

# Kerberos I - Overview

---

 [labs.lares.com/fear-kerberos-pt1](https://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).

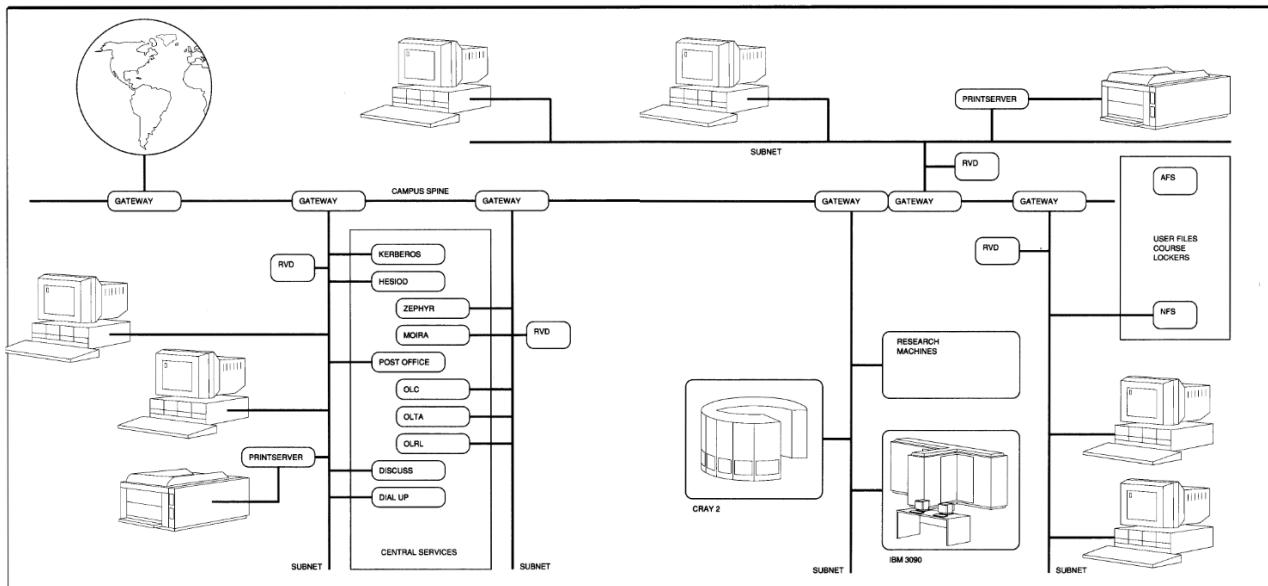
The Kerberos Blog series will include and cover:

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/ibmsj3103l.pdf](http://research.ibm.com/journal/sj/313/ibmsj3103l.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.

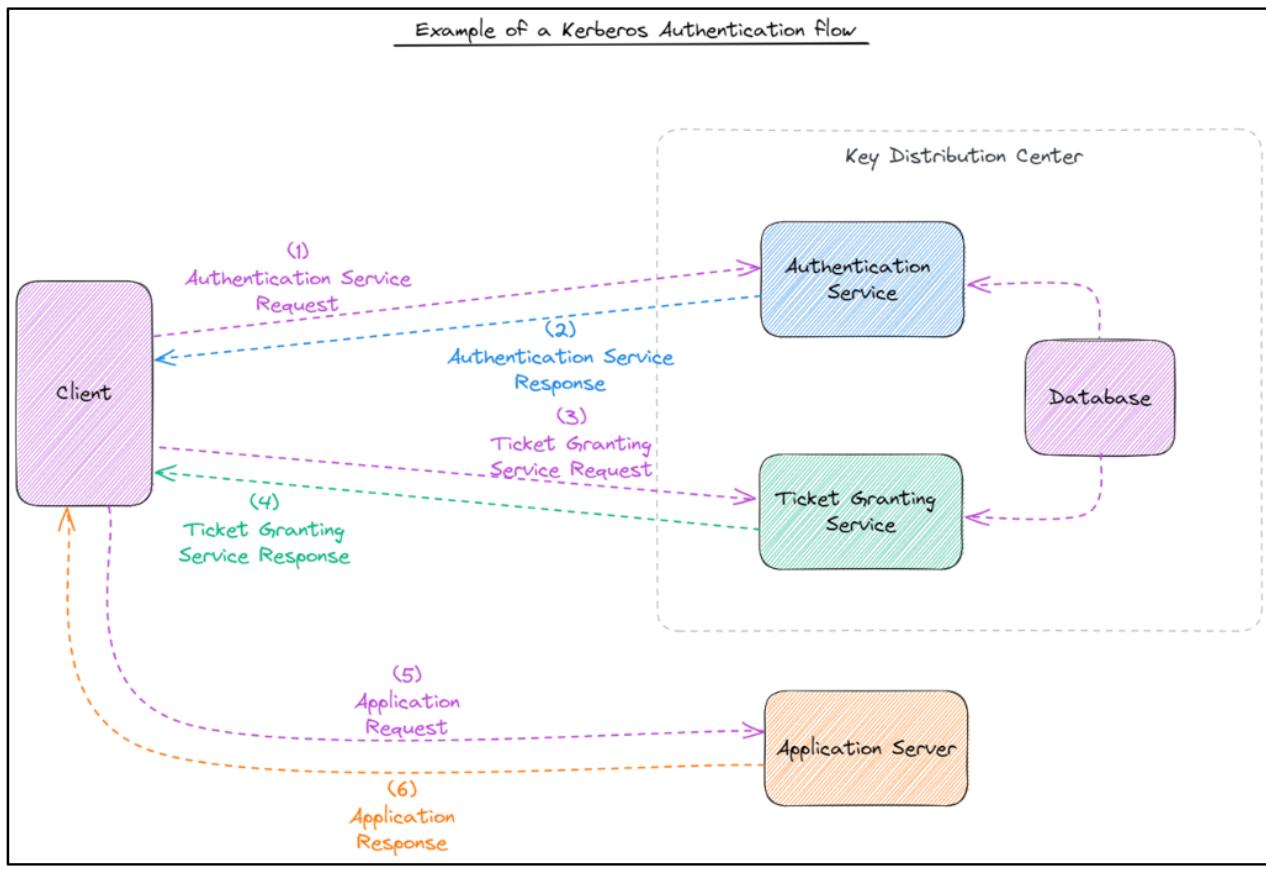
## Kerberos 101

---

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

The screenshot shows the 'FS1 Properties' dialog box with the 'Attribute Editor' tab selected. The 'Attributes' table lists several attributes and their values. The 'msDS-SupportedEncryptionTypes' attribute is highlighted with a blue selection bar, showing its value as '0x1C = ( RC4\_HMAC\_MD5 | AES128\_CTS )'. Other attributes listed include msDS-RevealedUsers, msDS-RevealOnDemand, msDS-SecondaryKrb..., msDS-Site-Affinity, msDS-SourceAnchor, msDS-SourceObjectDN, msDS-SyncServerUrl, msExchAssistantName, msExchHouseIdentifier, msExchLabeledURI, msIIS-FTPDir, msIIS-FTPRoot, and msImaging-HashAlg... . At the bottom of the dialog box are 'Edit', 'Filter', 'OK', 'Cancel', 'Apply', and 'Help' buttons.

Attribute	Value
msDS-RevealedUsers	<not set>
msDS-RevealOnDemand	<not set>
msDS-SecondaryKrb...	<not set>
msDS-Site-Affinity	<not set>
msDS-SourceAnchor	<not set>
msDS-SourceObjectDN	<not set>
<b>msDS-SupportedEncr...</b>	<b>0x1C = ( RC4_HMAC_MD5   AES128_CTS )</b>
msDS-SyncServerUrl	<not set>
msExchAssistantName	<not set>
msExchHouseIdentifier	<not set>
msExchLabeledURI	<not set>
msIIS-FTPDir	<not set>
msIIS-FTPRoot	<not set>
msImaging-HashAlg...	<not set>

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, 'aaaaaaaaaaaaaaaaaaaaaaaaaaaa'),
    (18, '858d1f9471e97aafc01f3765358e9496a0b8c41856d1e4137b305cbd7537dba8'),
    (17, '9ff49197a00e5c89f6af079b7e418c84'),
    (18, '663710fa919166efb4a5f0e5c3c50ee9e8674390a37bca3a45c0cf0356c87cb4'),
    (23, 'aaaaaaaaaaaaaaaaaaaaaaaaaaaa')
]

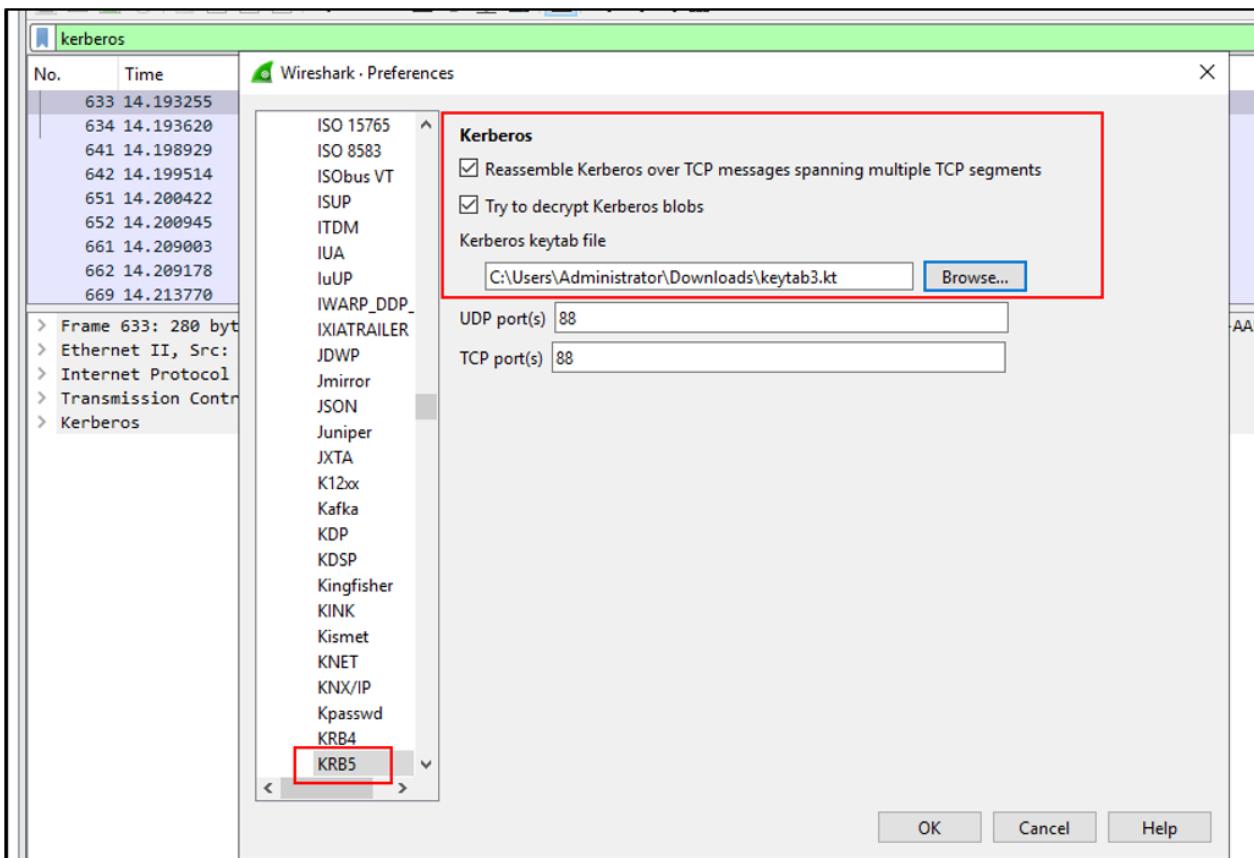
nkt = KeyTab()
nkt.entries = []

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:

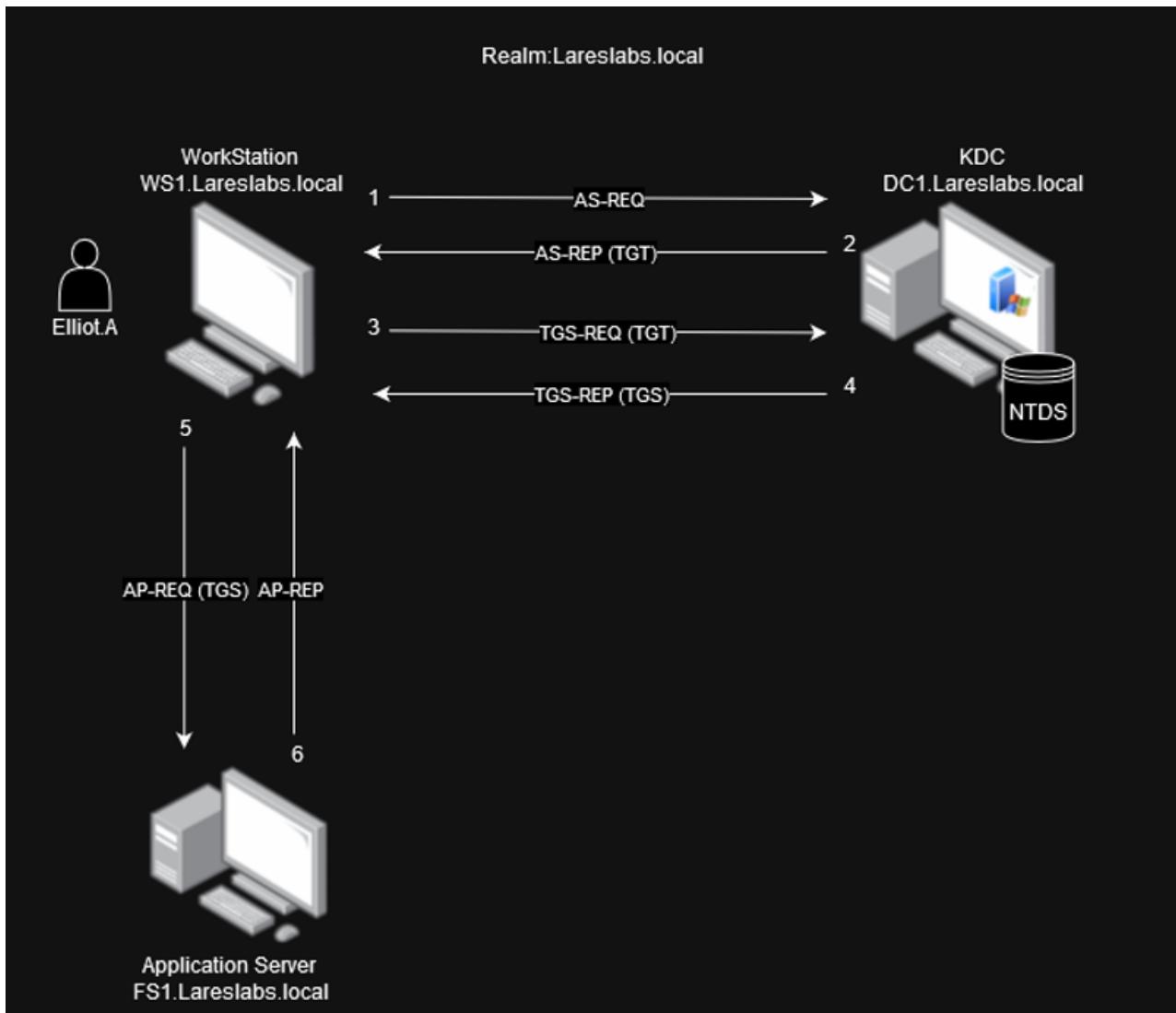
- **Blue highlighted parts**, Wireshark has been able to decrypted information using the Kerberos keys of the different principals.
- **Yellow highlighted parts**, indications of missing key data, resulting the blob not being decrypted.

2013 24.543382 192.168.25.133 192.168.25.155 KRBS 157 AS-REP
2022-24 545086 192.168.25.155 192.168.25.133 KRBS 232 TGS-REQ
> Frame 2013: 157 bytes on wire (1256 bits), 157 bytes captured (1256 bits) on interface '\Device\NPF_{61432CAB-77FA-4DFF-A24A-A9B8A52FF58D}', id 0
> Ethernet II, Src: VMware_89:3d:fe (00:0c:29:89:3d:fe), Dst: VMware_02:d1:82 (00:0c:29:02:d1:82)
> Internet Protocol Version 4, Src: 192.168.25.133, Dst: 192.168.25.155
> Transmission Control Protocol, Src Port: 88, Dst Port: 49170, Seq: 1461, Ack: 241, Len: 103
[2 Reassembled TCP Segments (1563 bytes): #2012(1460), #2013(103)]
✓ Kerberos
> Record Mark: 1559 bytes
✓ as-rep
pvno: 5
msg-type: krb-as-rep (11)
> padata: 1 item
crealm: LARESLABS.LOCAL
✓ cname
name-type: kRB5-NT-PRINCIPAL (1)
✓ cname-string: 1 item
CNameString: Darlene.A
> ticket
✓ enc-part
etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
kvno: 2
✓ cipher [truncated]: 4a6aa39fe6485c9f841ded408cf3ecce3098b013f446e4ca593cdccf951e5a5562e7331b1dc533175da6131123ca17b444eb516d7ff7dae77f0df608f9b6e36f5876
Missing keytype 18 usage 3 (id=missing.1)
> Provides learnt encTicketPart_key in frame 2013 keytype 18 (id=013.1 same=0) (af44795e...)
> Missing keytype 18 usage 3 missing in frame 2013 keytype 18 (id=missing.1 same=0) (00000000...)
Used keytab principal krbtgt@LARESLABS.LOCAL keytype 18 id=keytab.2 same=0) (858d1f94...)

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:

The screenshot shows two windows. The top window is NetworkMiner displaying a list of network traffic. A red box highlights a row where the client IP is 192.168.25.194 and the server IP is 192.168.25.193. The message type is KRB5 and the code is 270, with the error message 'KRB5KDC\_ERR\_PREAUTH\_REQUIRED'. The bottom window is a Windows PowerShell session running on a host with IP 192.168.25.194. It lists files in the '\\FS1\tools' directory.

					270 KRB Error: KRB5KDC_ERR_PREAUTH_REQUIRED
4004	301.322064	192.168.25.194	192.168.25.193	KRB5	270 KRB Error: KRB5KDC_ERR_PREAUTH_REQUIRED
4065	301.329500	192.168.25.193	192.168.25.194	KRB5	373 AS-REQ
4072	301.329902	192.168.25.194	192.168.25.193	KRB5	146 AS-REP
4074	301.330322	192.168.25.193	192.168.25.194	KRB5	1648 TGS-REQ
4082	301.330796	192.168.25.194	192.168.25.193	KRB5	142 TGS-REP
4085	301.331454	192.168.25.193	192.168.25.194	KRB5	
4090	301.331708	192.168.25.194	192.168.25.193	SMB2	1855 Session Setup Request
4092	301.332486	192.168.25.193	192.168.25.194	SMB2	314 Session Setup Response

```

PS C:\Users\elliott.a> dir \\FS1\tools

Directory: \\FS1\tools

Mode                LastWriteTime         Length Name
----                -              ----- 
-a---       2/21/2024  1:57 AM            0 license.txt
-a---       2/21/2024  1:57 AM            0 passwords.txt
-a---       2/21/2024  1:57 AM            0 software.txt

```

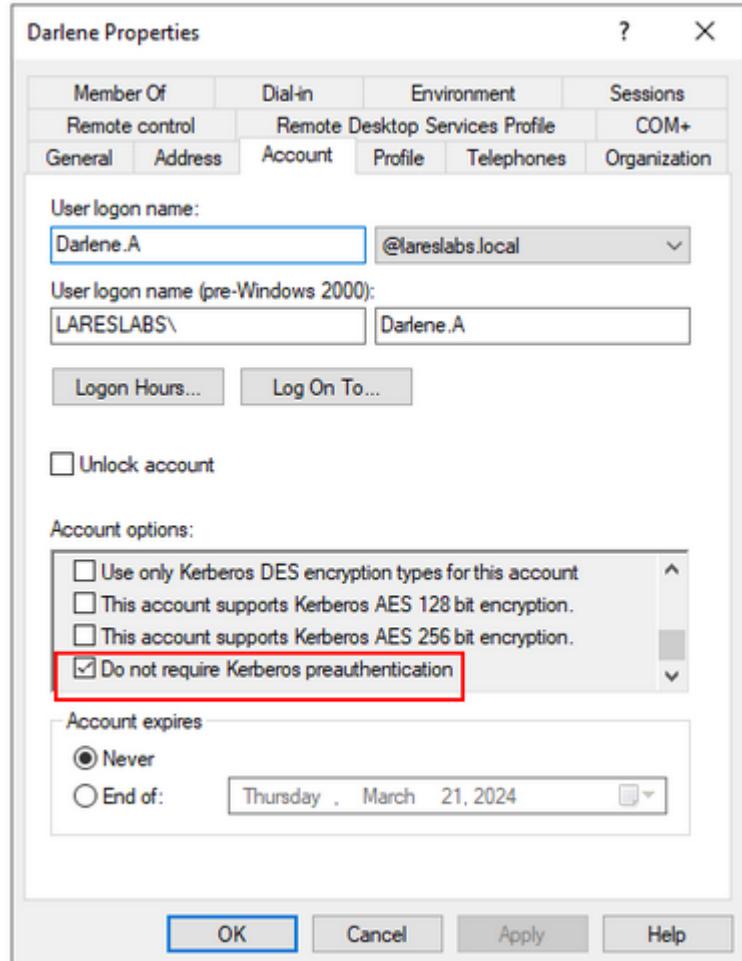
Kerberos auth flow

## 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":

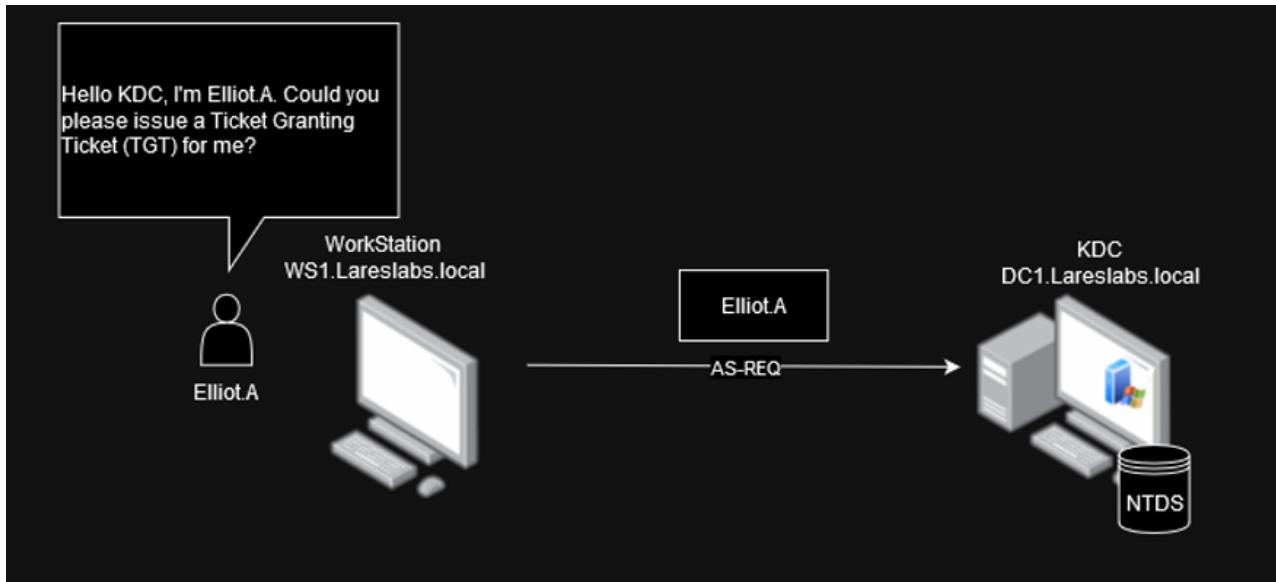


#### Do not require pre-authentication option

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.

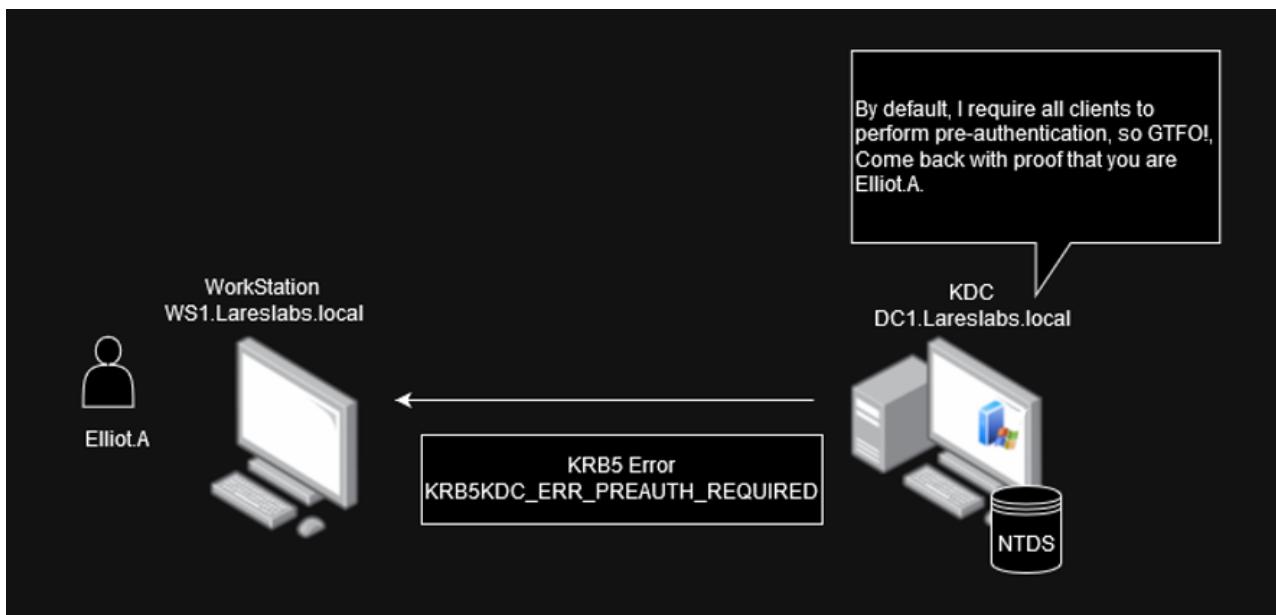
If you find an account with this option enabled, you can obtain an AS-REP message without knowing the client's secret key for that account (ASREProasting). We will discuss this and other 'Roasting' techniques in the second part of the series: Kerberos II—*Credential Access*.

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\_PREAMUTH\_REQUIRED):



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

359	27.115802	192.168.25.156	192.168.25.133	KRB5	293	AS-REQ
360	27.116283	192.168.25.133	192.168.25.156	KRB5	270	KRB Error: KRB5KDC_ERR_PREAMUTH_REQUIRED
367	27.116711	192.168.25.156	192.168.25.133	KRB5	373	AS-REQ
369	27.117312	192.168.25.133	192.168.25.156	KRB5	146	AS-REP
377	27.117899	192.168.25.156	192.168.25.133	KRB5	1648	TGS-REQ
380	27.118568	192.168.25.133	192.168.25.156	KRB5	142	TGS-REP
385	27.118949	192.168.25.156	192.168.25.155	SMB2	1855	Session Setup Request
387	27.119898	192.168.25.155	192.168.25.156	SMB2	314	Session Setup Response

```

< 
> Frame 359: 293 bytes on wire (2344 bits), 293 bytes captured (2344 bits) on interface \Device\NPF_{61432CAB-77FA-4DFF-A24A-A9B8\VMnet8
> Ethernet II, Src: VMware_ab:fc:4a (00:0c:29:ab:fc:4a), Dst: VMware_89:3d:fe (00:0c:29:89:3d:fe)
> Internet Protocol Version 4, Src: 192.168.25.156, Dst: 192.168.25.133
> Transmission Control Protocol, Src Port: 52564, Dst Port: 88, Seq: 1, Ack: 1, Len: 239
`- Kerberos
  > Record Mark: 235 bytes
    < Kerberos>
      > as-req
        < Kerberos>
          > pvno: 5
          > msg-type: krb-as-req (10)
          < Kerberos>
            > padata: 1 item
              > PA-DATA pA-PAC-REQUEST
            < Kerberos>
              > req-body
                < Kerberos>
                  > Padding: 0
                  > kdc-options: 40810010
                < Kerberos>
                  > cname
                    < Kerberos>
                      > name-type: KRB5-NT-PRINCIPAL (1)
                      < Kerberos>
                        > cname-string: 1 item
                          > CNameString: Elliot.A
                      < Kerberos>
                      > realm: LARESLABS.LOCAL
                    < Kerberos>
                  > sname
                < Kerberos>
              < Kerberos>
            < Kerberos>
          < Kerberos>
        < Kerberos>
      < Kerberos>
    < Kerberos>
  < Kerberos>
< Kerberos>

```

### 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***.

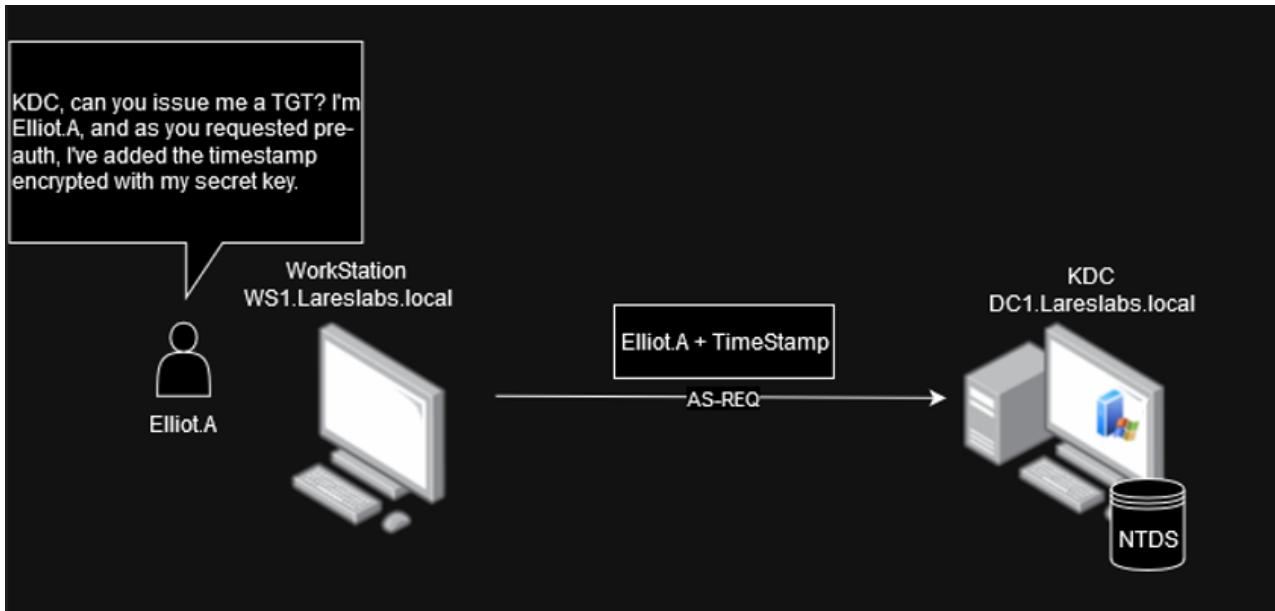
Once the KDC confirms that the client requires pre-authentication, the client must send a second AS-REQ message with more data to authenticate against the KDC.

### 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:

```

278 18.785040 192.168.25.156 192.168.25.133 KRBS5 373 AS-REQ
Frame 278: 373 bytes on wire (2984 bits), 373 bytes captured (2984 bits) on interface \Device\NPF_{61432CAB-77FA-4DFF-A24A-A9B
Ethernet II, Src: VMWare_ab:fc:4a (00:0c:29:ab:fc:4a), Dst: VMWare_89:3d:fe (00:0c:29:89:3d:fe)
Internet Protocol Version 4, Src: 192.168.25.156, Dst: 192.168.25.133
Transmission Control Protocol, Src Port: 53534, Dst Port: 88, Seq: 1, Ack: 1, Len: 319
Kerberos
  > Record Mark: 315 bytes
    > as-req
      > pwno: 5
      > msg-type: krb-as-req (10)
      > padata: 2 items
        > PA-DATA pa-ENC-TIMESTAMP
          > padata-type: pa-ENC-TIMESTAMP (2)
            > padata-value: 3041a003020112a23a043814b022f5d18a0bfced05ed3c4cb1ab2f3a0224c38df9de9777a249efb48cd70912cd8e6598c2
              etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
              cipher: -14b022f5d18a0bfced05ed3c4cb1ab2f3a0224c38df9de9777a249efb48cd70912cd8e6598c27a6683c9240cad764984ed457
        > PA-DATA pa-PAC-REQUEST
          > padata-type: pa-PAC-REQUEST (128)
            > padata-value: 3005a0030101ff
              include-pac: True
    > req-body
      > Padding: 0
      > kdc-options: 40810010
        > cname
          > name-type: kRB5-NT-PRINCIPAL (1)
            > cname-string: 1 item
              CNameString: Elliot.A
            realm: LARESLABS.LOCAL
        > sname
          > name-type: kRB5-NT-SRV-INST (2)
            > sname-string: 2 items
              SNameString: krbtgt
              SNameString: LARESLABS.LOCAL

```

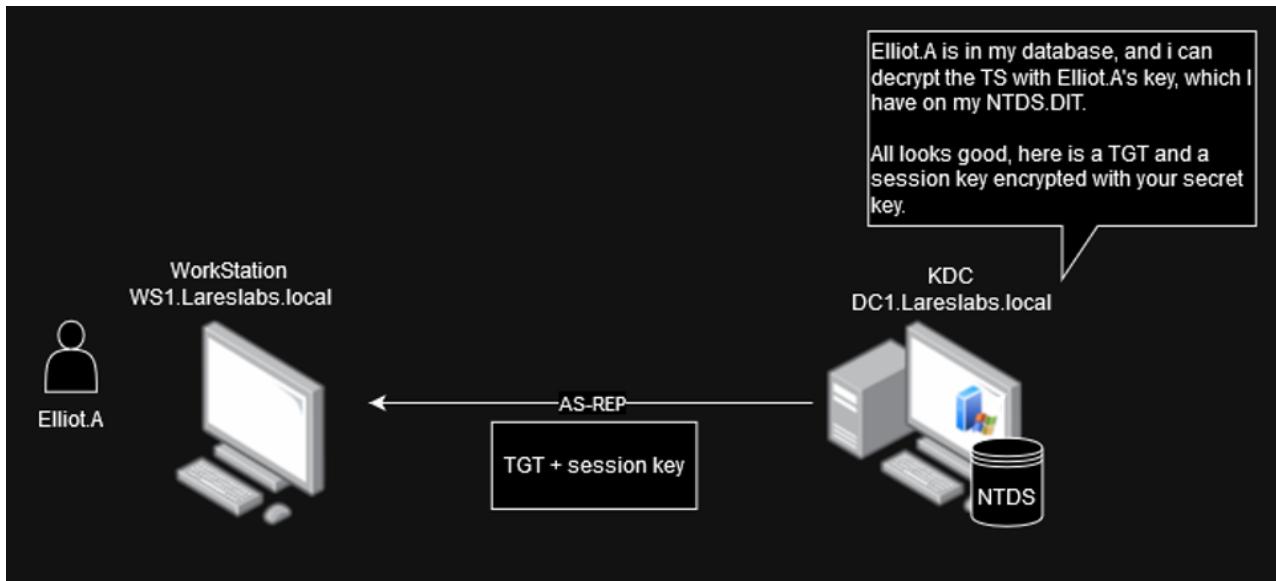
A red box highlights the "PA-DATA pa-ENC-TIMESTAMP" field, which is labeled "Encrypted with the user's kerberos key". Another red box highlights the "cname" field, which is labeled "Client Principal Name (cname)".

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\_PREAMUTH\_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.

369 27.117312 192.168.25.133 192.168.25.156 KRBS 146 AS-REP

```

> cname
< ticket
  tkt-vno: 5
  realm: LARESLABS.LOCAL
> sname
< enc-part
  etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
  kvno: 2
< cipher [truncated]: 744d992fdb1c9d8fd6492b36d70e4725676a48102655160b7487ec1a0ef61869e237f922ab916f4b9a4c5174b05d4f
  > Decrypted keytype 18 usage 2 using keytab principal krbtgt@LARESLABS.LOCAL (id=keytab.2 same=0) (858d1f94...)
< encTicketPart
  Padding: 0
  > flags: 40e10000
  < key
    < Learnt encTicketPart_key keytype 18 (id=369.1) (bdf654c7...)
      > [Expert Info (Chat/Security): Learnt encTicketPart_key keytype 18 (id=369.1) (bdf654c7...)]
      keytype: 18
      keyvalue: bdf654c72ebe562224ed852295cc75b130d5a210e535d40ada5c8e37a45c4a51
      crealm: LARESLABS.LOCAL
    < cname
      name-type: kRB5-NT-PRINCIPAL (1)
      < cname-string: 1 item
        CNameString: Elliot.A
    > transited
      authtime: Feb 28, 2024 01:41:00.000000000 Pacific Standard Time
      starttime: Feb 28, 2024 01:41:00.000000000 Pacific Standard Time
      endtime: Feb 28, 2024 11:41:00.000000000 Pacific Standard Time
      renew-till: Mar 6, 2024 01:41:00.000000000 Pacific Standard Time
      > authorization-data: 1 item
    < enc-part
      etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
      kvno: 2
    < cipher [truncated]: 75675614b767eda2589ccb2e
      < Decrypted keytype 18 usage 3 using keytab principal krbtgt@LARESLABS.LOCAL (id=keytab.4 same=0) (663710fa...)
        > [Expert Info (Chat/Security): Decrypted keytype 18 usage 3 using keytab principal krbtgt@LARESLABS.LOCAL (id=keytab.4 same=0) (663710fa...)]
        > [Expert Info (Chat/Security): Used keymap-all_keys num_keys=237 num_tries=206]
    > encASRepPart
  
```

**Part of the ticket encrypted with the krbtgt secret key**

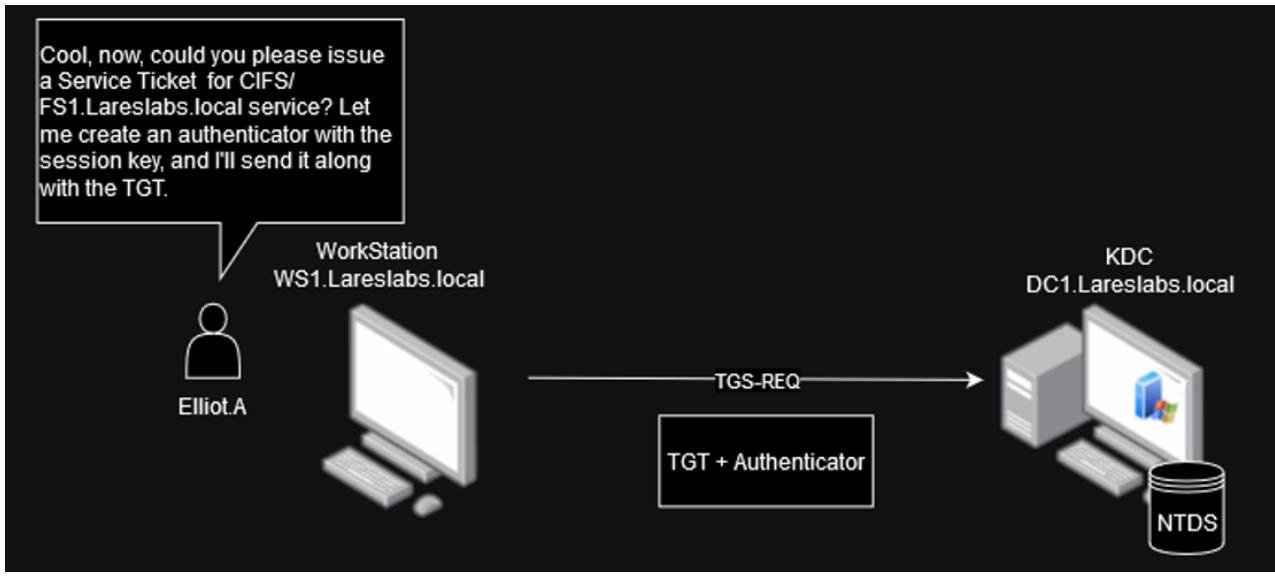
**TGS session key encrypted with the user's secret key**

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):

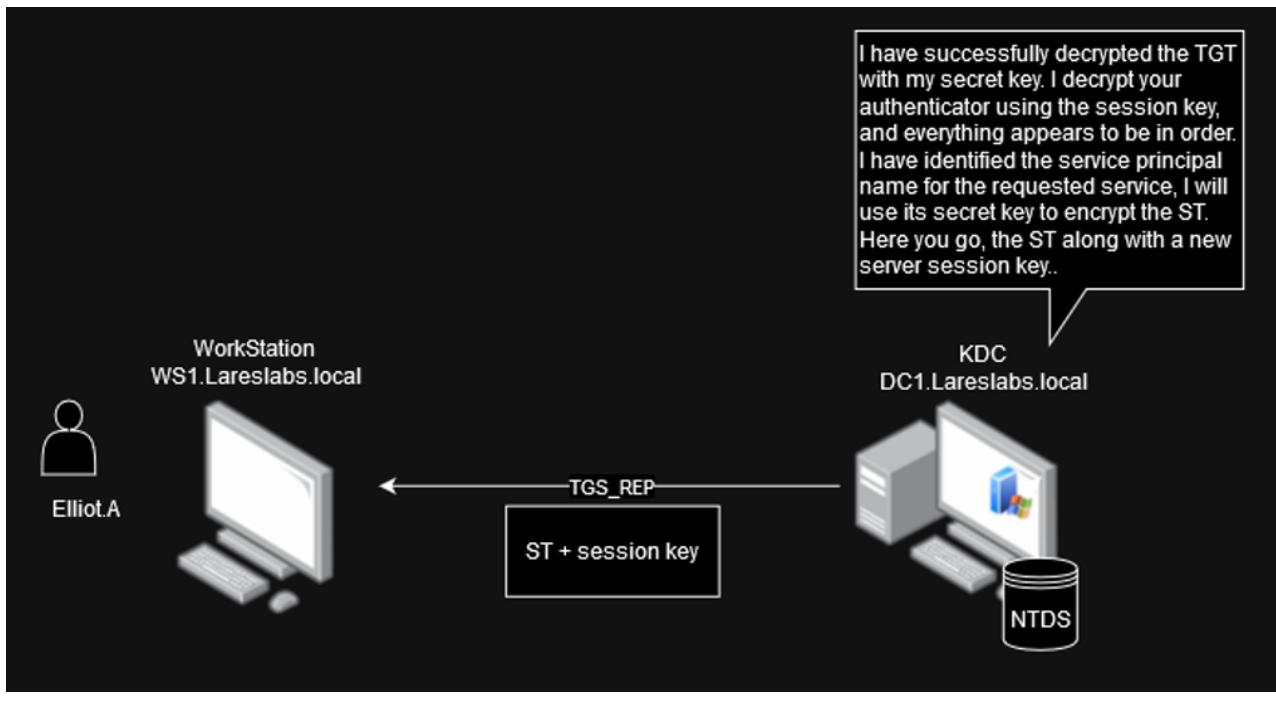
The Wireshark capture shows a TGS-REQ message (frame 483). The PA-TGS-REQ block is highlighted with a red box. Inside, the ticket and authenticator fields are expanded. The ticket field shows details like tkt-vno: 5, realm: LARESLABS.LOCAL, and sname. The authenticator field shows details like etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18), cipher [truncated], and cname. The re-body section shows the service name (cifs) and its type (KRB5-NT-SRV-INST (2)). Annotations in the screenshot explain that the ticket is encrypted with the krbtgt key and the authenticator is encrypted with the user's session key. The service name is identified as "Service".

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:

59747	419.780249	192.168.25.133	192.168.25.155	KRBS	1460 TGS-REP
-------	------------	----------------	----------------	------	--------------

```

> Record Mark: 1402 bytes
v tgs-rep
  pvno: 5
  msg-type: krb-tgs-rep (13)
  crealm: LARESLABS.LOCAL
  > cname
  v ticket
    tkt-vno: 5
    realm: LARESLABS.LOCAL
    > sname
    v enc-part
      etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
      kvno: 2
      v cipher [truncated]: 83202424aae03a18acffd6000c65c5f336484b545399c624b1c5ac0b84cd73d95c5806985345805f74dc8dae9a14c0
        > Decrypted keytype 18 usage 2 using keytab principal krbtgt@LARESLABS.LOCAL (id=keytab.2 same=0) (858d1f94...)
        v encTicketPart
          Padding: 0
          > flags: 60a10000
          v key
            > Learnt encTicketPart_key keytype 18 (id=59747.1) (474295f6...)
            keytype: 18
            keyvalue: 474295f62d4780180fce0d749b07b3860576a2e6f5e307559260a377413aff73
            crealm: LARESLABS.LOCAL
            > cname
            > transited
            authtime: Mar 7, 2024 14:16:29.000000000 Pacific Standard Time
            starttime: Mar 7, 2024 14:16:29.000000000 Pacific Standard Time
            endtime: Mar 8, 2024 00:16:29.000000000 Pacific Standard Time
            renew-till: Mar 14, 2024 15:16:29.000000000 Pacific Daylight Time
            > authorization-data: 1 item
      v enc-part
        etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
        v cipher [truncated]: 283588e535bce636fc94d43ecfc1ad7
          > Decrypted keytype 18 usage 8 using learnt encTicketPart_key
          > encTGSRepPart
    > Provides learnt encTicketPart_key in frame 59747 keytype 18 (id=59747.1 same=0) (474295f6...)
    > Provides learnt encTGSRepPart_key in frame 59747 keytype 18 (id=59747.2 same=0) (474295f6...)
    > Used keytab principal krbtgt@LARESLABS.LOCAL keytype 18 (id=keytab.2 same=0) (858d1f94...)
    > Used learnt encTicketPart_key in frame 59726 keytype 18 (id=59726.1 same=3) (d253b819...)

```

Encrypted with the service owner (MON2\$) kerberos secret key

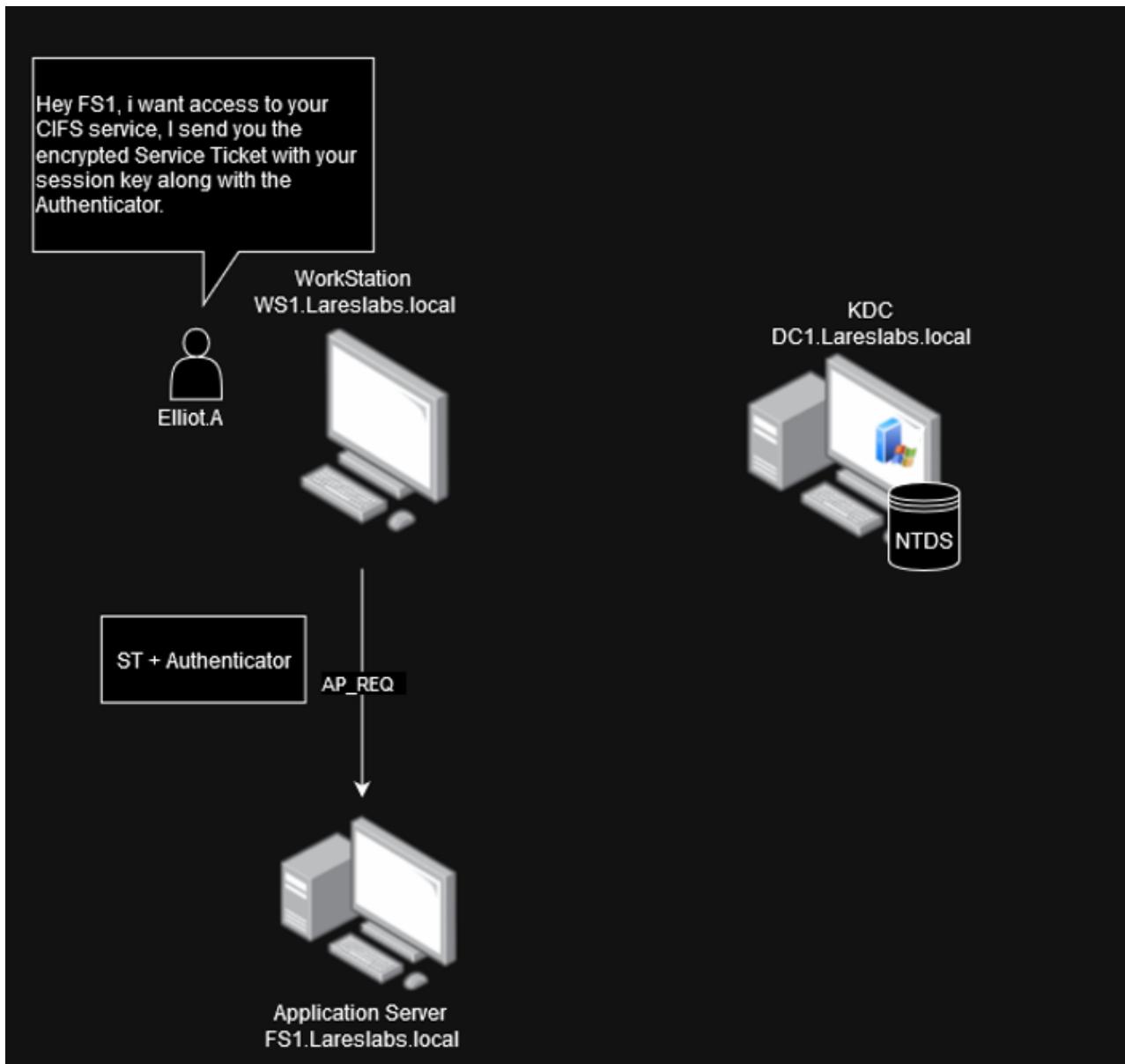
Encrypted with the user logon session key

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:

316 18.831367 192.168.25.156 192.168.25.155 SMB2 396 Session Setup Request  
 317 18.836890 192.168.25.156 192.168.25.156 SMB2 315 Session Setup Response

[2 Reassembled TCP Segments (1802 bytes): #307(1460), #316(342)]  
 NetBIOS Session Service  
 SMB2 (Server Message Block Protocol version 2)  
 > SMB2 Header  
 > Session Setup Request (0x01)  
 [Preauth Hash: 6d07a3da9ce4500417fdaa742fb0b6100bbb1723049f6768de51e062509d4948b581b1ac09b2f7b0926863a408b397a9c2f13746f]  
 > StructureSize: 0x0019  
 > Flags: 0  
 > Security mode: 0x01, Signing enabled  
 > Capabilities: 0x00000001, DFS  
 Channel: None (0x00000000)  
 Previous Session Id: 0x0000000000000000  
 Blob Offset: 0x00000058  
 Blob Length: 1710  
 > Security Blob [truncated]: 608206aa06062b0601050502a082069aa030302e06092a864882f71201020206092a864886f7120102020  
 > GSS-API Generic Security Service Application Program Interface  
 OID: 1.3.6.1.5.5.2 (SPNEGO - Simple Protected Negotiation)  
 > Simple Protected Negotiation  
 > negTokenInit  
 > mechTypes: 4 items  
 mechToken [truncated]: 6082065c06092a864886f7120102020100e82064b30820647a003020105a10302010ea207030500200000  
 > krb5\_blob [truncated]: 6082065c06092a864886f7120102020100e82064b30820647a003020105a10302010ea207030500200000  
 KRBS OID: 1.2.840.113554.1.2.2 (KRBS - Kerberos)  
 krb5\_tok\_id: KRBS\_AP\_REQ (0x0001)  
 > Kerberos  
 > ap-req  
 > pvno: 5  
 > msg-type: krb-ap-req (14)  
 > Padding: 0  
 > ap-options: 20000000  
 > ticket  
 > authenticator

Service Ticket, encrypted with the service owner kerberos key

Encrypted with the current service session key

### 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:

316 18.831367	192.168.25.156	192.168.25.155	SMB2	396 Session Setup Request
317 18.836890	192.168.25.155	192.168.25.156	SMB2	315 Session Setup Response
2011 24.542823	192.168.25.155	192.168.25.133	KRBS	294 AS-REQ
2013.24.543383	192.168.25.133	192.168.25.155	KRBS	157 AS-REP
> Frame 317: 315 bytes on wire (2520 bits), 315 bytes captured (2520 bits) on interface \Device\NPF_{61432CAB-77FA-4DFF-A24A-A9B8A52FF58D}, id 0				
> Ethernet II, Src: VMware_02:d1:82 (00:0c:29:02:d1:82), Dst: VMware_ab:fc:4a (00:0c:29:ab:fc:4a)				
> Internet Protocol Version 4, Src: 192.168.25.155, Dst: 192.168.25.156				
> Transmission Control Protocol, Src Port: 445, Dst Port: 53529, Seq: 589, Ack: 2096, Len: 261				
> NetBIOS Session Service				
✓ SMB2 (Server Message Block Protocol version 2)				
> SMB2 Header				
✓ Session Setup Response (0x01)				
[Preauth Hash: 6d07a3da9ce4500417fdaa742fb0b6100bbb1723049f6768de51e062509d4948b581b1ac09b2f7b0926863a408b397a9c2f13746f71b0c438a6b898983cd0685]				
> StructureSize: 0x0009				
> Session Flags: 0x0000				
> Blob Offset: 0x00000048				
> Blob Length: 185				
✓ Security Blob [truncated]: a181b63081b3a0030a0100a10b06092a864886f712010202a2819e04819b60819806092a864886f71201020202006f8188308185a003020105a10302010fa				
< GSS-API Generic Security Service Application Program Interface				
< Simple Protected Negotiation				
< negTokenTarg				
negResult: accept-completed (0)				
supportedMech: 1.2.840.48018.1.2.2 (MS_KRBS - Microsoft Kerberos)				
responseToken [truncated]: 60819806092a864886f71201020202006f8188308				
< krb5_blob [truncated]: 60819806092a864886f71201020202006f8188308				
KRB5 OID: 1.2.840.113554.1.2.2 (KRB5 - Kerberos 5)				
krb5_tok_id: KRBS_AP REP (0x0002)				
< Kerberos				
< ap-rep				
pvno: 5				
msg-type: krb-ap-rep (15)				
< enc-part				
etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)				
< cipher: 7d9d9dbf38e085e2be702e9911cc1101b5e09767b019f8b7d3dfe283e19bf232c391944b9ac80bc832e93bea035f36f7defcb1696a3483256c36f8f0e				
< Decrypted keytype 18 usage 12 using learnt encTicketPart_key in frame 298 (id=298.1 same=2) (c8ddadef4...)				
< [Expert Info (Chat/Security): Decrypted keytype 18 usage 12 using learnt encTicketPart_key in frame 298 (id=298.1 same=2) (c8ddadef4...)]				
< [Decrypted keytype 18 usage 12 using learnt encTicketPart_key in frame 298 (id=298.1 same=2) (c8ddadef4...)]				

(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.

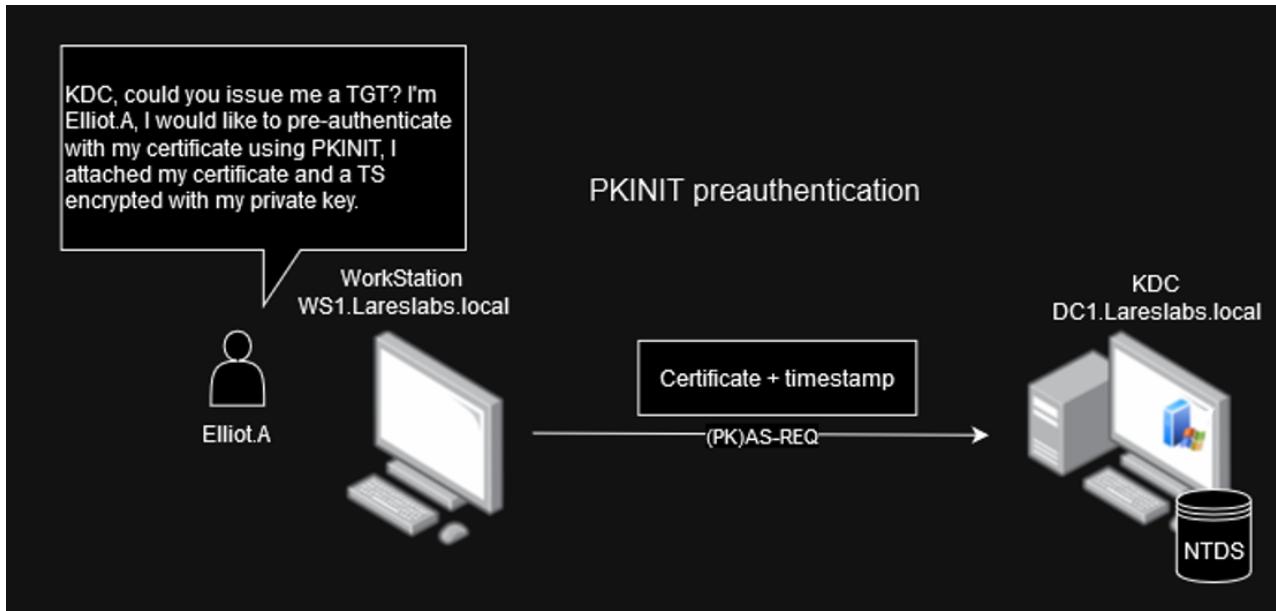
## Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)

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:

```

150 152.100101 192.168.25.134 192.168.25.133 KRBS 1249 AS-REQ
<
> Internet Protocol Version 4, Src: 192.168.25.134, Dst: 192.168.25.133
> Transmission Control Protocol, Src Port: 49440, Dst Port: 88, Seq: 1461, Ack: 1, Len: 1195
> [2 Reassembled TCP Segments (2655 bytes): #149(1460), #150(1195)]
` Kerberos
  > Record Mark: 2651 bytes
  ` as-req
    > pno: 5
    > msg-type: krb-as-req (10)
    ` padata: 2 items
      > PA-DATA PA-PAC-REQUEST
        > padata-type: pa-PAC-REQUEST (128)
        > padata-value: 3005a0030101ff
          include-pac: True
      ` PA-DATA pa-PK-AS-REQ
        > padata-type: pa-PK-AS-REQ (16)
        > padata-value [truncated]: 308209828082097e3082097a06092a864886f70d010702a082096b30820967020103310b300906052b0e03021a05003082019f06072b060105020
          > signedAuthPack (id-signedData)
            > contentype: 1.2.840.113549.1.7.2 (id-signedData)
            > SignedData
              > version: v3 (3)
              > digestAlgorithms: 1 item
              > encapsContentInfo (id-pkauthdata)
              > certificates: 1 item
                > CertificateChoices: certificate (0)
                  > certificate (id-at-commonName=Elliot.A,id-at-commonName=Users,dc=lareslabs,dc=local)
                > signerInfos: 1 item
              > AuthPack
                > pkAuthenticator
                  > cusec: 655217
                  > ctime: Mar 10, 2024 09:13:19.000000000 Pacific Daylight Time
                  > nonce: 634866384
                  > paChecksum: 22b71df19af6521c9bff9e12b5ce624f552c258f
                  > clientPublicValue
                  > clientDHNonce: 75b68885f467335537c2f801d4ce361d83a4e0fc8f6608h1261ahaa724e15bh9
` Signed Certificate
` PK Authenticator (Timestamp)

```

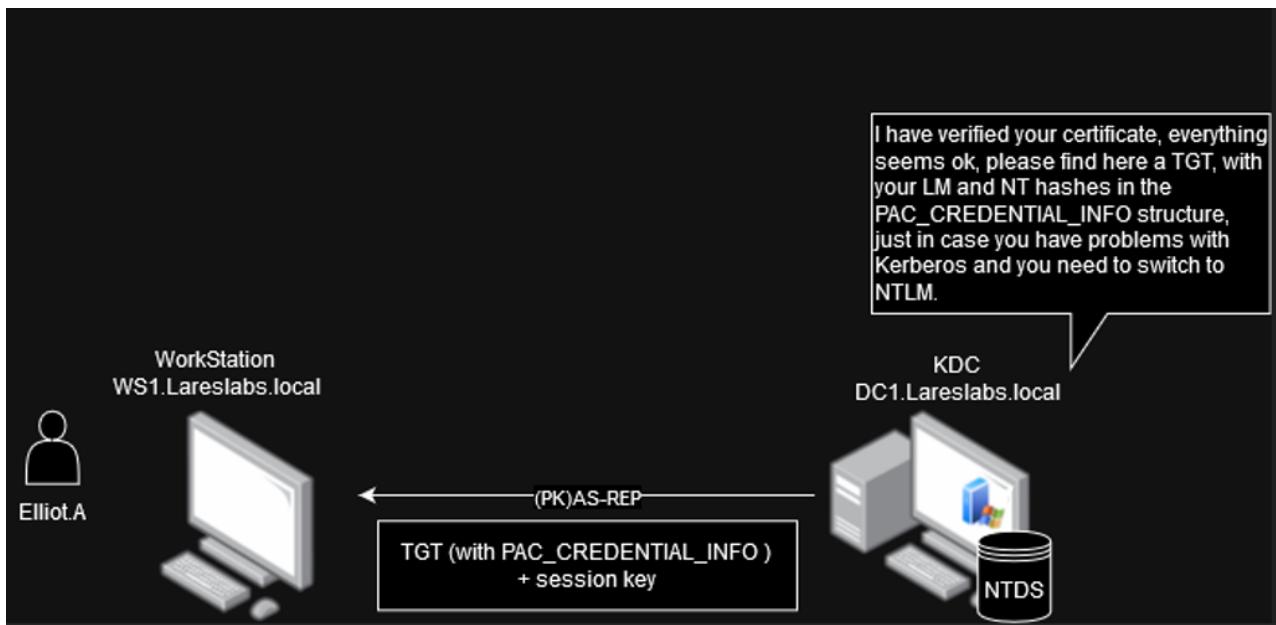
PK-AS-REQ

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-RFP

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](#)