

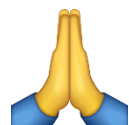
PV204 Security technologies



Authentication: passwords, OTP, FIDO U2F

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/1NWVvbe2ZB_kxAiBEAfw0Rae8Nd_3ZQye/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 (every Monday, 17-18:00)**

Join at
slido.com
#pv204_2021

COURSE TRIVIA:

PV204_00_COURSEOVERVIEW_2021.PPT

AUTHENTICATION & AUTHORIZATION

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...

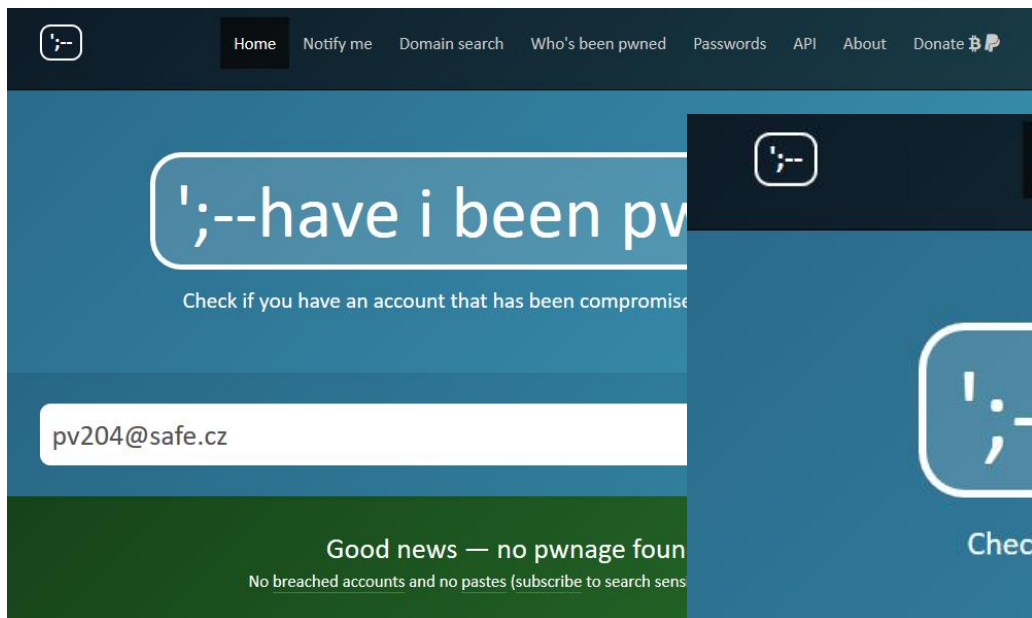
Problems associated with passwords

- 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)
- ...

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

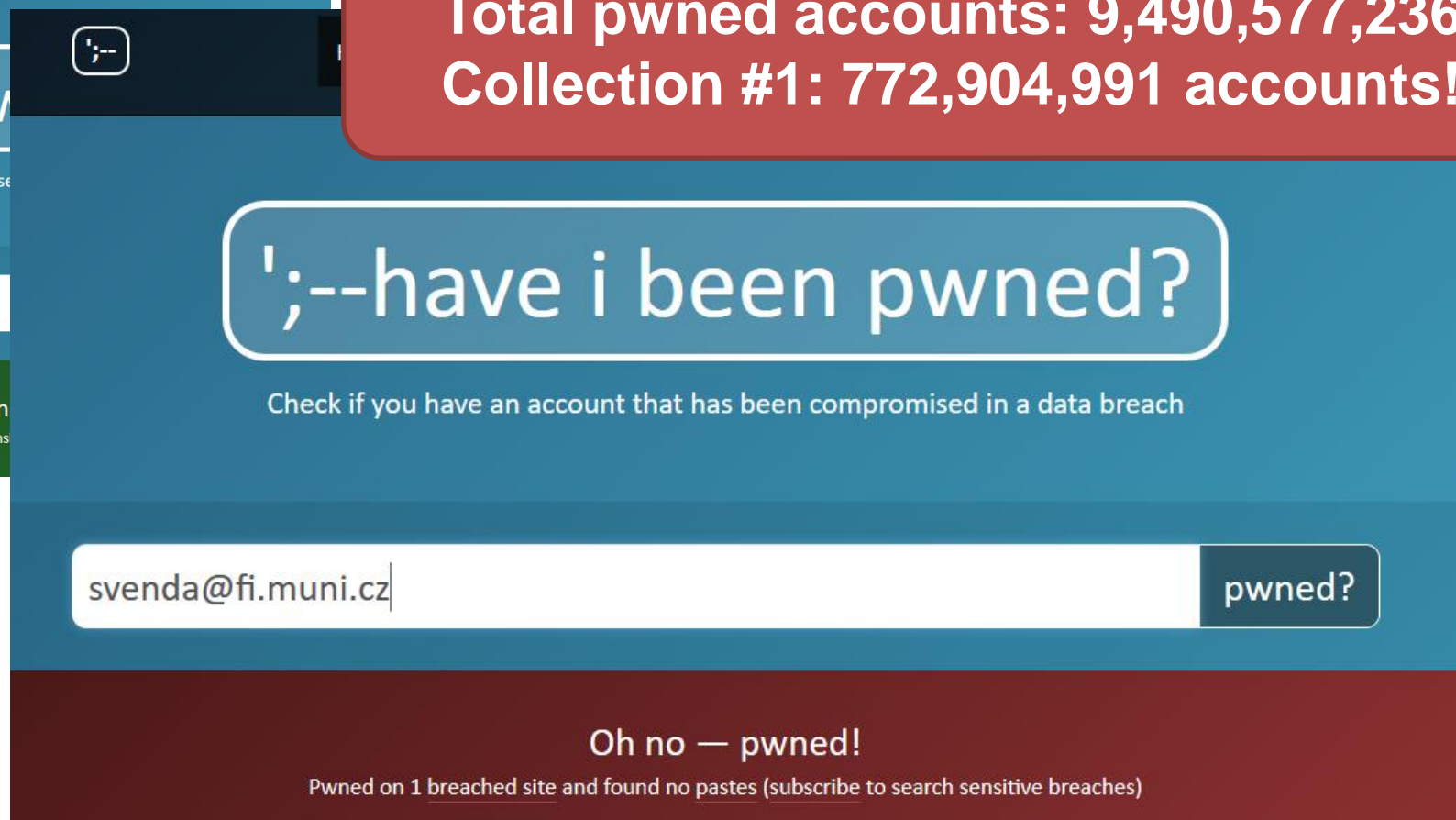
';--have i been pwned?

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: 9,490,577,236
Collection #1: 772,904,991 accounts!



';--have i been pwned?

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

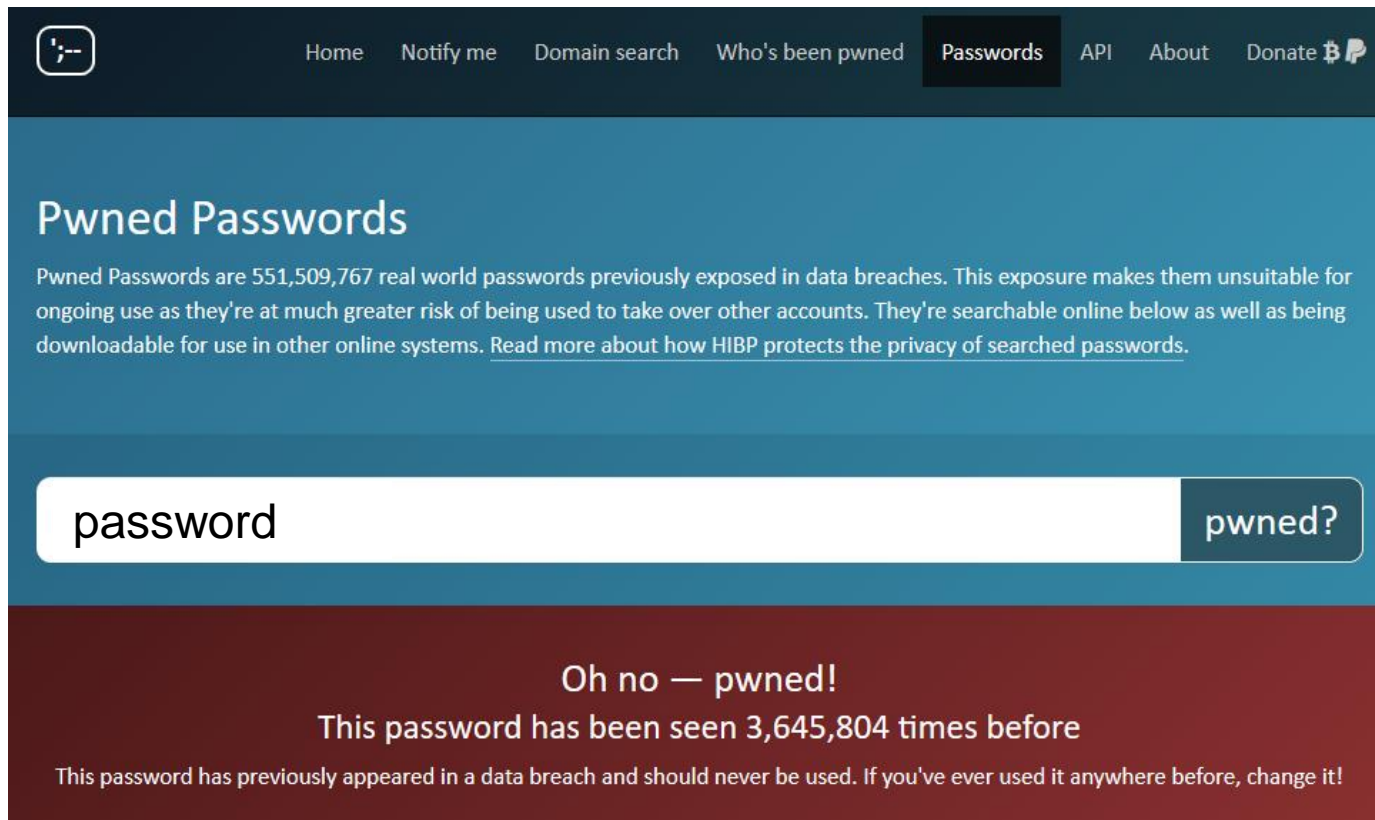
svenda@fi.muni.cz pwned?

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



The screenshot shows the 'Passwords' page of the Have I Been Pwned website. The navigation bar includes links for Home, Notify me, Domain search, Who's been pwned, Passwords (highlighted), API, About, and Donate. The main heading is 'Pwned Passwords', followed by a paragraph explaining that 551,509,767 real-world passwords were exposed in data breaches. Below this is a search bar containing the word 'password' and a 'pwned?' button. The result area is dark red and displays the message: '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/xJzUOGFEINz0:13226:0:99999:7:::
```

```
[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 (inputs 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: 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
 - Gawker/root.com leak: 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



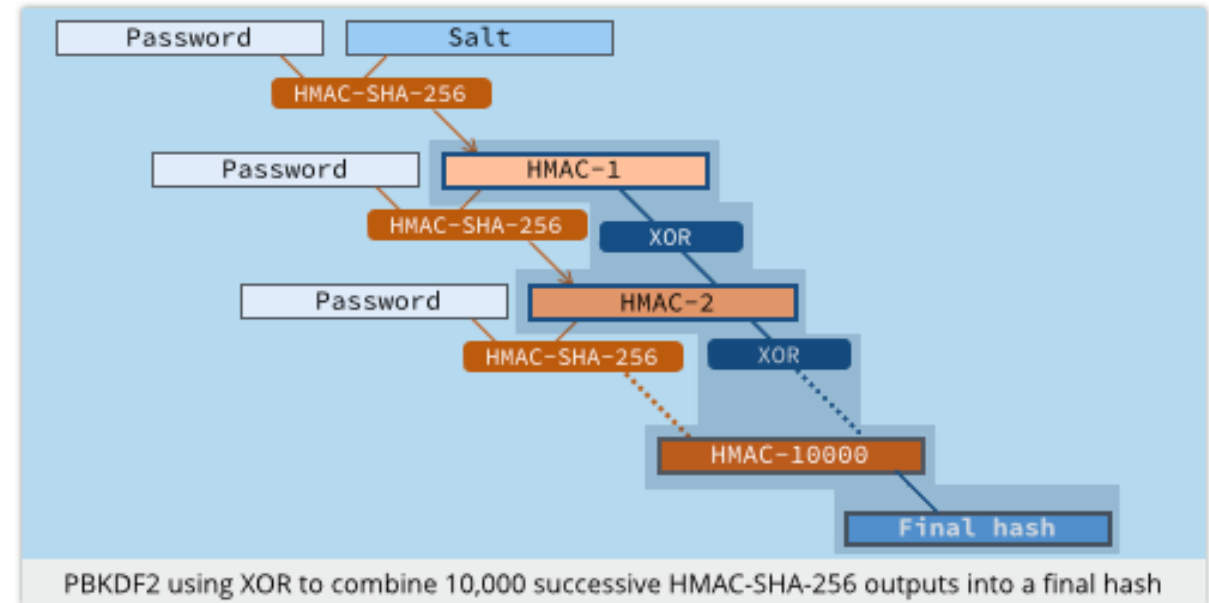
Some issues addressed by PAKE (Password Authenticated Key Exchange) protocols – future lecture



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>

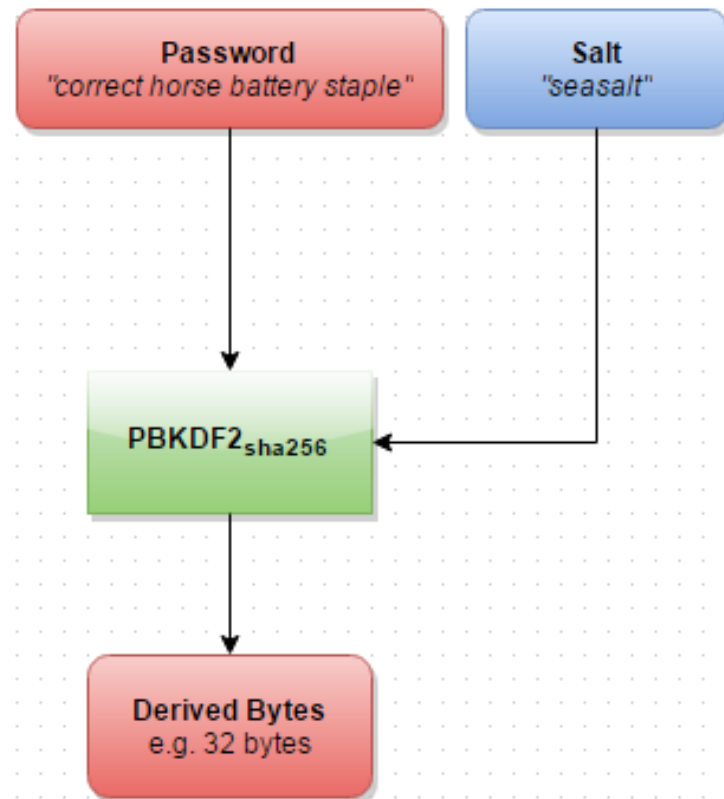
- Problem with custom-build hardware (GPU, ASIC)
 - Repeated iterations not enough to prevent bruteforce
 - (or would be too slow on standard CPU – user experience)
- Solution: function which requires large amount of memory

scrypt – memory hard function

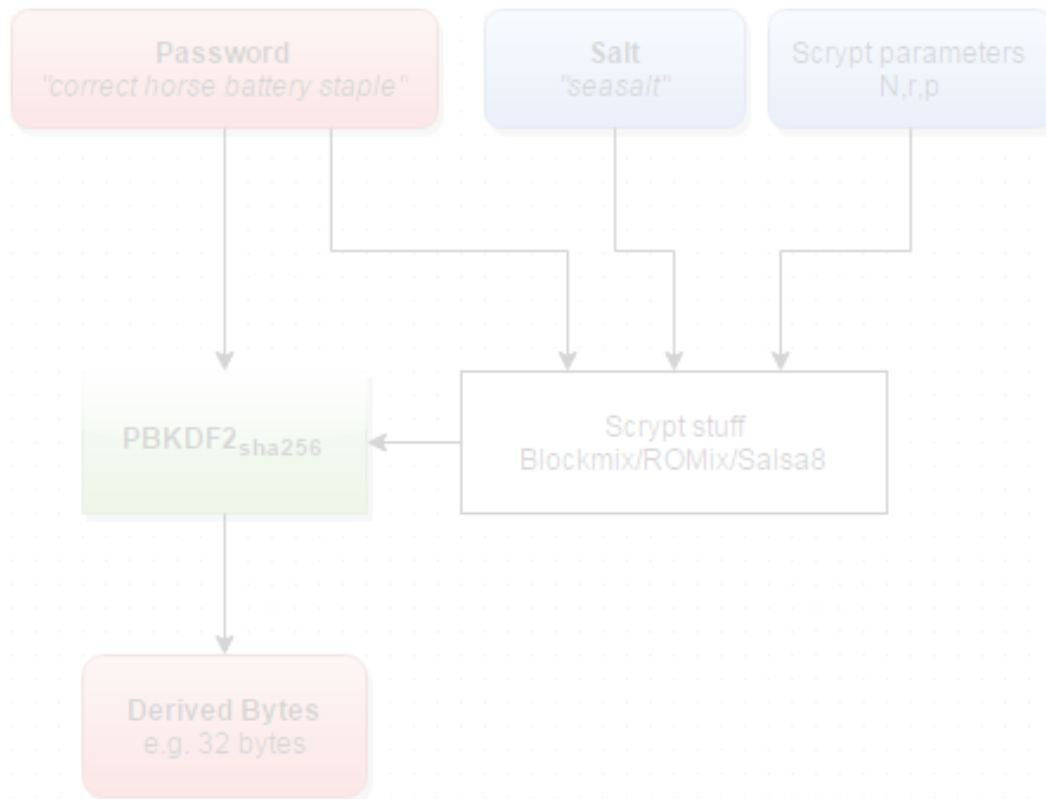
- Design as a protection against cracking hardware (usable against PBKDF2)
 - GPU, FPGA, ASICs...
 - <https://github.com/wg/scrypt/blob/master/src/main/java/com/lambdaworks/crypto/SCrypt.java>
- Memory-hard function
 - Force computation to hold r (parameter) blocks in memory
 - Uses PBKDF2 as outer interface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

Reuse of external PBKDF2 structure

Standard PBKDF2



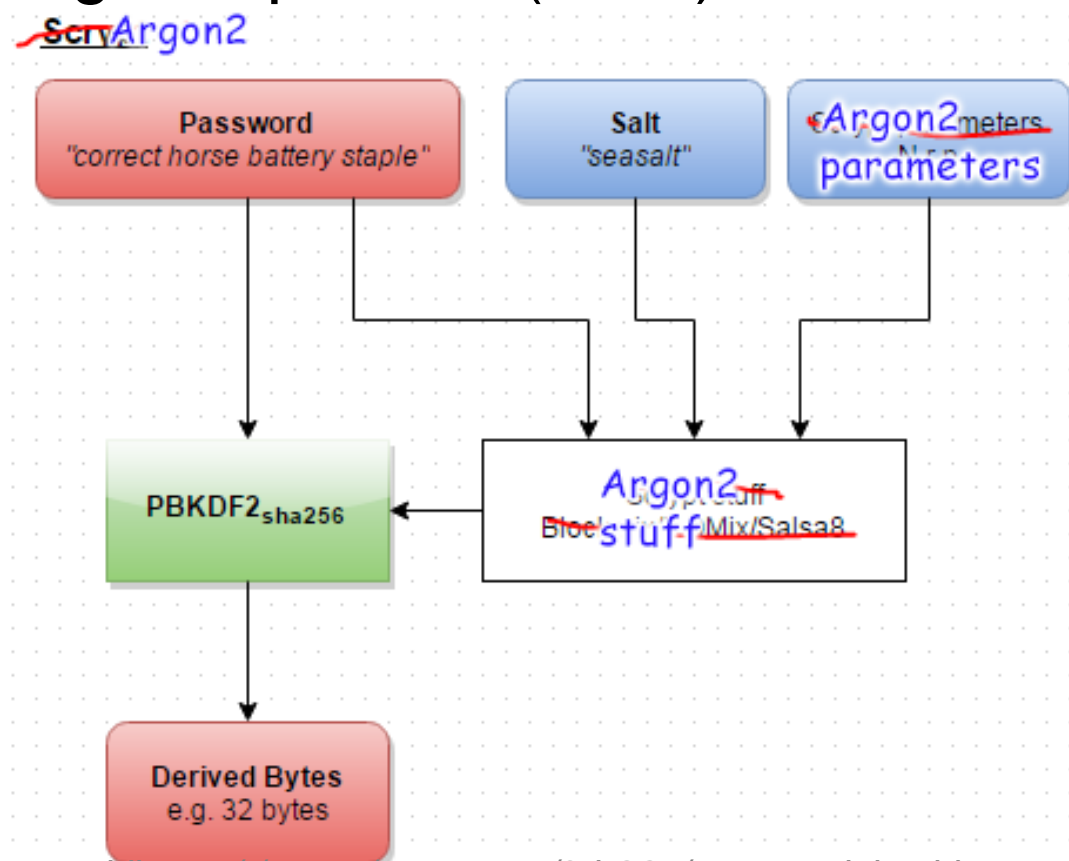
Script



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

Argon2

- Password hashing competition (PHC) winner, 2013



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

Problem solved?

- To: [cfrg at irtf.org](mailto:cfrg@irtf.org)
- Subject: [Cfrg] Argon2i, scrypt, balloon hashing, ...
- From: Phillip Rogaway <[rogaway at cs.ucdavis.edu](mailto:rogaway@cs.ucdavis.edu)>
- Date: Mon, 15 Aug 2016 15:37:21 -0700 (Pacific Daylight Time)
- Archived-at: <<https://mailarchive.ietf.org/arch/msg/cfrg/Xu9hCT6dqVmD50CezeR1MFsos0o>>
- Delivered-to: [cfrg at ietfa.amsl.com](mailto:cfrg@ietf.org)
- In-reply-to: <mailman.995.1471241877.1171.cfrg@irtf.org>
- List-archive: <<https://mailarchive.ietf.org/arch/browse/cfrg/>>
- List-help: <<mailto:cfrg-request@irtf.org?subject=help>>
- List-id: Crypto Forum Research Group <cfrg.irtf.org>
- List-post: <<mailto:cfrg@irtf.org>>
- List-subscribe: <<https://www.irtf.org/mailman/listinfo/cfrg>>, <<mailto:cfrg-request@irtf.org?subject=subscribe>>
- List-unsubscribe: <<https://www.irtf.org/mailman/options/cfrg>>, <<mailto:cfrg-request@irtf.org?subject=unsubscribe>>
- References: <mailman.995.1471241877.1171.cfrg@irtf.org>
- User-agent: Alpine 2.00 (WNT 1167 2008-08-23)

Problem: situation with PHC winner still unclear in 2019 ☹

I would like to gently suggest the CFRG not move forward with blessing any memory-hard hash function at this time. The area seems too much in flux, at this time, for this to be desirable. Really nice results are coming out apace. Standards can come too early, you know, just as they can come out too late.

phil

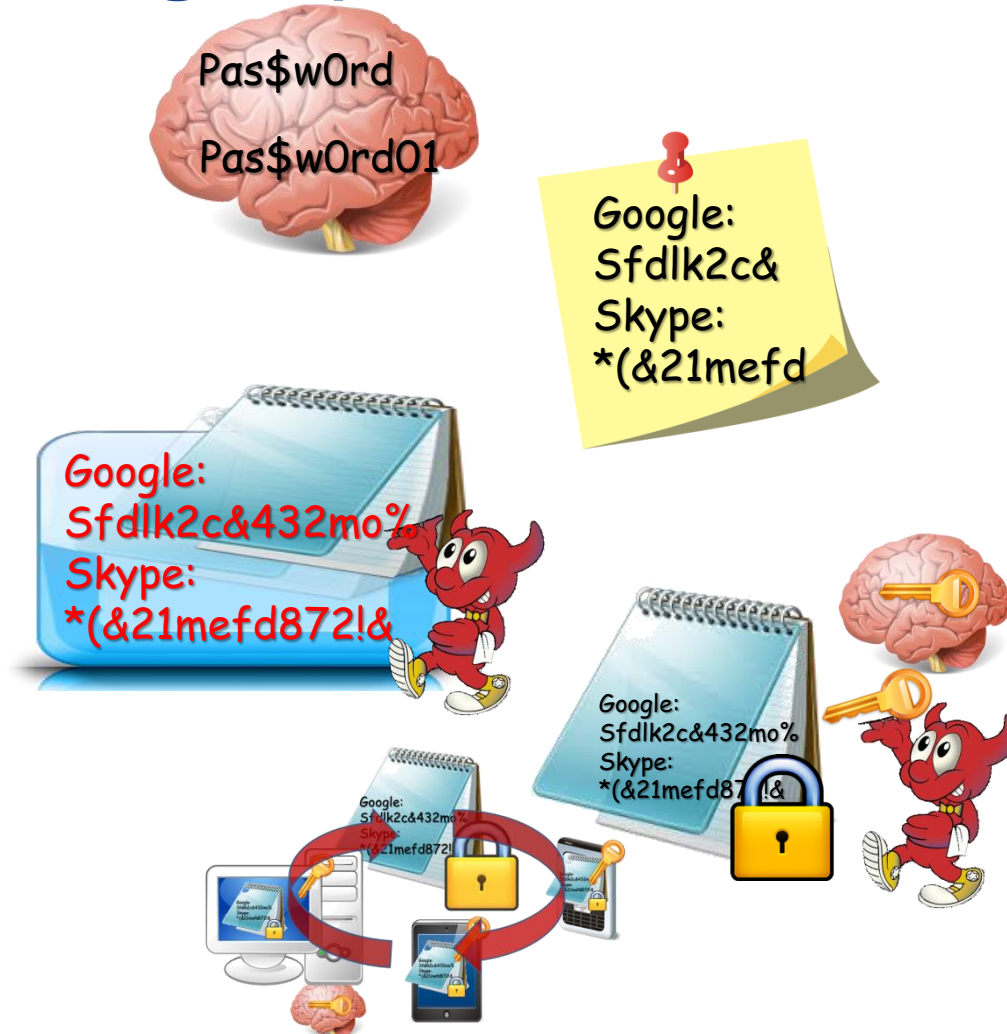
<https://www.ietf.org/mail-archive/web/cfrg/current/msg08439.html>

**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



- Firefox Lockwise <https://www.mozilla.org/en-US/firefox/lockwise/>
 - Part of the standard Firefox installation, sync between devices
 - Automatically checks for password leakage (Firefox Monitor)

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 may have been too "alarmist" in sounding the alarm bell over the potential security breach. But the anomaly 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.



But passwords are encrypted, right?



Case study

PASSWORD MANAGER FOR MULTIPLE DEVICES

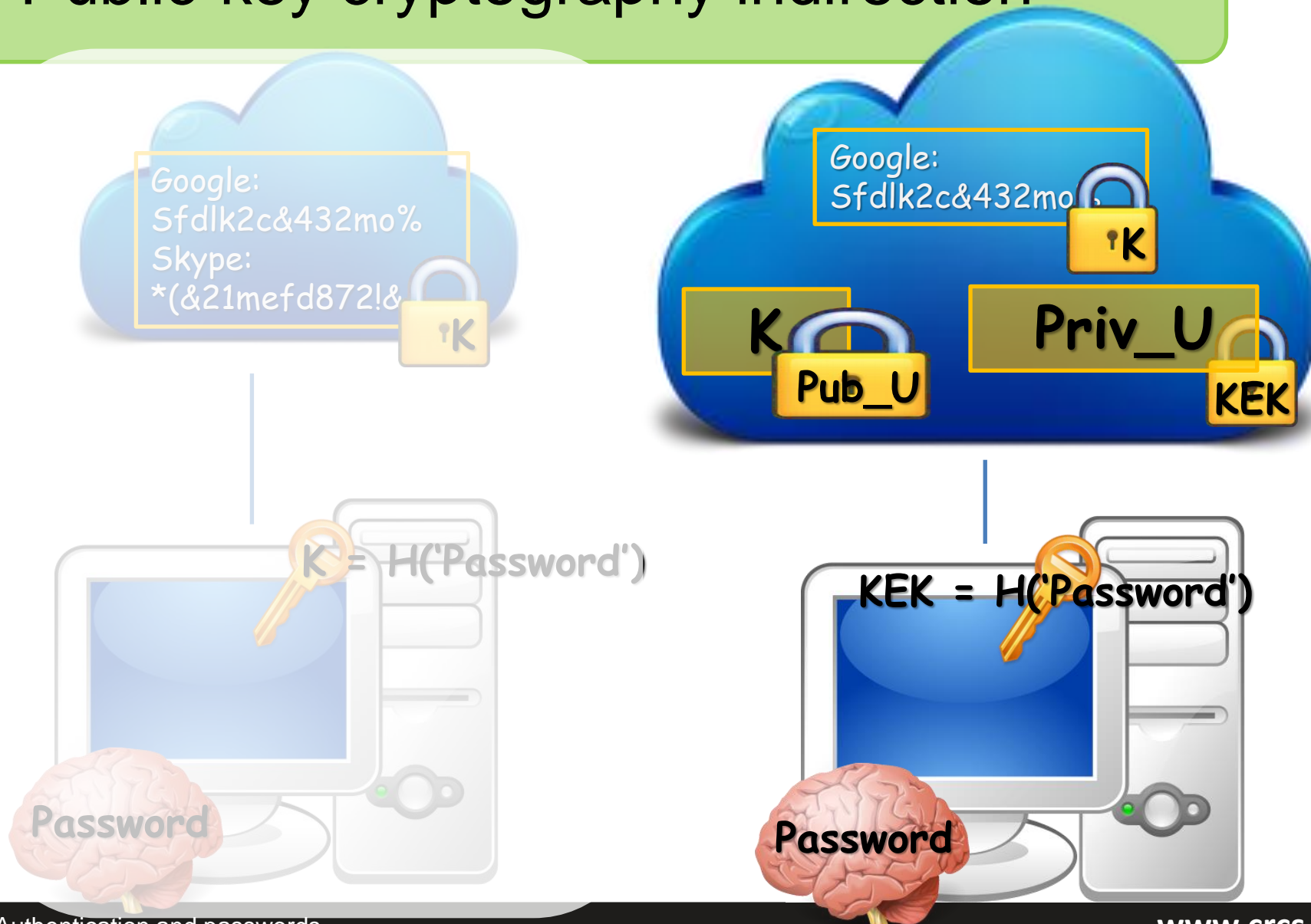
Functional and security assumptions

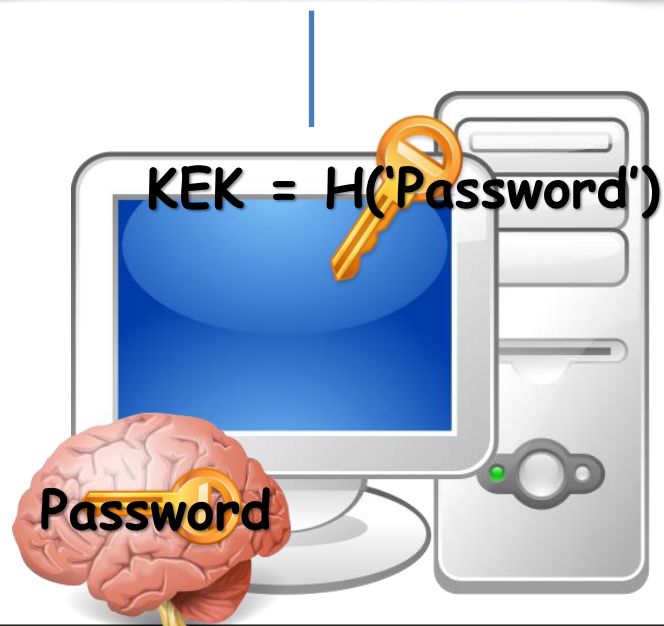
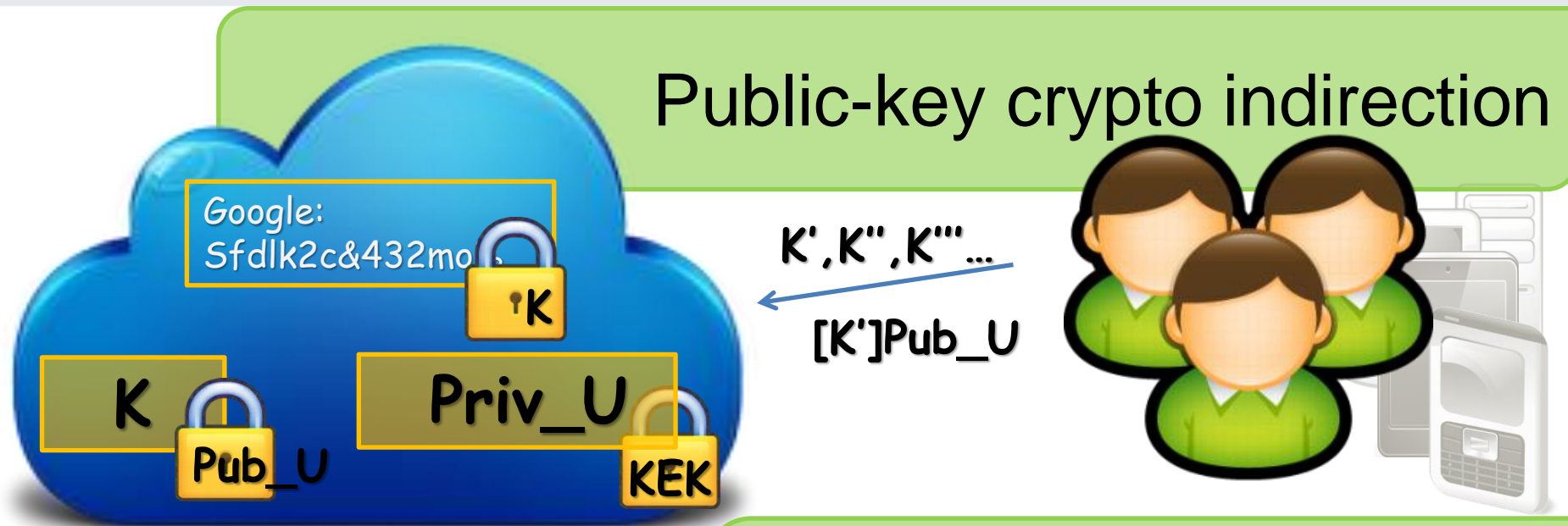
- Functional
 - User stores fixed secrets (passwords...)
 - User has multiple connected devices
 - Easy to use 😊
- Security
 - Service can't be trusted
 - User chooses weak password
 - Devices can be lost (and later revoked)
 - User has independent channel (phone)

Main security design principles

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

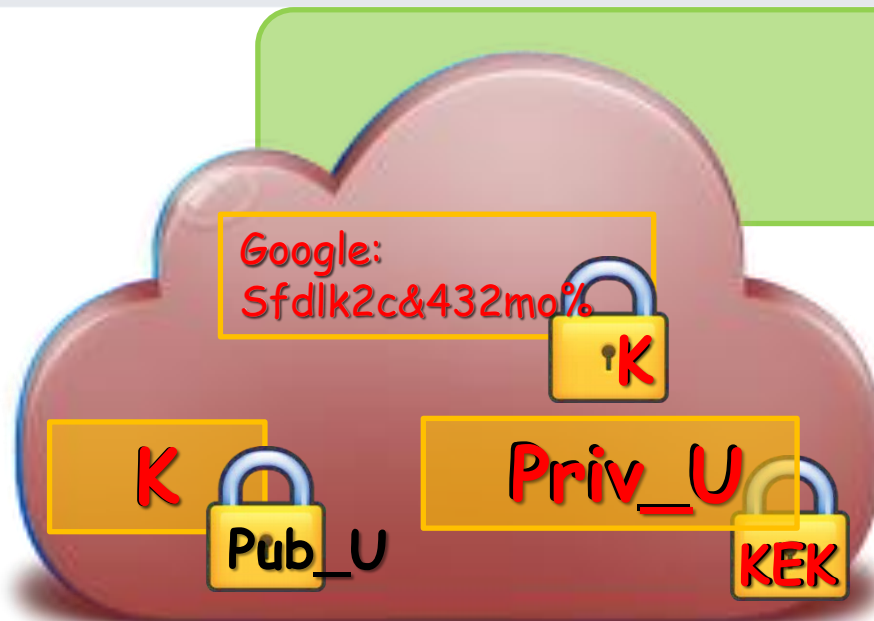
Public-key cryptography indirection





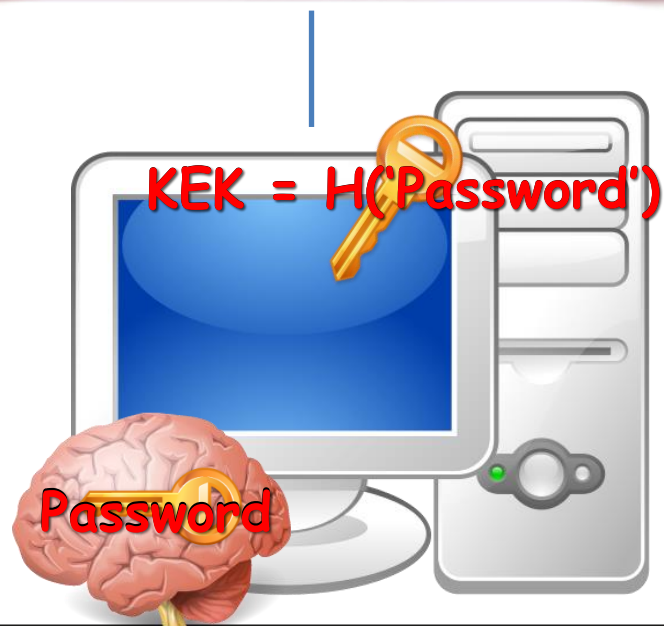
Public-key crypto indirection allows for asynchronous change of K

Long private key can be also stored on Service

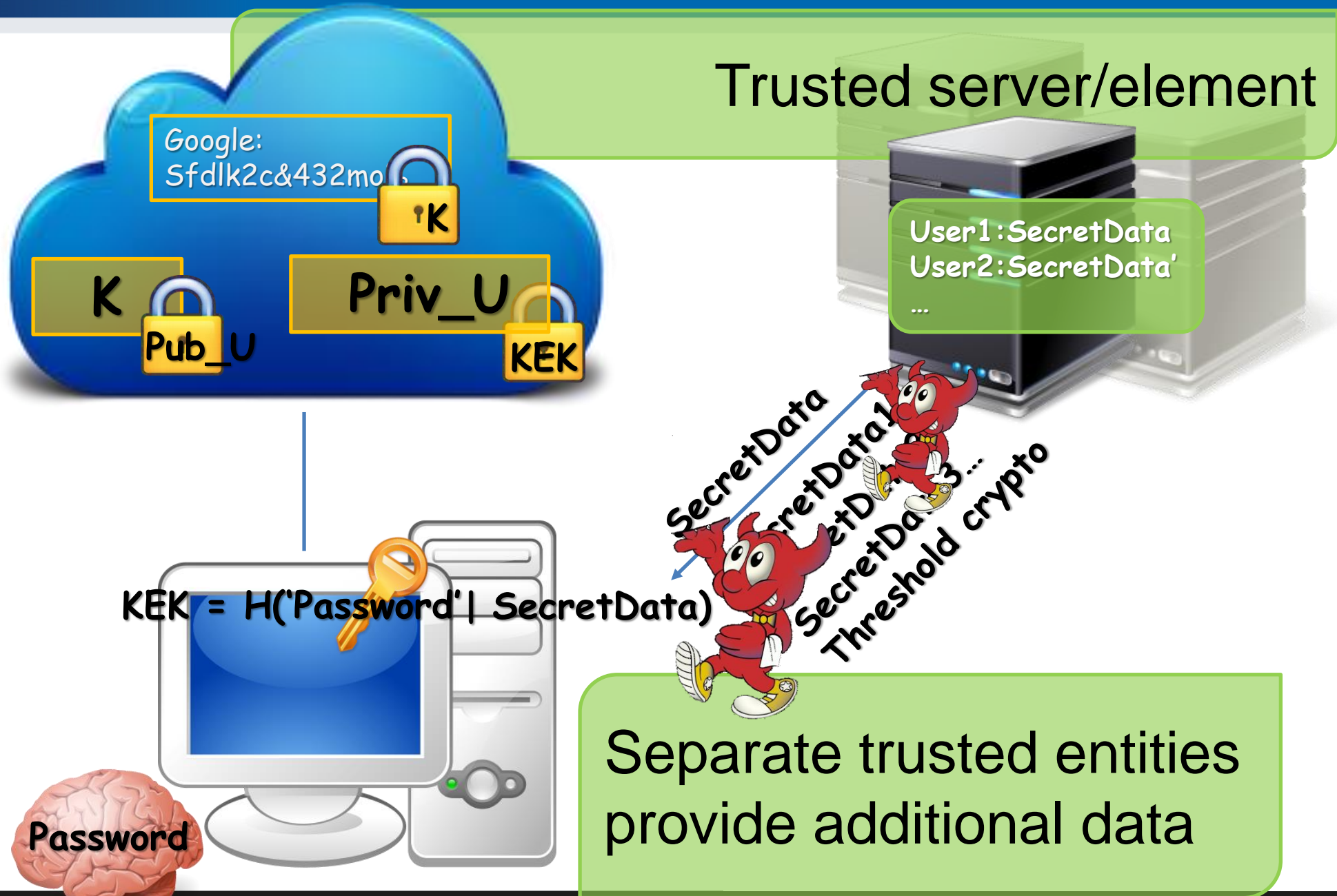


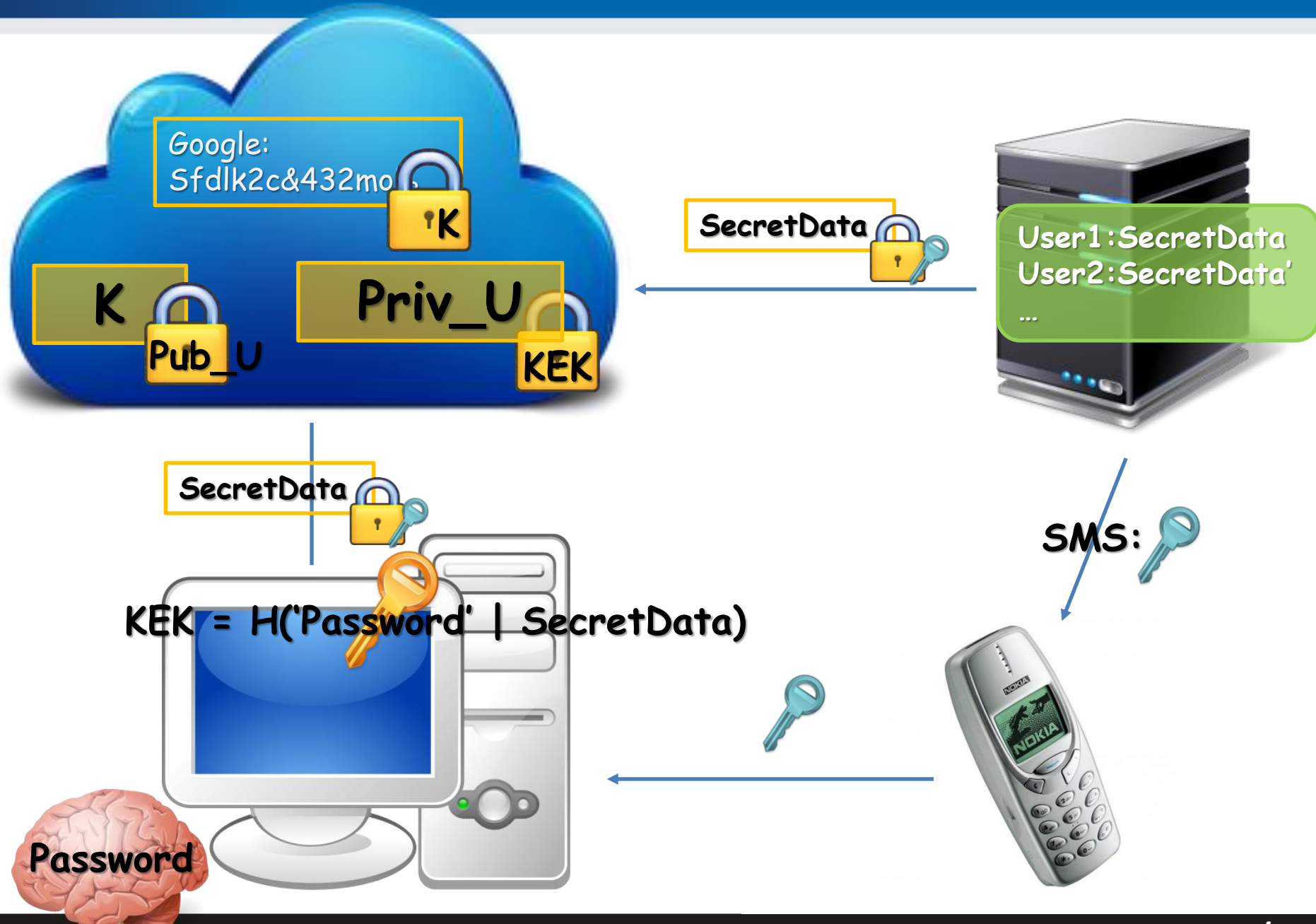
Weak password?

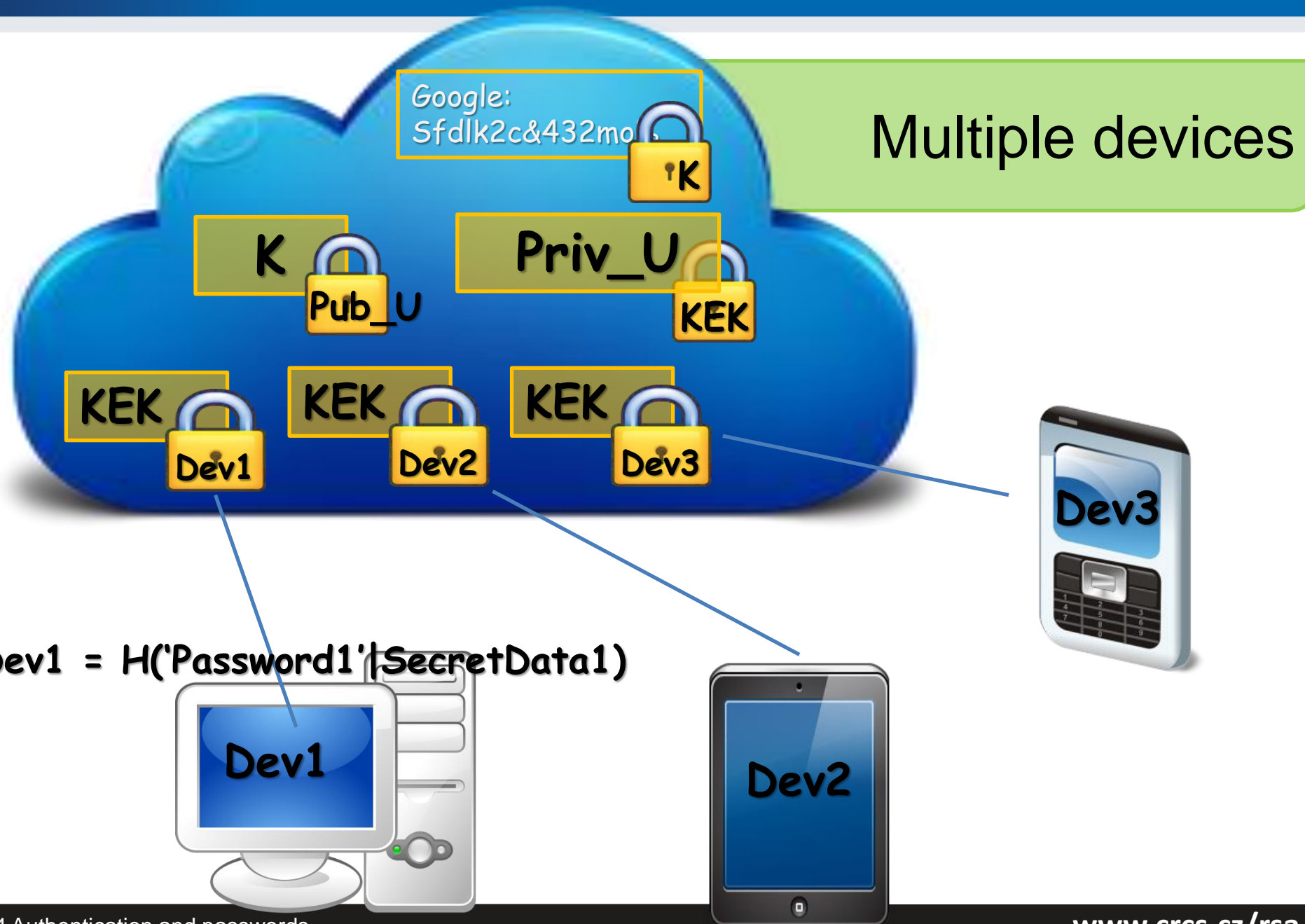
Users tend to have weak passwords...



Attacker has motivation for attacking the Service!







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's service showcased in 2013
- Lack of details until iOS Security report 02/2014
 - 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)



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
 - But limited number of tries allowed
- Trusted component of iCloud realized via internal HSM
 - 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

Recall: Problems associated with passwords

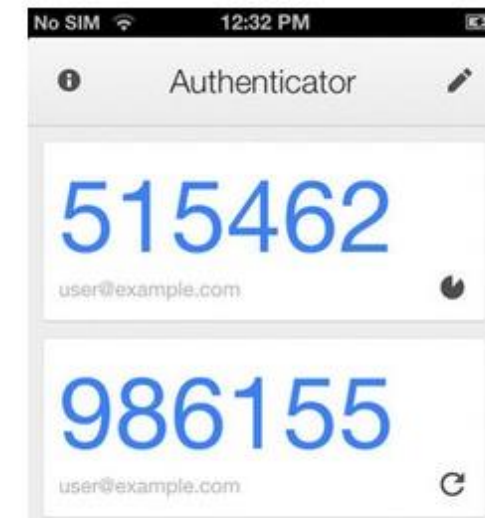
- How to create secure password?
- How to use password securely?
- How to store password securely?
- Same value is used for the long time (exposure)
- Value of password is independent from target operation (e.g., authorization of request)
- ...



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

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

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

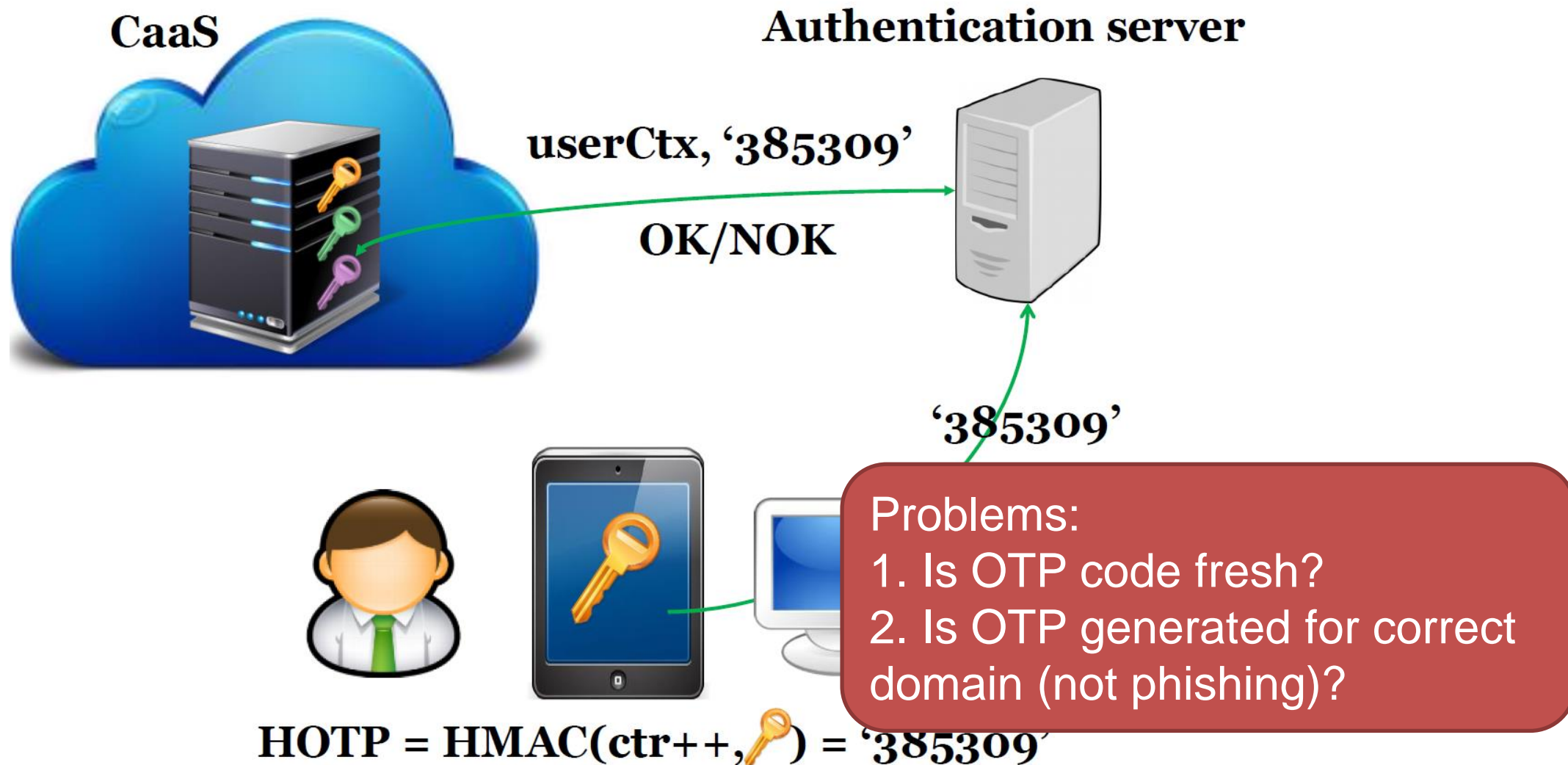
- Improves protection of client side
- Increases risk at Auth. server



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

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



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



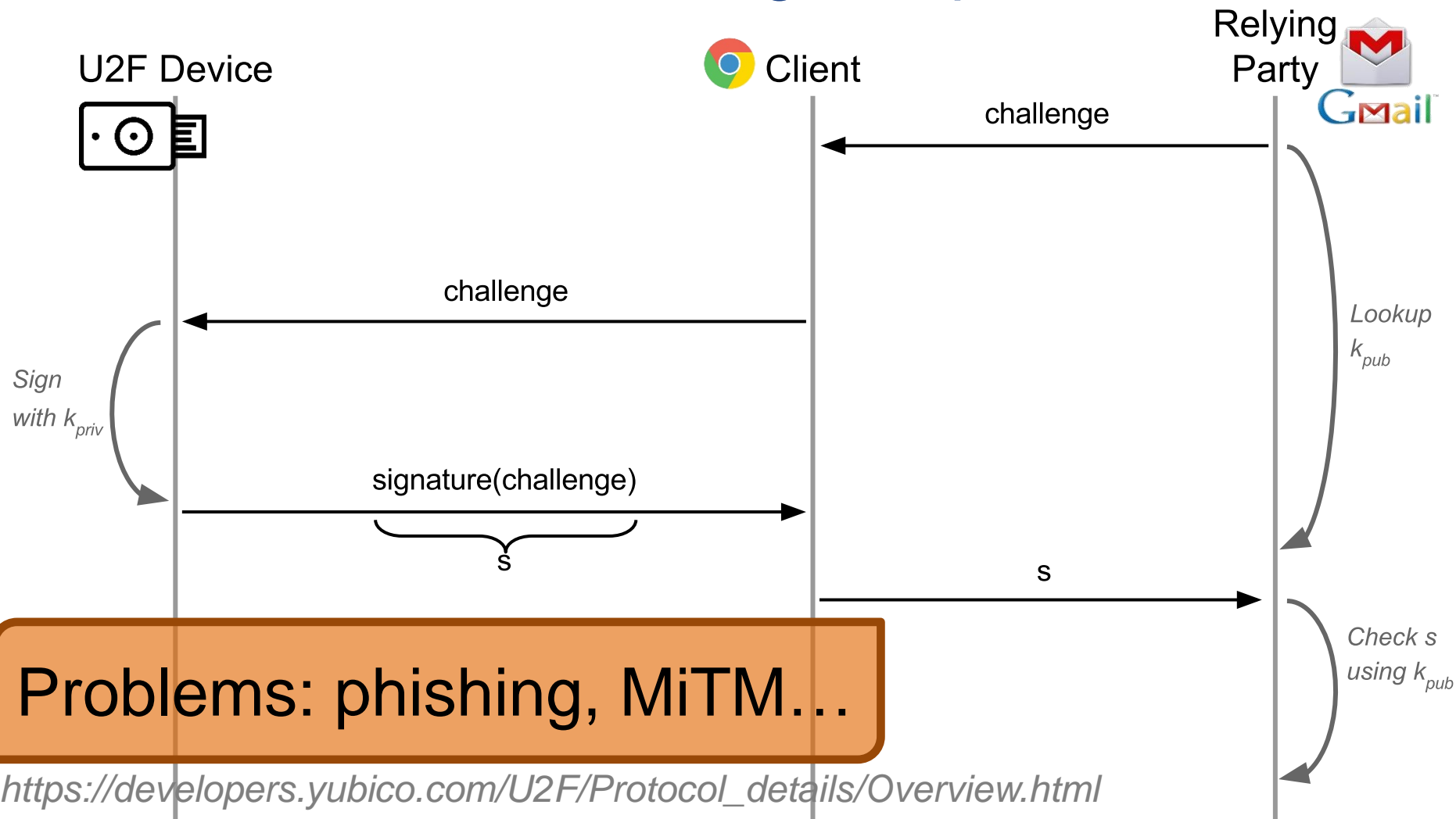
Mandatory reading: UCAM-CL-817

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

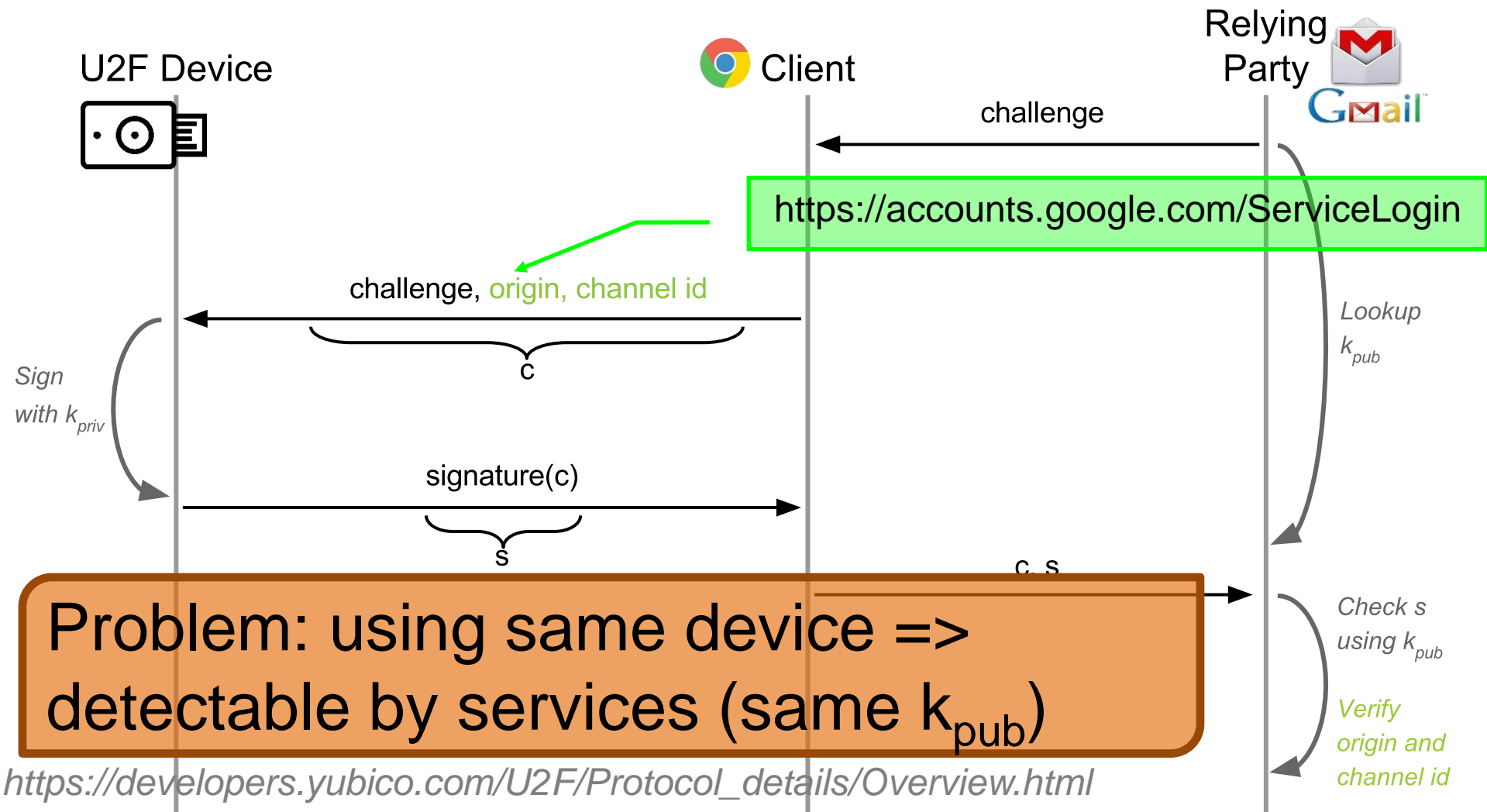
**IDEA: REPLACE PASSWORD BY
SMARTCARD WITH ASYMMETRIC KEYPAIR,
CHALLENGE-RESPONSE PROTOCOL AND
PREVENT PHISHING**

FIDO U2F PROTOCOL

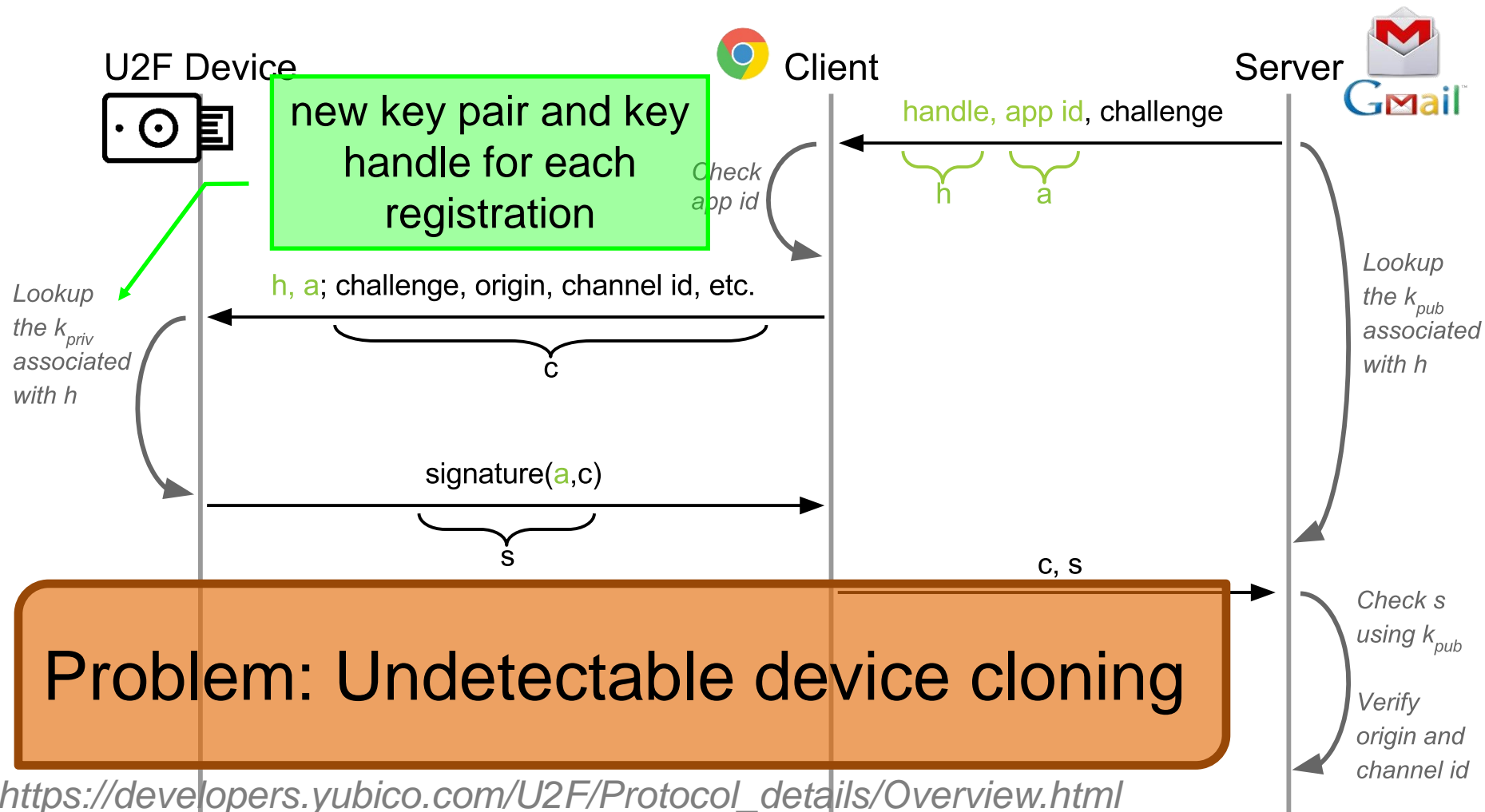
Revision 1: ECC-based challenge-response



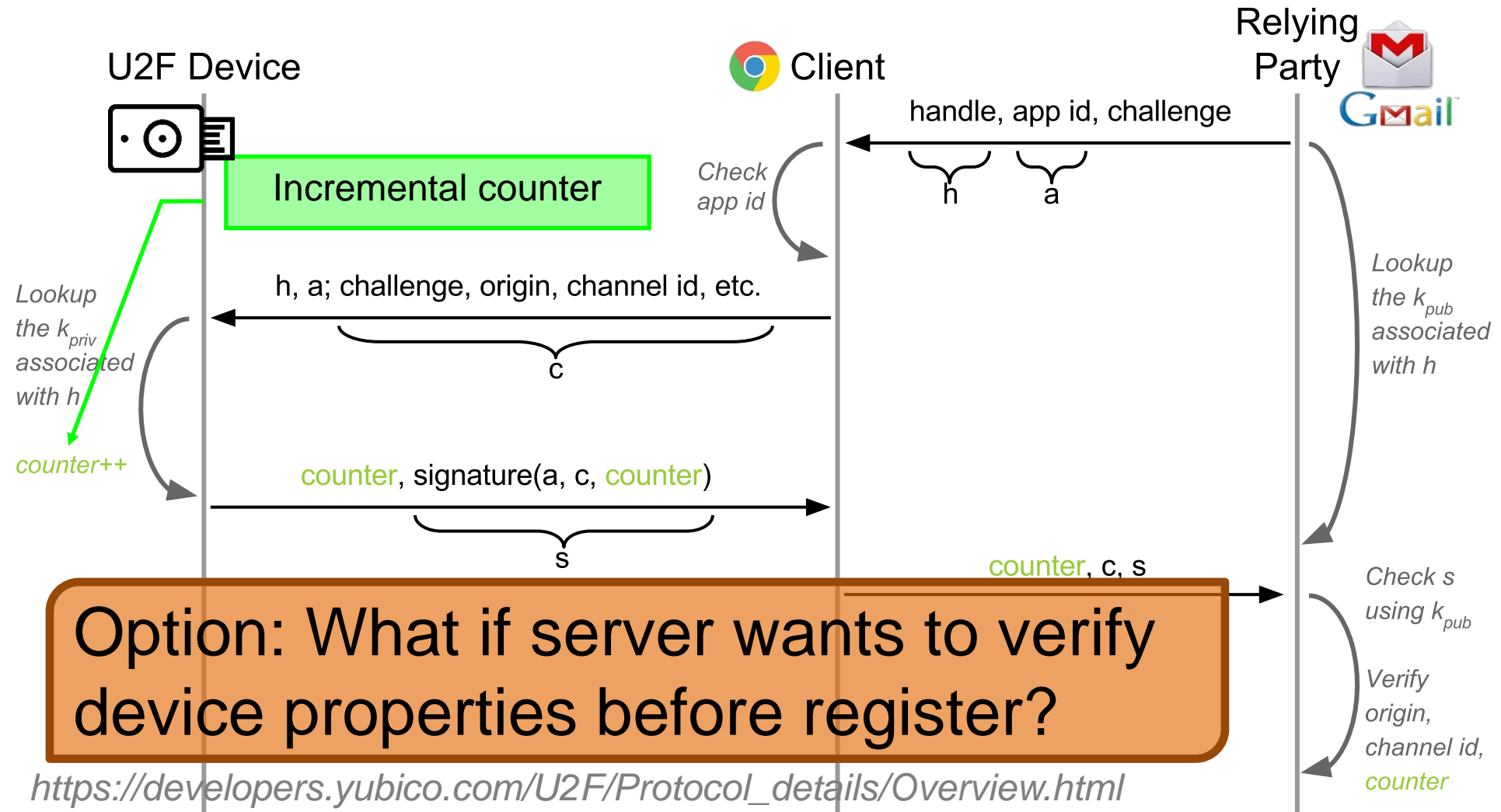
Revision 2: URI + TLS channel id added



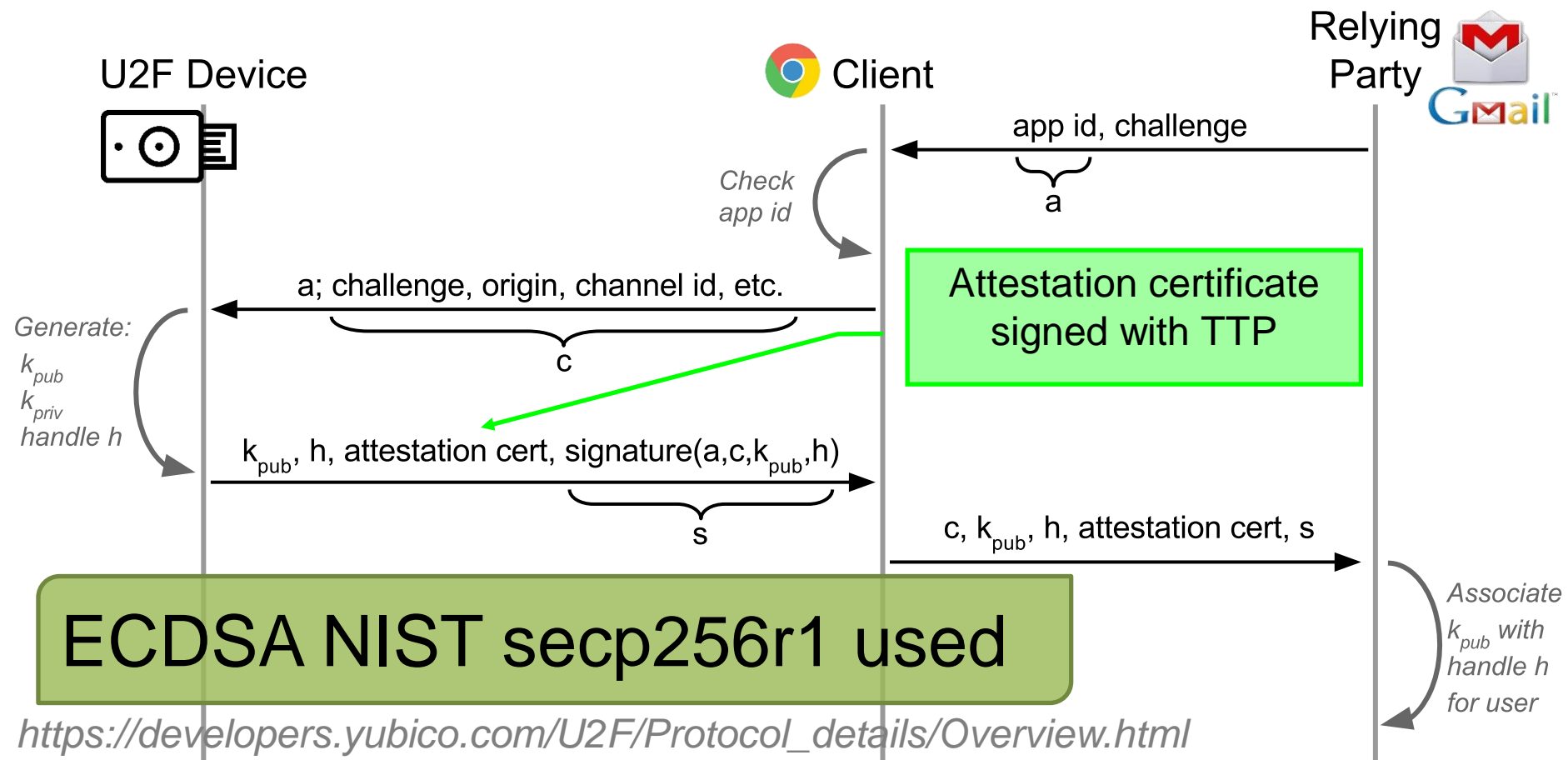
Revision 3: Application-specific key added



Revision 4: Authentication counter added



Revision 5: Device attestation added



FIDO U2F – current state

- FIDO alliance of major companies
- U2F → FIDO2 project (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



FIDO U2F devices

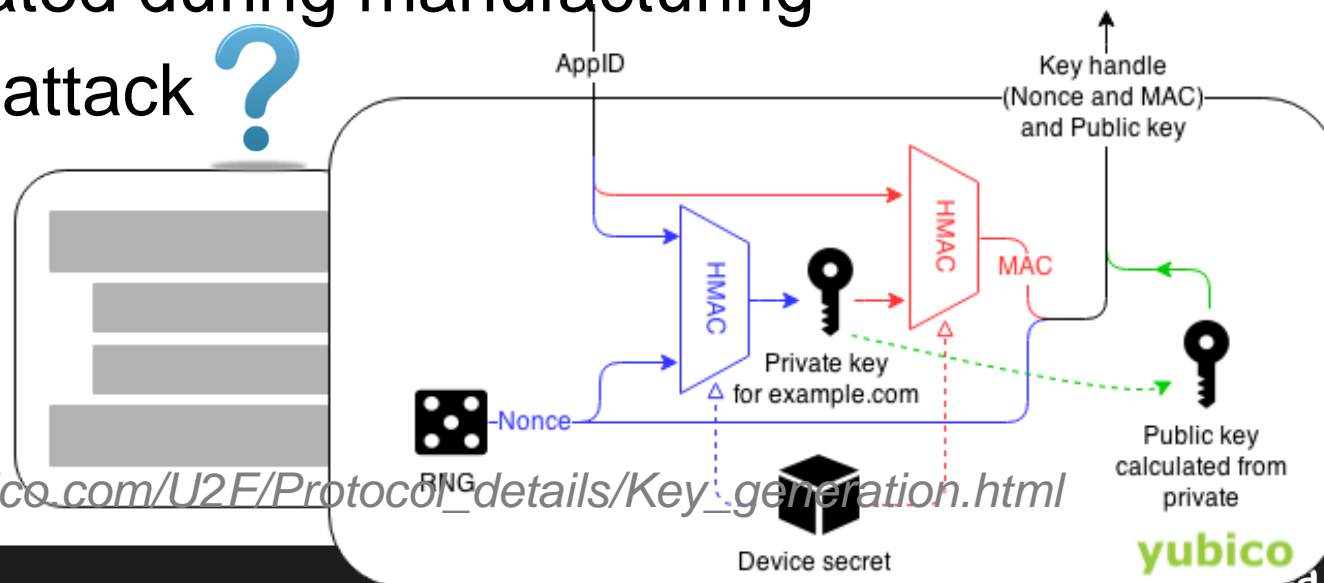


- Why have button? Is missing display problem?
- Recent problem: direct WebUSB API in Chrome
 - Malware bypass U2F API checking the URL
 - Legitimate URL is send from malicious page
 - <https://www.wired.com/story/chrome-yubikey-phishing-webusb/>
 - APDU-level communication: <https://npmccallum.gitlab.io/post/u2f-protocol-overview/>
- 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 ?



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

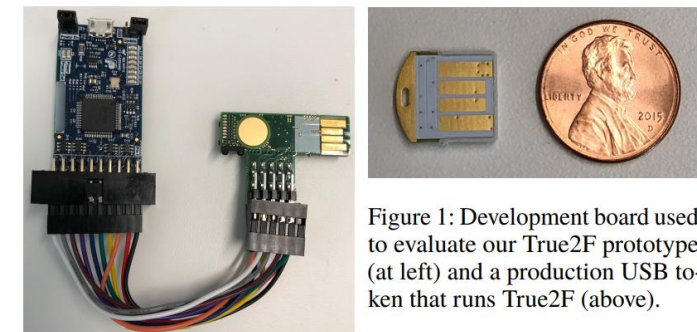
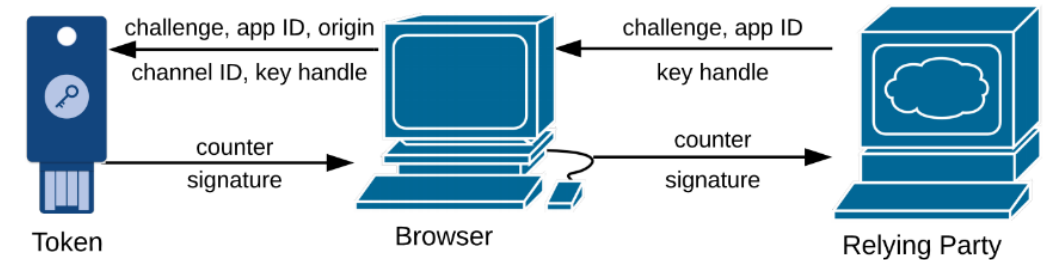


Figure 1: Development board used to evaluate our True2F prototype (at left) and a production USB token that runs True2F (above).

WebAuthn

- An API for accessing Public Key Credentials Level 1
- <https://www.w3.org/TR/webauthn/>
- Similar, but more complex standard than U2F => expect additional problems (not yet scrutinized enough)

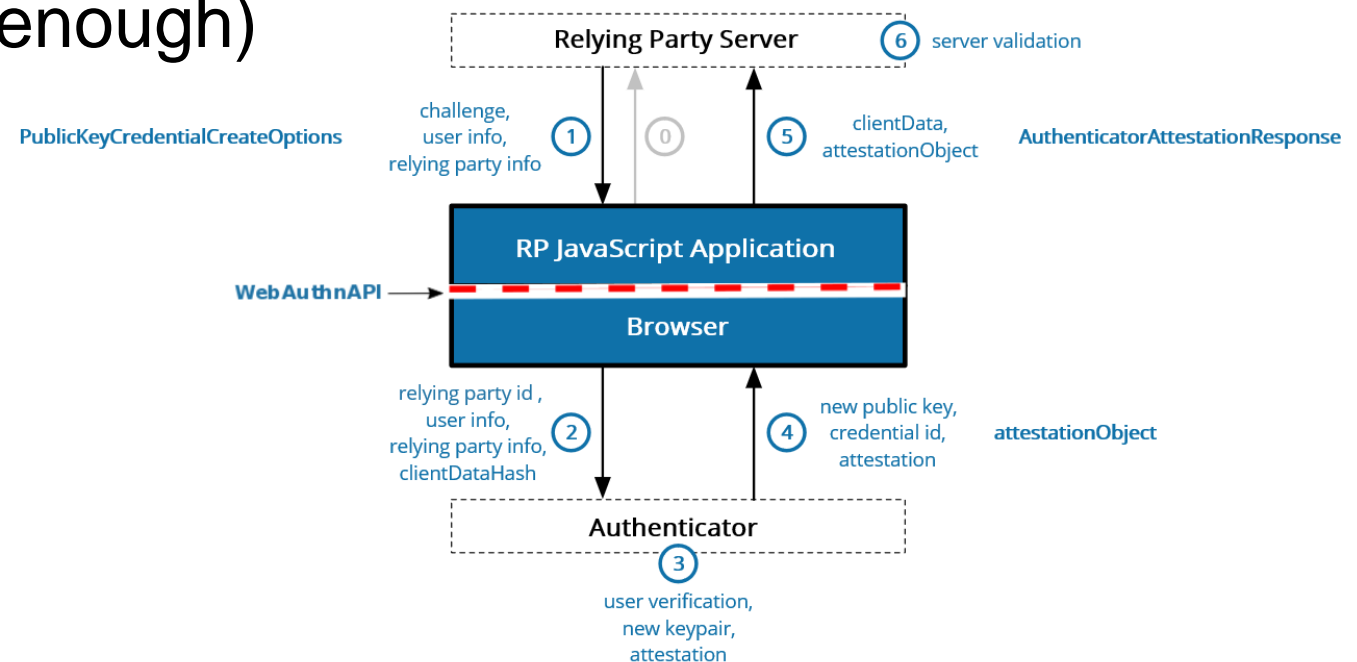


Figure 1 Registration Flow

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)
- **Mandatory reading:** UCAM-CL-817
 - At least chapters: II. Benefits, V. Discussion
 - Whole report is highly recommended



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 (every Monday, 17-18:00)**

Join at
slido.com
#pv204_2021