MFA: Multifactor Authentication

Fact

- **"Passwords alone are no longer sufficient"**

- Authentication of human operators can no longer be based solely on passwords
- Every guide / best practice / post-incident analysis states: **"Enforce multi-factor authentication (MFA)"**

2FA vs MFA

- **2FA** Two-Factor authentication
  - Something you **know** (password)
  - Something you **have** (smartphone or security key)

- search "`google titan`"
- search "`yubico security key`"

- We will consider 2FA and **MFA** synonyms

Second factors

- **Smartphone**
  - OTP SMS
  - OTP Authenticator App
  - Push notifications
- **SecurityKey** (USB/NFC/Bluetooth)

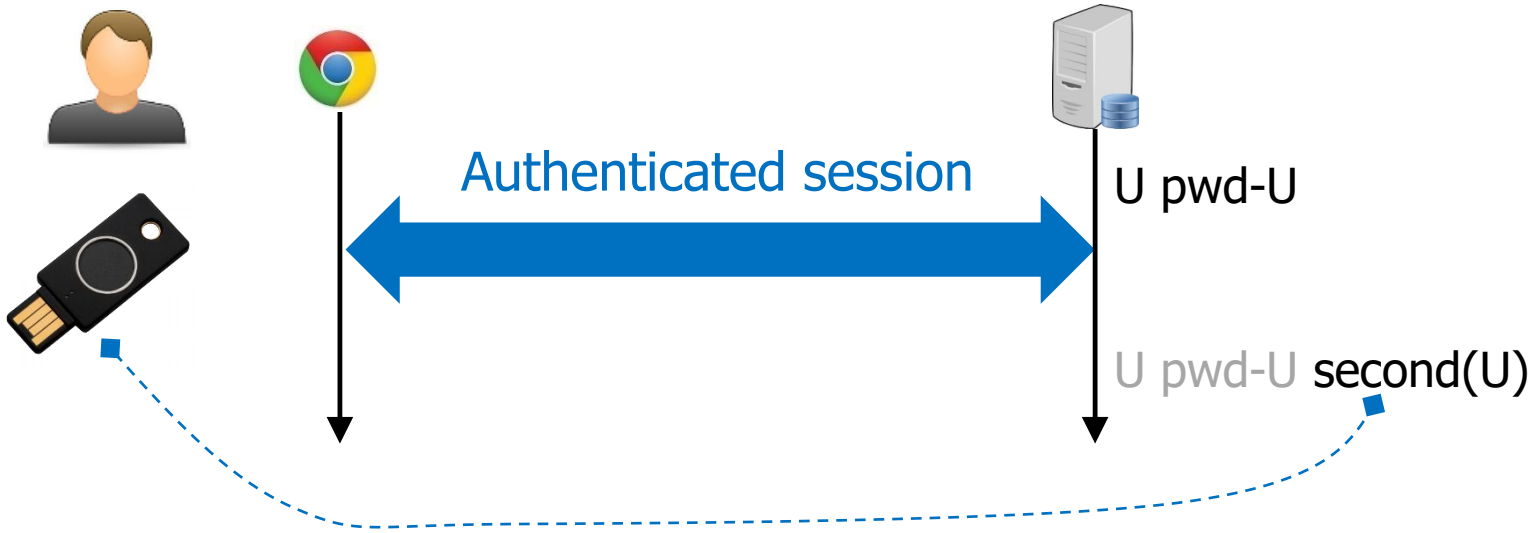
- SecurityKey **much more secure** than other methods
- **All enormously** more secure than password only

Our Focus

- **Web app** (BASIC/FORM over HTTPS)
- **Different** organizations

- Extremely relevant in practice

Step 1: Linking 2nd factor (once)



- User selects 2FA in **account options** (Service must support 2FA)
- Message exchange depends on Service and 2FA technique

Linking Example (I): Google

Turn on 2-Step Verification

With 2-Step Verification (also known as two-factor authentication), you add an extra layer of security to your account. After you set it up, you'll sign in to your account in two steps using:

- Something you know (your password)
- Something you have (like your phone or a security key dongle)

Computer | Android | iPhone & iPad

Step 1: Set up 2-Step Verification

1. Go to your Google Account .
2. On the left navigation panel, click **Security**.
3. On the *Signing in to Google* panel, click **2-Step Verification**.
4. Click **Get started**.
5. Follow the steps on the screen.

Linking Example (II): Lastpass

Enable Multifactor Authentication

As a LastPass user, you can enable Multifactor Authentication for your account as follows:

1. Log in to LastPass and access your Vault by doing either of the following:
   - Go to https://lastpass.com/?ac=1 and log in with your username and Master Password.
   - In your web browser toolbar, click the LastPass icon then click **Open My Vault**.
2. Select **Account Settings** in the left navigation.
3. Click on the **Multifactor Options** tab.
4. Click the Edit icon to the right of your desired multifactor option.

Linking Example (III): Facebook

Cos'è l'autenticazione a due fattori e come funziona?

Assistenza per computer Assistenza per mobile Condividi l'articolo

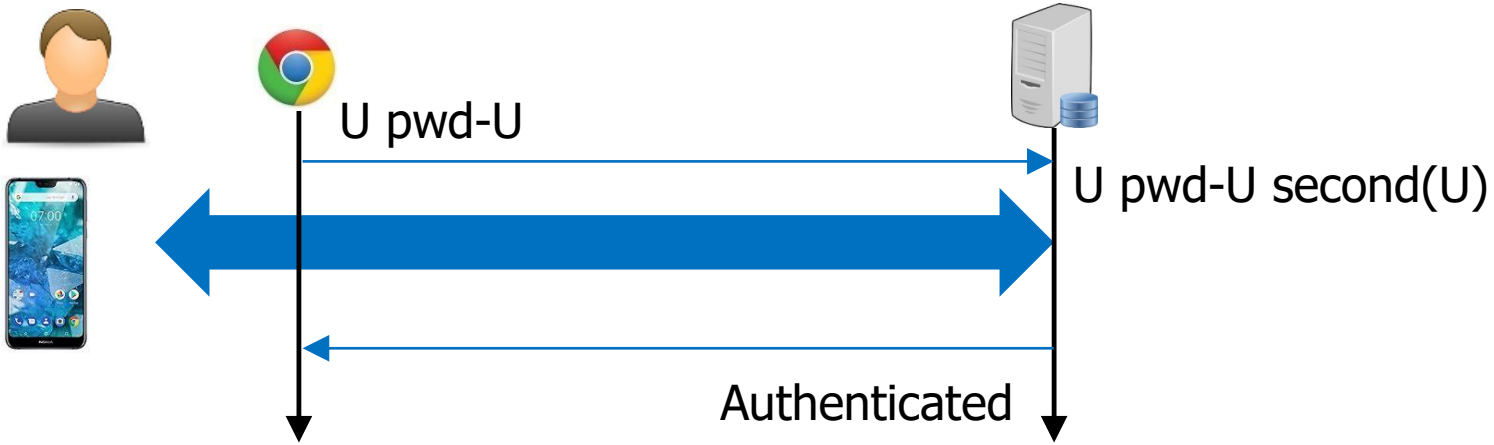
Se stai riscontrando problemi nell'accedere al tuo account Facebook, consulta prima questi suggerimenti.

Aiutandoti a proteggere il tuo account Facebook, l'autenticazione a due fattori è una funzione di protezione ulteriore rispetto alla password. Se attivi l'autenticazione a due fattori, ti verrà richiesto di inserire un codice di accesso speciale o di confermare il tentativo di accesso ogni volta che qualcuno prova ad accedere a Facebook con il tuo account da un computer o dispositivo mobile non riconosciuto. Puoi anche ricevere avvisi quando qualcuno prova a effettuare l'accesso con il tuo account da un computer non riconosciuto.

Per attivare o gestire l'autenticazione a due fattori:

1. Accedi alle impostazioni di Protezione e accesso cliccando su nell'angolo in alto a destra di Facebook, quindi su **Impostazioni** > **Protezione e accesso**.
2. Scorri fino a **Usa l'autenticazione a due fattori** e clicca su **Modifica**.

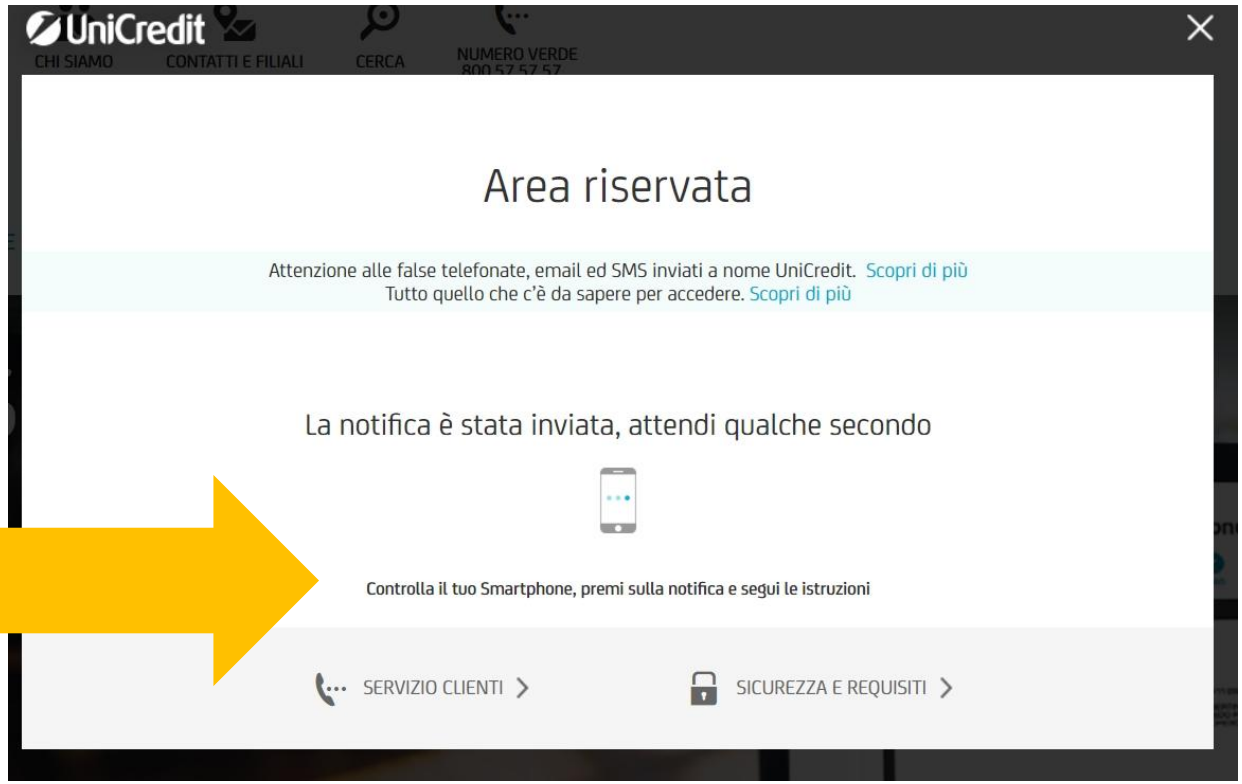
Step 2: Login (every time)



- Message exchange depends on Service and 2FA technique

Login Example (I): Unicredit

Push Notification



Login Example (II): SPID-PosteID

Push Notification

spid

Richiesta di accesso SPID 2 da

Units

Per accedere è necessaria un'ulteriore verifica (livello 2 di sicurezza SPID)

Accedi con App PosteID

Voglio ricevere una notifica sull'App PosteID

Preferisco generare un PIN temporaneo con l'App PosteID

OTP AuthenticatorApp

Verifica di avere l'ultima versione dell'App

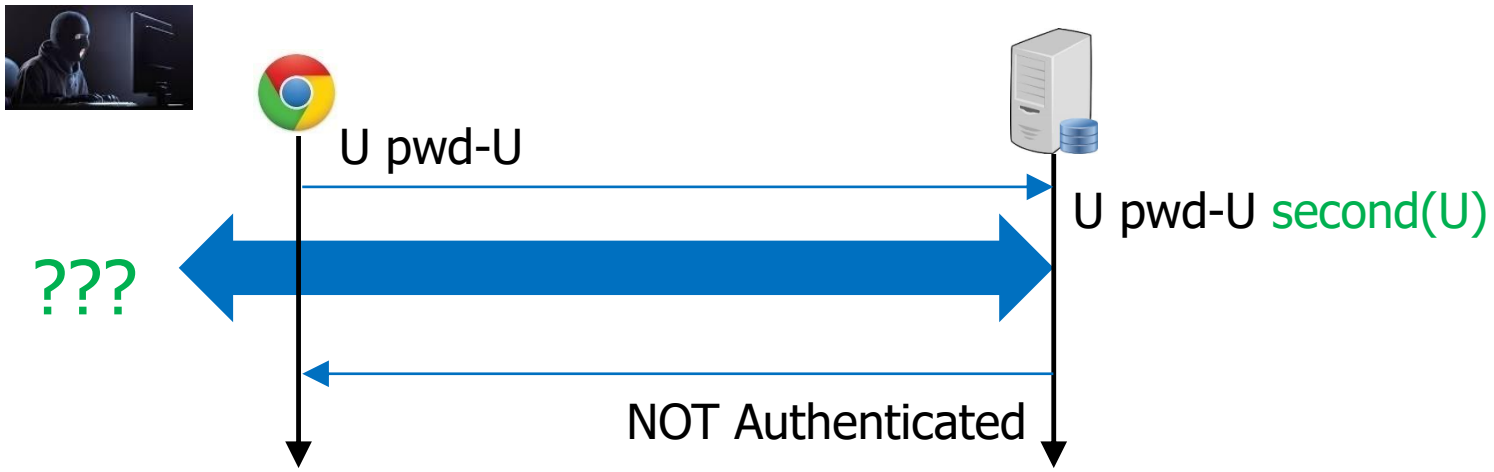
ANNULLA

Non puoi usare l'App PosteID? Accedi tramite codice SMS

OTP SMS

Threat Model: Stolen Password (I)

- Adversary has <U,P>



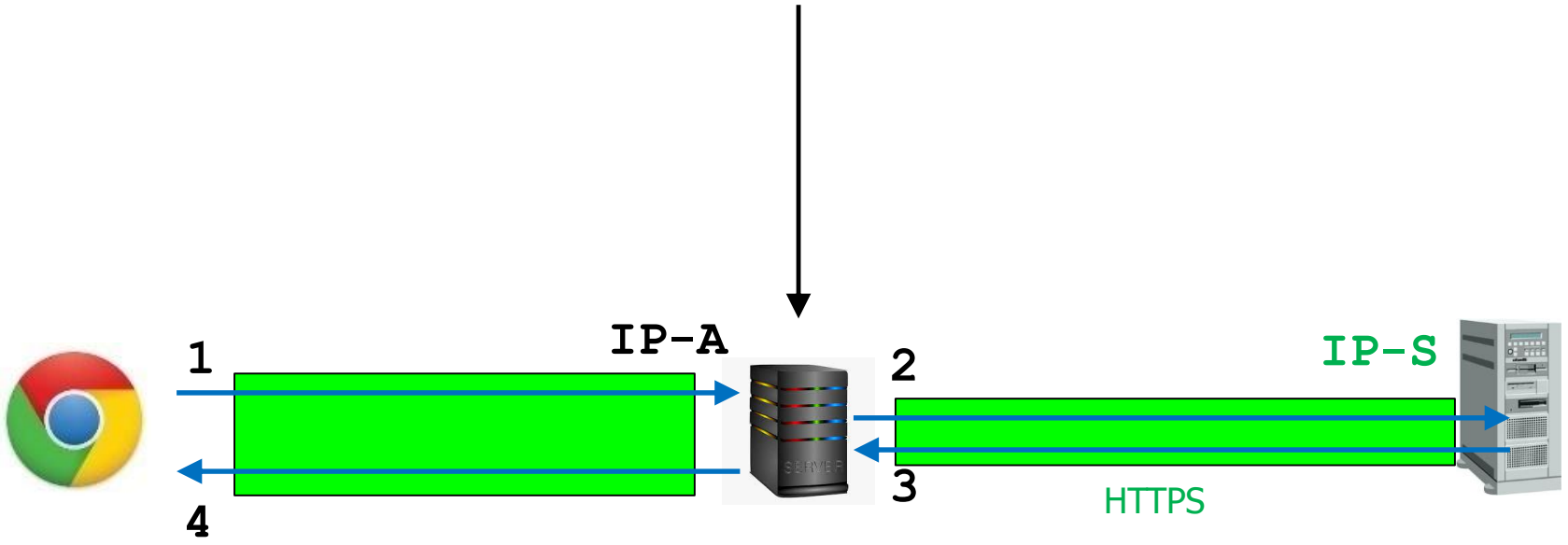
Threat Model: Stolen Password (II)

- **Smartphone**
  - OTP SMS — Solved
  - OTP Authenticator App — Solved
  - Push notifications — Solved
- **SecurityKey** (USB/NFC/Bluetooth) — Solved

Evil Proxy

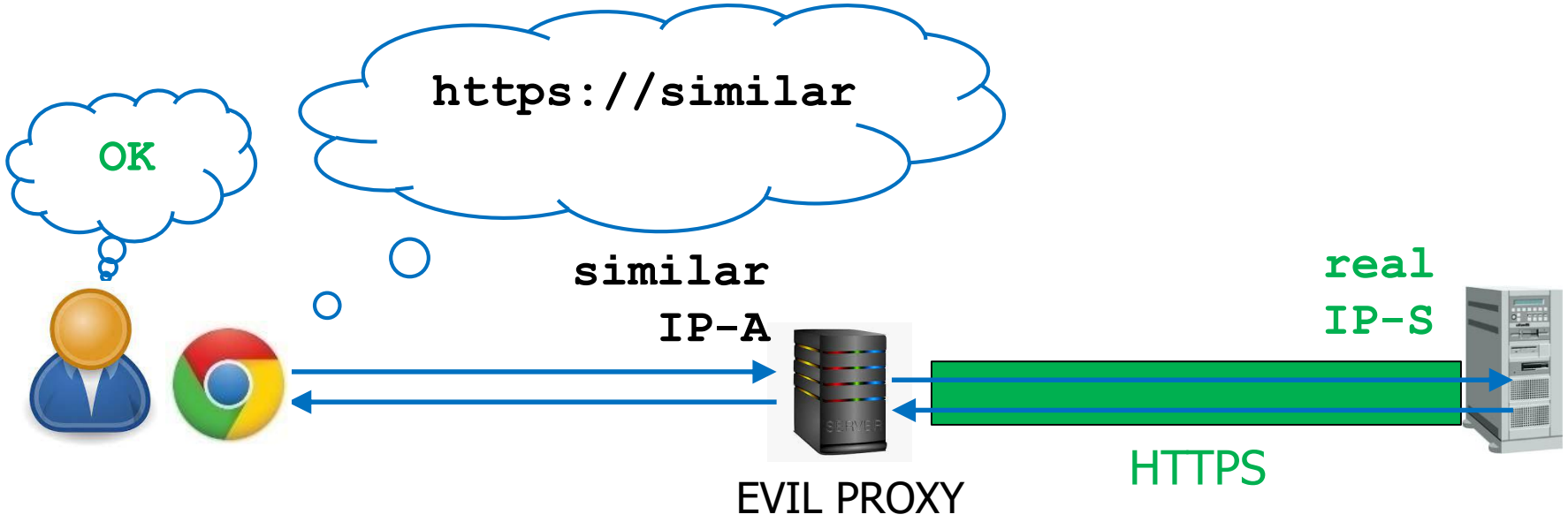
Proxy **specialized** for **AitM**

- Presents to C **all** resources of target website **without any local copy**
- Can target **many different** websites at the same time
- **Configuration** specifies what to **modify** and what to **log**



Threat Model: "Real-time Phishing" (I)

- User does **not** detect that is accessing the **wrong** URL



Threat Model: "Real-time Phishing" (II)

- **Smartphone**
  - OTP SMS — Not Solved
  - OTP Authenticator App — Not Solved
  - Push notifications — Not Solved
- **SecurityKey** (USB/NFC/Bluetooth) — Solved

Keep in mind

MFA is extremely important

- Very effective for very realistic threat model
- Enabling 2FA is a very high priority defensive investment

MFA is essential … Use of anything beyond the password significantly increases the costs for attackers, which is why **the rate of compromise of accounts using any type of MFA is less than 0.1% of the general population**.

Alex Weinert, Director of Identity Security Microsoft
November 2020

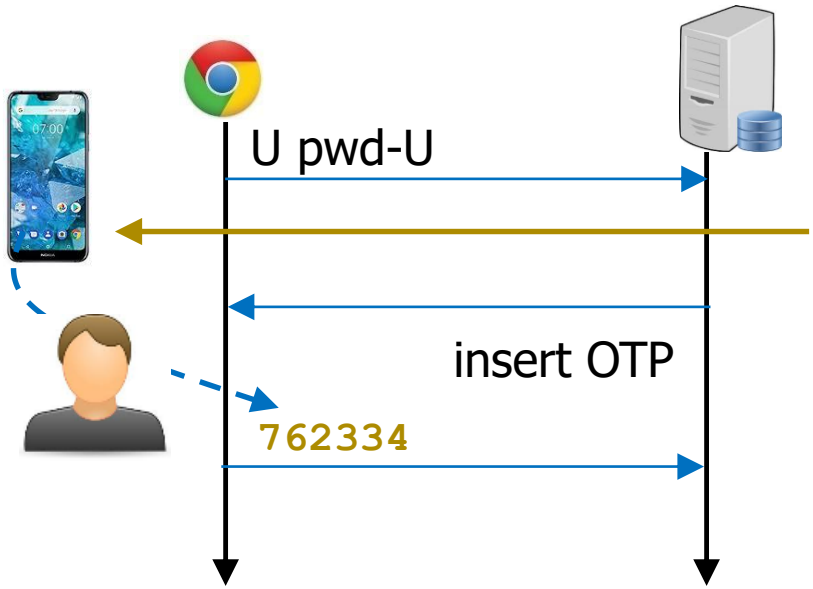
One-Time Passwords (OTP)

Second factor **(REMIND)**

- **Smartphone**
  - OTP SMS
  - OTP Authenticator App
  - Push notifications
- **SecurityKey** (USB/NFC/Bluetooth)

- SecurityKey **much more secure** than other methods
- **All enormously** more secure than password only

OTP SMS



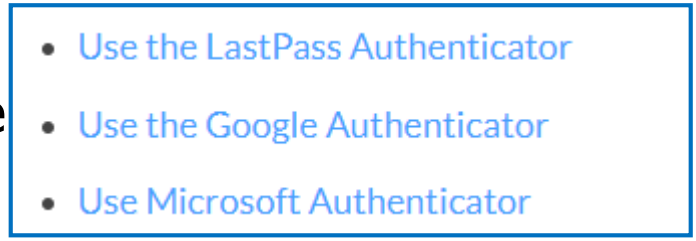
One-Time Password (**OTP**):

- 6-digit code with uniform distribution in [0,999999]
- Valid only for **this** login attempt

OTP AuthApp: Linking

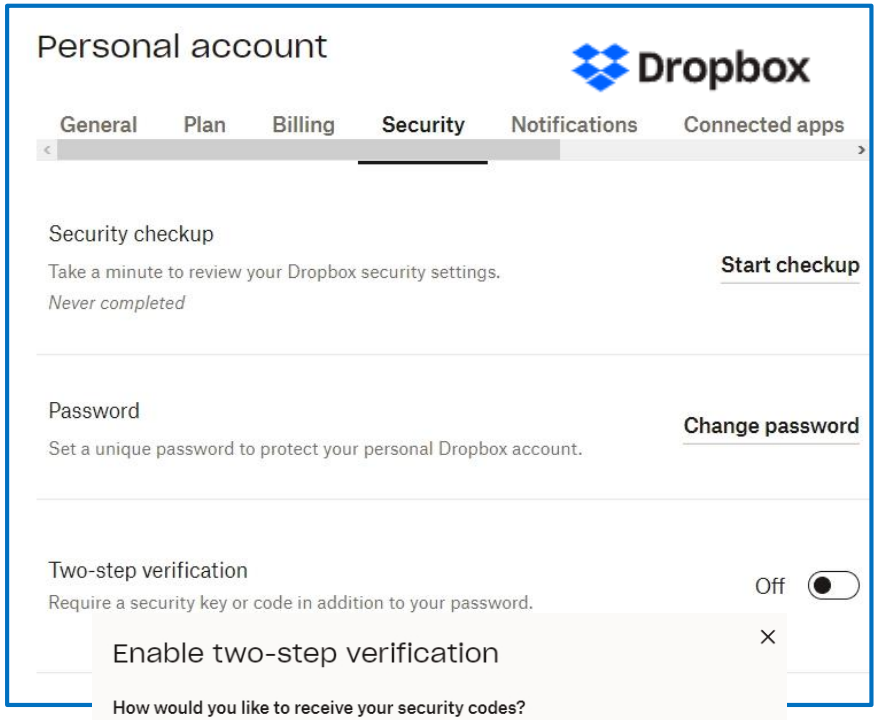
- **Generic** "Authenticator App" installed on User smartphone

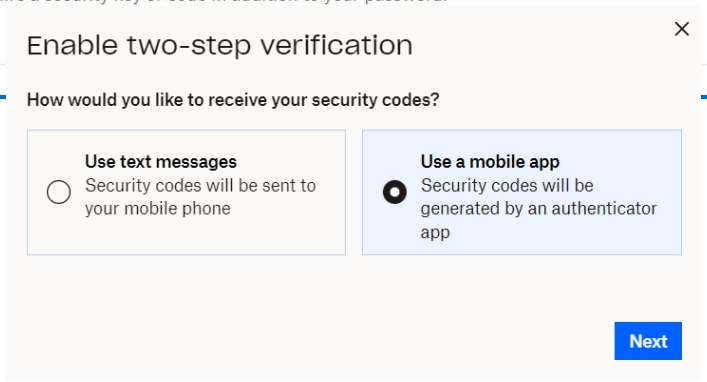
- User:
  1. Enable 2FA
  2. Choose AuthApp and Install on smartphone (if not installed already)

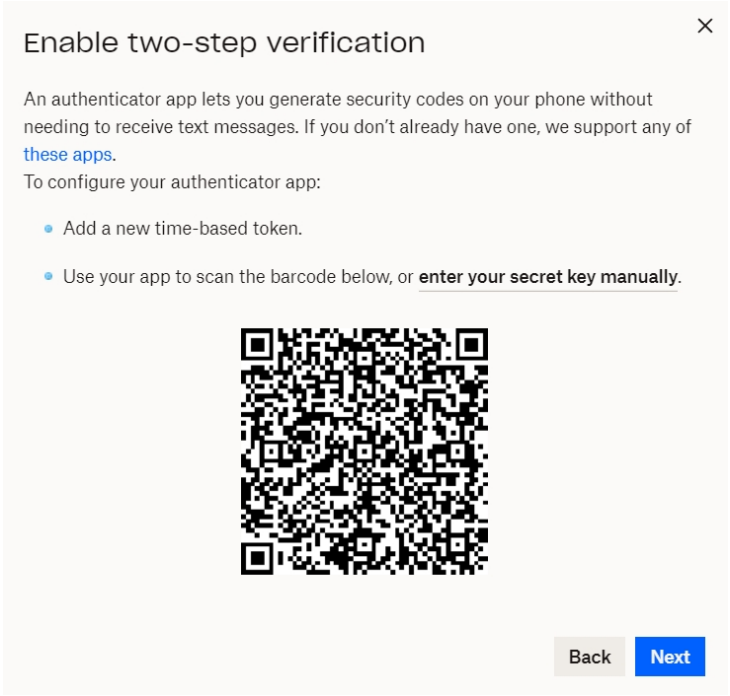


  3. Link account at Service with AuthApp on smartphone

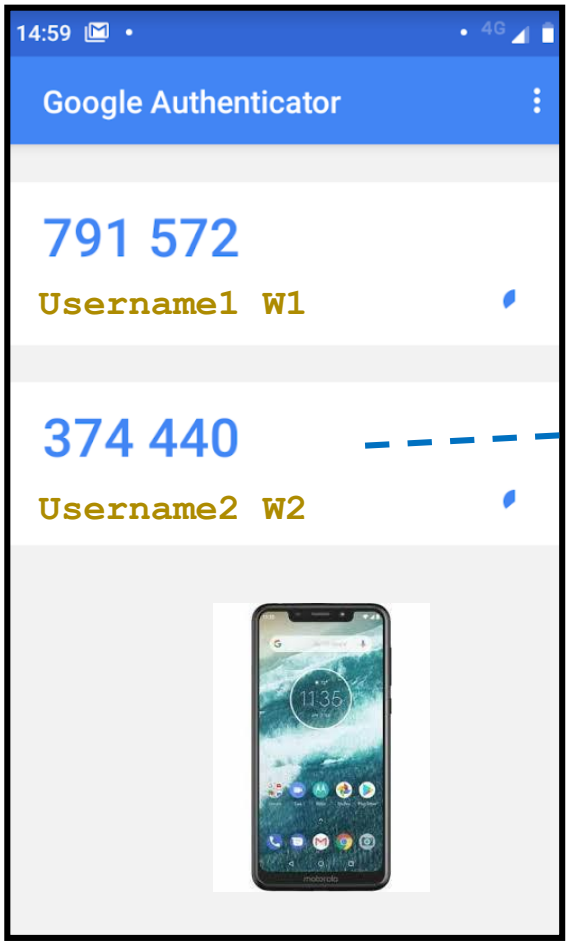
Example (outline)

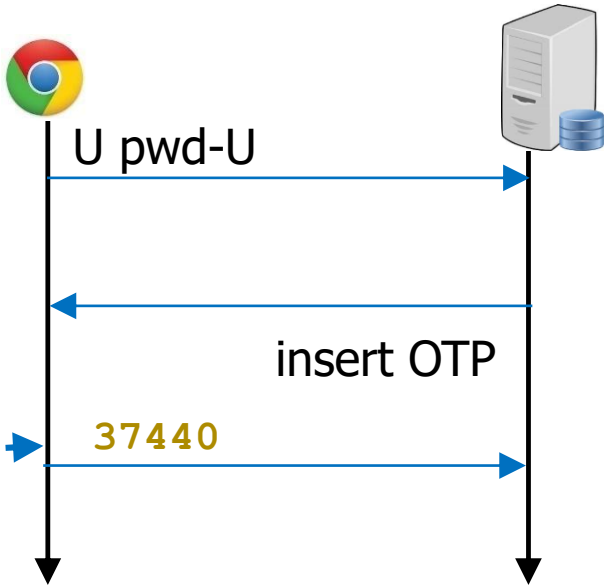






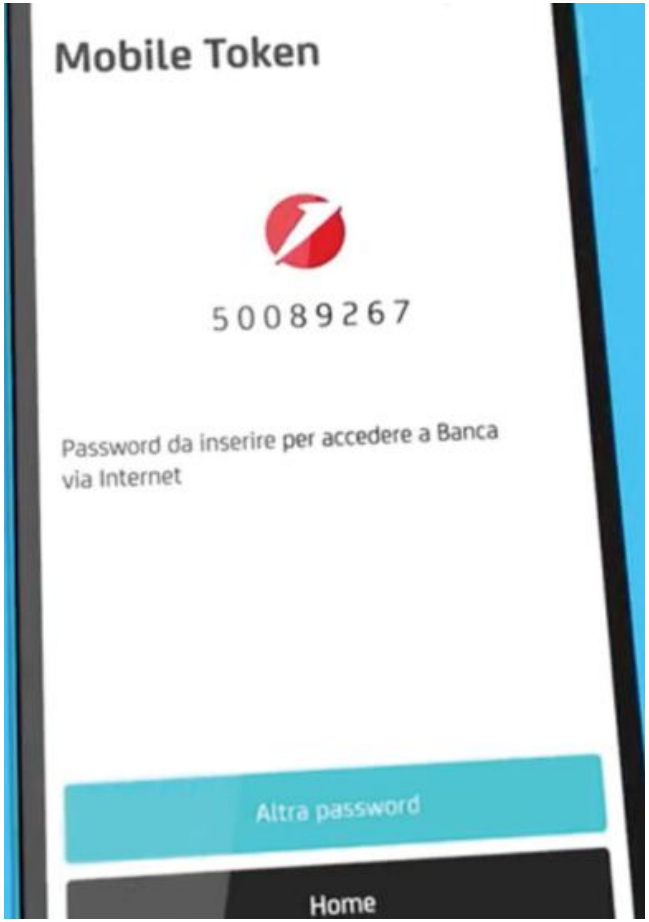
OTP AuthApp: Login





Remark

- A Service might use only "its own" Authenticator App



- Conceptually identical

OTP AuthApp: Implementation

AuthApp Linking: Requirement (I)

U1, K1

U1, K1

- Private key K1
  - Generated by service
  - Securely sent to AuthApp upon activation

AuthApp Linking: Requirement (II)

U1, K1
U2, K2
U3, K3

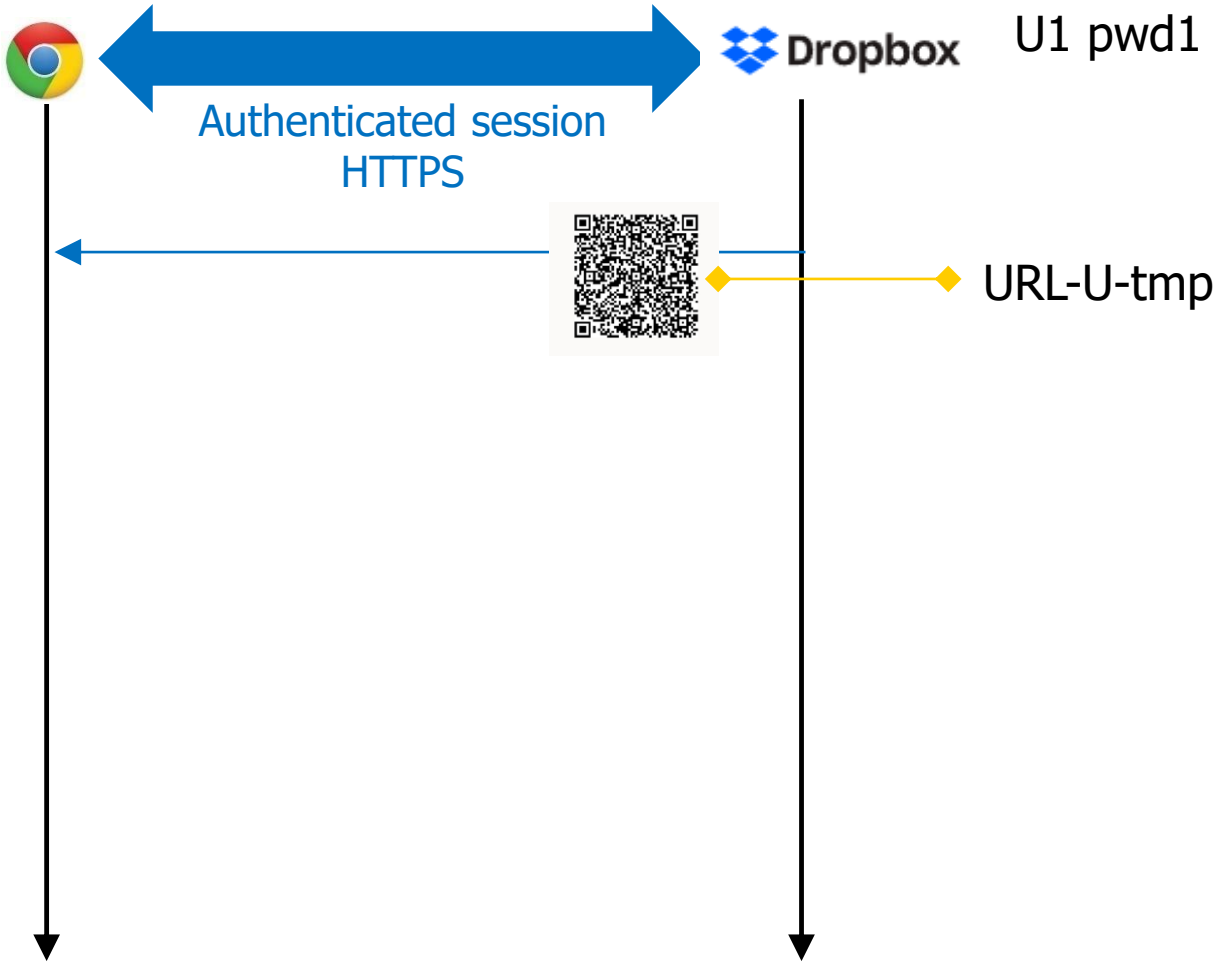
AuthApp usually linked to **several** services

U1, K1 Dropbox

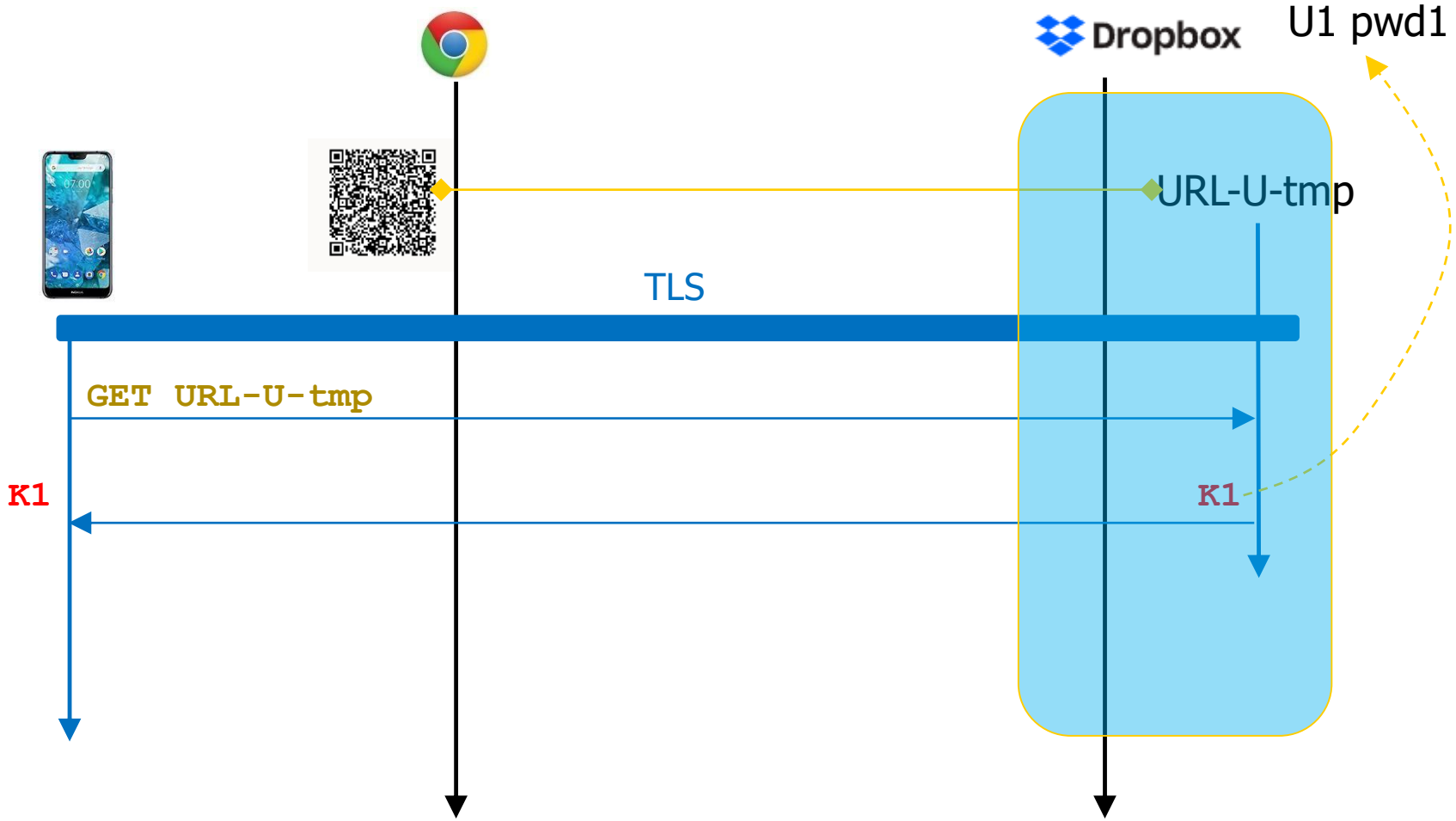
U2, K2

U3, K3 LastPass

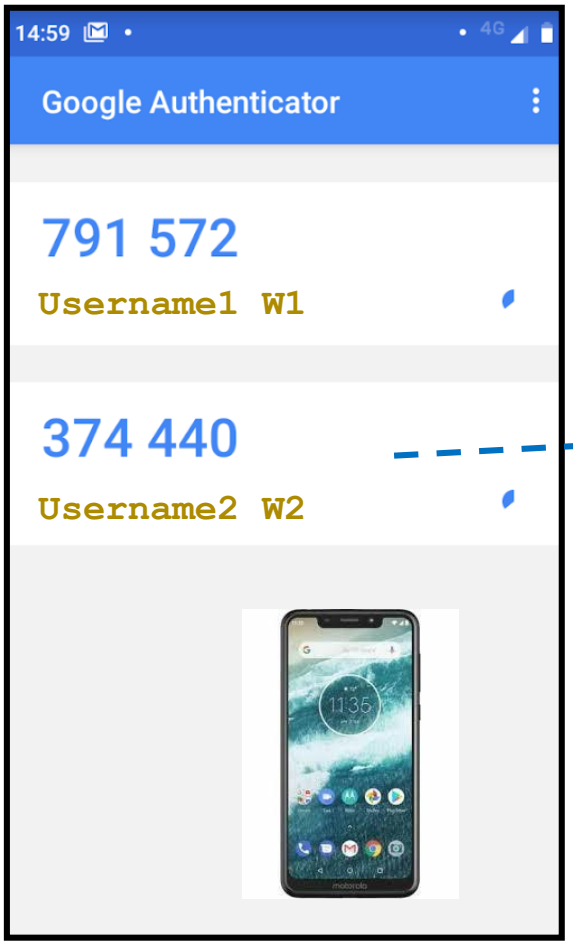
AuthApp Linking (I)

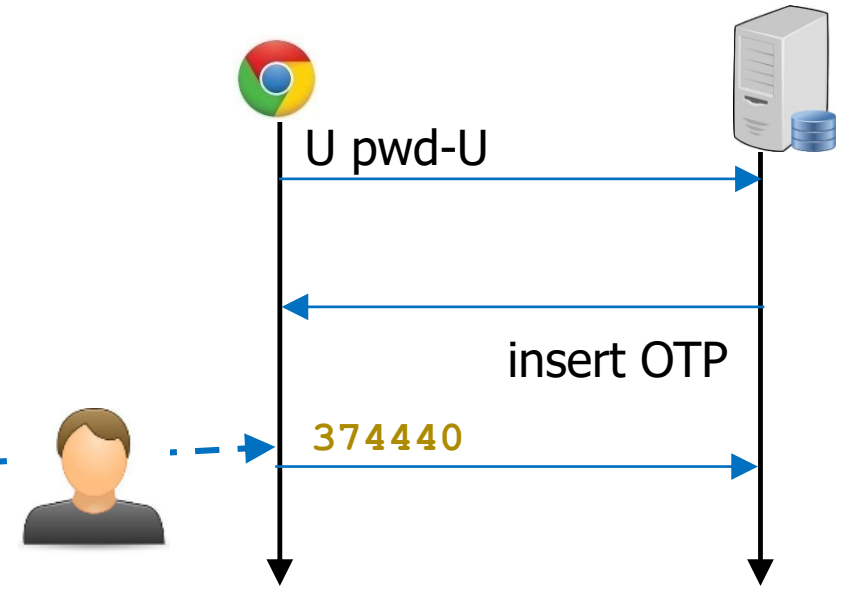


AuthApp Linking (II)



AuthApp Login: Requirement





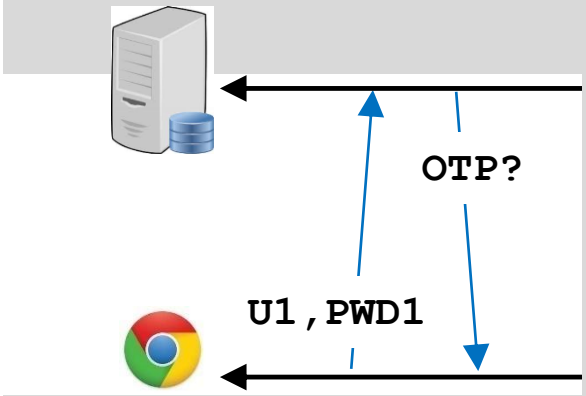
No communication Service - AuthApp

how can they agree?

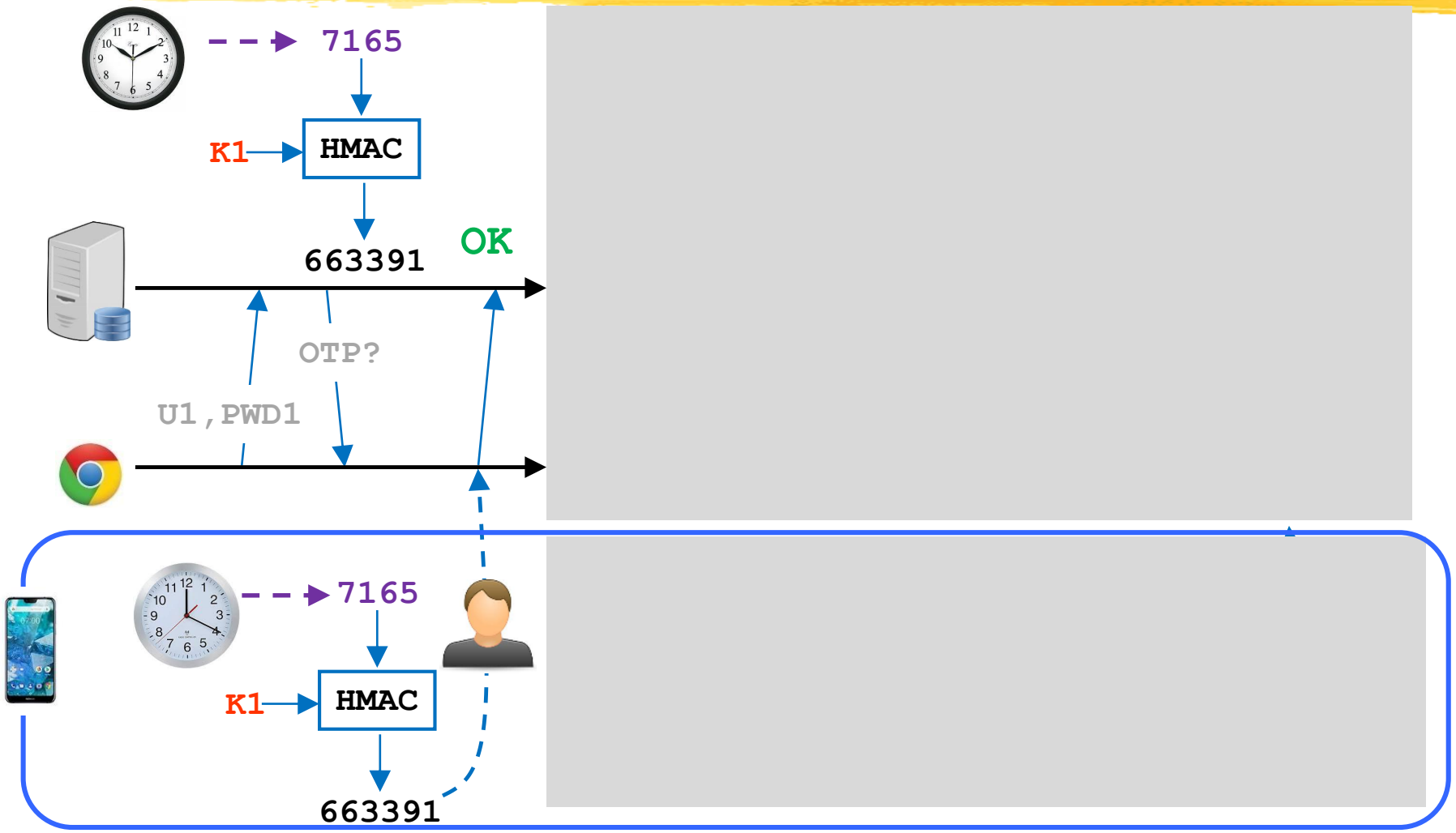
TOTP (Time-Based OTP): Basic Idea

- OTP(**t**) = ENCRYPT$_K$(**t** - T0) *// T0 conventional zero time*
- Read the clock and encrypt the result

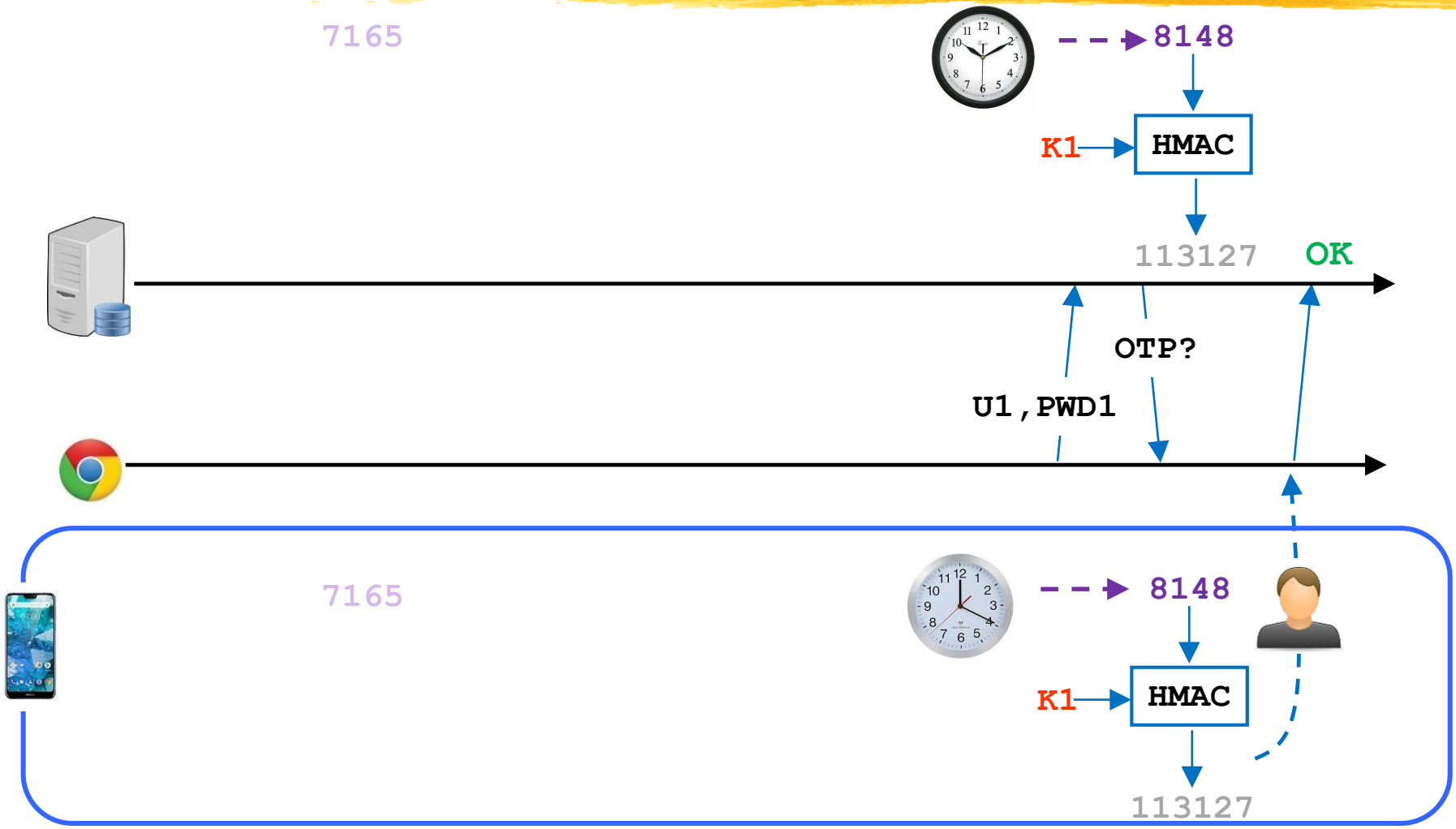
Example (I)



Example (II)



Example (III)


7165
8148
K1
HMAC
113127
OK
OTP?
U1,PWD1
7165
8148
K1
HMAC
113127

TOTP (Time-Based OTP): Some details

- Clocks cannot be perfectly synchronized
- Messages have latency

- OTP(**t**) = ENCRYPT$_K$(**t** - T0) **cannot work in practice**

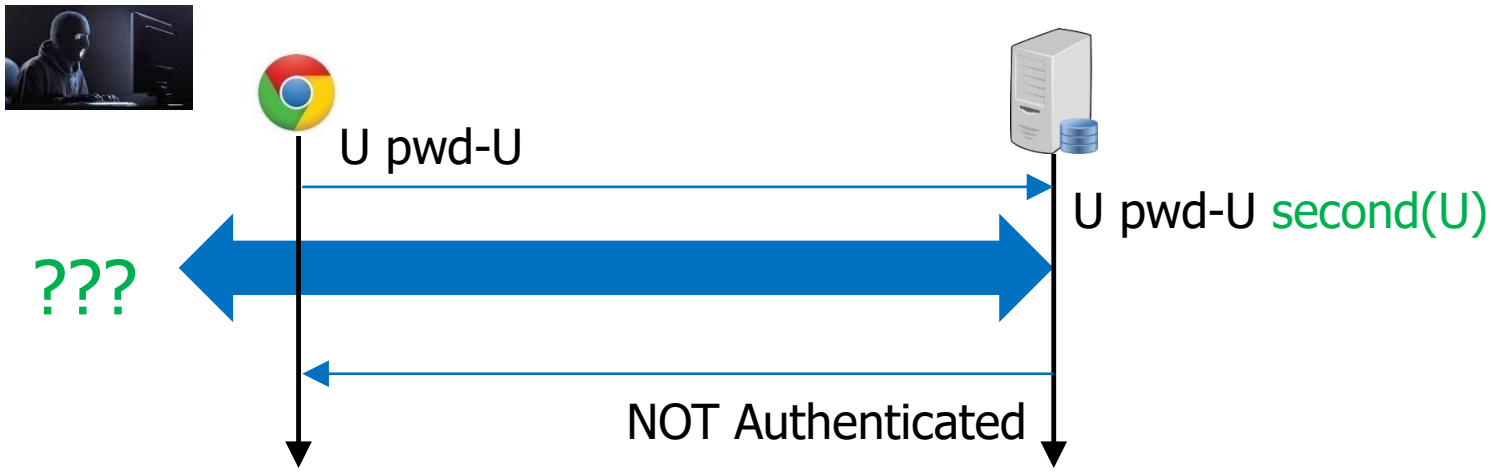
- OTP(**t**) = ENCRYPT$_K$(**t** - T0 **/ DX**) // DX = 30 s
- One-time password changes every 30 seconds

- Actual algorithm more complex
  https://en.wikipedia.org/wiki/Time-based_one-time_password

OTP Attacks

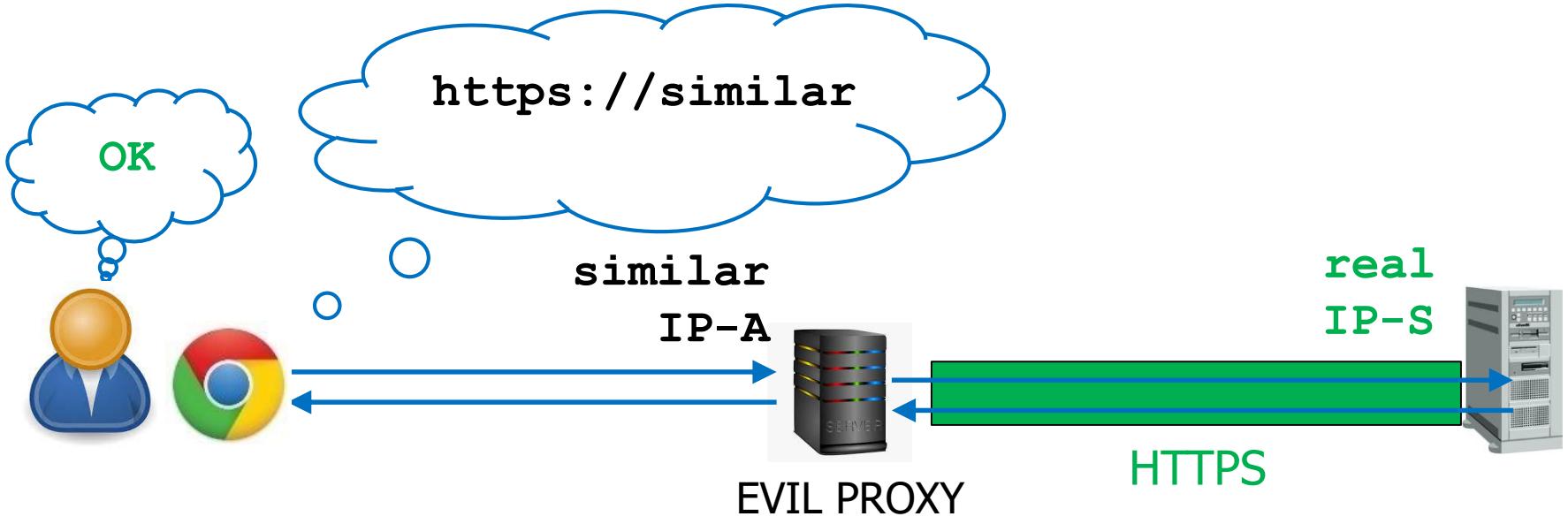
Threat Model: Stolen Password

- Adversary has <U,P>
- **Solved!**
- **Always keep in mind**

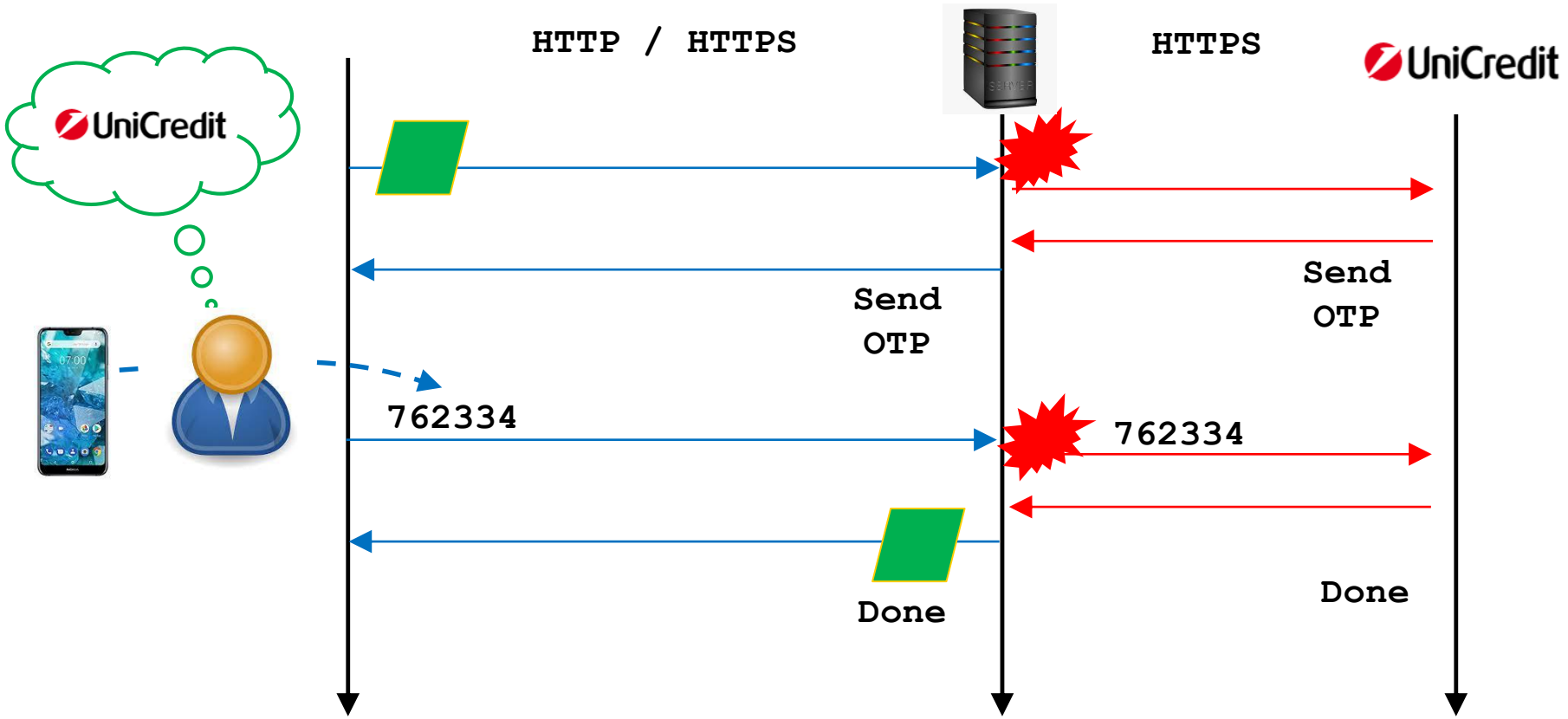


Threat Model: "Real-time Phishing" (REMIND)

- User does **not** detect that is accessing the **wrong** URL

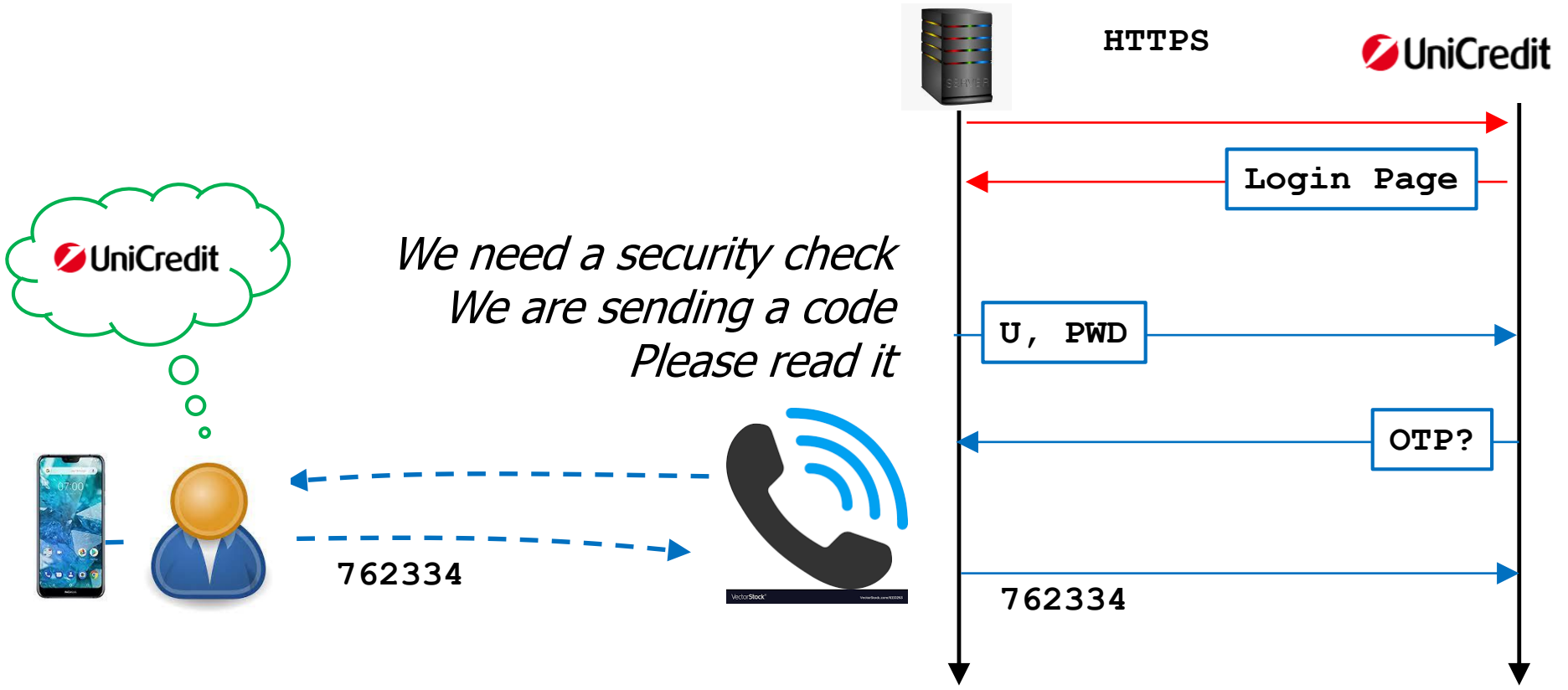


Threat Model: "Real-time phishing"



NOT Solved!

Other Threat Model: Vishing ("voice" phishing)



Unfortunately, it works…

IL PICCOLO

Cybertruffa vocale: 20 mila euro spariti

Chiamata e invio di un codice sul cellulare che la vittima è invitata a leggere a voce alta ma è l'ok a una transazione

04 Agosto, 2020

Search also "vishing" on Companion website

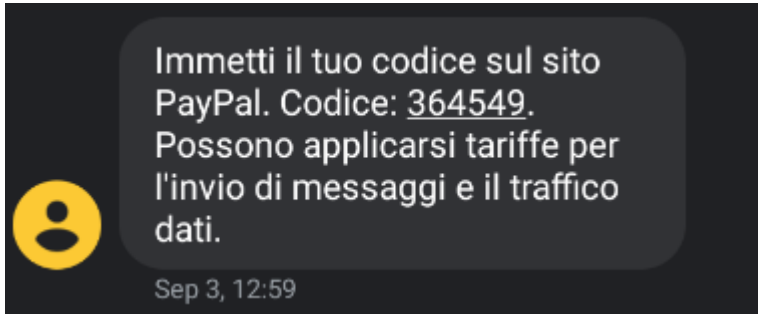
Keep in mind

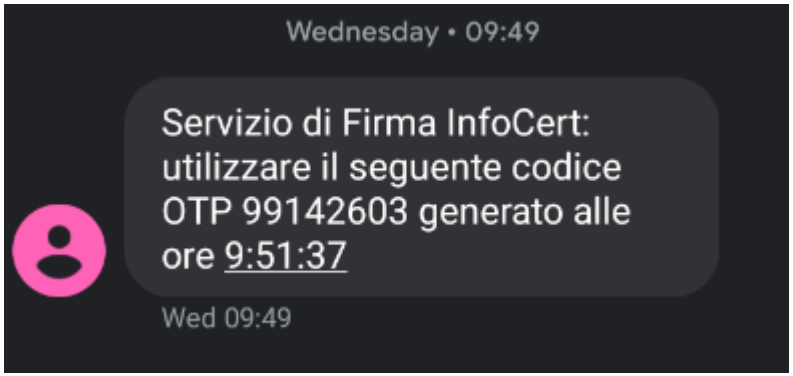
- OTP does **not** solve phishing

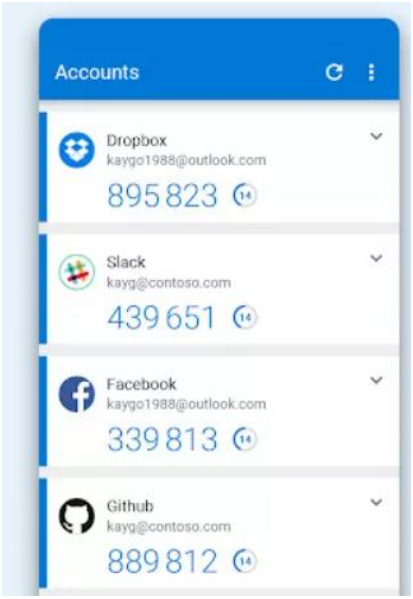
- OTP makes phishing much **more costly** to attackers
- ...but it is still a real danger

- Who am I giving this OTP to?
- For doing what?

Not very informative...







Much better


BBVA: Per eseguire il bonifico immediato di 610 EUR al conto di destinazione IT****[redacted] usa il codice 327272
Il tuo codice di verifica per accedere all'home banking 962341
BBVA: Per confermare il pagamento di 30,32 EUR effettuato con la tua carta ****[redacted] presso PAGOPA - WORLDLINE utilizza il codice 744660
30 nov, 18:16

Other Threat Models: OTP SMS

- Attacker knows password

  +

- Malware on smartphone
  - Read, forward, delete SMS
- SIM swap (fraudulent SIM change)
  - Phone number fraudulently taken by another SIM
- SMS routing attacks
  - SS7 phone protocol weakness: SMS sent to Attacker

- **Realistic** (search "MFA Attacks" on Companion website)
- **Not solved**

Other Threat Models: OTP AuthApp

- Attacker knows password
  +
- Malware on smartphone
  - Read, forward, delete SMS
- SIM swap (fraudulent SIM change)
  - Phone number fraudulently taken by another SIM
- SMS routing attacks
  - SS7 phone protocol weakness: SMS sent to Attacker

AuthApp does not grant any "screenshot rights" to any other app

OTP do not travel across phone network

- **Realistic** (search "MFA Attacks" on Companion website)
- **Solved**

SMS vs AuthApp

... it's time to start your **move away from the SMS** and voice Multi-Factor Authentication (MFA) mechanisms.

... It bears repeating, however, that **MFA is essential** – we are discussing **which** MFA method to use, not **whether** to use MFA.....

Alex Weinert, Director of Identity Security Microsoft
November 2020

OTP: Privacy Implications

- OTP-SMS:
  - Service must know User **phone number**
  - Service **might abuse** this information
    (e.g. as an identifier for linking identities across different marketing databases)

LILY HAY NEWMAN SECURITY OCT 9, 2019 2:32 PM

WIRED

Never Trust a Platform to Put Privacy Ahead of Profit

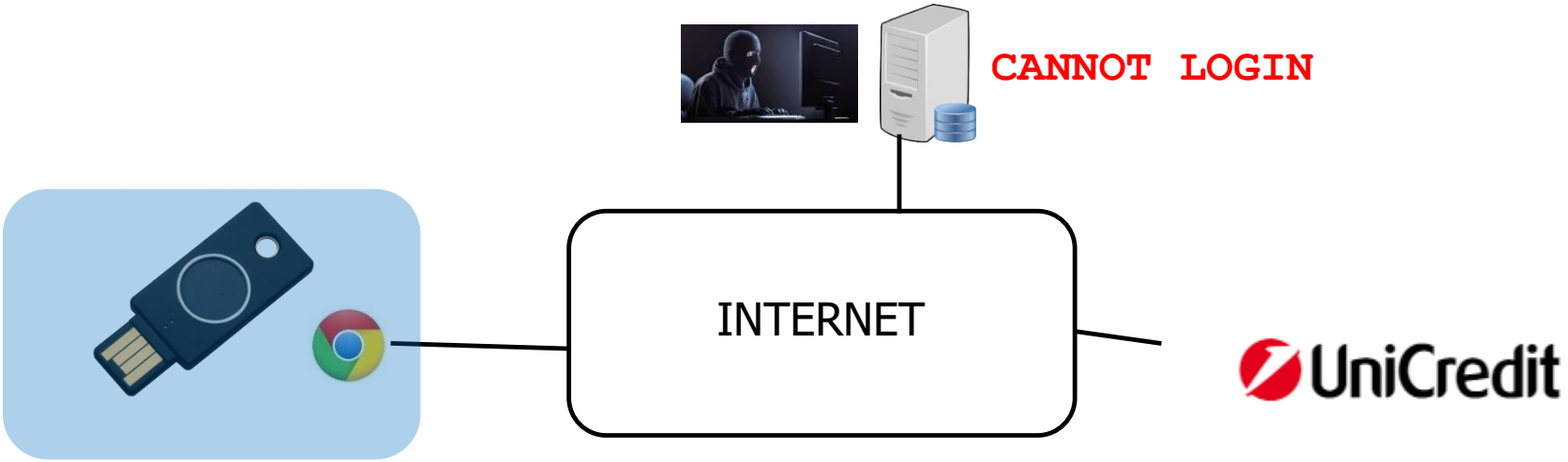
Twitter used phone numbers provided for two-factor authentication to target ads—just like Facebook did before.

- OTP-AuthApp:
  - Service need not know User phone number

Security Keys

Security Key

SecKey must be close to the Browser
(USB / Bluetooth / NFC)



Security Key Linking: Requirement

DNS NAME

W1, U1, Kpriv-1, Kpub-1
W2, U2, Kpriv-2, Kpub-2
W3, U3, Kpriv-3, Kpub-3

One SecKey associated with **many** services

W1 U1, Kpub1 — Dropbox

W2 U2, Kpub2

W3 U3, Kpub3 — LastPass

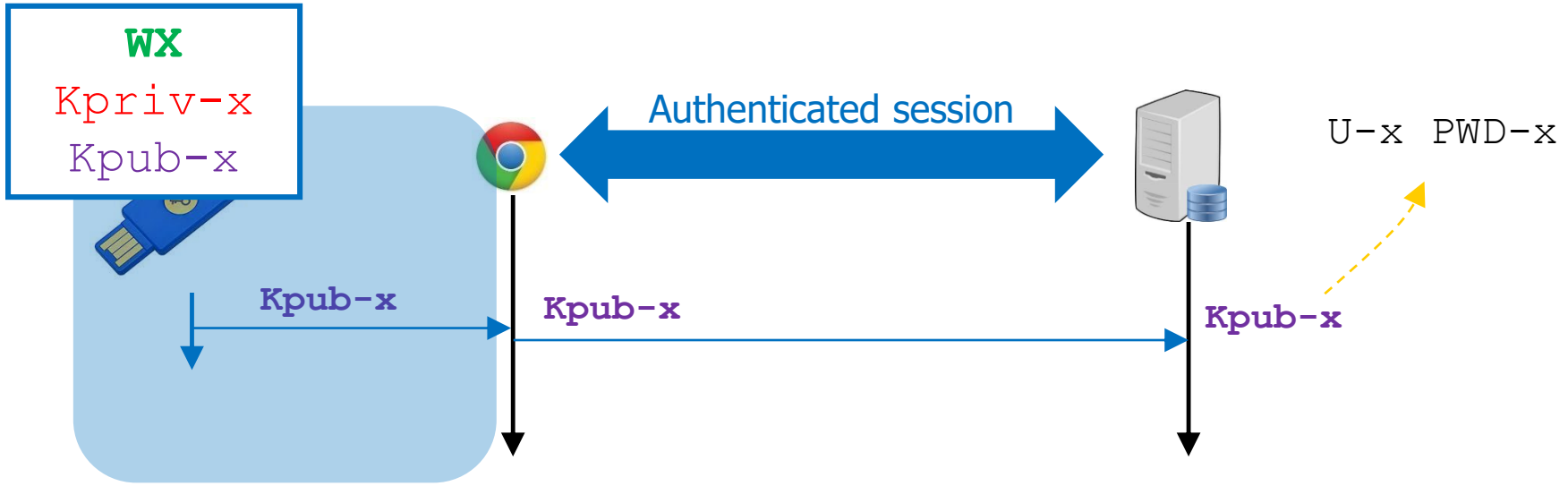
Remark

- Username omitted from next slides for ease of description
  - **One** username and keypair for each service

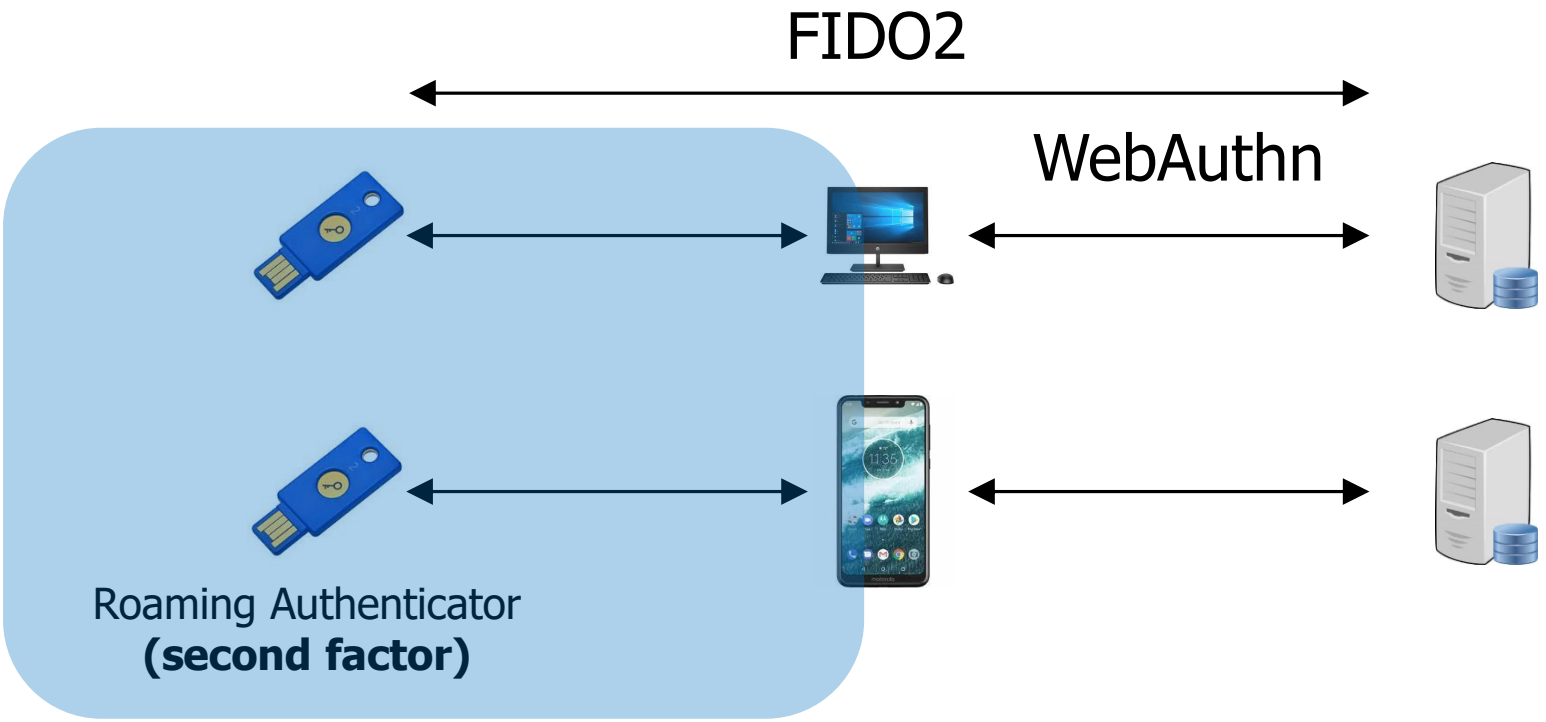
W1, ~~U1,~~ Kpriv-1, Kpub-1
W2, ~~U2,~~ Kpriv-2, Kpub-2
W3, ~~U3,~~ Kpriv-3, Kpub-3

Security Key Linking: Implementation

1. User authenticates to WX with `U-x, PWD-x`
2. SecKey generates <`Kpriv-x`, `Kpub-x`> to be used **only** with WX
3. SecKey sends `Kpub-x` to WX securely
4. **WX associates** `Kpub-x` with `U-x`

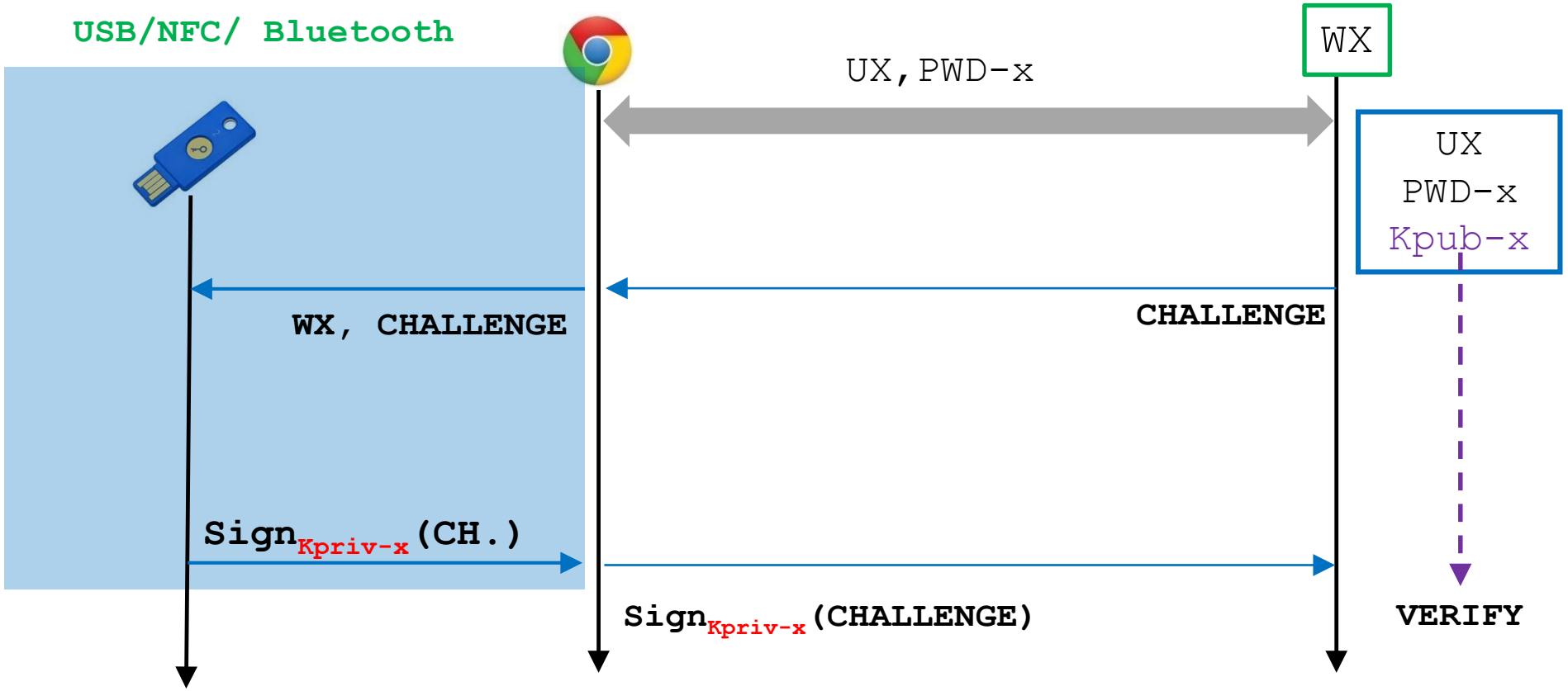


Open Standards (very complex...)



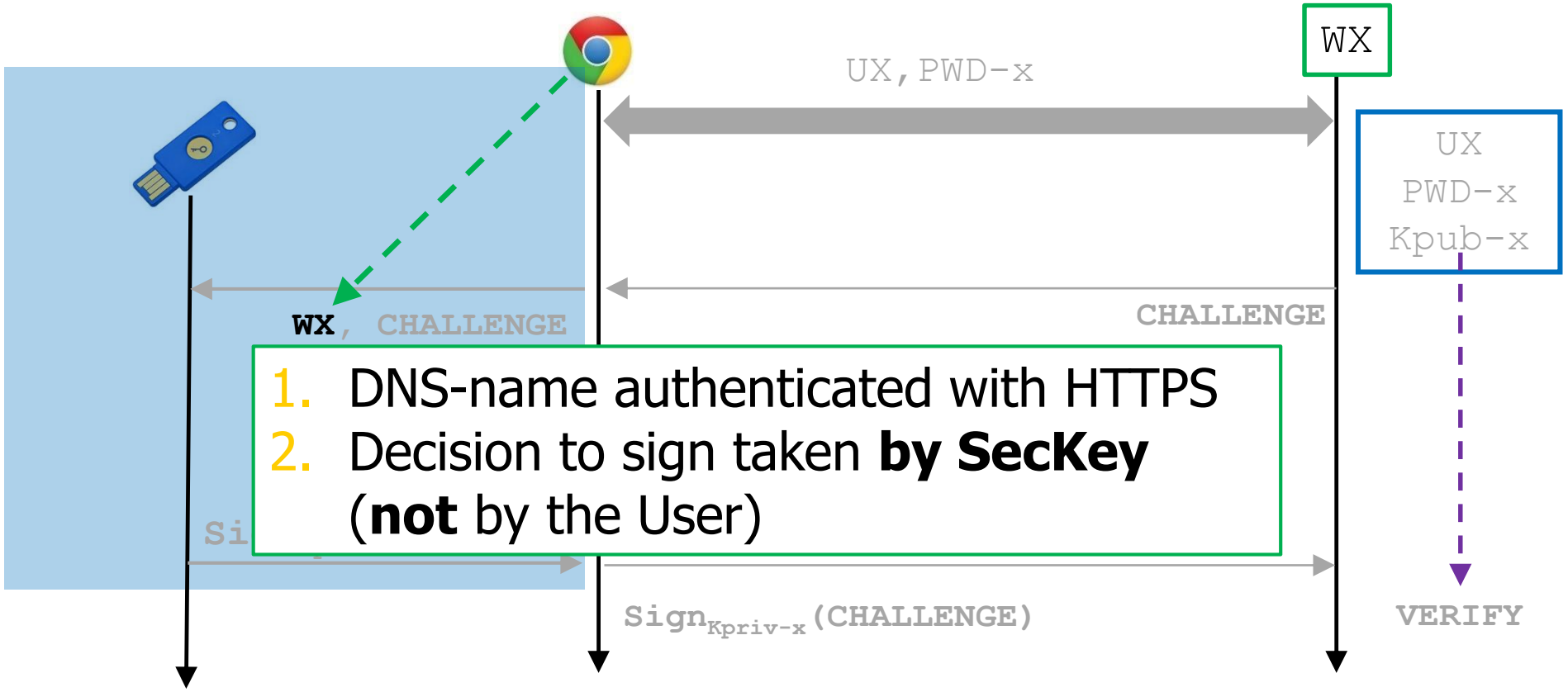
No direct communication
Service ↔ Second factor

Security Key: Login



Real flow more complex

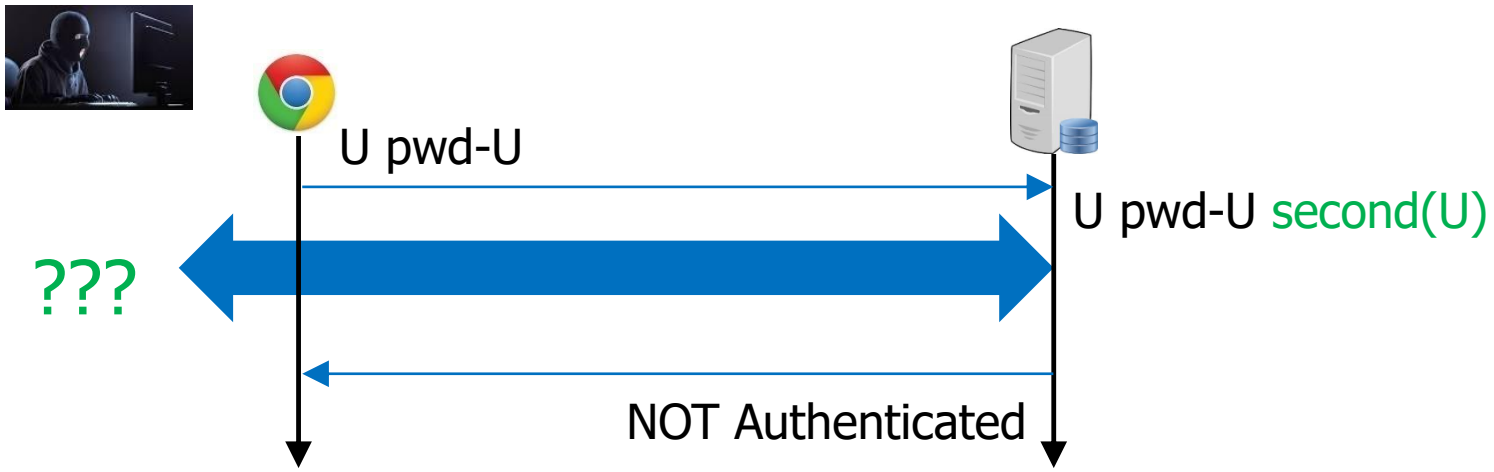
Key facts


WX
UX, PWD-x
UX
PWD-x
Kpub-x
WX, CHALLENGE
CHALLENGE
1. DNS-name authenticated with HTTPS
2. Decision to sign taken **by SecKey** (**not** by the User)
Sign$_{Kpriv-x}$(CHALLENGE)
VERIFY

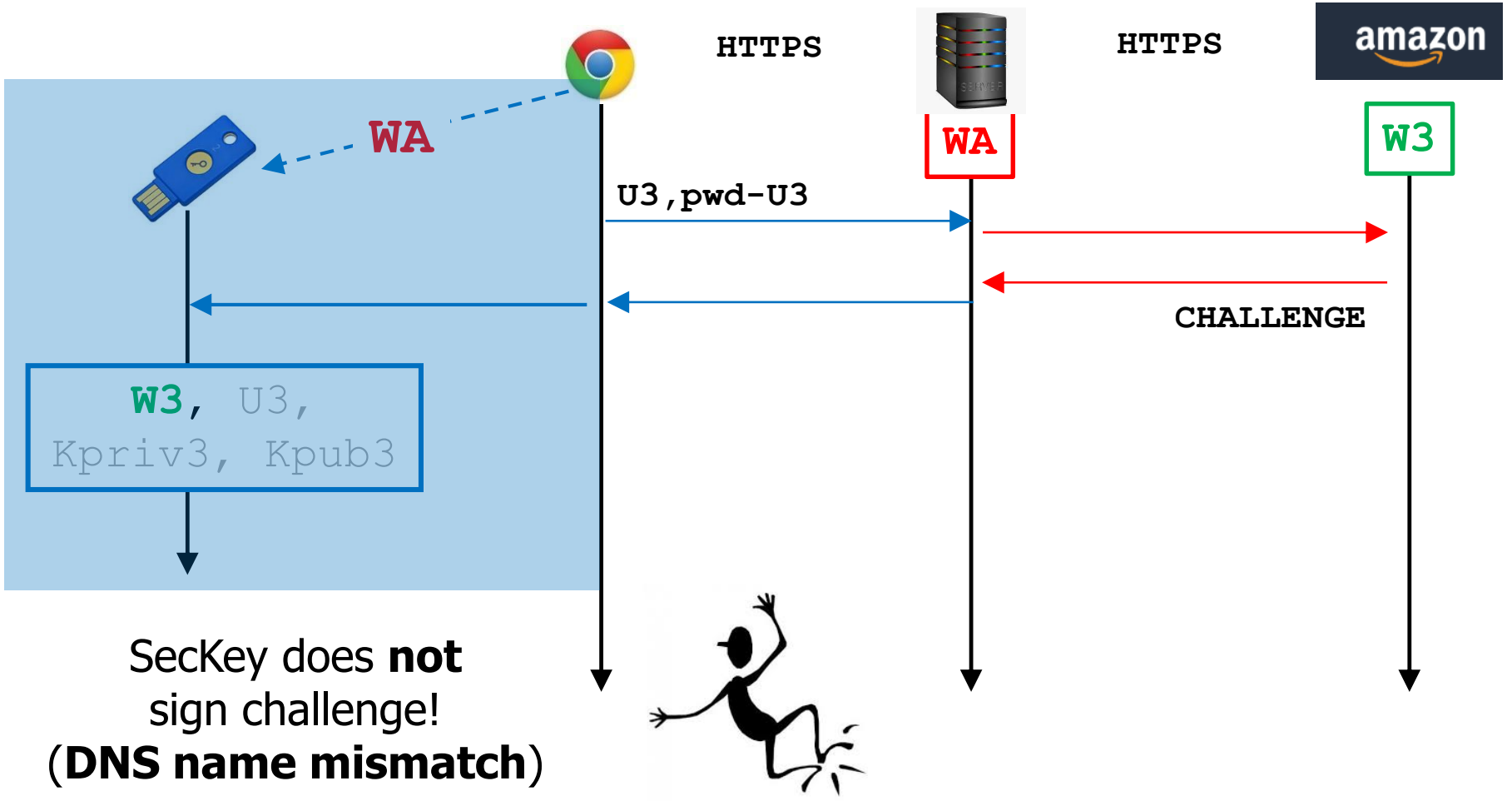
Security Key: Attacks

Threat Model: Stolen Password

- Adversary has <U,P>
- **Solved!**
- **Always keep in mind**

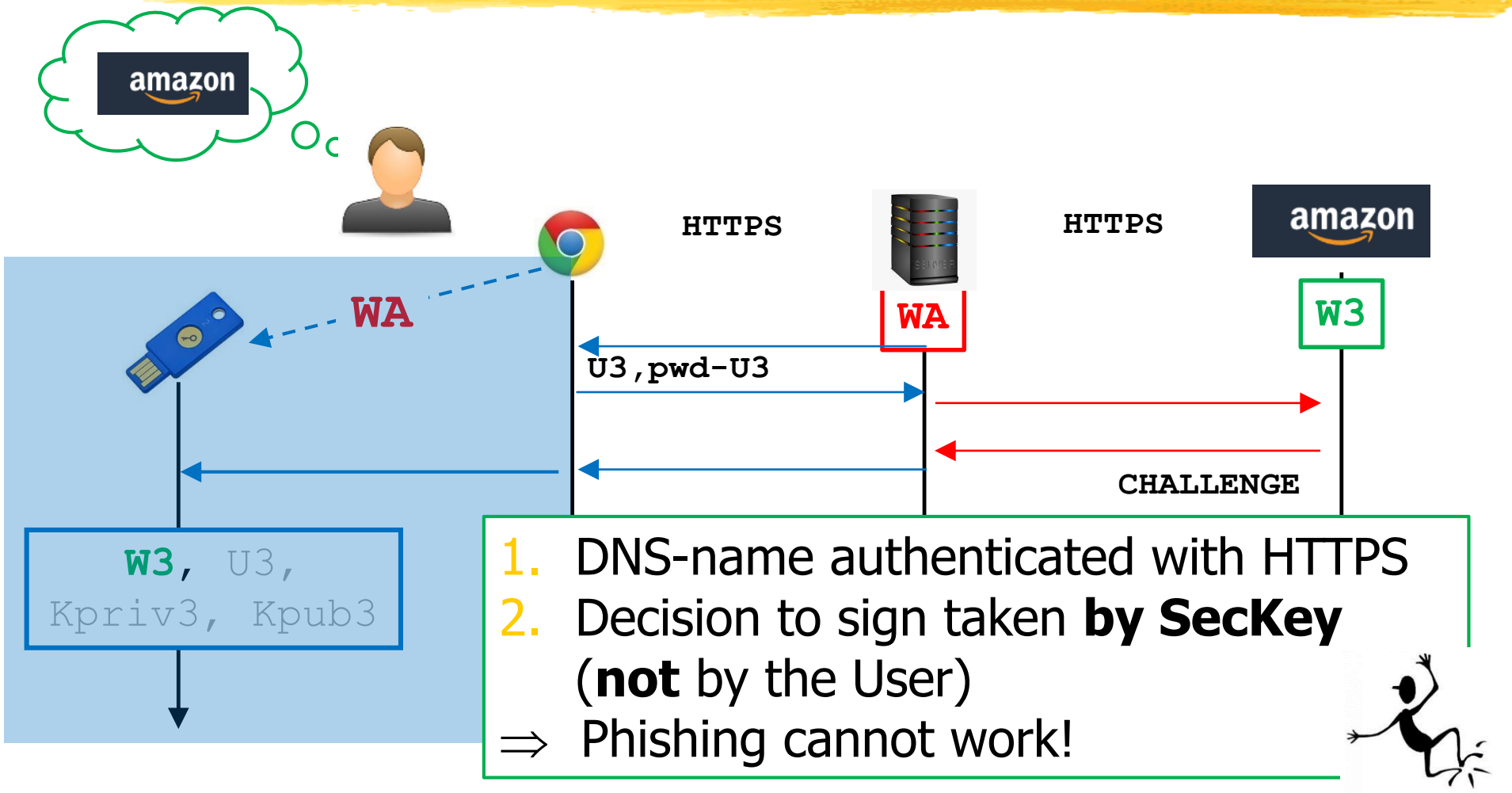


Real-time Phishing Solved! (I)



SecKey does **not** sign challenge! (**DNS name mismatch**)

Real-time Phishing Solved! (II)



Unsolved Threat models (out of scope)

1. Attacker has **valid certificate for S name**
   - Browser cannot discriminate between real and fake service

2. Attacker has **malware on Browser device**
   - Malware can alter/forge Browser / SecKey traffic

Remark

MANY (omitted) complex details for coping with crucial requirements

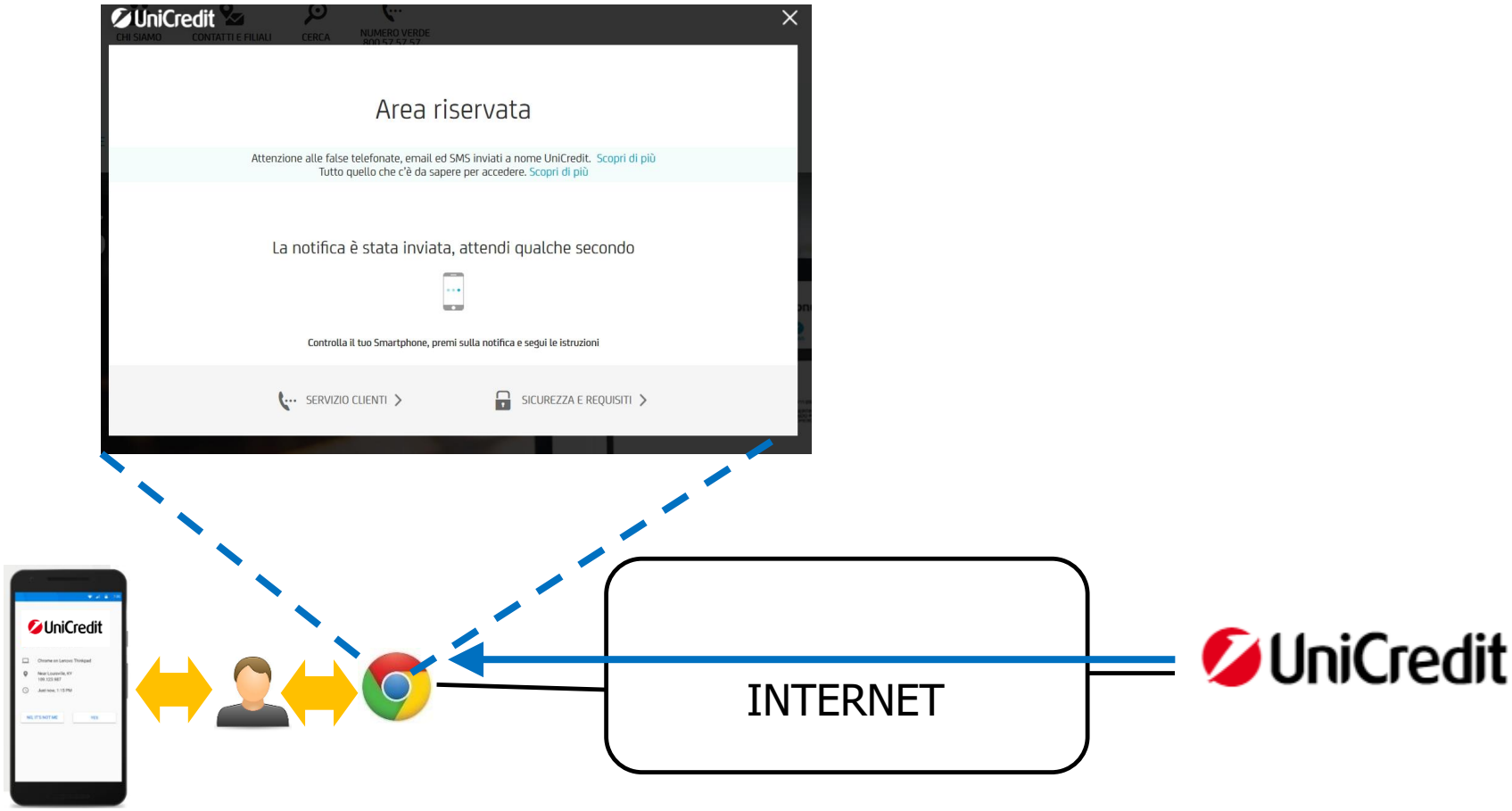
- Attacker has physical access to SecKey (loss, stealing, brief access)
  - Cloning must be very difficult
  - Extracting set of service names must be very difficult
- Attacker may be the Manufacturer
  - If and when we realize it, certain SecKeys can no longer be trusted; Service must be able to know who the Manufacturer and product id are
- Sets of Services might collude to link the respective user identities
  - Service cannot identify which specific SecKey it is interacting with

Push notifications

Second factor (REMIND)

- **Smartphone**
  - OTP SMS
  - OTP Authenticator App
  - Push notifications
- **SecurityKey** (USB/NFC/Bluetooth)

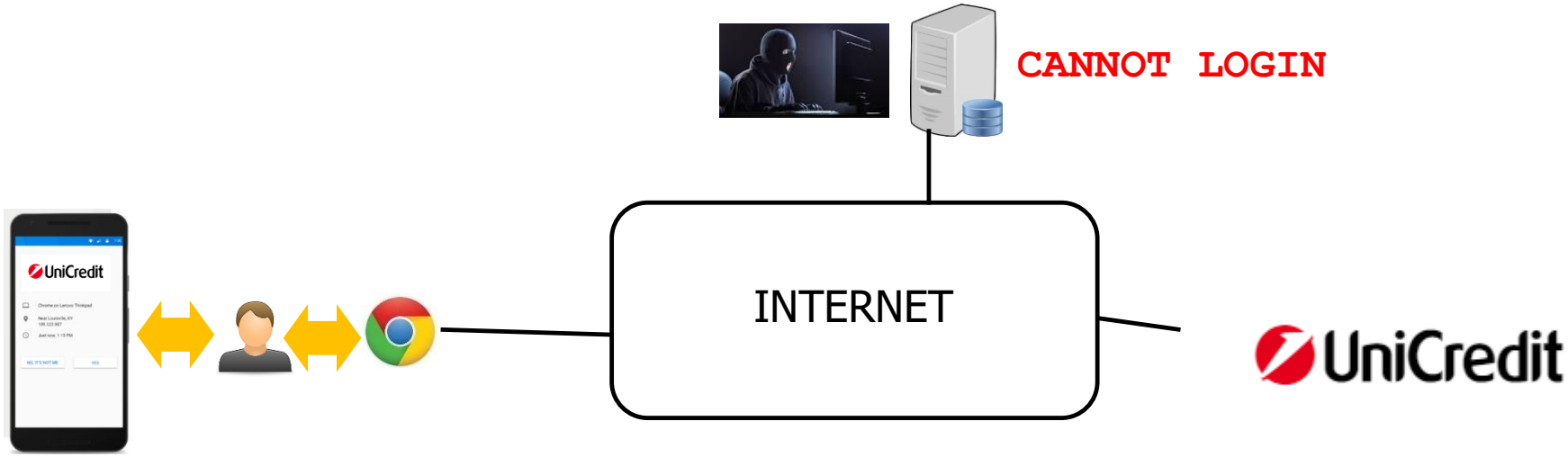
Smartphone Push Notification (I)



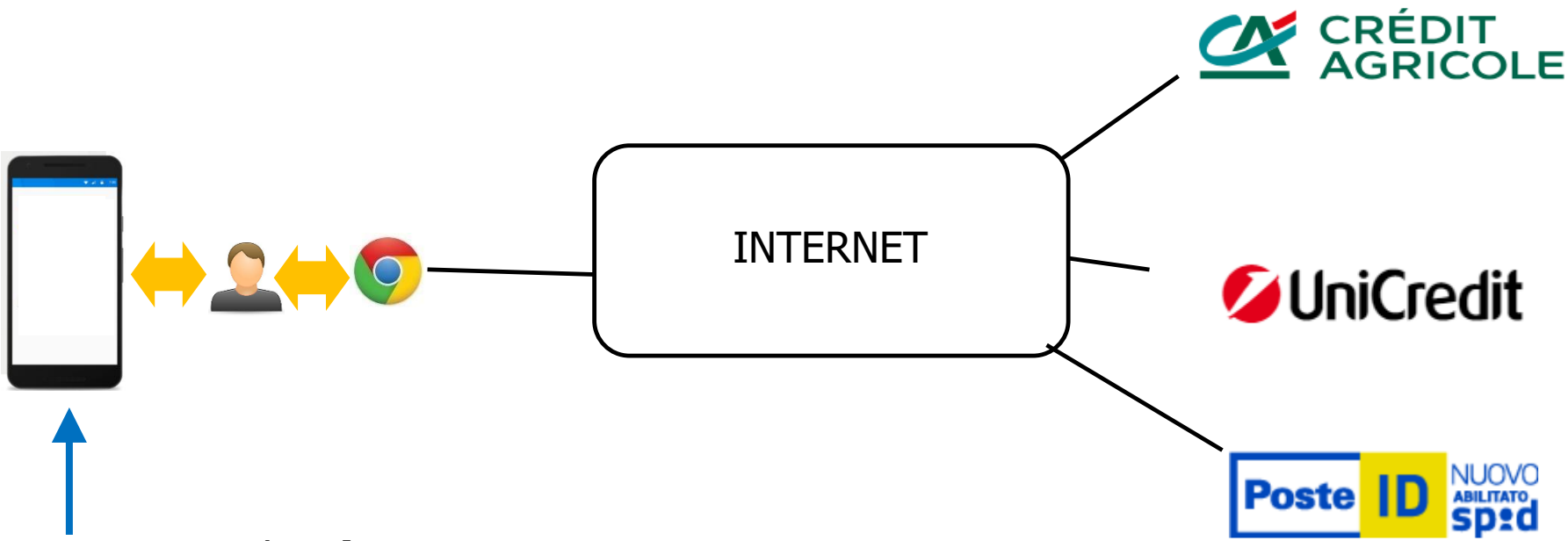
Smartphone Push Notification (II)

~~SecKey must be close to the Browser~~
~~(USB / Bluetooth / NFC)~~

Smartphone must be close to the User

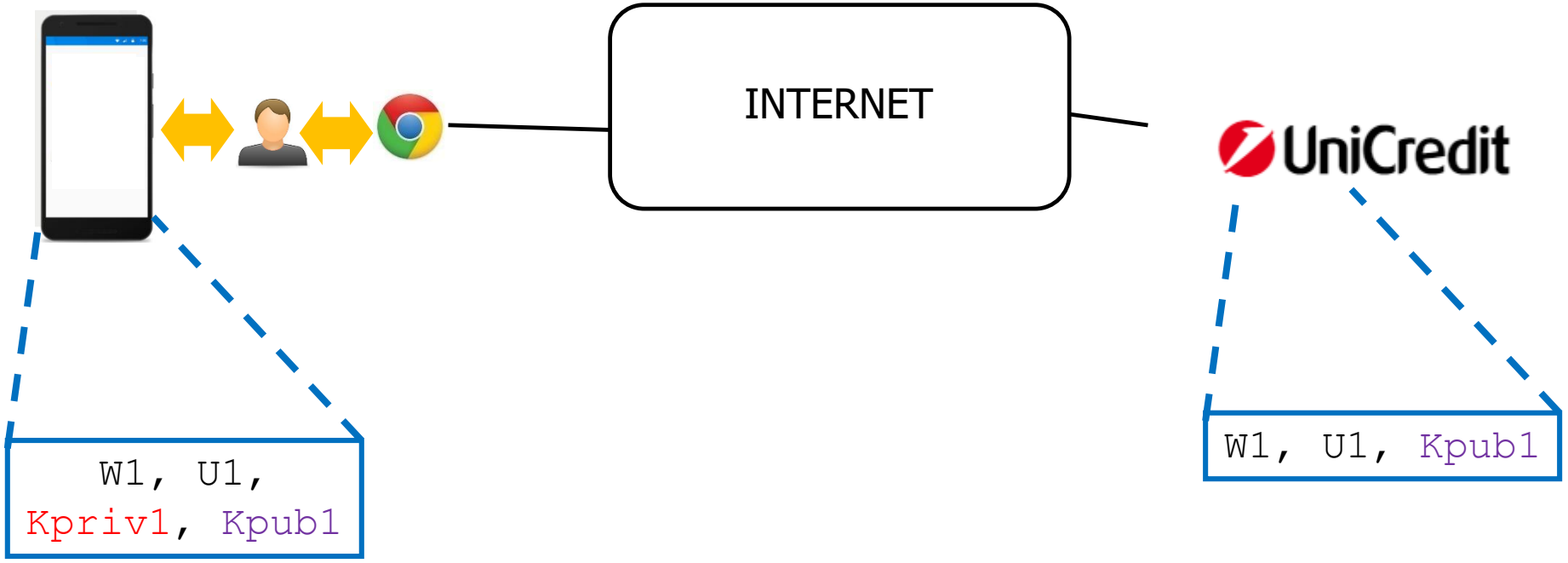


Smartphone Push Notification (III)

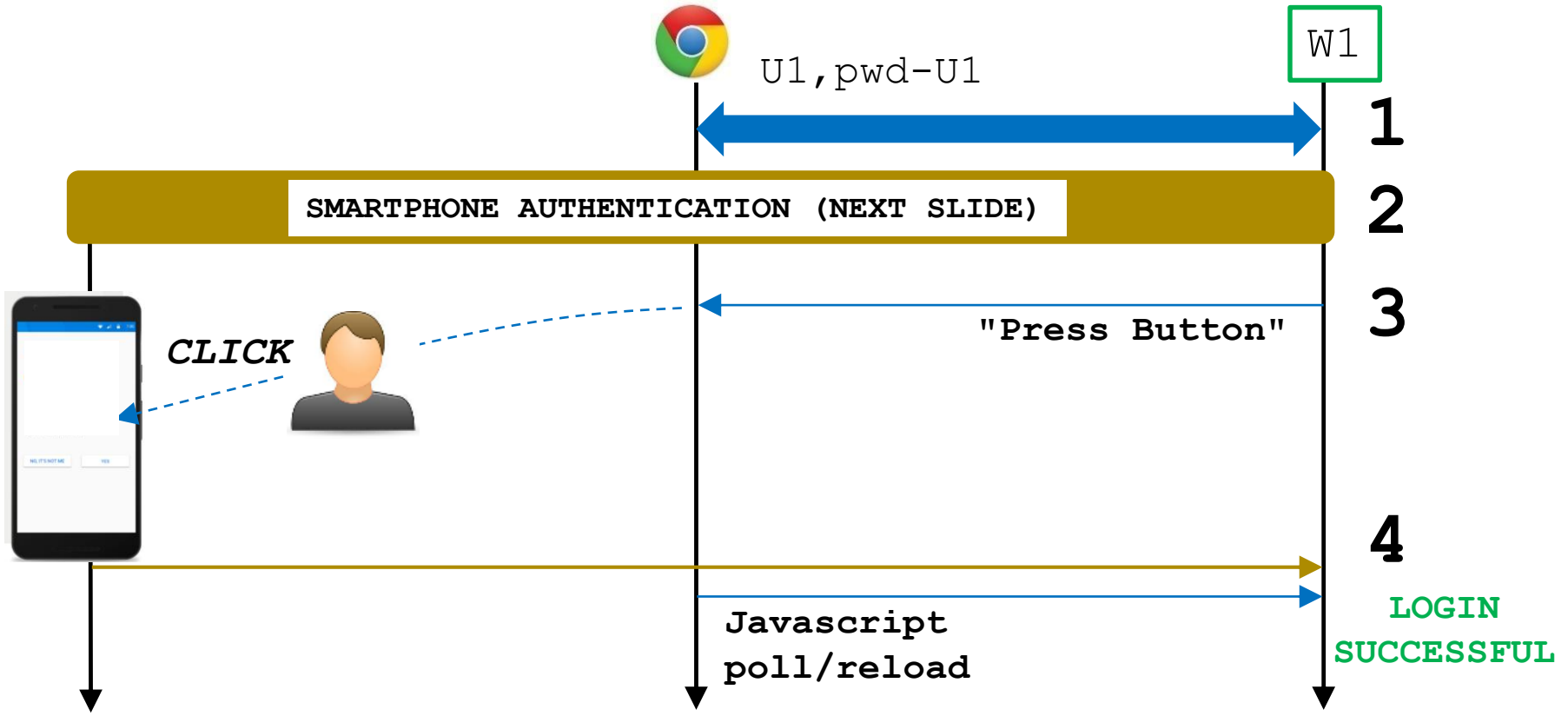


Every service has **its own** App where it sends notifications

Linking Requirement (Implementation omitted)



Login (Outline) (I)



Key Problem

U1,pwd-U1

W1

1

2

- "**push**" ⇒
  Service should connect to **Smartphone**
- What is its IP-address?
- It is usually **private** and **dynamic**

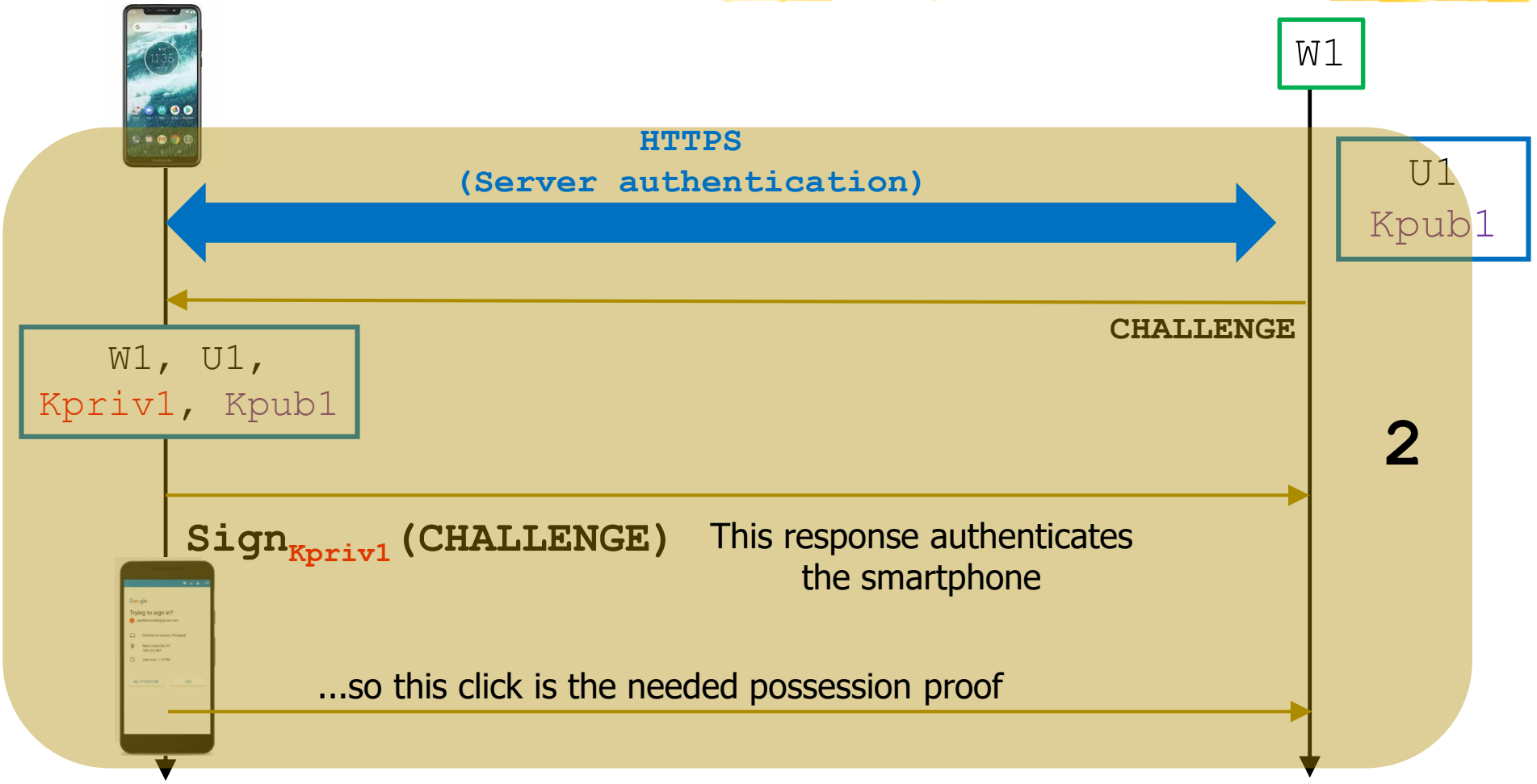
Solution (Outline)

- Service should connect to Smartphone

- What is its IP-address?
- It is usually **private** and **dynamic**
- Service sends notifications to a **cloud service**
- Every smartphone:
  - Continuously **polls** that cloud service
  - Connects as a TCP client and checks whether there is any notification

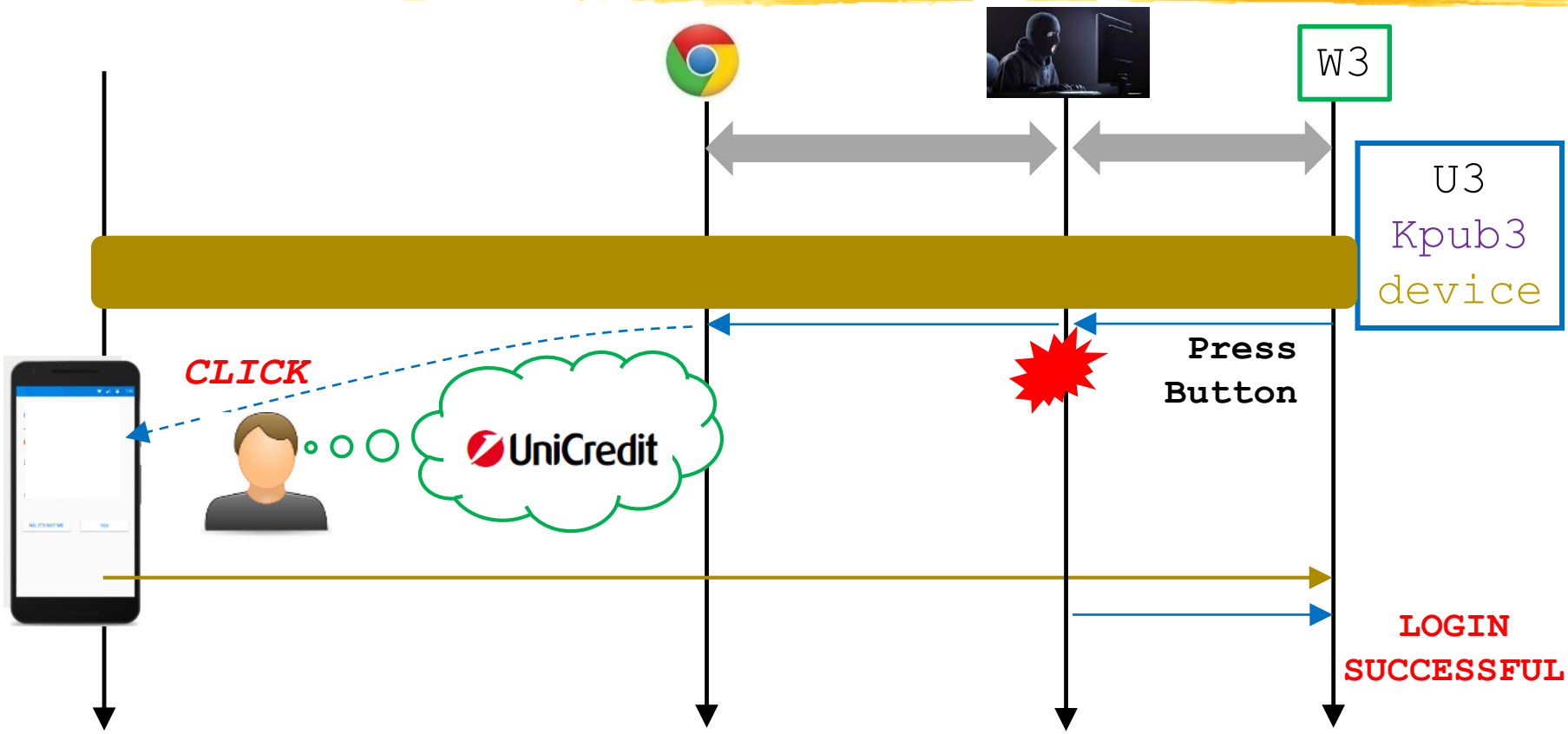
- Alternative implementation:
  - User launches smartphone app that acts as a client and connects to the service
  - Next slide assumes this pattern

Login (Outline) (II)



Push notifications Attacks

Threat Model: "Real-time phishing"



NOT Solved!

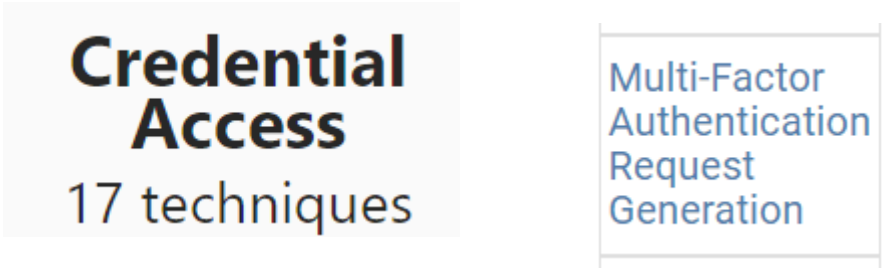
Keep in mind: Not Phishing-Resistant

W3

U3
Kpub3
device

- User might not understand which **Browser** is logging in
- User must read and be careful (exactly the phishing issue)

MFA Bombing



- Adversaries may **continuously** repeat login attempts in order to **bombard** users with MFA push notifications, SMS messages, and phone calls, potentially resulting in **the user finally accepting the authentication request** in response to "MFA fatigue."

- Unbelievable but it may indeed work...

MFA: Limitations

Keep in mind (REMIND)

MFA is extremely important

- Very effective for very realistic threat model
- Enabling 2FA is a very high priority defensive investment

MFA is essential … Use of anything beyond the password significantly increases the costs for attackers, which is why **the rate of compromise of accounts using any type of MFA is less than 0.1% of the general population**.

Alex Weinert, Director of Identity Security Microsoft
November 2020

Summary of Limitations

- **Everything but SecKey**
  - Phishing / Voice Phishing
  - Who am I authorizing?
  - For doing what?

- OTP AuthApp better than OTP – SMS
  - Malware
  - SIM swap
  - SMS routing

Practical Considerations

Our Focus (REMIND)

- **Web app** (BASIC/FORM over HTTPS)
- **Different** organizations

- Extremely relevant in practice

Hhmmm…

- **Web app** (BASIC/FORM over HTTPS)
- **Different** organizations

- Can I enable it on our **Enterprise Wi-Fi**?
- …on our **mail server**?
- …on our **workstation/notebook logons**?

Fact(s)

- Many complex technologies exist for supporting MFA **within** organizations
  - Service software must support MFA and such technologies (obviously)

- Some services are **intrinsically unable** to support MFA ("**legacy protocols**")
  - POP/SMTP?
  - Enterprise Wi-Fi?
  - SMB?
  - …

Consequence

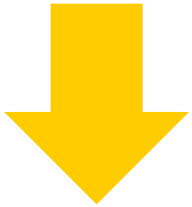
- Attacker knows <U,P>
- MFA mandatory on S1,….,SN but **not enabled** on S-X

- Attacker can impersonate U on S-X

- Do we know **the set of S-X** in our env?
- Do we know **what can be done** from there?

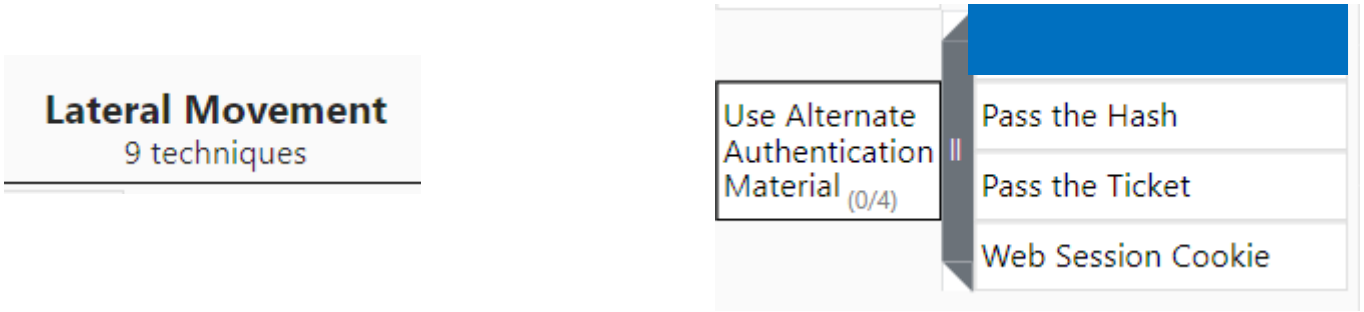
Does 2FA protect AFTER authentication?

- 2FA is checked **only** at the **beginning** of a session
  - Webapp login
  - Workstation logon (if it were deployed in such a setting)



- 2FA does **not** defend against attacks **during** an authenticated session
  - Stealing of authentication cookie
  - Pass-the-hash / Pass-the-ticket

Use Alternate Authentication Material

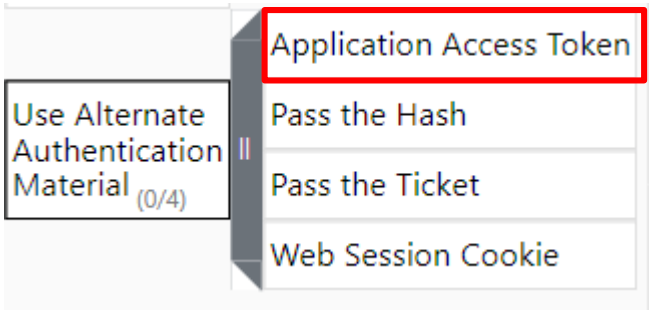


- Alternate authentication material is **legitimately generated** by systems **after** a user or application successfully authenticates by providing a valid identity and the required authentication factor(s).
- By **stealing** alternate authentication material, adversaries are able to bypass system access controls and authenticate to systems **without knowing the plaintext password** or **any additional authentication factors**.
- **HUGE problem**

Recent Example

Safeguarding Salesforce: What You Need to Know About the OAuth Token Compromise

Rapid7



- Google Cloud Threat Intelligence recently reported a data theft campaign…attackers **stole Salesforce OAuth and refresh tokens** from a third-party integration (Salesloft Drift) and used them to access and exfiltrate sensitive data including AWS access keys, passwords, and Snowflake tokens.
- OAuth tokens are trusted by Salesforce and can provide **persistent access without requiring stolen passwords or bypassing multi-factor authentication (MFA).** This makes token abuse especially difficult to detect through traditional security controls.

Passwordless Login (Passkey)

Warning: Terminology

- "Special" handling of **2FA@S** from **certain devices**
  - Only password (no 2FA)
  - Not even password (!)

- **Terminology not uniform** across vendors / consortia
- **Myriad** of different scenarios:
  - We will give a "simplified and general" description
  - Mapping to specific cases not easy
  - Search "passwordless" on companion website
- Key terms:
  - Trusted device
  - Passwordless device (or login)
  - **Passkey**

Trusted Device

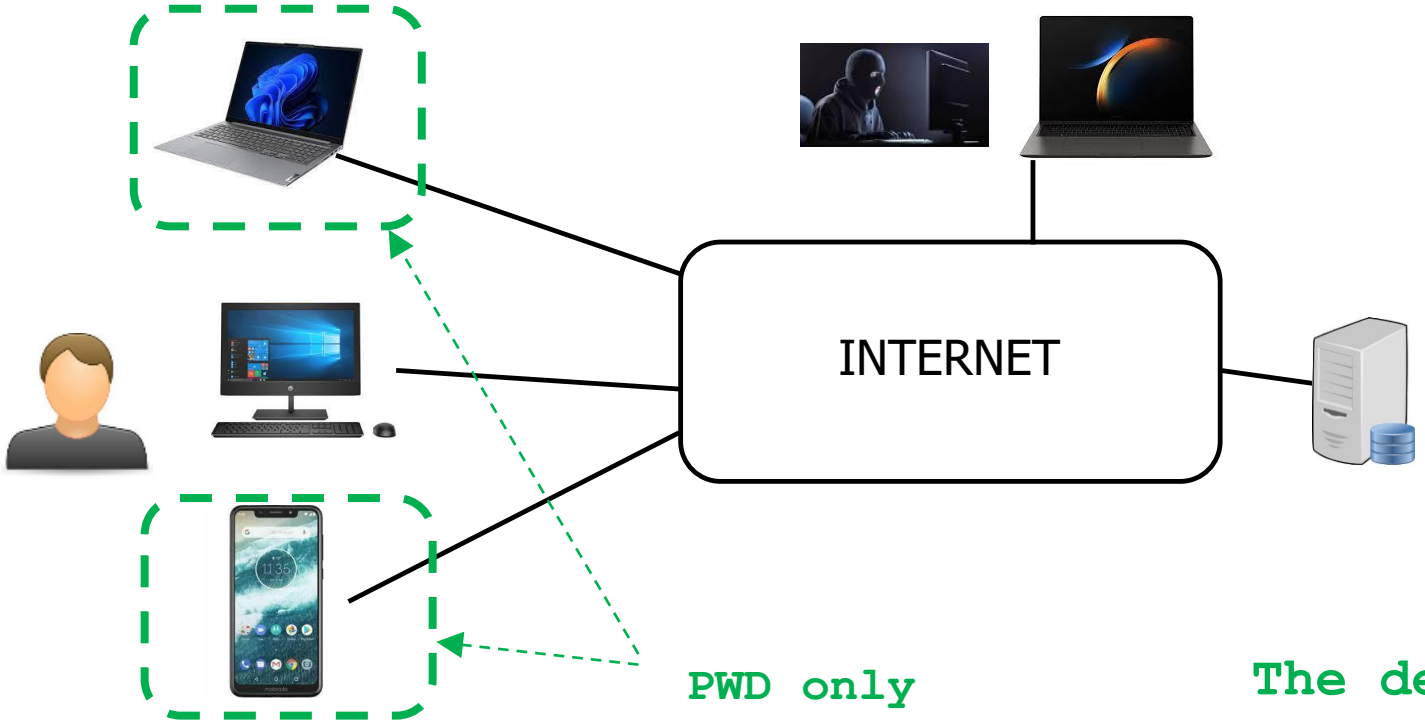
Trusted Device: Functionality

- Scenario:
  - User has activated **2FA** on Service S

- User may declare a certain device D-K **trusted**:
  - User authenticates from D-K with **password only**

- Much easier to use

Trusted Device ≈ 2nd Factor

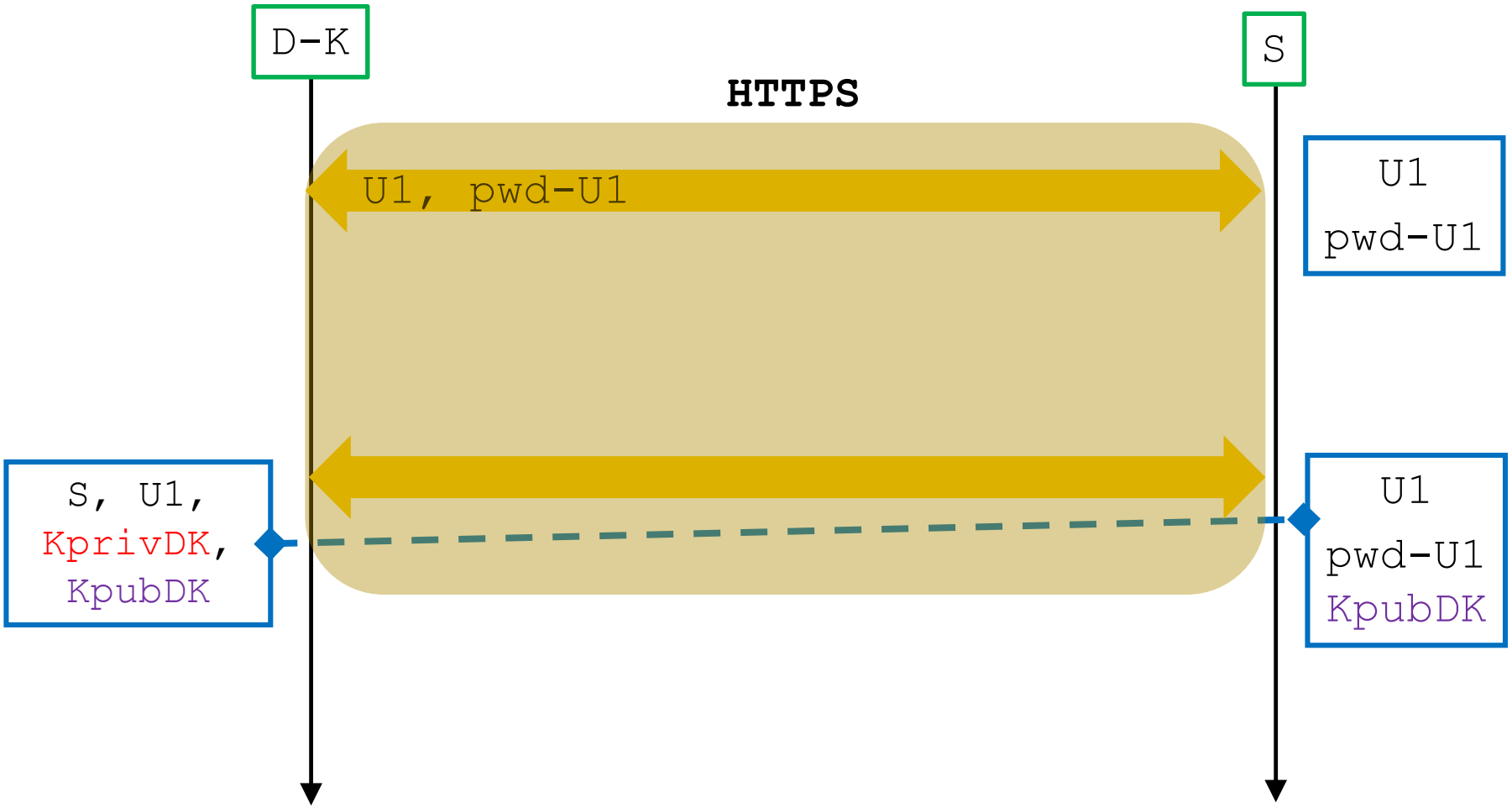


PWD only
Second factor
NOT needed

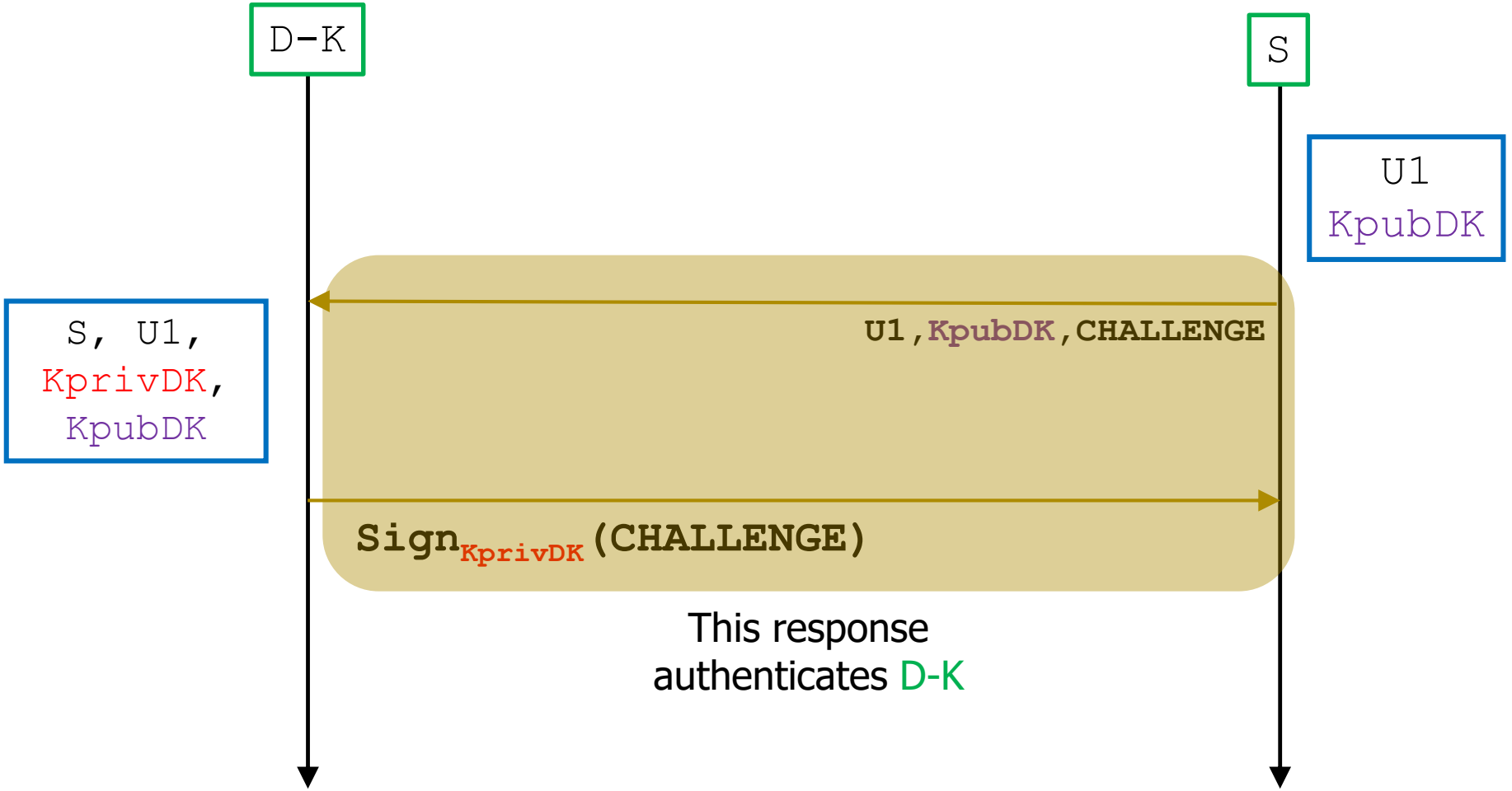
≈

The device itself
is
the second factor

Trusted Device Establishment (Outline)



Trusted Device Authentication (Outline)



Remark 1

- User may declare device D-K **trusted**:
- This can **only** happen while User is logged in from D-K with password **and 2nd factor**

- Obvious

Remark 2

- User may declare device D-K **trusted**:
  - User authenticates from D-K with **password only**

- S can decide **autonomously** to occasionally **require second factor** anyway
  - D-K connects from an anomalous geographic location
  - D-K was declared trusted long time ago
  - …

Hhmmm…

- User may declare device D-K **trusted**:
  - User authenticates from D-K with **password only**

- Attacker knows U, PWD-U@S
- + **steals trusted device** D-K
  - Or **physical access** for some time

- Attacker can access S

Remark 3

- User may declare device D-K **trusted**:
  - User authenticates from D-K with **password only**

- U must:
  - **Protect** access to D-K with password **different** from PWD-U@S (as soon as D-K is declared trusted)

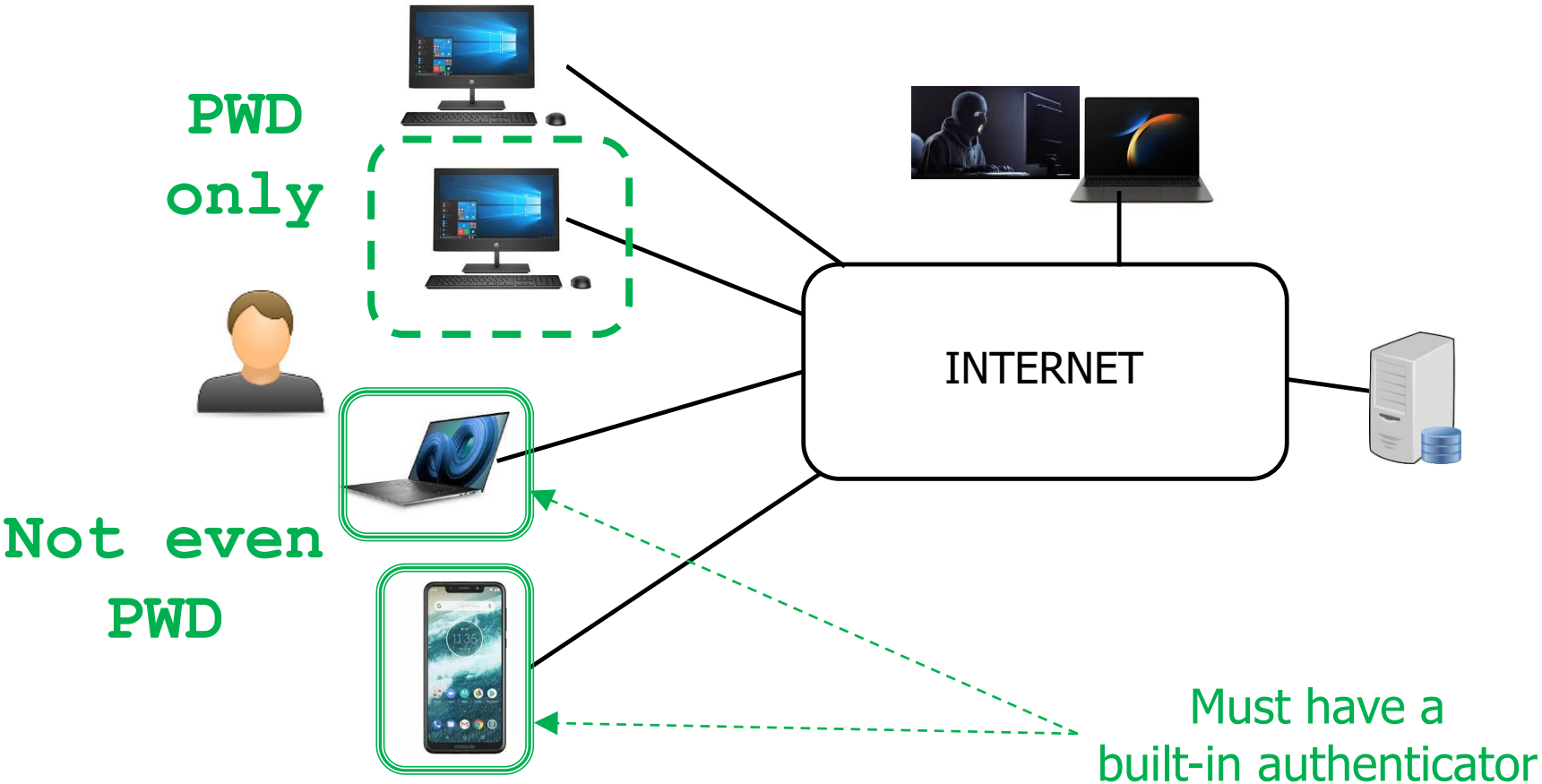
  - **Revoke** trusted status of D-K asap (in case D-K is lost / stolen)

Passwordless Device

Passwordless Device: Functionality

- User has declared D-K **trusted**:
  - User authenticates from D-K with password only
- ...and **passwordless**:
  - User authenticates from D-K **without any password** (!)
- **Requirement**: D-K must have a "**built-in authenticator**"
  - Fingerprint reader
  - Face recognition

Passwordless Device: Example

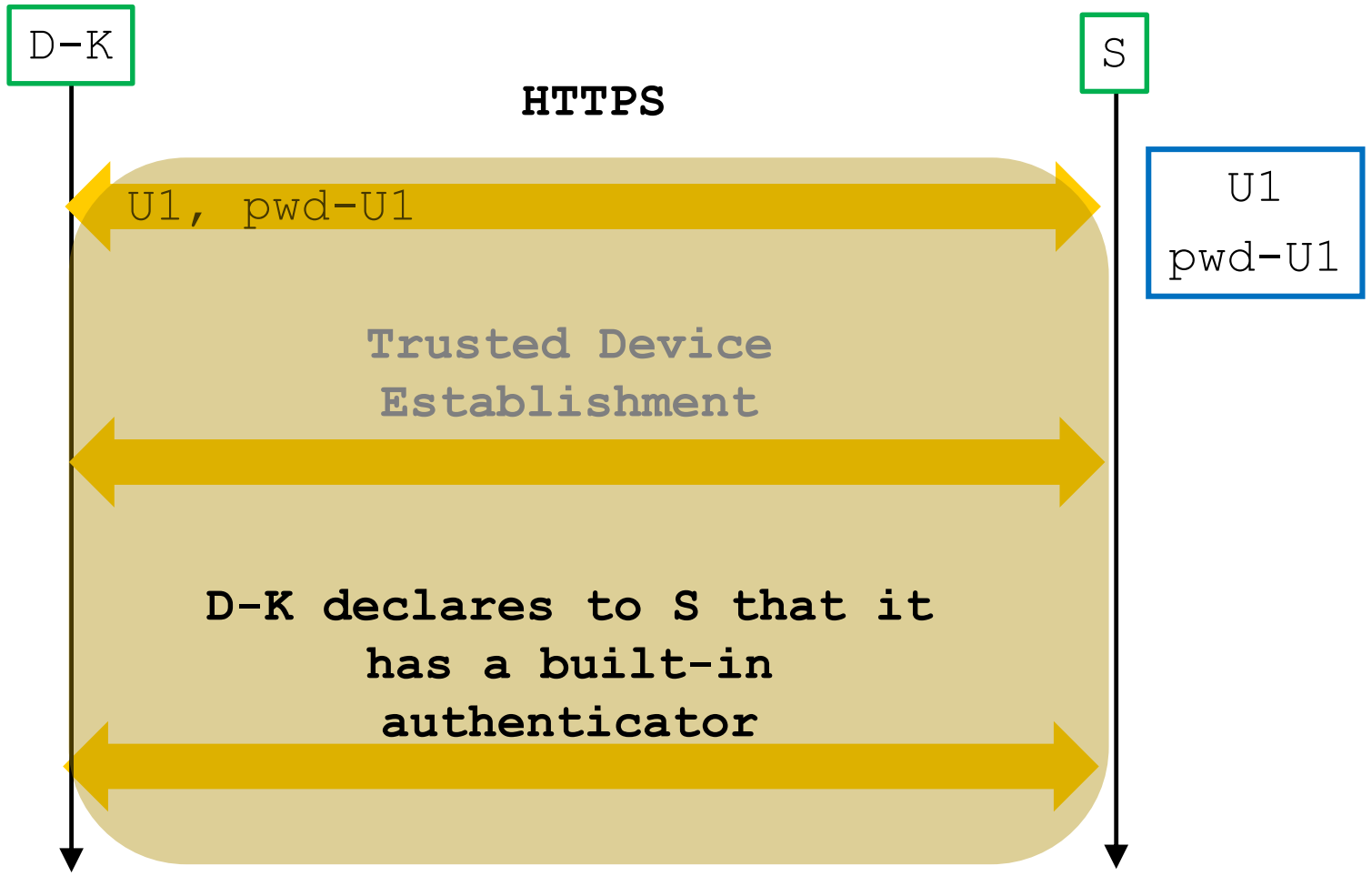


Remark

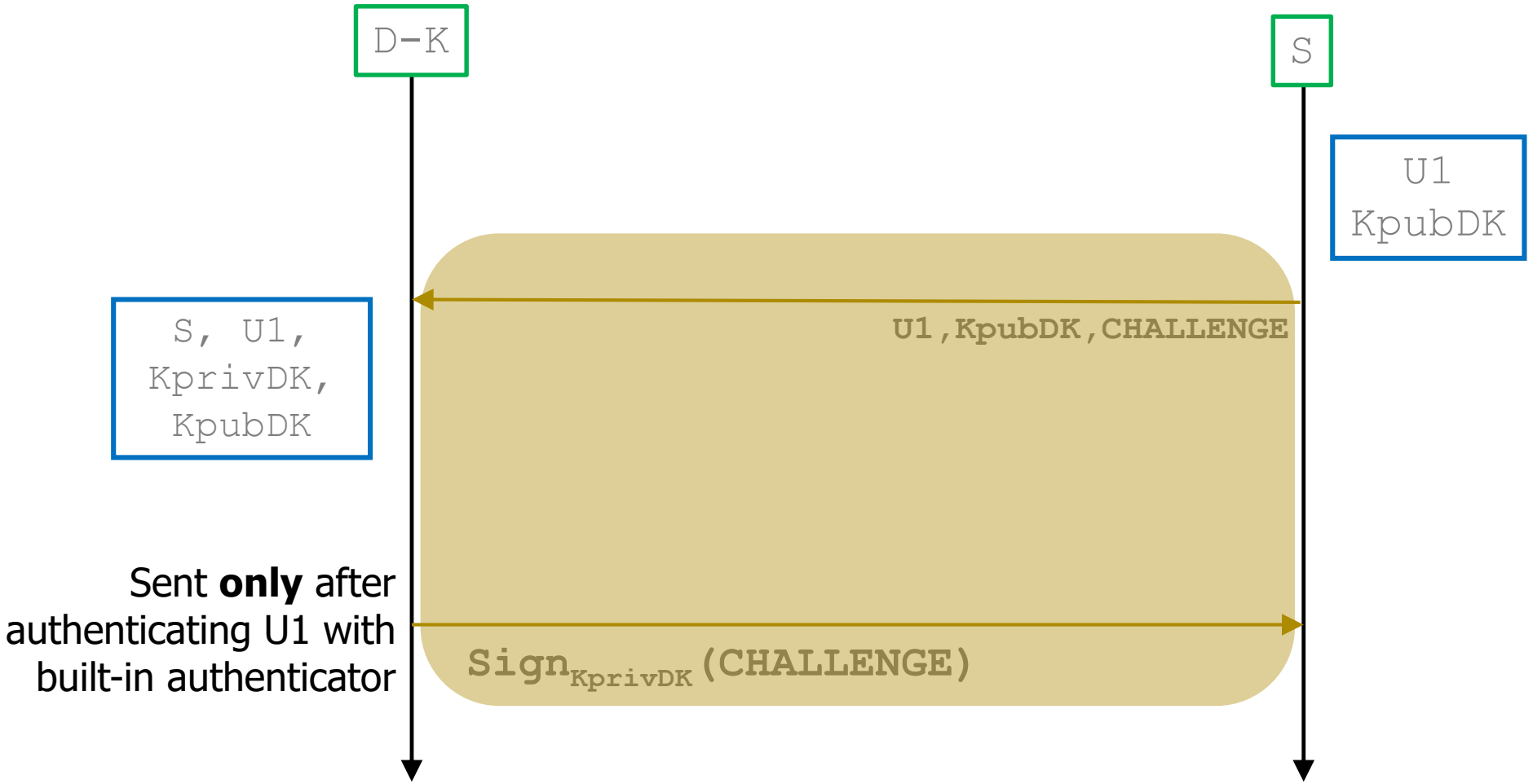
- User declare D-K **trusted** and **passwordless**:
  - User connects from D-K **without any password (!)**

- Common scenario when:
  - S = bank, SPID-enabled app
  - D-K = smartphone

Passwordless Device Establishment (Outline)



Passwordless Device Login (Outline)



Summary

- Trusted device
  - Device proves knowledge of private key to S
- Passwordless device
  - Trusted device with biometric authenticator

- **Login** from Trusted device
  - User proves knowledge of password **to S**
- **Login** from passwordless device
  - User proves biometric property **to device**

Hhmmm…

- Passwordless device

How can it be secure???

I don't like it

Passwordless is MORE "secure"!

- **Login** from Trusted device
  - User proves knowledge of password **to S**
- **Login** from passwordless device
  - User proves biometric property **to device**

- Passwords are **phishable**
- **No** risk of disclosing password from passwordless device!

Passkeys

Passkeys: Example (I)

Security and Privacy › Your Security ›

About Passkey

Passkeys are a convenient and secure way to sign in to your Amazon account without using a password.

With passkeys, you can sign in to your Amazon account by simply using your face, fingerprint, or the PIN that you use to unlock your device. You will not need to provide your Amazon password to sign in.

Passkeys are secure and convenient sign in options as they:

- Work on most major platforms and browsers. For example, iPhones, Android phones, Apple and Windows desktops.
- Are end-to-end encrypted. Your passkeys and biometric information are never shared with Amazon, making your account safe from phishing attacks or data breaches.
- Allow you to still use your Amazon password to sign in, if you prefer.

Passkeys: Example (II)

The simplest and most secure way to sign in to your Google Account

Passkeys are an easier and more secure alternative to passwords.
They let you sign in with just your fingerprint, face scan or screen lock.

Simple

Passkeys offer a convenient and simple experience that uses your device lock, such as your fingerprint, face, pin or pattern to sign in to your Google Account.

Secure

Passkeys provide the strongest protection. They can never be guessed or reused, helping keep your private information secure against attackers.

Private

Your biometric data, such as fingerprint or face scan, is stored on your personal device and never shared with Google.

Passkeys in a nutshell

- Prevalent terminology:
  - "**Passkey** for S stored on D-K"

  ≡

  - D-K is **trusted** and **passwordless** for service S

- Passkey ≈ KPRIV-DK
- In certain cases, passkey can be **migrated** to other (trusted) devices

Passkey vs Long term Cookie

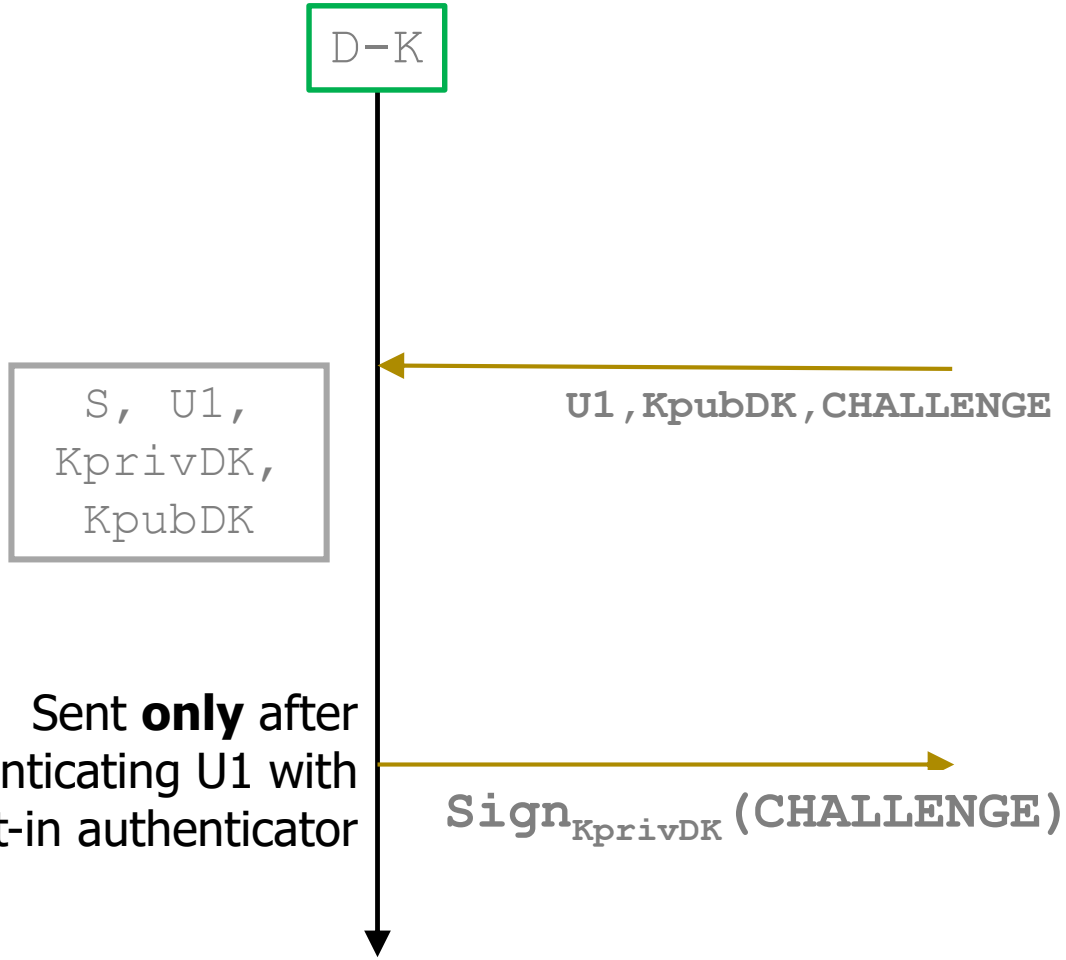
- IF Adversary can read information on D-K
- THEN They are "≈equivalent"

- Cookies:
  - Used after authentication
  - Transmitted in every request
  - Stored on Service

- Passkeys:
  - Used for authentication
  - Never leave the device

Hhmmm…

How can I be guaranteed that D-K will indeed behave this way?



TRUSTED device!

- User point of view:
  1. "I **trust** built-in auth by D-K"
  2. "I **trust** D-K to tell S the truth"