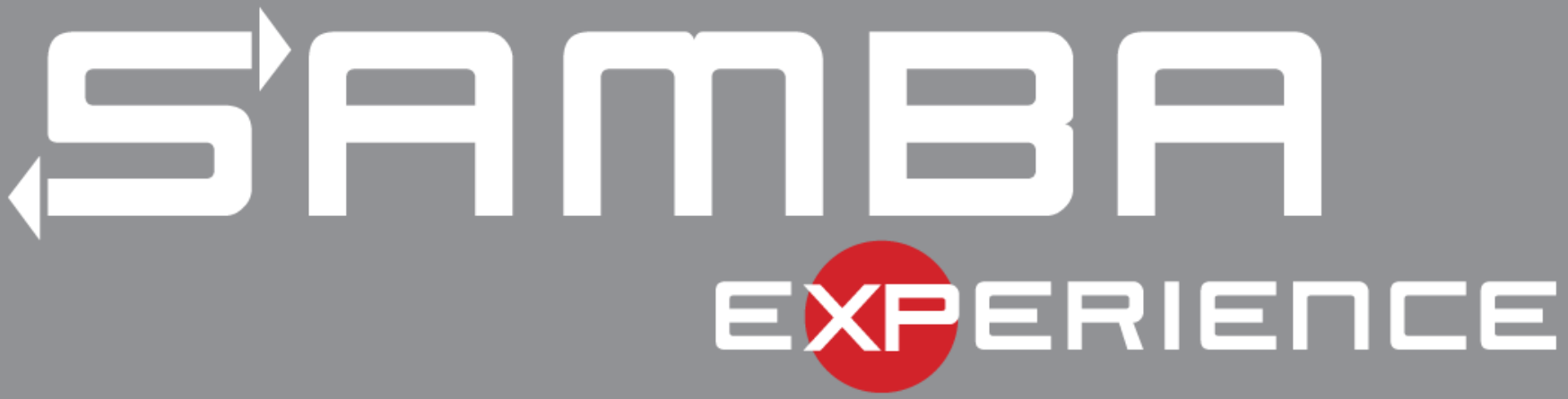


Modern Kerberos Features

within Samba

Stefan Metzmacher <metze@samba.org>

Samba Team / SerNet

2020-05-27

https://samba.org/~metze/presentations/2020/SambaXP/

Topics

- The basics of Kerberos (krb5)
- What is S4U2Self
- What is FAST/CompoundIdentity
- What does existing Kerberos libraries support
- Using S4U2Self/FAST in winbindd
- Challenges of adding new Features
- Protocol Testing with Python
- Questions?

The basics of Kerberos (krb5) (Part1)

- Kerberos is an authentication protocol
  - Defined in RFC 4120 and others
  - Its design consists of 3 components (Clients, KDCs, Servers)
  - A Realm is typically based on DNS-Names, e.g. EXAMPLE.COM
  - Strong mutual authentication is offered, which provides replay protection
  - GSSAPI/SPENEGO is used for client to server authentication
- Kerberos uses strong symmetric key crypto:
  - aes256-cts-hmac-sha1-96 (by default)
  - aes128-cts-hmac-sha1-96 is also possible, but never really used
  - arcfour-hmac-md5 is still available and uses the unsalted NTHASH
  - des based crypto is deprecated/disabled in modern networks
- public-key crypto is also available (PKINIT):
  - Typically authentication with smartcards
  - Or other certificate based methods

The basics of Kerberos (krb5) (Part1)

- Kerberos is an authentication protocol
  - Defined in RFC 4120 and others
  - Its design consists of 3 components (Clients, KDCs, Servers)
  - A Realm is typically based on DNS-Names, e.g. EXAMPLE.COM
  - Strong mutual authentication is offered, which provides replay protection
  - GSSAPI/SPENEGO is used for client to server authentication
- Kerberos uses strong symmetric key crypto:
  - aes256-cts-hmac-sha1-96 (by default)
  - aes128-cts-hmac-sha1-96 is also possible, but never really used
  - arcfour-hmac-md5 is still available and uses the unsalted NTHASH
  - des based crypto is deprecated/disabled in modern networks
- public-key crypto is also available (PKINIT):
  - Typically authentication with smartcards
  - Or other certificate based methods

The basics of Kerberos (krb5) (Part1)

- Kerberos is an authentication protocol
  - Defined in RFC 4120 and others
  - Its design consists of 3 components (Clients, KDCs, Servers)
  - A Realm is typically based on DNS-Names, e.g. EXAMPLE.COM
  - Strong mutual authentication is offered, which provides replay protection
  - GSSAPI/SPENEGO is used for client to server authentication
- Kerberos uses strong symmetric key crypto:
  - aes256-cts-hmac-sha1-96 (by default)
  - aes128-cts-hmac-sha1-96 is also possible, but never really used
  - arcfour-hmac-md5 is still available and uses the unsalted NTHASH
  - des based crypto is deprecated/disabled in modern networks
- public-key crypto is also available (PKINIT):
  - Typically authentication with smartcards
  - Or other certificate based methods

The basics of Kerberos (krb5) (Part2)

- The central "Key Distribution Center" (KDC)
  - Stores encryption keys (typically based on passwords)
  - Client Principals, e.g. administrator@EXAMPLE.COM
  - Ticket Granting Ticket (TGT) principal, e.g. krbtgt/EXAMPLE.COM@EXAMPLE.COM
  - Server Principals, e.g. cifs/files.example.com@EXAMPLE.COM
  - It provides an "Authenication Service" (AS)
  - It provides a "Ticket Granting Service" (TGS)
  - Both services of the KDC provide (grant) Tickets
- A Ticket consists of a unencrypted part containing:
  - The realm of the granting KDC
  - The service principal within the KDC's realm
- The encrypted part of the Ticket:
  - Is encrypted with the shared secret between KDC and service
  - The encryption type and the key version (kvno) identify the key
  - It contains details about the client/user
  - A random ticket session key with a midterm lifetime, e.g. 10 hours

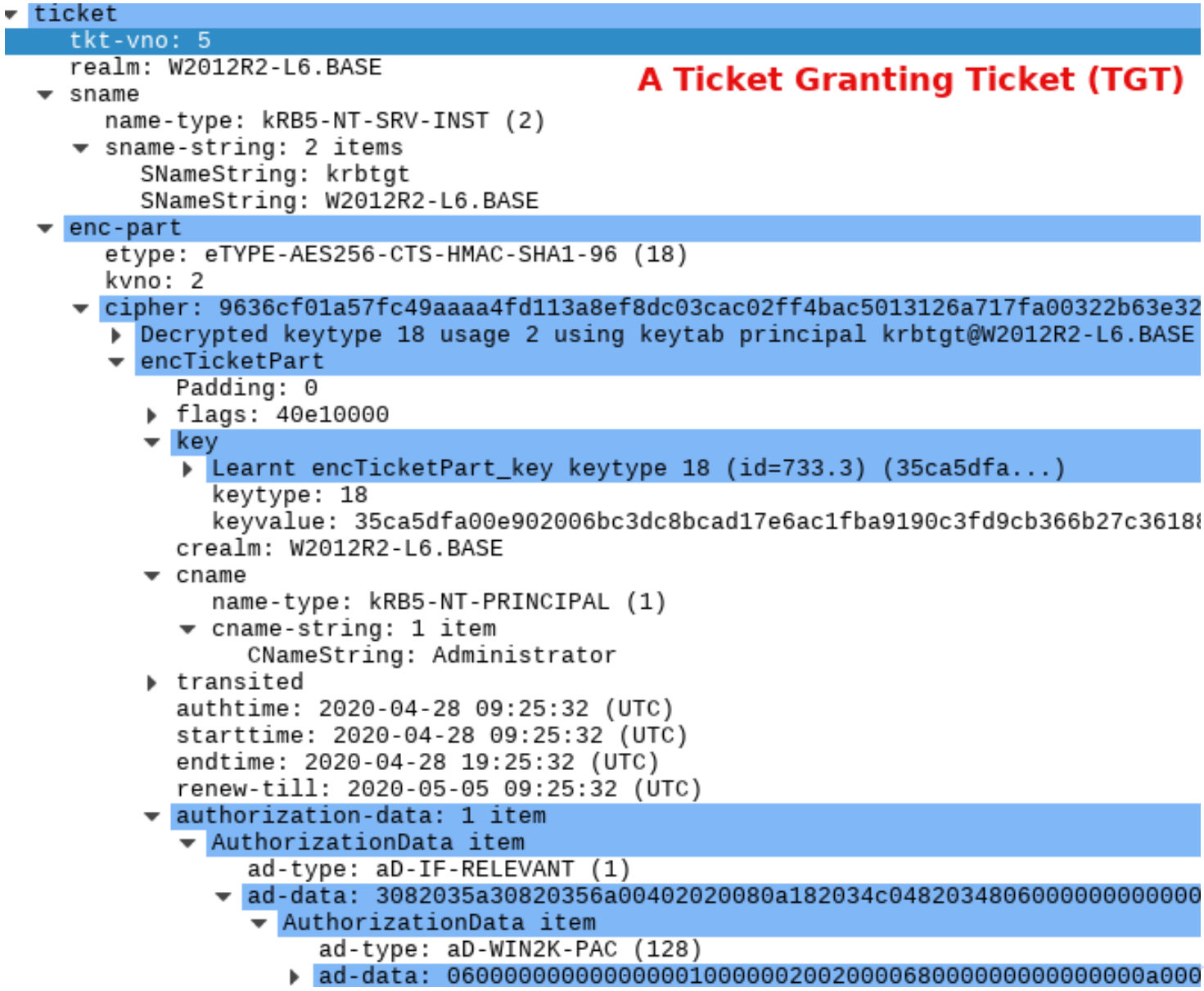
The basics of Kerberos (krb5) (Part2)

- The central "Key Distribution Center" (KDC)
  - Stores encryption keys (typically based on passwords)
  - Client Principals, e.g. administrator@EXAMPLE.COM
  - Ticket Granting Ticket (TGT) principal, e.g. krbtgt/EXAMPLE.COM@EXAMPLE.COM
  - Server Principals, e.g. cifs/files.example.com@EXAMPLE.COM
  - It provides an "Authenication Service" (AS)
  - It provides a "Ticket Granting Service" (TGS)
  - Both services of the KDC provide (grant) Tickets
- A Ticket consists of a unencrypted part containing:
  - The realm of the granting KDC
  - The service principal within the KDC's realm
- The encrypted part of the Ticket:
  - Is encrypted with the shared secret between KDC and service
  - The encryption type and the key version (kvno) identify the key
  - It contains details about the client/user
  - A random ticket session key with a midterm lifetime, e.g. 10 hours

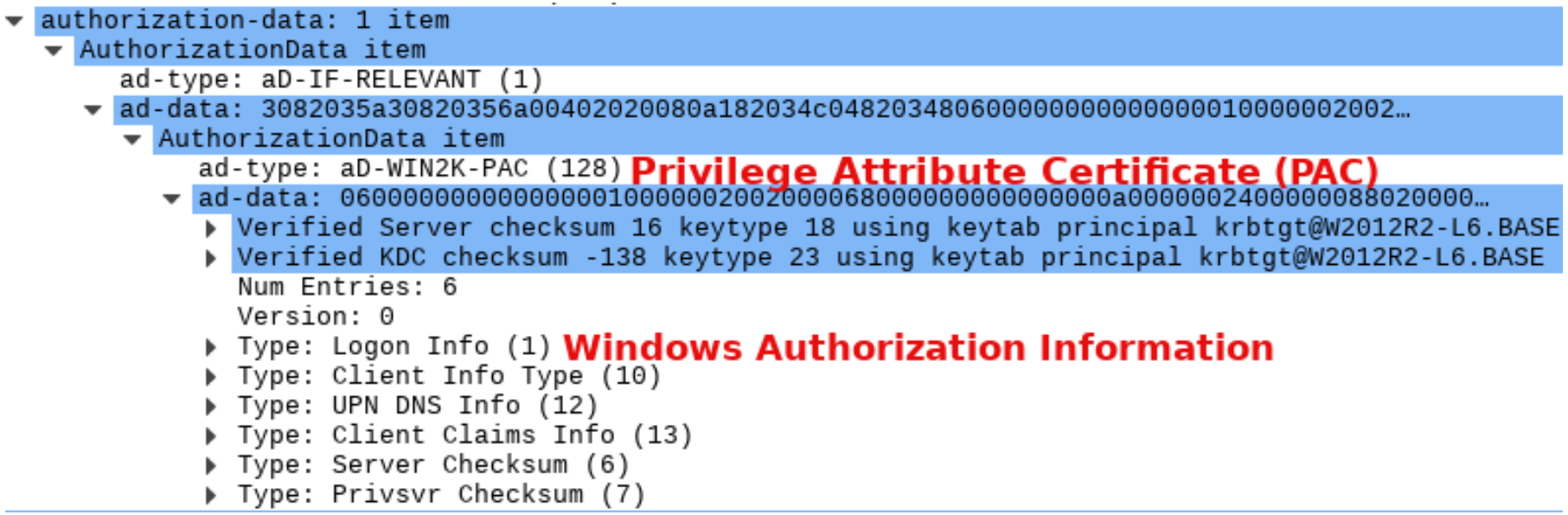
The basics of Kerberos (krb5) (Part2)

- The central "Key Distribution Center" (KDC)
  - Stores encryption keys (typically based on passwords)
  - Client Principals, e.g. administrator@EXAMPLE.COM
  - Ticket Granting Ticket (TGT) principal, e.g. krbtgt/EXAMPLE.COM@EXAMPLE.COM
  - Server Principals, e.g. cifs/files.example.com@EXAMPLE.COM
  - It provides an "Authenication Service" (AS)
  - It provides a "Ticket Granting Service" (TGS)
  - Both services of the KDC provide (grant) Tickets
- A Ticket consists of a unencrypted part containing:
  - The realm of the granting KDC
  - The service principal within the KDC's realm
- The encrypted part of the Ticket:
  - Is encrypted with the shared secret between KDC and service
  - The encryption type and the key version (kvno) identify the key
  - It contains details about the client/user
  - A random ticket session key with a midterm lifetime, e.g. 10 hours

The Details of a Ticket (Part1)

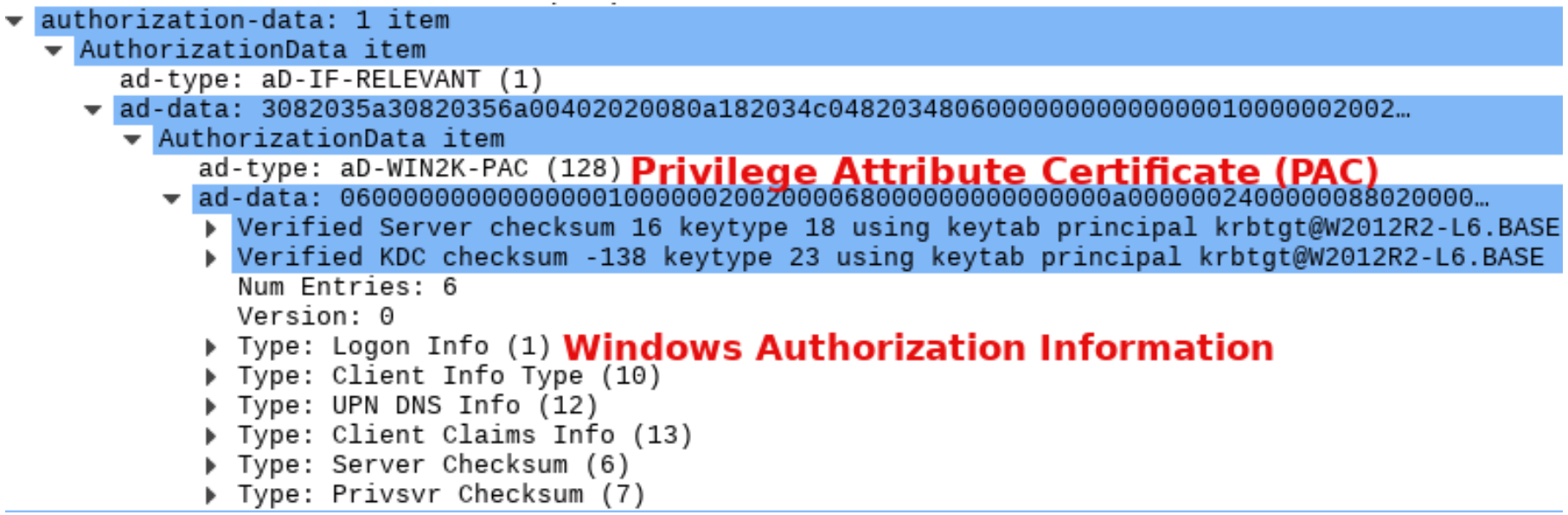


The Details of a Ticket (Part2)



- Server and KDC/Privsvr Checksums:
  - Protect the Authorization Information from changing
- "Logon Info" contains
  - The full Windows Authorization Token with group memberships

The Details of a Ticket (Part2)



- Server and KDC/Privsvr Checksums:
  - Protect the Authorization Information from changing
- "Logon Info" contains
  - The full Windows Authorization Token with group memberships

The Details of a Ticket (Part3)

```
▼ PAC_LOGON_INFO:
    Referent ID: 0x00020000
    Logon Time: Apr 28, 2020 11:21:14.090883000 CEST
    Logoff Time: Infinity (absolute time)
    Kickoff Time: Infinity (absolute time)
    PWD Last Set: Mar 20, 2015 10:57:31.494778400 CET
    PWD Can Change: Mar 21, 2015 10:57:31.494778400 CET
    PWD Must Change: Infinity (absolute time)
  ▶ Acct Name: Administrator
  ▶ Full Name
  ▶ Logon Script
  ▶ Profile Path
  ▶ Home Dir
  ▶ Dir Drive
    Logon Count: 3220
    Bad PW Count: 1
    User RID: 500
    Group RID: 513
    Num RIDs: 5
  ▶ GroupIDs
  ▶ User Flags: 0x00000020
    User Session Key: 00000000000000000000000000000000
  ▶ Server: W2012R2-188
  ▶ Domain: W2012R2-L6
  ▶ SID pointer:
    Dummy1 Long: 0x00000000
    Dummy2 Long: 0x00000000
  ▶ User Account Control: 0x00000210
    Dummy4 Long: 0x00000000
    Dummy5 Long: 0x00000000
    Dummy6 Long: 0x00000000
    Dummy7 Long: 0x00000000
    Dummy8 Long: 0x00000000
    Dummy9 Long: 0x00000000
    Dummy10 Long: 0x00000000
    Num Extra SID: 2
  ▶ SID_AND_ATTRIBUTES_ARRAY:
  ▶ ResourceGroupIDs
```

The Authentication Service (AS) Exchange (Part1)

- The AS-Exchange authenticates a client/user
  - The client proves its identity to the KDC
  - The KDC returns a Ticket Granting Ticket (TGT)
  - Typically two round trips

- First AS-REQ without Pre-Authentication
  - Gives an Error-Message with PRE-AUTH-REQUIRED
  - Returns the Password-Salt
  - May also provide the capabilities of the KDC

- AS-REQ with Password Pre-Authentication
  - A timestamp is encrypted with the client/user key
  - A ticket for the krbtgt service is returned in the AS-REP
  - The content of the encTicketPart is only known to the KDC
  - The content of the encASRepPart is encrypted with the client/user key
  - encTicketPart and encASRepPart contain the same ticket session key
  - The TGT's ticket session key is a shared secret between client and KDC

The Authentication Service (AS) Exchange (Part1)

- The AS-Exchange authenticates a client/user
  - The client proves its identity to the KDC
  - The KDC returns a Ticket Granting Ticket (TGT)
  - Typically two round trips
- First AS-REQ without Pre-Authentication
  - Gives an Error-Message with PRE-AUTH-REQUIRED
  - Returns the Password-Salt
  - May also provide the capabilities of the KDC
- AS-REQ with Password Pre-Authentication
  - A timestamp is encrypted with the client/user key
  - A ticket for the krbtgt service is returned in the AS-REP
  - The content of the encTicketPart is only known to the KDC
  - The content of the encASRepPart is encrypted with the client/user key
  - encTicketPart and encASRepPart contain the same ticket session key
  - The TGT's ticket session key is a shared secret between client and KDC

The Authentication Service (AS) Exchange (Part1)

- The AS-Exchange authenticates a client/user
  - The client proves its identity to the KDC
  - The KDC returns a Ticket Granting Ticket (TGT)
  - Typically two round trips
- First AS-REQ without Pre-Authentication
  - Gives an Error-Message with PRE-AUTH-REQUIRED
  - Returns the Password-Salt
  - May also provide the capabilities of the KDC
- AS-REQ with Password Pre-Authentication
  - A timestamp is encrypted with the client/user key
  - A ticket for the krbtgt service is returned in the AS-REP
  - The content of the encTicketPart is only known to the KDC
  - The content of the encASRepPart is encrypted with the client/user key
  - encTicketPart and encASRepPart contain the same ticket session key
  - The TGT's ticket session key is a shared secret between client and KDC

The Authentication Service (AS) Exchange (Part2)

```
▸ Internet Protocol Version 4, Src: 172.31.99.189, Dst: 172.31.9.188
▸ Transmission Control Protocol, Src Port: 49163, Dst Port: 88, Seq: 3829371254, Ack: 3818202977, Len:
▾ Kerberos
  ▸ Record Mark: 315 bytes
  ▾ as-req
      pvno: 5
      msg-type: krb-as-req (10)
    ▾ padata: 2 items
      ▾ PA-DATA pA-ENC-TIMESTAMP
        ▾ padata-type: pA-ENC-TIMESTAMP (2)
          ▾ padata-value: 303da003020117a236043433f05e451883c424c3a59fad7fe347581a91eaec42b945fb26…
              etype: eTYPE-ARCFOUR-HMAC-MD5 (23)
            ▾ cipher: 33f05e451883c424c3a59fad7fe347581a91eaec42b945fb265e6bb3838def8e178f861b…
              ▸ Decrypted keytype 23 usage 1 using keytab principal Administrator@W2012R2-L6.BASE
                patimestamp: 2020-04-22 14:19:23 (UTC)
                pausec: 351183
      ▾ 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: administrator
        realm: w2012r2-l6.base
      ▾ sname
          name-type: kRB5-NT-SRV-INST (2)
        ▾ sname-string: 2 items
            SNameString: krbtgt
            SNameString: w2012r2-l6.base
        till: 2037-09-13 02:48:05 (UTC)
        rtime: 2037-09-13 02:48:05 (UTC)
        nonce: 71702650
      ▸ etype: 6 items
      ▸ addresses: 1 item W2012R2-189<20>
```

AS-REQ with Password Pre-Authentication

The Authentication Service (AS) Exchange (Part3)

```
▼ as-rep
    pvno: 5
    msg-type: krb-as-rep (11)
    crealm: W2012R2-L6.BASE
  ▼ cname
      name-type: kRB5-NT-PRINCIPAL (1)
    ▼ cname-string: 1 item
        CNameString: Administrator
  ▼ ticket
      tkt-vno: 5
      realm: W2012R2-L6.BASE
    ▼ sname
        name-type: kRB5-NT-SRV-INST (2)
      ▼ sname-string: 2 items
          SNameString: krbtgt
          SNameString: W2012R2-L6.BASE
    ▶ enc-part
  ▼ enc-part
      etype: eTYPE-ARCFOUR-HMAC-MD5 (23)
      kvno: 1
    ▼ cipher: 656c0716f51d2c1de417b8c981b461178d1e90fa470ec81b17cecc9d1c2365635db726ff…
      ▶ Decrypted keytype 23 usage 3 using keytab principal Administrator@W2012R2-L6.BASE
      ▼ encASRepPart
        ▶ key
        ▶ last-req: 1 item
          nonce: 71702650
          key-expiration: 2037-09-14 02:48:05 (UTC)
          Padding: 0
        ▶ flags: 40e10000
          authtime: 2020-04-22 14:19:23 (UTC)
          starttime: 2020-04-22 14:19:23 (UTC)
          endtime: 2020-04-23 00:19:23 (UTC)
          renew-till: 2020-04-29 14:19:23 (UTC)
          srealm: W2012R2-L6.BASE
        ▼ sname
            name-type: kRB5-NT-SRV-INST (2)
          ▼ sname-string: 2 items
              SNameString: krbtgt
              SNameString: W2012R2-L6.BASE
        ▶ caddr: 1 item W2012R2-189<20>
        ▶ encrypted-pa-data: 1 item
```

AS-REP returns a TGT

encASRepPart mirrors:

* the ticket session key
* other details of the ticket

The Client/Server Authentication (AP) Exchange (Part1)

- The AP-Exchange authenticates a client to a service
  - The client proves knowledge about the provides Ticket
  - It can be used directly for GSSAPI client to server authentication
  - But it can also be used to authenticate requests to the KDC
- AP-REQ provides a Ticket and an Authenticator
  - The Authenticator is encrypted with the ticket session key
  - The Authenticator contains the client principal of the ticket
  - It also contains the current time of the client
  - It may contain a Checksum in order to protect other fields
  - The GSSAPI-Checksum (0x8003) contains a negotiation structure
  - It may contain a random initiator subkey and sequence number
  - It may contain optional AuthorizationData
- AP-REP provides mutual authentication to the AP-Exchange
  - It is also encrypted with the ticket session key
  - That proves that the service as able to decrypt the ticket
  - It echoes the client time from the Authenticator
  - It may contain a random acceptor subkey and sequence number

The Client/Server Authentication (AP) Exchange (Part1)

- The AP-Exchange authenticates a client to a service
  - The client proves knowledge about the provides Ticket
  - It can be used directly for GSSAPI client to server authentication
  - But it can also be used to authenticate requests to the KDC
- AP-REQ provides a Ticket and an Authenticator
  - The Authenticator is encrypted with the ticket session key
  - The Authenticator contains the client principal of the ticket
  - It also contains the current time of the client
  - It may contain a Checksum in order to protect other fields
  - The GSSAPI-Checksum (0x8003) contains a negotiation structure
  - It may contain a random initiator subkey and sequence number
  - It may contain optional AuthorizationData
- AP-REP provides mutual authentication to the AP-Exchange
  - It is also encrypted with the ticket session key
  - That proves that the service as able to decrypt the ticket
  - It echoes the client time from the Authenticator
  - It may contain a random acceptor subkey and sequence number

The Client/Server Authentication (AP) Exchange (Part1)

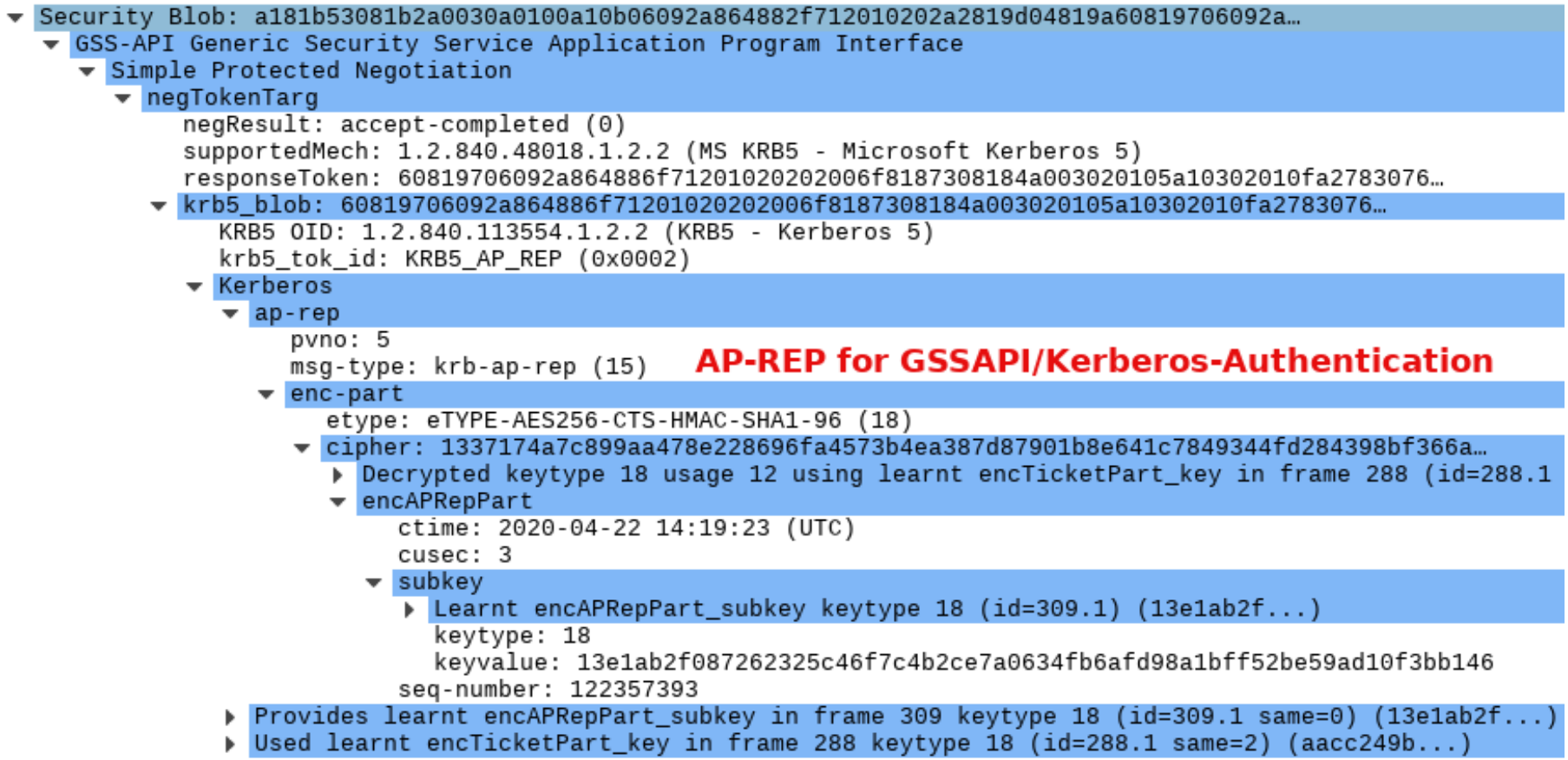
- The AP-Exchange authenticates a client to a service
  - The client proves knowledge about the provides Ticket
  - It can be used directly for GSSAPI client to server authentication
  - But it can also be used to authenticate requests to the KDC
- AP-REQ provides a Ticket and an Authenticator
  - The Authenticator is encrypted with the ticket session key
  - The Authenticator contains the client principal of the ticket
  - It also contains the current time of the client
  - It may contain a Checksum in order to protect other fields
  - The GSSAPI-Checksum (0x8003) contains a negotiation structure
  - It may contain a random initiator subkey and sequence number
  - It may contain optional AuthorizationData
- AP-REP provides mutual authentication to the AP-Exchange
  - It is also encrypted with the ticket session key
  - That proves that the service as able to decrypt the ticket
  - It echoes the client time from the Authenticator
  - It may contain a random acceptor subkey and sequence number

The Client/Server Authentication (AP) Exchange (Part2)

AP-REQ for GSSAPI/Kerberos-Authentication

```
▼ ap-req
    pvno: 5
    msg-type: krb-ap-req (14)
    Padding: 0
  ▶ ap-options: 20000000
  ▶ ticket
  ▼ authenticator
      etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
    ▼ cipher: dda67b22e1d49257a90adfdfe28a13d6d89502e0db982e79ace138b2623aaa808ddcc6ad…
      ▶ Decrypted keytype 18 usage 11 using learnt encTicketPart_key in frame 288 (id=288.1 same=2) (aacc249
      ▼ authenticator
          authenticator-vno: 5
          crealm: W2012R2-L6.BASE
        ▼ cname
            name-type: kRB5-NT-PRINCIPAL (1)
          ▼ cname-string: 1 item
              CNameString: Administrator
        ▼ cksum
            cksumtype: cKSUMTYPE-GSSAPI (32771)
            checksum: 10000000000000000000000000000000000000230000000100b205768205ae308205aa…
            Length: 16
            Bnd: 00000000000000000000000000000000
            .... .... .... .... ...0 .... .... .... = DCE-style: Not using DCE-STYLE
            .... .... .... .... .... .... ..1. .... = Integ: Integrity protection (signing) may be invoked
            .... .... .... .... .... .... ...0 .... = Conf: Do NOT use Confidentiality (sealing)
            .... .... .... .... .... .... .... 0... = Sequence: Do NOT enable out-of-sequence detection
            .... .... .... .... .... .... .... .0.. = Replay: Do NOT enable replay protection
            .... .... .... .... .... .... .... ..1. = Mutual: Request that remote peer authenticates itself
            .... .... .... .... .... .... .... ...1 = Deleg: Delegate credentials to remote peer
            DlgOpt: 1
            DlgLen: 1458
          ▶ krb-cred
          cusec: 3
          ctime: 2020-04-22 14:19:23 (UTC)
        ▶ subkey
          seq-number: 71416561
        ▶ authorization-data: 1 item
```

The Client/Server Authentication (AP) Exchange (Part3)



The Ticket-Granting Service (TGS) Exchange (Part1)

- The TGS-Exchange allows the client/user to tickets for server
  - If a client wants to access a service it needs a service ticket
  - The client can use its TGT to get a service ticket
- TGS-REQ provides an AP-REQ and information about the service
  - The PA-TGS-REQ contains an AP-REQ to authenticate the request
  - The service principal is given in the body.
- TGS-REP typically returns a service ticket
  - The content of the entTicketPart is only known to the service
  - encTGSRepPart is encrypted with the TGT session key
  - encTicketPart and encTGSRepPart contain the same ticket session key
  - The ticket session key is a shared secret between client and server
- TGS-REQ can also return a referral TGT
  - The service principal may be located in different realm
  - A referral TGT looks like krbtgt/SERVER.REALM@CLIENT.REALM
  - The client retries at SERVER.REALM

The Ticket-Granting Service (TGS) Exchange (Part1)

- The TGS-Exchange allows the client/user to tickets for server
  - If a client wants to access a service it needs a service ticket
  - The client can use its TGT to get a service ticket
- TGS-REQ provides an AP-REQ and information about the service
  - The PA-TGS-REQ contains an AP-REQ to authenticate the request
  - The service principal is given in the body.
- TGS-REP typically returns a service ticket
  - The content of the entTicketPart is only known to the service
  - encTGSRepPart is encrypted with the TGT session key
  - encTicketPart and encTGSRepPart contain the same ticket session key
  - The ticket session key is a shared secret between client and server
- TGS-REQ can also return a referral TGT
  - The service principal may be located in different realm
  - A referral TGT looks like krbtgt/SERVER.REALM@CLIENT.REALM
  - The client retries at SERVER.REALM

The Ticket-Granting Service (TGS) Exchange (Part1)

- The TGS-Exchange allows the client/user to tickets for server
  - If a client wants to access a service it needs a service ticket
  - The client can use its TGT to get a service ticket
- TGS-REQ provides an AP-REQ and information about the service
  - The PA-TGS-REQ contains an AP-REQ to authenticate the request
  - The service principal is given in the body.
- TGS-REP typically returns a service ticket
  - The content of the entTicketPart is only known to the service
  - encTGSRepPart is encrypted with the TGT session key
  - encTicketPart and encTGSRepPart contain the same ticket session key
  - The ticket session key is a shared secret between client and server
- TGS-REQ can also return a referral TGT
  - The service principal may be located in different realm
  - A referral TGT looks like krbtgt/SERVER.REALM@CLIENT.REALM
  - The client retries at SERVER.REALM

The Ticket-Granting Service (TGS) Exchange (Part1)

- The TGS-Exchange allows the client/user to tickets for server
  - If a client wants to access a service it needs a service ticket
  - The client can use its TGT to get a service ticket
- TGS-REQ provides an AP-REQ and information about the service
  - The PA-TGS-REQ contains an AP-REQ to authenticate the request
  - The service principal is given in the body.
- TGS-REP typically returns a service ticket
  - The content of the entTicketPart is only known to the service
  - encTGSRepPart is encrypted with the TGT session key
  - encTicketPart and encTGSRepPart contain the same ticket session key
  - The ticket session key is a shared secret between client and server
- TGS-REQ can also return a referral TGT
  - The service principal may be located in different realm
  - A referral TGT looks like krbtgt/SERVER.REALM@CLIENT.REALM
  - The client retries at SERVER.REALM

The Ticket-Granting Service (TGS) Exchange (Part2)

```
▼ tgs-req
    pvno: 5
    msg-type: krb-tgs-req (12)
  ▼ padata: 2 items
    ▼ PA-DATA pA-TGS-REQ
      ▼ padata-type: pA-TGS-REQ (1)
        ▼ padata-value: 6e82053e3082053aa003020105a10302010ea2070305000000000a38204816182047
          ▼ ap-req
              pvno: 5
              msg-type: krb-ap-req (14)
              Padding: 0
            ▶ ap-options: 00000000
            ▶ ticket
            ▼ authenticator
                etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
              ▼ cipher: 7962f94008b22c4f82132ce6f45b5080138c2c660935c529aa35842a6b8921b48ea
                ▶ Decrypted keytype 18 usage 7 using learnt encTicketPart_key in frame 276
                ▼ authenticator
                    authenticator-vno: 5
                    crealm: W2012R2-L6.BASE
                  ▼ cname
                      name-type: kRB5-NT-PRINCIPAL (1)
                    ▼ cname-string: 1 item
                        CNameString: Administrator
                  ▼ cksum
                      cksumtype: cKSUMTYPE-RSA-MD5 (7)
                      checksum: 2e907aefb7c2e901ce1db2e1a26c2557
                    cusec: 1
                    ctime: 2020-04-22 14:19:23 (UTC)
                    seq-number: 71702603
  ▶ PA-DATA pA-PAC-OPTIONS
▼ req-body
    Padding: 0
  ▶ kdc-options: 40810000
    realm: W2012R2-L6.BASE
  ▼ sname
      name-type: kRB5-NT-SRV-INST (2)
    ▼ sname-string: 2 items
        SNameString: cifs
        SNameString: w2012r2-188.w2012r2-l6.base
    till: 2037-09-13 02:48:05 (UTC)
    nonce: 71702603
  ▶ etype: 5 items
  ▶ enc-authorization-data
```

AP-REQ within a TGS-REQ using the TGT from the AS-REP

The Ticket-Granting Service (TGS) Exchange (Part3)

```
▼ tgs-rep
    pvno: 5
    msg-type: krb-tgs-rep (13)
    crealm: W2012R2-L6.BASE
  ▼ cname
      name-type: kRB5-NT-PRINCIPAL (1)
    ▼ cname-string: 1 item
        CNameString: Administrator
  ▶ ticket
  ▼ enc-part
      etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
    ▼ cipher: f9514721510e74ab6aa03b9a630f088c3ddf30e1fc8f8ca5321588d0022df6f0387(
      ▶ Decrypted keytype 18 usage 8 using learnt encTicketPart_key in frame 276
      ▼ encTGSRepPart
        ▶ key
        ▶ last-req: 1 item
          nonce: 71702603
          Padding: 0
        ▶ flags: 40a50000
          authtime: 2020-04-22 14:19:23 (UTC)
          starttime: 2020-04-22 14:19:23 (UTC)
          endtime: 2020-04-23 00:19:23 (UTC)
          renew-till: 2020-04-29 14:19:23 (UTC)
          srealm: W2012R2-L6.BASE
        ▼ sname
            name-type: kRB5-NT-SRV-INST (2)
          ▼ sname-string: 2 items
              SNameString: cifs
              SNameString: w2012r2-188.w2012r2-l6.base
        ▶ encrypted-pa-data: 2 items
```

TGS-REP returns a Service Ticket

encTGSRepPart mirrors:
* the ticket session key
* other details of the ticket

Full GSSAPI-SPNEGO Kerberos Authentication

```
266 16:19:23,633714 172.31.99.189   172.31.9.188    KRB5    AS-REQ
267 16:19:23,635954 172.31.9.188    172.31.99.189   KRB5    KRB Error: KRB5KDC_ERR_PREAUTH_REQUIRED
274 16:19:23,639049 172.31.99.189   172.31.9.188    KRB5    AS-REQ
276 16:19:23,640708 172.31.9.188    172.31.99.189   KRB5    AS-REP
285 16:19:23,643592 172.31.99.189   172.31.9.188    KRB5    TGS-REQ
288 16:19:23,651244 172.31.9.188    172.31.99.189   KRB5    TGS-REP
297 16:19:23,654939 172.31.99.189   172.31.9.188    KRB5    TGS-REQ
300 16:19:23,656231 172.31.9.188    172.31.99.189   KRB5    TGS-REP
307 16:19:23,657824 172.31.99.189   172.31.9.188    SMB2    Session Setup Request
309 16:19:23,659965 172.31.9.188    172.31.99.189   SMB2    Session Setup Response
```

Get TGT

Get Service Ticket

Get Delegation TGT

GSSAPI/SPNEGO

- Client to KDC
  - The client gets a Ticket Granting Ticket (TGT) via the AS-Exchange
  - The client uses the TGT for the TGS-Exchange to get a Service Ticket
  - The Service Ticket may contain OK-AS-DELEGATE
  - If so the client uses the initial TGT to get a fresh delegation TGT
- Client to Service (e.g. SMB server)
  - The client uses the Service ticket for the GSSAPI AP-REQ
  - The GSSAPI-Checksum contains the delegation TGT
  - The delegation is exclusive for the specific server
  - The delegation ticket session key needs to be isolated
  - The server returns an AP-REP with an acceptor subkey
  - The acceptor subkey is the base for signing/encryption

Full GSSAPI-SPNEGO Kerberos Authentication

```
266 16:19:23,633714 172.31.99.189   172.31.9.188    KRB5   AS-REQ
267 16:19:23,635954 172.31.9.188    172.31.99.189   KRB5   KRB Error: KRB5KDC_ERR_PREAUTH_REQUIRED
274 16:19:23,639049 172.31.99.189   172.31.9.188    KRB5   AS-REQ
276 16:19:23,640708 172.31.9.188    172.31.99.189   KRB5   AS-REP
285 16:19:23,643592 172.31.99.189   172.31.9.188    KRB5   TGS-REQ
288 16:19:23,651244 172.31.9.188    172.31.99.189   KRB5   TGS-REP
297 16:19:23,654939 172.31.99.189   172.31.9.188    KRB5   TGS-REQ
300 16:19:23,656231 172.31.9.188    172.31.99.189   KRB5   TGS-REP
307 16:19:23,657824 172.31.99.189   172.31.9.188    SMB2   Session Setup Request
309 16:19:23,659965 172.31.9.188    172.31.99.189   SMB2   Session Setup Response
```

Get TGT

Get Service Ticket

Get Delegation TGT

GSSAPI/SPNEGO

- Client to KDC
  - The client gets a Ticket Granting Ticket (TGT) via the AS-Exchange
  - The client uses the TGT for the TGS-Exchange to get a Service Ticket
  - The Service Ticket may contain OK-AS-DELEGATE
  - If so the client uses the initial TGT to get a fresh delegation TGT
- Client to Service (e.g. SMB server)
  - The client uses the Service ticket for the GSSAPI AP-REQ
  - The GSSAPI-Checksum contains the delegation TGT
  - The delegation is exclusive for the specific server
  - The delegation ticket session key needs to be isolated
  - The server returns an AP-REP with an acceptor subkey
  - The acceptor subkey is the base for signing/encryption

Full GSSAPI-SPNEGO Kerberos Authentication

```
266 16:19:23,633714 172.31.99.189   172.31.9.188    KRB5   AS-REQ
267 16:19:23,635954 172.31.9.188    172.31.99.189   KRB5   KRB Error: KRB5KDC_ERR_PREAUTH_REQUIRED
274 16:19:23,639049 172.31.99.189   172.31.9.188    KRB5   AS-REQ
276 16:19:23,640708 172.31.9.188    172.31.99.189   KRB5   AS-REP
285 16:19:23,643592 172.31.99.189   172.31.9.188    KRB5   TGS-REQ
288 16:19:23,651244 172.31.9.188    172.31.99.189   KRB5   TGS-REP
297 16:19:23,654939 172.31.99.189   172.31.9.188    KRB5   TGS-REQ
300 16:19:23,656231 172.31.9.188    172.31.99.189   KRB5   TGS-REP
307 16:19:23,657824 172.31.99.189   172.31.9.188    SMB2   Session Setup Request
309 16:19:23,659965 172.31.9.188    172.31.99.189   SMB2   Session Setup Response
```

Get TGT

Get Service Ticket

Get Delegation TGT

GSSAPI/SPNEGO

- Client to KDC
  - The client gets a Ticket Granting Ticket (TGT) via the AS-Exchange
  - The client uses the TGT for the TGS-Exchange to get a Service Ticket
  - The Service Ticket may contain OK-AS-DELEGATE
  - If so the client uses the initial TGT to get a fresh delegation TGT
- Client to Service (e.g. SMB server)
  - The client uses the Service ticket for the GSSAPI AP-REQ
  - The GSSAPI-Checksum contains the delegation TGT
  - The delegation is exclusive for the specific server
  - The delegation ticket session key needs to be isolated
  - The server returns an AP-REP with an acceptor subkey
  - The acceptor subkey is the base for signing/encryption

S4U, FAST, Compound Identity

- S4U2Self/S4U2Proxy ([MS-SFU]):
  - Allow the usage of Kerberos of an impersonated user
  - Typically when the frontend authentication didn't use Kerberos
- Flexible Authentication Secure Tunneling (FAST) (RFC6113):
  - Protects the AS-REQ with a relative weak user password
  - The protection is based on the strong machine account password
  - It prevents offline dictionary attacks
  - It allows Compound Identities to be used
  - The PAC within service tickets contains a DEVICE_INFO element
  - The DEVICE_INFO contains a subset of the machine accounts LOGON_INFO
  - The service sees from on which device the client was authenticated

S4U, FAST, Compound Identity

- S4U2Self/S4U2Proxy ([MS-SFU]):
  - Allow the usage of Kerberos of an impersonated user
  - Typically when the frontend authentication didn't use Kerberos
- Flexible Authentication Secure Tunneling (FAST) (RFC6113):
  - Protects the AS-REQ with a relative weak user password
  - The protection is based on the strong machine account password
  - It prevents offline dictionary attacks
  - It allows Compound Identities to be used
  - The PAC within service tickets contains a DEVICE_INFO element
  - The DEVICE_INFO contains a subset of the machine accounts LOGON_INFO
  - The service sees from on which device the client was authenticated

S4U2Self Request (Part1)

```
tgs-req
    pvno: 5
    msg-type: krb-tgs-req (12)
  padata: 4 items
    PA-DATA pA-TGS-REQ
      padata-type: pA-TGS-REQ (1)
        padata-value: 6e8205b1308205ada003020105a10302010ea2070305000000000a38204ca618204c630…
          ap-req
              pvno: 5
              msg-type: krb-ap-req (14)
              Padding: 0
            ap-options: 00000000
            ticket
            authenticator
                etype: eTYPE-ARCFOUR-HMAC-MD5 (23)
              cipher: 15060a7e25ee362cf53e7f2104c9e4b485acf56e172754542a32795119e149b957860cbe…
                Decrypted keytype 23 usage 7 using learnt encTicketPart_key in frame 548 (id=548.1 same=2)
                authenticator
                    authenticator-vno: 5
                    crealm: S2-W2012-L4.S1-W2012-L4.W2012R2-L4.BASE
                  cname
                      name-type: kRB5-NT-PRINCIPAL (1)
                    cname-string: 1 item
                        CNameString: UB1604-165$
                  cksum
                      cksumtype: cKSUMTYPE-RSA-MD5 (7)
                      checksum: 519fc74e4afe7cbcbad71ef27b1bdf52
                    cusec: 1954
                    ctime: 2020-01-27 12:58:49 (UTC)
                  subkey
    PA-DATA pA-FX-FAST
    PA-DATA pA-FOR-X509-USER
      padata-type: pA-FOR-X509-USER (130)
        padata-value:
          user-id
              nonce: 617889277
            cname
                name-type: kRB5-NT-ENTERPRISE-PRINCIPAL (10)
              name-string: 1 item
                  KerberosString: somebla2@BLA2
              crealm: BLA2.BASE
              Padding: 0
              options: 20000000
          checksum
    PA-DATA pA-FOR-USER
      padata-type: pA-FOR-USER (129)
        padata-value:
          name
              name-type: kRB5-NT-ENTERPRISE-PRINCIPAL (10)
            name-string: 1 item
                KerberosString: somebla2@BLA2
              realm: BLA2.BASE
            cksum
              auth: Kerberos
  req-body
```

S4U2Self Request uses the server's TGT

PA-FOR-X509-USER:
* **Modern way for S4U2Self**
* **Missing in Samba KDCs**
* **A client principal or X509-Certificate can be used to indentify the user**
* **Enterprise Principal are supported by Windows KDCs**

PA-FOR-USER:
* **Legacy way for S4U2Self**
* **Also supported in Samba KDCs**
* **Can only specifiy the client principal**
* **Enterprise Principals doesn't seem to work against Windows KDCs**

S4U2Self Request (Part2)

S2-W2012-L4...		411	KRB5	AS-REQ
		412	KRB5	KRB Error: KRB5KDC_ERR_PREAUTH_REQUIRED
		422	KRB5	AS-REQ **TGT for UB1604-165$@S2-W2012-L4.S1-W2012-L4.W2012R2-L4.BASE**
		425	KRB5	AS-REP
		433	KRB5	AS-REQ **AS-REQ for somebla2@BLA2@S2-W2012-L4...**
		434	KRB5	KRB Error: KDC_ERR_WRONG_REALM **Referred to bla.base**
BLA.BASE		449	KRB5	AS-REQ **AS-REQ for somebla2@BLA2@BLA.BASE**
		450	KRB5	KRB Error: KDC_ERR_WRONG_REALM **Referred to bla2.base**
BLA2.BASE		466	KRB5	AS-REQ **AS-REQ for somebla2@BLA2@BLA2.BASE**
		467	KRB5	KRB Error: KRB5KDC_ERR_PREAUTH_REQUIRED **=> BLA2.BASE knows it**
S2-W2012-L4...		475	KRB5	TGS-REQ **Request: krbtgt/BLA2.BASE@S2-W2012-L4...**
		479	KRB5	TGS-REP **=> Referral TGT: krbtgt/S1-W2012-L4...@S2-W2012-L4...**
S1-W2012-L4...		501	KRB5	TGS-REQ **Request: krbtgt/BLA2.BASE@S1-W2012-L4...**
		505	KRB5	TGS-REP **=> Referral TGT: krbtgt/W2012R2-L4...@S1-W2012-L4...**
	RODC	522	KRB5	TGS-REQ **Request: krbtgt/BLA2.BASE@W2012R2-L4.BASE**
W2012R2-L4...	RWDC	527	KRB5	TGS-REQ **=> Proxied from RODC to RWDC**
	RWDC	529	KRB5	TGS-REP **=> Back from RWDC to RODC**
	RODC	535	KRB5	TGS-REP **=> Referral TGT: krbtgt/BLA.BASE@W2012R2-L4.BASE**
BLA.BASE		544	KRB5	TGS-REQ **Request: krbtgt/BLA2.BASE@BLA.BASE**
		548	KRB5	TGS-REP **=> Final-Referral TGT: krbtgt/BLA2.BASE@BLA.BASE**
BLA2.BASE		556	KRB5	TGS-REQ **S4U2Self for host/UB1604-165.S2-W2012-L4...@BLA2.BASE**
		560	KRB5	TGS-REP **=> Referral TGT: krbtgt/BLA.BASE@BLA2.BASE S4U2Self-PAC**
BLA.BASE		568	KRB5	TGS-REQ **S4U2Self for host/UB1604-165.S2-W2012-L4...@BLA.BASE**
		574	KRB5	TGS-REP **=> Referral TGT: krbtgt/W2012R2-L4...@BLA.BASE S4U2Self-PAC**
	RODC	582	KRB5	TGS-REQ **S4U2Self for host/UB1604-165.S2-W2012-L4...@W2012R2-L4.BASE**
W2012R2-L4...	RWDC	587	KRB5	TGS-REQ **=> Proxied from RODC to RWDC**
	RWDC	589	KRB5	TGS-REP **=> Back from RWDC to RODC**
	RODC	595	KRB5	TGS-REP **=> Referral TGT: krbtgt/S1-W2012-L4...@W2012R2-L4.BASE S4U2Self-PAC**
S1-W2012-L4...		604	KRB5	TGS-REQ **S4U2Self for host/UB1604-165.S2-W2012-L4...@S1-W2012-L4...**
		608	KRB5	TGS-REP **=> Referral TGT: krbtgt/S2-W2012-L4...@S1-W2012-L4... S4U2Self-PAC**
S2-W2012-L4...		616	KRB5	TGS-REQ **S4U2Self for host/UB1604-165.S2-W2012-L4...@S2-W2012-L4...**
		620	KRB5	TGS-REP **S4U2Self Ticket for somebla2@BLA2@BLA2.BASE**

AS-REQ with FAST

TGS-REQ with FAST, Compound Identity

```
▼ tgs-req
    pvno: 5
    msg-type: krb-tgs-req (12)
  ▼ padata: 2 items
    ▼ PA-DATA pA-TGS-REQ
      ▼ padata-type: pA-TGS-REQ (1)
        ▼ padata-value: 6e8205a5308205a1a003020105a10302010ea20703050000000000a38204b9618204b530…
          ▼ ap-req
              pvno: 5
              msg-type: krb-ap-req (14)
              Padding: 0
            ▶ ap-options: 00000000
            ▶ ticket
            ▶ authenticator
    ▼ PA-DATA pA-FX-FAST
      ▼ padata-type: pA-FX-FAST (136)
        ▼ padata-value: a082066730820663a08205733082056fa003020101a1820566048205626e82055e308205…
          ▼ armored-data
            ▼ armor
                armor-type: fX-FAST-ARMOR-AP-REQUEST (1)
              ▼ armor-value: 6e82055e3082055aa003020105a10302010ea20703050000000000a38204906182048c30…
                ▼ ap-req
                    pvno: 5
                    msg-type: krb-ap-req (14)
                    Padding: 0
                  ▶ ap-options: 00000000
                  ▶ ticket
                  ▶ authenticator
            ▶ req-checksum
            ▼ enc-fast-req
                etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
              ▼ cipher: bc18410514074ca4e15a236f7c43f5e4e8e4cb809865d6dbf1212e61c04ac1116c901a24…
                ▼ Derived KrbFastReq_FAST_armorKey keytype 18 (id=743.5) (02f51922...)
                  ▶ [Expert Info (Chat/Security): Derived KrbFastReq_FAST_armorKey keytype 18 (id=743.5) (
                  ▶ [Expert Info (Chat/Security): SRC1 learnt authenticator_subkey in frame 743 keytype 18
                  ▶ [Expert Info (Chat/Security): SRC2 learnt encTicketPart_key in frame 143 keytype 18 (i
                ▼ Derived KrbFastReq_explicitArmorKey keytype 18 (id=743.6) (449a420e...)
                  ▶ [Expert Info (Chat/Security): Derived KrbFastReq_explicitArmorKey keytype 18 (id=743.6
                  ▶ [Expert Info (Chat/Security): SRC1 derived KrbFastReq_FAST_armorKey in frame 743 keyty
                  ▶ [Expert Info (Chat/Security): SRC2 learnt authenticator_subkey in frame 743 keytype 18
                ▶ Decrypted keytype 18 usage 51 using derived KrbFastReq_explicitArmorKey in frame 743 (id=
                  Padding: 0
                ▶ fast-options: 00000000
                ▶ padata: 1 item
                ▼ req-body
                    Padding: 0
                  ▶ kdc-options: 40810000
                    realm: W2012R2-L6.BASE
                  ▶ sname
                    till: 2037-09-13 02:48:05 (UTC)
                    nonce: 2000422842
                  ▶ etype: 5 items
▶ req-body
```

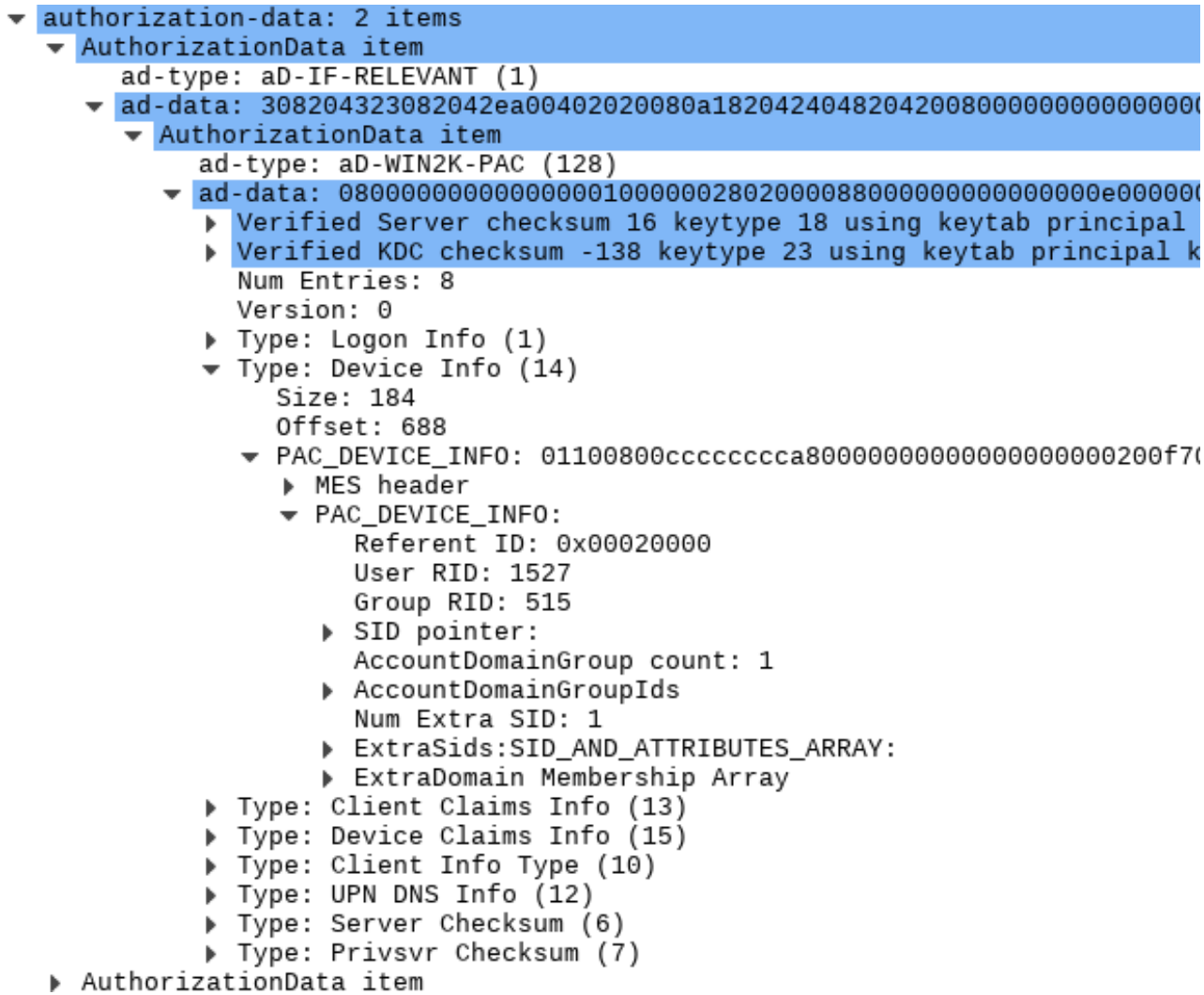
TGS-REQ with a user TGT

Armored with a machine account TGT:
*** used for the FAST_armorKey**

The explicitArmorKey combines:
*** FAST_armorKey**
*** PA_TGS_REQ authenticator_subkey**

This forms the Compound Identity

PAC with DEVICE_INFO for Compound Identity



Using S4U2Self in winbindd (Part1)

- winbindd provides group membership information for users
  - For tools like 'id', 'wbinfo -i', 'wbinfo –user-sids' and others
- Typically winbindd gets the Authorization Token via authentication
  - Eiter via netr_LogonSamLogon vor NTLM
  - Or via the "PAC Logon Info" element of the Kerberos service ticket
- There're some situations when a service needs to impersonate a user locally:
  - This can happen without getting an authentication for that user.
  - SSH public-key authentication, sudo or nfs3 access are tyipical use cases.

Using S4U2Self in winbindd (Part1)

- winbindd provides group membership information for users
  - For tools like 'id', 'wbinfo -i', 'wbinfo –user-sids' and others
- Typically winbindd gets the Authorization Token via authentication
  - Eiter via netr_LogonSamLogon vor NTLM
  - Or via the "PAC Logon Info" element of the Kerberos service ticket
- There're some situations when a service needs to impersonate a user locally:
  - This can happen without getting an authentication for that user.
  - SSH public-key authentication, sudo or nfs3 access are tyipical use cases.

Using S4U2Self in winbindd (Part1)

- winbindd provides group membership information for users
  - For tools like 'id', 'wbinfo -i', 'wbinfo –user-sids' and others
- Typically winbindd gets the Authorization Token via authentication
  - Eiter via netr_LogonSamLogon vor NTLM
  - Or via the "PAC Logon Info" element of the Kerberos service ticket
- There're some situations when a service needs to impersonate a user locally:
  - This can happen without getting an authentication for that user.
  - SSH public-key authentication, sudo or nfs3 access are tyipical use cases.

Using S4U2Self in winbindd (Part2)

- winbindd tries to get the 'tokenGroups' of the user object via LDAP
  - There're a lot of situations where this doesn't work, e.g. with OUTBOUND only trusts.
  - It is a very hard task because the expanding and filtering at the trust boundaries of the transitive chain can't be simulated.
  - So the result is often wrong!
- The only reliable solution is S4U2Self ([MS-SFU]):
  - It allows a service to ask a KDC for a service ticket for a given user.
  - From a given SID we can only lookup the NT4-Names of the account
  - We need to use Enterprise-Principals like, user@DOMAIN1@DOMAIN2.EXAMPLE.COM
  - Sadly there're quite some bugs in current versions of MIT Kerberos and Heimdal (both client and server)

Using S4U2Self in winbindd (Part2)

- winbindd tries to get the 'tokenGroups' of the user object via LDAP
  - There're a lot of situations where this doesn't work, e.g. with OUTBOUND only trusts.
  - It is a very hard task because the expanding and filtering at the trust boundaries of the transitive chain can't be simulated.
  - So the result is often wrong!
- The only reliable solution is S4U2Self ([MS-SFU]):
  - It allows a service to ask a KDC for a service ticket for a given user.
  - From a given SID we can only lookup the NT4-Names of the account
  - We need to use Enterprise-Principals like, user@DOMAIN1@DOMAIN2.EXAMPLE.COM
  - Sadly there're quite some bugs in current versions of MIT Kerberos and Heimdal (both client and server)

krb5_{init,tkt}_creds_step() APIs (Part1)

- The usage of S4U2Self with trusted domains/realms is complex:
  - The example showed that a lot of transiting KDCs must be reached
  - We should use site-aware KDCs (domain controllers) for all steps
- Currently winbindd prepares a custom krb5.conf
  - It fills in the KDC ip addresses for the default realm
  - But it's not possible to know all hops before calling krb5 functions
- It would be good if the Kerberos libraries would only do Kerberos
  - We can do (site-aware) DC lookups much more efficient.
  - It would be good to do the networking interaction on our own.
  - We should do parallel async requests in order to avoid long timeouts.

krb5_{init,tkt}_creds_step() APIs (Part1)

- The usage of S4U2Self with trusted domains/realms is complex:
  - The example showed that a lot of transiting KDCs must be reached
  - We should use site-aware KDCs (domain controllers) for all steps
- Currently winbindd prepares a custom krb5.conf
  - It fills in the KDC ip addresses for the default realm
  - But it's not possible to know all hops before calling krb5 functions
- It would be good if the Kerberos libraries would only do Kerberos
  - We can do (site-aware) DC lookups much more efficient.
  - It would be good to do the networking interaction on our own.
  - We should do parallel async requests in order to avoid long timeouts.

krb5_{init,tkt}_creds_step() APIs (Part1)

- The usage of S4U2Self with trusted domains/realms is complex:
  - The example showed that a lot of transiting KDCs must be reached
  - We should use site-aware KDCs (domain controllers) for all steps
- Currently winbindd prepares a custom krb5.conf
  - It fills in the KDC ip addresses for the default realm
  - But it's not possible to know all hops before calling krb5 functions
- It would be good if the Kerberos libraries would only do Kerberos
  - We can do (site-aware) DC lookups much more efficient.
  - It would be good to do the networking interaction on our own.
  - We should do parallel async requests in order to avoid long timeouts.

krb5_{init,tkt}_creds_step() APIs (Part2)

- There are step APIs, which allow doing things on our own:
  - They just generate Request PDUs and return the designated realm
  - The result from a KDC should be passed in the next round
  - This continues as long as the CONTINUE flag is returned.

```
krb5_error_code
krb5_init_creds_step(krb5_context context,
                     krb5_init_creds_context ctx,
                     krb5_data *in,
                     krb5_data *out,
                     krb5_realm *realm,
                     unsigned int *flags); /* ..._CONTINUE flag */
krb5_error_code
krb5_tkt_creds_step(krb5_context context,
                    krb5_tkt_creds_context ctx,
                    krb5_data *in,
                    krb5_data *out,
                    krb5_realm *realm,
                    unsigned int *flags); /* ..._CONTINUE flag */
```

- It's ideal for us, but they are sadly not feature complete:
  - MIT doesn't allow S4USelf and S4U2Proxy via these APIs
  - Heimdal has only an unexported krb5_init_creds_step() function
  - There are work in progress patches for MIT and Heimdal

krb5_{init,tkt}_creds_step() APIs (Part2)

- There are step APIs, which allow doing things on our own:
  - They just generate Request PDUs and return the designated realm
  - The result from a KDC should be passed in the next round
  - This continues as long as the CONTINUE flag is returned.

```
krb5_error_code
krb5_init_creds_step(krb5_context context,
                     krb5_init_creds_context ctx,
                     krb5_data *in,
                     krb5_data *out,
                     krb5_realm *realm,
                     unsigned int *flags); /* ..._CONTINUE flag */
krb5_error_code
krb5_tkt_creds_step(krb5_context context,
                    krb5_tkt_creds_context ctx,
                    krb5_data *in,
                    krb5_data *out,
                    krb5_realm *realm,
                    unsigned int *flags); /* ..._CONTINUE flag */
```

- It's ideal for us, but they are sadly not feature complete:
  - MIT doesn't allow S4USelf and S4U2Proxy via these APIs
  - Heimdal has only an unexported krb5_init_creds_step() function
  - There are work in progress patches for MIT and Heimdal

krb5_{init,tkt}_creds_step() APIs (Part2)

- There are step APIs, which allow doing things on our own:
  - They just generate Request PDUs and return the designated realm
  - The result from a KDC should be passed in the next round
  - This continues as long as the CONTINUE flag is returned.

```
krb5_error_code
krb5_init_creds_step(krb5_context context,
                     krb5_init_creds_context ctx,
                     krb5_data *in,
                     krb5_data *out,
                     krb5_realm *realm,
                     unsigned int *flags); /* ..._CONTINUE flag */
krb5_error_code
krb5_tkt_creds_step(krb5_context context,
                    krb5_tkt_creds_context ctx,
                    krb5_data *in,
                    krb5_data *out,
                    krb5_realm *realm,
                    unsigned int *flags); /* ..._CONTINUE flag */
```

- It's ideal for us, but they are sadly not feature complete:
  - MIT doesn't allow S4USelf and S4U2Proxy via these APIs
  - Heimdal has only an unexported krb5_init_creds_step() function
  - There are work in progress patches for MIT and Heimdal

krb5_{init,tkt}_creds_step() APIs (Part3)

- For Samba we need to have async non-blocking functions:
  - Async programming in Samba use the tevent_req infrastructure
- We abstract the network details in 'struct smb_krb5_network':
  - This allows us to use different strategies
  - winbindd may use a different strategie than cmdline tools
  - It also avoids linking dependencies.

```
struct tevent_req *smb_krb5_network_transaction_send(
                                TALLOC_CTX *mem_ctx,
                                struct tevent_context *ev,
                                struct smb_krb5_network *net_ctx,
                                const char *realm,
                                uint32_t ds_flags, /* netr_DsRGetDCName_flags */
                                const DATA_BLOB req_blob);
NTSTATUS smb_krb5_network_transaction_recv(struct tevent_req *req,
                                           TALLOC_CTX *mem_ctx,
                                           DATA_BLOB *rep_blob);
```

krb5_{init,tkt}_creds_step() APIs (Part3)

- For Samba we need to have async non-blocking functions:
  - Async programming in Samba use the tevent_req infrastructure
- We abstract the network details in 'struct smb_krb5_network':
  - This allows us to use different strategies
  - winbindd may use a different strategie than cmdline tools
  - It also avoids linking dependencies.

```
struct tevent_req *smb_krb5_network_transaction_send(
                                TALLOC_CTX *mem_ctx,
                                struct tevent_context *ev,
                                struct smb_krb5_network *net_ctx,
                                const char *realm,
                                uint32_t ds_flags, /* netr_DsRGetDCName_flags */
                                const DATA_BLOB req_blob);
NTSTATUS smb_krb5_network_transaction_recv(struct tevent_req *req,
                                           TALLOC_CTX *mem_ctx,
                                           DATA_BLOB *rep_blob);
```

krb5_{init,tkt}_creds_step() APIs (Part3)

- For Samba we need to have async non-blocking functions:
  - Async programming in Samba use the tevent_req infrastructure
- We abstract the network details in 'struct smb_krb5_network':
  - This allows us to use different strategies
  - winbindd may use a different strategie than cmdline tools
  - It also avoids linking dependencies.

```
struct tevent_req *smb_krb5_network_transaction_send(
                                TALLOC_CTX *mem_ctx,
                                struct tevent_context *ev,
                                struct smb_krb5_network *net_ctx,
                                const char *realm,
                                uint32_t ds_flags, /* netr_DsRGetDCName_flags */
                                const DATA_BLOB req_blob);
NTSTATUS smb_krb5_network_transaction_recv(struct tevent_req *req,
                                           TALLOC_CTX *mem_ctx,
                                           DATA_BLOB *rep_blob);
```

krb5_{init,tkt}_creds_step() APIs (Part4)

- In combination we'll have the following low level functions
  - They build the foundation for more complex things
  - We'll have only one GENSEC gsskrb5 implementation
  - S4U2Self, S4U2Proxy can be implemented on top

```
struct tevent_req *smb_krb5_init_creds_get_send(
        TALLOC_CTX *mem_ctx,
        struct tevent_context *ev,
        struct smb_krb5_network *net_ctx,
        krb5_context krb5_ctx,
        krb5_init_creds_context init_creds_ctx);
NTSTATUS smb_krb5_init_creds_get_recv(struct tevent_req *req);

struct tevent_req *smb_krb5_tkt_creds_get_send(
        TALLOC_CTX *mem_ctx,
        struct tevent_context *ev,
        struct smb_krb5_network *net_ctx,
        krb5_context krb5_ctx,
        krb5_tkt_creds_context tkt_creds_ctx);
NTSTATUS smb_krb5_tkt_creds_get_recv(struct tevent_req *req);
```

krb5_{init,tkt}_creds_step() APIs (Part4)

- In combination we'll have the following low level functions
  - They build the foundation for more complex things
  - We'll have only one GENSEC gsskrb5 implementation
  - S4U2Self, S4U2Proxy can be implemented on top

```
struct tevent_req *smb_krb5_init_creds_get_send(
        TALLOC_CTX *mem_ctx,
        struct tevent_context *ev,
        struct smb_krb5_network *net_ctx,
        krb5_context krb5_ctx,
        krb5_init_creds_context init_creds_ctx);
NTSTATUS smb_krb5_init_creds_get_recv(struct tevent_req *req);

struct tevent_req *smb_krb5_tkt_creds_get_send(
        TALLOC_CTX *mem_ctx,
        struct tevent_context *ev,
        struct smb_krb5_network *net_ctx,
        krb5_context krb5_ctx,
        krb5_tkt_creds_context tkt_creds_ctx);
NTSTATUS smb_krb5_tkt_creds_get_recv(struct tevent_req *req);
```

Highlevel Samba APIs (Part1)

- At the application level we'll have some simple functions
  - The most common thing is a login into the local machine
  - This would be used for pam_winbind with Kerberos
  - We use the common cli_credentials abstraction for user and machine

APIs for a local Kerberos login, e.g. in winbindd:

```
struct tevent_req *smb_krb5_kinit_login_send(TALLOC_CTX *mem_ctx,
                                             struct tevent_context *ev,
                                             struct loadparm_context *lp_ctx,
                                             struct cli_credentials *user_creds,
                                             const char *machine_spn,
                                             struct cli_credentials *machine_creds,
                                             struct gensec_settings *gensec_settings,
                                             struct auth4_context *auth_context);
NTSTATUS smb_krb5_kinit_login_recv(struct tevent_req *req,
                                   TALLOC_CTX *mem_ctx,
                                   struct auth_session_info **_session_info);
NTSTATUS smb_krb5_kinit_login(struct loadparm_context *lp_ctx,
                              struct cli_credentials *user_creds,
                              const char *machine_principal,
                              struct cli_credentials *machine_creds,
                              struct gensec_settings *gensec_settings,
                              struct auth4_context *auth_context,
                              TALLOC_CTX *mem_ctx,
                              struct auth_session_info **_session_info);
```

Highlevel Samba APIs (Part1)

- At the application level we'll have some simple functions
  - The most common thing is a login into the local machine
  - This would be used for pam_winbind with Kerberos
  - We use the common cli_credentials abstraction for user and machine

APIs for a local Kerberos login, e.g. in winbindd:

```
struct tevent_req *smb_krb5_kinit_login_send(TALLOC_CTX *mem_ctx,
                                             struct tevent_context *ev,
                                             struct loadparm_context *lp_ctx,
                                             struct cli_credentials *user_creds,
                                             const char *machine_spn,
                                             struct cli_credentials *machine_creds,
                                             struct gensec_settings *gensec_settings,
                                             struct auth4_context *auth_context);
NTSTATUS smb_krb5_kinit_login_recv(struct tevent_req *req,
                                   TALLOC_CTX *mem_ctx,
                                   struct auth_session_info **_session_info);
NTSTATUS smb_krb5_kinit_login(struct loadparm_context *lp_ctx,
                              struct cli_credentials *user_creds,
                              const char *machine_principal,
                              struct cli_credentials *machine_creds,
                              struct gensec_settings *gensec_settings,
                              struct auth4_context *auth_context,
                              TALLOC_CTX *mem_ctx,
                              struct auth_session_info **_session_info);
```

Highlevel Samba APIs (Part2)

- In order to use S4U2Self we'll have a simple function
  - It takes the machine account credentials
  - And the user principal for the impersonated user
  - It creates a special cli_credentials structure
  - This can be used as any other cli_credentials object
  - Typically as user_creds for smb_krb5_kinit_login()

APIs for S4U2Self, e.g. in winbindd:

```
NTSTATUS cli_credentials_s4u_upn_creds(TALLOC_CTX *mem_ctx,
                                       struct cli_credentials *machine_creds,
                                       const char *machine_spn,
                                       const char *user_upn,
                                       struct cli_credentials **_s4u_user_creds);
```

Highlevel Samba APIs (Part2)

- In order to use S4U2Self we'll have a simple function
  - It takes the machine account credentials
  - And the user principal for the impersonated user
  - It creates a special cli_credentials structure
  - This can be used as any other cli_credentials object
  - Typically as user_creds for smb_krb5_kinit_login()

APIs for S4U2Self, e.g. in winbindd:

```
NTSTATUS cli_credentials_s4u_upn_creds(TALLOC_CTX *mem_ctx,
                                       struct cli_credentials *machine_creds,
                                       const char *machine_spn,
                                       const char *user_upn,
                                       struct cli_credentials **_s4u_user_creds);
```

Highlevel Samba APIs (Part3)

- In order to use FAST for Compound Identity we'll have a simple function
  - It takes the machine account credentials
  - And the user credentials
  - It creates a special cli_credentials structure
  - This can be used as any other cli_credentials object
  - Typically as user_creds for smb_krb5_kinit_login()

APIs for FAST, CompoundIdentity, e.g. in winbindd:

```
NTSTATUS cli_credentials_compound_creds(TALLOC_CTX *mem_ctx,
                                        struct cli_credentials *machine_creds,
                                        struct cli_credentials *user_creds,
                                        struct cli_credentials **_compound_user_creds);
```

Highlevel Samba APIs (Part3)

- In order to use FAST for Compound Identity we'll have a simple function
  - It takes the machine account credentials
  - And the user credentials
  - It creates a special cli_credentials structure
  - This can be used as any other cli_credentials object
  - Typically as user_creds for smb_krb5_kinit_login()

APIs for FAST, CompoundIdentity, e.g. in winbindd:

```
NTSTATUS cli_credentials_compound_creds(TALLOC_CTX *mem_ctx,
                                        struct cli_credentials *machine_creds,
                                        struct cli_credentials *user_creds,
                                        struct cli_credentials **_compound_user_creds);
```

Challenges of adding new Features (Part1)

- Adding the missing features to upstream MIT and Heimdal
  - We need to do quite a bit as research to find how the protocols works
  - New features to be added for Samba should be complete
  - Libraries with half implemented features are useless
  - They would make the code in Samba way too complex to work with
  - We would not be able to test all combinations!
  - We found more than once: untested code is broken code!
- It's also very time consuming to discuss the correct APIs
  - Upstream MIT/Heimdal may reject changes, which use legacy concepts
- Currently we need to handle 3 different Kerberos libraries:
  - External MIT
  - External Heimdal
  - Internal Heimdal (imported copy of upstream from 2011)

Challenges of adding new Features (Part1)

- Adding the missing features to upstream MIT and Heimdal
  - We need to do quite a bit as research to find how the protocols works
  - New features to be added for Samba should be complete
  - Libraries with half implemented features are useless
  - They would make the code in Samba way too complex to work with
  - We would not be able to test all combinations!
  - We found more than once: untested code is broken code!
- It's also very time consuming to discuss the correct APIs
  - Upstream MIT/Heimdal may reject changes, which use legacy concepts
- Currently we need to handle 3 different Kerberos libraries:
  - External MIT
  - External Heimdal
  - Internal Heimdal (imported copy of upstream from 2011)

Challenges of adding new Features (Part1)

- Adding the missing features to upstream MIT and Heimdal
  - We need to do quite a bit as research to find how the protocols works
  - New features to be added for Samba should be complete
  - Libraries with half implemented features are useless
  - They would make the code in Samba way too complex to work with
  - We would not be able to test all combinations!
  - We found more than once: untested code is broken code!
- It's also very time consuming to discuss the correct APIs
  - Upstream MIT/Heimdal may reject changes, which use legacy concepts
- Currently we need to handle 3 different Kerberos libraries:
  - External MIT
  - External Heimdal
  - Internal Heimdal (imported copy of upstream from 2011)

Challenges of adding new Features (Part2)

- Syncing the internal Heimdal with upstream
  - This would make things much easier for new features
  - It would bring support for FAST, which would also help the AD DC
  - But it comes with a risk of breaking AD DC setups
- We currently only have very limited Kerberos testing
  - We only do highlevel tests with gssapi usage
  - We have some special tests abusing send_to_kdc hooks
  - The interaction with send_to_kdc depends on implementation details
  - We don't have real protocol detail testing

Challenges of adding new Features (Part2)

- Syncing the internal Heimdal with upstream
  - This would make things much easier for new features
  - It would bring support for FAST, which would also help the AD DC
  - But it comes with a risk of breaking AD DC setups
- We currently only have very limited Kerberos testing
  - We only do highlevel tests with gssapi usage
  - We have some special tests abusing send_to_kdc hooks
  - The interaction with send_to_kdc depends on implementation details
  - We don't have real protocol detail testing

Protocol Testing with Python

- We recently added infrastructure for protocol tests:
  - This is based on pyasn1 and cryptography.hazmat
  - It allows testing each bit in the protocol
  - Very similar to our DCERPC raw_protocol testing and smbtorture
- We have just some simple tests
  - But it's relatively easy to add more detailed tests
  - They will make it much easier to upgrade Heimdal safely
  - It will also add confidence when making the MIT KDC production ready
- Researching new features
  - Protocol tests help finding details about S4U2Self or FAST
  - Much easier than protyping than the C libraries
  - Wireshark Kerberos decryption also helps a lot
  - wireshark/master (~3.3.0) from yesterday has a much improved kerberos dissector

Protocol Testing with Python

- We recently added infrastructure for protocol tests:
  - This is based on pyasn1 and cryptography.hazmat
  - It allows testing each bit in the protocol
  - Very similar to our DCERPC raw_protocol testing and smbtorture
- We have just some simple tests
  - But it's relatively easy to add more detailed tests
  - They will make it much easier to upgrade Heimdal safely
  - It will also add confidence when making the MIT KDC production ready
- Researching new features
  - Protocol tests help finding details about S4U2Self or FAST
  - Much easier than protyping than the C libraries
  - Wireshark Kerberos decryption also helps a lot
  - wireshark/master (~3.3.0) from yesterday has a much improved kerberos dissector

Protocol Testing with Python

- We recently added infrastructure for protocol tests:
  - This is based on pyasn1 and cryptography.hazmat
  - It allows testing each bit in the protocol
  - Very similar to our DCERPC raw_protocol testing and smbtorture
- We have just some simple tests
  - But it's relatively easy to add more detailed tests
  - They will make it much easier to upgrade Heimdal safely
  - It will also add confidence when making the MIT KDC production ready
- Researching new features
  - Protocol tests help finding details about S4U2Self or FAST
  - Much easier than protyping than the C libraries
  - Wireshark Kerberos decryption also helps a lot
  - wireshark/master (~3.3.0) from yesterday has a much improved kerberos dissector

Questions?

- Stefan Metzmacher, metze@samba.org
- https://www.sernet.com
- https://samba.plus

Slides: https://samba.org/~metze/presentations/2020/SambaXP/