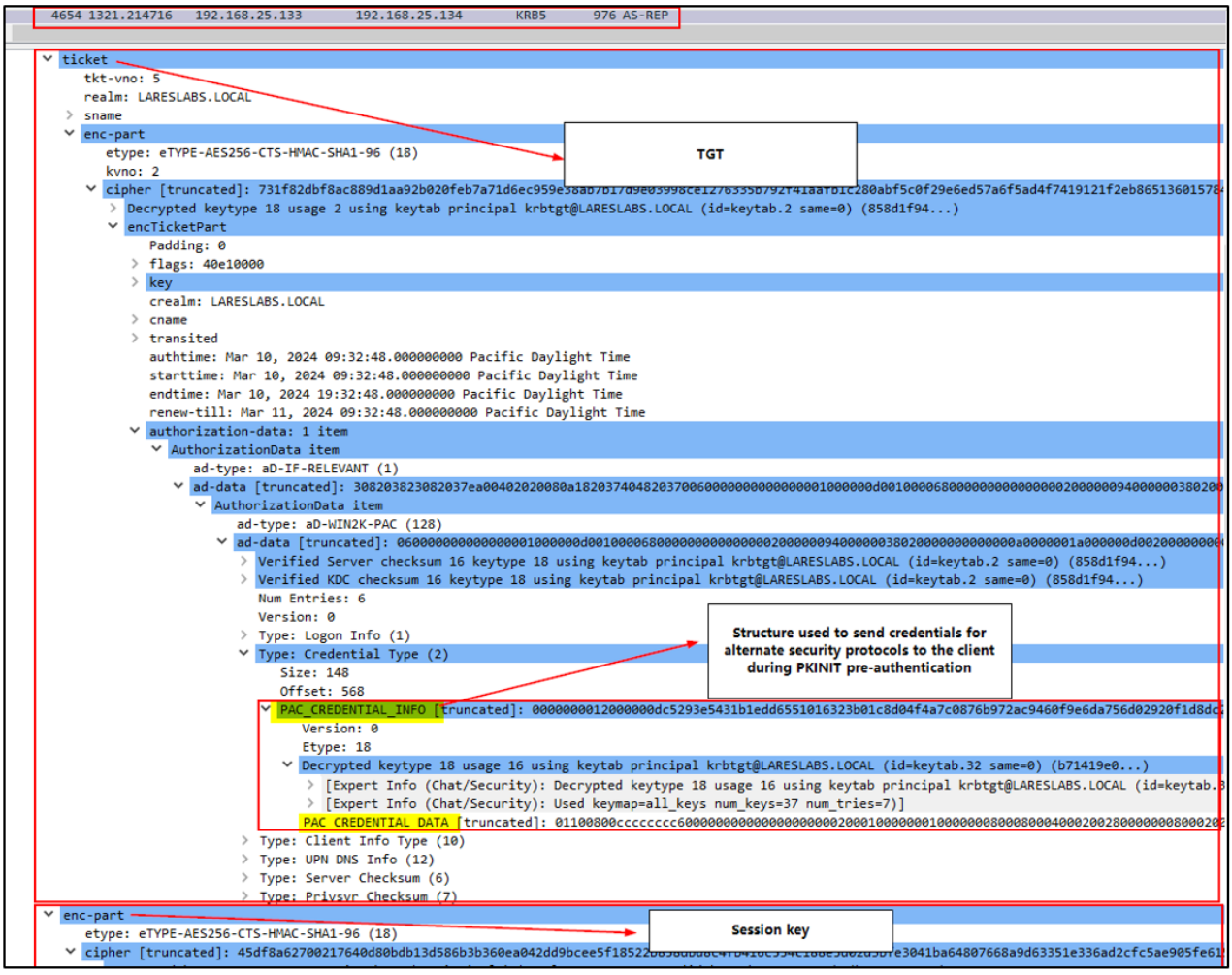
If everything is OK, the KDC responds with an TGT encrypted with the krbtgt´s private key and a session key.

In addition, **KDC includes the user's LM and NT hash in the PAC (PAC_CREDENTIAL_INFO structure) of the TGT.** This feature will allow the client to switch to NTLM if the remote host does not accept Kerberos.

Since the PAC is embedded within the TGT, which is encrypted with one of the Kerberos keys for the krbtgt account, extracting the PAC is currently impossible.



PK-AS-REP



Wireshark PK-AS-REP

Due to this, it is possible to use the User to User (U2U) extension to request a TGS ticket with PAC encrypted with the TGT´s session key instead of the krbtgt key and get the user's hash.

In the next post of the series, we will delve deeper into the UnPAC the Hash technique to explore this process more comprehensively.

Wrapping things up ...

In the first part of the Kerberos series, we've set the groundwork for the following parts of the series, covering the following:

- An overview of Kerberos.
- The concepts.
- Encryption types.
- The Kerberos Authentication Flow.
- Public Key Cryptography for Initial Authentication (PKINIT).

This concludes our first approach to Kerberos. In the next post, we will explore different (ab)use techniques and discuss ticket and delegation management in depth. The following posts will be published in the future, and their posts can be found linked below.

- Kerberos II - Credential Access.
- Kerberos III - User Impersonation.
- Kerberos IV - Delegations.

Resources

- Attl4s - You do (not) Understand Kerberos.
- Looking back at Project Athena.
- Microsoft - Kerberos Authentication Overview.
- MIT - Kerberos papers and documentation.
- GoogleProjectZero - Kerberos Relay Requirements.
- Maksim Chudakov - I understand.
- Kerberos PKINIT: what, why, and how (to break it).
- Tarlogic - Kerberos