

PV204 Security technologies



Authentication: passwords, OTP, FIDO U2F

IS, 1998

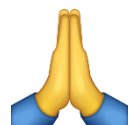


2021



Petr Švenda  svenda@fi.muni.cz  [@rngsec](https://twitter.com/rngsec)

Centre for Research on Cryptography and Security, Masaryk University



Please report any inaccuracies or suggestions for improvements here:

<https://drive.google.com/file/d/1WNMU4fmG7sMMcuHHA-jgXQgG2nWkUCoJ/view?usp=sharing>

CRCS

Centre for Research on
Cryptography and Security

www.fi.muni.cz/crocs



P PetrS

0 👍

Is my password brute-force-able if consists of 9 printable characters?

- **Place/upvote questions in slido while listening to lecture video**
- **We will together discuss these during every week lecture Q&A**

Join at

slido.com

#pv204_2024

COURSE TRIVIA: **PV204_00_COURSEOVERVIEW_2024.PDF**

Basic terms

- **Identification**
 - Establish what the (previously unknown) entity is
- **Authentication**
 - Verify if entity is really what it claims to be
- **Authorization (access control)**
 - Define an access policy to use specified resource
 - Check if entity is allowed (authorized) to use resource
- Authentication may be required before an entity allowed to use resource to which is authorized

Options for authentication

- Something you:
 1. Know (password, key)
 2. Have (token, smartcard)
 3. Are (biometrics)
- Combination of multiple options – two-factor authentication (or more)
 1. Registration phase (how is new user added)
 2. Verification phase (how is user's claimed identity verified)
 3. Recovery phase (what if user forgot/lost authentication credentials)

PASSWORDS

Mode of usage for passwords

- Verify by direct match (`provided_password == expected_password?`)
 - Example: *HTTP basic access* authentication
 - Be aware of plaintext storage on server
 - Be aware of potential side-channels (mismatch on Xth character)
- Verify by match of derived value (`hash(password | salt)`)
 - Be aware of rainbow tables and brute-force crackers
- Derive key: Password → cryptographic key
 - Example: `key = PBKDF2(password)`
- Used to establish authenticated key
 - Example: Password + Diffie-Hellman → authenticated key...

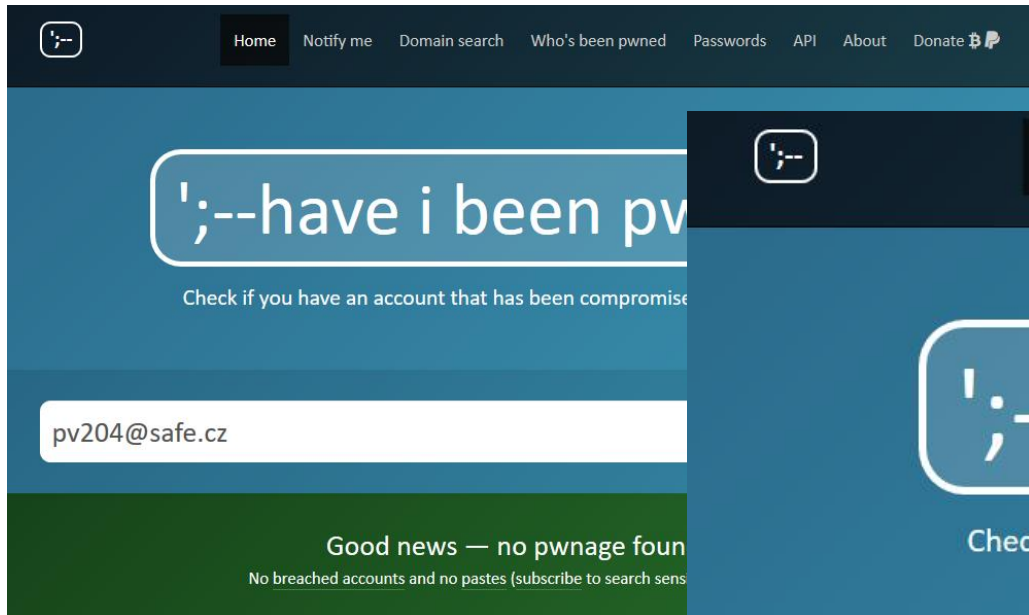
Problems associated with passwords

- How to create strong password?
- How to use password securely?
- How to store password securely?
- Same value is used for the long time (exposure)
- Value of password is independent from the target operation (e.g., authorization of bank transfer request)
- User usually can't memorize long-enough password
- ...

Where the passwords can be compromised?

1. Client side (malware on user computer)
2. Database storage
 - Cleartext storage
 - Backup data (“tapes”)
 - Server compromise, misconfiguration
3. Host machine (memory, history, cache)
4. Network transmission (network sniffer, proxy logs)
5. Hardcoded secrets (inside app binary)
 - Difficult to **detect** compromise and **change** after the exposure

<https://haveibeenpwned.com/> (Troy Hunt)



Home Notify me Domain search Who's been pwned Passwords API About Donate

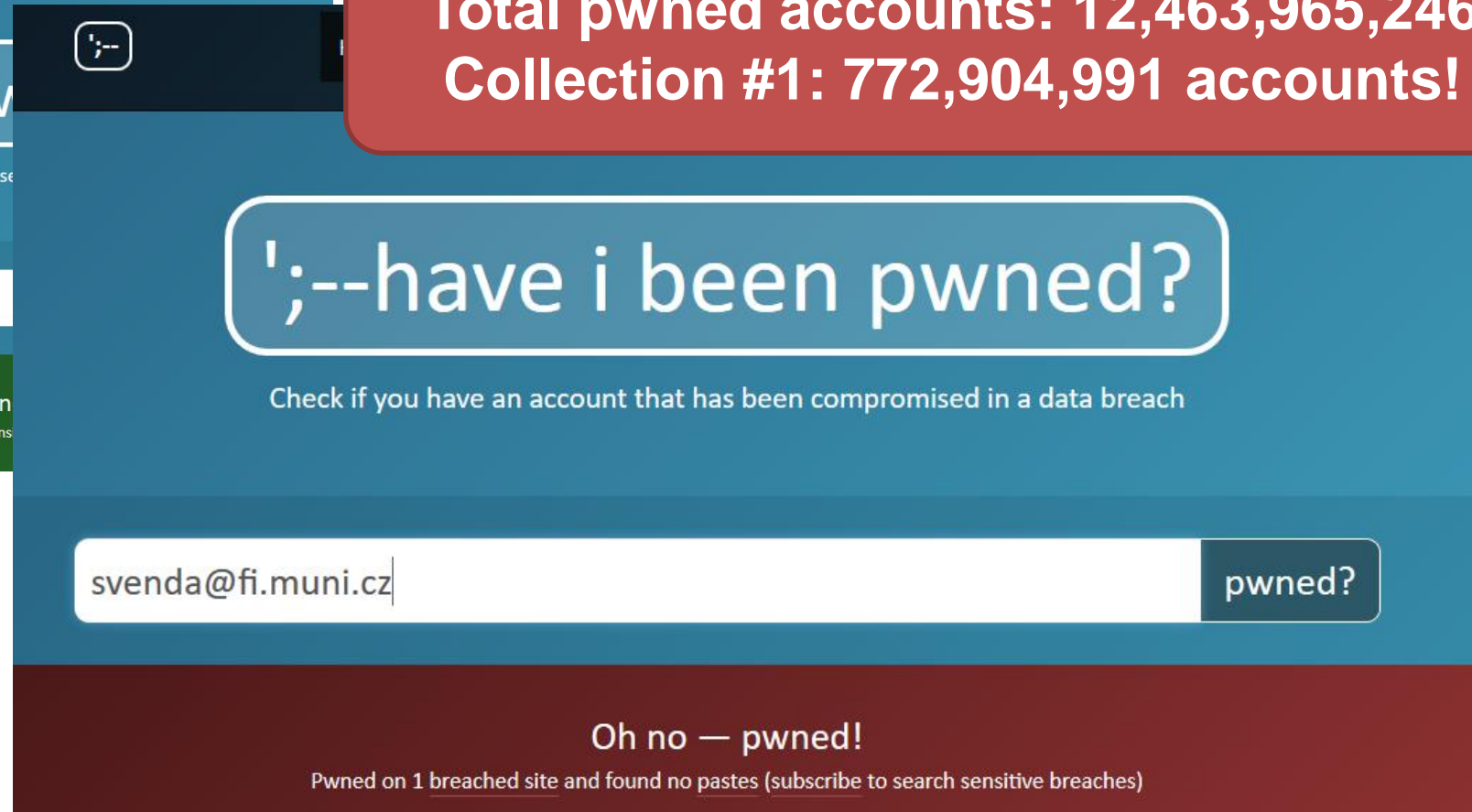
';--have i been pwned?

Check if you have an account that has been compromised in a data breach

pv204@safe.cz

Good news — no pwnage found
No breached accounts and no pastes (subscribe to search sensitive breaches)

Total pwned accounts: 12,463,965,246
Collection #1: 772,904,991 accounts!



';--have i been pwned?

Check if you have an account that has been compromised in a data breach

svenda@fi.muni.cz pwned?

Oh no — pwned!
Pwned on 1 breached site and found no pastes (subscribe to search sensitive breaches)

<https://haveibeenpwned.com/Passwords>

- Check how many times was given password found in leaked datasets

Home Notify me Domain search Who's been pwned Passwords API About Donate

Pwned Passwords

Pwned Passwords are 551,509,767 real world passwords previously exposed in data breaches. This exposure makes them unsuitable for ongoing use as they're at much greater risk of being used to take over other accounts. They're searchable online below as well as being downloadable for use in other online systems. [Read more about how HIBP protects the privacy of searched passwords.](#)

password pwned?

Oh no — pwned!
This password has been seen 3,645,804 times before

This password has previously appeared in a data breach and should never be used. If you've ever used it anywhere before, change it!

Password “hardening” ideas

1. Hash password by one-way function (shall be hard to invert)
2. Slowdown cracking attempts (less potential passwords tried)
3. Enable users to have long, random and unique passwords
4. Have unique password for every authentication attempt
5. Replace/complement passwords with something else (e.g., smartcard)
6. Bind response to server domain name (to prevent phishing)



In follow-up slides, we will discuss these ideas one by one

IDEA: HASH PASSWORDS

/etc/ {passwd, shadow}

Central file(s) describing UNIX user accounts.

/etc/passwd

- Username
- UID
- Default GID
- GCOS
- Home directory
- Login shell

/etc/shadow

- Username
- Encrypted password
- Date of last pw change.
- Days 'til change allowed.
- Days `til change required.
- Expiration warning time.
- Expiration date.

```
student:x:1000:1000:Example User,,555-1212,:/home/student:/bin/bash
```

```
student:$1$w/UuKtLF$otSSvXtSN/xJzUOGFEINz0:13226:0:99999
```

```
[root@arch01 ~]# cat /etc/shadow | sed 's/michael/test/' | sed 's/mbo/joe/'
root:$6$4GxAA08J$AB7vFkLSCxtVdVMcPav8jZ5u4ZsyG22hy1cqWPdnQgqL84VesJNQYFXSwhfwkhT
UeHNxYwjJUGe8U/sjITBhq/:16672:::::::
bin:x:14871:::::::
daemon:x:14871:::::::
mail:x:14871:::::::
ftp:x:14871:::::::
http:x:14871:::::::
uuid:x:14871:::::::
dbus:x:14871:::::::
nobody:x:14871:::::::
systemd-journal-gateway:x:14871:::::::
systemd-timesync:x:14871:::::::
systemd-network:x:14871:::::::
systemd-bus-proxy:x:14871:::::::
systemd-resolve:x:14871:::::::
systemd-journal-upload:!:16672:::::::
systemd-journal-remote:!:16672:::::::
avahi:!:16672:::::::
polkitd:!:16672:0:99999:7:::
joe:$6$TA4PsLzF$ch961z/ppk1VrmVAqSjSEdf75FIahttselx/bsDdjSXLt8cmsIoX9eAKfVm8epuD
KGvYV1xkohA37aeEvmu8d1:16672:0:99999:7:::
test:$6$PNkLwU7L$2Hm8YRMGgRoxxt4srAzGBZJFxFU7S
kZ1PwBzY1aHAVZu29wSBpJ0:16735:0:99999:7:::
```

Joe; insecure

(Hashed-)Password cracking

- Scenario: dump of database with password hashes, find original password
- Password cracking attacks
 - Brute-force attack (up to 8 characters)
 - Dictionary attack (passwords with higher probability tried first)
 - Patterns: Dictionary + brute-force (Password[0-9]*)
 - Rainbow tables (time-memory trade-off)
 - Parallelization (many parallel cores)
 - GPU/FPGA/ASIC speedup of cracking
- Tools
 - Generic: Hashcat, John the Ripper, Brutus, RainbowCrack...
 - Targeted to application: TrueCrack, Aircrack-NG...



Password reality (from many breaches + pwd cracking)

- User has usually weak password
 - >60% were (dictionary) brute-forced
- Server/service is frequently compromised
 - Server-side compromises are now very frequent
- Users do not use unique passwords between services
 - Gawker and root.com leaks: 76% had the exact same password
- Different authentication channels may not be independent
 - Web-browsing + SMS on smart phones?
- Account recovery is often easier to guess than original password



Re-enter your password

Pick a secret question

Select your secret question...

- Select your secret question...
- What street did you grow up on?
- What is your mother's maiden name?
- What is the name of your first school?
- What is your pet's name?
- What is your father's middle name?
- What is your school's mascot?

--Month-- --Day-- --Year--

You must be at least 18 years old to use eBay.

Insecure password handling ... what is the attack?

- Verify by direct match (`provided_password == expected_password?`)
 - Attack: compromise plain passwords on server
- $\text{pwdTag}_i = \text{SHA-2}(\text{"password"})$
 - Same passwords from multiple users => same resulting `pwdTag`
 - Attack: Large pre-computed “rainbow” tables allow for very quick check common passwords
- $\text{pwdTag}_i = \text{SHA-2}(\text{"password"} \mid \text{salt})$
 - Use of rainbow tables “prevented” by addition of random (and potentially public) *salt*
 - Attack: dictionary-based brute-force still possible
- $\text{pwdTag}_i = \text{AES}(\text{"password"}, \text{secret_key})$
 - Attack: If `secret_key` is leaked => direct decryption of all stored `pwdTags` => passwords

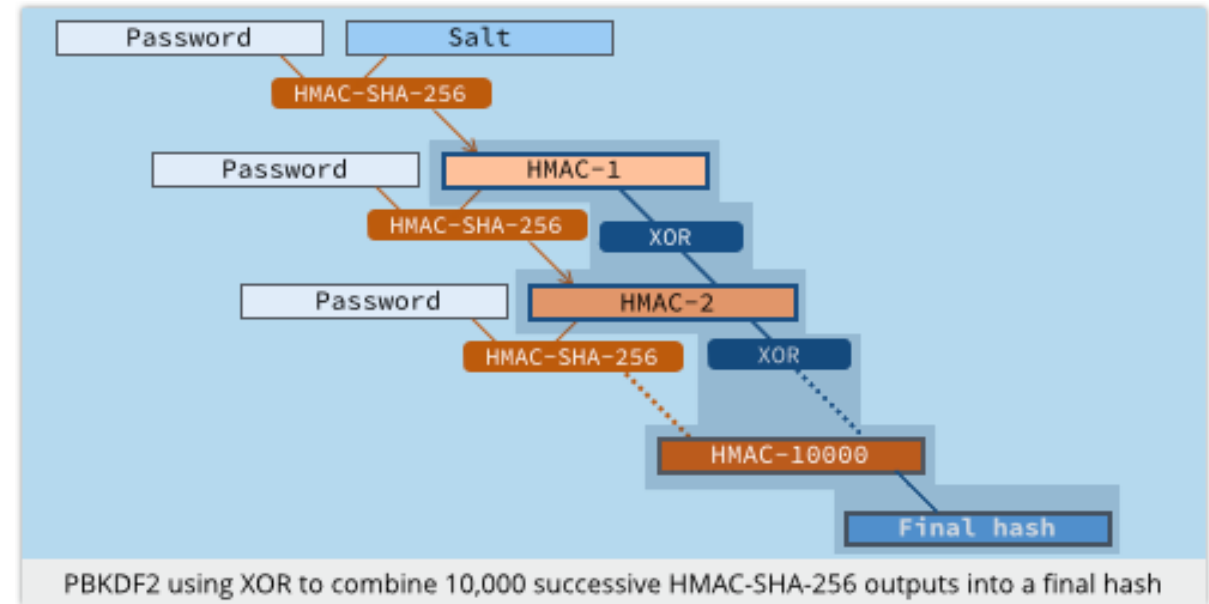


Some issues addressed by PAKE (Password Authenticated Key Exchange) protocols – future lecture

IDEA: SLOWDOWN CRACKING ATTEMPTS

Derivation of secrets from passwords

- PBKDF2 function, widely used
 - Password is key for HMAC
 - Salt added
 - Many iterations to slow derivation



Source: <https://nakedsecurity.sophos.com>

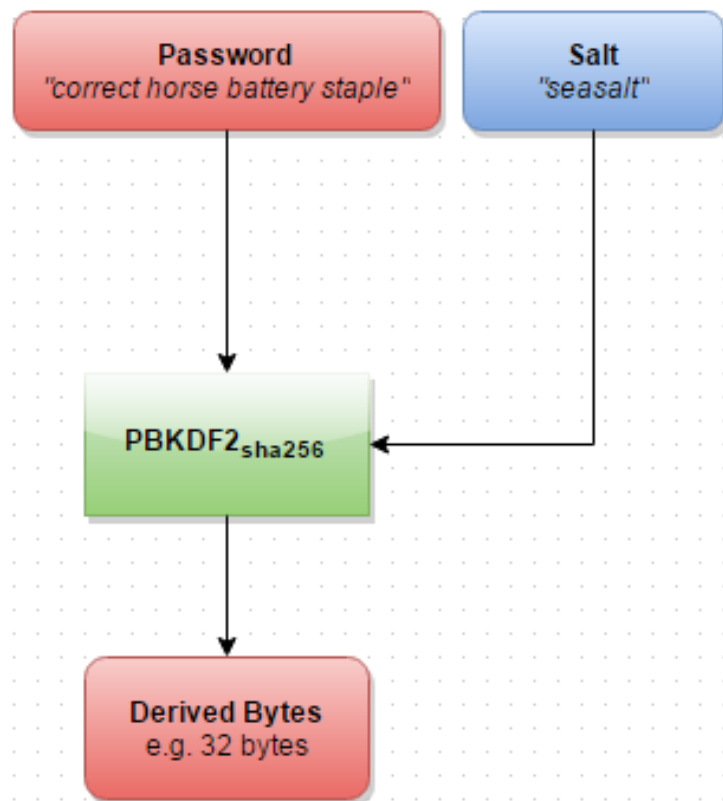
- Problem with custom-build hardware (GPU, ASIC)
 - Repeated iterations not enough to prevent bruteforce
 - (or would be too slow on standard CPU – user experience)
- Solution: function which requires large amount of memory

scrypt – memory hard function

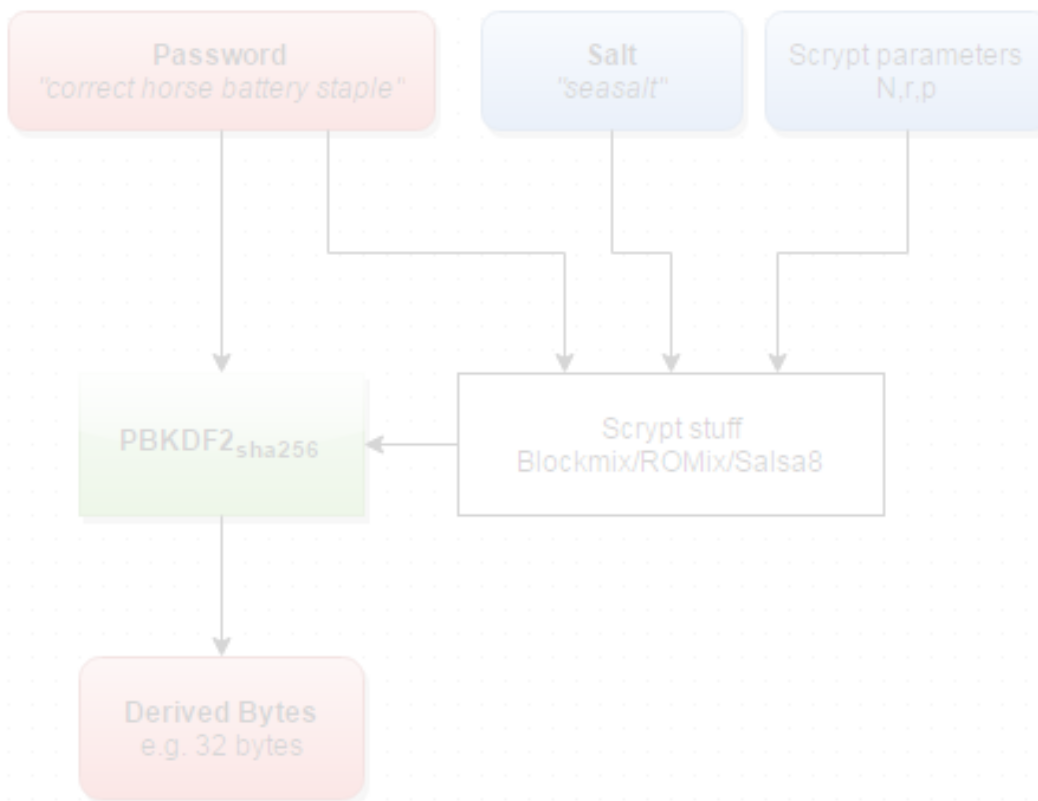
- Design as a protection against cracking hardware (usable against PBKDF2)
 - GPU, FPGA, ASICs...
 - <https://github.com/wg/scrypt/blob/master/src/main/java/com/lambdaworks/crypto/SCrypt.java>
- Memory-hard function
 - Force computation to hold r (parameter) blocks in memory
 - Uses PBKDF2 as outer interface
- Improved version: NeoScrypt (uses full Salsa20)

Reuse of external PBKDF2 structure

Standard PBKDF2



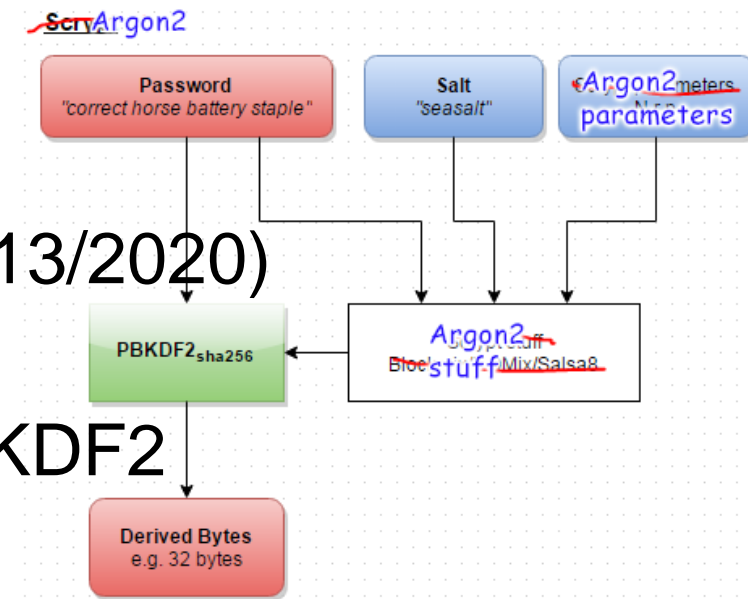
Script



https://www.reddit.com/r/crypto/comments/3dz285/password_hashing_competition_phc_has_selected/

Argon2 – memory hard function

- Password hashing competition (PHC) winner (2013/2020)
 - Large (configurable) memory size is required
- Memory hard functions are (slowly) replacing PBKDF2
 - Now available in OpenSSL 3.2
- Why it slows down GPU cracking?
 - GeForce RTX 4080 X3 16GB (9 728 cores, 16GB)
 - GPU has thousands cores => thousands PBKDF2 passwords tested in parallel
 - If Argon2 is used with 1GB memory required => max 16 passwords in parallel
- Why not parametrize with 16GB?
 - Legitimate user must also have available memory (mobile phone...)



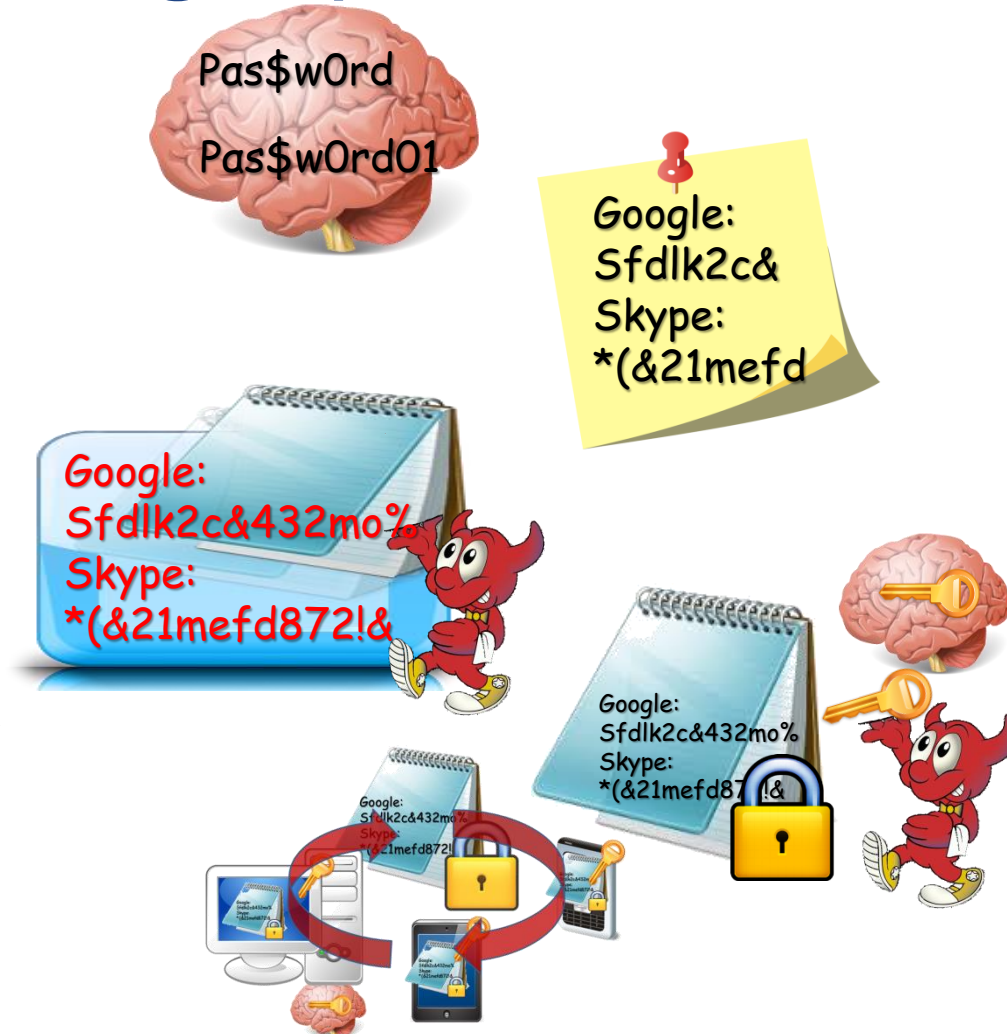
https://www.reddit.com/r/crypto/comments/3dz285/password_hashing_competition_phc_has_selected/

**IDEA: LONG, RANDOM AND UNIQUE
PASSWORDS**

PASSWORD MANAGERS

Evolution of password (managers)

1. Human memory only
2. Write it down on paper
3. Write it into file
4. Use local password manager



Remote password managers



CNET > Security > LastPass CEO reveals details on security breach

LastPass CEO reveals details on security breach

CEO of the password management company, which is dealing with a likely breach, tells PC World that users with strong master passwords should be safe,

users with strong master passwords should be safe,



Following yesterday's **revelation of a likely security breach** at password management company LastPass, the company's CEO is revealing more details about the incident and trying to offer some comfort and advice to his users.

Speaking yesterday with **PC World**, LastPass CEO Joe Siegrist admits he

But passwords are encrypted, right?

may have been too "alarmist" in sounding the alarm bell over the potential security breach. But the anomalies the company found when looking over its logs raised too much of a red flag.

Siegrist explained that he doesn't think a lot of data would've been hacked,

but just enough to capture a small number of user names and passwords.





Karim Toubba
CEO, LastPass

[More Articles](#)

Join our newsletter

Enter your email for updates from the LastPass Blog.

Email Address:

December 22, 2022

Notice of F

Update as of Thursday, Dec

To Our LastPass Commur

We recently notified you
based storage service, wh
data. In keeping with our
update regarding our ong

LastPass' post has even elicited a response from a competitor, 1Password — on Wednesday, the company's principal security architect Jeffrey Goldberg wrote a post for its site titled "Not in a million years: It can take far less to crack a LastPass password." In it, Goldberg calls LastPass' claim of it taking a million years to crack a master password "highly misleading," saying that the statistic appears to assume a 12 character, randomly generated password. "Passwords created by humans come nowhere near meeting that requirement," he writes, saying that

Devil is in the details

- How are passwords encrypted? (PBKDF2 or Argon2? Parameters?)
- How are legacy users handled? (possible smaller parameters)
- Is everything encrypted? (URL, notes, IPs...)
- Is recovery possible? How?

Case study

PASSWORD MANAGER FOR MULTIPLE DEVICES

Functional and security assumptions

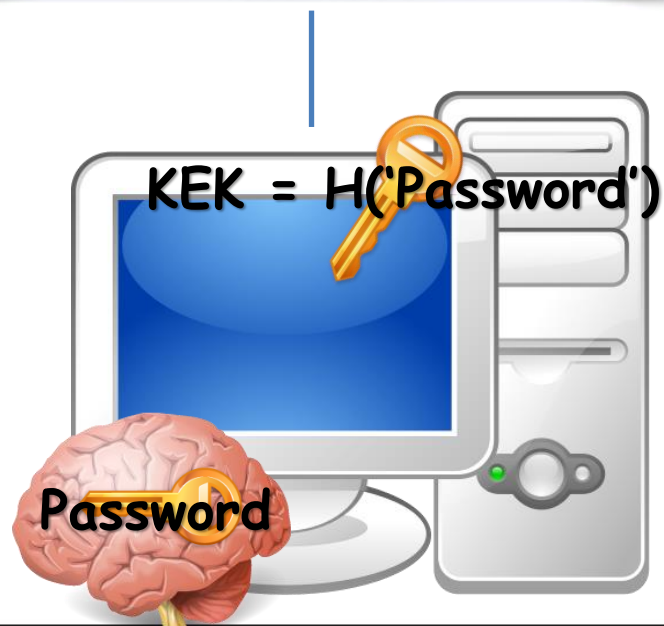
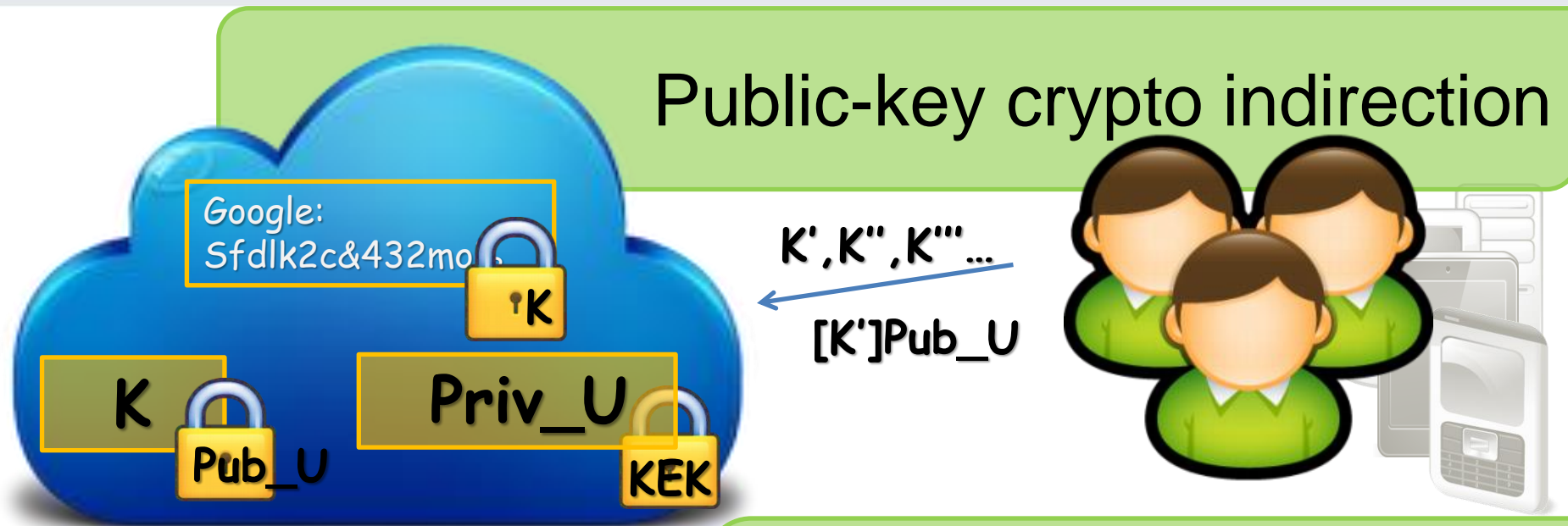
- Functional
 - User stores fixed secrets (passwords...)
 - User has multiple connected devices
 - Easy to use 😊
- Security
 - Service can't be trusted
 - User chooses weak password
 - Devices can be lost (and later revoked)
 - User has independent channel (phone)

Main security design principles

- I. Treat storage service as untrusted and perform security sensitive operations on client
- II. Make necessary trusted component as small as possible
- III. Prevent offline brute-force, but don't expect strong password from user
 - add entropy from other source
- IV. Make transmitted sensitive values short-lived
- V. (Trusted hardware can provide additional support)

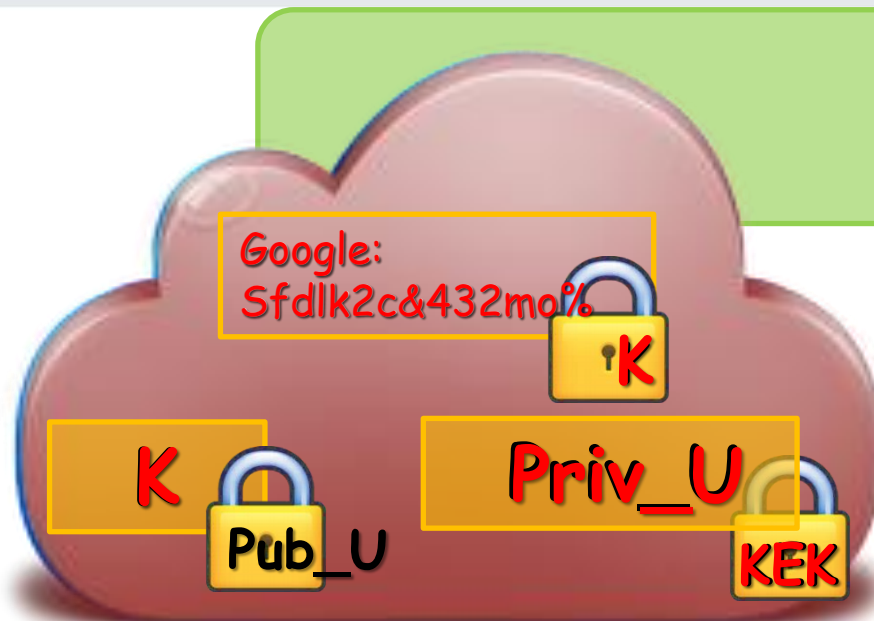
Public-key cryptography indirection





Public-key crypto indirection allows for asynchronous change of K

Long private key can be also stored on Service

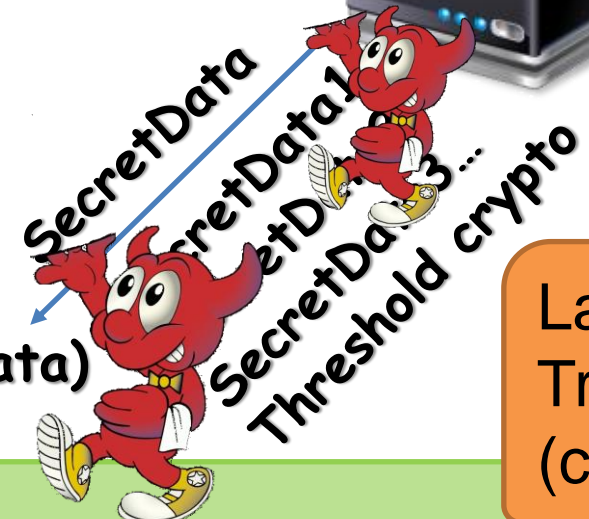
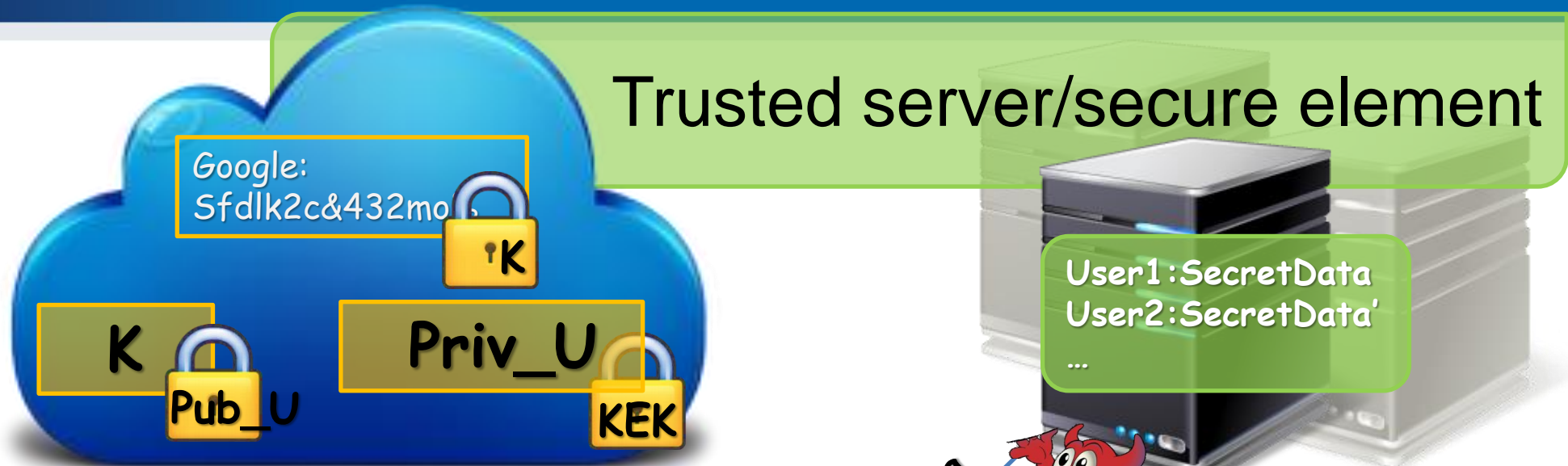


Weak password?

Users tend to have weak passwords...

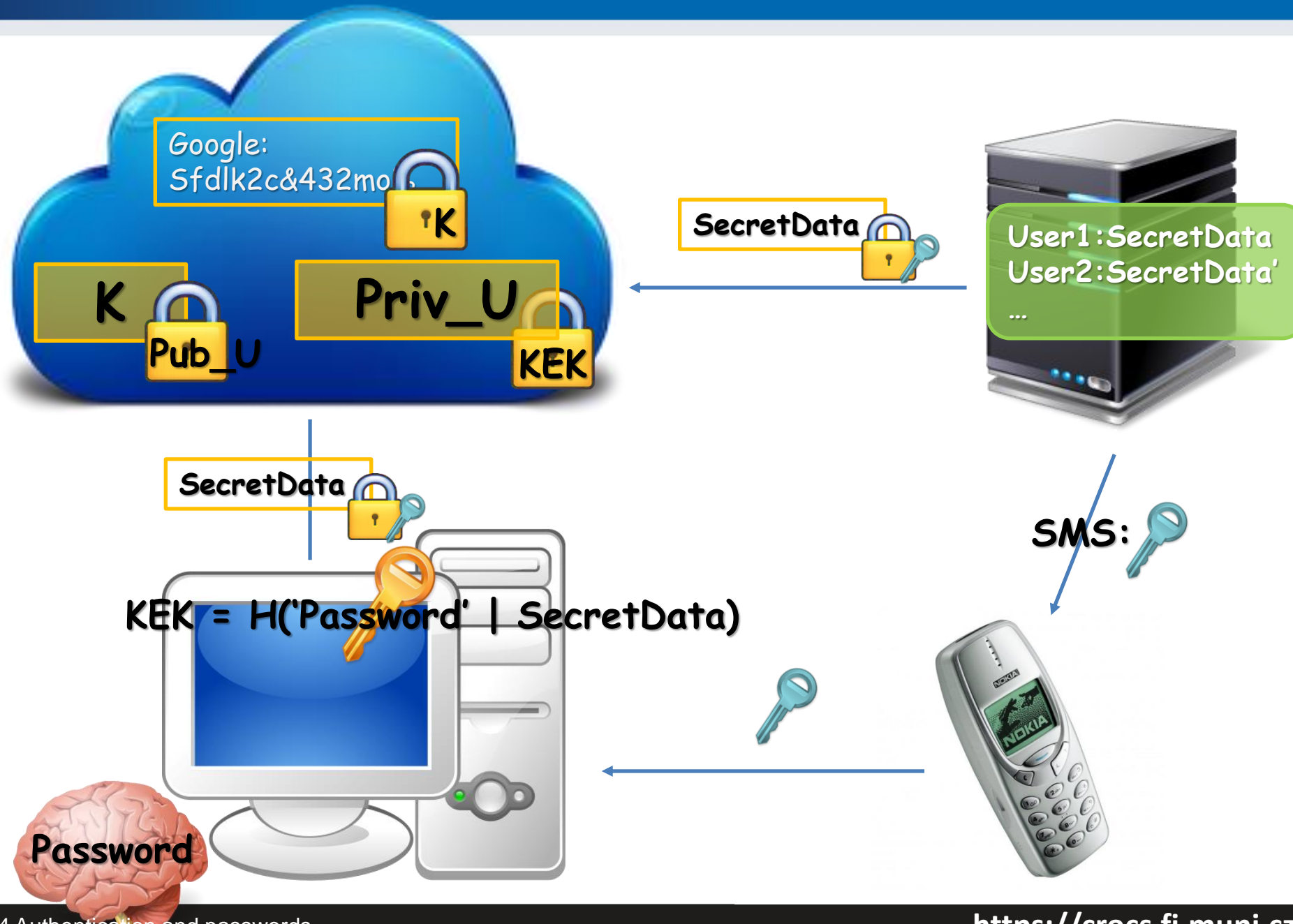


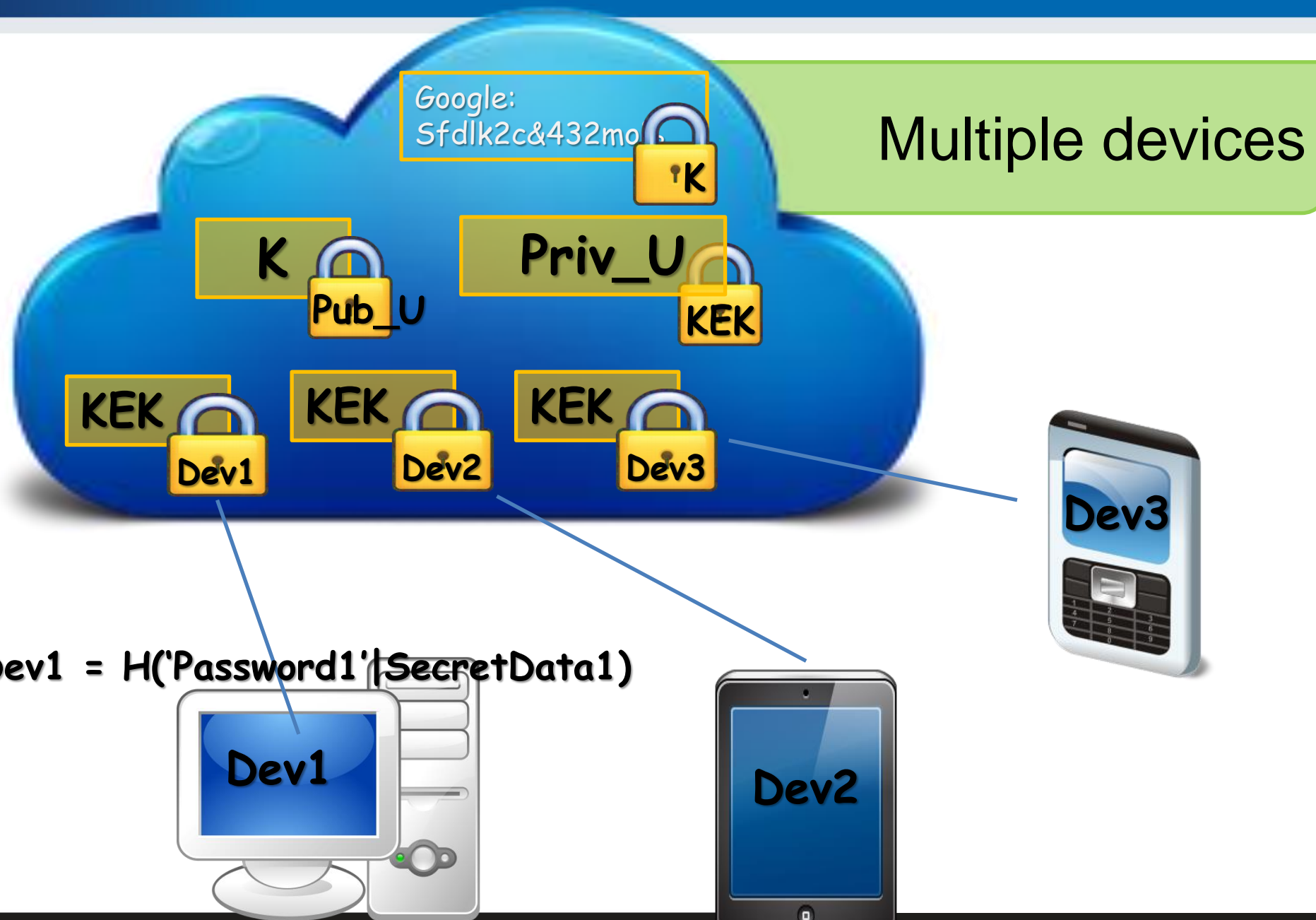
Attacker has motivation for attacking the Service!



Larger attack surface on Trusted server (connection from world)

Separate trusted entities provide additional data





Other operations

- Device management (new, remove, revoke)
- Device authentication
- Group management (users, boards, secrets)
- Password change, private key change
- Access recovery
- ...

Devil is in the details...



Do we have some implementations?

- Apple iCloud Keychain - service showcased in 2013
- Lack of details until iOS Security report 02/2014
 - https://web.archive.org/web/20150319073804/https://www.apple.com/business/docs/iOS_Security_Guide.pdf
 - <https://blog.cryptographyengineering.com/2016/08/13/is-apples-cloud-key-vault-crypto/> (M.Green)
- Current platform details (Advanced Data Protection for iCloud)
 - https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf



Apple's iCloud Keychain

- Multiple similarities to the described example
 - Layer of indirection via asymmetric cryptography
 - Support for multiple devices
 - Asynchronous operations via application tickets
 - Authorization and signature of additional devices
 - User phone registered and required
- Still reliance on user's (potentially weak) password for recovery
 - But only limited number of tries allowed
- Trusted component via internal HSM (Hardware Security Module)
 - Recovery mode with 4-digit code (default, can be set longer)
 - HSM will decrypt recovery key only after code validation
 - Note: only 4 digits is not an issue here – HSM enforce limited # retries



**IDEA: HAVE UNIQUE PASSWORD FOR
EVERY AUTHENTICATION ATTEMPT
ONE-TIME PASSWORDS: HOTP & TOTP**

ONE-TIME PASSWORDS

Recall: Problems associated with passwords

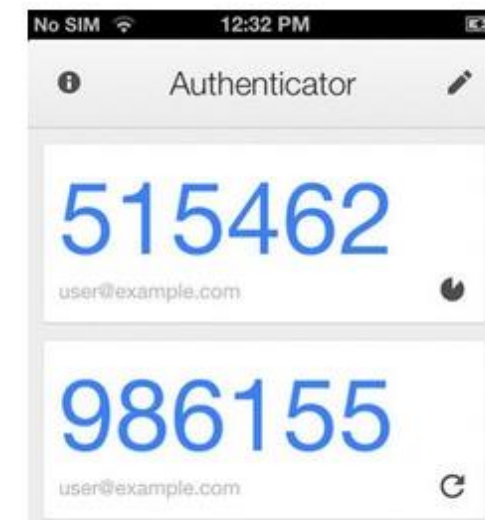
- How to create secure password?
- How to use password securely?
- How to store password securely?
- Same value is used for the long time (exposure)
- Value of password is independent from target operation (e.g., authorization of request)
- ...



One-time passwords tries to address these issues

HMAC-based One-time Password Algorithm (RFC 4226)

- HMAC-based One-time Password Algorithm (HOTP)
 - Secret key K
 - Counter (challenge) C
 - $HMAC(K, C) = SHA1(K \oplus 0x5c5c... \parallel SHA1(K \oplus 0x3636... \parallel C))$
 - $HOTP(K, C) = Truncate(HMAC(K, C)) \& 0x7FFFFFFF$
 - $0x7FFFFFFF$ mask to drop most significant bit (portability)
 - $HOTP\text{-Value} = HOTP(K, C) \bmod 10^d$ ($d \dots \#$ of digits)
- Many practical implementations
 - E.g., Google Authenticator
- <https://en.wikipedia.org/wiki/HOTP>



HOTP – items, operations

- Logical operations
 1. Generate initial state for new user and distribute key
 2. Generate HOTP code and update state (user)
 3. Verify HOTP code and update state (auth. server)
- Security considerations of HOTP
 - Client compromise
 - Server compromise
 - Repeat of counter/challenge
 - Counter mismatch tolerance window
 - Phishing – user enters HOTP code at phishing website

Time-based One-time Password Algorithm

- Very similar to HOTP
 - Time used instead of counter
- Requires synchronized clocks
 - In practice realized as time window
- Tolerance to gradual desynchronization possible
 - Server keeps device's desynchronization offset
 - Updates with every successful login



OCRA: OATH Challenge-Response Algorithm

- Initiative for Open Authentication (OATH)
- OCRA is authentication algorithm based on HOTP
- OCRA code = $\text{CryptoFunction}(K, \text{DataInput})$
 - K : a shared secret key known to both parties
 - *DataInput*: concatenation of the various input data values
 - Counter, challenges, $H(\text{PIN/Passwd})$, session info, $H(\text{time})$
 - Default *CryptoFunction* is HOTP-SHA1-6
 - <https://tools.ietf.org/html/rfc6287>
- Don't confuse with OAuth (delegation of authentication)
 - The OAuth 2.0 Authorization Framework (RFC6749)
 - TLS-based security protocol for accessing HTTP service

Authentication server

$\text{HMAC}(\text{ctr}++, \text{key}) == \text{'385309'}$?

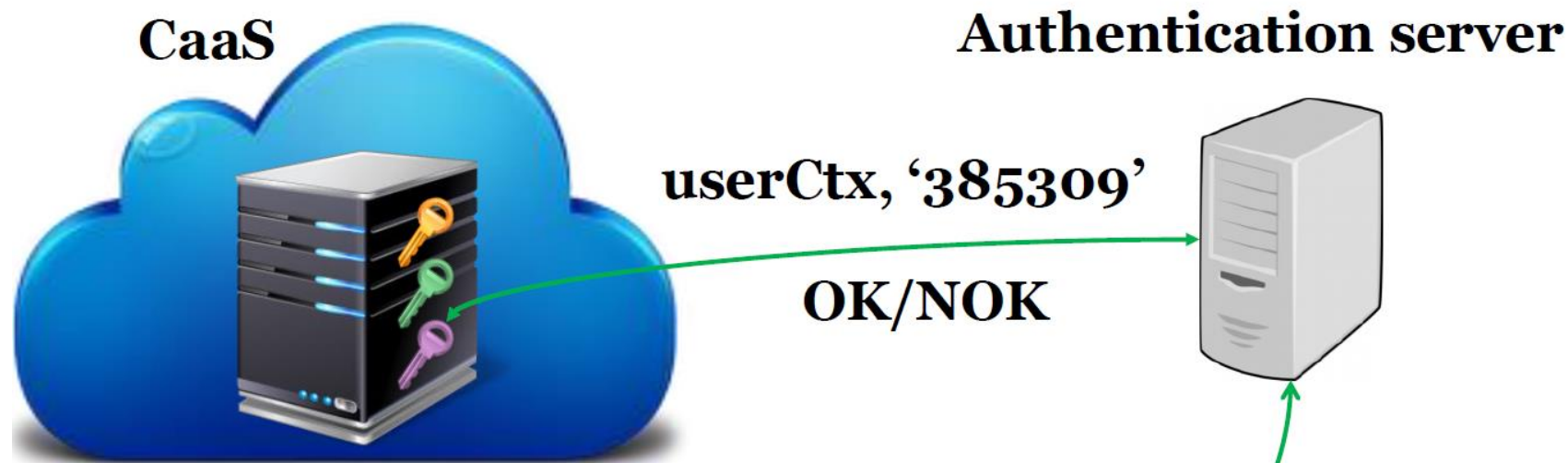
- Improves protection of client side
- Increases risk at Auth. server



$\text{HOTP} = \text{HMAC}(\text{ctr}++, \text{key}) = \text{'385309'}$

Increased risk at *OTP verification server

- More secure against client compromise
 - Using OTP instead of passwords, $KDF(\text{time}|\text{key})$,
- But what if server is compromised?
 - database hacks, temporal attacker presence
 - E.g., Heartbleed – dump of OTP keys
- Possible solution
 - Trusted hardware on the server
 - OTP code verified inside trusted environment
 - OTP key never leaves the hardware



'385309'

Problems:

1. Is OTP code fresh?
2. Is OTP generated for correct domain (not phishing)?

$$\text{HOTP} = \text{HMAC}(\text{ctr}++, \text{key}) = \text{'385309'}$$

Possible password replacements

- Cambridge's TR – wide range of possibilities listed
 - *The quest to replace passwords: a framework for comparative evaluation of Web authentication schemes*
 - <https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-817.pdf>
- Many different possibilities, but passwords are cheap to start with, a lot of legacy code exists and no mechanism offers all benefits



Mandatory reading: UCAM-CL-817

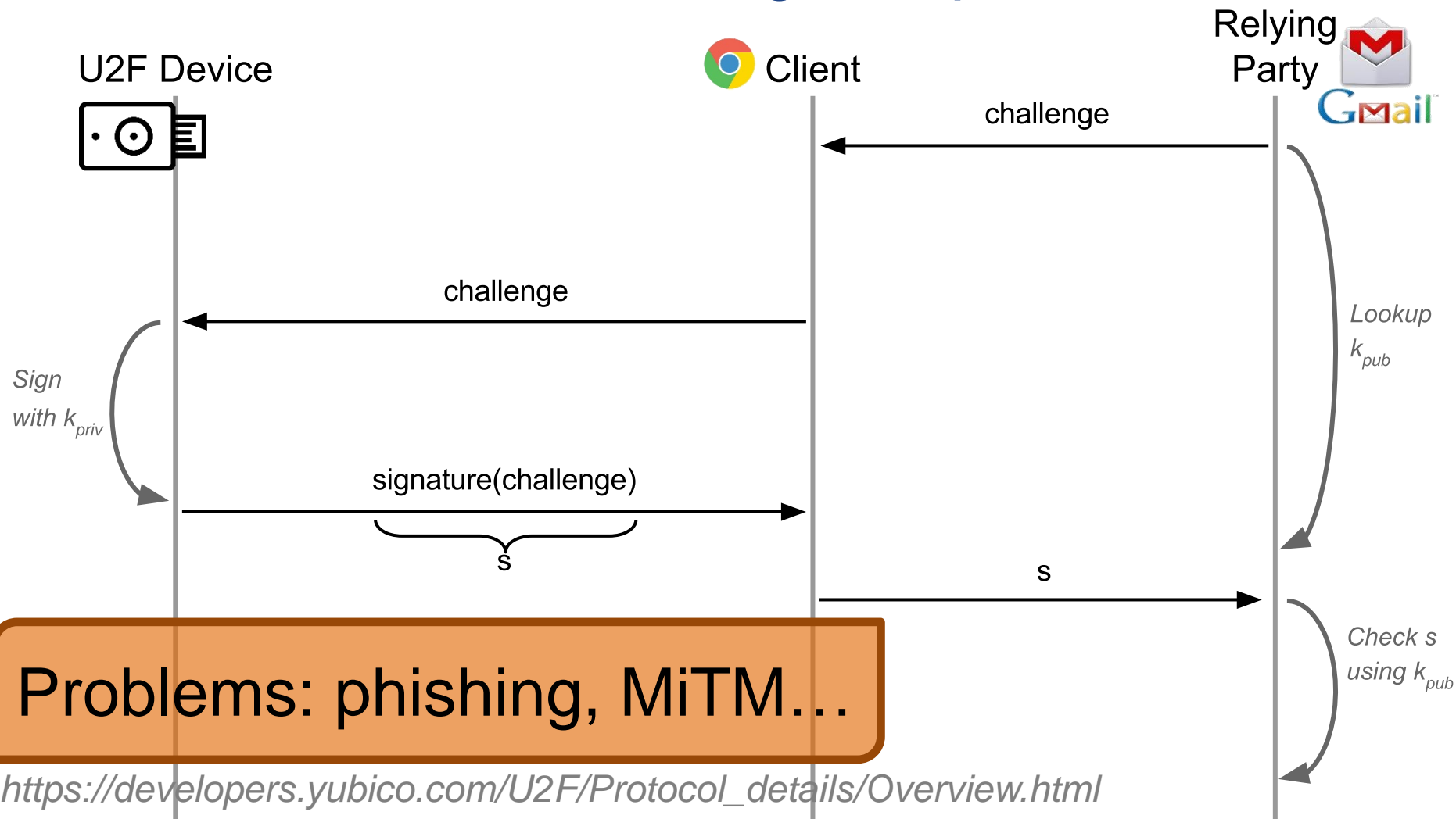
- At least chapters: II. Benefits, V. Discussion
- Whole report is highly recommended

**IDEA: REPLACE PASSWORD BY
SMARTCARD WITH ASYMMETRIC KEYPAIR,
CHALLENGE-RESPONSE PROTOCOL AND
PREVENT PHISHING**

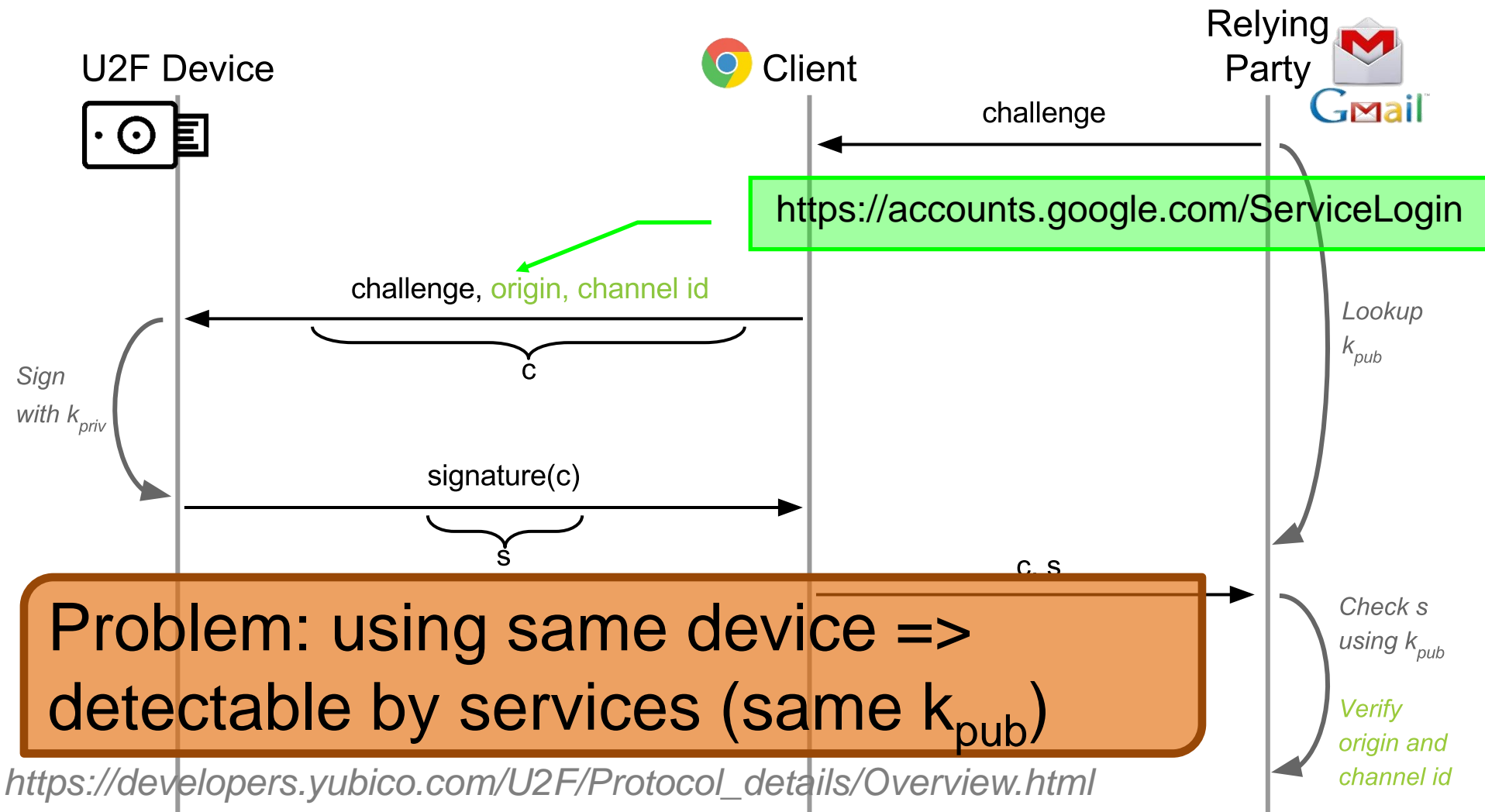
FIDO U2F PROTOCOL

(U2F → FIDO2 → WEBAUTHN)

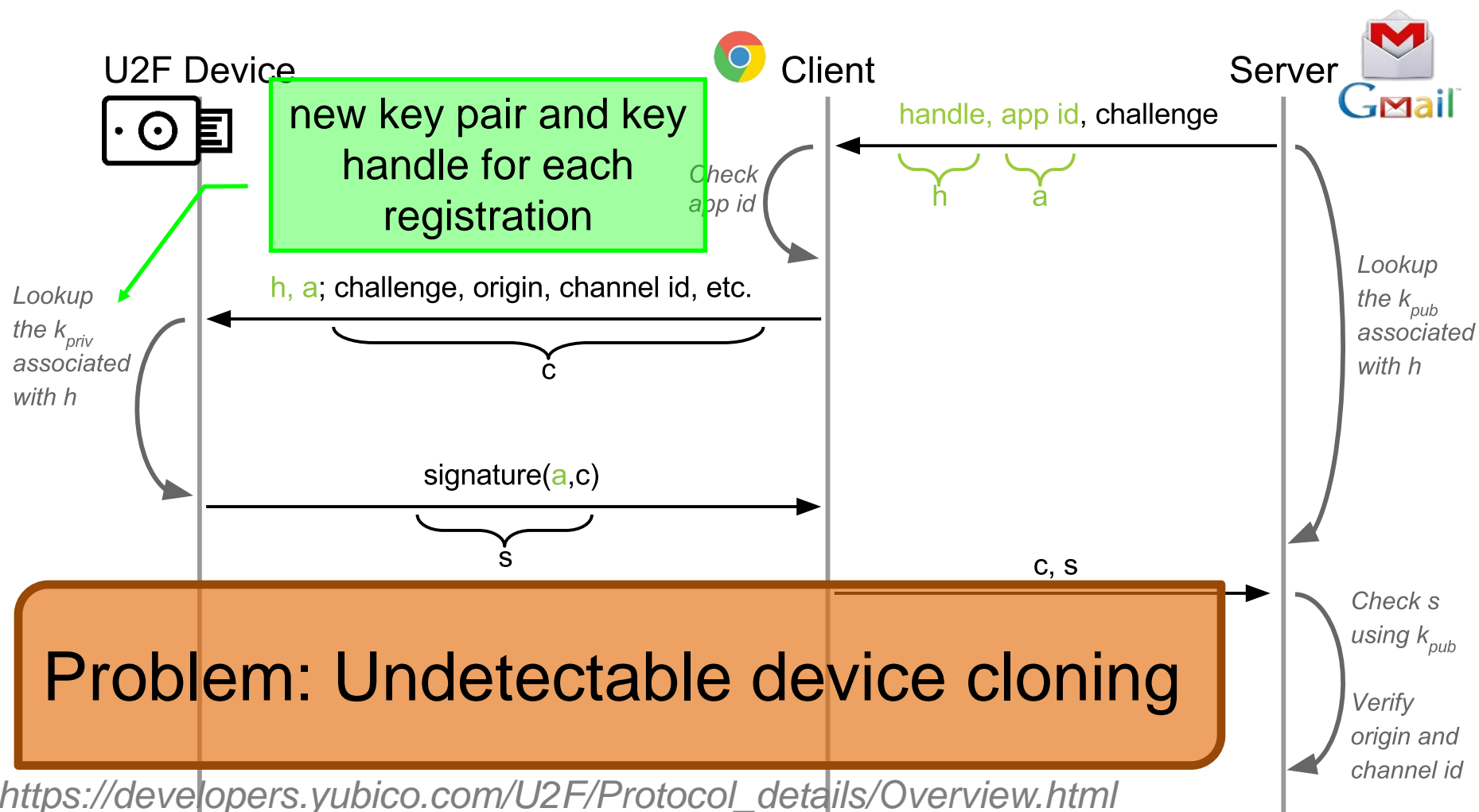
Revision 1: ECC-based challenge-response



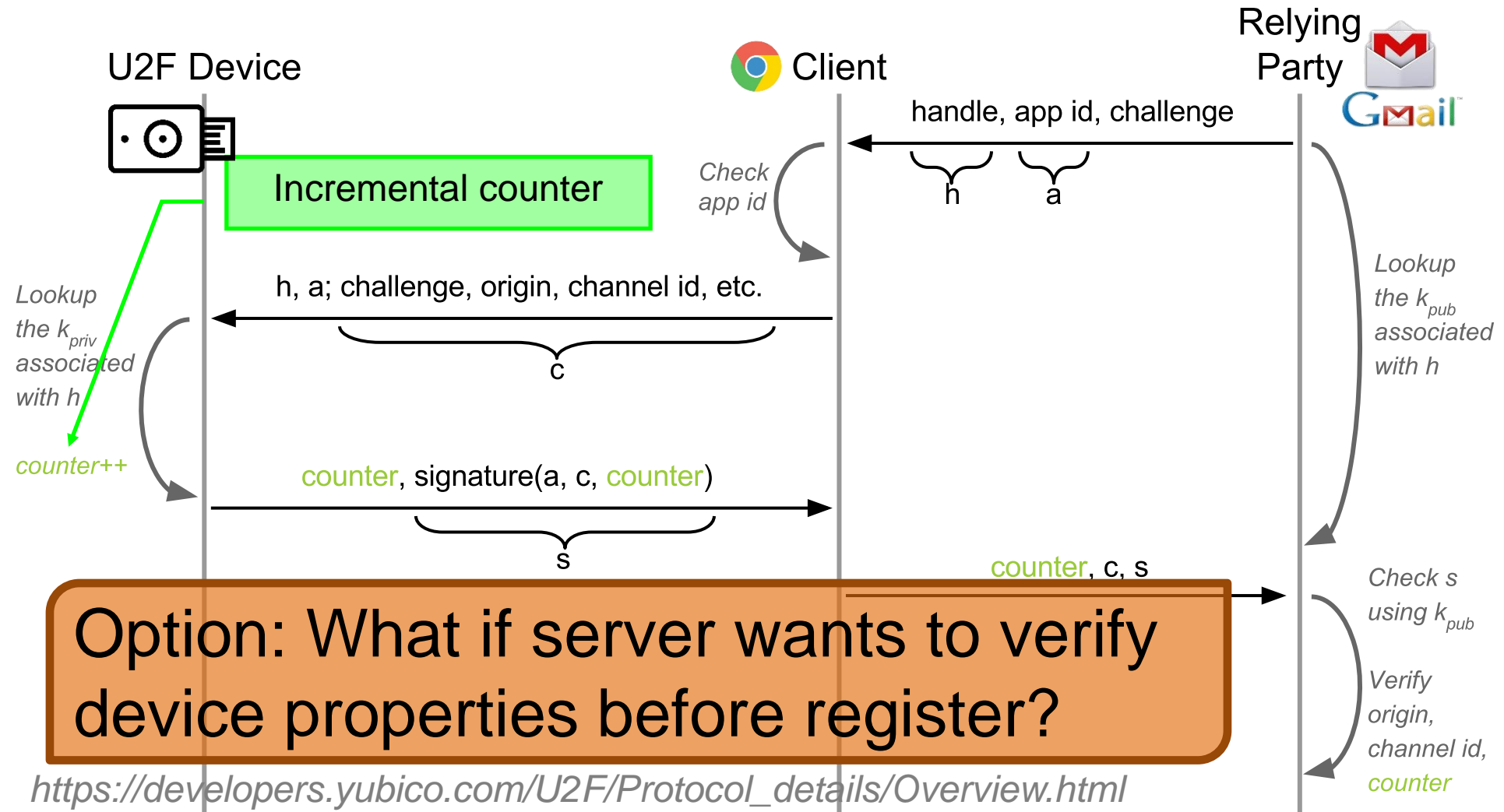
Revision 2: URI + TLS channel id added



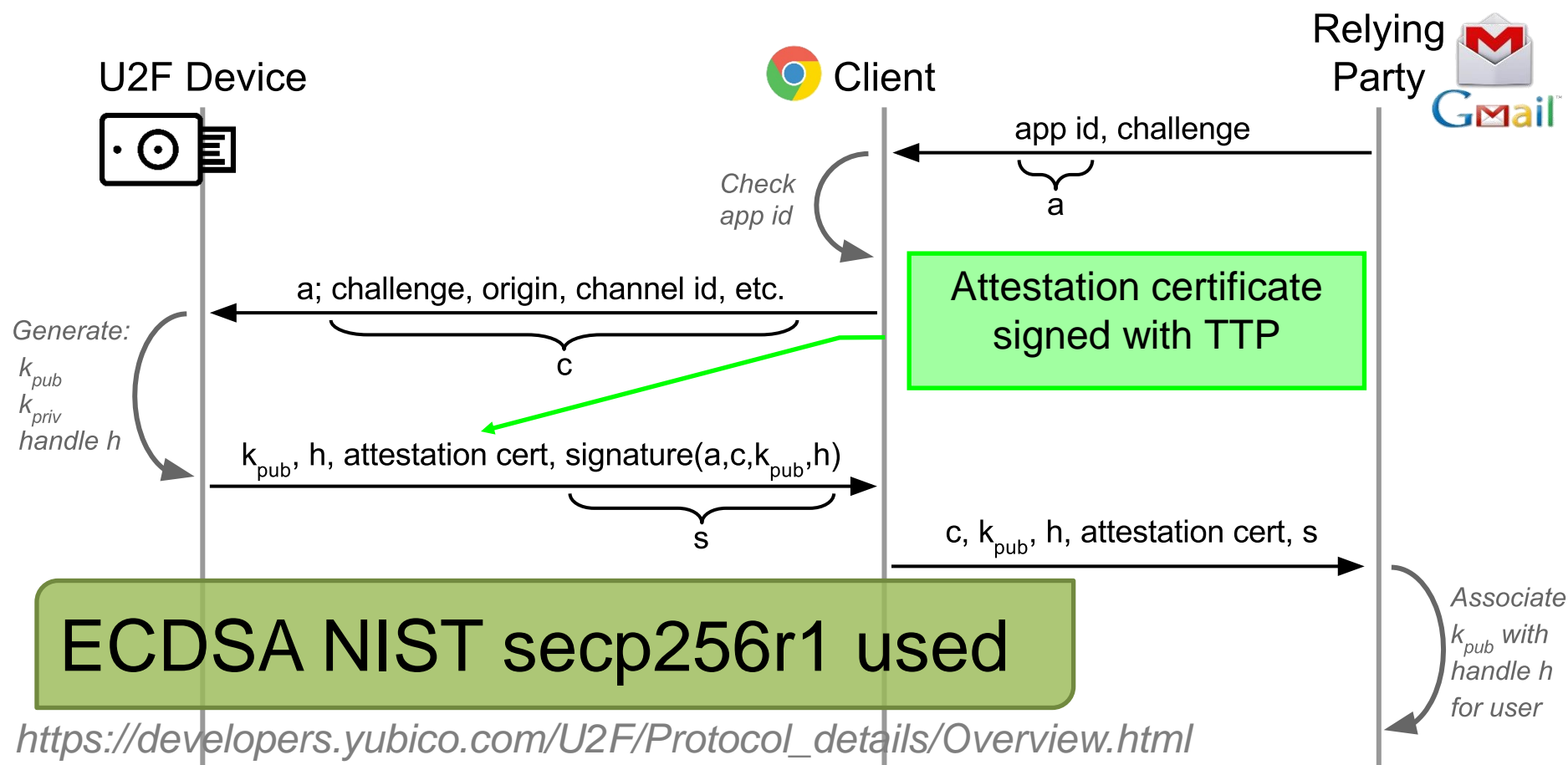
Revision 3: Application-specific key added



Revision 4: Authentication counter added



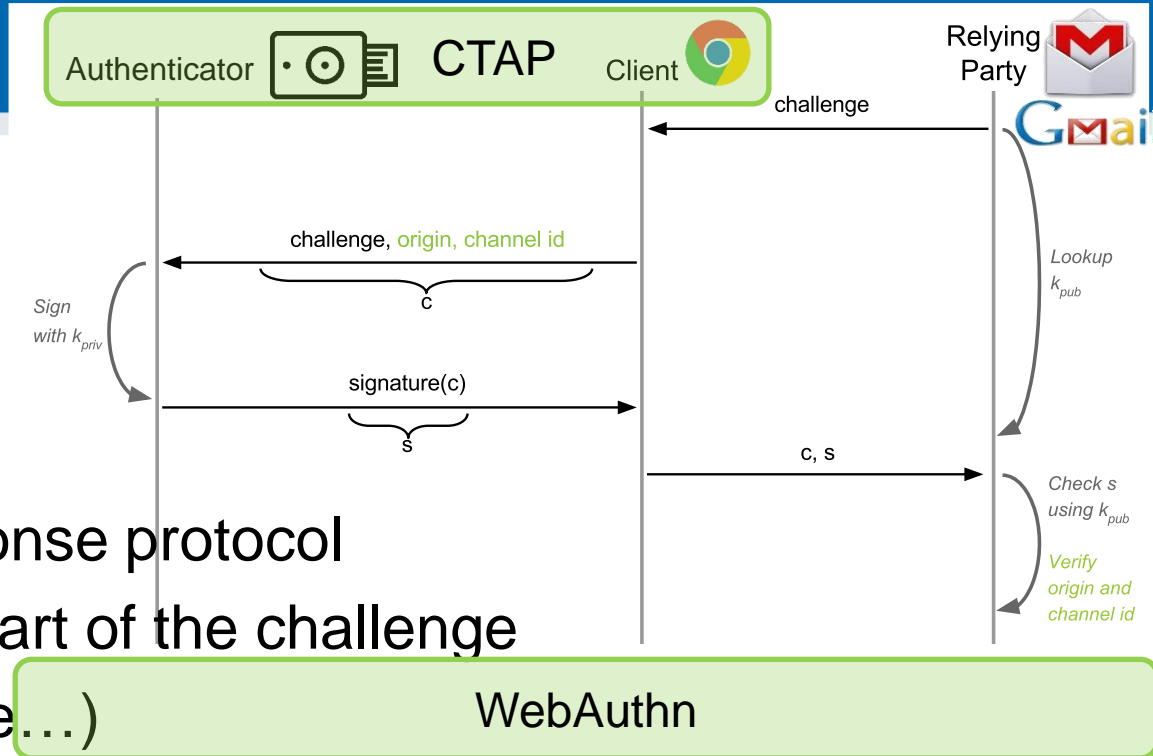
Revision 5: Device attestation added



FIDO U2F – current state

- FIDO alliance of major companies
- U2F → FIDO2 → WebAuthn (more than “just” U2F)
- Original U2F protocol extended and moved under W3 as WebAuthn
 - <https://www.w3.org/TR/webauthn/>
- Large selection of tokens now available (including open-hardware)
- Android added systematic support for FIDO U2F (02/2019)
 - Android phone acts as U2F token
 - <https://www.wired.com/story/android-passwordless-login-fido2>
- Google Smart Lock app on iOS uses secure enclave and acts as FIDO token
- Since iOS 13.3. USB, NFC, and Lightning FIDO2-compliant security keys in Safari browser (12/2019)





CTAP/WebAuthn stack

- WebAuthn Protocol

- Asymmetric crypto-based challenge-response protocol
- Browser inserts actual URL (origin) as a part of the challenge
- Private key stored and used (token, phone...)
- An API for accessing Public Key Credentials Level 2 (level=version)
 - Official documentation: <https://www.w3.org/TR/webauthn/>

- Client to Authenticator Protocol (CTAP)

- Protocol between browser and authenticator
- Authenticator = initially hardware token, but now range of devices (phones, calculators...)

https://developers.yubico.com/U2F/Protocol_details/Overview.html

True2F FIDO U2F token

- Yubikey 4 has single master key
 - To efficiently derive keypairs for separate Relying parties (Google, GitHub...)
 - Inserted during manufacturing phase (what if compromised?)
- Additional SMPC protocols (protection against backdoored token)
 - Secure Multi-Party Computation (SMPC) will be covered later
 - Verifiable insertion of browser randomness into final keypairs
 - Prevention of private key leakage via ECDSA padding
- Backward-compatible (Relying party, HW)
- Efficient: 57ms vs. 23ms to authenticate

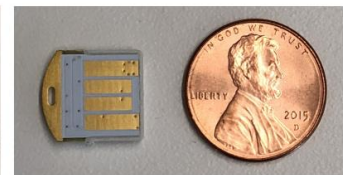
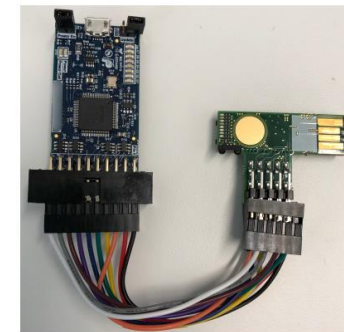
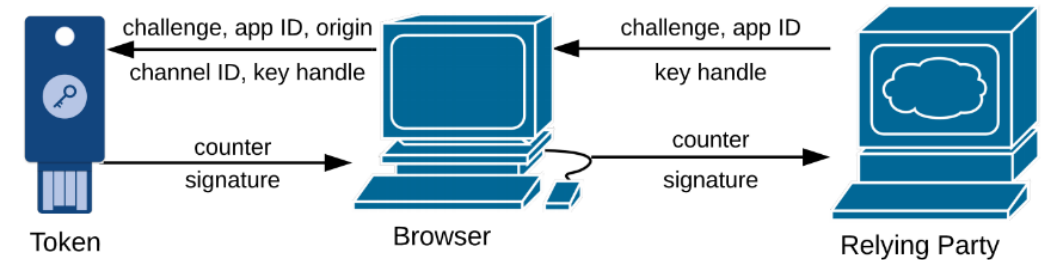


Figure 1: Development board used to evaluate our True2F prototype (at left) and a production USB token that runs True2F (above).

WebAuthn - evolution of U2F protocol

- An API for accessing Public Key Credentials Level 2
 - Official documentation: <https://www.w3.org/TR/webauthn/>
 - (Level means version here 😊)
- Similar, but more complex standard than U2F
- Client to Authenticator Protocol (CTAP)
 - protocol for communication between browser and token (authenticator)
 - USB, NFC, Bluetooth
 - CTAP 2.2 adds support for the hybrid transport (FIDO Cross-Device Authentication flow, aka Passkeys)
- Explanation, demo page <https://webauthn.guide/#about-webauthn>

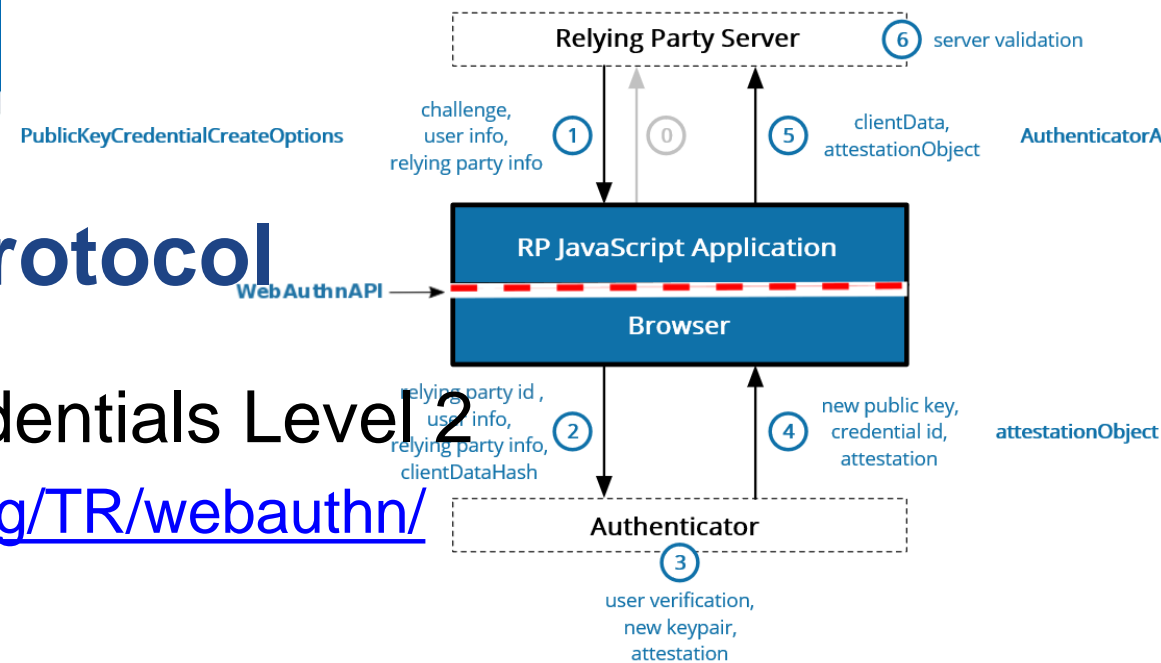


Figure 1 Registration Flow

Missing piece? “passkeys”

- ✓ Authentication on PC with FIDO2 token
- ✓ Authentication on mobile phone (with or without token)
- ? Authentication on PC **without** FIDO2 token?
 - Idea of “passkeys” (multi-device FIDO credentials)
 - WebAuthn (“U2F”) protocol used for base authentication (private keys needed)
 - Replace hardware token with mobile phone
 - Connect mobile phone with PC using Bluetooth LE (BLE) or websockets
 - Now supported natively by Apple (Keychain), Google (Password Manager) and Microsoft (Hello)
 - <https://media.fidoalliance.org/wp-content/uploads/2022/03/How-FIDO-Addresses-a-Full-Range-of-Use-Cases-March24.pdf>
 - <https://passkeys.dev/docs/reference/specs/>

FIDO U2F devices

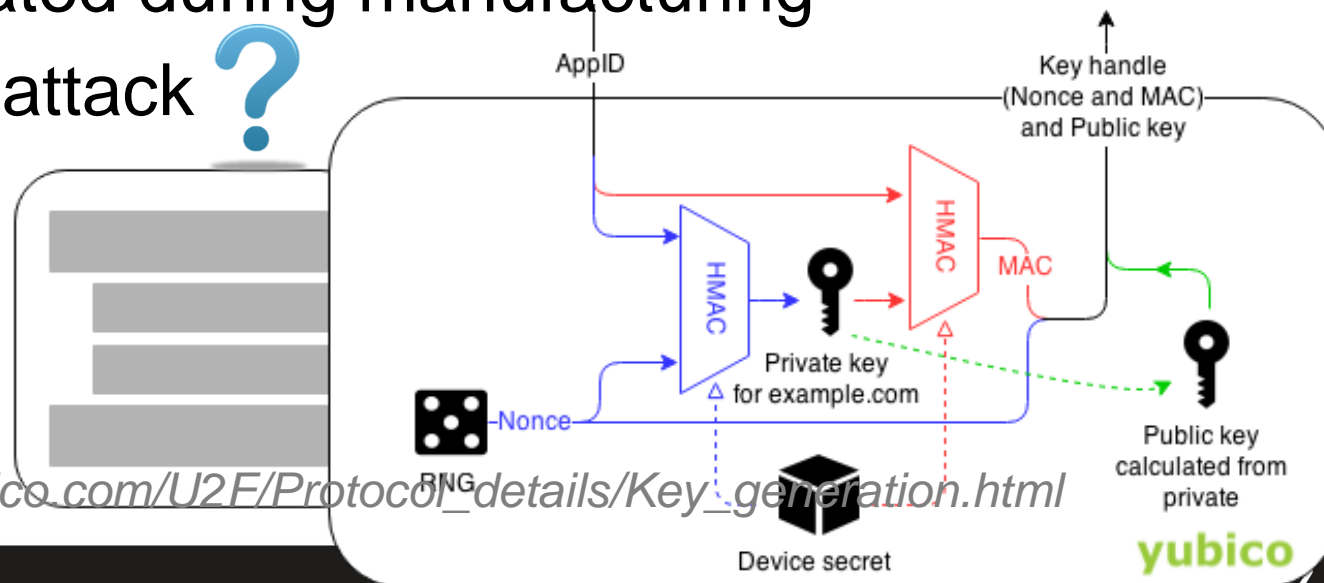


- Why have button? Is missing display problem?
- Existing past attack: direct WebUSB API in Chrome
 - Malware bypass U2F API checking the URL
 - Legitimate URL is send from malicious page
 - <https://www.wired.com/story/chrome-yubikey-phishing-webusb/>
 - APDU-level communication: <https://npmccallum.gitlab.io/post/u2f-protocol-overview/>
- Well known is Yubikey, but open-source hardware and/or software-only implementations also possible
 - <https://github.com/conorpp/u2f-zero>
 - <https://github.com/solokeys/solo>



Always dig for implementation details

- How are ECC keys generated and stored?
- Yubikey saves storage memory by deriving ECC private keys from master secret instead of randomly generating new one
 - Possible as the ECC private key is random value
- Device secret generated during manufacturing
- What is the possible attack ?



Summary

- Passwords have multiple issues, but are hard to be replaced
- Major server-side breaches now very common
- Important to use passwords securely (guidelines)
- One-time passwords and tokens getting more used
- Password manager with synchronization over multiple devices is not straightforward, but doable (e.g., Apple's iCloud Keychain)
- FIDO2/Webauthn is major authentication improvement – use it!
- **Mandatory reading:** UCAM-CL-817
 - At least chapters: II. Benefits, V. Discussion
 - Whole report is highly recommended

Top questions (1) ▾



PetrS

0 👍

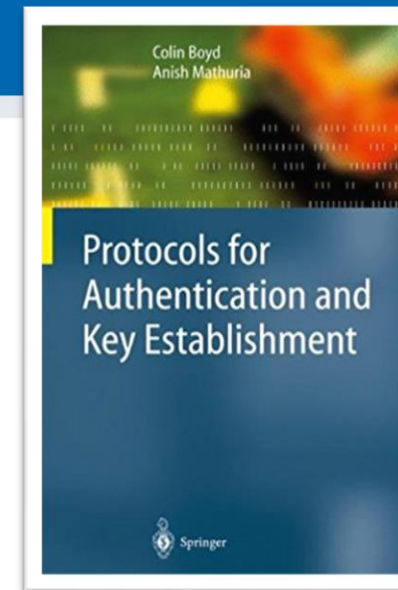
Is my password brute-force-able if consists of 9 printable characters?

Join at

slido.com

#pv204_2024

Hierarchy of authentication and key establishment goals



Protocols for Authentication and Key Establishment By Colin Boyd, Anish Mathuria

Common (mis-)Assumptions

1. User has strong password
2. Server/service is hard to compromise
3. User have unique passwords
4. Different authentication channels are independent
5. Recovery

Password cracking defenses

- Don't transmit or store in plaintext
- Process password on client, transmit only digest
- Don't encrypt, hash instead
- Use salt to prevent rainbow tables attack
- Use memory-hard KDF algorithms
 - To slow down custom build hardware
 - Use strong KDF to derive keys (PBKDF2→Argon2)
- Use password-authenticated key exchange instead of password check

Handling passwords in source code

- Limiting memory exposure
 - Load only when needed
 - Erase right after use
 - Pass by reference / pointer to prevent copy in memory
 - Derive session keys
- Don't hardcode password into application binary
- Nice presentation (K. Kohli, examples how NOT to):
http://www.slideshare.net/amiable_indian/insecure-implementation-of-security-best-practices-of-hashing-captchas-and-caching-presentation

Hard-coded password might be visible both in application binary and memory

The screenshot shows OllyDbg running AES_PolarSSL.exe. The CPU window displays assembly code for the main thread. A memory dump window is open, showing a dump of memory starting at address 0020B000. The dump contains a string of characters that appears to be a password: "SecurePassword:nbu123.#0Xr#0\r#0*r#0...0/##0*0_#2#00...iB0.°10.lut...4..X0..d²"...4..X0..é. .é6#04=I r...l. .-3hu.d²"%..=x~v.d²"w†B R... ..d²"... ..x:..... 'q=vU',\$,.....- .+x~vm2#0.d²"...

```

011D1000 $ 55      PUSH EBP
011D1001 . 8BEC     MOV EBP,ESP
011D1003 . 83EC 10  SUB ESP,10
011D1006 . 56       PUSH ESI
011D1007 . 833D 20501D01 CMP DWORD PTR DS:[aes_init_done],0
011D100E √ 75 0F   JNZ SHORT AES_Pola.011D101F
011D1010 . E8 1B1A0000 CALL AES_Pola.aes_gen_tables
011D1015 . C705 20501D01 MOV DWORD PTR DS:[aes_init_done],1
011D101F > 8B45 10  MOV EAX,DWORD PTR SS:[EBP+10]
011D1022 . 8945 F4  MOV DWORD PTR SS:[EBP-C],EAX
011D1025 . 817D F4 800000 CMP DWORD PTR SS:[EBP-C],80
011D102C √ 74 14   JE SHORT AES_Pola.011D1042
011D102E . 817D F4 C00000 CMP DWORD PTR SS:[EBP-C],0C0
011D1035 √ 74 16   JE SHORT AES_Pola.011D104D
011D1037 . 817D F4 000100 CMP DWORD PTR SS:[EBP-C],100
011D103E √ 74 18   JE SHORT AES_Pola.011D1058
011D1040 √ EB 21   JMP SHORT AES_Pola.011D1063
011D1042 > 8B4D 08  MOV ECX,DWORD PTR SS:[EBP+8]
011D1045 . C701 0A000000 MOV DWORD PTR DS:[ECX],0A
011D104B √ EB 20   JMP SHORT AES_Pola.011D106D
011D104D > 8B55 08  MOV EDX,DWORD PTR SS:[EBP+8]
011D1050 . C702 0C000000 MOV DWORD PTR DS:[EDX],0C
011D1056 √ EB 15   JMP SHORT AES_Pola.011D106D
011D1058 > 8B45 08  MOV EAX,DWORD PTR SS:[EBP+8]
011D105B . C700 0E000000 MOV DWORD PTR DS:[EAX],0E
011D1061 √ EB 0A   JMP SHORT AES_Pola.011D106D
011D1063 > B8 00F8FFFF MOV EAX,-800
011D1068 √ E9 0B060000 JMP AES_Pola.011D1678
011D106D > 8B4D 08  MOV ECX,DWORD PTR SS:[EBP+8]
011D1070 . 83C1 08  ADD ECX,8
011D1073 . 894D FC  MOV DWORD PTR SS:[EBP-4],ECX
011D1076 . 8B55 08  MOV EDX,DWORD PTR SS:[EBP+8]
011D1079 . 8B45 FC  MOV EAX,DWORD PTR SS:[EBP-4]
011D107C . 8942 04  MOV DWORD PTR DS:[EDX+4],EAX
011D107F . C745 F8 000000 MOV DWORD PTR SS:[EBP-8],0
011D1086 √ EB 09   JMP SHORT AES_Pola.011D1091
011D1088 > 8B4D F8  MOV ECX,DWORD PTR SS:[EBP-8]
011D108B . 83C1 01  ADD ECX,1
011D108E . 894D F8  MOV DWORD PTR SS:[EBP-8],ECX
011D1091 > 8B55 10  MOV EDX,DWORD PTR SS:[EBP+10]
011D1094 . C1FA 05  SAR EDX,5
011D1097 . 3955 F8  CMP DWORD PTR SS:[EBP-8],EDX

```

```

Dump - 0020B000..0020FFFF
0020F97F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0020F98F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0020F99F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0020F9AF 00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .....
0020F9BF 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .....
0020F9CF 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .....
0020F9DF 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .....
0020F9EF 01 53 65 63 75 72 65 50 61 73 73 77 6F 72 64 3A @SecurePassword:
0020F9FF 6E 62 75 31 32 33 00 10 01 58 72 10 01 5C 72 10 nbu123.#0Xr#0\r#
0020FA0F 01 60 72 10 01 00 00 00 00 20 FA 20 00 F8 2F 1D 0*#0...0/##
0020FA1F 01 60 FA 20 00 05 32 1D 01 01 00 00 00 00 08 E1 4F 0*0_#2#00...iB0
0020FA2F 00 F8 7C 4F 00 7C 75 74 0B 00 00 00 00 00 00 00 ..°10.lut...4..X0
0020FA3F 00 00 E0 FD 7E 00 00 00 00 34 FA 20 00 58 01 00 ..d²"...4..X0
0020FA4F 00 9C FA 20 00 89 36 1D 01 34 CD 49 DA 00 00 00 ..é. .é6#04=I r...
0020FA5F 00 6C FA 20 00 AA 33 68 76 00 E0 FD 7E AC FA 20 ..l. .-3hu.d²"%
0020FA6F 00 F2 9E EE 76 00 E0 FD 7E 77 18 E1 52 00 00 00 ..=x~v.d²"w†B R...
0020FA7F 00 00 00 00 00 00 E0 FD 7E 00 00 00 00 00 00 00 .....
0020FA8F 00 00 00 00 00 78 FA 20 00 00 00 00 00 FF FF FF .....x:.....
0020FA9F FF 05 71 F2 76 EB 27 2C 24 00 00 00 00 C4 FA 20 ..'q=vU',$,.....-
0020FAAF 00 C5 9E EE 76 6D 32 1D 01 00 E0 FD 7E 00 00 00 ..+x~vm2#0.d²"...

```

Alternative to hardcoded passwords/keys

- Don't use passwords 😊
- Ask the user for a password
- Keep secrets in a separate file
- Encrypt stored secrets
- Store secrets in protected database
- Use already existing authentication credentials
- CERN guidelines
 - https://security.web.cern.ch/security/recommendations/en/password_alternatives.shtml



Group activity

- Form group of 3-4 members (mix, not your neighbours)
 - Introduce yourself with your name
- Discuss and write down on paper:
 - What method(s) you use for authentication (password...)
 - Is server using other authentication factor?
 - How you store the authentication secret? (brain-only...)
- Time limit: 5 minutes

- Now return back to your original seat (if you wish 😊)



Activity:

- Think about one or two surprising things from this lecture
- I want to hear at least 5 of these, tell me please 😊