This work is distributed under the CC BY-SA license, unless stated otherwise.

Hardware Security

3. Authenticators

Ján Jančár

Summer 2026

Today

Passwords → Authenticators → WebAuthn/Passkeys

Passwords

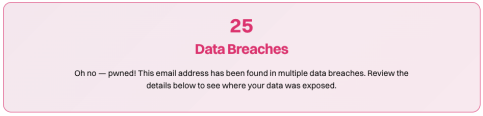
and what is wrong with them

The password problem

- Passwords
- Problems?
  - Low entropy
  - Password reuse
  - Phishing
  - Password leaks: haveibeenpwned.com

Have I Been Pwned

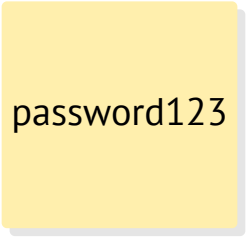
Have I Been Pwned



- Aggregates data & password leaks
- You can check your email address
- I do
- You can also check your **passwords**
  - Yes, entering your passwords into a third party site feels wrong
  - It uses k-anonymity so you should be fine

Humans?

- Solution 1: Just fix humans
  - Not how this works
  - List of the most common passwords



Rank	Password	Uses
1	123456	21,627,656
2	admin	21,030,012
3	12345678	8,274,408
4	123456789	5,673,712
5	12345	3,950,777
6	password	3,545,119
7	Aa123456	2,520,728
8	1234567890	1,418,939
9	Pass@123	1,210,039
10	admin123	1,087,247

Password managers

- Solution 2: Password manager
- High entropy if generated
- No reuse if used correctly
- Phishing-resistant
- How?
  - Host/domain check
  - Can be bypassed by user, sometimes

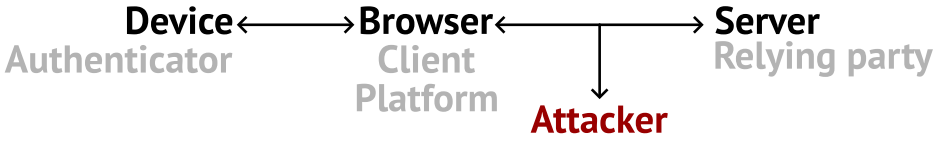
One-time-password via text message

- 2nd factor: require something in addition to a password
- Make that something a one-time-password and pass it *out-of-band*
  - Via some different channel (that the attacker **does not control**)
- Solution 3: text message one-time-code
- Problems?
  - SIM swapping
  - Needs phone reception
- Variant: Email the code
- Problems?
  - Email used to be unreliable (sometimes still is)
  - If attacker has access to email, it is usually game over anyway

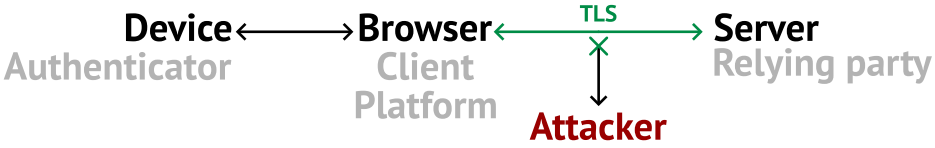
Model

Device
Authenticator
Browser
Client
Platform
Server
Relying party

Model


Device
Authenticator
Browser
Client
Platform
Server
Relying party
Attacker

Model


Device
Authenticator
Browser
Client
Platform
TLS
Server
Relying party
Attacker

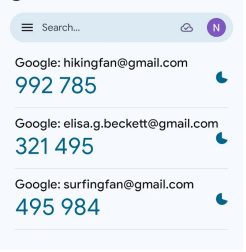
Model

hell.com
hello.com
Device
Browser
Server
Authenticator
Client
Platform
Relying party
Attacker

HMAC-based One-Time Password

- Solution 4: HOTP
- RFC 4226
- Share a key between server and client's device (once)
- Derive one-time-passwords (for each auth)
- Counter based (both server and client have state that evolves)
- 3 operations:
  - **Register:** Server generates master key K, sends to client's device (e.g., QR code)
  - **Generate:** Client advances counter C and generates OTP as $\mathrm{HMAC}_K(C)$ then does some truncation and converts to 6 digit number $\pmod{10^6}$
  - **Verify:** Server advances counter C, generates OTP the same way as client, and compares to the received one.
- What can go wrong? (not security-wise)
  - Desynchronization: Client advances, server does not
  - Solved by checking window of counters (C-1, C, C+1)
- What can go wrong security wise?
  - Phishing: User enters OTP into a phishing website, attacker relays
  - Eavesdropping/Shoulder-surfing
  - Client/server compromise

Time-based One-Time Password



- Solution 5: TOTP
- RFC 6238
- Time-based instead of counter-based
- Both client and server use their clock (assume it is synchronized)
- 3 operations:
  - **Register:** Server generates master key K, sends to client (e.g., QR code)
  - **Generate:** Client computes time step $T = \lfloor \text{now}/30\ \text{seconds} \rfloor$, generates OTP as $\text{HMAC}_K(T) \pmod{10^6}$
  - **Verify:** Server computes same OTP using its clock, allows small time window tolerance (±1 step)
- What can go wrong? (not security-wise)
  - Clock drift: Client and server clocks differ
  - Solved by allowing window of ±1–2 time steps
  - Server can keep track of clients time drift
- Security-wise?
  - Same as HOTP: OTP can be relayed by phishing site
  - Eavesdropping/Shoulder-surfing
  - Time window increases attack surface slightly
  - Client/server compromise

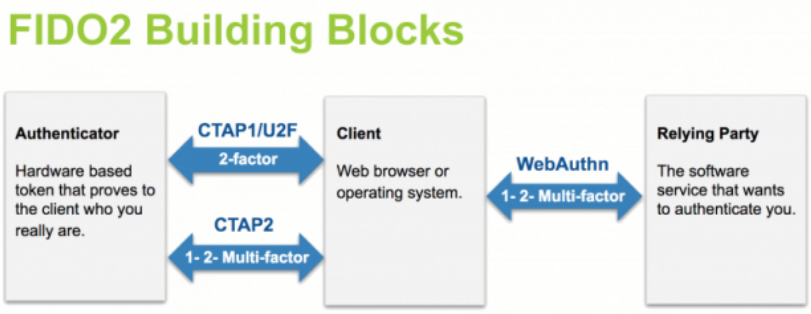
The phishing problem

- User can be tricked to enter an OTP into a phishing website
- How do we solve the phishing problem?
  - What can we do to the OTP to stop this?
  - Prevent the user from entering anything at all **and**
  - Somehow, embed into the OTP, information about the website
  - TLS!

Authenticators

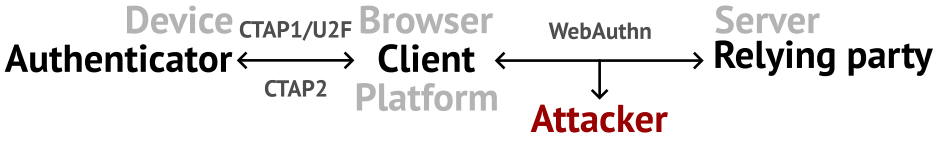
and how they help us

Some terms



- FIDO2 is the overarching project for several components/standards
- WebAuthn is what happens between the **Relying Party** and the **Client**
- CTAP1/U2F and CTAP2 is what happens between the **Client** and the **Authenticator**
- A **credential** is data one entity presents to another in order to *authenticate* the former to the latter

Some terms



- FIDO2 is the overarching project for several components/standards
- WebAuthn is what happens between the **Relying Party** and the **Client**
- CTAP1/U2F and CTAP2 is what happens between the **Client** and the **Authenticator**
- A **credential** is data one entity presents to another in order to *authenticate* the former to the latter

U2F

and the second factor beginnings

U2F (Universal 2nd Factor)

- Now obsolete, but we will start with it
- 2nd factor **device/authenticator** with a button
- **Basic idea**:
  - Use public-key cryptography for authentication
  - Private key safely stored on the **authenticator**
  - **Relying party** only has the public key
  - **Authenticator** signs challenges after user presses a button

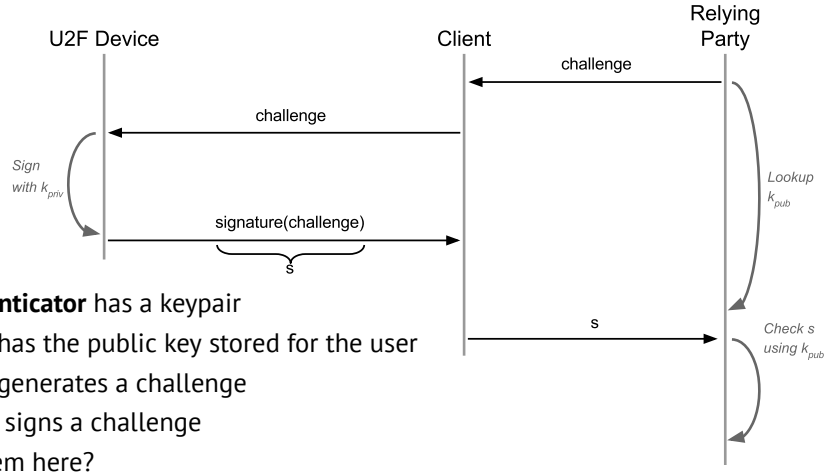
https://commons.wikimedia.org/wiki/File:FIDO2_USB_token.png

https://solokeys.com/cdn/shop/products/MG_2266_5388bffa-4d87-4b75-bae0-28bf9c12190d_1032x1032.jpg

U2F (Universal 2nd Factor)

Idea 1 (public key credentials)

https://developers.yubico.com/U2F/Protocol_details/auth_flow1.svg



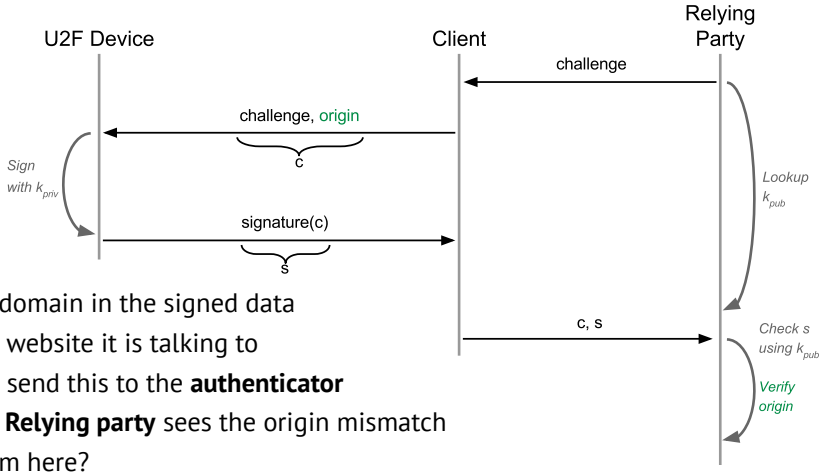
Authentication

- Assume the **authenticator** has a keypair
- The **relying party** has the public key stored for the user
- The **relying party** generates a challenge
- The **authenticator** signs a challenge
- What is the problem here?
  - Attacker can still phish. They will just relay the challenge and signature

U2F (Universal 2nd Factor)

Idea 2 (phishing/MiTM resistance)

https://developers.yubico.com/U2F/Protocol_details/auth_flow2.svg



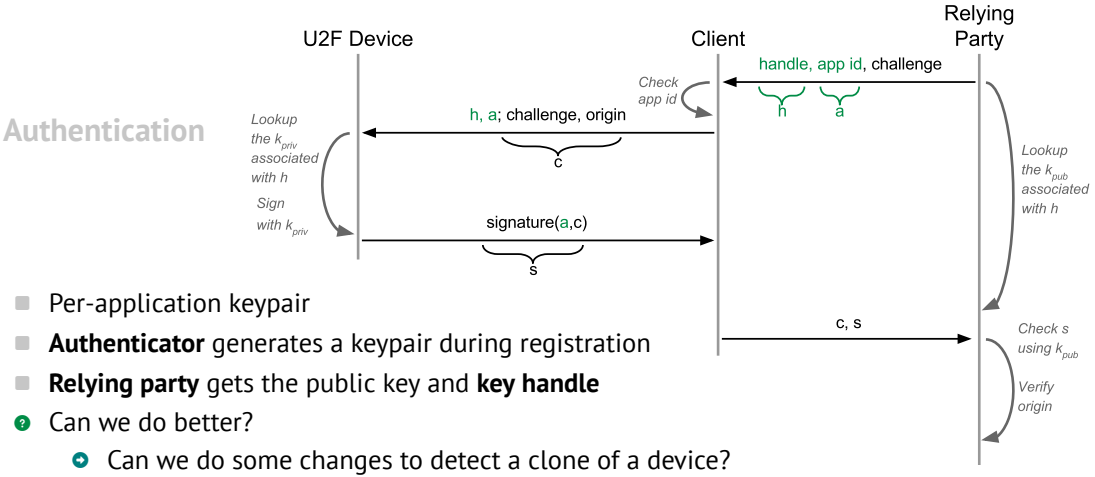
Authentication

- Include the origin domain in the signed data
- **Client** knows what website it is talking to
- **Client** is trusted to send this to the **authenticator**
- No more phishing, **Relying party** sees the origin mismatch
- What is the problem here?
  - Fixed keypair: No privacy between services

U2F (Universal 2nd Factor)

Idea 3 (Better privacy/unlinkability)

https://developers.yubico.com/U2F/Protocol_details/auth_flow3.svg

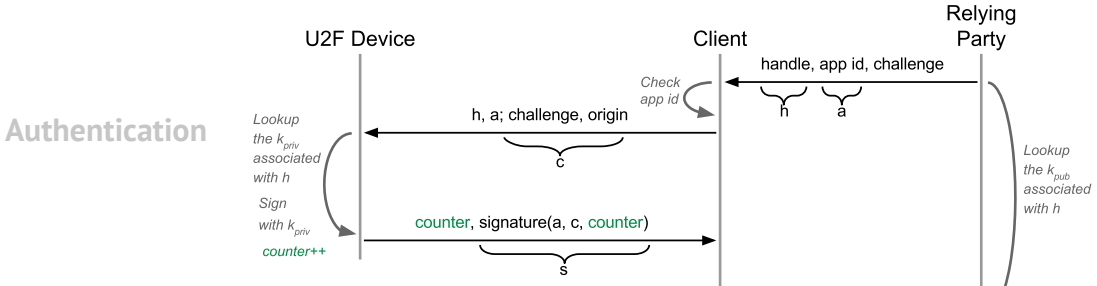


- Per-application keypair
- **Authenticator** generates a keypair during registration
- **Relying party** gets the public key and **key handle**
- Can we do better?
  - Can we do some changes to detect a clone of a device?
  - Can we give the **relying party** some way to assure the user has a genuine device?

U2F (Universal 2nd Factor)

Idea 4 (Clone detection)

https://developers.yubico.com/U2F/Protocol_details/auth_flow4.svg

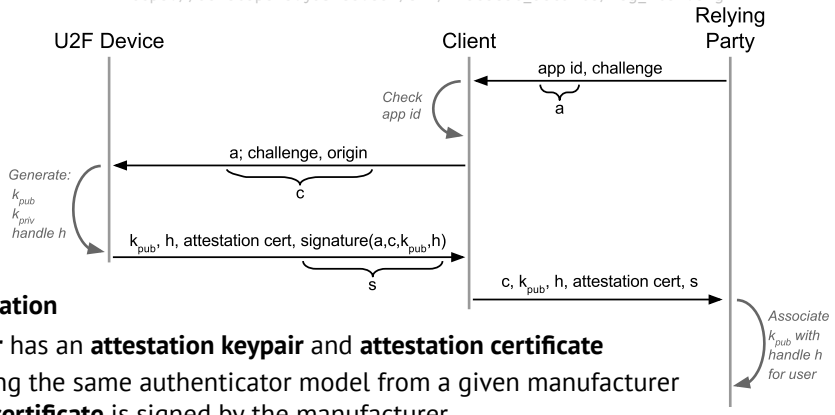


- Add a counter to the **authenticator** and the **relying party**
- **Authenticator** increments counter on every authentication
  - One **global** counter
- **Relying party** checks that it is not lower than what it expects
  - Warns the user if it is lower than expected

U2F (Universal 2nd Factor)

Idea 5 (Device attestation)

https://developers.yubico.com/U2F/Protocol_details/reg_flow.svg



- Add to the **registration**
- The **authenticator** has an **attestation keypair** and **attestation certificate**
  - Shared among the same authenticator model from a given manufacturer
  - **Attestation certificate** is signed by the manufacturer
- The **authenticator** signs the enrollment flow message using the **attestation key**
- What is the problem here?
  - If one device leaks the attestation private key, the whole authenticator model can be spoofed

U2F (Universal 2nd Factor)

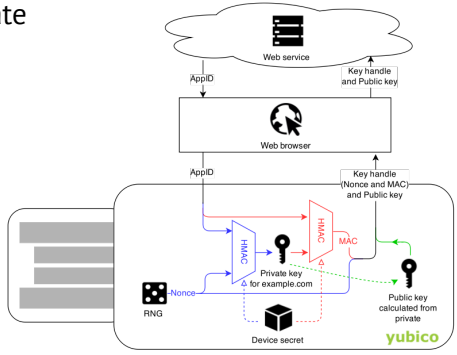
Cryptography

- ECDSA for the signatures (on `secp256r1`)
- SHA256 for hashing

U2F (Universal 2nd Factor)

Key handle

- The key handle is generated by the **authenticator** and then stored by the **relying party**
- The **authenticator** can offload storage of its private key and be almost stateless
  - Store only one **global** symmetric key *k*
  - Generate a keypair on registration
  - *handle* = AEAD_{k}(private key, relying party ID)
  - During authentication, decrypt and use
- Or, the **Yubikey** (older) approach
  - Store only one **global** symmetric key *k*
  - Sample a random *nonce* on registration
  - private key = HMAC_{k}(*nonce*, relying party ID)
  - *mac* = HMAC_{k}(private key, relying party ID)
  - *handle* = (*nonce*, *mac*)
  - During authentication, re-derive private key and verify *mac*



ℹ This is why the clone detection counter is **global**

U2F (Universal 2nd Factor)

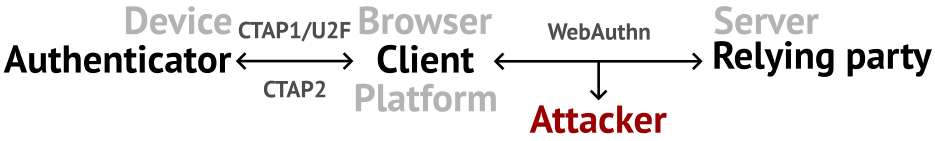
What is missing?

- No user verification (only touch, no PIN/biometric)
- No passwordless login (always second factor)
- No credential discovery (server must provide key handle)
- Limited authenticator UX (no account selection)
- Weak attestation trust model (shared attestation keys)
- No secure channel to authenticator

CTAP & WebAuthn

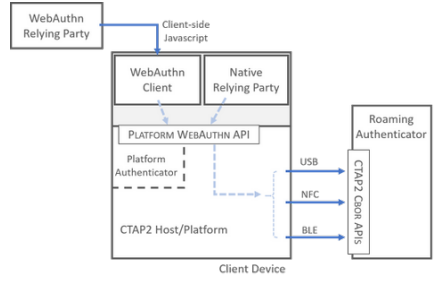
and the modern FIDO2 approach

CTAP & WebAuthn



CTAP2 (U2F++)

- CTAP1 = U2F
- CTAP2 adds
  - Secure channel (ECDH)
  - User verification (PIN/biometric)
  - Credential discovery (discoverable credentials)
- Same as U2F
  - Per-RP key pairs
  - Origin binding (client enforced)
  - Counters for clone detection
- **Roaming authenticator**
  - An authenticator separate from the client
  - Yubikey/Solokey, Google Titan Security Key
- **Platform authenticator**
  - The platform is the authenticator
  - Windows Hello, Apple Touch ID / Face ID, Android



https://learn.microsoft.com/en-us/windows/security/identity-protection/hello-for-business/webauthn-apis

CTAP2

The evil maid attack (kind of)

- You leave your authenticator unattended
- The evil maid comes in and uses it
- It only requires user touch (no verification)
- The **evil maid attack** is slightly different
  - The evil maid alters the device in some undetectable way to later gain access
  - Installs a key-logger to get the password & PIN

https://commons.wikimedia.org/wiki/File:Smedley_maid_illustration_1906.jpg

CTAP2

Idea: Add user verification

- Problem
  - U2F only checks presence (touch)
- Fix
  - PIN / biometric on authenticator

CTAP2

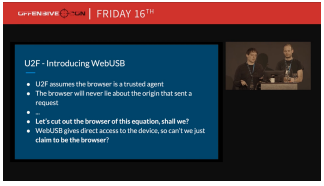
Idea: Add secure channel

- Problem
  - User verification PIN is entered on the client
  - USB / NFC / BLE between is untrusted
- Fix (kind-of)
  - ECDH → encrypted CTAP channel

CTAP2

WebUSB attacks

- U2F and CTAP2 assumes the browser is **trusted** to not lie about the origin
- *"The WebUSB API provides a way to expose non-standard Universal Serial Bus (USB) compatible devices services to the web, to make USB safer and easier to use."* (WebUSB API)
- What can go wrong?
  - Attacker from hell.com can spoof to be from hello.com
- Browsers disallow certain USB devices in WebUSB



https://www.youtube.com/watch?v=pUa6nWWTO4o

CTAP2

Not so secure channel

- *Provable Security Analysis of FIDO2*
- *"Then, we show CTAP2 can only be proved secure in a weak sense; meanwhile, we identify a series of its design flaws and provide suggestions for improvement."*
- *"CTAP2 cannot achieve UF security because in the binding phase it uses unauthenticated Diffie-Hellman key exchange which is vulnerable to MITM attacks."*
- *"we propose ... which replaces the use of unauthenticated Diffie-Hellman in the binding phase with a password-authenticated key exchange (PAKE) protocol."*
- Why did CTAP2 not use a true PAKE?
  - Password-Authenticated Key Agreement
  - Protocols designed for low-entropy shared secrets
  - Establish a shared key and mutual authentication
  - Limit brute-force guesses and prevent MiTM
  - No offline brute-force

CTAP2

Idea: Remove key handles

- Problem
  - U2F requires server-side key handle lookup
- Fix
  - Discoverable credentials (stored on authenticator)
  - **Authenticator** lists credentials
  - **Client** can present **credential** selection UI

Passkeys (Discoverable credentials)

- Credentials stored on the **authenticator**
- Can be synced
  - Implementation-dependent: Google Password Manager, Apple iCloud Keychain, Microsoft Password Manager …
  - Complicated cryptography (1, 2; 1, 2, 3; 1) to ensure end-to-end encrypted credentials
  - Bound to screen lock / Touch ID / Face ID
- Enables passwordless login (1F)

Smartcards

- The authenticators may not look like smartcards
- Often they are
  - FIDO2Applet
  - ledger-u2f-javacard
  - u2f-javacard
- Closed source ones may be as well

https://commons.wikimedia.org/wiki/File:FIDO2_USB_token.png

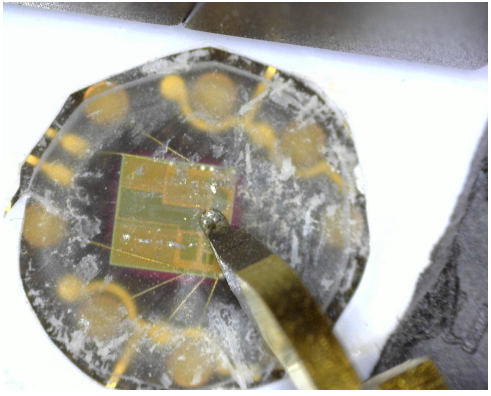
https://solokeys.com/cdn/shop/products/MG_2266_5388bffa-4d87-4b75-bae0-28bf9c12190d_1032x1032.jpg

Smartcards

It is hardware after all

NinjaLab

- A Side Journey to Titan
- Full extraction of the ECDSA signing key from Google Titan
- Electro-magnetic side-channel attack



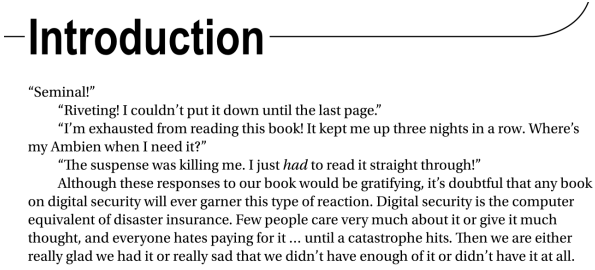
Takeaways

- Understand the evolution from passwords → U2F → FIDO2/Passkeys
- Know threat models and limitations
- Grasp the cryptography & protocols

Enable 2FA where it makes sense for you. Use passkeys.
Backup the **security codes**!

Recommended reading

- For **this** lecture
  - Skim The quest to replace passwords: a framework for comparative evaluation of Web authentication schemes (UCAM-CL-TR-817)
    - Somewhat old but interesting overview of many authentication options and their properties
- For the **next** lecture
  - Read A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security, Chapter 3
    - A quick tutorial to TPMs and their use cases



Introduction

"Seminal!"

"Riveting! I couldn't put it down until the last page."

"I'm exhausted from reading this book! It kept me up three nights in a row. Where's my Ambien when I need it?"

"The suspense was killing me. I just *had* to read it straight through!"

Although these responses to our book would be gratifying, it's doubtful that any book on digital security will ever garner this type of reaction. Digital security is the computer equivalent of disaster insurance. Few people care very much about it or give it much thought, and everyone hates paying for it … until a catastrophe hits. Then we are either really glad we had it or really sad that we didn't have enough of it or didn't have it at all.

References

- webauthn.guide
- WebAuthn and FIDO2 overview (YouTube)
- Yubico: U2F key wrapping
- Yubico U2F docs
- Yubico CTAP docs
- U2F protocol details
- WebAuthn 2.0 standard
- UCAM-CL-TR-817
- A Side Journey to Titan
- Provable Security Analysis of FIDO2
- OffensiveCon18 - Markus Vervier and Michele Orru - Oh No, Where's FIDO?
- Vendors, Disclosure, and a bit of WebUSB Madness
- Chrome Lets Hackers Phish Even 'Unphishable' Yubikey Users