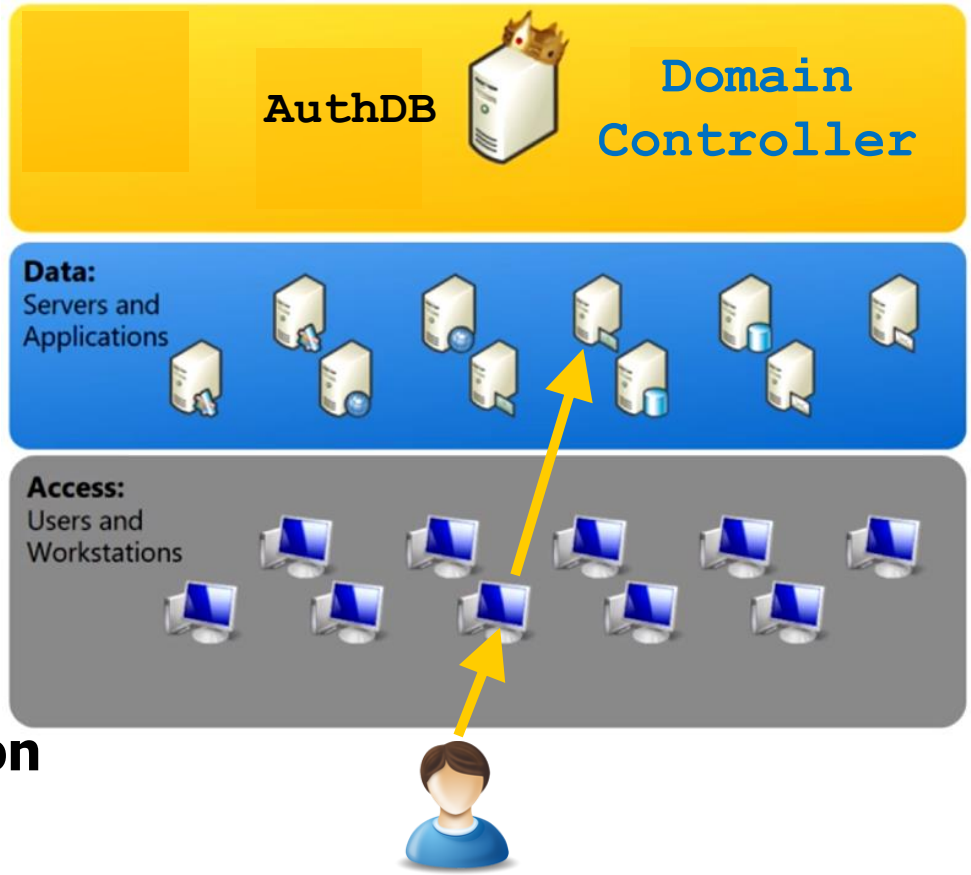
Authentication in Windows Active Directory

Windows Preliminaries

REMIND

- Every IT object stored in DC
  - Identities
  - Credentials
  - Resources
  - Access Rights



- Each resource executes **authentication** and **authorization** by interacting with DC

Local Accounts vs Domain Accounts

- **Local** accounts:
  - Defined in **SAM Database** (Security Accounts Manager)
- **Domain** accounts:
  - Defined in **Domain Controller**
  - File `c:\Windows\NTDS\NTDS.dit`

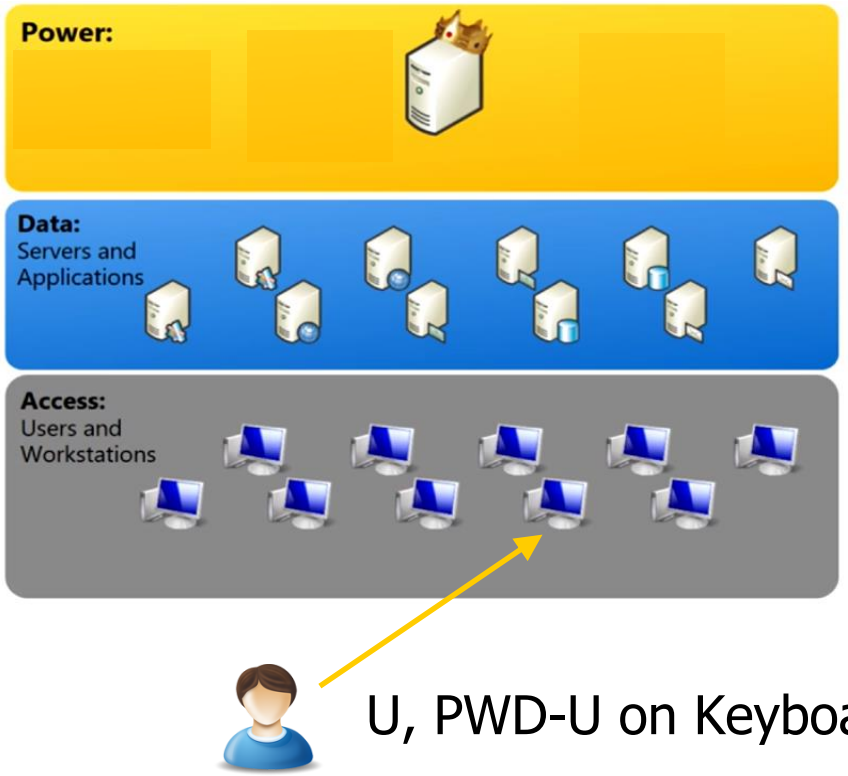
- Logon on computer C allowed to:
  - **Local** accounts of C
  - **Domain** accounts with **logon access right** on C

Accounts: User, Computer, Service

- **User** accounts — human operators
- **Service** accounts — server applications
- **Computer** accounts — machines

- Computer account:
  - Full privilege on local resources
  - Password "impossible to guess"
  - Name ends with '$'
- Names of Local accounts:
  - `WORKGROUP\name` — User, Service
  - `NT AUTHORITY\SYSTEM` — Computer
- Names of Domain accounts
  - `domain_name\name` — User, Service, Computer

Interactive Logon

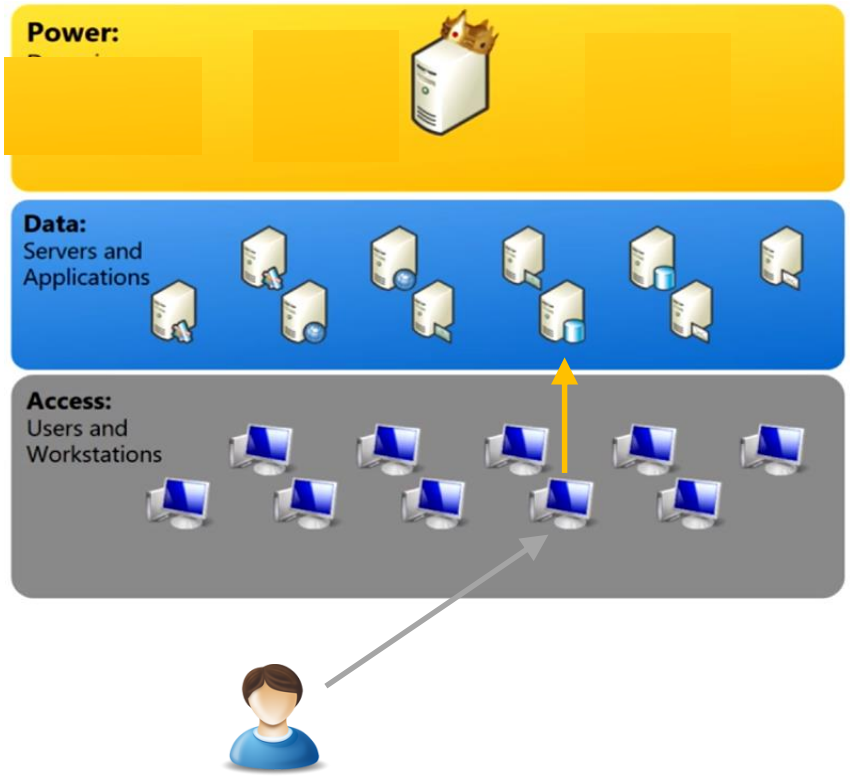


Interactive logon of domain account U

Workstation needs to:

- **Authenticate** U
- Make sure U has **access rights** for interactive logon on that workstation

Network Logon



Domain account U is in interactive logon **Network** logon on service S

S needs to:

- **Authenticate** U
- Make sure U has **access rights** for network logon on S

Network Logon: Just to have an idea (I)

- **Execute command** on remote_h with credentials user_x / pass_x

- **psexec** \\remote_h -u user_x -p pass_x cmd /c "YourCommandHere"
- Protocol SMB (port 445)

- **net use** \\remote_h /user:user_x pass_x
  psexec \\remote_h cmd /c "YourCommandHere"
- Protocol SMB (port 445)

Network Logon: Just to have an idea (II)

- Execute command on remote_h with credentials user_x / pass_x
- Powershell

```
$remoteComputer = "remote_h"
$credential = New-Object
-TypeName System.Management.Automation.PSCredential
-ArgumentList @("user_x", (ConvertTo-SecureString -String "pass_x"
                                                  -AsPlainText -Force))
Invoke-Command -ComputerName $remoteComputer
                        -Credential $credential
                        -ScriptBlock {
                                cmd /c "YourCommandHere"
                        }
```

- Protocol WinRM over HTTPS (port 5986)

Network Logon: Just to have an idea (III)

- **Mount network printer** with credentials user_x / pass_x
- **net use** LPT1: \\print_server\printer_name /user:user_x pass_x
- Protocol SMB (port 445)

- **Mount remote folder** with credentials user_x / pass_x
- **net use G:** \\remote_h\remote_f /user:user_x pass_x
- Protocol SMB (port 445)

- MANY different ways for doing the same thing...

Services

- Clients connect to Services by specifying the **service name**
  - Several ways of naming
  - UNC (previous examples)
  - SPN (Kerberos tickets)
  - Complex rules for determining which one to use (out of scope)
  - Service name specifies protocol, host, instance, port number

- Each Service is associated with an **account**
  - It may be a **User** account
  - Full control on the Service

To make a LONG story short

- Client-Server application protocols above TCP
- Microsoft-specified extensions for **each** such protocol:
  1. **Negotiate** authentication protocol
     - Service lists supported protocols:
       - NTLM
       - NTLM + Signing & Sealing
       - Kerberos
       - Kerberos + Signing & Sealing
     - Client chooses the strongest it supports
  2. **Authentication** protocol

- Details out of scope

Threat model: Network Attacker

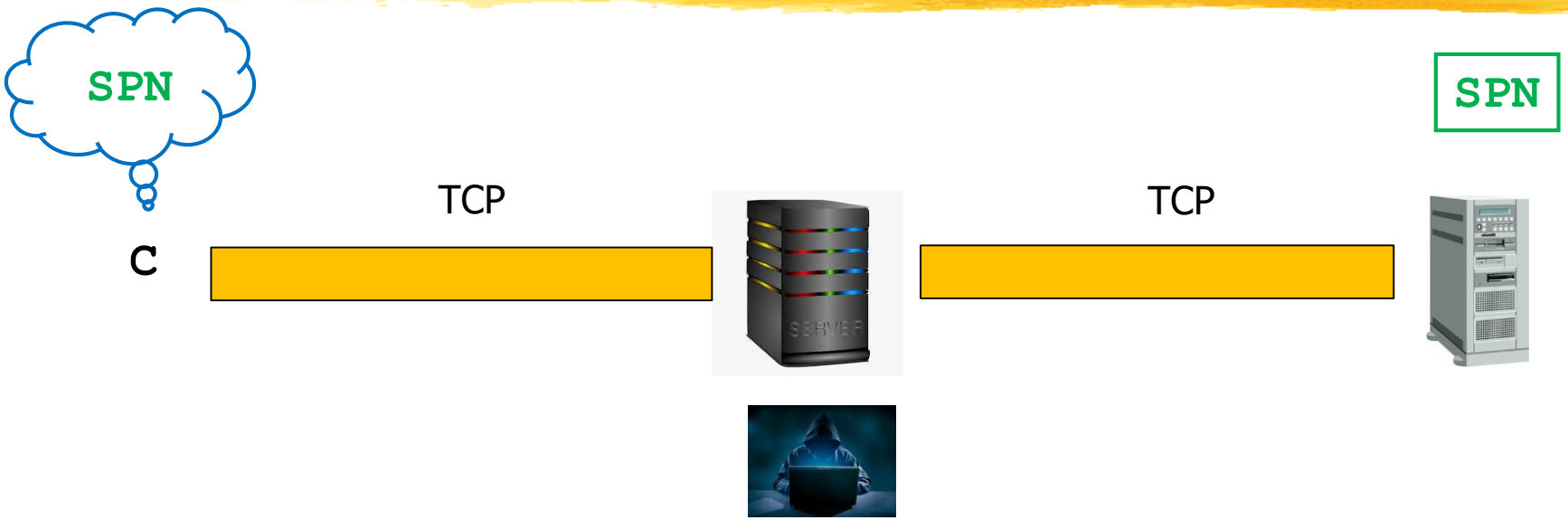
Threat model: Network Attacker

1. **Observe** & Communicate
   - Credential Access / Network Sniffing

2. **MITM**
   - Credential Access / AiTM (4 sub-techniques)
   - Credential Access / Forced Authentication
   - Some of them facilitated by Valid Account (Assume Breach: Attacker controls one machine)

   - Relatively simple to obtain in Windows environments
   - Key threat model today

MITM (AiTM)



MITM at the TCP level

Attack Objective

- Obtain **"guessing material"** from execution of authentication protocol
  - Useful only for guessing passwords of User / Service accounts
  - Passwords of Computer accounts are random 30-chars strings

Or

- **Impersonate** one of the two parties in execution of authentication protocol

Private Key Crypto: A few "practical" remarks

Remind

- MAC_K(msg)
  - **M**essage **A**uthentication **C**ode
    for message msg computed with private key K
  - **Fixed** Length
    - e.g., 128 byte: 2^(128*8) possible values
  - Provides Authentication + Integrity
  - **≈Signature** verifiable **only** by **one** entity

Common Terminology: HMAC

- **H**MAC(K,msg) or **H**MAC$_K$(msg)

- Common parlance (intuitive but inaccurate):
  - "Encrypt msg with K"
  - No secrecy, no decryption: not really an "encryption"

  - "Cryptographically secure hash"
  - When you compute a hash there is no key involved

Private Key Crypto: Practical Remark (I)

- **Secrecy** always comes with **Integrity**

- In every practical application:
  1. Modify an encrypted message or file
  2. Decryption fails

- Basic idea:
  - $E_K(msg)$ $HMAC(K, E_K(msg))$
  - Other combinations might leak information about `msg` or `K` (cryptoanalysis out of scope)

Corollary 1

- In practice, in a protocol based on **private key crypto** you **always** have:
  - Integrity
  - Mutual Authentication (the other end knows the private key)

`msg,HMAC(K,msg)`

or

  - Secrecy
  - Integrity
  - Mutual Authentication (the other end knows the private key)

E_K(msg) HMAC(K, E_K(msg))

Windows Terminology

- **Signing**
  - Integrity
  - Mutual Authentication (the other end knows the private key)
- Signing & **Sealing**
  - Secrecy
  - Integrity
  - Mutual Authentication (the other end knows the private key)
- Kerberos: Safe and Private

Our Notation

- When describing messages with **secrecy**
  we will **not** write the HMAC explicitly (but it is there!)

- Secrecy
- Integrity
- Mutual Authentication (the other end knows the private key)

$E_K(msg)\,HMAC(K, E_K(msg))$

$E_{K\_US}(T_{U1})\,E_{K\_US}(resp1)$

means

$E_{K\_US}(T_{U1})\,HMAC(K\_US, E_{K\_US}(T_{U1}))$

$E_{K\_US}(resp1)\,HMAC(K\_US, E_{K\_US}(resp1))$

Crypto "Curiosity"

- In practice, keys for HMAC and encryption must be **different**
  - Cryptoanalysis out of scope
- Every protocol derives them from a common "master key"
- We will **never** discuss this detail

- $E_{K1}(msg)\,HMAC(K2, E_{K1}(msg))$

Private Key Crypto: Practical Remark (II)

- When **decrypting**, you can always tell **immediately** whether you used the "correct key" (i.e., the one used for encrypting)

- In every practical application:
  1. Encrypt a message or file with k
  2. Decryption with k1 <> k fails

- Basic idea:
  - $E_K(msg)$ $HMAC(K, E_K(msg))$
  - Other combinations might leak information about `msg` or `K` (cryptoanalysis out of scope)

Corollary 2

- In practice, when trying to decrypt a message/file encrypted with a private key,
  you do **not** need **any info about the cleartext**

- Offline guessing of AuthDB
- Offline guessing of message $E_K(msg)$ $HMAC(K, E_K(msg))$
- ...

Prudent Engineering Practice (I)

- **"Every key must be changed every now and then"**
- Fundamental principle for **every** practical application

- Basic reason: "you never know"
  - **IF** a key was obtained "somehow" THEN **all** traffic would be decrypted
- Key change:
  - Less time for (trying to) obtain the key
  - Less traffic will be decrypted

"You never know"

- **IF** a key was obtained "somehow" THEN **all** traffic would be decrypted
- Key might be **stolen from memory**
- Key might be weak (derived from **password**) and **offline guessing** on captured ciphertext succeeds
- Attacker might have much more powerful guessing hw than believed
- Crypto alg might have vulnerabilities
  → Size of key space much smaller than believed

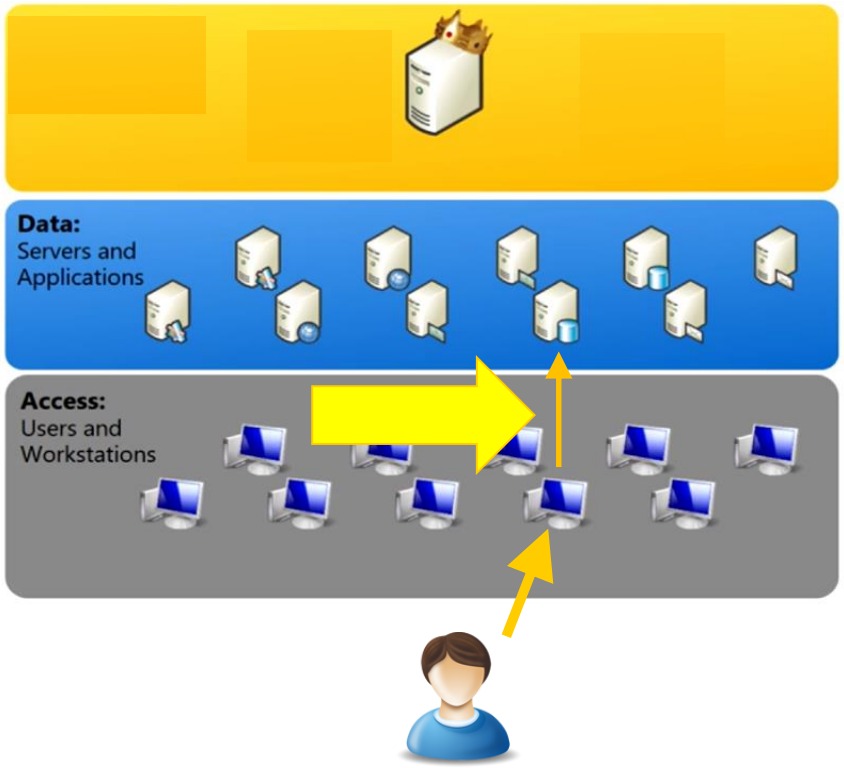
Prudent Engineering Practice (II)

- "Every key must be changed every now and then"
- **"Weak keys should generate as little traffic as possible"**
- Fundamental principle for **every** practical application of crypto

- A key derived from a password should be used **only** for **deriving** other keys
  - Less opportunities for capturing traffic of a weak key
  - Less traffic for cryptonanalysis of a weak key (issue only if crypto alg is vulnerable)

NTLM

Our focus: Network Logon



Interactive logon with NTLM intricate and **not relevant**

- **C authenticates with S**
- At the end, S asks DC for authentication and authorization
- Secure channel S↔DC between computer accounts at bootstrap
- NTLM+Signing&Sealing (for mutual auth, see later)

omitted

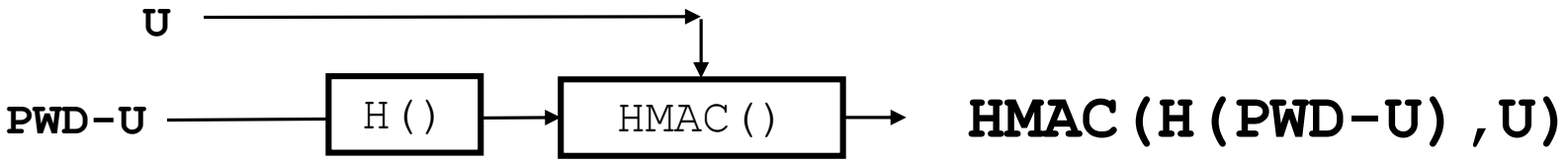
Hash Terminology

- **NTLM hash**:
  - Password hash
  - **Nothing** to do with NTLM **protocol**
    (nothing to do with the protocol)
  - Allows full impersonation of account

- The NTLM protocol uses:
  - NTLMv2 hash ≈"Username encrypted with password"
  - **NetNTLM hash** ≈"Random number encrypted with NTLMv2 hash"

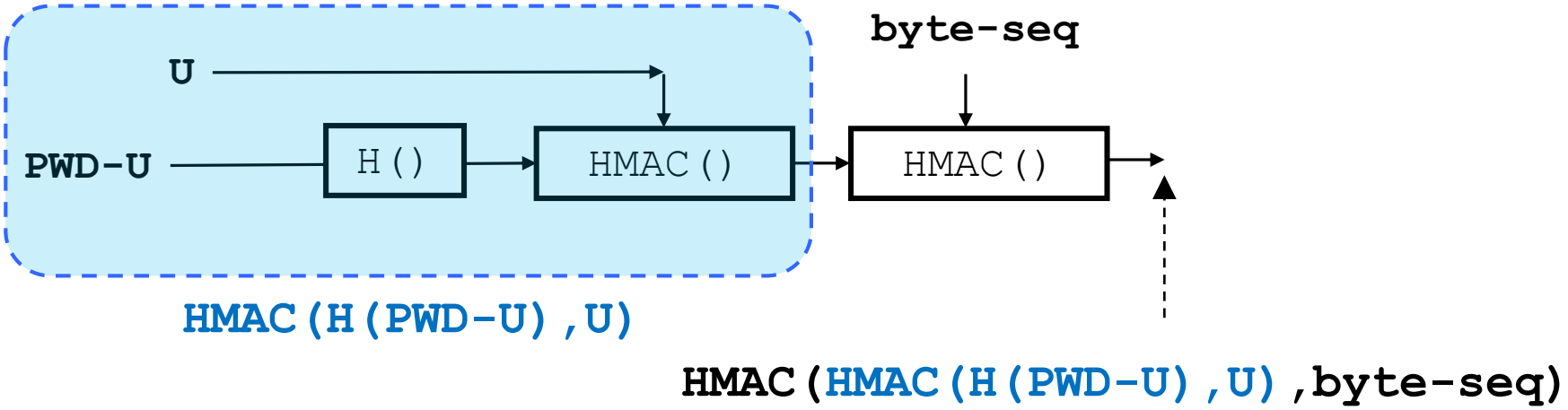
  - An Observe Attacker can observe **NetNTLM hash**
    (not NTLMv2 hash)

NTLMv2 Hash (intermediate, not important)



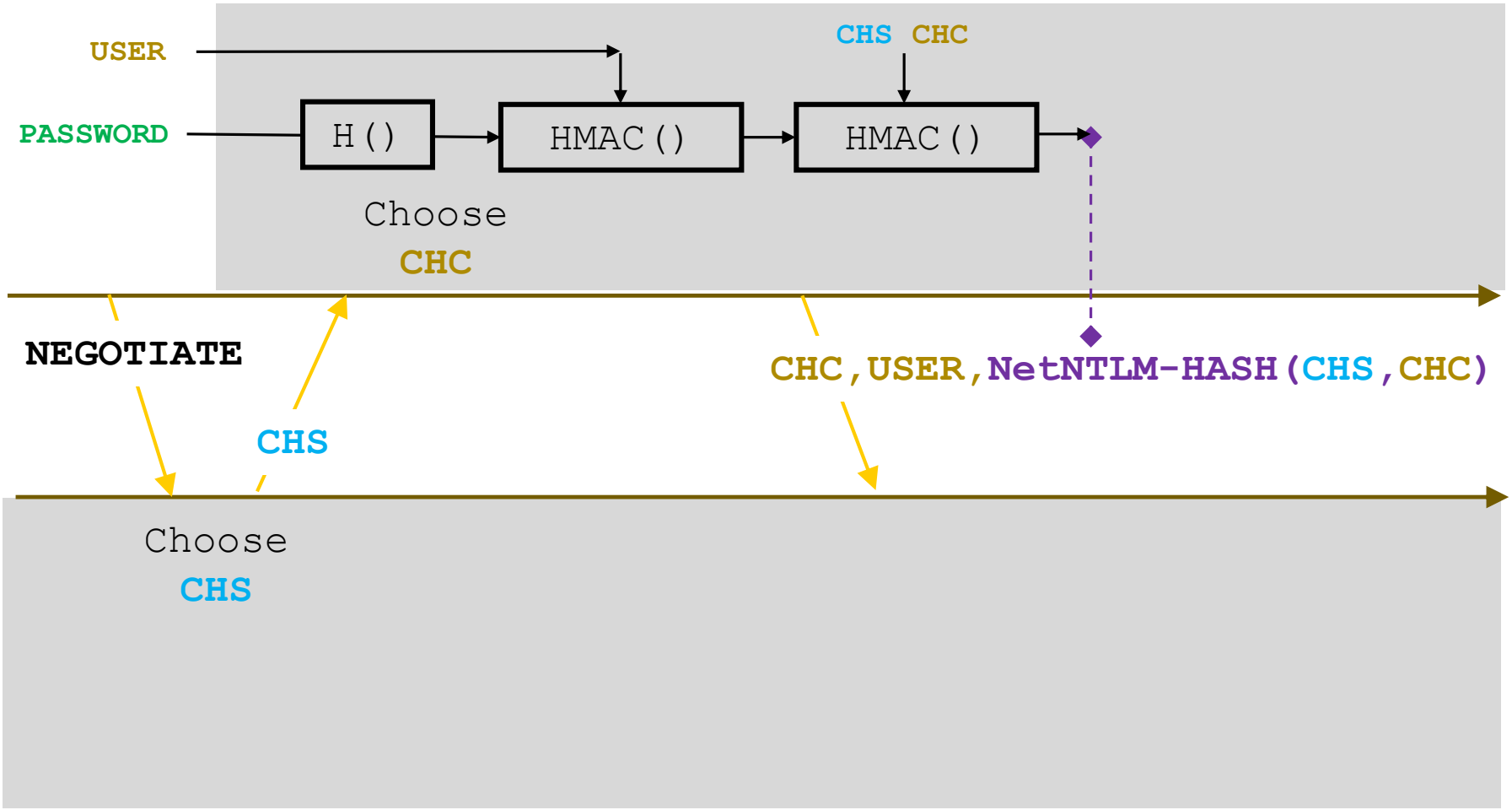
- Intuition (inaccurate): "Encrypt U with H(PWD-U)"
  - More complex in reality...does not matter to us

NetNTLM Hash (Important)

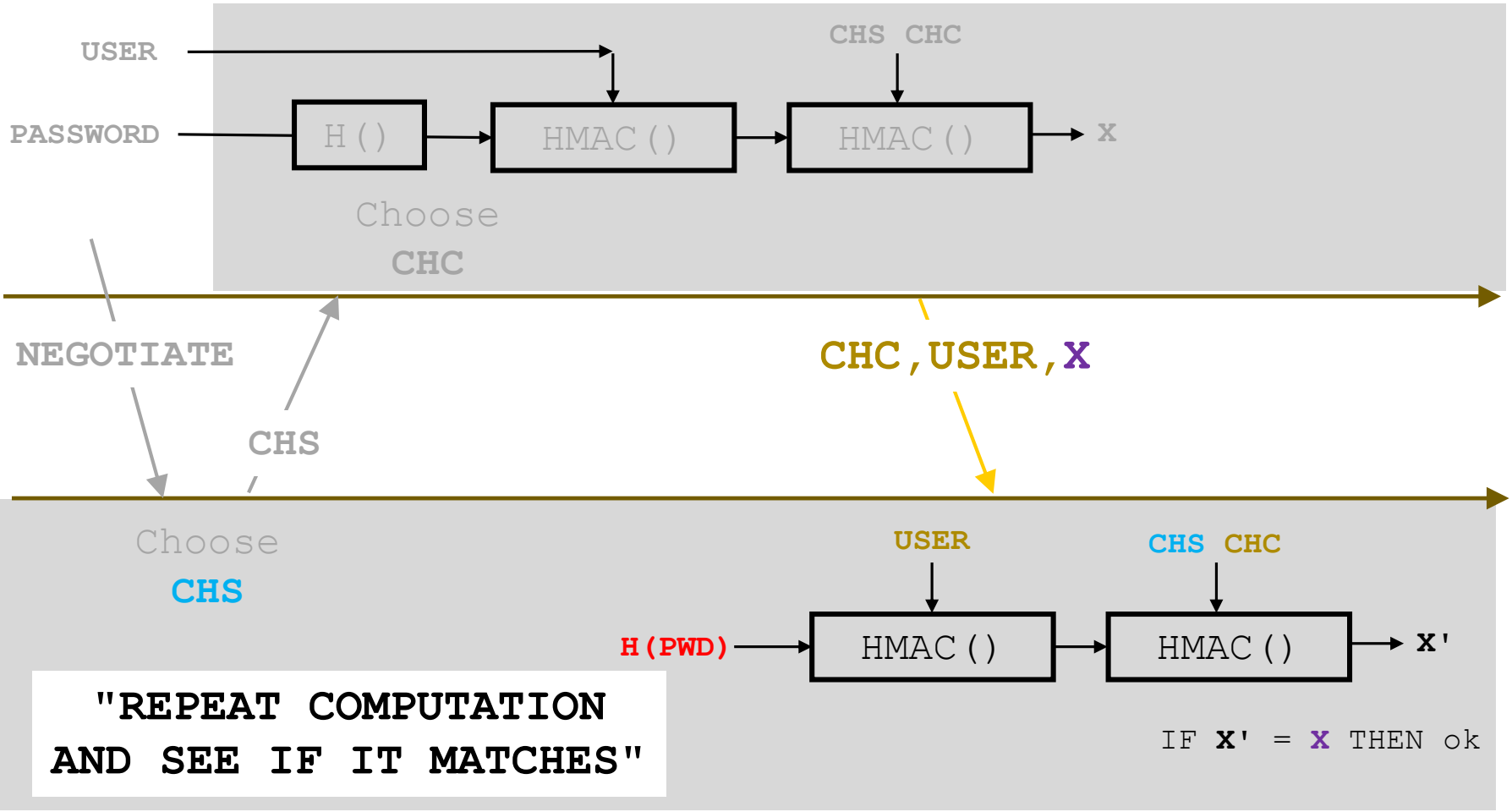


- Intuition (inaccurate): "Encrypt byte-seq with NTLMv2 hash"
  - More complex in reality...does not matter to us

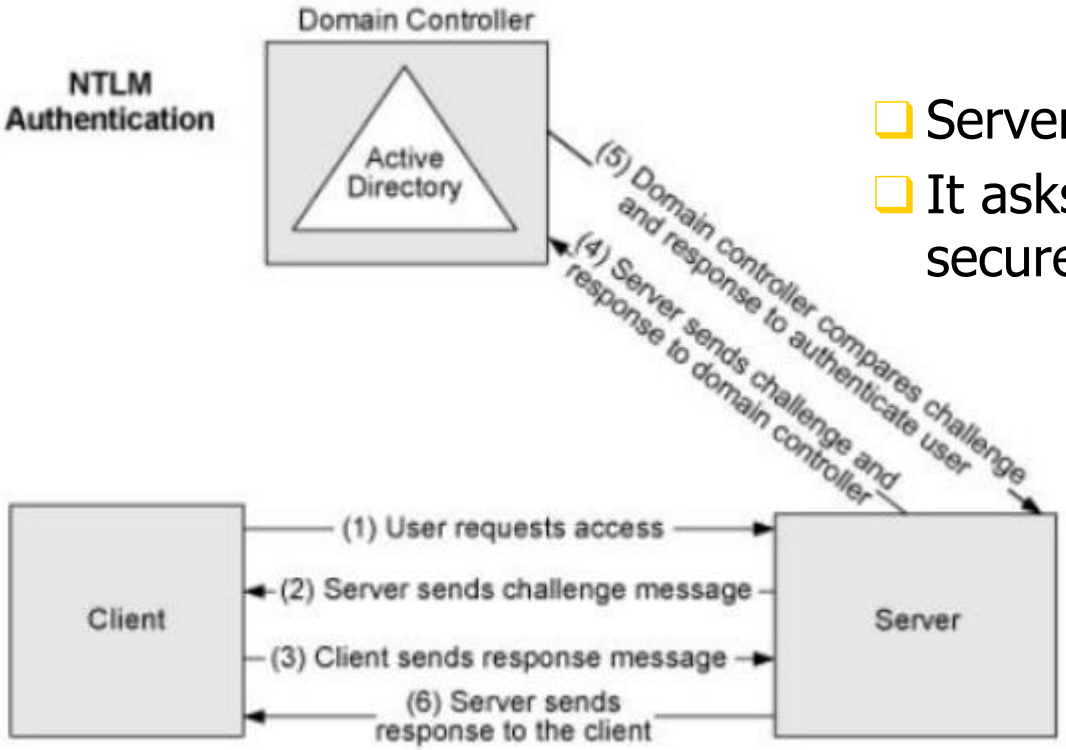
NTLM Execution (I)



NTLM Execution (II)

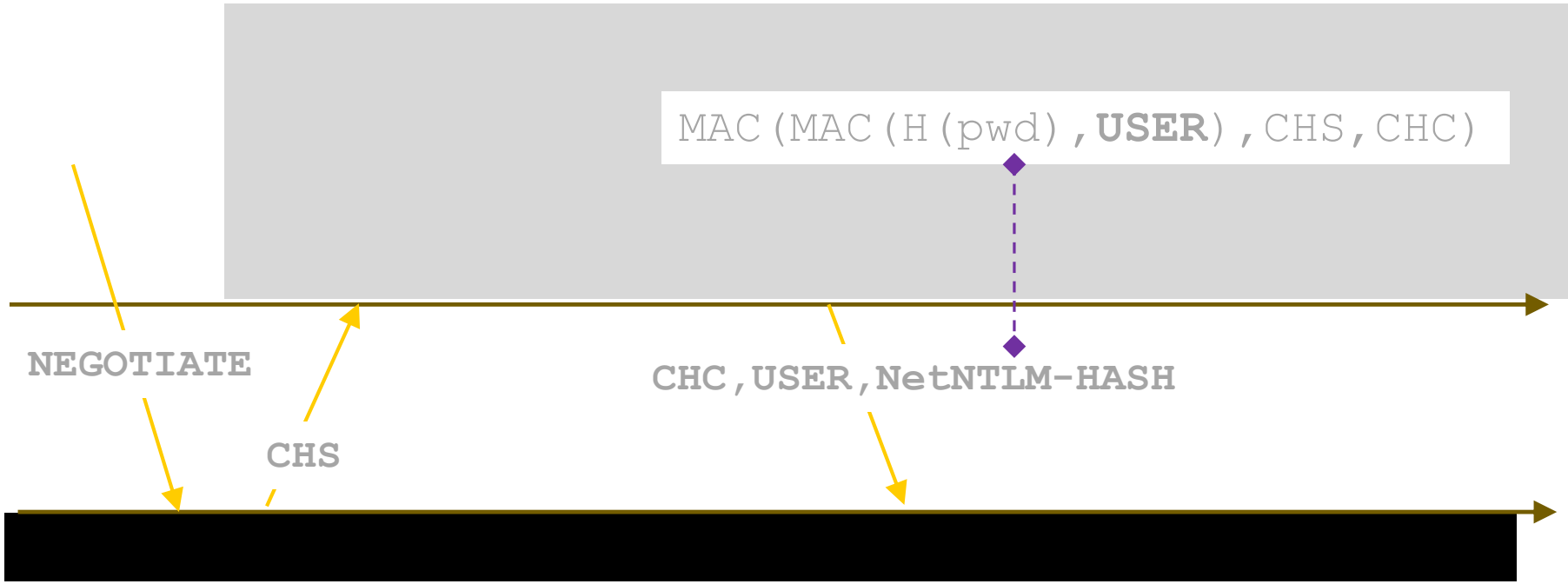


Remind



- Server does not have AuthDB
- It asks domain controller over secure channel

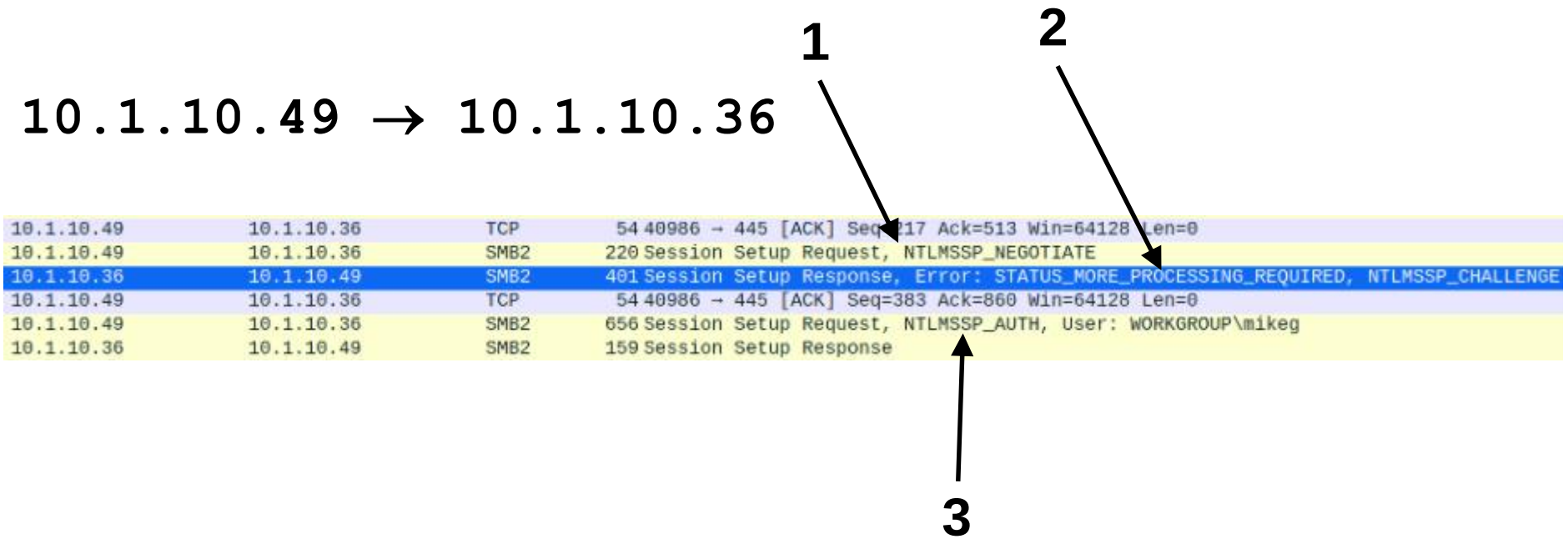
Remark: Server NOT authenticated



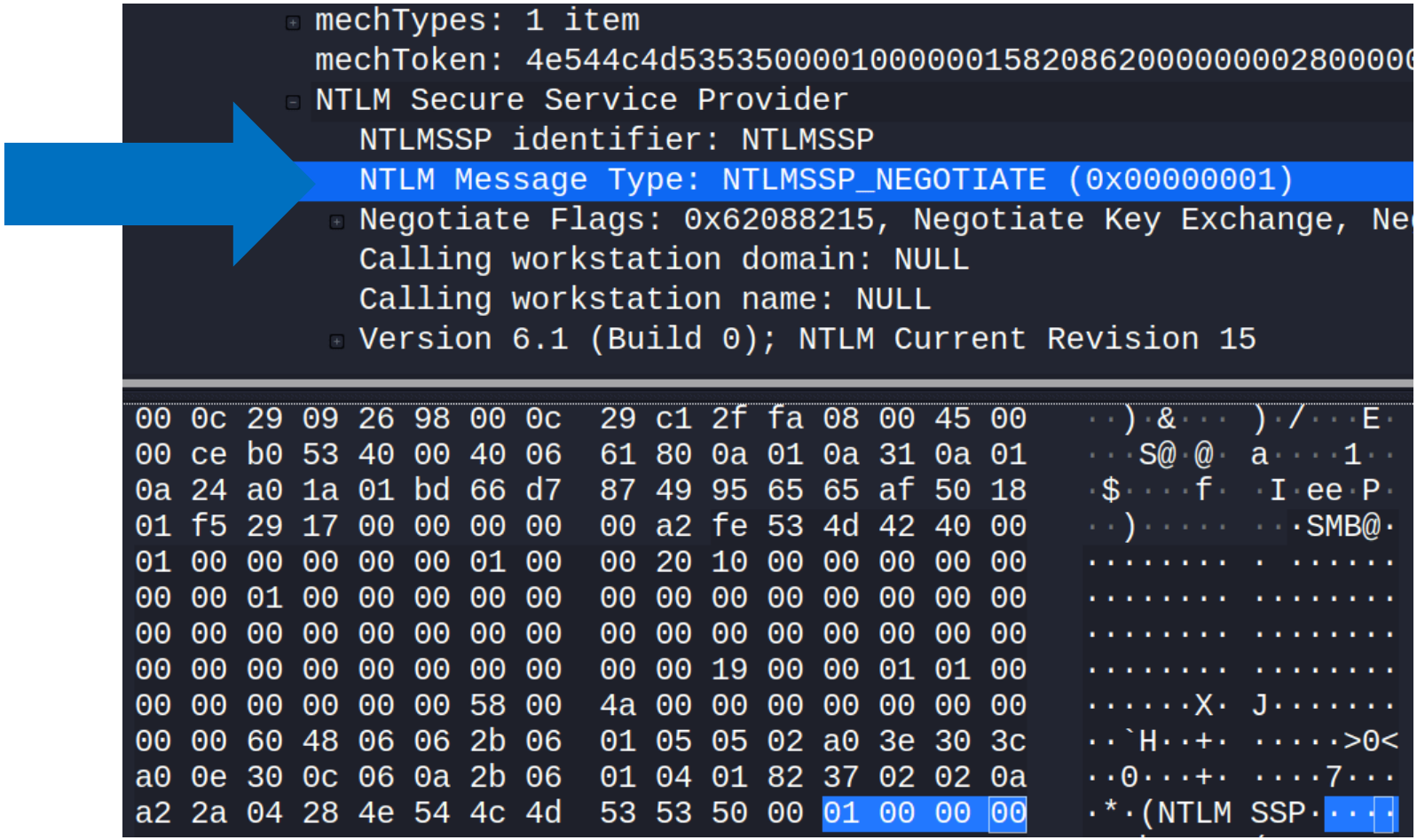
- Server **does not prove** its knowledge of `H(PWD)`
- **Client** authentication (**unilateral**)

Network Traffic Example (SMB2)

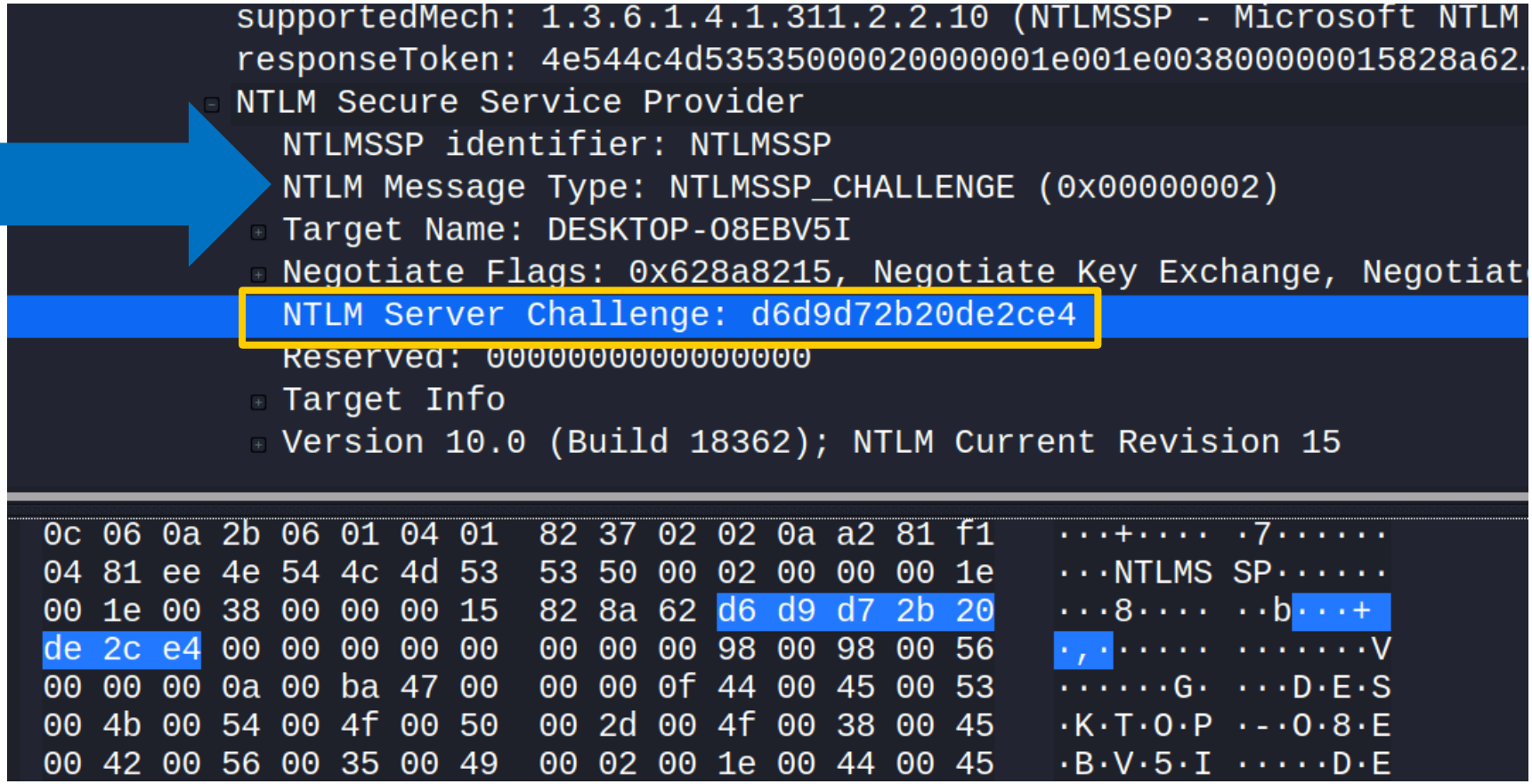
10.1.10.49 → 10.1.10.36



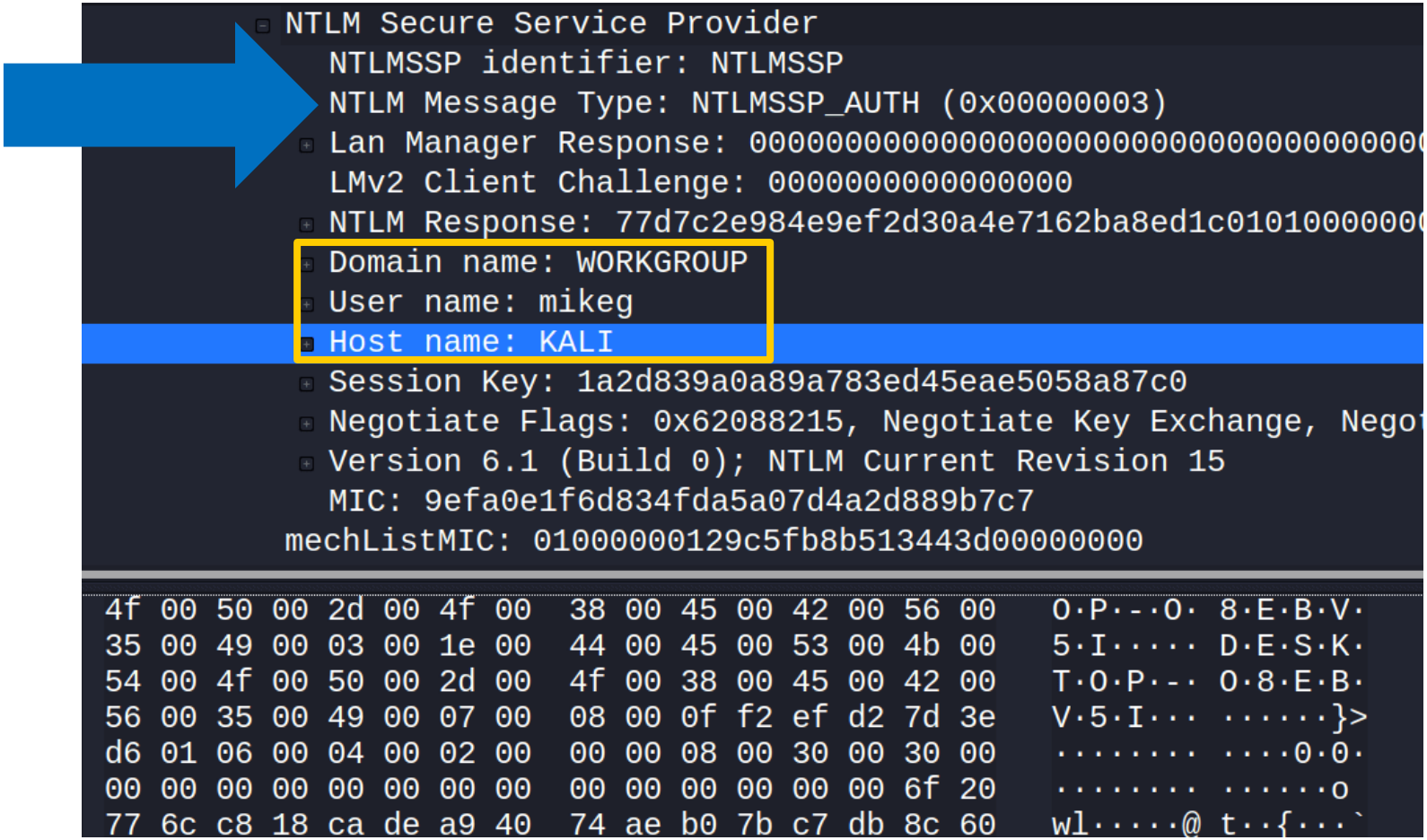
Network Traffic Example: Msg 1



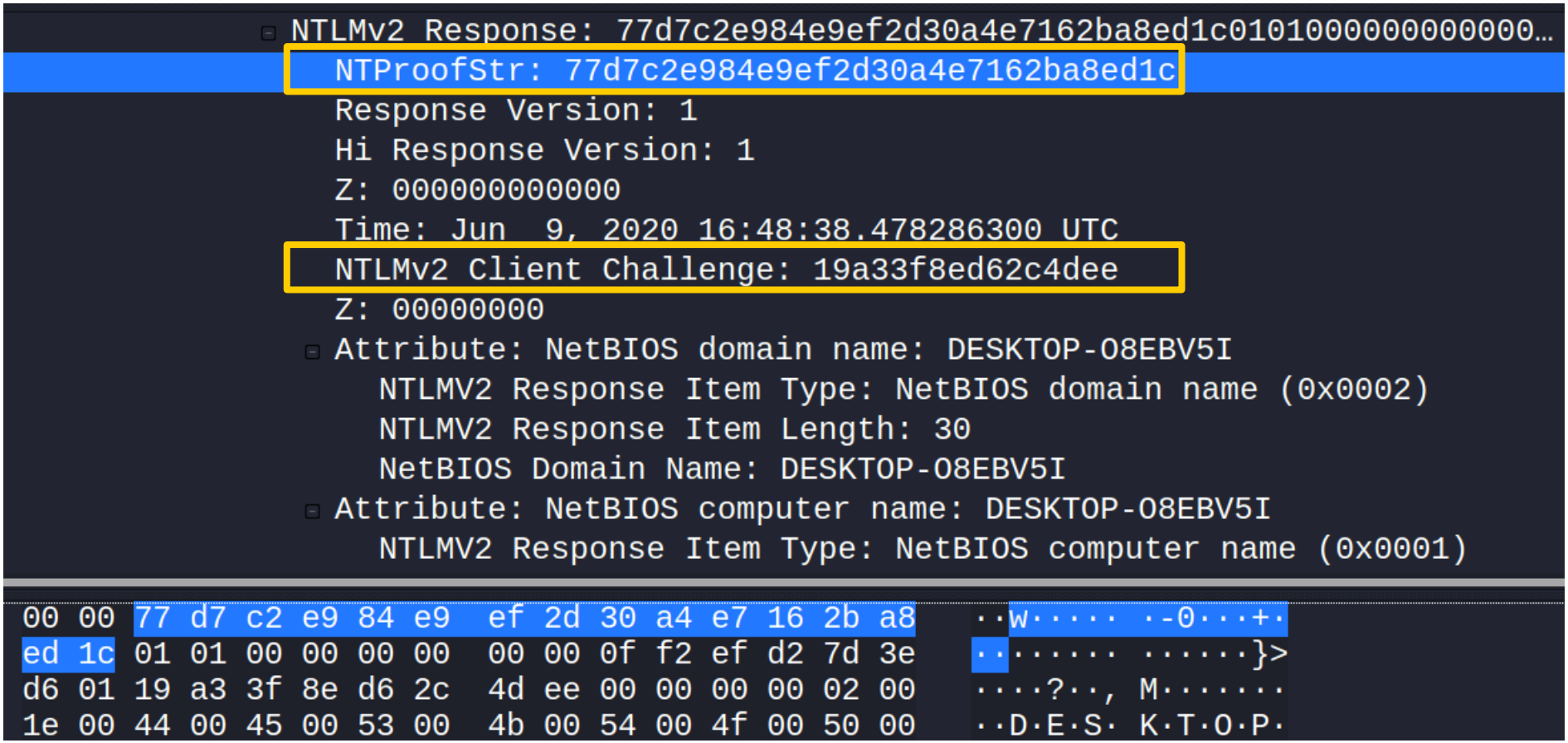
Network Traffic Example: Msg 2



Network Traffic Example: Msg 3 (I)



Network Traffic Example: Msg 3 (II)



NTLM Attacks

NTLM Attacks: Pass the Hash (PtH)

Important question

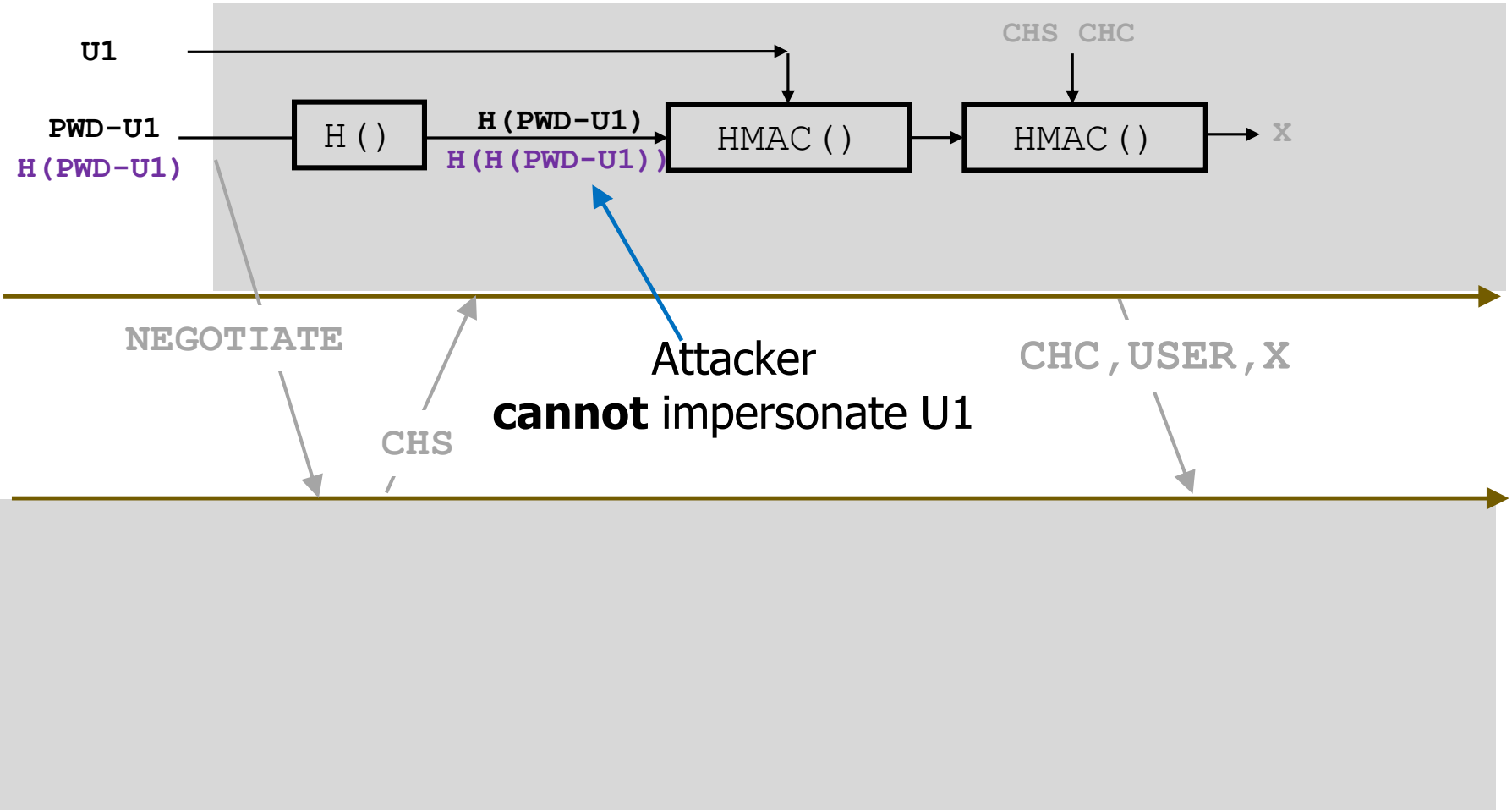
- Attacker knows `H(PWD-U1)`
- Attacker does **not** know `PWD-U1`

- Can Attacker impersonate `U1` with NTLM?

Important Remark

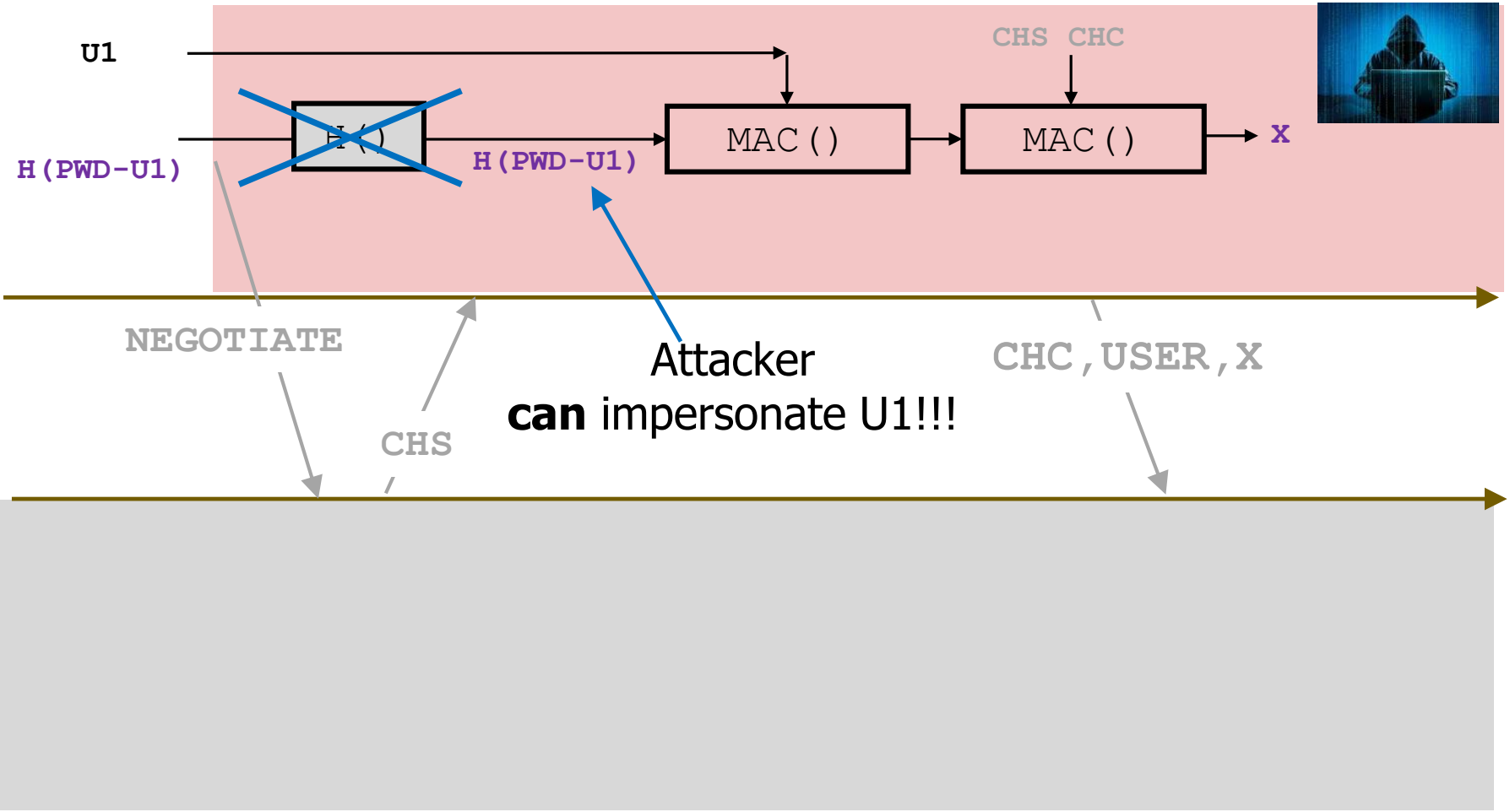
- Knowledge of **H(PWD) suffices** to impersonate the corresponding user with NTLM
- Offline guessing on network authentication targets PWD (**not** H(PWD))
  - Credential Access / Network Sniffing, Password Cracking
- H(PWD) can be stolen with a **different** threat model
  - Credential Access / O.S. Credential Dumping

"Legitimate" NTLM Client Software



pth-winexe

(and many others...)



Windows INTRINSIC feature

- Knowledge of **H(PWD) suffices** to impersonate the corresponding user with NTLM

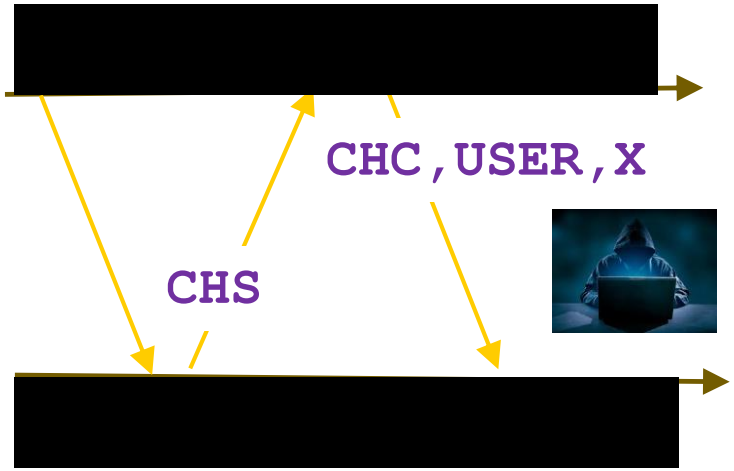
- Stealing of SAM / NTDS is a **catastrophe**

Technical "Curiosity"

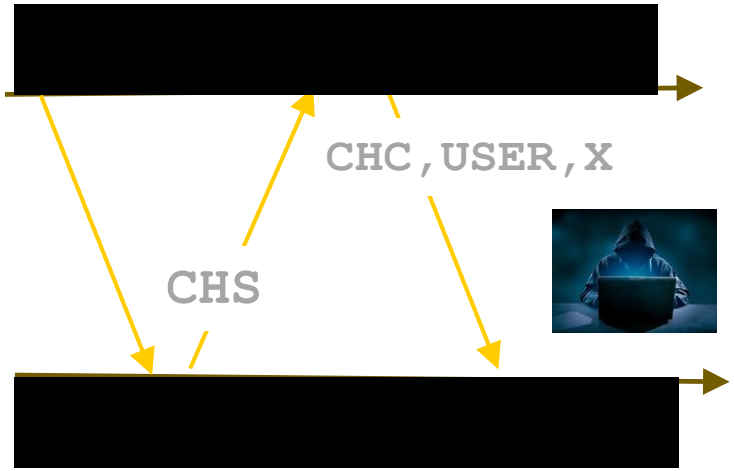
- Knowledge of **H(PWD) suffices** to impersonate the corresponding user with NTLM
- Tricky **limitations** regarding which operations a certain user can execute from a remote location
  - But H(PWD) does authenticate the remote user
- No limitations for local operations

NTLM Attacks: Network Sniffing

Threat Model: Observe

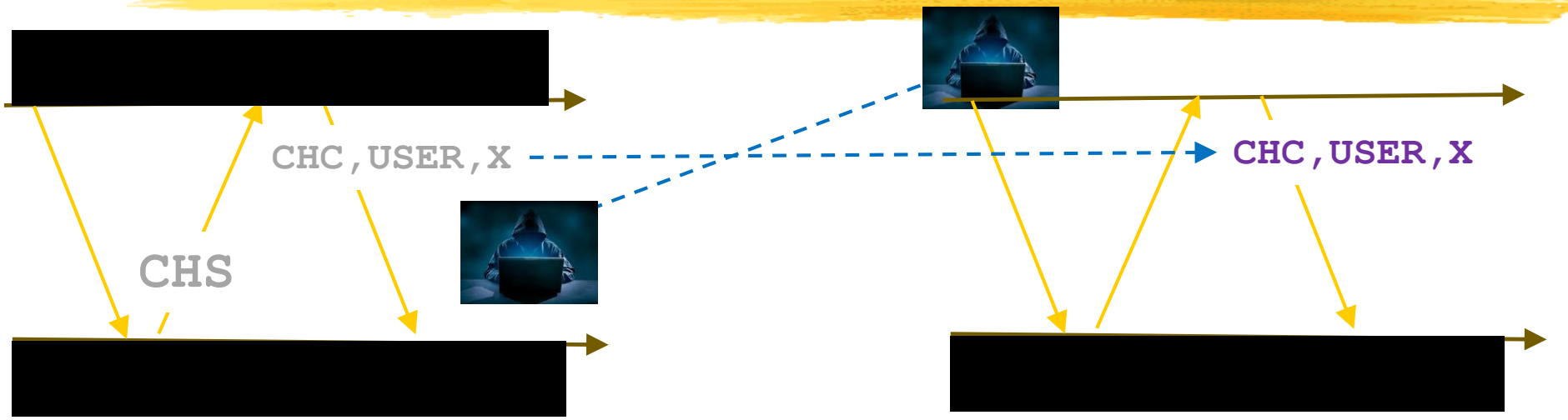


Impersonate Server?



- **Impersonate** one of the two parties in execution of authentication protocol
- Attacker is not in the middle:
  - It **cannot** impersonate Server

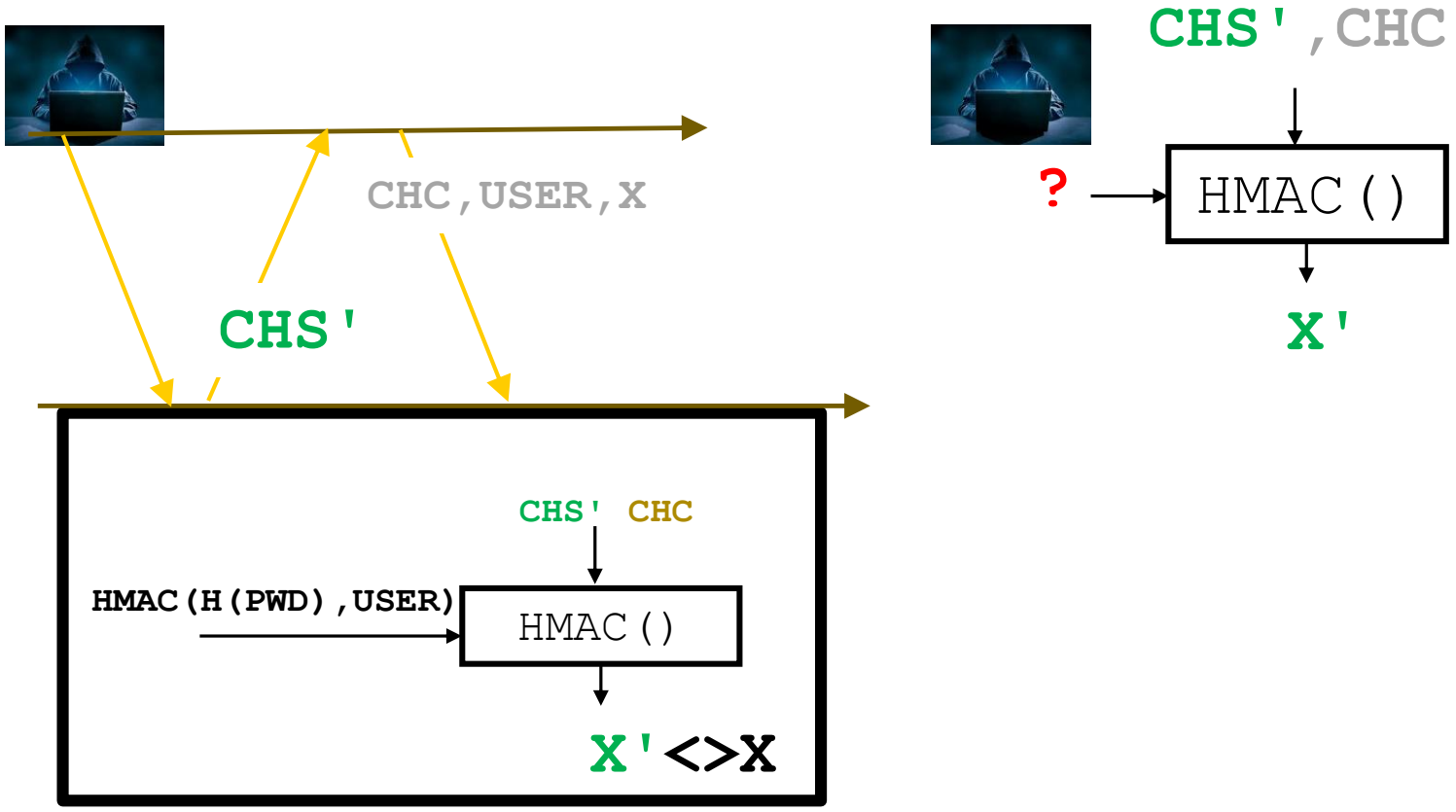
Impersonate Client? (I)



- It can try to impersonate Client by **replaying traffic**

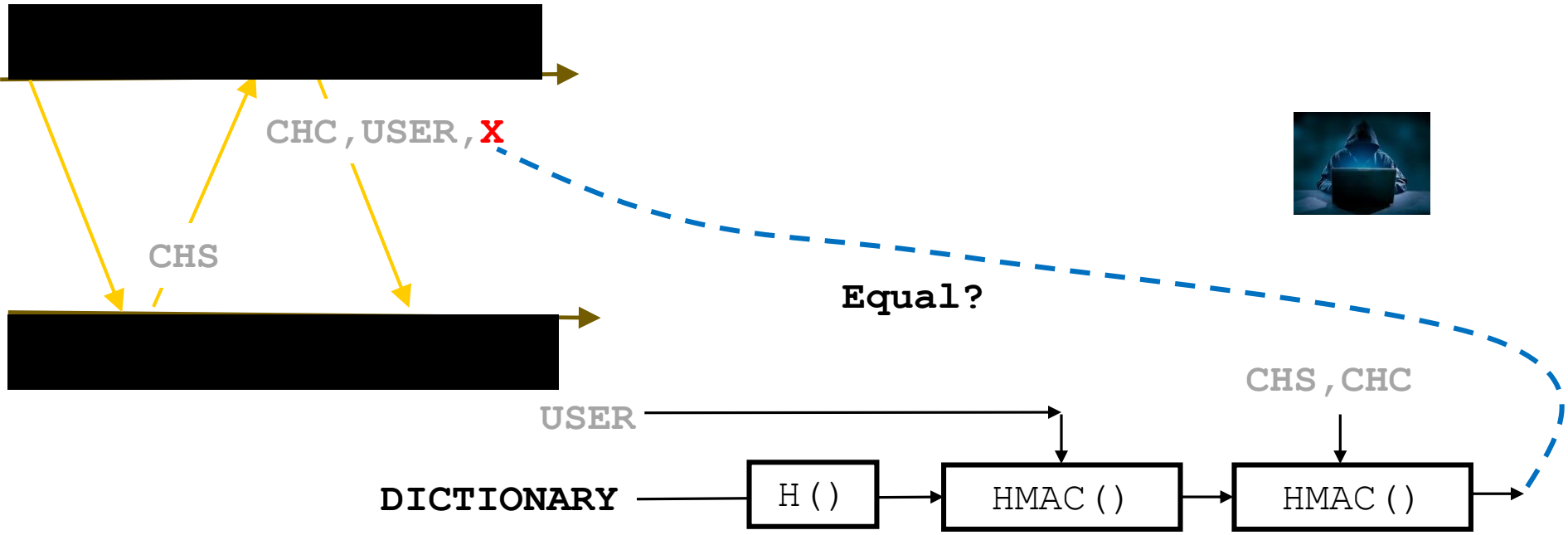
Impersonate Client? (II)

- Replay does **not** work: CHS changes at **every** execution



All that can be done

- Obtain **"guessing material"** from execution of authentication protocol
- Credential Access / Network Sniffing, Password Cracking



Offline Guessing on Network Authentication

- **Conceptually equivalent**:
  - Offline guessing of AuthDB Hashed**+Salted**
  - Offline guessing of Authentication Protocol execution
- Hashes **cannot** be computed in advance

```
foreach p ∈ candidate-password-set
    x := H(concat(p,user.salt))
    IF x == user.H THEN p is ok
```

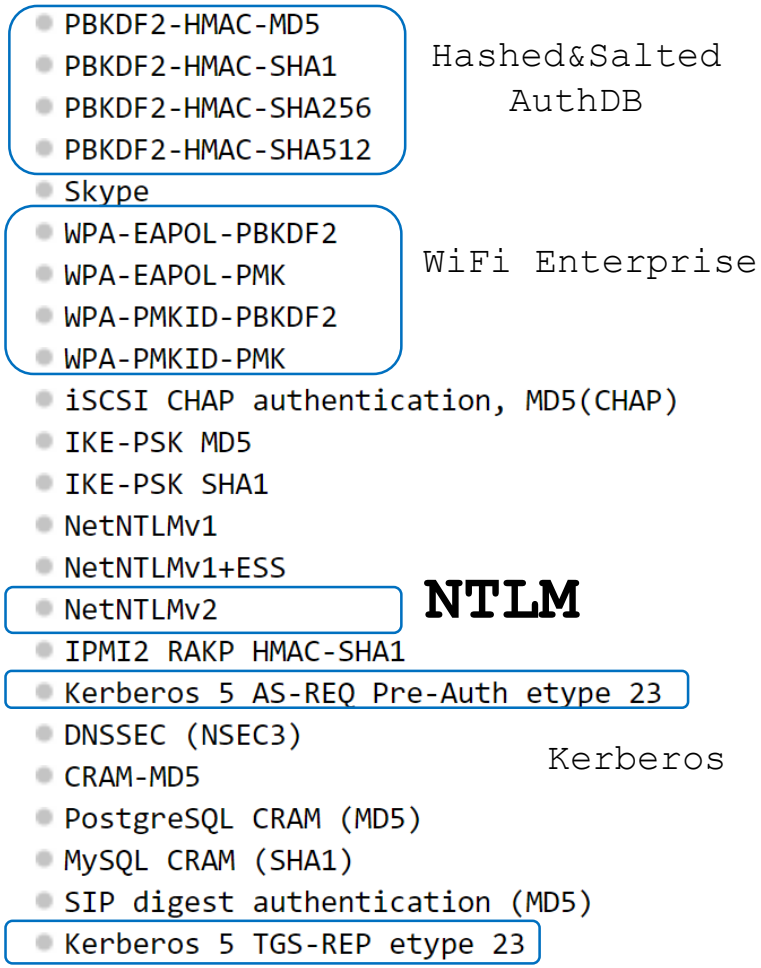
AuthDB Hashed+Salted

```
foreach p ∈ candidate-password-set
    x := HMAC(HMAC(H(p),USER),chc chs)
    IF x == NetNTLM-HASH THEN p is ok
```

Auth Protocol Execution NTLM

REMIND

- Modern cracking tools handle most
  - AuthDB formats
  - Authentication protocols



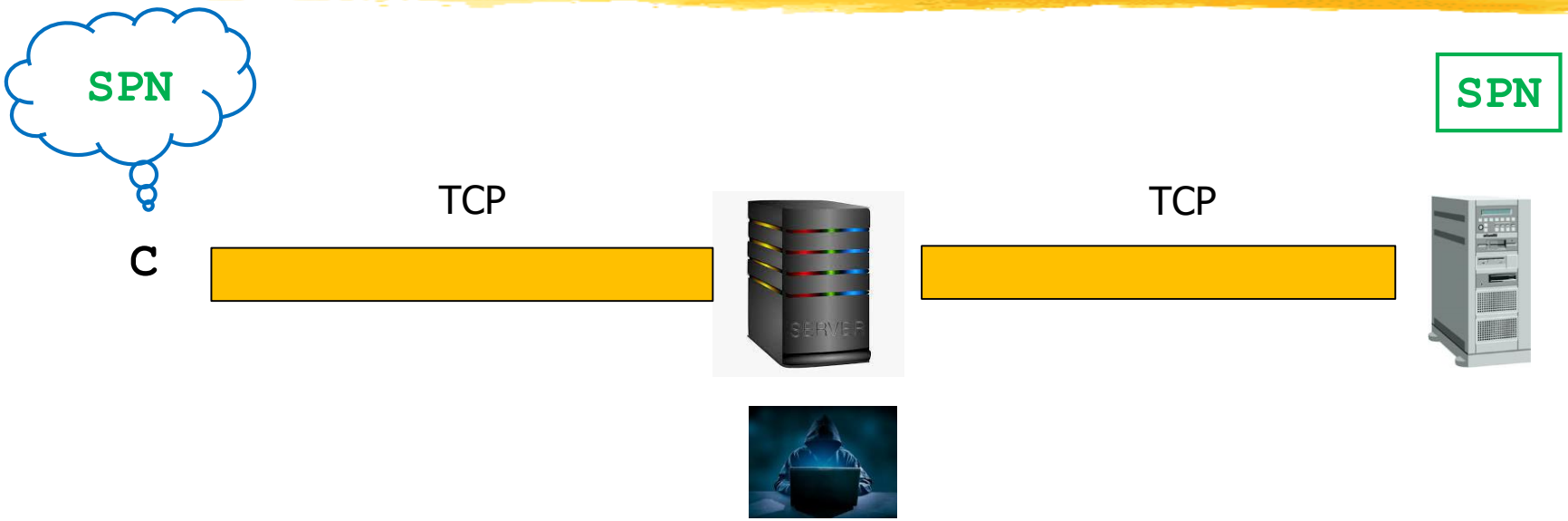
Remark

	#hash/sec		Relative	
NTLM	715.6	G	1.4	M
NetNTLMv2	27795	M	53.9	K
Kerberos 5, etype 23, TGS-REP	6826.3	M	13.2	K
WPA-EAPOL-PBKDF2 (Iterations: 4095)	6120.2	K	11.9	
bcrypt $2*$, Blowfish (Unix) (Iterations: 32)	515.7	K	1.0	

- **Five orders of magnitude** more guesses than bcrypt

NTLM Attacks: Collect guessing material

Threat Model: MITM



MITM at the TCP level

Relatively simple to obtain this capability in Windows networks
(5 MITRE ATT&CK Techniques)

Remind: HTTPS

HTTPS

C S

server cert

Server Authentication

$E_{KPUB-S}(K)$

$E_K(req)\,MAC_K(req)$

Secrecy
Integrity
Server Authentication

$E_K(resp)\,MAC_K(resp)$

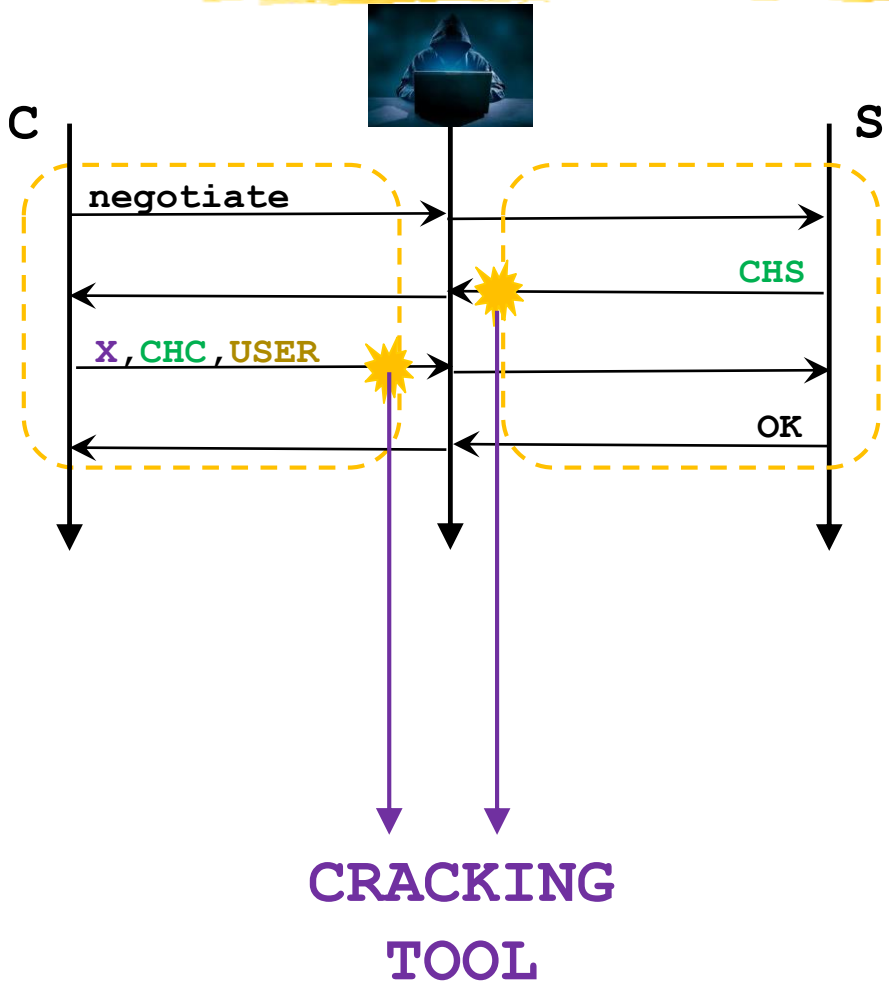
Attacker can **only** **block** communication

(Attacker cannot know K)

NTLM

Nothing specifies desired SPN

(client-only auth.)



Bad news to keep in mind (I)

Fact:

- Relatively simple to be MITM in Windows networks
- ...thus capturing one NTLM execution is **relatively simple**

Consequence:

- Offline guessing attacks in Windows environments must be considered **unavoidable**
- ...and keep in mind those attacks can make **a lot of guesses**

Bad news to keep in mind (II)

Fact:

- Most organizations have **Single Sign On** (SSO): **One** password for **all** the services of the organization
- Guessing attacks with **many** guesses are **unavoidable**

Consequence:

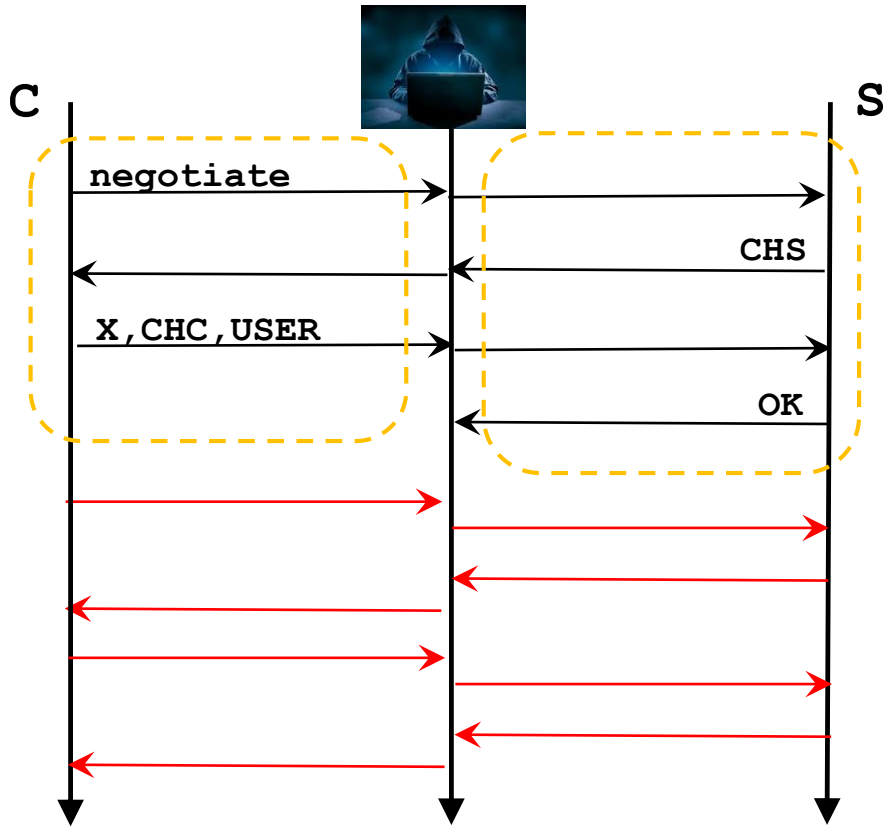
- Windows passwords have a **huge value** for Attackers (and carry a **huge risk** for Defenders)
- Having a **strong** Windows password is **extremely important** in large organizations

NTLM Attacks: Impersonate Server

Impersonate Server: Easy

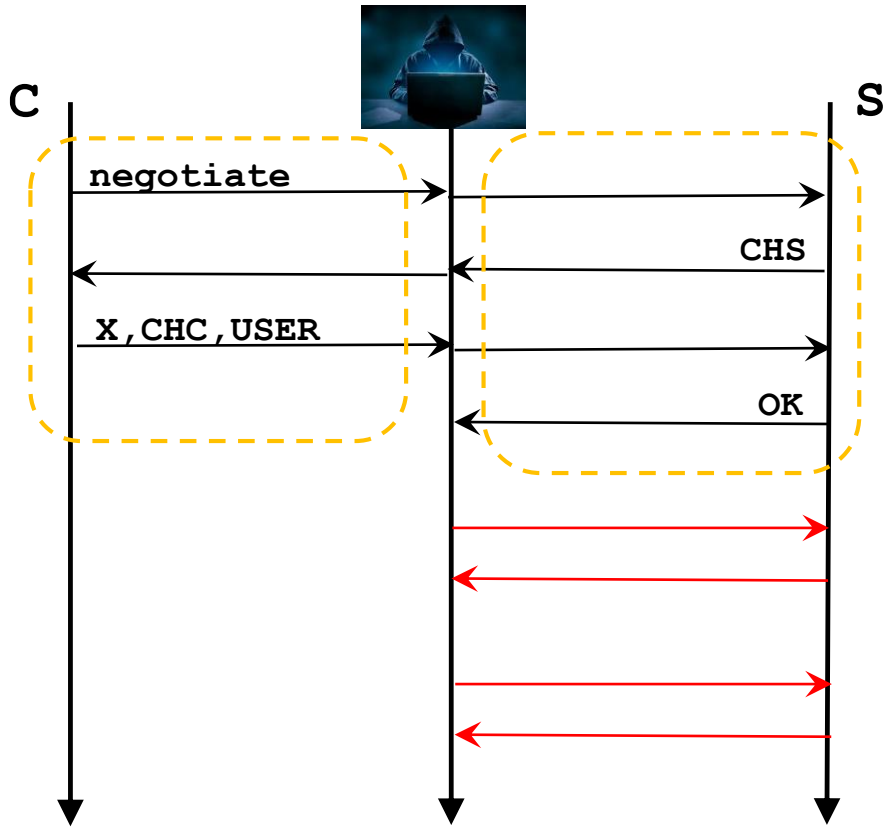
- Modify traffic at will
- No need to guess password!

Nothing specifies desired SPN



Impersonate Client: Easy

- Execute any command at will
- No need to guess password!



MITM against NTLM

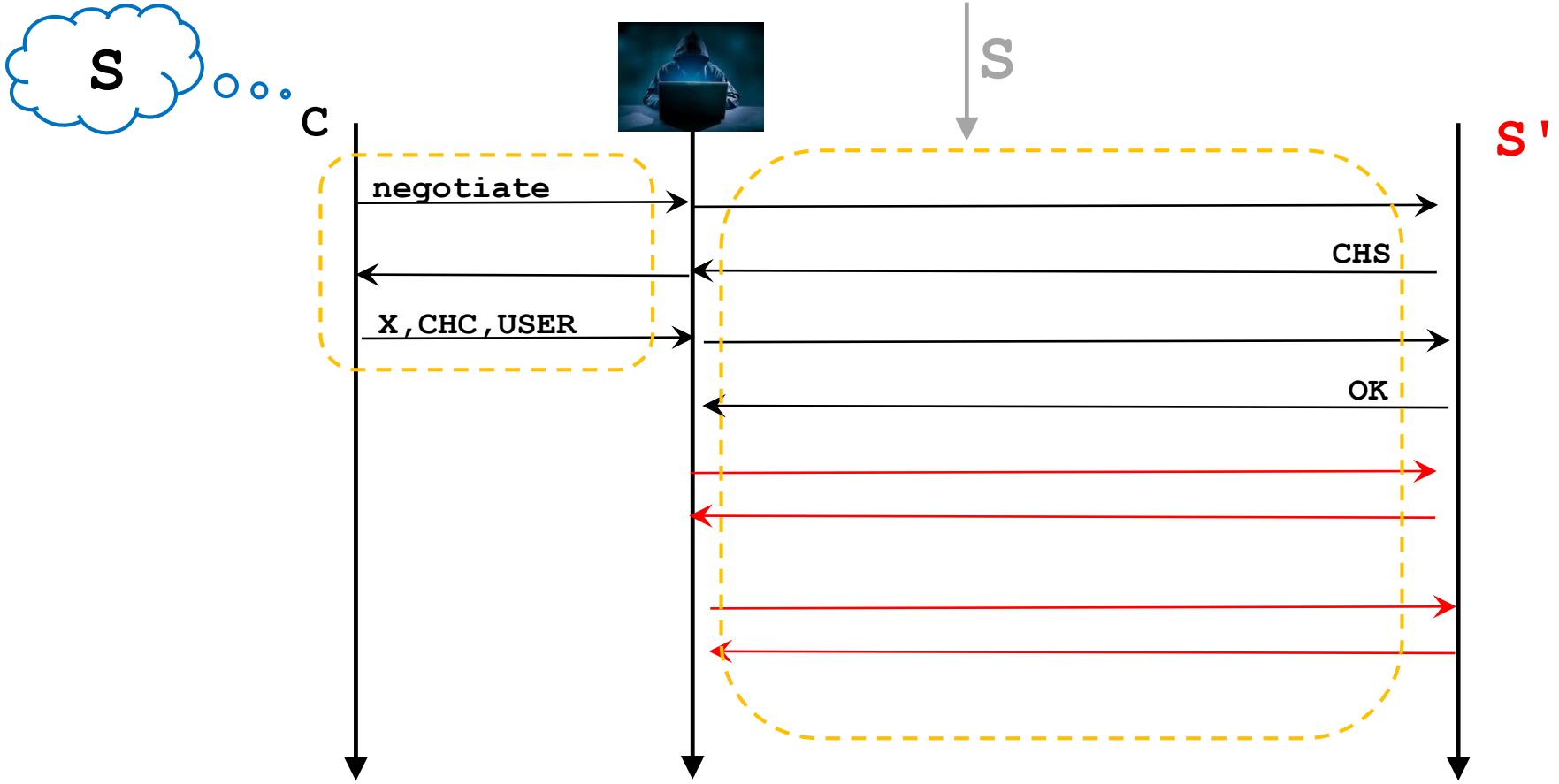
- USER@C contacts S with NTLM
- MITM Attacker

- Attacker collects guessing material
- Attacker executes **arbitrary** commands on S with identity USER

- BIG PROBLEM

NTLM Relay Attack

Impersonate Client: Easy **even on a DIFFERENT S'!**



NTLM Relay Attack

- USER@C contacts S with NTLM
- MITM Attacker

- Attacker collects guessing material
- Attacker executes **arbitrary** command on S with identity USER
- Attacker executes **arbitrary** commands on **arbitrary** S' with identity USER

- HUGE PROBLEM

Think a moment about the implications…

- USER@C contacts a printer (or whatever) with NTLM
- MITM executes **arbitrary** command on **arbitrary** S' with identity USER

- USER belongs to administrator group
- S' = domain controller

⇒ Game over

Bad news to keep in mind (III)

Fact:

- Relatively simple to be MITM in Windows networks

Consequence:

- Guessing attacks and NTLM Relay attacks in Windows environments must be considered **unavoidable**

NTLM Relay in practice

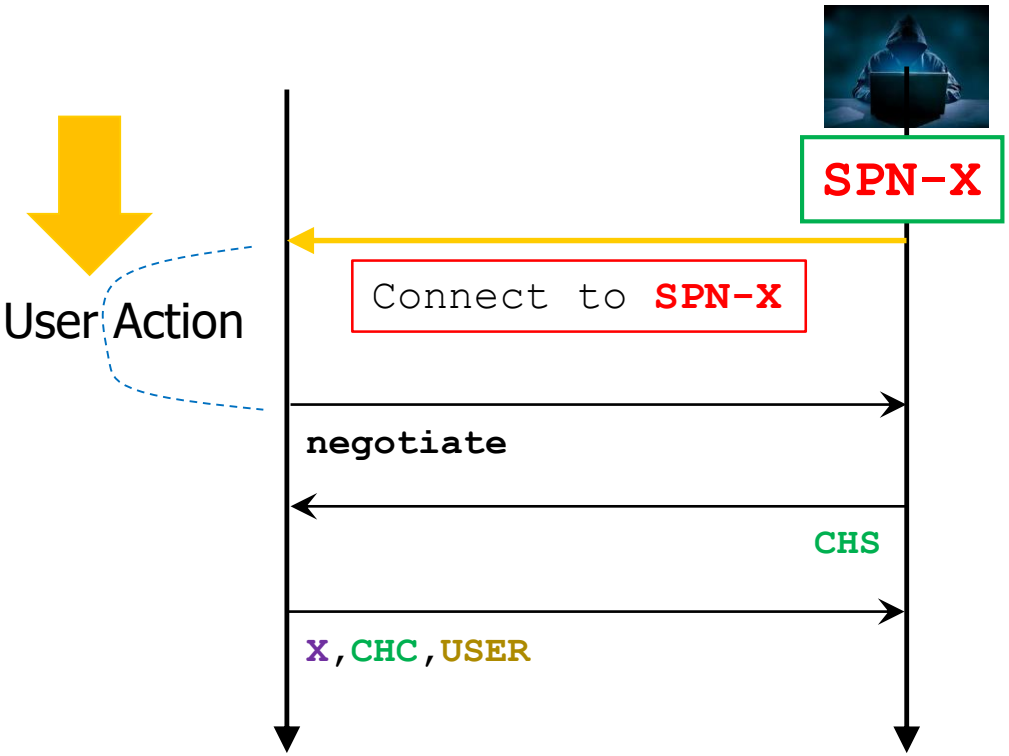
What Attacker does NOT control

- Attacker has little or no control on:
  - Which C
  - Which USER
  - Which SPN
  - **When** USER@C decides to connect to SPN (if at all)

- "Wait and hope something good happens"

Forced Authentication

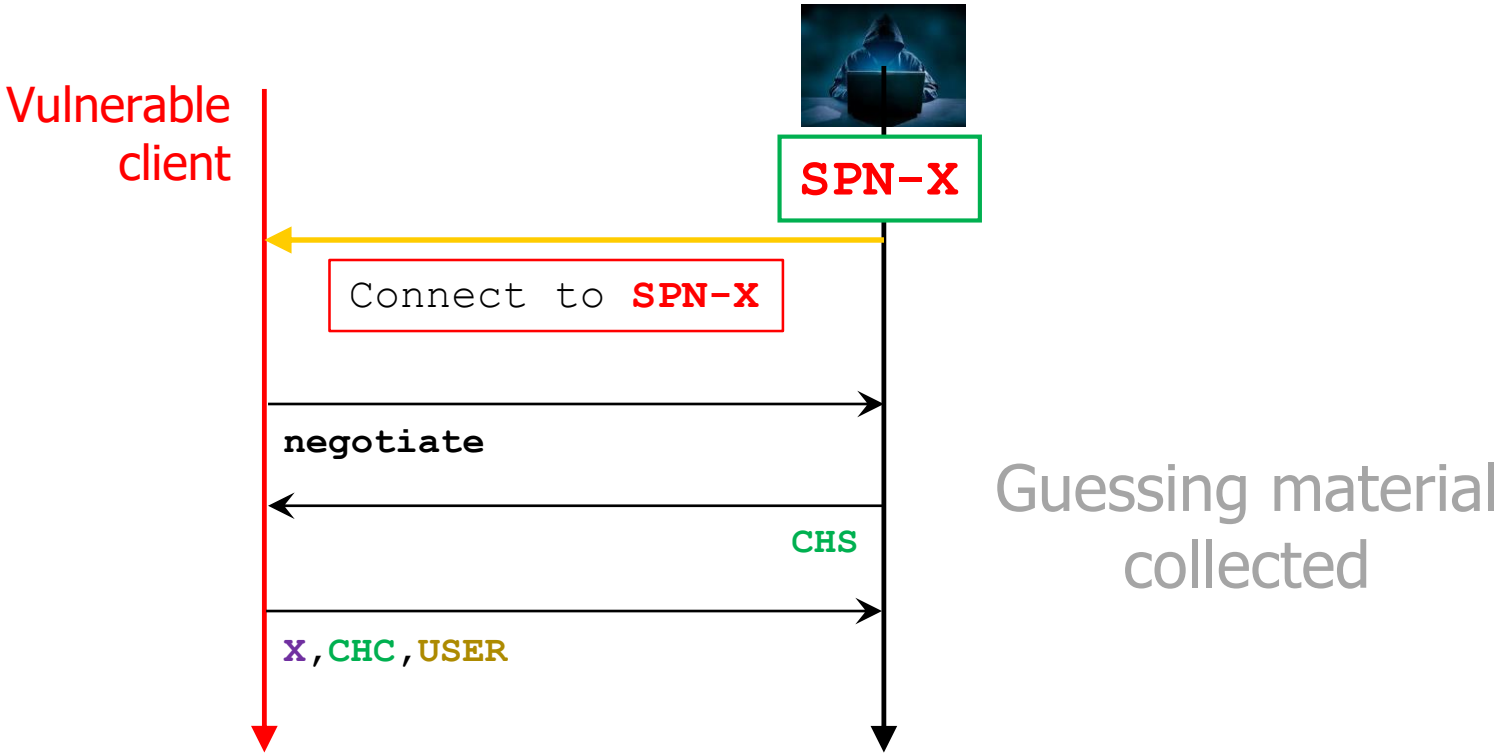
- Credential Access / Forced Authentication
- One of the 5 techniques for becoming MITM



Guessing material collected

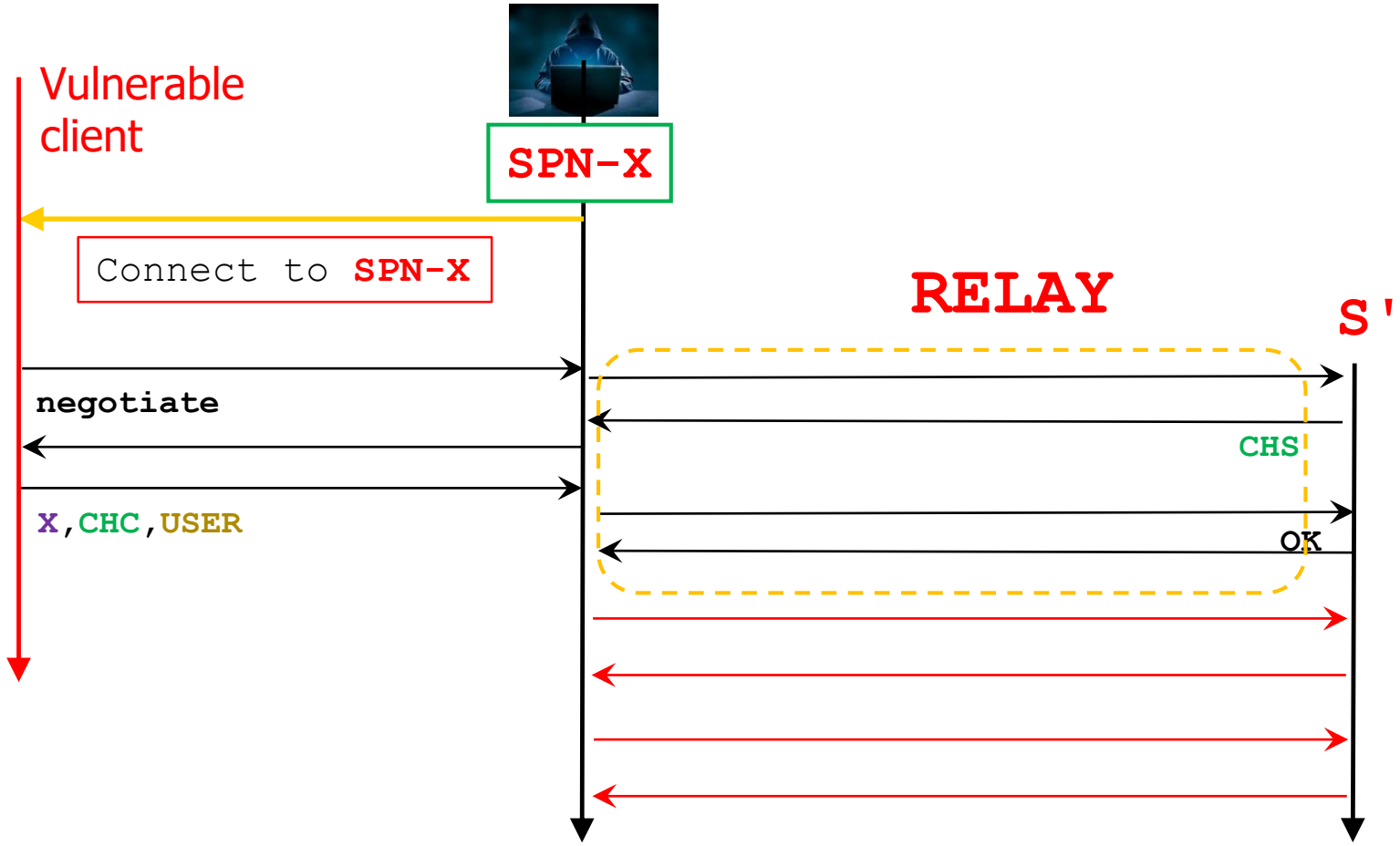
Coerced Authentication (I)

- Certain **vulnerabilities** can be exploited for Credential Access / Forced Authentication **without any user involvement**



Guessing material collected

Coerced Authentication (II)



Think for a moment…

- With Coerced Auth vuln, Attacker has:
  - **Full control on C, USER, When**

- Coerced Auth:
  "Impersonate whoever you want, whenever you want..."
- NTLM Relay:
  "...on the SPN that you want"

- For certains vulns there may be partial control of USER

Real Incident (I)

- Russian hackers targeted European military and transport organizations in newly discovered spying campaign
  https://edition.cnn.com/2023/03/15/politics/russian-hackers-europe-military-organizations-microsoft/index.html

- Vulnerability CVE-2023-23397 (Microsoft Outlook)
- **Coerced authentication** of **Domain Administrator**

- With a **Relay** attack,
  total compromise of an **entire organization** easily

Real Incident (II)

- Launch process P for Relay attack to Domain Controller
  - Command to be relayed to DC omitted here for simplicity
- Store file F on P and make F accessible on SMB+NTLM

- Prepare Outlook meeting invitation with F attached to be fetched from P
- Send invitation to user U in **Domain Admin** group

- When U opens Outlook, Outlook fetches F automatically from P (**coerced auth vulnerability**)
- P relays to Domain Controller as U

- **GAME OVER**

Keep in mind

- 1 Coerced Authentication Vulnerability + NTLM
- 1 Credential (for sending email within the org)

- **Total compromise** with just 3 o 4 commands
- **No** collaboration/action from Domain Admin user

Remarks on NTLM

Remark

- NTLM is used in **every** Windows system
- **NTLM Relay** attacks are a **huge** risk (possible in default configuration)

- "Applications are generally advised **not** to use NTLM "
  1. Microsoft
  2. 2010

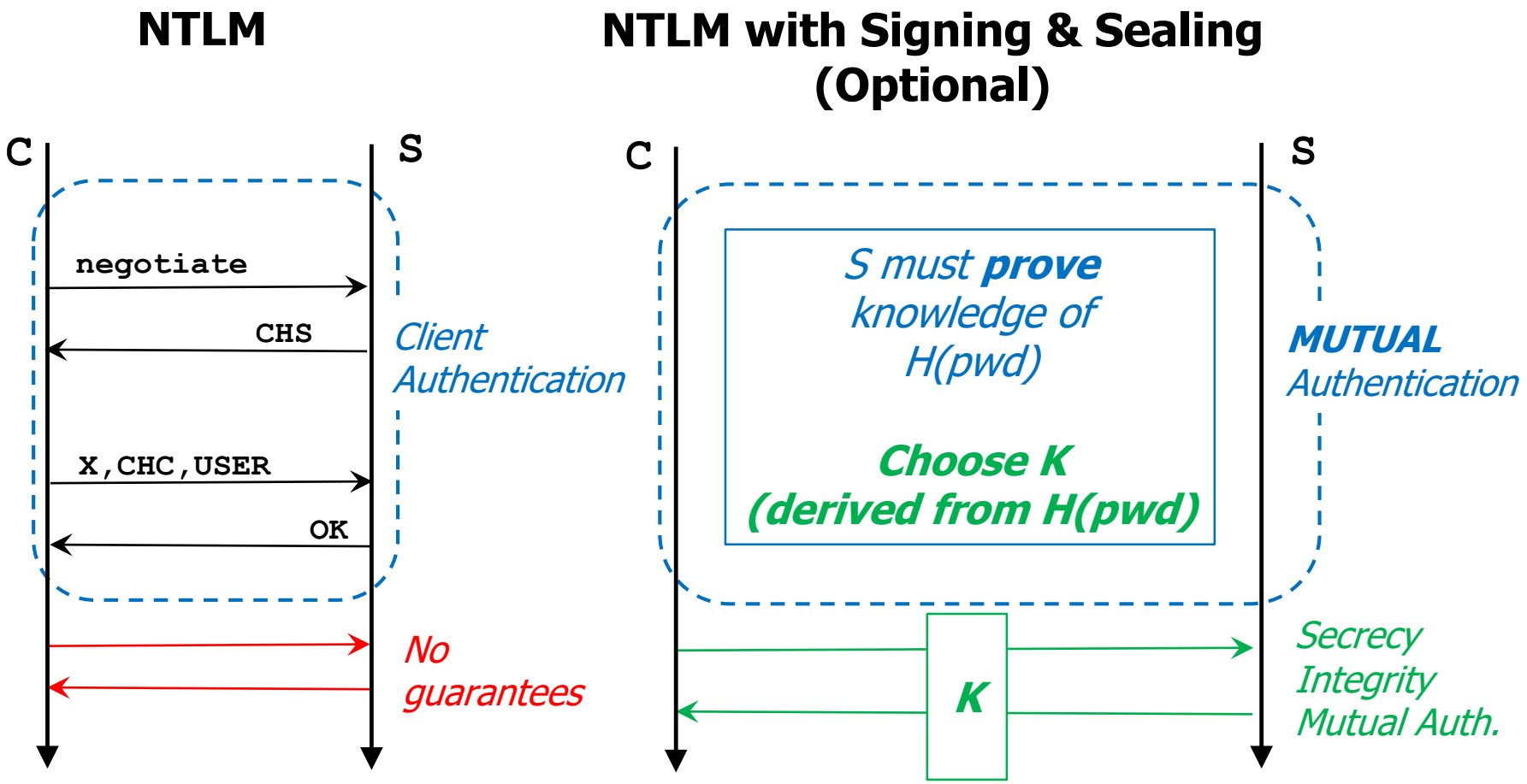
Curiosity

- Every Windows AD system has a group called **Protected Users**
- Accounts in this group are automatically managed with a tighter security policy:
  - …
  - Cannot authenticate with NTLM
  - …

- Not sure how many orgs **know** and **enforce** this...

NTLM: Signing & Sealing

NTLM: Signing & Sealing (I)



NTLM: Signing & Sealing (II)

- Each application message is **signed** and **sealed**
  HMAC_K(msg),E_K(msg)

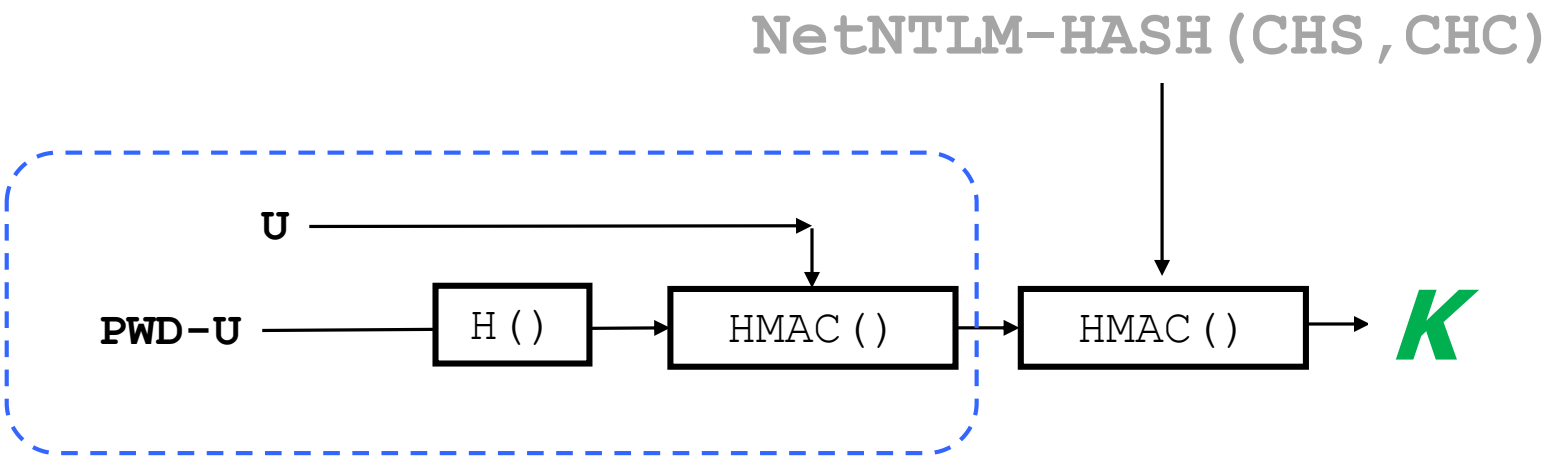
- Signing = Integrity + Authentication (HMAC)
- Sealing = Secrecy (Encryption)

NTLM: Signing & Sealing (III)

- MITM:
  - **Cannot** execute relay attacks (does **not** know K)
  - Can only collect guessing material (and block)

- **Not enabled by default**
  - In practice, huge attack surface
  - It can (and **should**) be enabled!
  - Separately for each service...

Curiosity: NTLM Session Key



Known **only** to USER and S

Kerberos Preliminaries

Kerberos (I)

- 1978: Protocol by **Needham** and **Schroeder**
- 1987: Kerberos v1 (Project Athena MIT)
- 1993: Kerberos v5
- 2005: Kerberos v5-update
- Used in Windows Active Directory since Windows 2000

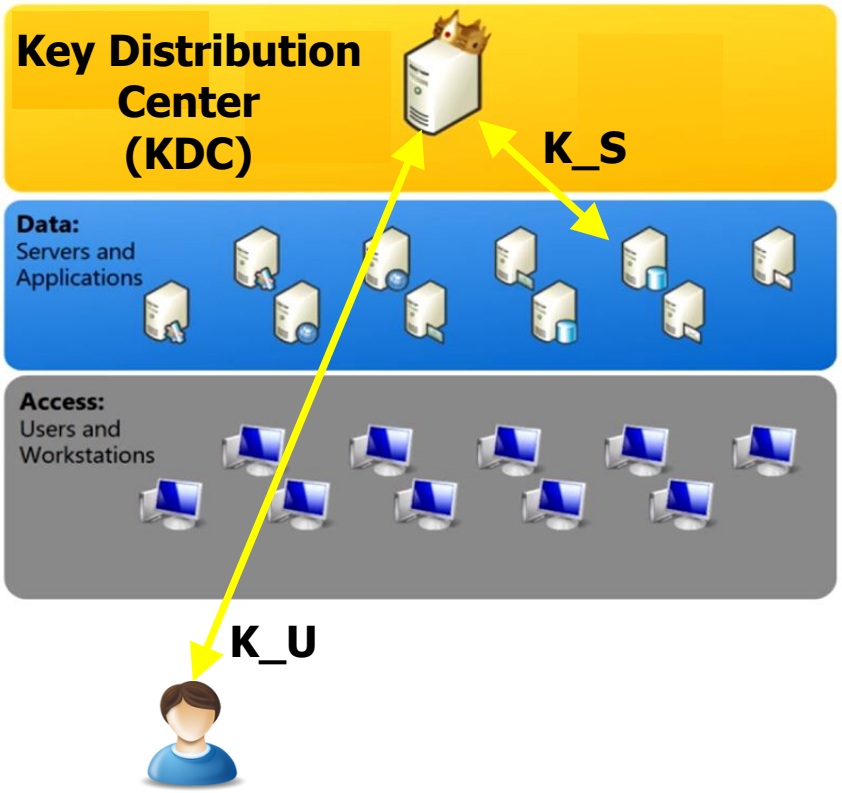
- Notice the delay between research and "technology transfer"
- Some differences between Microsoft and Standard

Kerberos (II)

- **Service configured** with list of supported authentication protocols
  - NTLM — Client Authentication
  - NTLM + Signing & Sealing — Mutual Authentication
  - Kerberos — Mutual Authentication
  - Kerberos + Signing & Sealing — Mutual Authentication
- **No** relay attacks on Kerberos!

- There is actually one complex case in which Kerberos relay can be done on one specific service (local IPSec service)
- Omitted

Kerberos Architecture



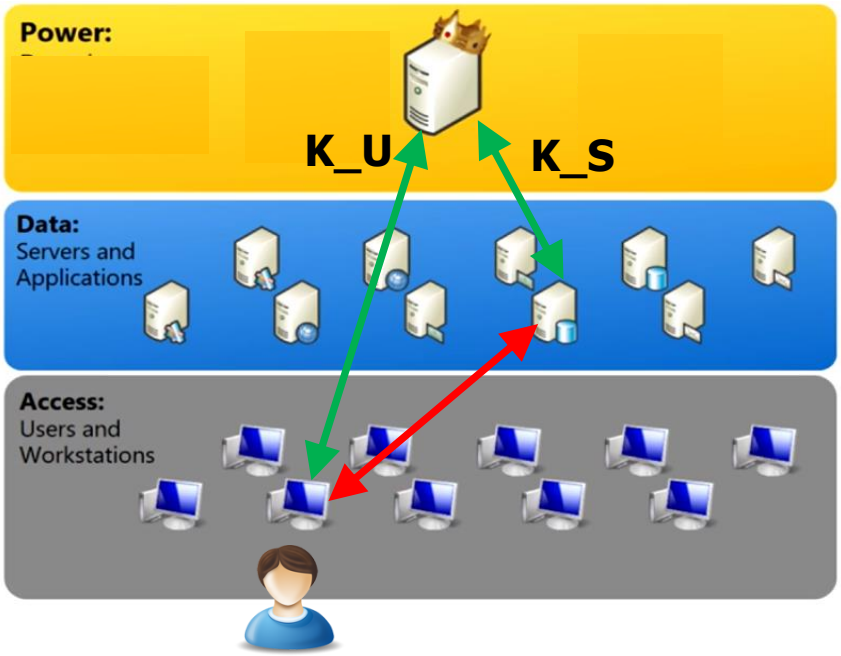
- Every user U has **secret key** K_U
- Every service S has **secret key** K_S
- Derived **deterministically** from **password**
- **KDC** knows **all** secrets keys
- Every node has a "good clock"

Account Keys and KDC

- **KDC** knows **all** secrets keys
- Foreach account `acc-x`:
  - `acc-X, H(pwd-X)` // must support NTLM
  - `K_X = DetermFunc(H(pwd-X))`
- Stealing of NTDS is a catastrophe (PtH)

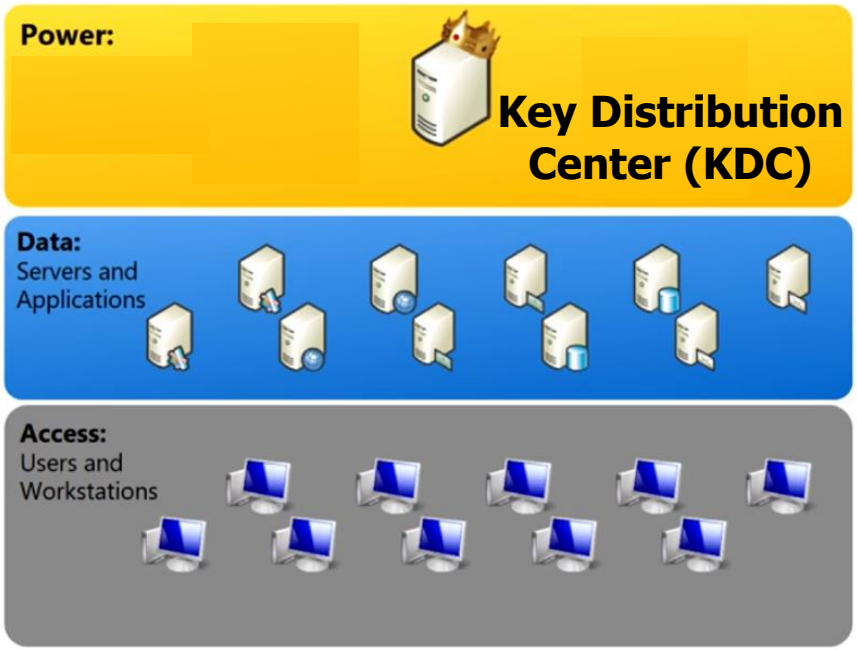
- "Original" Kerberos:
  `acc-x, salt-x, H(concat(salt-x,pwd-x))`

Fundamental Problem (Needham & Schroeder)



- **U and S do not share any key**
- How can they ensure **mutual authentication**?
- How can they **derive a key** for signing & sealing?

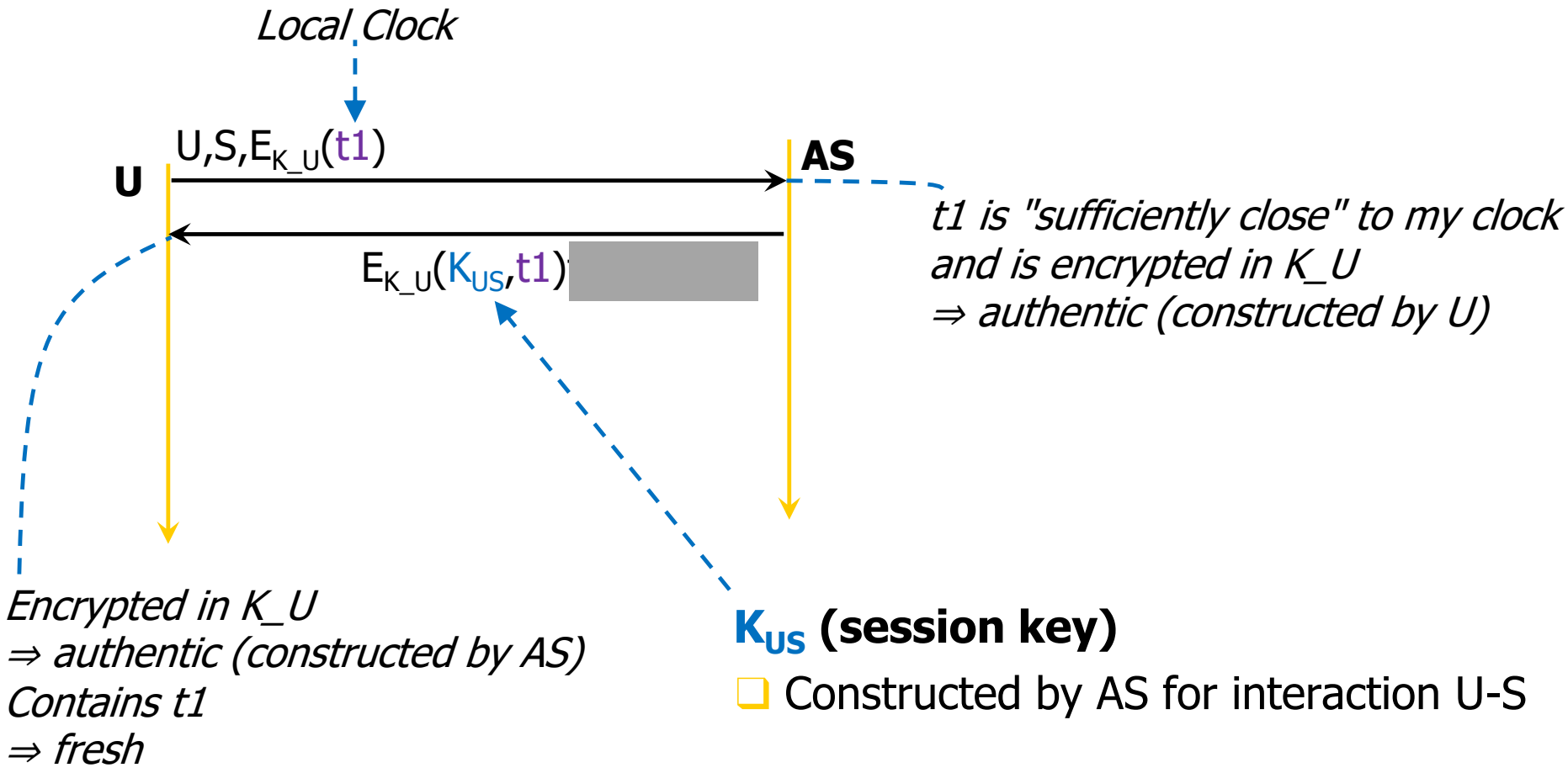
KDC = AS



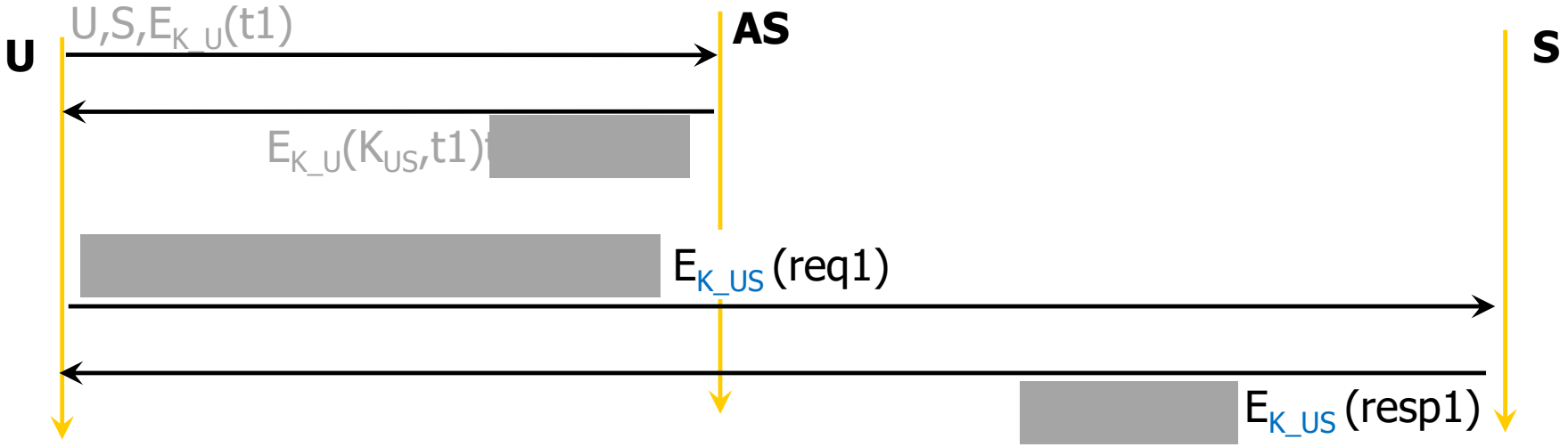
KDC executes:

- **Authentication** Service (AS):
  - Issues **tickets** for accessing services
- Associated with Service Account **`krbtgt`**

“Almost”-Kerberos (I)

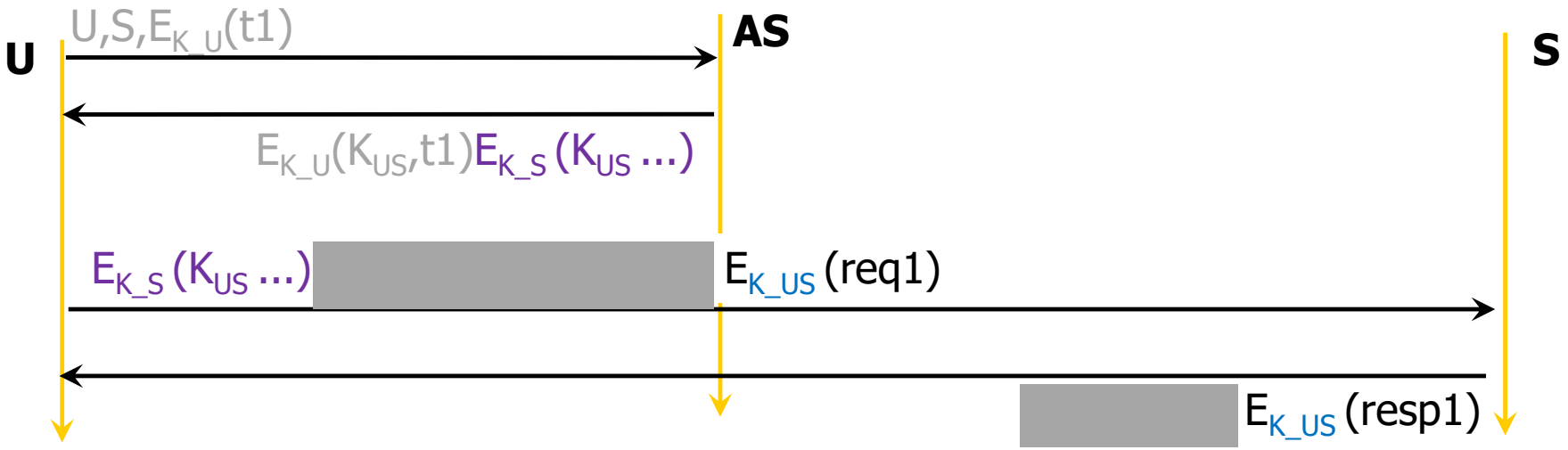


“Almost”-Kerberos (II)



*How can **S** know K_US???*

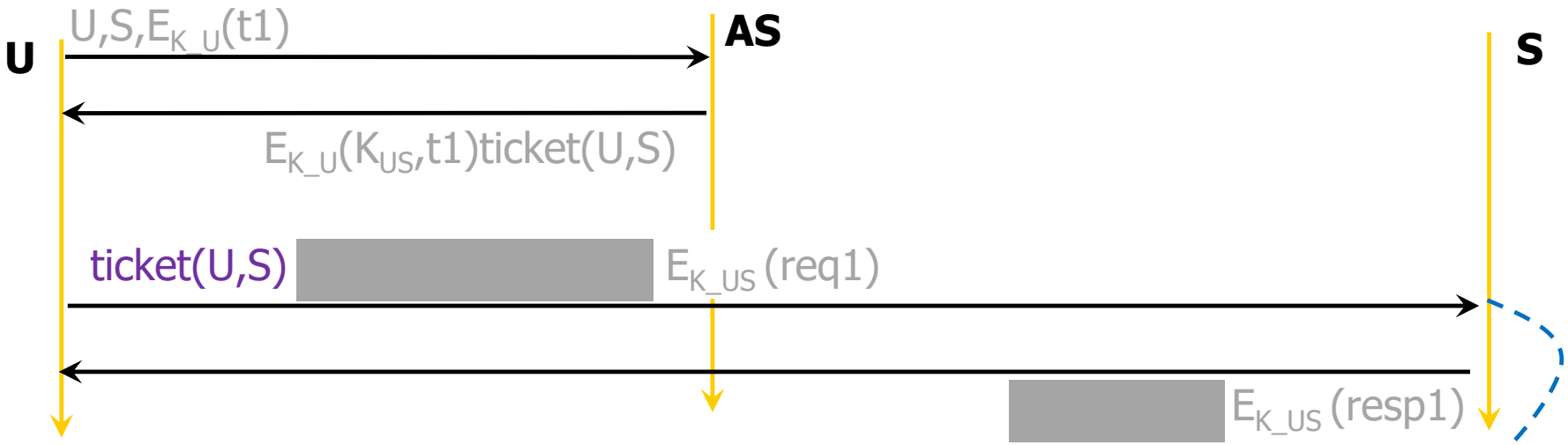
“Almost”-Kerberos (III-a)



$E_{K\_S}(K_{US}$ …) (**service ticket**)

- Contains K_{US}
- Can be read ("open") **only** by S
- Can be issued **only** by AS
- …
- …

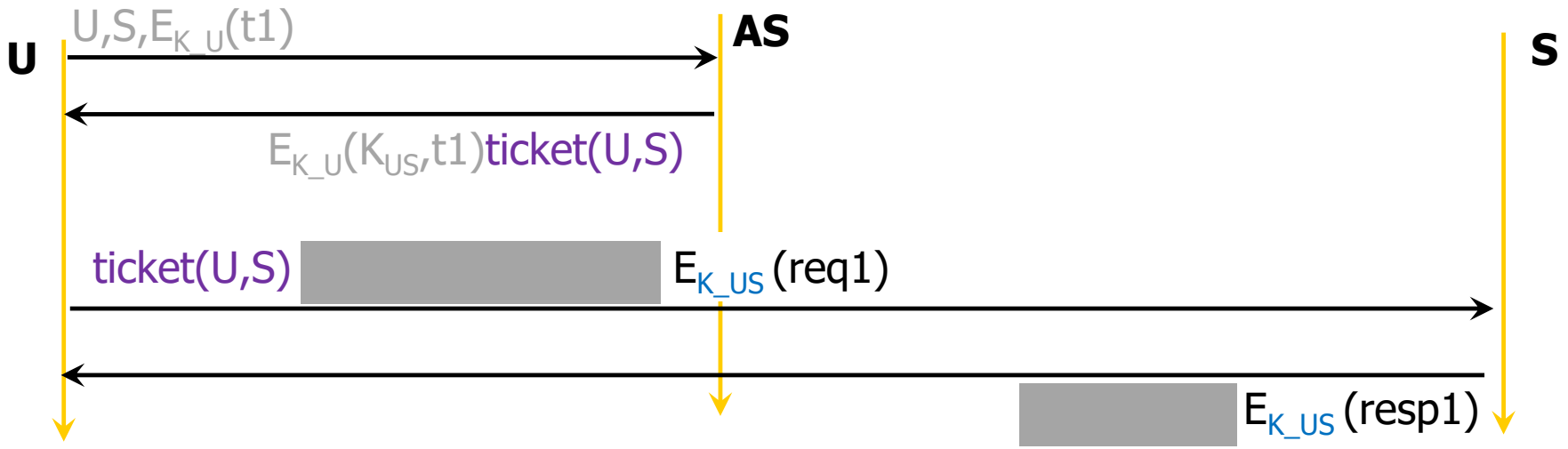
“Almost”-Kerberos (III-b)



ticket(U,S)

1. Authentic?
2. Intact?
3. …

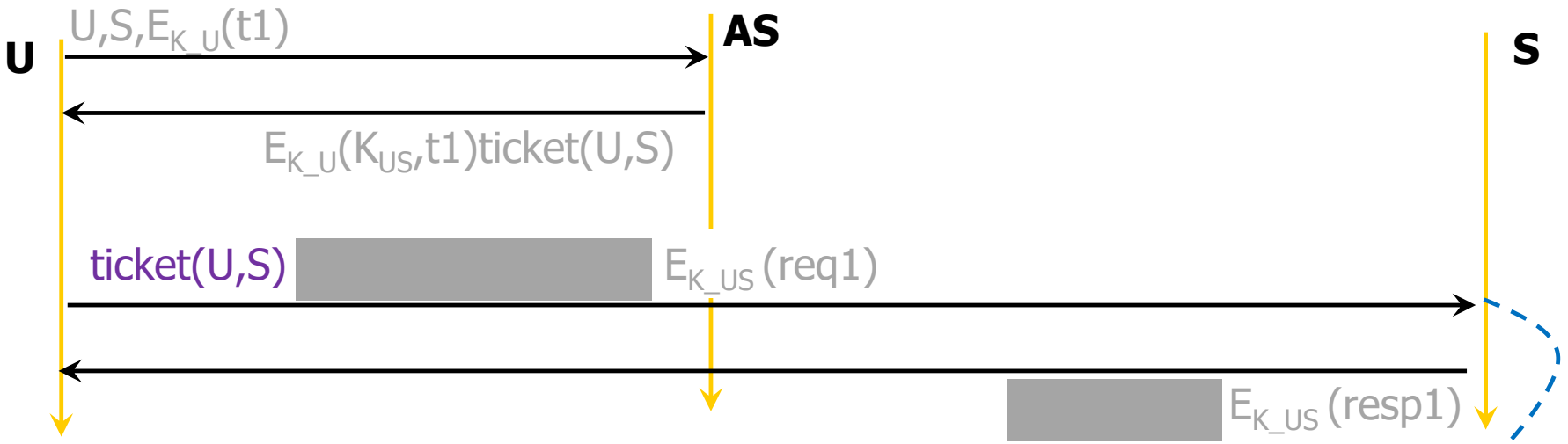
Additional Guarantee



ticket(U,S) (**service ticket**)

- Contains K_{US}
- Can be read ("open") **only** by S
- Can be issued **only** by AS
- Can be used **only** by U**@IP-U**
- Has a short **lifetime** (minutes)

“Almost”-Kerberos (IV)



ticket(U,S)

1. Authentic?
2. Intact?
3. Presented by U@IP-U?
4. Expired?

Service Ticket

ticket(U,S) (**service ticket**)

- Contains K_{US}
- Can be read ("open") **only** by S
- Can be issued **only** by AS
- Can be used **only** by U**@IP-U**
- Has a short **lifetime** (minutes)

$$E_{K\_S}(K_{US}, U, S, IP\text{-}U, T_{AS}, LIFE)$$

Remark 1 on Service Ticket (Tiny little detail)

$E_{K\_S}(K_{US}, U, \mathbf{S}, IP\text{-}U, T_{AS}, LIFE)$

- It also contains SPN
- Account S has **one** password but could be associated with **multiple** SPN

- Omitted for simplicity

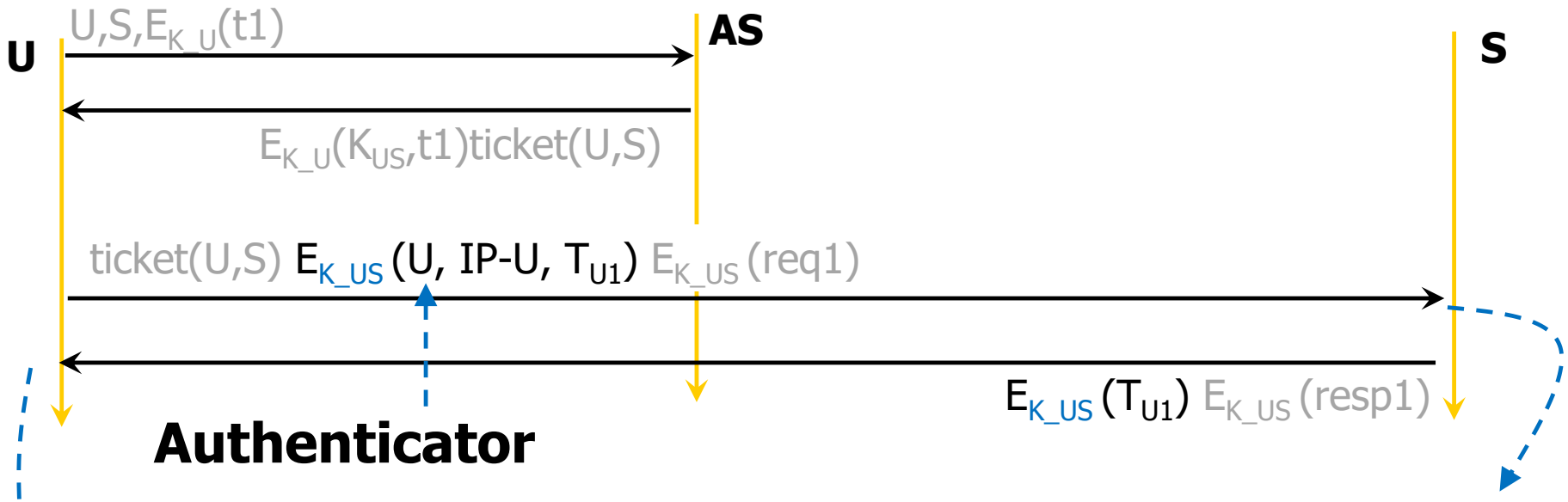
Remark 2 on Service Ticket (Authorization)

$E_{K\_S}(K_{US}, \mathbf{U}, S, IP\text{-}U, T_{AS}, LIFE)$

- U described as a **PAC** (Privilege Attribute Certificate)
  1. U
  2. **List of groups that U belongs to**
  3. Certain info useful for authorization

- S decides whether to **authorize** a request on R based on:
  - ACL of R
  - PAC in the received service ticket

“Almost”-Kerberos (V)

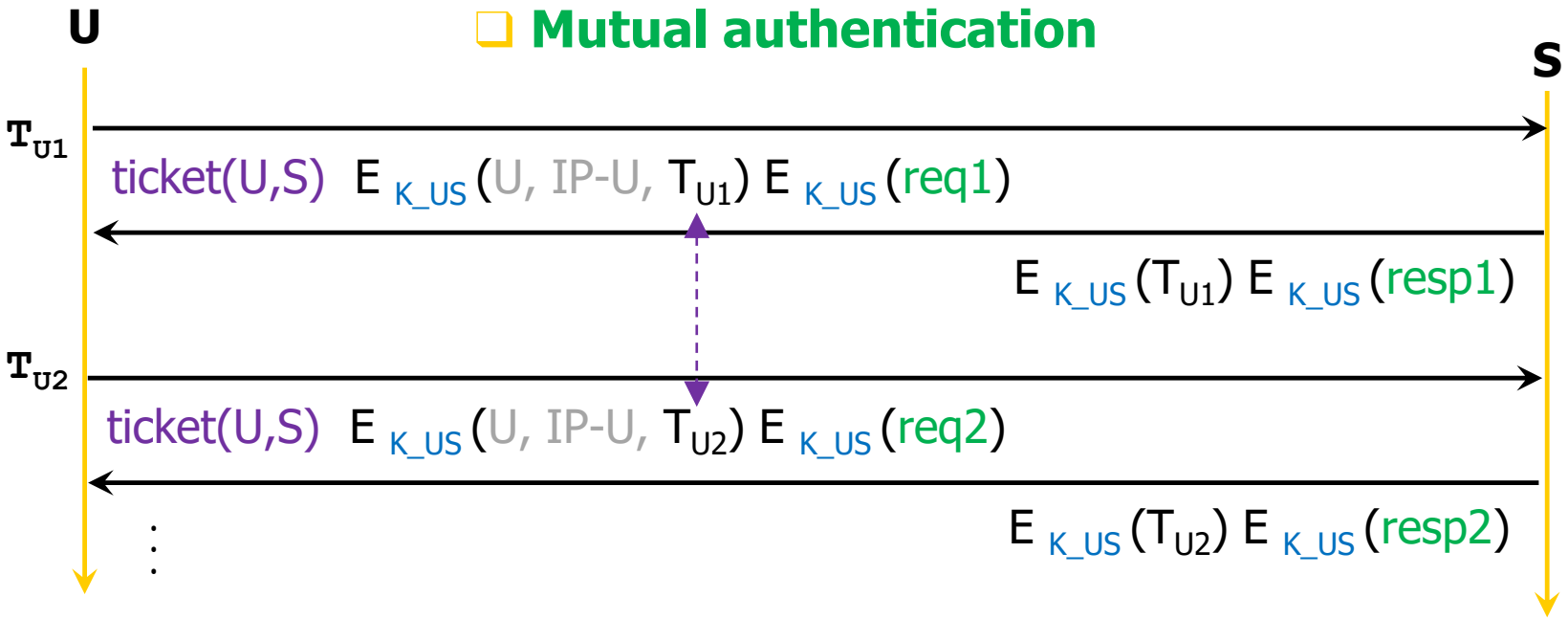


Authenticator contains U and
The request is sent from IP-U
*⇒ Sender is the **legitimate owner** of ticket(U,S)*

It was able to decrypt Authenticator
*⇒ response authentic and **fresh***

Communicating with S: Signing&Sealing

- Secrecy
- Integrity
- Mutual authentication

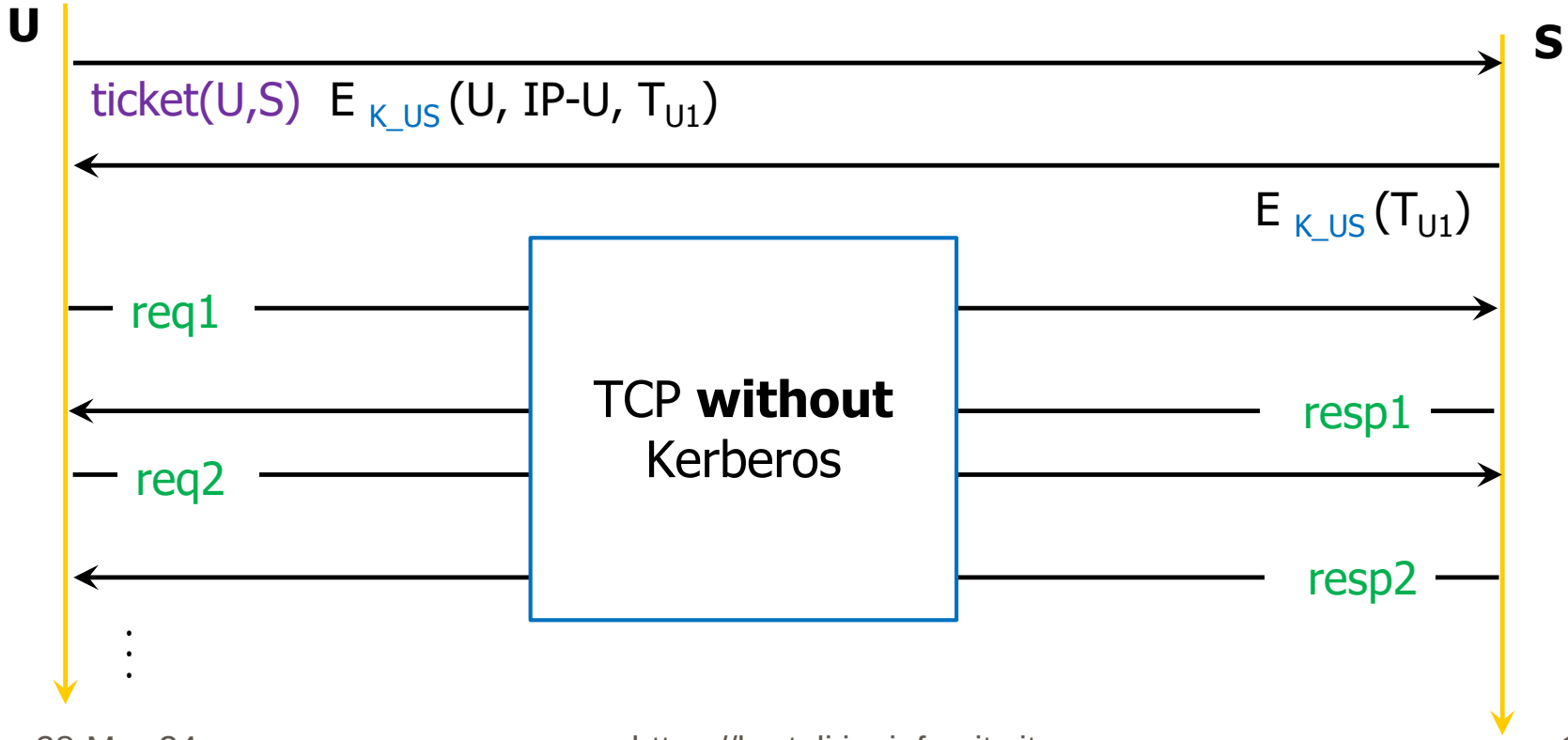


- Ticket **always the same** (until expiration)
- Authenticator **changes** at each interaction

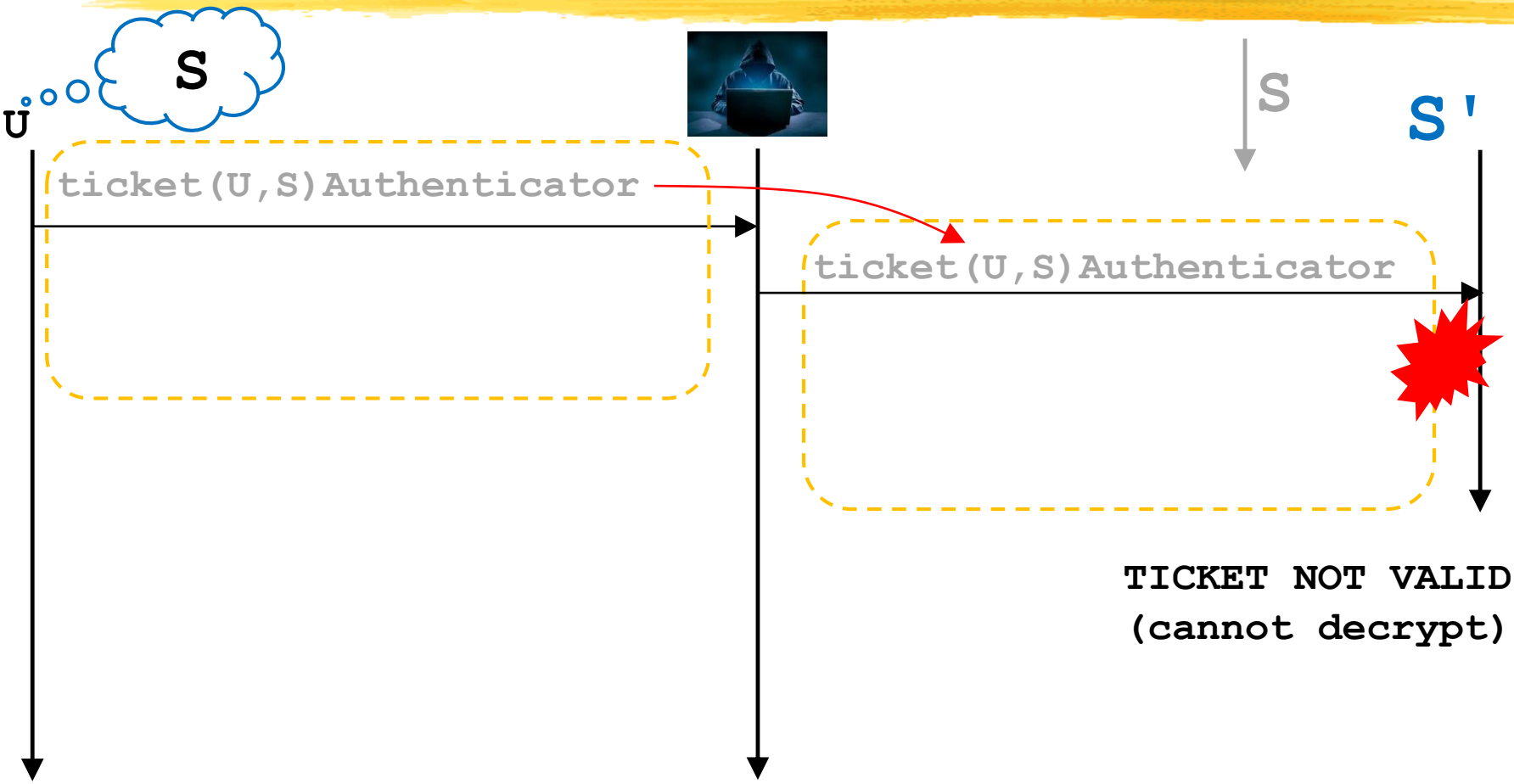
Communicating with S: Default

- ~~Secrecy~~
- ~~Integrity~~
- Mutual authentication

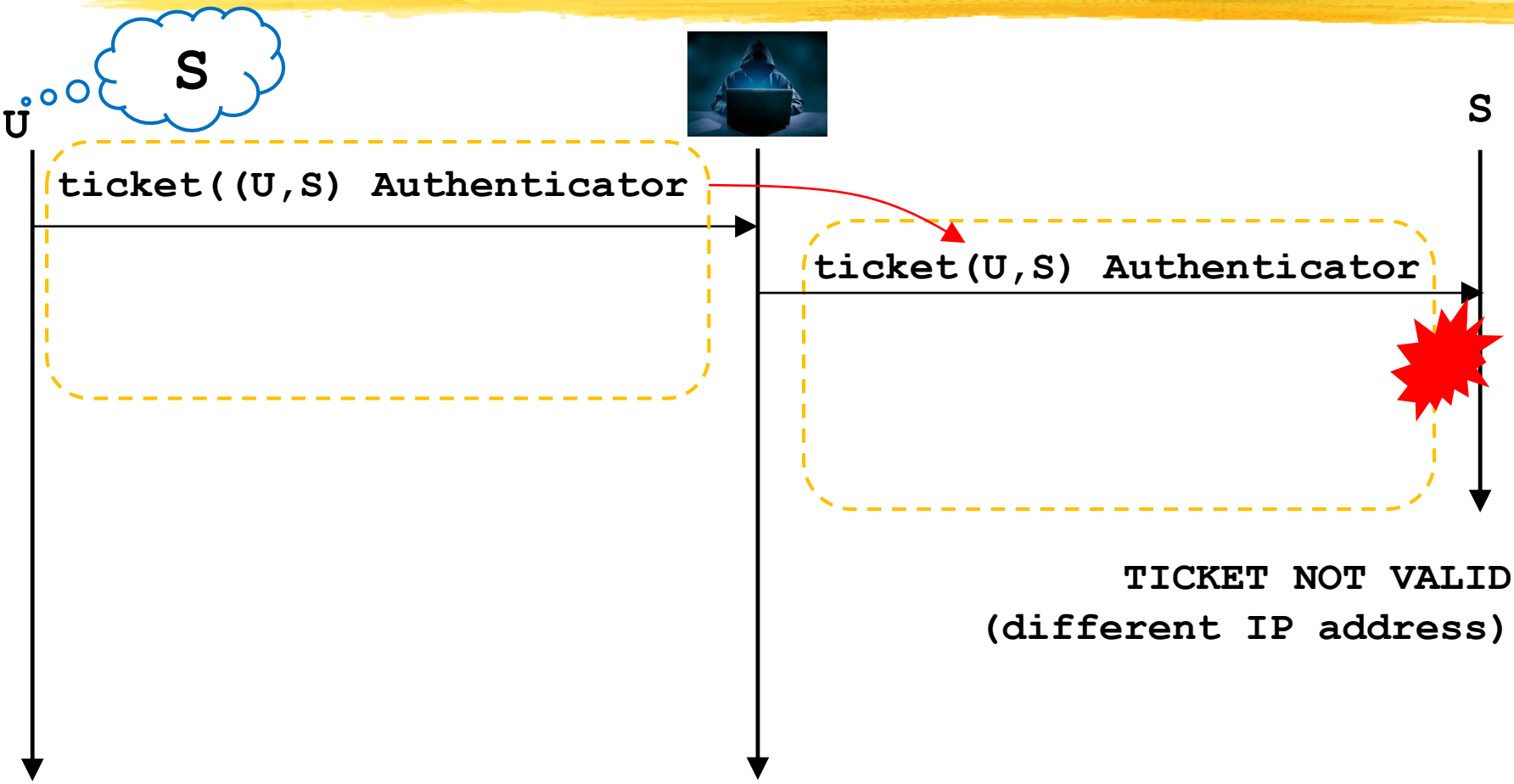
ONLY in the first interaction



Do NOT underestimate default: NO RELAY

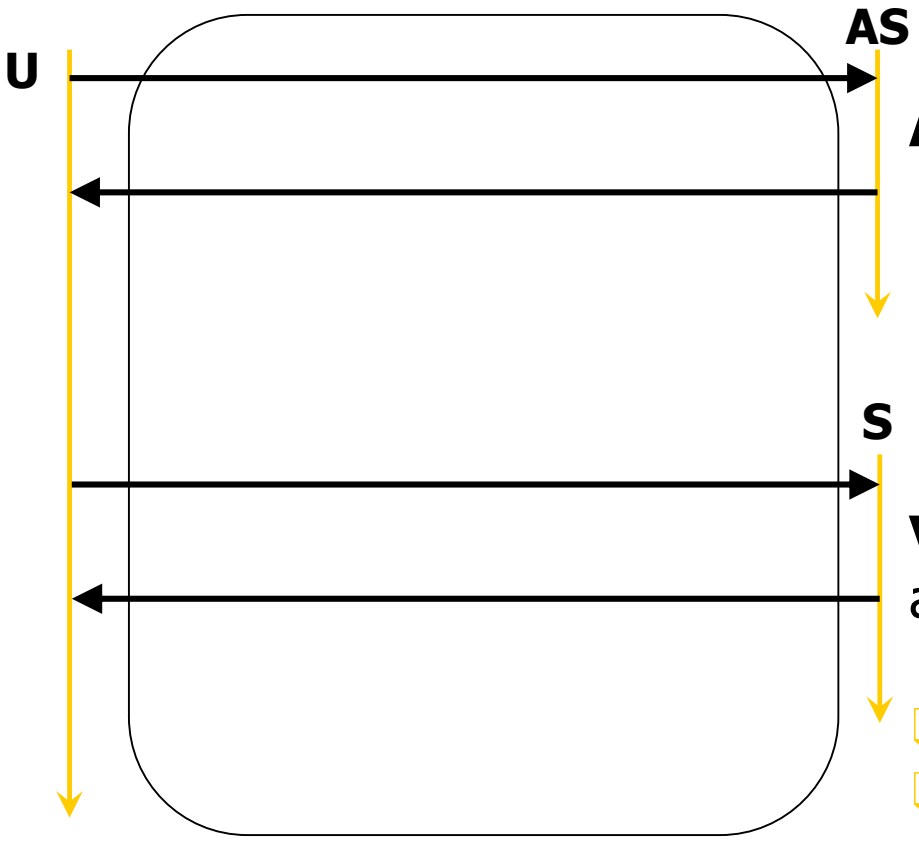


Do NOT underestimate default



Digression: Authenticate vs Verify

General Framework (I)



Authenticates U

Verifies that U has been "recently" authenticated by AS

- Communication on **untrusted** channel
- No direct communication **with AS**

OAuth / SAML

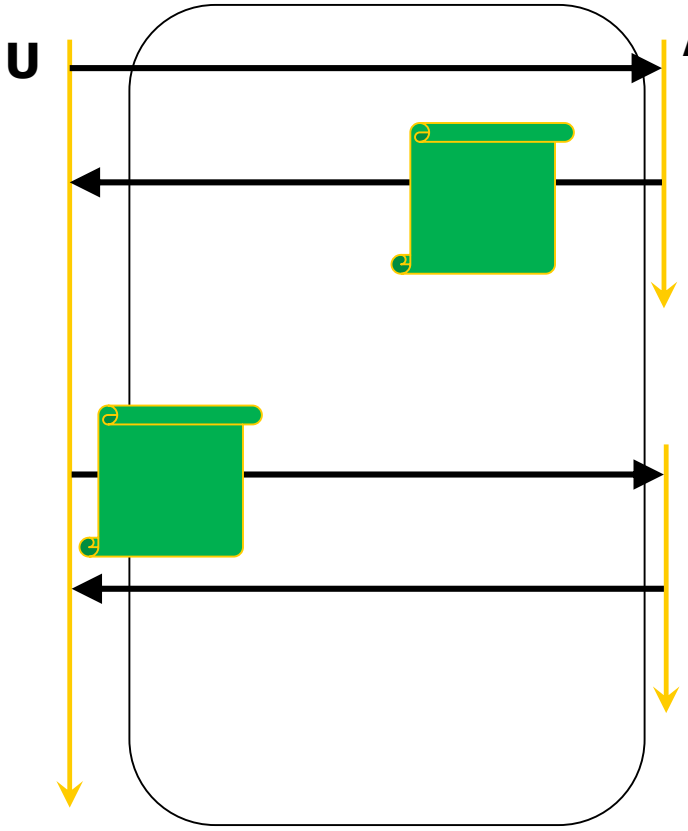
- Authenticator and Verifier may belong to **different** organizations
- Protocol could be over **HTTPS** and suitable client-side for **browsers**

- Example: SPID
- Example: Non-Google Service (e.g., ChatGPT) with Google Authentication

- Authenticator could issue **Authorization Assertions** and Verifier could decide to apply them in its organization

- Example: UniTS authenticates for Office365 services at Microsoft

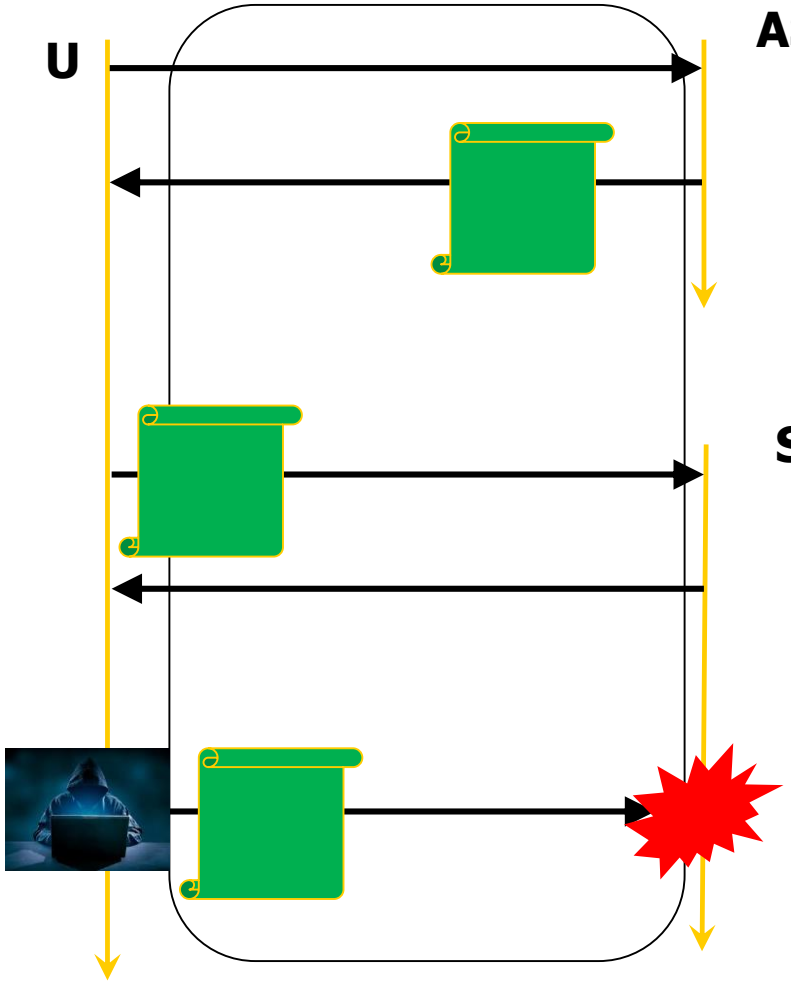
General Framework (II)



- **Authentication Assertion**
  "AS has authenticated U at time T"

- Cryptographic guarantees on assertion:
  - Authentic
  - Intact
  - "Fresh"

General Framework (III)



- **Authentication Assertion**
  "AS has authenticated U at time T"

- Cryptographic guarantees on ssertion:
  - Authentic
  - Intact
  - "Fresh"

  +
- **Protocol** ensuring it is presented by its **legitimate owner**

Understanding “Almost”-Kerberos

Attack Objective (REMIND)

- Obtain **"guessing material"** from execution of authentication protocol
  - IF offline guessing successful
  - THEN password of observed account(s)
- Analyzed **later** (in "real" Kerberos)

Or

- **Impersonate** one of the two parties in execution of authentication protocol
- **Next slides** (and keep in mind NO RELAY)

REMIND

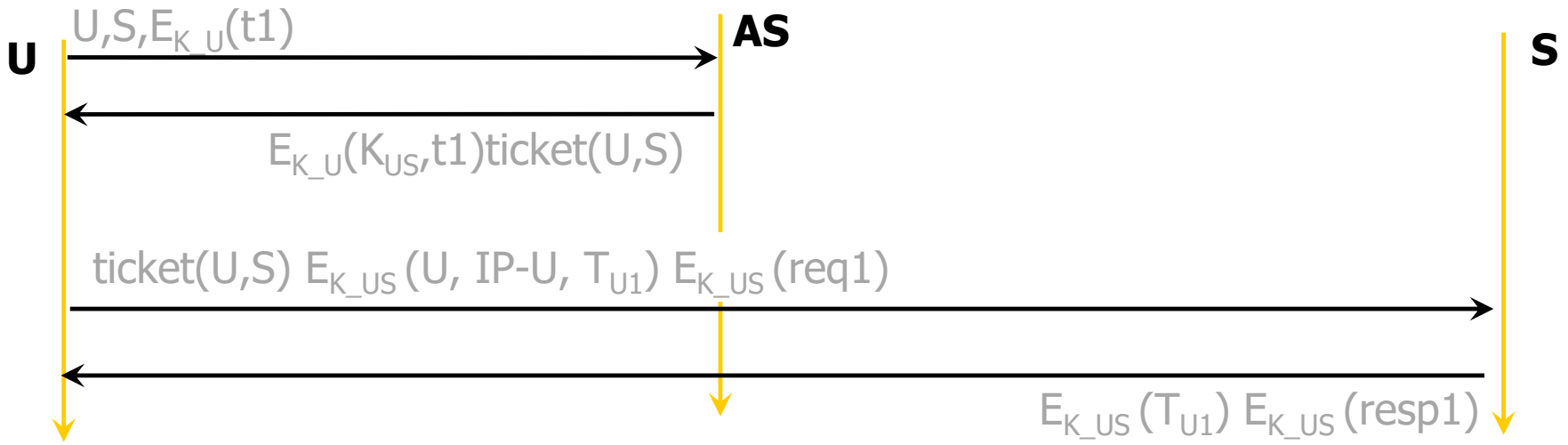
- When describing messages with **secrecy**
  we will **not** write the HMAC explicitly (but it is there!)

- Secrecy
- Integrity
- Mutual Authentication (the other end knows the private key)

$E_K(msg)\,HMAC(K, E_K(msg))$

$E_{K\_US}(T_{U1})\,E_{K\_US}(resp1)$
means
$E_{K\_US}(T_{U1})\,HMAC(K\_US, E_{K\_US}(T_{U1}))$
$E_{K\_US}(resp1)\,HMAC(K\_US, E_{K\_US}(resp1))$

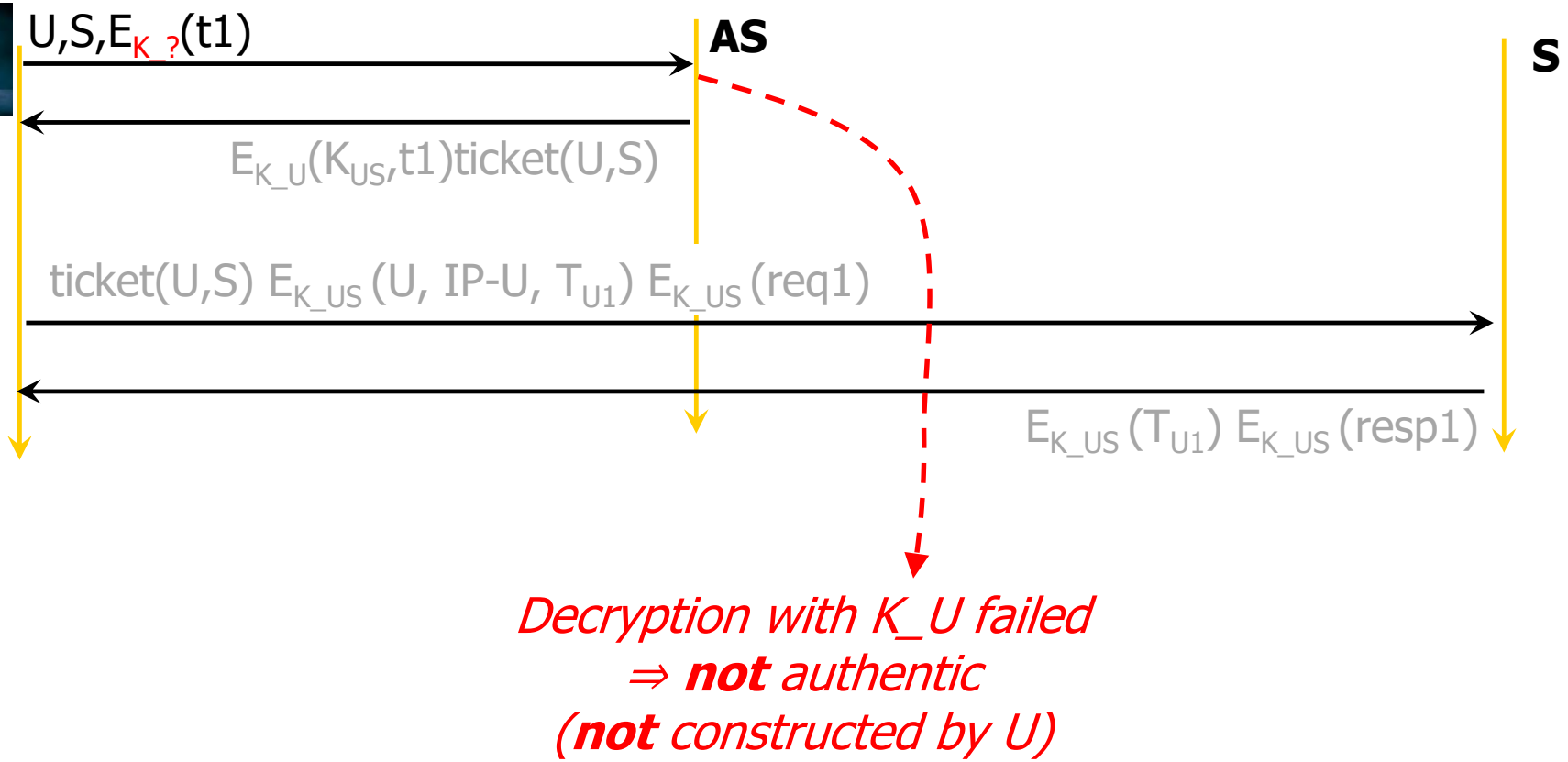
Fact



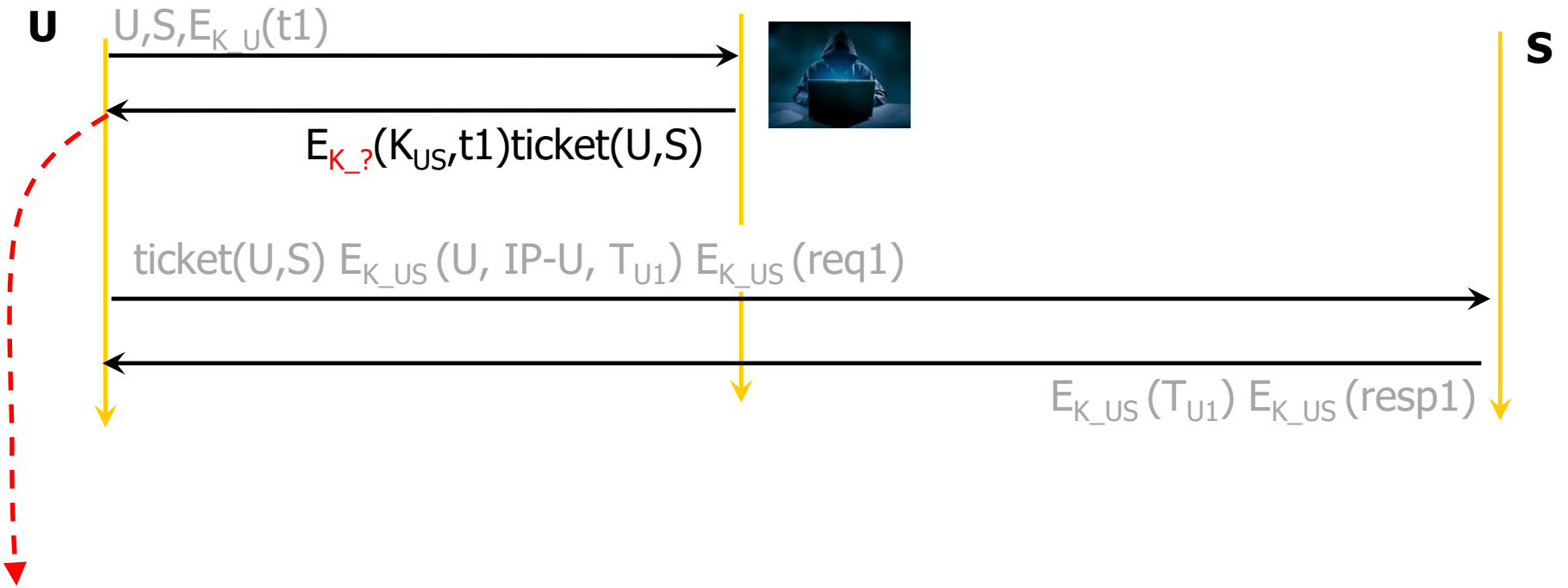
- Every interaction has:
  - Secrecy
  - Integrity
  - Mutual authentication

- Network Attacker cannot:
  - Read
  - Modify
  - Forge

Cannot forge: Example (I)

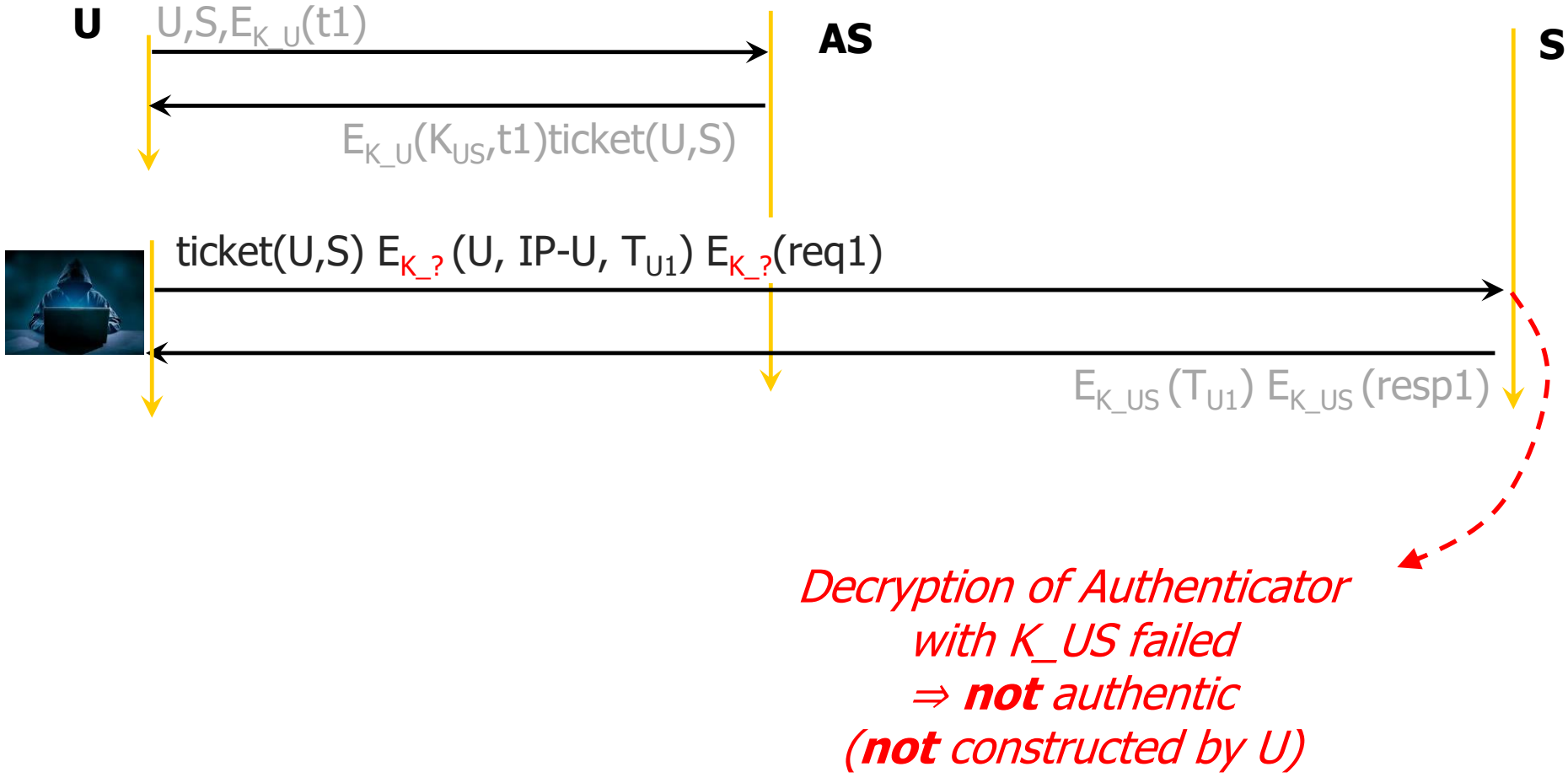


Cannot forge: Example (II)

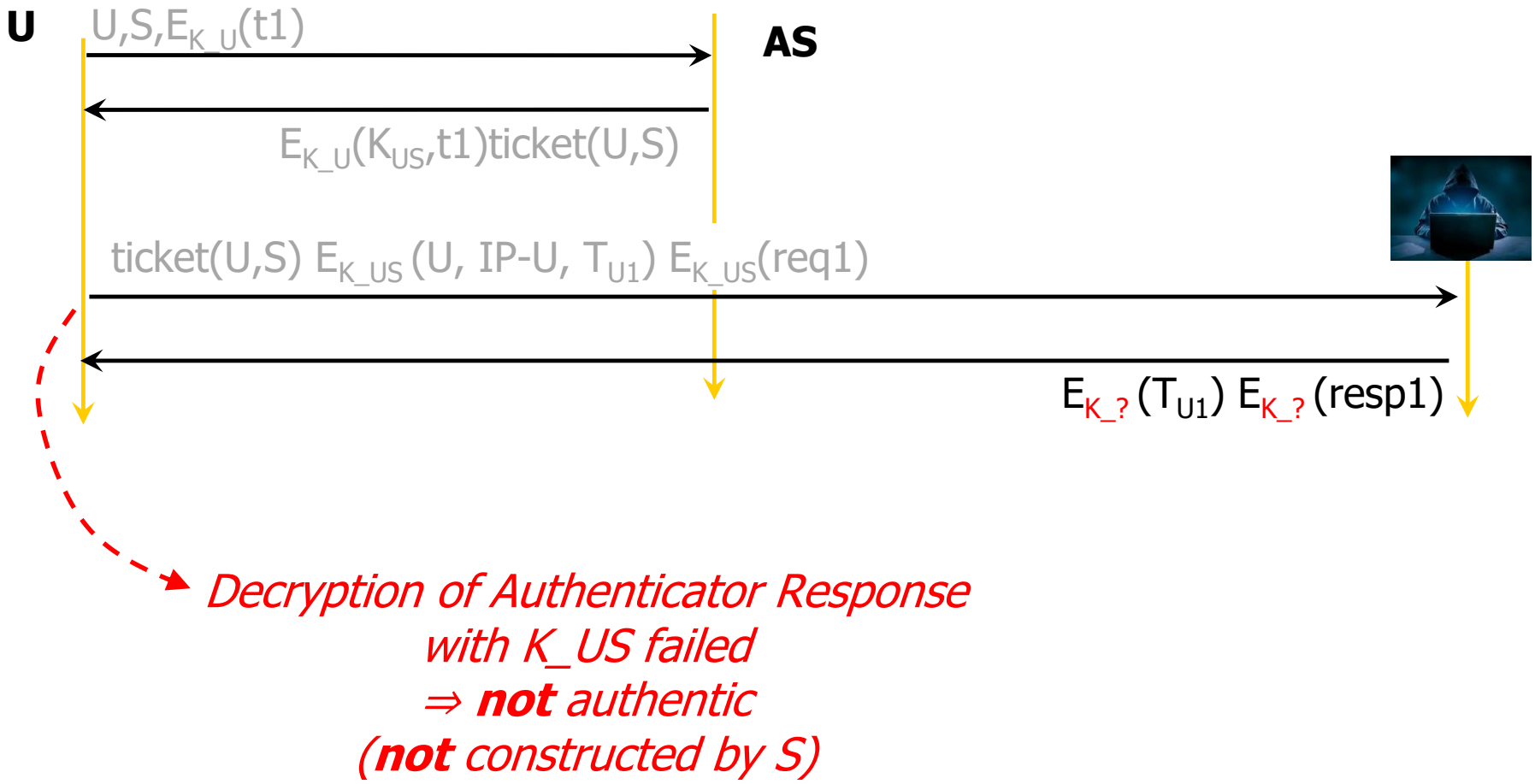


Decryption with K_U failed
*⇒ **not** authentic*
*(**not** constructed by AS)*

Cannot forge: Example (III)

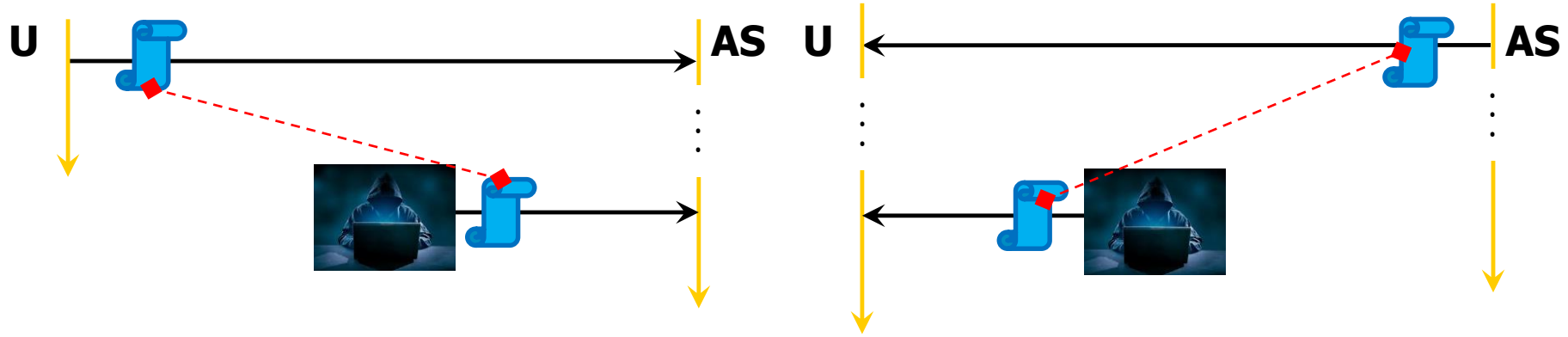


Cannot forge: Example (IV)

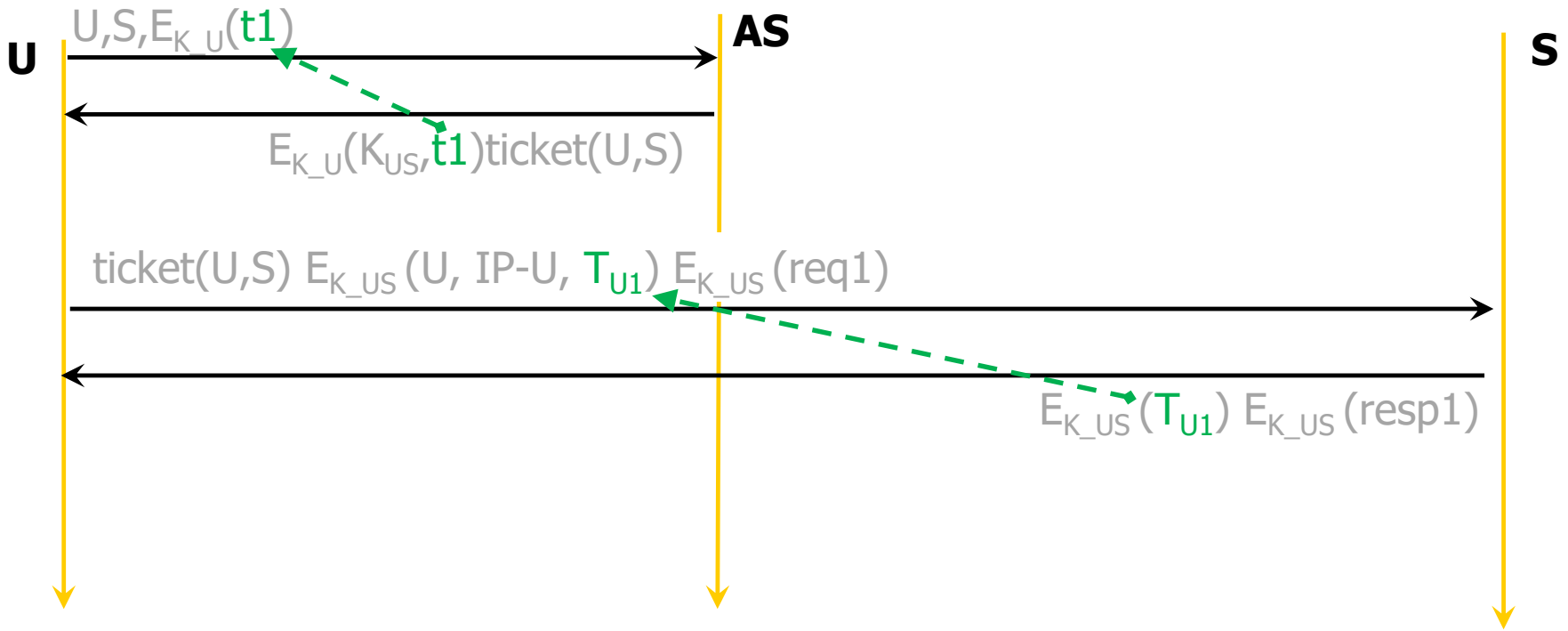


Replay (≠ Relay)

- Network Attacker cannot read / modify / forge messages
- Network Attacker cannot **RELAY**
  - Messages addressed to S: Send "now" to **S'**
- Network Attacker could **REPLAY**:
  - Messages addressed to X: Send "**later**" to X **again**

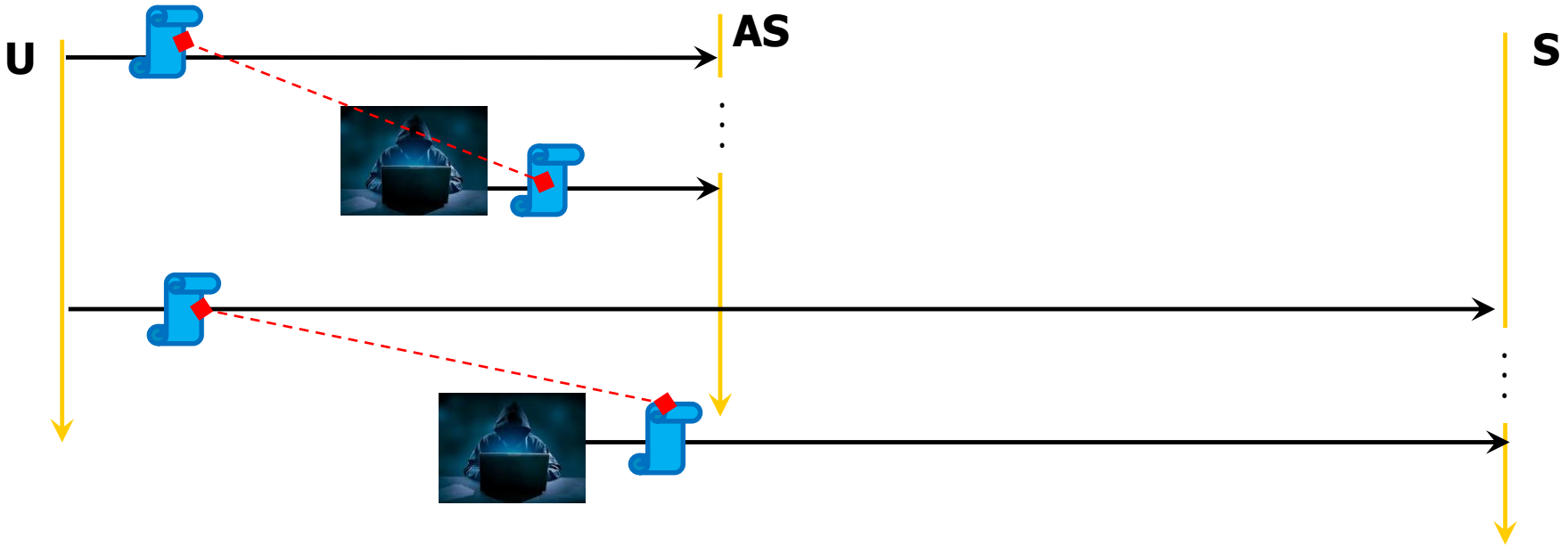


Response Replay: Detected



- Every request carries an **unique** timestamp
- **Response replay** detected easily (timestamp will be different from the expected one)

Request Replay ?

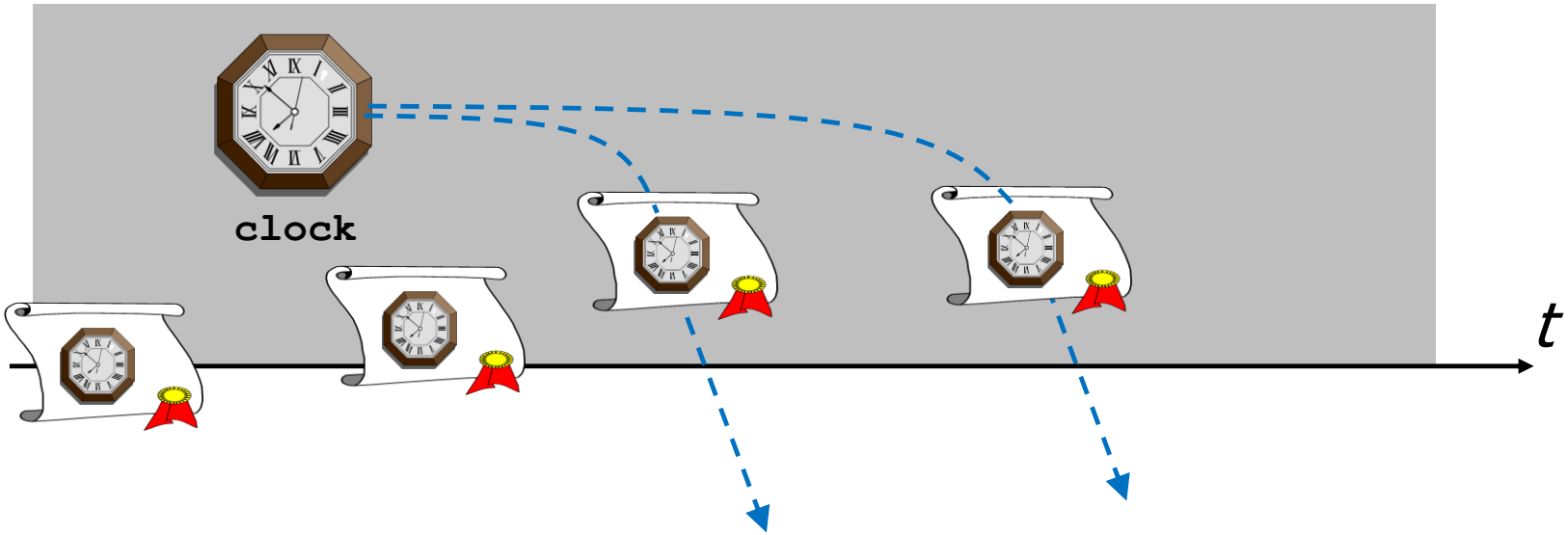


- Requests are not matched to anything
- Receiver would process the replayed request!
  (BIG problem: **one** request executed **many** times…)

Request Replay: Defense

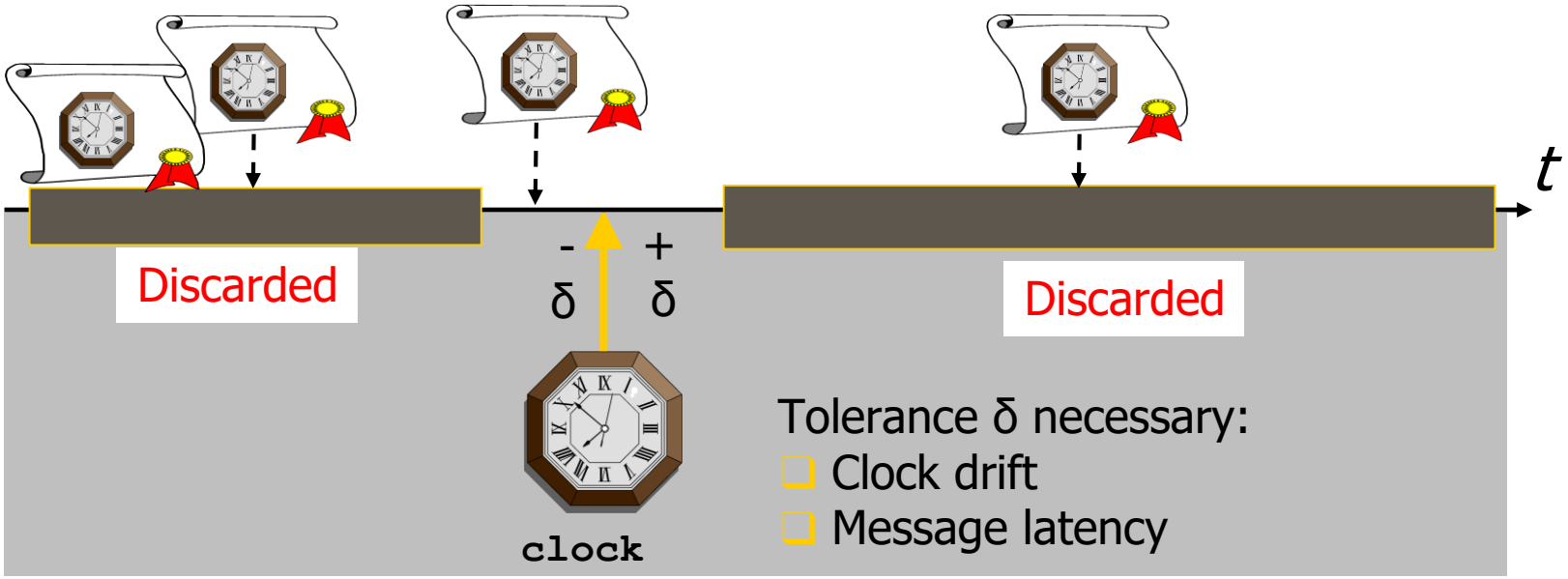
Request Replay: Defense (I)

- Client: inserts `clock` in **every** message sent

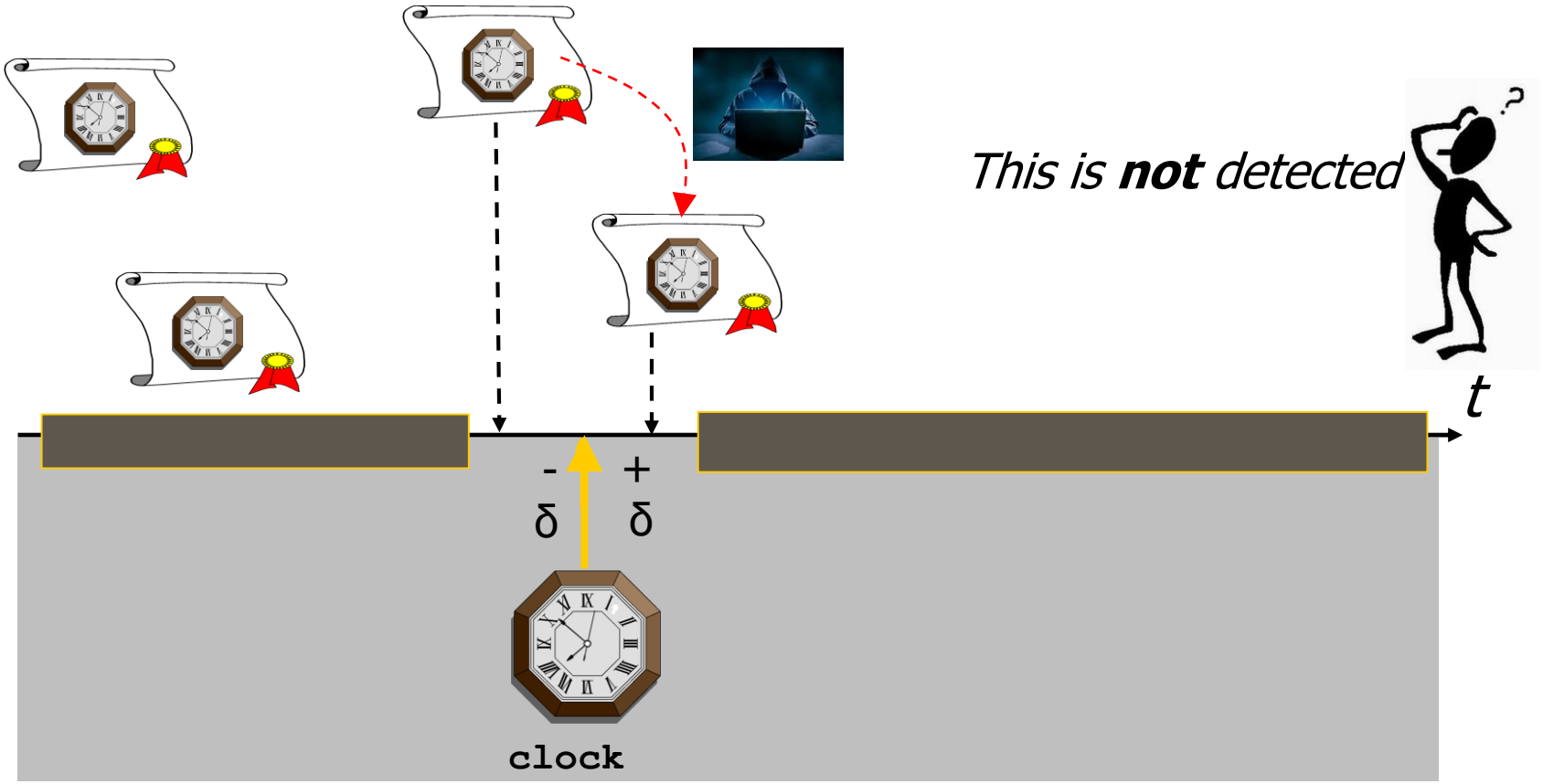


Request Replay: Defense (II)

- **Service**:
  1. Compare msg.clock to rcv.clock
  2. Discard every message received with $|\text{msg.clock} - \text{rcv.clock}| > \delta$
  3. ...

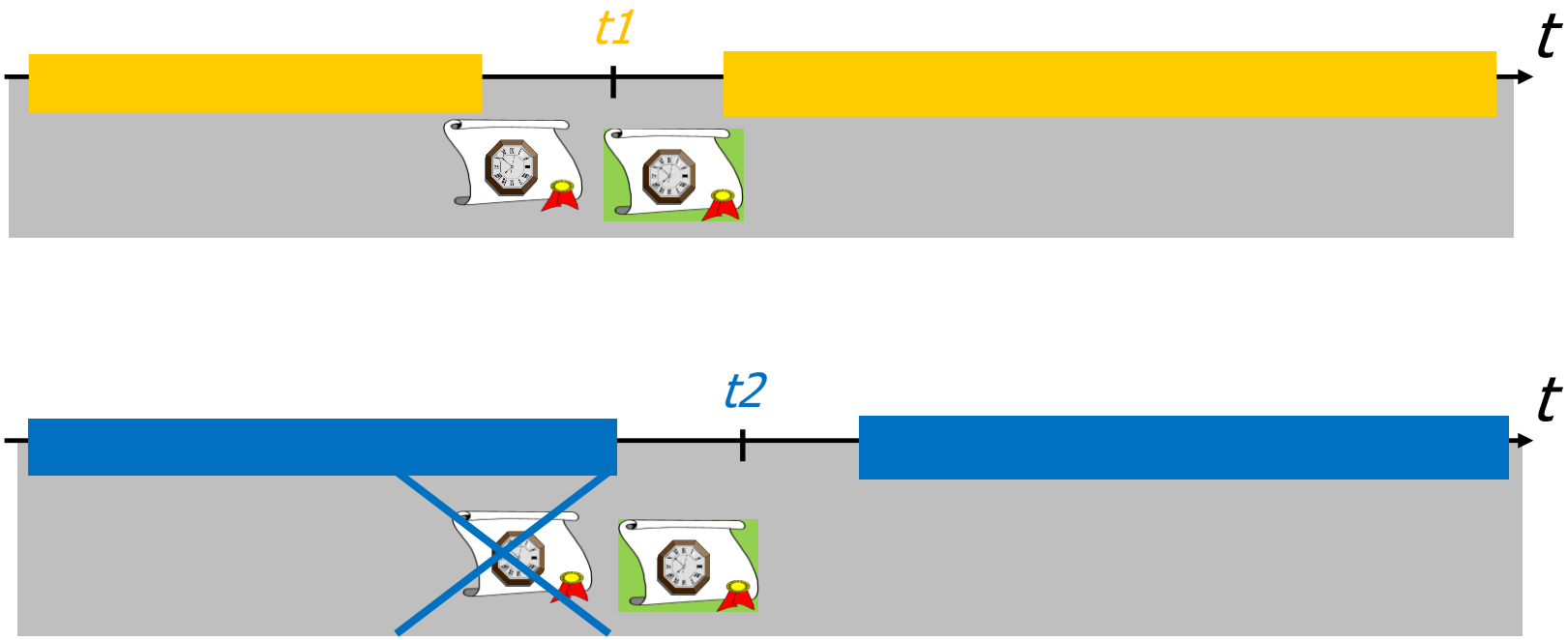


Hhmmm...



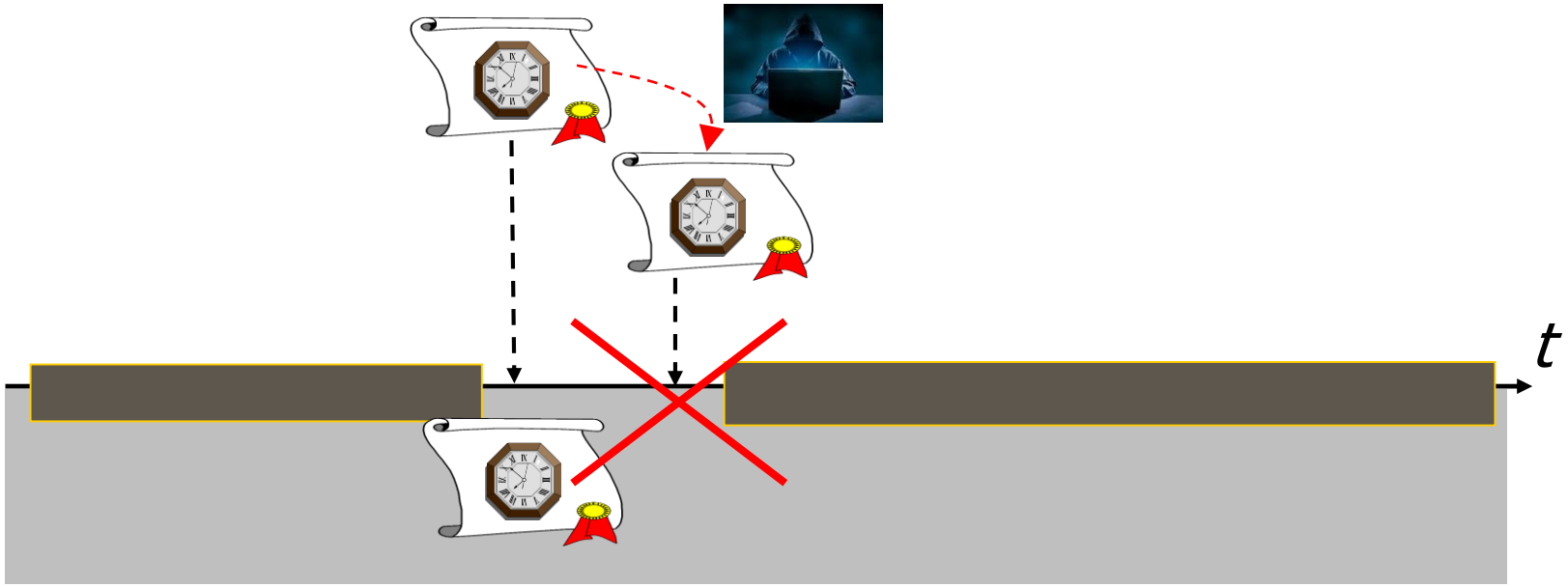
Replay Cache

- Service:
  - Store in a **Replay Cache** all **received** messages with msg.clock still within δ



Request Replay to S: Defense (III)

- Service:
  1. Compare msg.clock to rcv.clock
  2. Discard every message received with $|msg.clock - rcv.clock| > \delta$
  3. IF msg is in Replay Cache THEN discard msg



Remark

- Service:
  1. Compare msg.clock to rcv.clock
  2. Discard every message received with $|msg.clock - rcv.clock| > \delta$
  3. IF msg is in Replay Cache THEN discard msg

- Technique used in **many** modern protocols

- OAuth, SPID

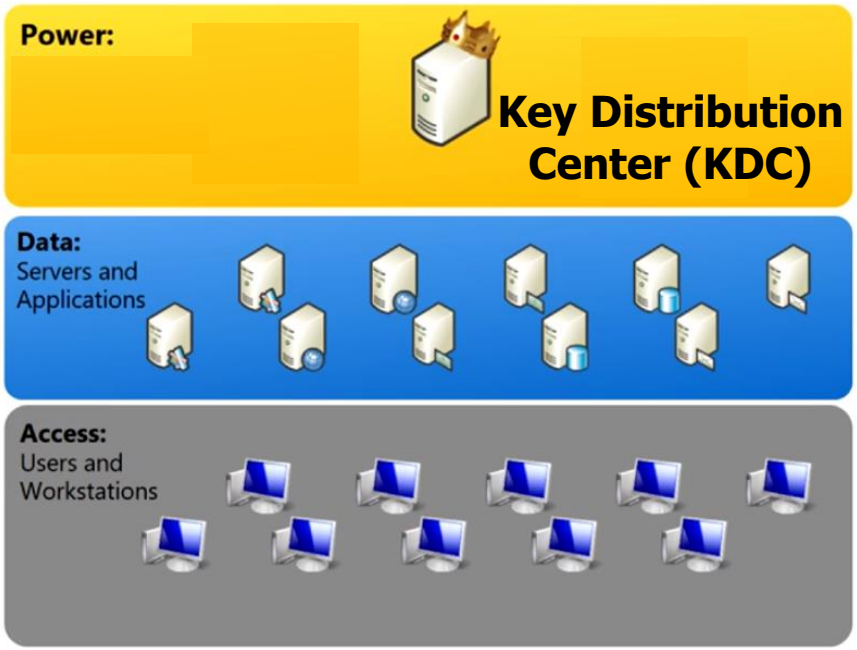
Kerberos

Problem with "Almost"-Kerberos

- Ticket lifetime in the order of minutes
  ⇒ **K_U** is reused **every few minutes** (for contacting AS)

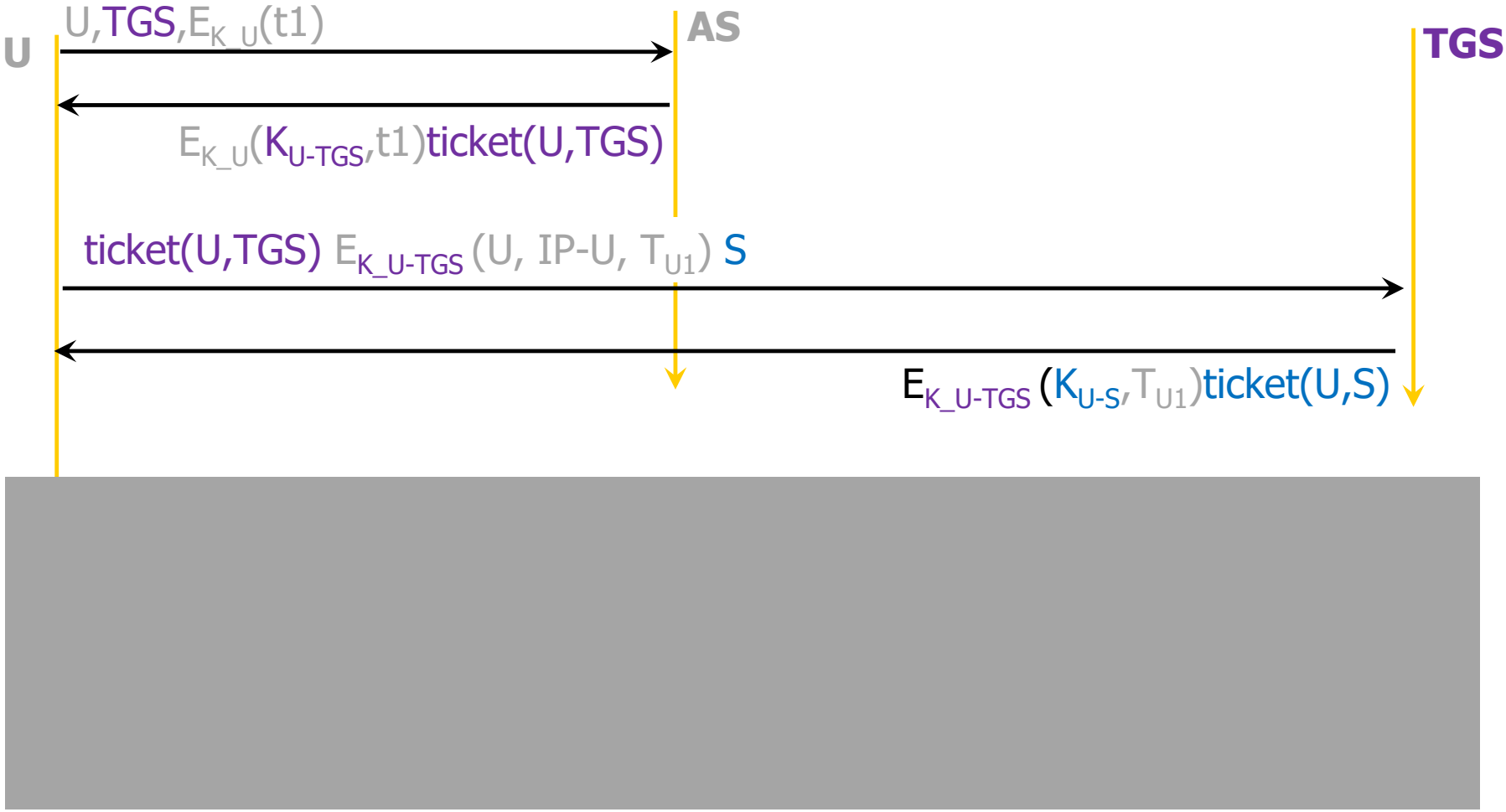
1. Either **K_U** is kept in memory or U has to retype password every few minutes
2. New traffic generated with **K_U** every few minutes (potentially useful for cryptoanalysis)
   - IF crypto algorithm is perfect THEN no problem (but you never know...)
   - **K_U** is a **weak** key (derived from a password): its protection is important

KDC = AS + TGS

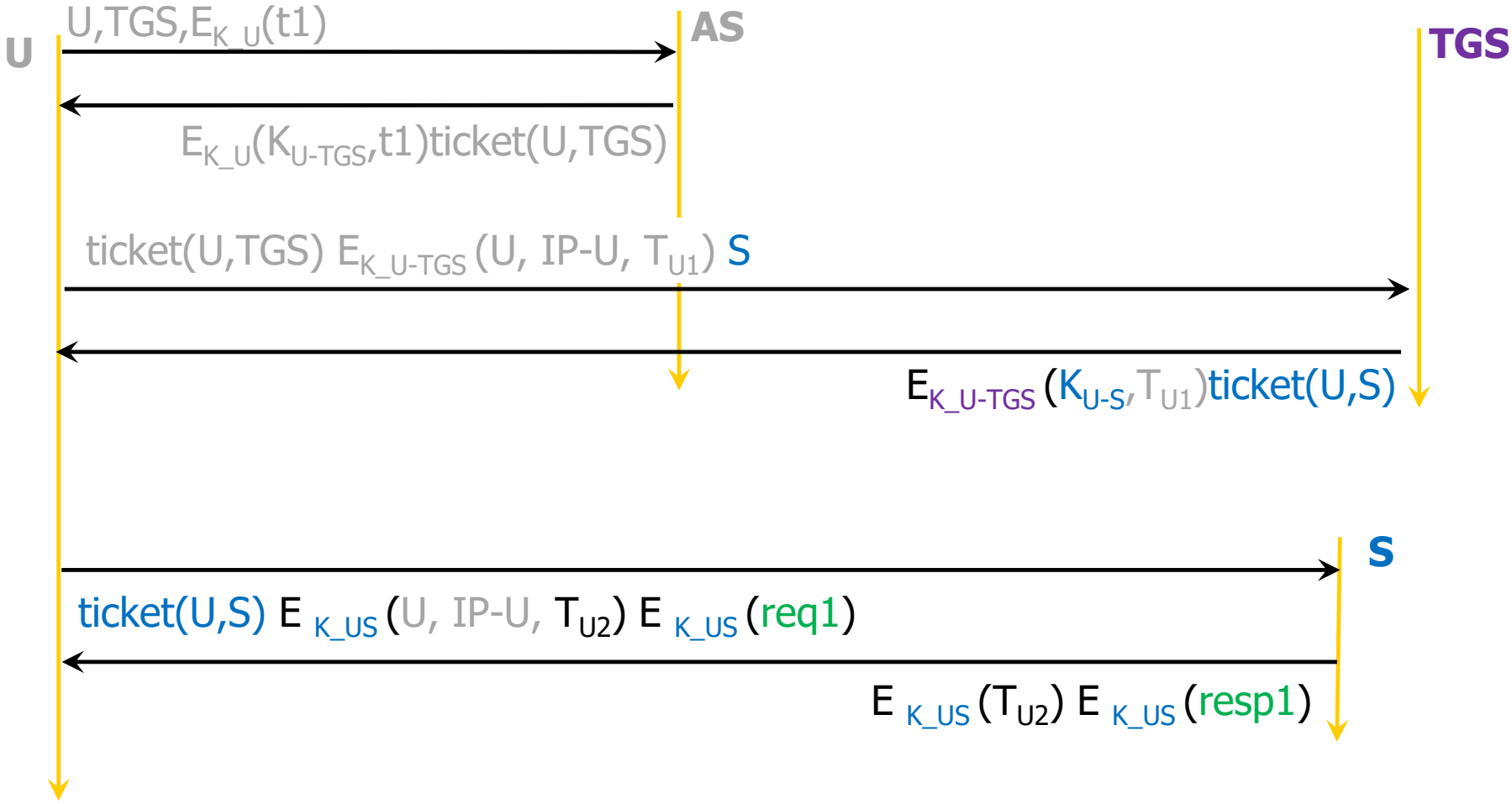


- KDC executes **2** services
- **Authentication** Service (AS):
  - Issues tickets **only** for TGS
- **Ticket Granting** Service (TGS):
  - Issues tickets for **other** services
- Associated with Service Account `krbtgt`

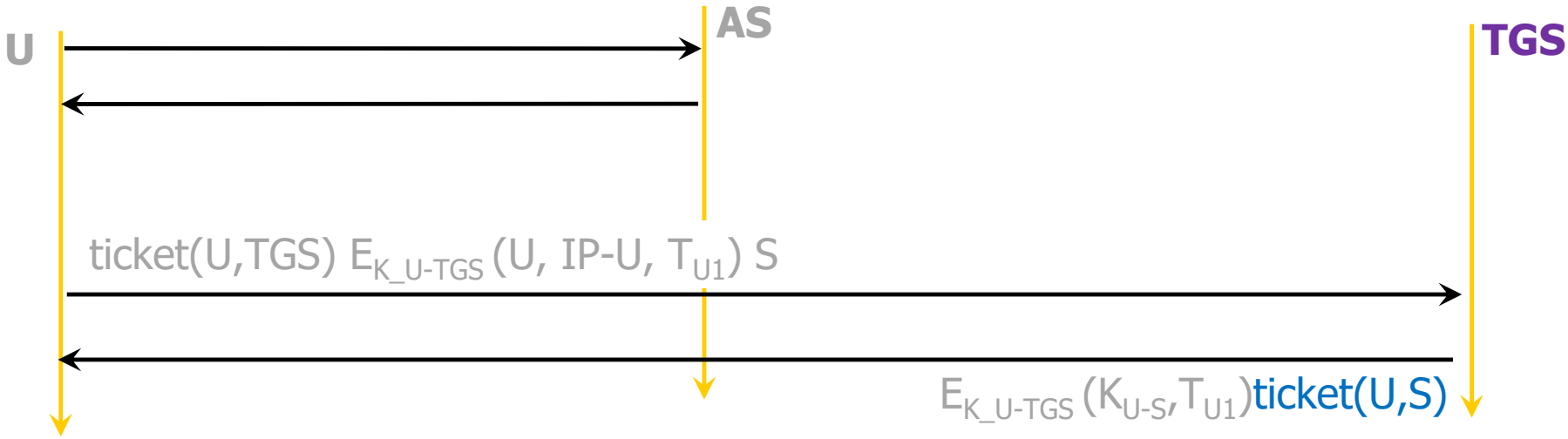
Kerberos (I)


U
AS
TGS
U,TGS,E_K_U(t1)
E_K_U(K_U-TGS,t1)ticket(U,TGS)
ticket(U,TGS) E_K_U-TGS (U, IP-U, T_U1) S
E_K_U-TGS (K_U-S,T_U1)ticket(U,S)

Kerberos (II)


U
AS
TGS
S
U,TGS,E_K_U(t1)
E_K_U(K_U-TGS,t1)ticket(U,TGS)
ticket(U,TGS) E_K_U-TGS (U, IP-U, T_U1) S
E_K_U-TGS (K_U-S,T_U1)ticket(U,S)
ticket(U,S) E K_US (U, IP-U, T_U2) E K_US (req1)
E K_US (T_U2) E K_US (resp1)

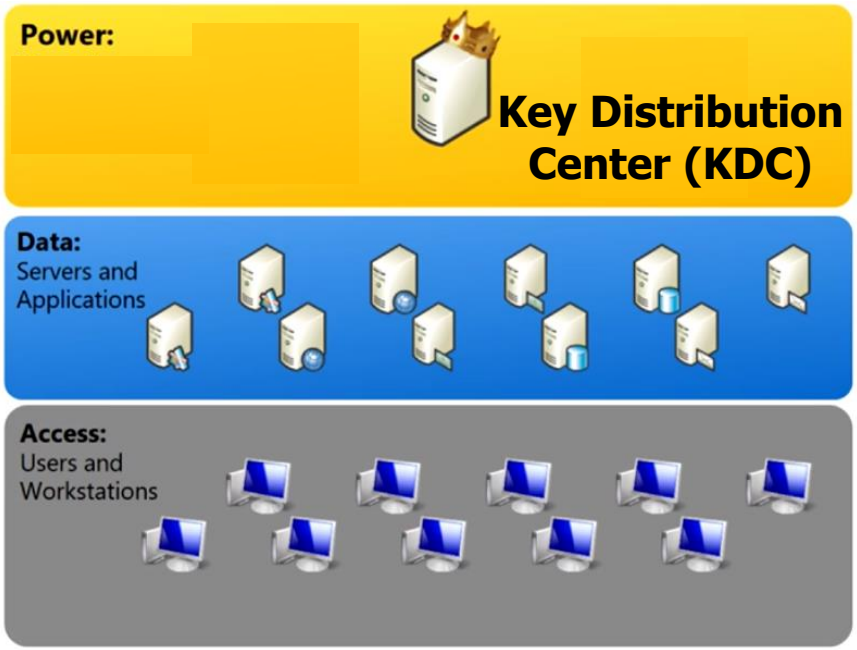
Hhhmm...



ticket(U,S) is encrypted in K_S

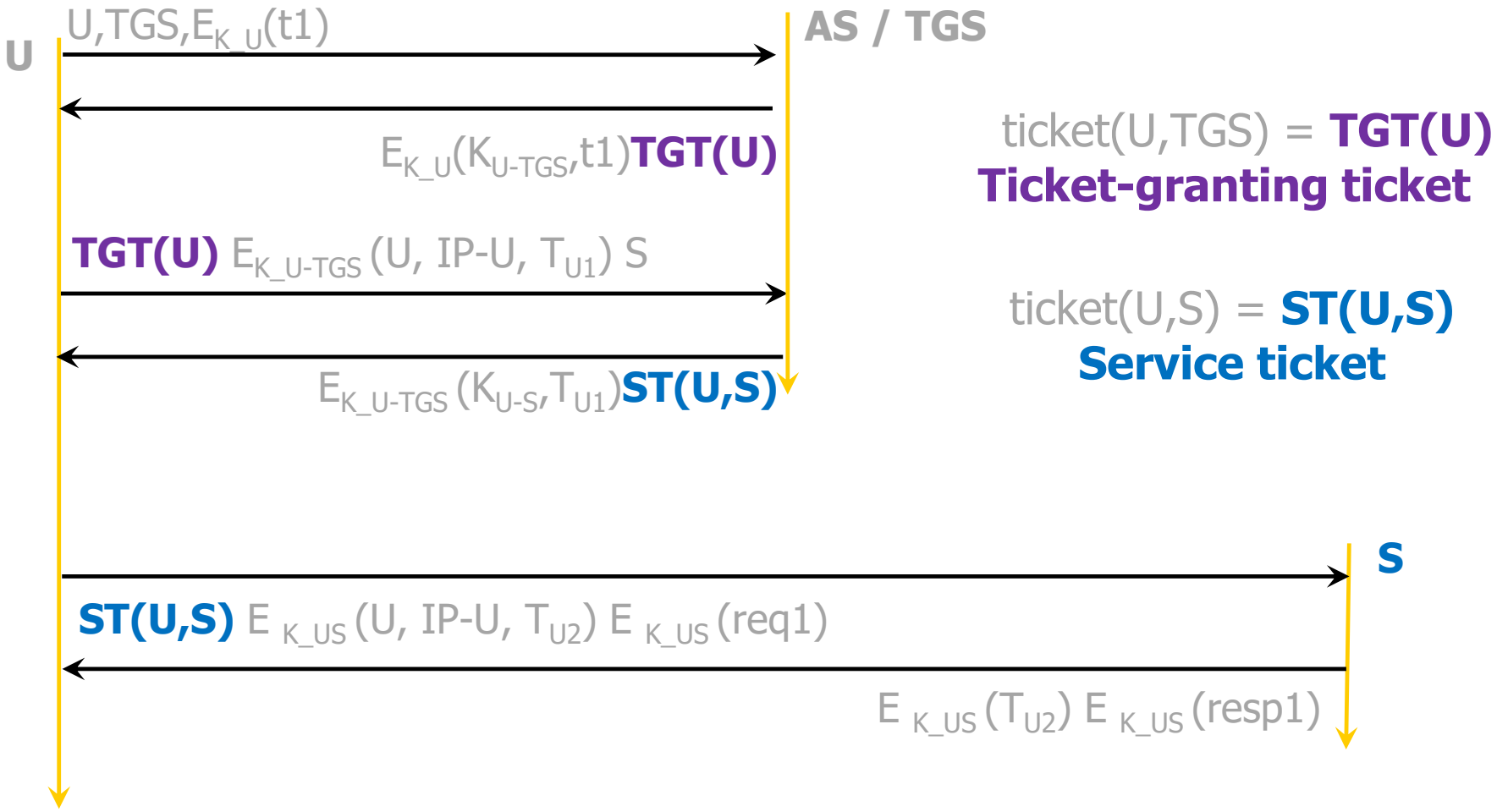
How can TGS know that key?

KDC = AS + TGS



- KDC executes **2** services
- Authentication Service (AS)
- Ticket Granting Service (TGS)
- AuthDB known to AS **and TGS**
- The separation between AS and TGS is **only a conceptual one**
- One single process (KDC) listening on one port (88)

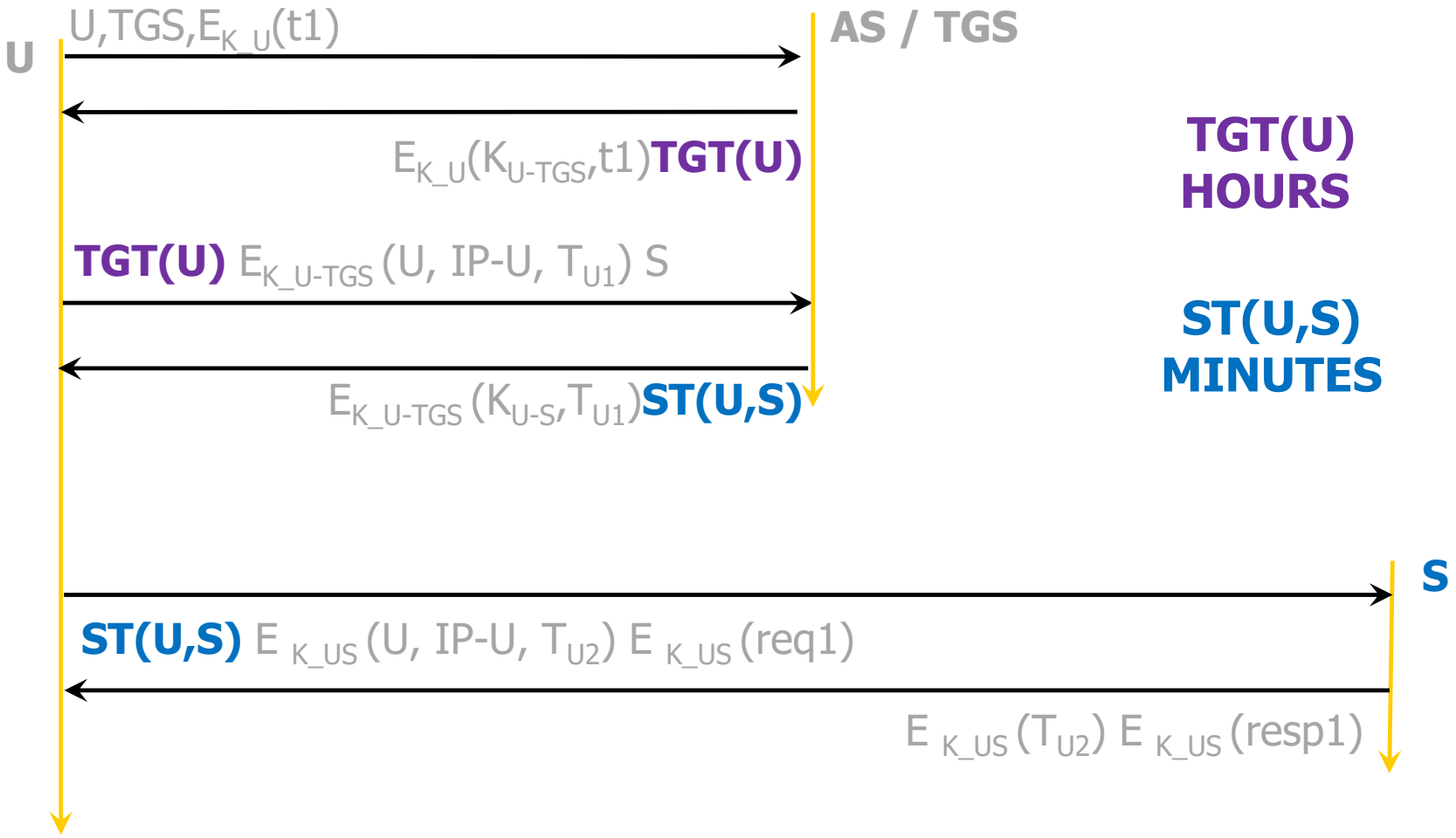
Practical Terminology: Tickets



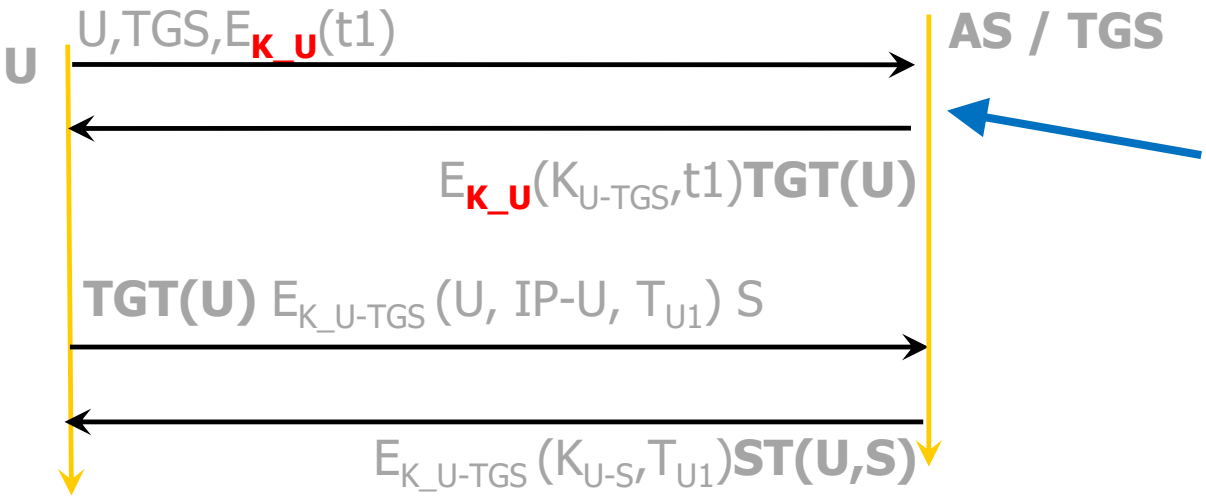
ticket(U,TGS) = **TGT(U)**
Ticket-granting ticket

ticket(U,S) = **ST(U,S)**
Service ticket

Ticket Lifetime



K_U: Problem solved!

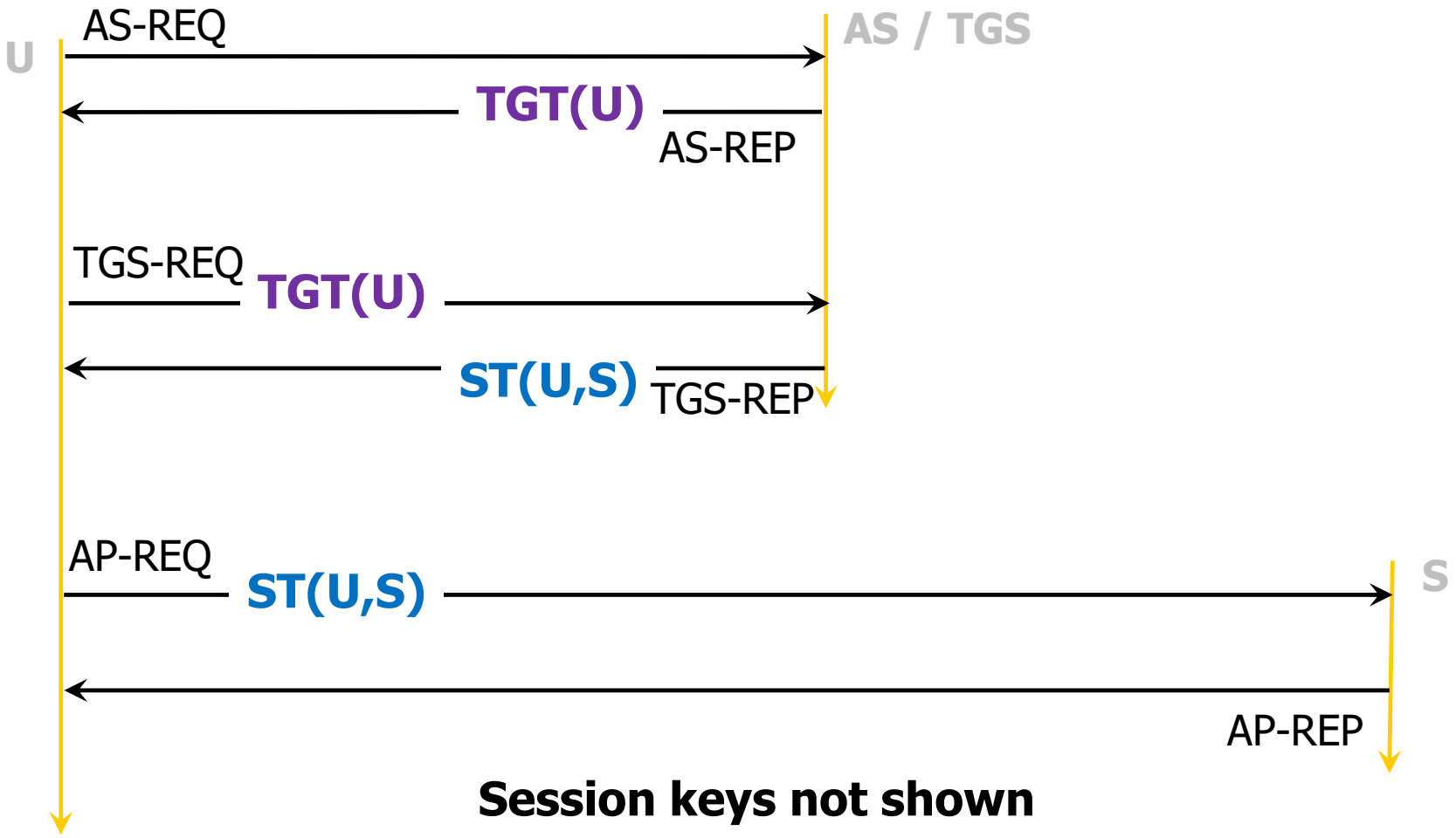


1. Logon
2. After few hours
3. After few hours
4. ...

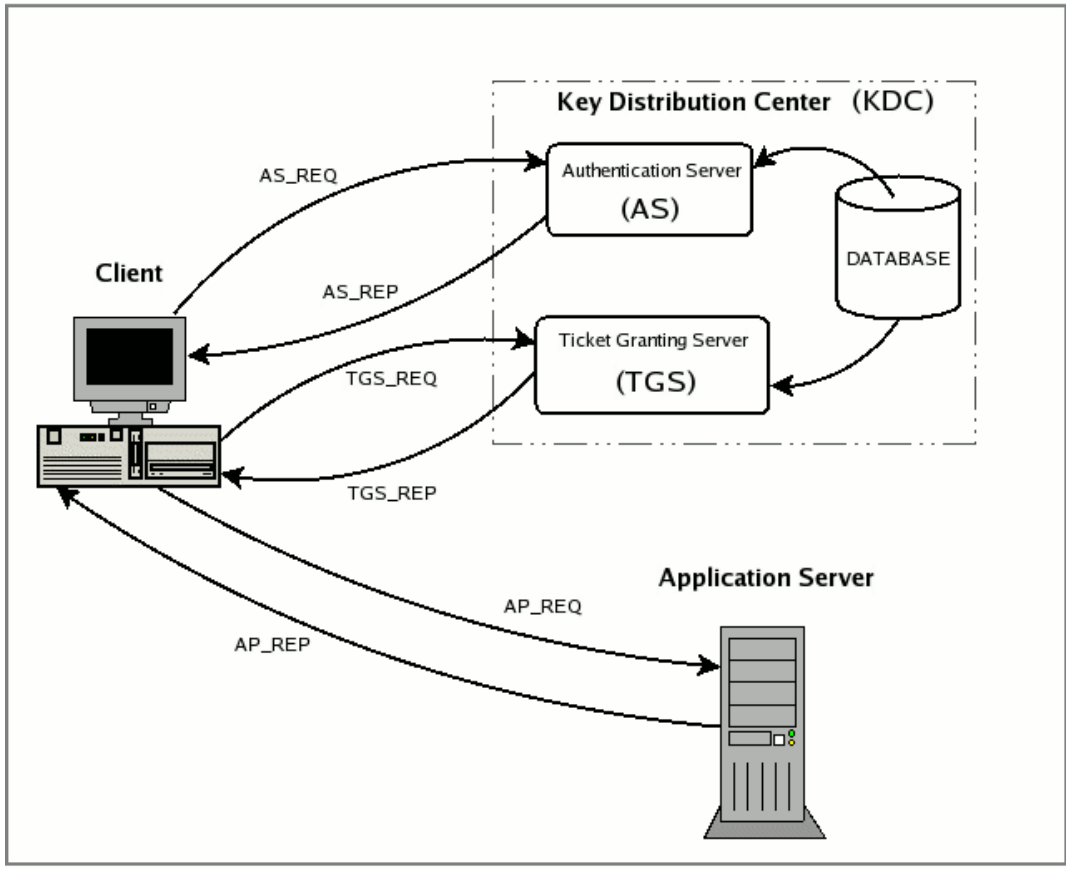
- TGT(U) lifetime in the order of **hours** ⇒ **K_U** reused every few **hours**

1. **K_U** need **not** be kept in memory
   (U may be required to retype password after a few hours)
2. **Very little traffic** generated with **K_U**

Practical Terminology: Messages



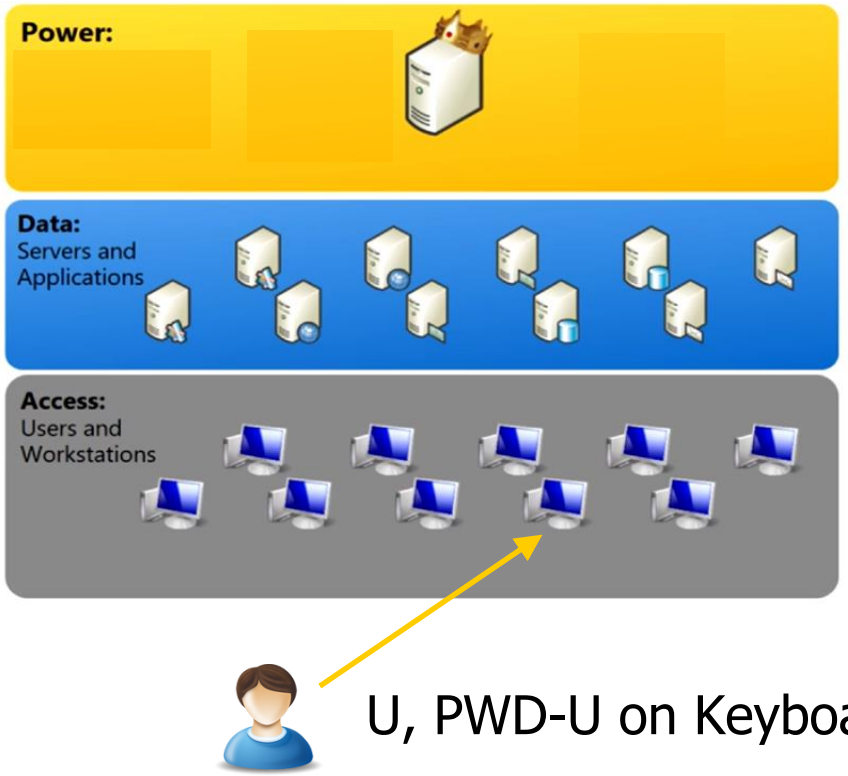
Another Drawing



Tickets / Session keys not shown

Windows Logon

Interactive Logon (REMIND)



Interactive logon of domain account U

Workstation needs to:

- **Authenticate** U
- Make sure U has **access rights** for interactive logon on that workstation

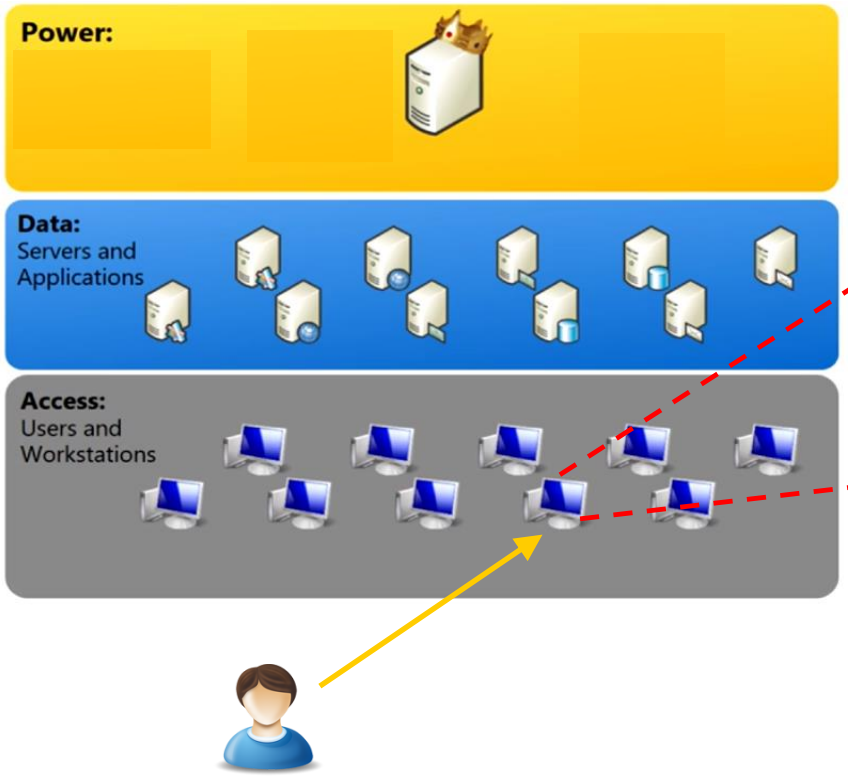
Interactive Logon (I)

U, PWD-U on Keyboard+Screen

Workstation **`wks_name`**:

- Computes H(PWD-U)
- IF U is local account
- THEN Search <U, H(PWD-U)> in SAM
  ...
- ELSE Acquire TGT(U) // K_U = H(PWD-U) is needed
  Acquire ST(U, **`wks_name`**)
  Extract PAC from ST
  Verify that U has access rights on **`wks_name`**

Interactive Logon (II)



Workstation memory:

U, H(PWD-U)

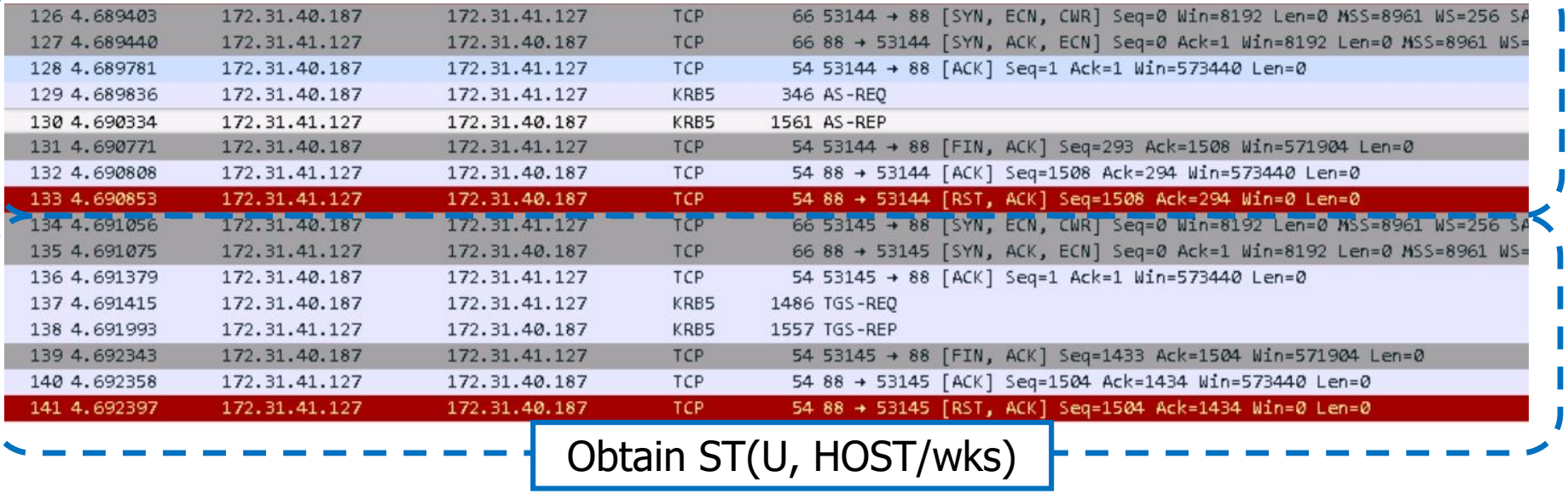
TGT(U) + $K_{U\text{-}TGS}$

ST(U, wks_name) + $K_{U\text{-}H}$

Data of other accounts omitted
(e.g., wks_name account)

Network Capture Example

Obtain TGT(U)

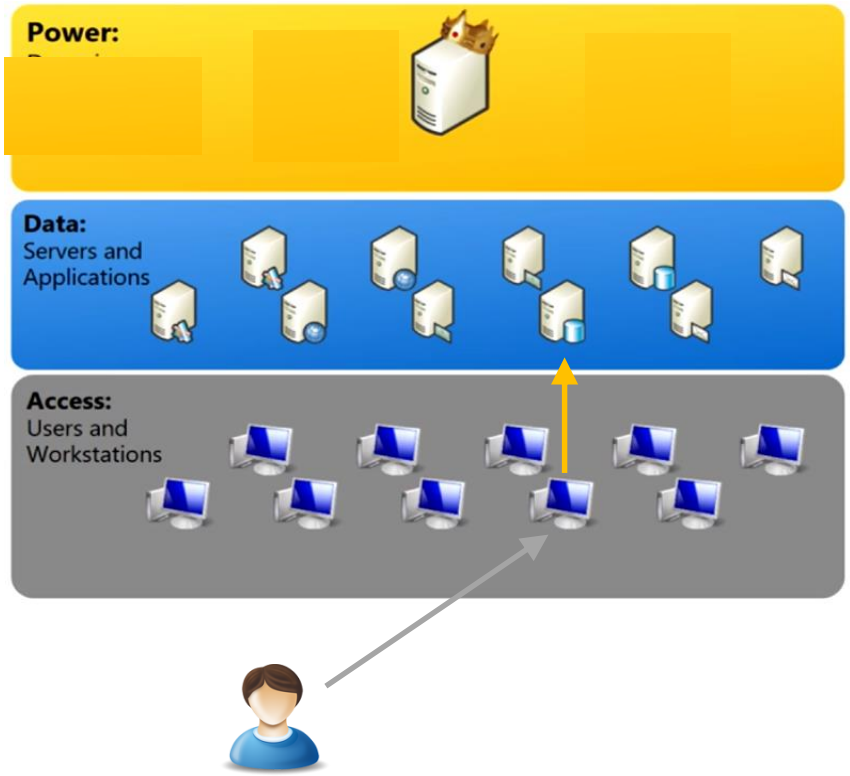


126 4.689403	172.31.40.187	172.31.41.127	TCP	66	53144 → 88 [SYN, ECN, CWR] Seq=0 Win=8192 Len=0 MSS=8961 WS=256 SA
127 4.689440	172.31.41.127	172.31.40.187	TCP	66	88 → 53144 [SYN, ACK, ECN] Seq=0 Ack=1 Win=8192 Len=0 MSS=8961 WS=
128 4.689781	172.31.40.187	172.31.41.127	TCP	54	53144 → 88 [ACK] Seq=1 Ack=1 Win=573440 Len=0
129 4.689836	172.31.40.187	172.31.41.127	KRB5	346	AS-REQ
130 4.690334	172.31.41.127	172.31.40.187	KRB5	1561	AS-REP
131 4.690771	172.31.40.187	172.31.41.127	TCP	54	53144 → 88 [FIN, ACK] Seq=293 Ack=1508 Win=571904 Len=0
132 4.690808	172.31.41.127	172.31.40.187	TCP	54	88 → 53144 [ACK] Seq=1508 Ack=294 Win=573440 Len=0
133 4.690853	172.31.41.127	172.31.40.187	TCP	54	88 → 53144 [RST, ACK] Seq=1508 Ack=294 Win=0 Len=0
134 4.691056	172.31.40.187	172.31.41.127	TCP	66	53145 → 88 [SYN, ECN, CWR] Seq=0 Win=8192 Len=0 MSS=8961 WS=256 SA
135 4.691075	172.31.41.127	172.31.40.187	TCP	66	88 → 53145 [SYN, ACK, ECN] Seq=0 Ack=1 Win=8192 Len=0 MSS=8961 WS=
136 4.691379	172.31.40.187	172.31.41.127	TCP	54	53145 → 88 [ACK] Seq=1 Ack=1 Win=573440 Len=0
137 4.691415	172.31.40.187	172.31.41.127	KRB5	1486	TGS-REQ
138 4.691993	172.31.41.127	172.31.40.187	KRB5	1557	TGS-REP
139 4.692343	172.31.40.187	172.31.41.127	TCP	54	53145 → 88 [FIN, ACK] Seq=1433 Ack=1504 Win=571904 Len=0
140 4.692358	172.31.41.127	172.31.40.187	TCP	54	88 → 53145 [ACK] Seq=1504 Ack=1434 Win=573440 Len=0
141 4.692397	172.31.41.127	172.31.40.187	TCP	54	88 → 53145 [RST, ACK] Seq=1504 Ack=1434 Win=0 Len=0

Obtain ST(U, HOST/wks)

- KDC on 172.31.41.127
- AS and TGS on port number 88

Network Logon (REMIND)



U is in interactive logon on workstation

Network logon on S

S needs to:

- **Authenticate** U
- Make sure U has **access rights** for network logon on S

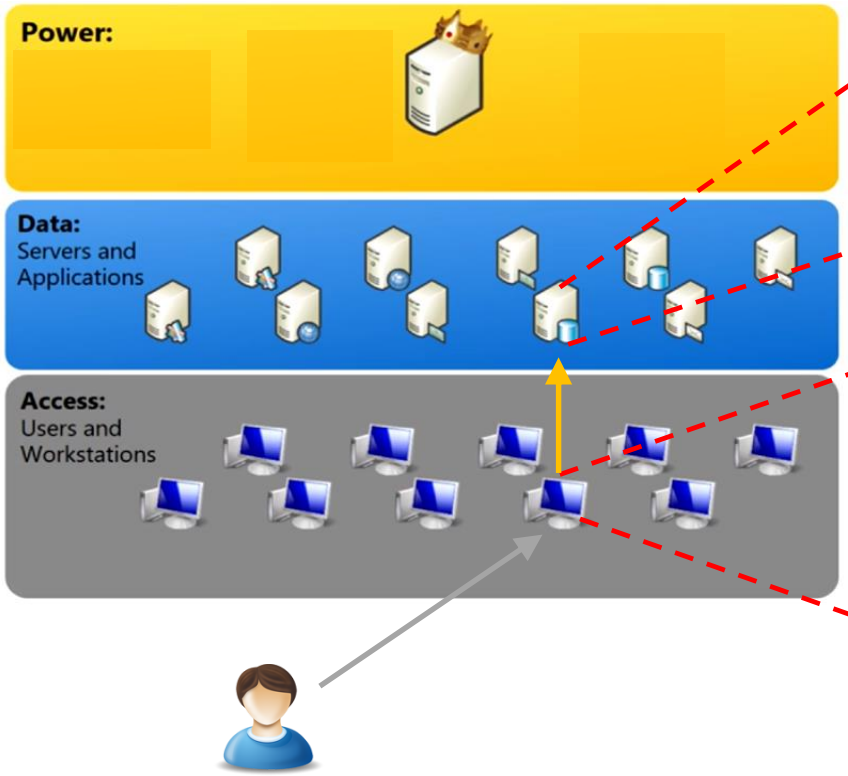
Network Logon (I)

Workstation memory:
U, H(PWD-U)
TGT(U) + $K_{U\text{-}TGS}$
ST(U, wks_name) + $K_{U\text{-}H}$

Workstation:

1. Acquire ST(U, S) // TGT(U) and $K_{U\text{-}TGS}$ needed
2. Store ST(U,S) + $K_{U\text{-}S}$
3. Connect to S (that will check access rights of U)

Network Logon (II)



Server memory:

…

Nothing related to U

Workstation memory:

U, H(PWD-U)

TGT(U) + K_{U-TGS}

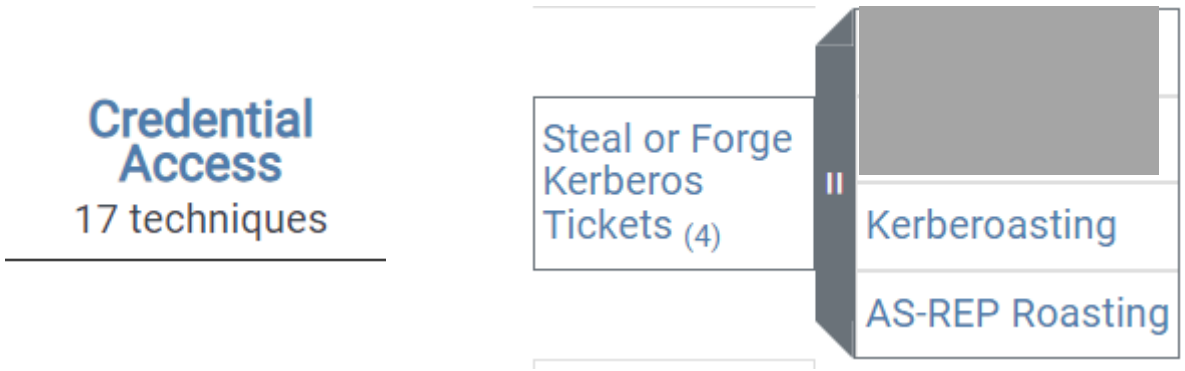
ST(U, wks_name) + K_{U-H}

ST(U, S) + K_{U-S}

Data of other accounts omitted
(e.g., computer accounts)

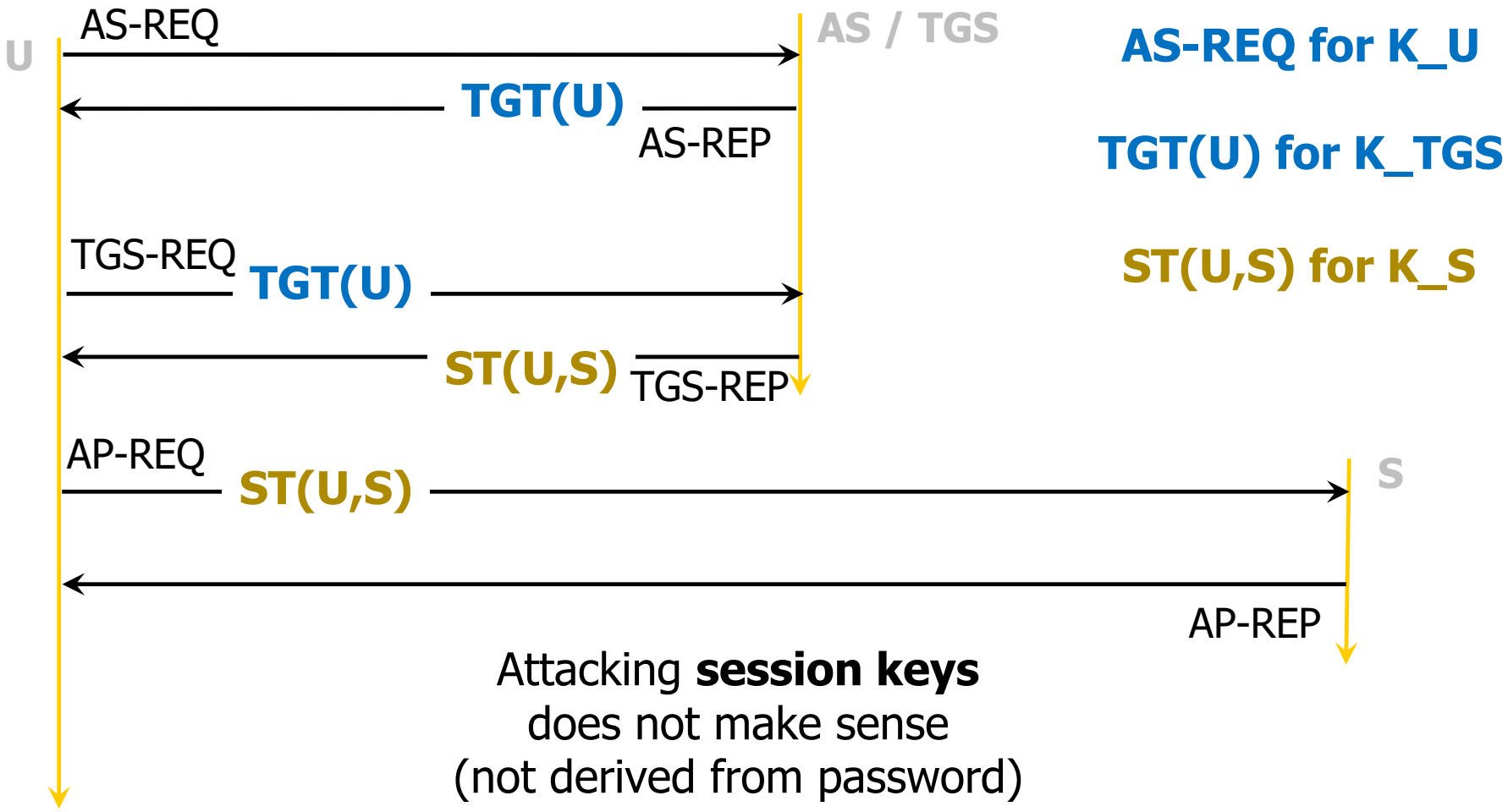
Kerberos: Password Cracking Attacks

Kerberos: Password Cracking Attacks



- Adversaries may **abuse a valid Kerberos TGT** or **sniff network traffic** to obtain a ticket that may be **vulnerable to Brute Force (Password Cracking)**.

Sniff Network Traffic



Attacking **session keys**
does not make sense
(not derived from password)

Password Cracking

- Kerberos 5, etype 23, AS-REQ Pre-Auth
- Kerberos 5, etype 23, TGS-REP
- Kerberos 5, etype 23, AS-REP
- Kerberos 5, etype 17, TGS-REP
- Kerberos 5, etype 18, TGS-REP

HashCat / John the Ripper

- foreach **x** ∈ **lookup-table**
  - IF Decrypt_**x**(**GUESSING-MATERIAL**) ok THEN **x** is ok

- Remember than you can always tell whether you decrypted with the correct key
- Windows Kerberos does not salt password hash

Abuse Valid TGT

- **Threat Model**: Attacker has credentials of **one** user U
- **Extremely realistic**
  (and **much easier** to obtain than Network Sniffing)

1. Send LDAP queries for collecting info
2. Collect ST of:
   1. Certain services (**Kerberoasting**)
   2. Certain users (**AS-REP Roasting**)
3. Offline guessing on their password hashes

Kerberoasting

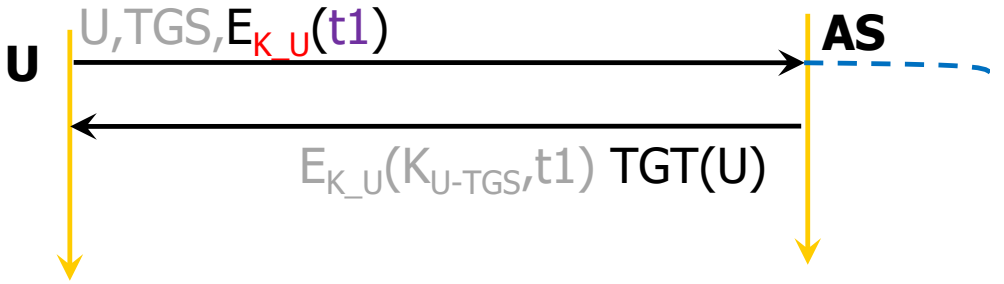
- Determine set of SPN **installed long time ago**
  - Lists of commons SPNs
  - Lists of useful LDAP queries

1. Ask ST(U,S) of every S
2. Offline guessing on H(PWD-S)

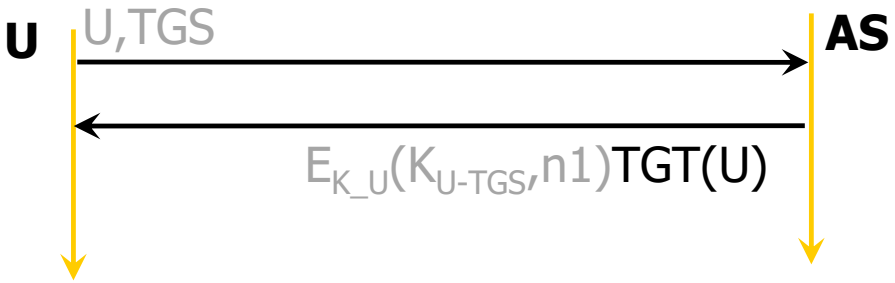
- Effective: Probably weak password
- Dangerous: **Overprivilege** of service accounts is very frequent

Pre-authentication

- AS responds only upon receiving a **proof of knowledge** of K_U



- Some users may be configured "logon **without pre-authentication**"



Can Ask TGT(U) of any such user!

AS-REP Roasting

- Determine set of users with "Logon NO PRE-AUTH"

1. Ask TGT(UX) of every UX
2. Offline guessing on H(PWD-UX)

- Effective: Probably weak password

Attack Optimization

- Default Kerberos configuration
  - AES-encrypted tickets by default
  - RC4-encrypted tickets can be requested

- RC4 is 800 times faster than AES
- Ask RC4-encrypted tickets and 800 times faster guesses!
  - Imagine **completing a 40 hour work week in 3 minutes**

Keep in mind

- Threat Model: Attacker has credentials of one user U
- **Any domain user** can **easily** attempt to crack passwords of many accounts
  - High likelihood of success
  - High likelihood of strong impact
- Practically relevant risk

Detection

Data Source	Data Component
Active Directory	Active Directory Credential Request

Monitor for anomalous activity, such as enabling Audit Kerberos Service Ticket Operations to log Kerberos TGS service ticket requests. Particularly investigate irregular patterns of activity (ex: accounts making numerous requests, Event ID 4768 and 4769, within a small time frame, especially if they also request RC4 encryption [Type 0x17], pre-authentication not required [Type: 0x0]).