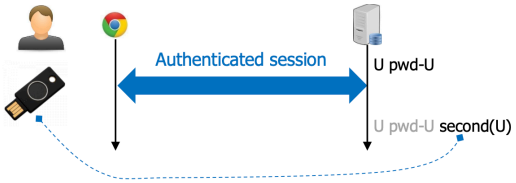
MFA

MFA: Multifactor Authentication

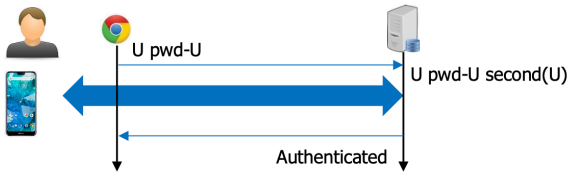
- 2FA: is a two factor authentication with something you know (usually the password) and something you have (like a smartphone or a security key), for us 2FA and MFA will be the same
- Second factors:
  - Smartphone: OTP SMS, OTP authenticator App, Push notifications
  - Security key: through USB, NFC, Bluetooth. This method is much more secure than the smartphone one
  - Both the methods are enormously more secure than using the password only
- Our focus: web app (basic/form over HTTPS), different organizations, extremely relevant in practice
- It is very useful also for authentication for legacy protocols (SMTP, POP, SSH), in the same organization
  - Much less used because is very difficult to deploy on legacy applications

Step 1: linking 2nd factor (once)



- The user selects 2FA in the account options, ovviamente the service must support 2FA
- The message exchange depends on the service and on the 2FA technique
- Penso che quello che serve sapere sia: c'è una sessione autenticata tra user e servizio, lo user decide di voler usare l'autenticazione a più fattori e concorda con il servizio il secondo fattore quale sia (nell'esempio è una chiave in chiavetta)
- Linking examples: google, lastpass, Facebook

Step 2: login (every time)

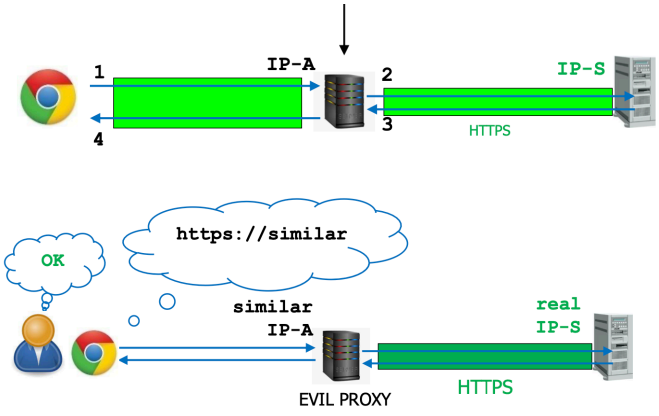


- Lo user fornisce la password per accedere al servizio, dopodiché il servizio rimanda alla verifica del secondo fattore, una volta approvato l'accesso anche con il secondo fattore, lo user è autenticato
- Lo scambio di messaggi dipende dal servizio e dalla 2FA technique
- Examples: unicredit, SPID-PosteID

Threat model: stolen password

- The adversary has the username and the password: the cannot authenticate because he does not have the second factor
- With this threat model, the 2FA solves all the problem, tranne il discorso delle push notifications,

Threat model: Real time Phishing



- **Evil proxy:** proxy specialized for AitM (adversary in the middle)
  - Tool designed to bypass the multi factor authentication
  - Presents to C all the resources of the target website without any local copy, can target many different websites at the same time and the configuration specifies what to modify and what to log
- The user does not detect that he is accessing the wrong URL
- With this threat model, the 2FA solves only the security key method, not the smartphone one (not the OTP SMS/authenticator app nor the push notification)

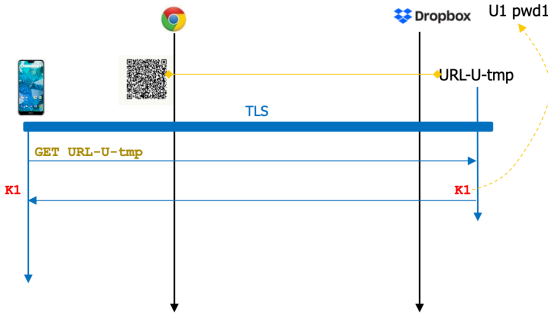
NB MFA is extremely important: it is very effective for very realistic threat models, enabling 2FA is a very high priority defensive investment. MFA is essential: the use of anything beyond the password significantly increases the cost for attacker, which is the reason why the rate of compromise of account using any type of MFA is less than 0.1% of the general population

One time Passwords (OTP)

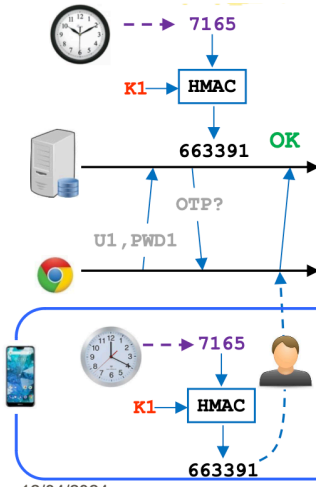
- OTP SMS: a one time password is sent to the user's smartphone, is a 6 digit code with uniform distribution [0, 999999], valid only for one login attempt
- OTP authApp: linking
  - It is a generic authenticator app installed on the user's smartphone. The user has to enable the 2FA on the service, choose the AuthApp, install it on the smartphone and then link the account of the service with AuthApp
  - The login occurs though a one time password generated by the AuthApp, same as the SMS OTP
  - A service might use only its own Authenticator App (come poste ID), conceptually is the same thing

OTP AuthApp: implementation

- AuthApp linking requirement
  - The user has to have a private key K1 generated by the service and securely sent to AuthApp upon activation
  - If the AuthApp is linked to several services (cosa comune), for every service there is an account name and a private key associated



-authApp linking:
-Il servizio manda al client il URL-U-temp, instaurano una connessione TLS (criptata quindi), una volta fatto questo il servizio può mandare allo user la chiave riferita a quel servizio da salvare nell AuthApp
-Di base non c'è una comunicazione diretta tra servizio e AuthApp, quindi come fa questa a generare un codice che il servizio accetta?
-TOTP (time-based OTP): OTP(t)=ENCRYPT_K(t-T0) dove T0 è convenzionalmente il tempo 0



- Viene fatto l'HMAC del clock locale utilizzando la chiave condivisa e confrontato con il codice che viene ricevuto dallo user (che ottiene il codice dall'authapp che fa lo stesso procedimento)
- Problema: i clock potrebbero non essere perfettamente sincronizzati e i messaggi scambiati hanno una certa latenza, quindi questo discorso non può funzionare in pratica, ma viene calcolato:
- The one time password changes every 30 seconds (the actual algorithm is more complex)

$$OTP(t) = ENCRYPT_K(t - T0 \;/\; DX) \qquad // DX = 30 s$$

OTP ATTACKS

- Threat model:
  - Stolen password → solved
  - Real time phishing → not solved (l'attaccante fa finta di essere il servizio e inoltra il second factor authentication al servizio al quale lui vuole accedere, un po' come NTLM con il man in the middle che impersona il server)
  - Vishing: voice phishing
    - Chiamare a far leggere il codice OTP per poter accedere al servizio per conto dell'utente che non capisce e legge il codice

NB OTP does not solve phishing, but makes it much more costly to attacker (but still remains a real danger)

- Excursus su come i messaggi contenenti gli OTP dovrebbero essere un minimo informativi nel senso in cui dovrebbero contenere info su a cose serve quel codice (almeno il servizio)
- Other threat models:
  - OTP SMS: if the attacker knows the password
    - and the smartphone has a malware → the attacker can read, forward and delete the SMS
    - And there is a SIM swap (fraudulent SIM change) → the phone number fraudulently taken by another SIM
    - And there is a SMS routing attack → SS7 phone protocol weakness → the SMS is sent to the attacker
    - Realistic and not solved threat model

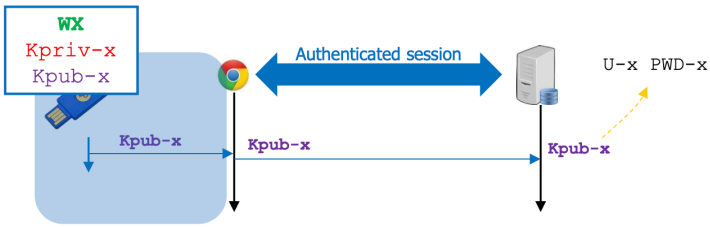
MFA

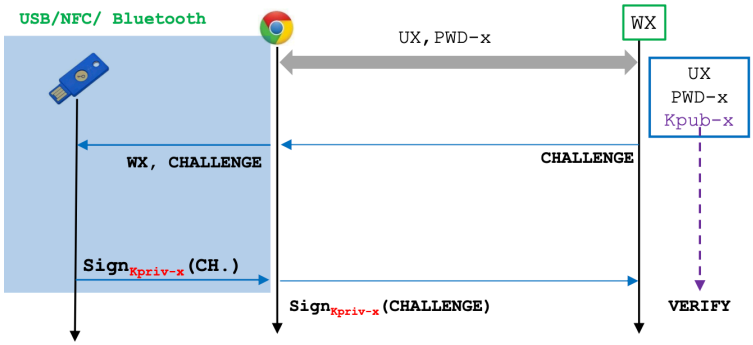
- OTP AuthApp: it the attacker knows the password
  - and the smartphone has a malware → AuthApp does not grant any screenshot rights to any other app
  - And there is a SIM swap (fraudulent SIM change) → OTP do not travel across phone networks
  - And there is a SMS routing attack → SS7 phone protocol weakness → OTP do not travel across phone networks
  - Realistic and solved threat model
- SMS vs AuthApp: it's time to start the move away from the SMS and voice Multi factor Authentication mechanisms. However, that MFA is essential
- OTP privacy implications:
  - OTP SMS: the service must know the user phone number and might abuse of this information's
  - OTP AuthApp: the service does not need to know the user phone number

---

Security keys

- SecKey must be close to the Browser
- Linking requirement: praticamente nella chiavetta (per esempio) c'è una lista di DNS name con associato il nome utente e la coppia di chiavi pubblica e privata. Quindi c'è una security key associata a diversi servizi
- Username omitted for easy description
- Implementation:
  1. User authenticates to WX with UX and PWD-X
  2. SecKey generated <Kpriv-x, Kpub-x> to be used only with WX
  3. SecKey sends Kpub-x to WX securely
  4. WX associated Kpub-x with UX
- Login:





-Real flow more complex
-Dal browser vengono mandati username a password al servizio, il quale ha associata una chiave publica per quello user autenticato, il servizio manda un challenge al browser che lo inoltra alla SecKey, la SecKey, ricevendo nome del servizio e challenge, lo firma criptando il CH con la sua chiave privata associata al servizio, dopodiché questo viene inoltrato al servizio che decripta il pacchetto con la chiave pubblica associata e verifica che il CH corrisponda a quello inizialmente inviato
-Key facts: DNS-name is authenticated with HTTPS, decision to sing are taken by the SecKey, not by the user

- Open standardas very complex…there is no direct communication service-second factor (FIDO2)

Security key attacks

- Threat models
  - Stolen password: the adversary has U,PWD, but does not have SecKey so cannot authenticate → problem solved
  - Real time Phishing: SecKey does not sign the challenge if the DNS name is not in its table (NB decision to sing is taken by SecKey and not by the user)→ problem solved
- Unsolved threat models (out of scope)
  - Attacker with valid certificate for S name: browser cannot discriminate between real and fake services
  - Attacker has a malware on the browser device: the malware can alter/forge Browser/SecKey traffic

NB 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 a set of service name must be also very difficult

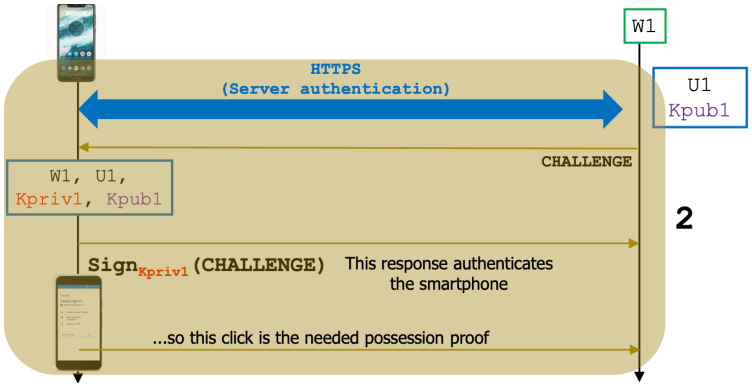
MFA

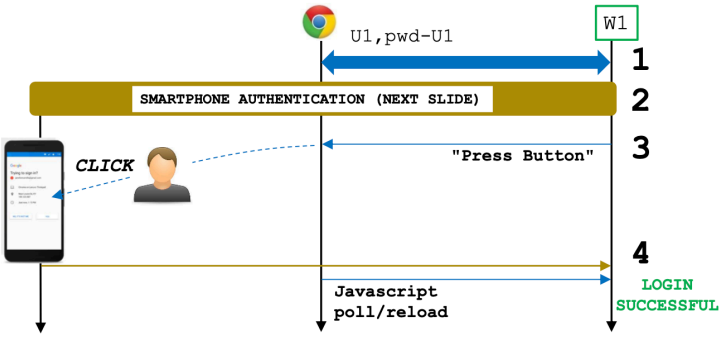
- Attacker may be the manufacturer: if and when we realize it, certain SecKey can no longer be trusted, service must be able to know who the manufacturer and product ID are
- Sets of services might collude to link the respective user identities: service cannot identify which specific SecKey it is interacting with

- Smartcard:
  - A smartcard is actually a SecKey for only one predefined service, it is based on the same principle and with similar defensive strength
  - It is used as a legally binding identity proof in some countries, used in certain intra-organization settings, dedicated reader might be needed

Push notifications

- Every service has its own app where it sends notifications
- Smartphone push notification: the smartphone must be close to the user
- Login outline
- Key problem: the service should connect to the smartphone, how to know the IP address (usually private and dynamic) and how to make sure it is indeed the smartphone of the user

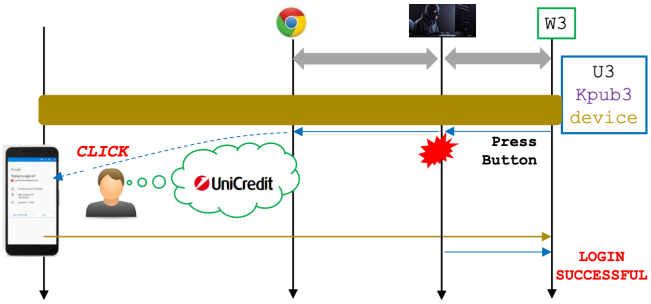




-Solution: every smartphone continuously polls a cloud service, connects as a TCP client and checks whether there is any notification. Upon 2FA linking, the smartphone generated a Kpriv-Kpub keypair, upon login, the smartphone proves knowledge of Kpriv
-There is an HTTPS authenticated connection, the service has a Kpub associated with the user, the service sends a challenge to the smartphone, which signs with its private key the CH and responds to authenticate itself

Push notifications Attacks

- Threat models:
  - Real time phishing: not solved → praticamente l'attaccante fa il man in the middle e cerca di connettersi a un servizio, il quale manda una push notification, se lo user apre al notifica e autorizza l'accesso (non sapendo cosa sta facendo), allora l'attaccante comunque entra nel servizio
  - The user might not understand which browser is logging in
  - THE USER MUST READ AND BE CAREUL (exactly the phishing issue)



MFA bombing: (technique for credential access)

- Adversaries may continuously repeat login attempts in order to bombard users with MFA push notifications, SMS messages, and phone call, potentially resulting in the user finally accepting the authentication request in response to MFA fatigue
  - Unbelievable but it may work

MFA summary of limitations

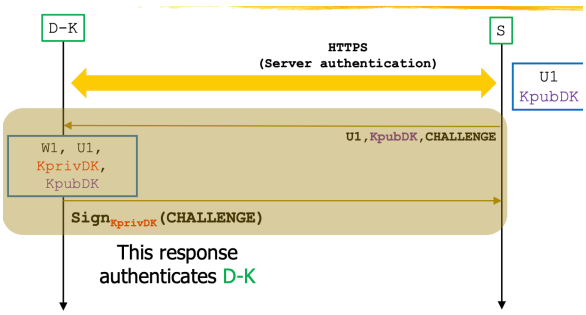
- 2FA is checked only at the beginning of a session (Webapp login or workstation logon if if was deployed in such a setting) ⟶2FA does not defend against attacks during an authenticated session (stealing of authentication cookie, pass the hash, pass the ticket)
- Use alterante authentication material: (lateral movement)
  - Alternate authentication material is legitimately generated by systems after a user of application successfully authenticates by providing a valid identity and the required authentication factors
  - Catching alternate authentication material allows the system to verify an identity has successfully authenticated without asking the user to reenter authentication factors
  - 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
- Everything but SecKey does not protect from phishing/voice phishing, o anche il controllo di chi sto autorizzando e per cosa lo sto autorizzando
- OTP AuthApp is better than OTP SMS (malware, SIM swap, SMS routing)
- Everything is possible for the attacker after the initial access
- Limitation for recovery procedures for loss of second factor ⟶ be careful on the email password

Passwordless login (PassKey)

- A special handling for 2FA@S is possible for certain devices ⟶ entering only the password or not even that
- Terminology not uniform across vendors/consortia
- There is a myriad of different scenarios (we will see a simplified and general description), mapping a specific case is not easy
- Key terms: trusted device, Passwordless device (or login), PassKey

Trusted device

- functionality: the scenario is a user that has activated 2FA on a service S, the user may declare a certain device D-K trusted, in this situation the user authenticated form D-K providing the password only (much more easier to use)
- Implementation:
  - S behavior: if the user is at D-K, then password only, else password and 2nd factor
  - New problem for S: device authentication: S must behave differently depending on the device being used by the user
  - Basic idea: S and D-K agree on a Kpriv_Dk and Kpub_DK key pair, S stores the user name D-K and the Kpub_DK, D-K stores his Kpriv_DK, so at the login time D-K proves to S knowledge of Kpriv_DK (similar to smartphone authentication)
  - Device authentication ⟶



- Remarks:
  - The user can declare a device D-K trusted only while the user is logged in form D-K with password and 2nd factor
  - S can decide autonomously to occasionally require the second factor anyway (like if D-K connects from an anomalous geographic location or if D-K was declared trust a long time ago)
  - If the attacker steals the trusted device D-K, the attacker can access to S
    - In this case the user must revoke the trusted status of D-K asap and protect access to D-K with a different password from the one it was before stealing

Passwordless device

- Functionality: same scenario as before
  - The user authenticates from D-K without any password

MFA

- Requirements: D-K must have a built in authenticator → fingerprint reader or face recognition
- A common scenario is when the service is the bank, SPID enabled app and the D-K (trusted and passwordless device) is a smartphone

❑S behavior:
- ❑IF User is at D-K
- ❑THEN ~~Password~~
  Ask D-K "authenticate U with built-in auth"
  - ❑IF D-K answers SUCCESS
  - ❑THEN No password
  - ❑ELSE Password
- ❑ELSE Password and 2nd factor

-Implementation:
-The user point of view is that he has to trust the built in auth made by D-K and trust D-K to tell S the truth

---

Summary

- Trusted device: proves knowledge of the private key to the service
- Passwordless device: is a trusted device with biometric authenticator
- Login from a trusted devices: the user can prove knowledge of private key to S or prove knowledge of the password to S
- Login from passwordless device: the user can prove knowledge of private key to S or prove biometric property to the device
- Passwordless is more secure: the passwords are phishable, there is no risk of disclosing passwords from a passwordless device
- Passkey in a nutshell:
  - Prevalent terminology: D-K is trusted and passwordless for service = Passkey for S stored on D-K
  - Passkey è praticamente Kpriv-DK
  - In certain cases, passkey can be migrated to other trusted devices

---

Loss of second factor

- What to do if the second factor is stolen or lost
  - People are often reluctant to use 2FA because they fear the permanent logout
- Common scenario: One TRUSTED still available
  - The user access from the available trusted device, revoke the old second factor and define a new one
  - There is no lockout, only a transient problem: unable access from device that is not trusted anymore
- Worst case scenario: No TRUSTED left (uncommon)
  - Every service supports a 2FA recovery that must be setup in advance → no lockout
- 2FA recovery:
  - OTP sent to one destination of our choice (recovery phone/email) → the phone must not be the second factor recovery (it would not be a 2FA recovery), the email has to be protected by a secure password (different from the service password)
  - Backup codes: OTP downloaded in advance
- Recovery codes: is the most secure 2FA recovery, but you have to print them, store them safely, remember were they are and having them available if needed

Do not fear account lockout, just review and maintain 2FA recovery into carefully, do not use a recovery password identical to the service password