

Replacing passwords with FIDO2

Who am I?

- Nils Amiet
- Research team @



Passwords are a problem

“71% of accounts are guarded by password used on multiple sites” - **TeleSign**

“62% of breaches involved the use of stolen credentials, brute force or phishing” - **Verizon**

“The vast majority of data breaches are caused by stolen or weak credentials” - **Kaspersky**

“86% of users would like to replace work-related password with fingerprint recognition technology if given the option” – **Secret Double Octopus**

“There is a consensus on the need to move away from passwords” - **Forrester**

FIDO2

- Developed by FIDO Alliance
 - FIDO = Fast IDentity Online
- 2 specifications
 - FIDO2 = WebAuthn + CTAP
- Addresses multiple authentication use cases
 - **Passwordless** (single factor)
 - **Multi factor** (passwordless + PIN or biometrics)
 - Second factor (CTAP1 / U2F)
 - Backwards compatible with U2F (Universal 2nd Factor) standard

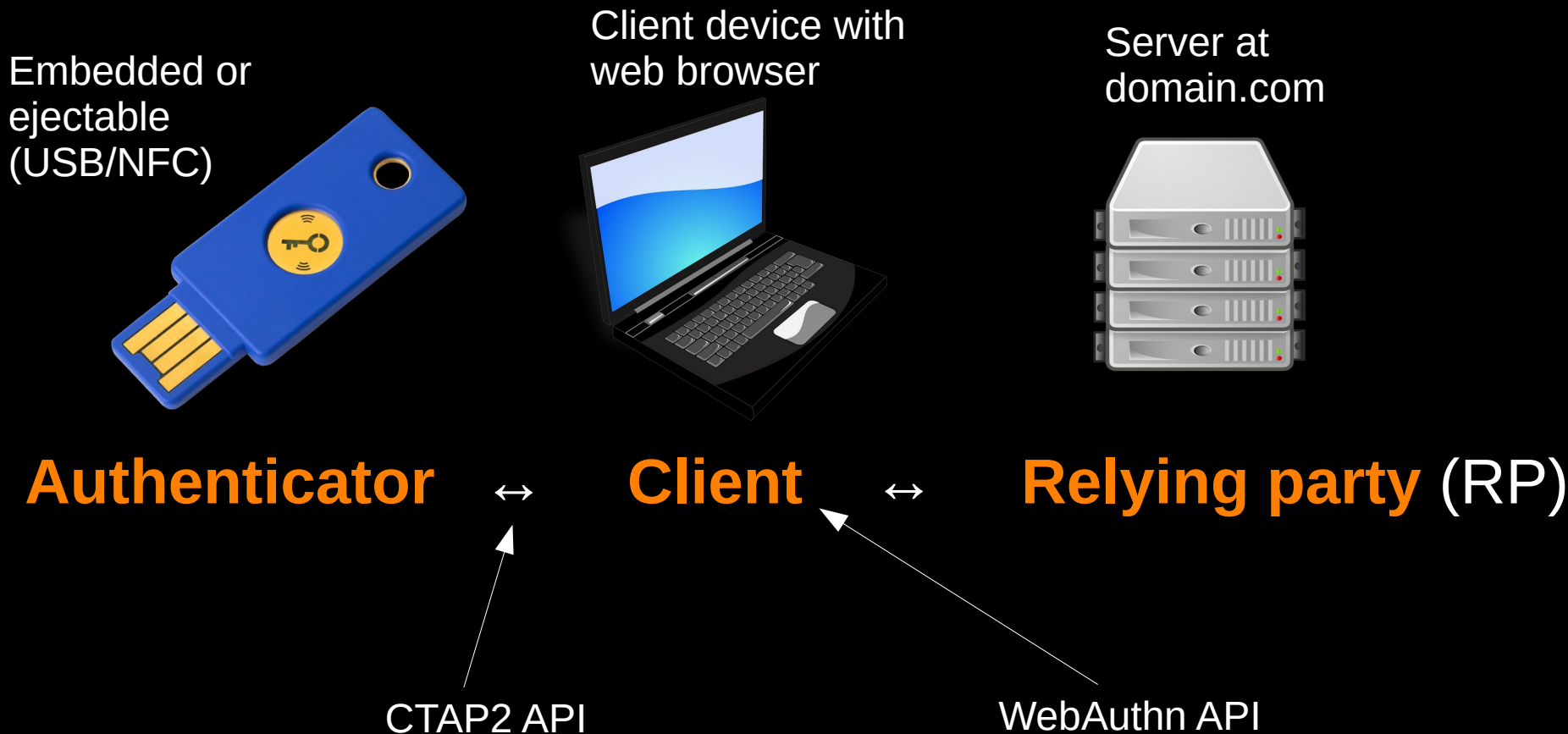
FIDO Alliance founded by:



Today, members also include:



Overview



Purpose of these 2 specifications

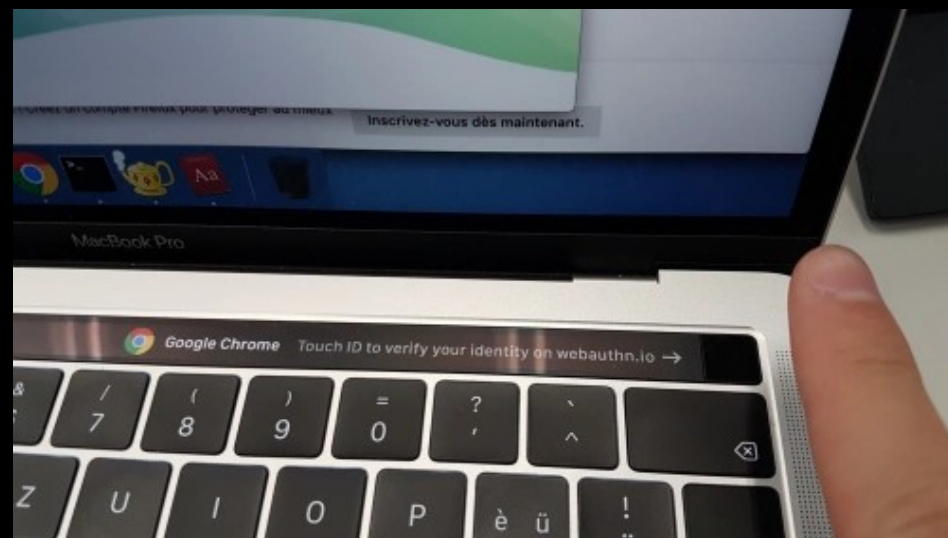
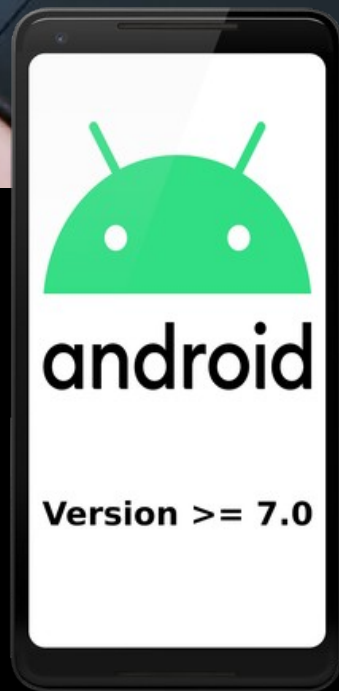
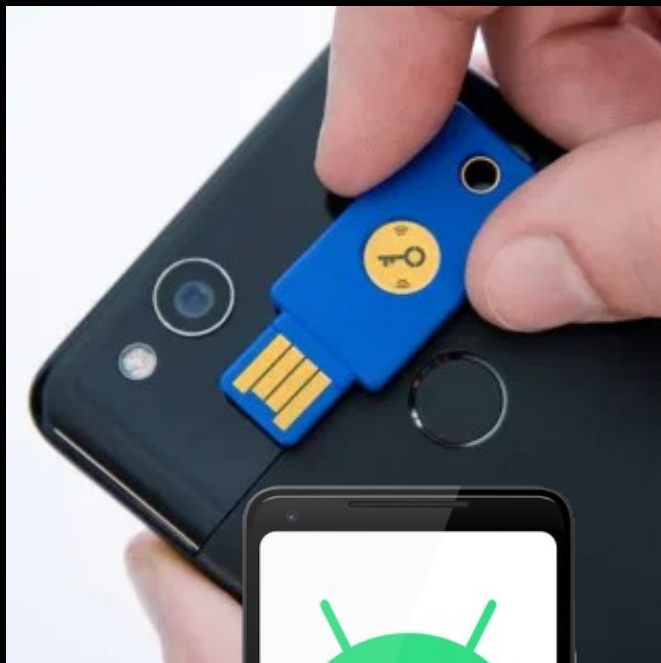
WebAuthn

- **WebAuthn**
 - For web browsers
 - Javascript API
- **CTAP (Client To Authenticator Protocol)**
 - API between client and authenticator
 - Standard for all ejectable authenticators
 - Messages encoded in Concise Binary Object Representation (CBOR) format, RFC 7049
 - Also for desktop apps, command-line apps

Authenticators

- 2 authenticator types
 - **Platform authenticator** (Embedded/non-ejectable)
 - Your smartphone
 - Your laptop/desktop
 - **Roaming authenticator** (Ejectable)
 - A security key (USB or NFC)
 - Many vendors
 - Open source: Solo Key, see also: OpenSK
 - Entry price about \$20 USD





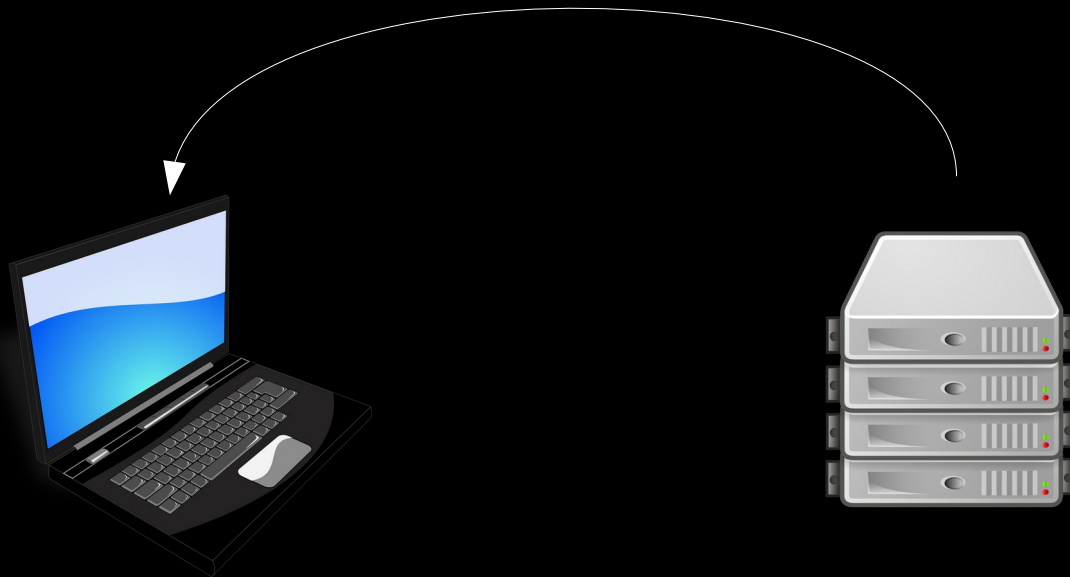
How does it work?

Registration



Registration

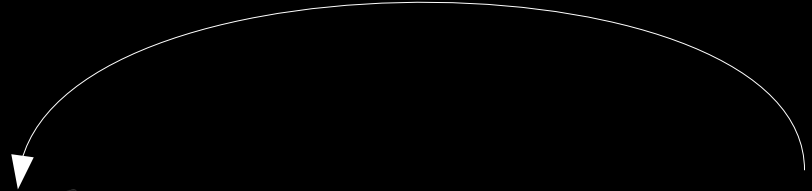
1) Serve **registration** page
that includes JavaScript



Registration

2) User clicks
register button

1) Serve **registration** page
that includes JavaScript



Registration

2) User clicks
register button

1) Serve **registration** page
that includes JavaScript

3) Call authenticator



Registration

2) User clicks
register button

1) Serve **registration** page
that includes JavaScript

3) Call authenticator

4) User presence (UP) check,
User verification (UV) check (optional)



Registration

2) User clicks
register button

1) Serve **registration** page
that includes JavaScript

3) Call authenticator

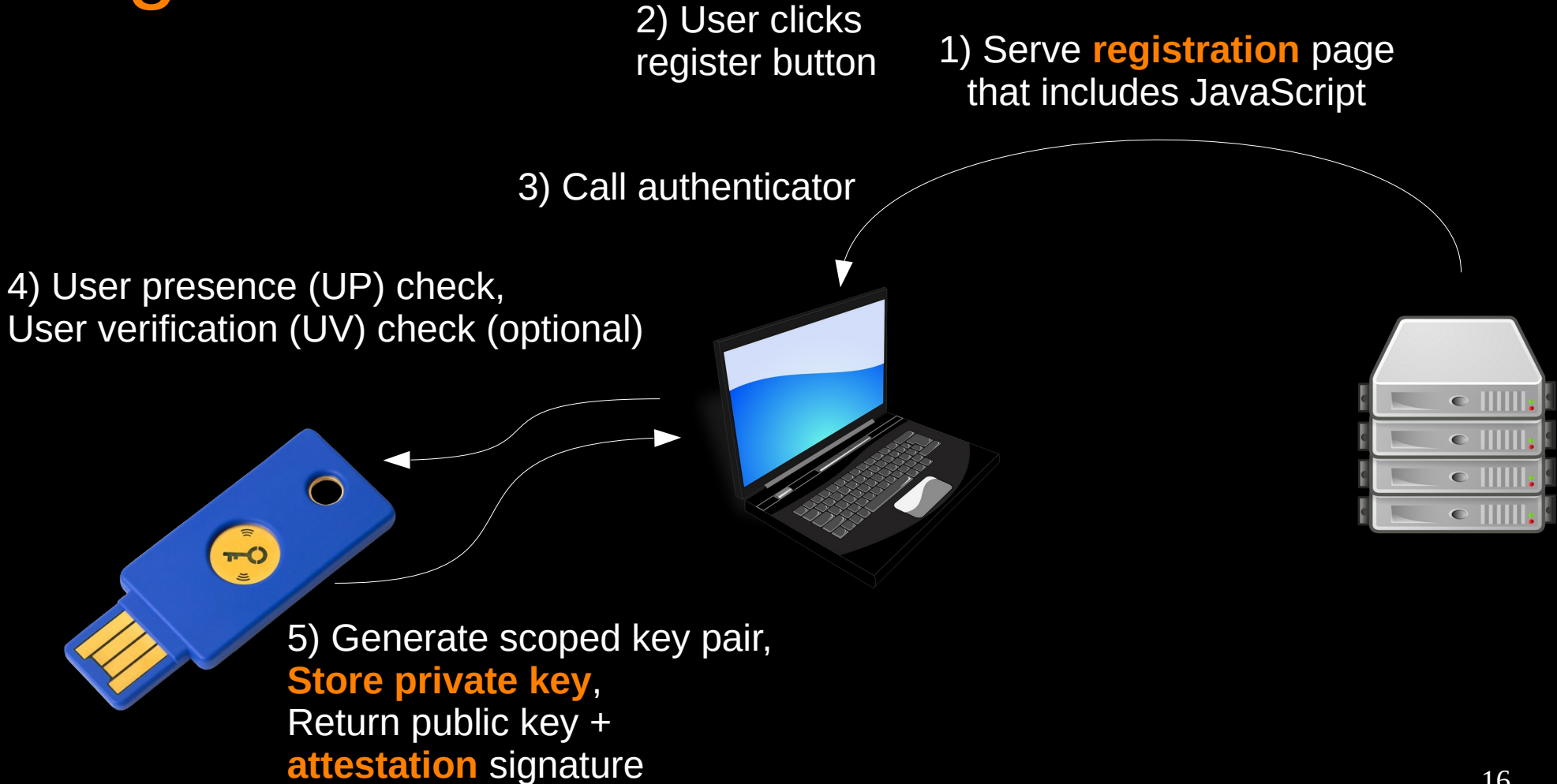
4) User presence (UP) check,
User verification (UV) check (optional)



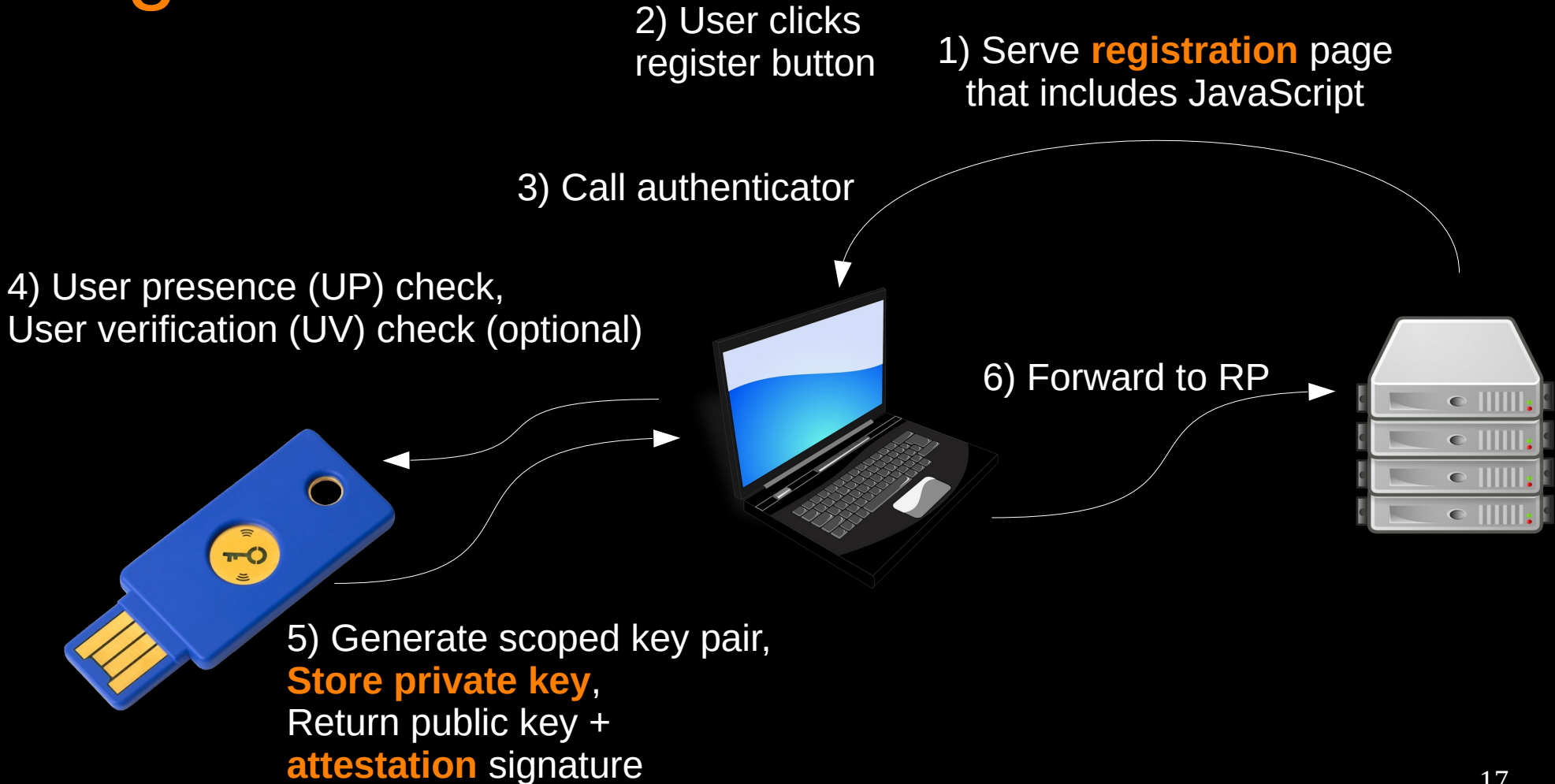
5) Generate scoped key pair,
Store private key,
Return public key +
attestation signature



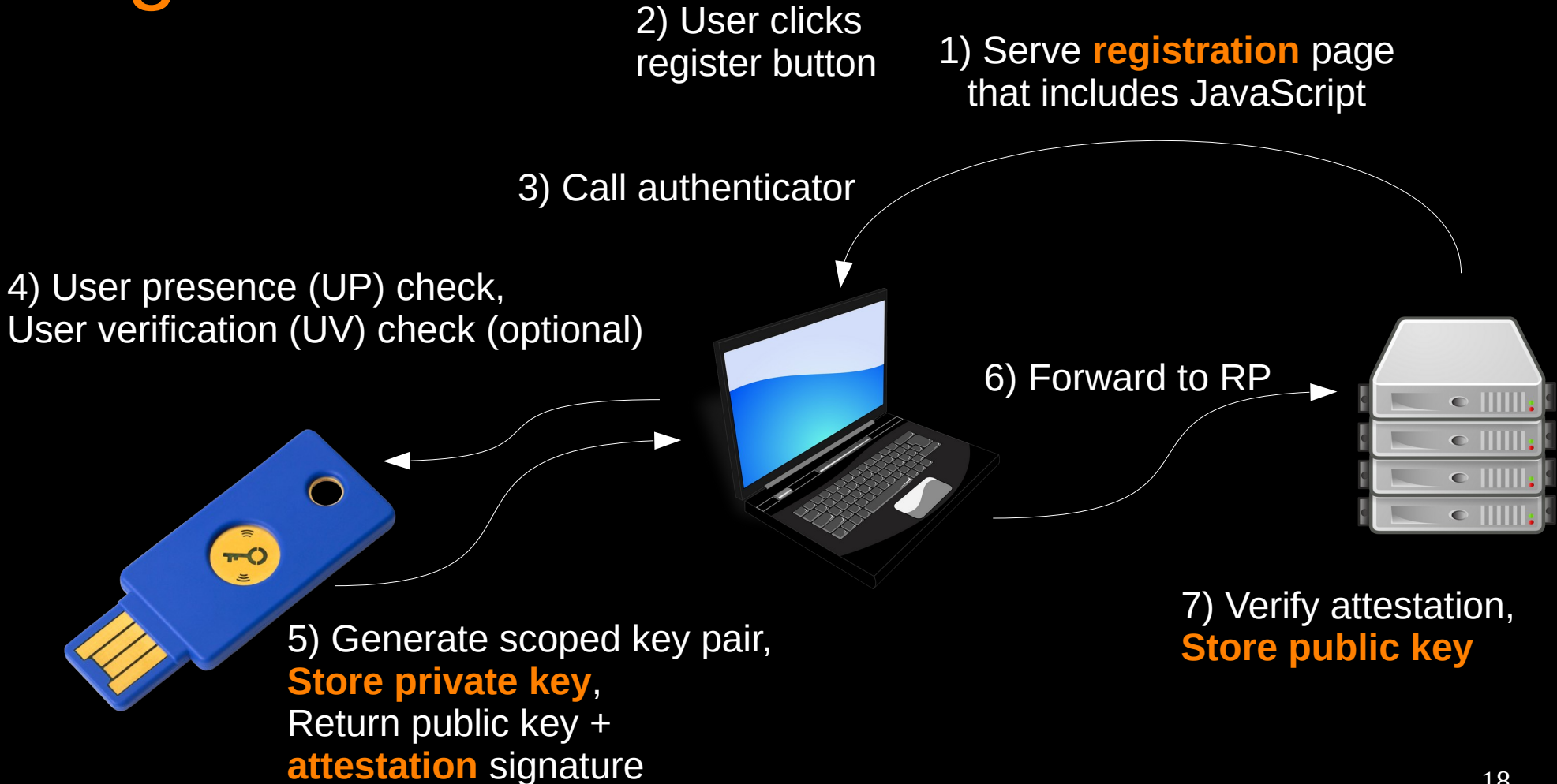
Registration



Registration



Registration

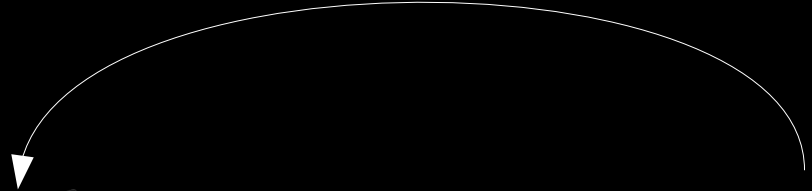


Authentication



Authentication

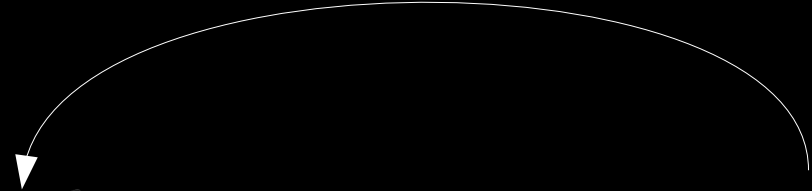
1) Serve **sign-in** page
that includes JavaScript



Authentication

2) User clicks
sign-in button

1) Serve **sign-in** page
that includes JavaScript



Authentication

2) User clicks
sign-in button

1) Serve **sign-in** page
that includes JavaScript

3) Call authenticator



Authentication

2) User clicks
sign-in button

1) Serve **sign-in** page
that includes JavaScript

3) Call authenticator

4) UP + UV checks



Authentication

2) User clicks
sign-in button

1) Serve **sign-in** page
that includes JavaScript

3) Call authenticator

4) UP + UV checks

5) Return **assertion** signature



Authentication

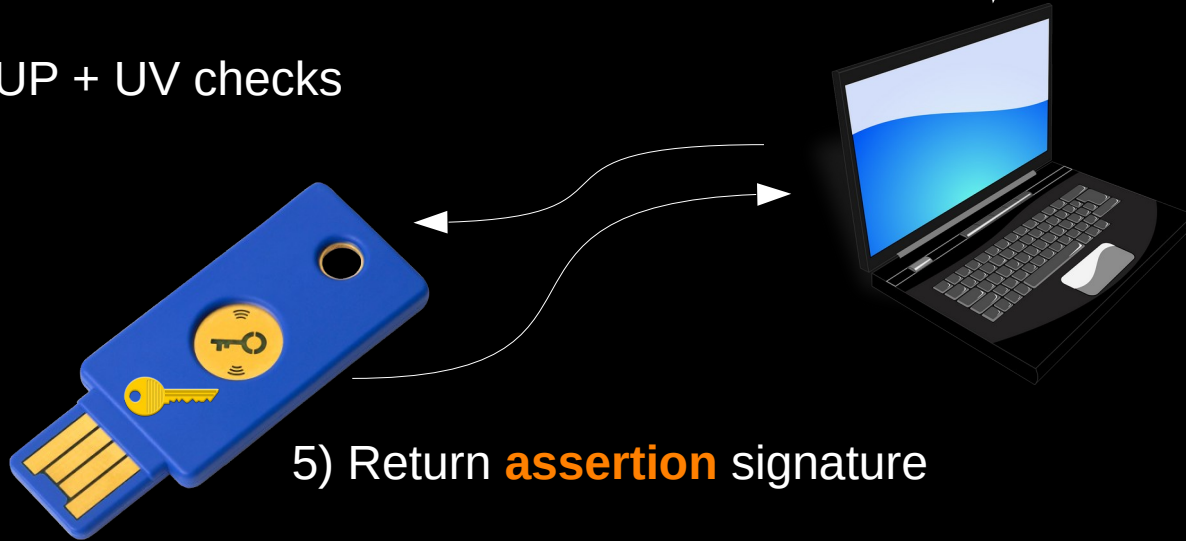
2) User clicks
sign-in button

1) Serve **sign-in** page
that includes JavaScript

3) Call authenticator

4) UP + UV checks

5) Return **assertion** signature



Authentication

2) User clicks
sign-in button

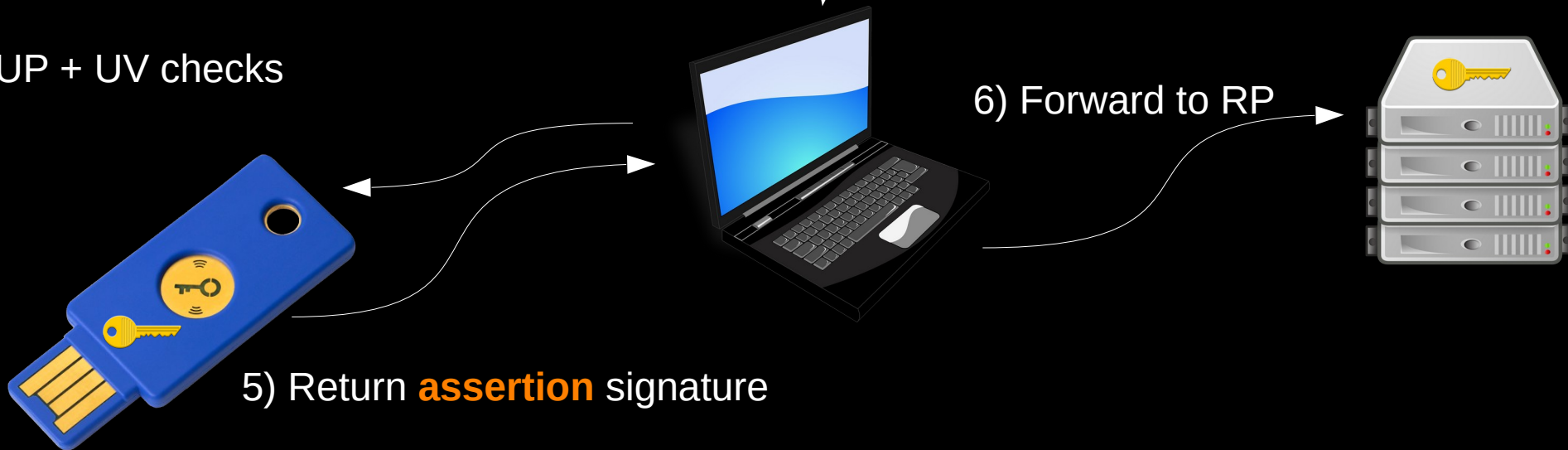
1) Serve **sign-in** page
that includes JavaScript

3) Call authenticator

4) UP + UV checks

5) Return **assertion** signature

6) Forward to RP



Authentication

2) User clicks
sign-in button

1) Serve **sign-in** page
that includes JavaScript

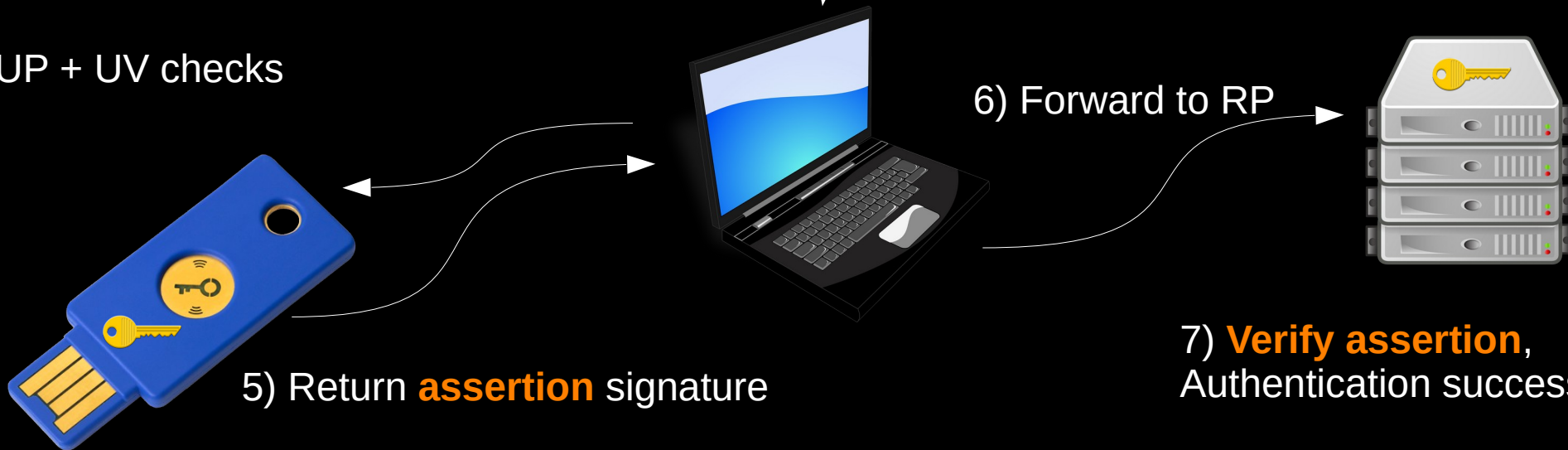
3) Call authenticator

4) UP + UV checks

5) Return **assertion** signature

6) Forward to RP

7) **Verify assertion**,
Authentication successful



Actor responsibilities



Authenticator main responsibilities

- **User presence** check
 - Tap authenticator
- **User verification** check (if supported)
 - PIN or biometrics
 - Yes, UV check is performed **client-side** (!)
- **Generate and store credentials**
- Produce signatures (**attestations** and **assertions**)



Client main responsibilities

- Act as **proxy** between authenticator and relying party
- Few other things
 - Example: if multiple accounts
 - Implement account selection logic



Relying party main responsibilities

- Verify attestations
- Verify assertions
- Check initial options (UV, ...)
- **Store public keys**
- Generate and verify challenges (prevent replay attack)
- Make authentication decision:
 - Authenticator characteristics and compromise status
 - Clone detection



Attestations



Why do we need attestations?

- RP can **trust authenticator is what it claims to be** by:
 - Verifying attestation signature using pre-established chain of trust
- If trusted, RP can:
 - Verify authenticator security level
 - Build an authenticator acceptance policy
 - Trust authenticity of authenticator data (including UV flag)



What is an attestation signature?

- Attestation is **optional** (!)
- Signature created during registration
- Signature is computed over:
 - **Authenticator data** (generated public key, AAGUID, UP, UV, etc.), and
 - Hash of **client data** (challenge, server origin, etc.)
- Multiple **attestation types**
 - Each attestation type provides a different trust model



Attestation types

- Basic attestation
- Self attestation
- Attestation CA (AttCA)
- ECDAA
- None



Basic attestation

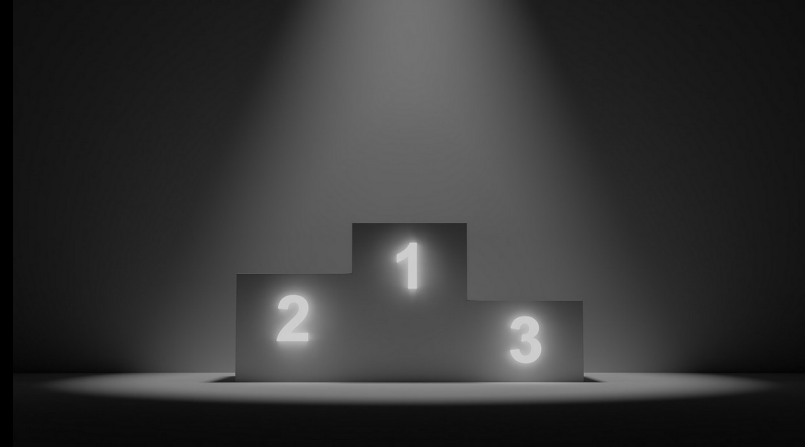
- **Attestation private key** (burned in at factory)
 - Attestation certificate (contains public key)
 - Also certificate chain
- **Privacy vs compromise impact:** same attestation private key for ~100'000 authenticators of same model
 - Sweet spot for privacy and security
 - Ensure users cannot be tracked
 - Limit impact in case of attestation key compromise
- **Key compromise impact**
 - Cannot distinguish original authenticators and fake ones using leaked key
 - Authenticators registered before compromise are not impacted

Self attestation

- Generate key pair
- Sign using generated private key
 - Similar to self-signed certificates
- **Does not prove** that the authenticator is what it claims to be (!)
 - Only proves ownership of public key

Best attestation type?

- On paper, ECDA for strict security policies
 - Banking, government
- ECDA secure implementation is non-trivial
- Not every RP requires this security level
- In practice, may use **Basic attestation**, or not care about attestation at all
- Does not make a lot of sense to use complex attestation type with authenticators that do not provide strong protection against physical attacks



Assertions
(not attestations)



What is an assertion signature?

- Signature **created during sign-in**
- Produced using generated private key
- Is verified by RP using corresponding public key
- Also computed over:
 - Authenticator data
 - Hash of client data
- Many possible public key algorithms

APIs overview

WebAuthn operations

- **navigator.credentials.create()**
 - Parameter: PublicKeyCredentialCreationOptions
 - Delegates credential creation to authenticator
 - Receives attestation in response
- **navigator.credentials.get()**
 - Parameter: PublicKeyCredentialRequestOptions
 - Asks authenticator for signature
- Extensions
 - appid (compatibility with U2F)
 - uvm (RP wants to know which UV method was used)
 - ...



CTAP2 operations

- **authenticatorMakeCredential** (0x01)
 - Generate a new key pair
 - Return an attestation signature and a public key
- **authenticatorGetAssertion** (0x02)
 - Return an assertion signature using existing private key
- Other operations
 - Get info
 - Client PIN
 - Reset
 - **CTAP 2.1** new operations
 - Bio Enrollment (e.g. fingerprint)
 - Credential management
 - Vendor commands: 0x40 to 0xBF
- Extensions
 - hmac-secret
 - Example: password manager

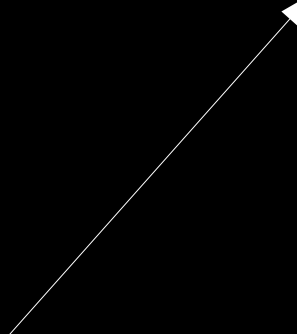


FIDO Metadata Service (MDS)

Metadata service

- Authenticator **vendors publish info** about their product there
 - Security features, characteristics
- **RPs download entries** periodically
 - Build trust store using those entries
 - Be alerted if product X's attestation key is compromised
 - Must request access token, manually renew yearly
- https://fidoalliance.org/specs/fido-security-requirements-v1.0-fd-20170524/fido-authenticator-metadata-requirements_20170524.html

What info is there in the MDS?

- List of entries
 - AAGUID
 - Status reports
 - Url of entry => download
 - Downloaded entry
 - Description
 - **Attestation root certificates**
 - UV methods
 - Key protection
 - CryptoStrength
 - Supported public key algorithms
 - ... see FIDO metadata statements documentation
- 

Security measures



Security measures

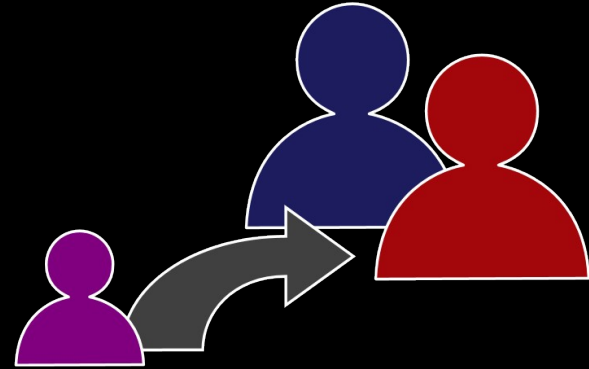
- **Authenticator cloning detection** (signature counter)
- Failed PIN attempts
 - 3 failures => must unplug and replug device
 - Avoid malicious device locking
 - 8 failures => must reset device
 - Erases all previously generated keys stored on authenticator
- **Scoped** credentials
 - Keys are linked to an origin (domain) => Avoid fishing
- Physical theft protection (PIN or biometric)



Token Binding

- RFC 8471
- Bind security tokens (e.g session cookie) to a TLS connection
 - **Prevents session hijacking**
- Not really used in practice (!)
 - Web browser support is lacking
 - Edge (EdgeHTML-based versions) supports it, Chrome dropped support
- WebAuthn: Token binding ID can be specified in client data
- <https://groups.google.com/a/chromium.org/forum/#!msg/blink-dev/OkdLUyYmY1E/w2ESAeshBgAJ>

Adoption



FIDO2 support

- **Passwordless use case**

- Microsoft.com
 - Set user-agent to Edge on Windows
- <Your site here soon>

- **2FA use case**

- Many sites
- Easy to upgrade from U2F to FIDO2 2FA

- **CTAP2-only**

- OpenSSH >= 8.2 supports private keys stored on CTAP2 compatible devices
 - `ssh-keygen -t ecdsa-sk -O resident`

“I-mark” logo can be displayed to tell users your service supports FIDO2



WebAuthn

WebAuthn

- Chrome
- Firefox
- Safari
- Edge
- Also on mobile



CTAP2

- Android
 - USB, NFC
- iOS
 - Lightning, NFC (iPhone 7 or later)
- Windows, MacOS, Linux
 - USB

Platform Authenticators

- Any Mac with Touch ID (touch bar)
- Any Android 7.0+ smartphone
- Any Windows machine with Windows Hello

Implementation

- Python-fido2
- Many existing libraries on Github
 - Both for client and server-side
- Pull entries from Metadata service (!)
- Do not blacklist vendors
 - Authenticator acceptance policy should be based on security characteristics (if any)
 - https://developers.yubico.com/WebAuthn/WebAuthn_Developer_Guide/WebAuthn_Readiness_Checklist.html

Is the password problem solved?



Problem solved?

- No need to choose/remember/change passwords anymore
- Protocol prevents password re-use
- Invulnerable to phishing
- Strong protection against network attacks

Takeaways



FIDO2 best practices



- Make sure to **register a backup authenticator**
 - In case of physical theft, loss, your house burns, etc.
 - You won't be locked out of your account if you have a backup method to sign-in
 - You can sign-in with the backup authenticator and revoke the stolen authenticator
- Set a PIN or biometric on your authenticator
 - The attacker still needs your PIN or fingerprint to sign-in



Password vs PIN

- “But you’re replacing the password with a PIN!”
- Password is sent over network and is vulnerable to all network attacks
- PIN is local
 - PIN does not need to be changed as often
- PIN cannot be brute forced
- Alternatively, use biometrics



FIDO2 is still young

- CTAP 2.1 is on the way
- Few websites support passwordless FIDO2
 - Please add FIDO2 support to your service
 - Use attestations if possible



More resources

- <https://research.kudelskisecurity.com>
 - FIDO2 blog post series
- Live demo
 - <https://webauthn.io>
- <https://loginwithfido.com>
- <https://webauthn.guide>
- <https://fidoalliance.org/fido2>

Thank you

- Questions?