PV204 Security technologies

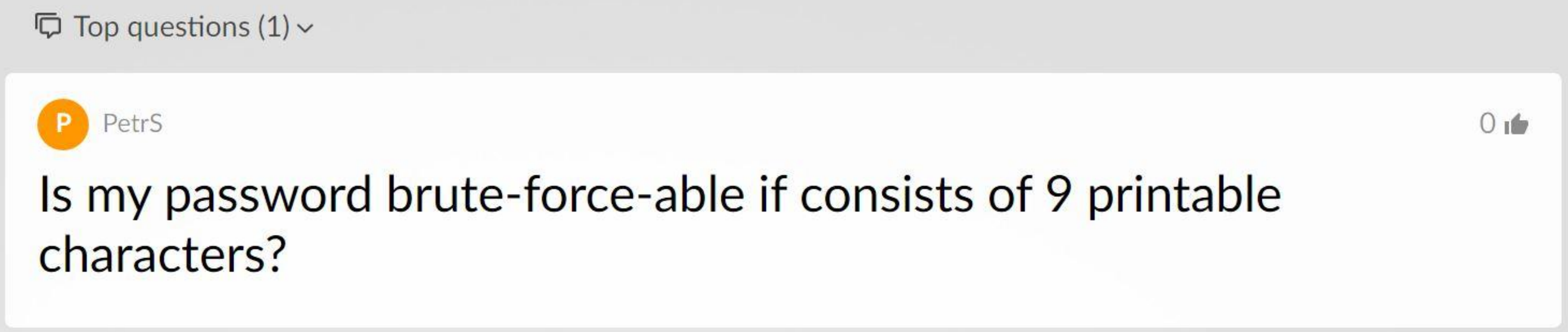
Authentication and secure channels I.

Passwords, Password managers, iVault, OTP, Nostr

Petr Švenda ***svenda@fi.muni.cz*** ***@rngsec***
Centre for Research on Cryptography and Security, Masaryk University

CROCS
Centre for Research on Cryptography and Security

🙏 *Please report any inaccuracies or suggestions for improvements here:*
https://drive.google.com/file/d/1Pb2i01GX8NYC4Q1b_N7SDWlEO8c2PPWy/view?usp=sharing


Top questions (1)

PetrS 0

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

- Place questions, topics and news you would be interested to discuss
- We will together discuss these during every week lecture Q&A (towards the end)

Join at
slido.com

#pv204_2025

COURSE TRIVIA: PV204_00_COURSEOVERVIEW_2025.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)

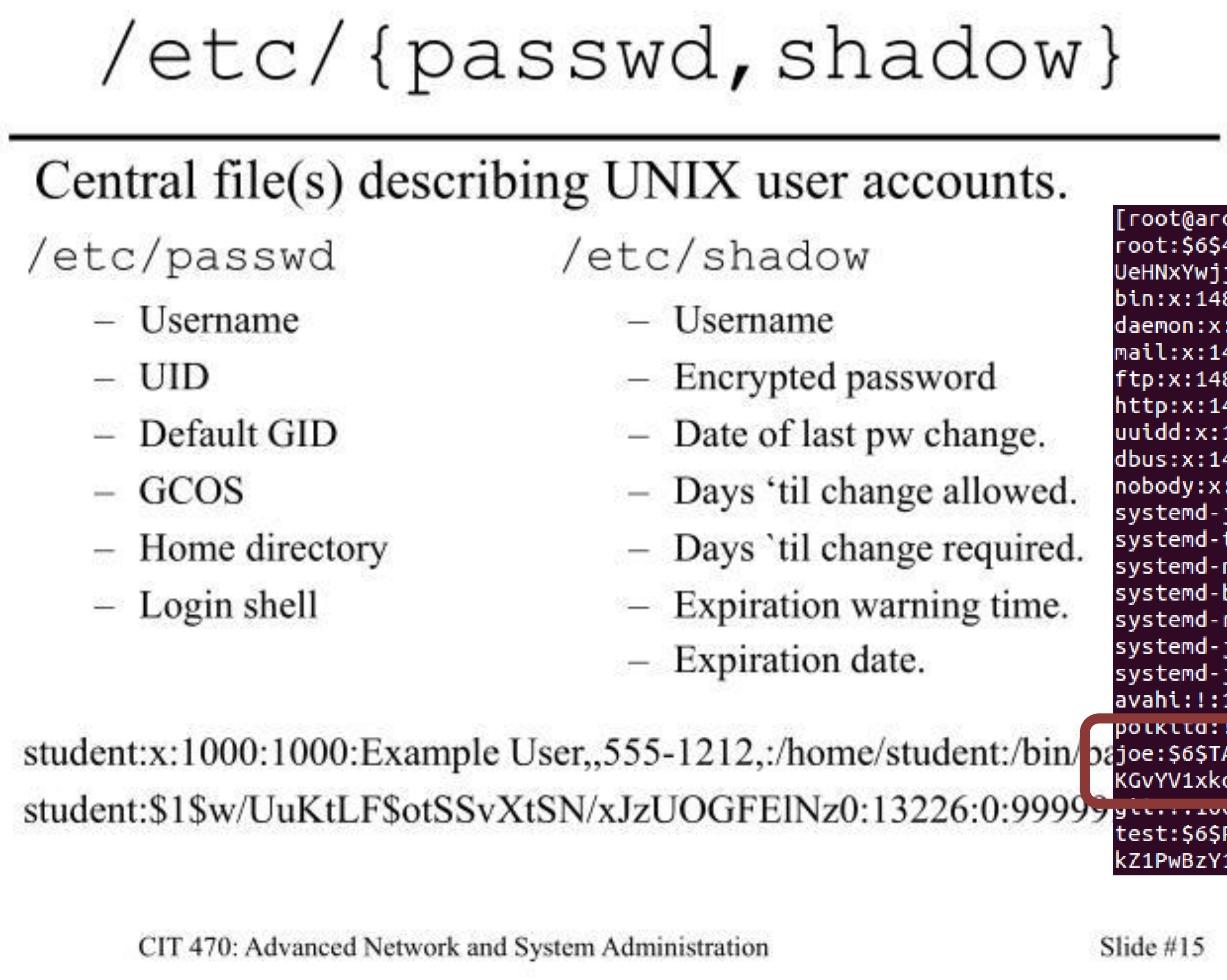
- Difficult to detect compromise and change after the compromise

Password "hardening" ideas

1. Hash password by one-way function (hard(er) 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}

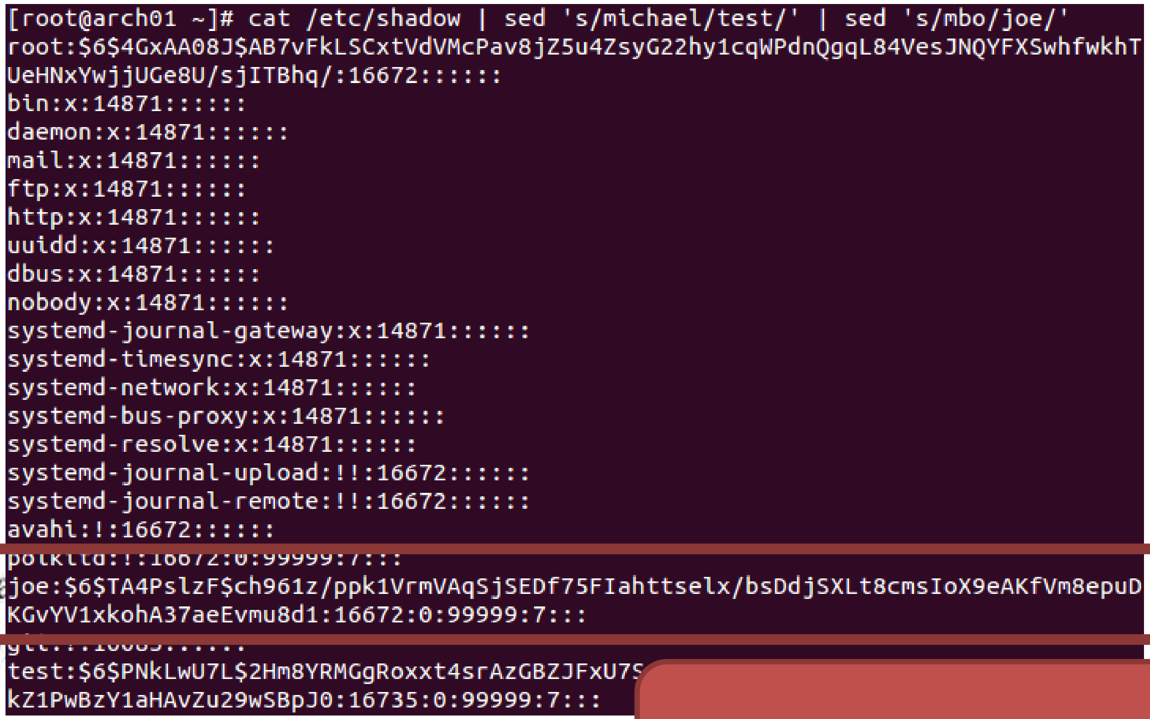
Central file(s) describing UNIX user accounts.

/etc/passwd	/etc/shadow
– Username	– Username
– UID	– Encrypted password
– Default GID	– Date of last pw change.
– GCOS	– Days 'til change allowed.
– Home directory	– Days `til change required.
– Login shell	– Expiration warning time.
	– Expiration date.

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

student:$1$w/UuKtLF$otSSvXtSN/xJzUOGFElNz0:13226:0:9999

CIT 470: Advanced Network and System Administration   Slide #15



```
[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::::::
uuidd: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:::
git:!:[illegible]::::::
test:$6$PNkLwU7L$2Hm8YRMGgRoxxt4srAzGBZJFxU7S
kZ1PwBzY1aHAvZu29wSBpJ0:16735:0:99999:7:::
```

Joe; insecure

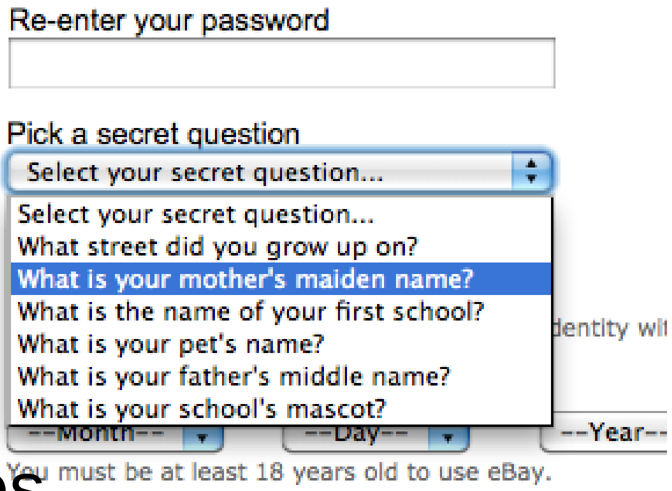
(Hashed-)Password cracking

- 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, energy cost of cracking
- Tools
  - Generic: Hashcat, John the Ripper, Brutus, RainbowCrack…
  - Targeted to application: TrueCrack, Aircrack-NG…

';--have i been pwned?

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

Password reality (from many breaches + pwd cracking)


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?

dentity wi

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

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

- 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% users had exactly 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

Insecure password handling … what is the attack?

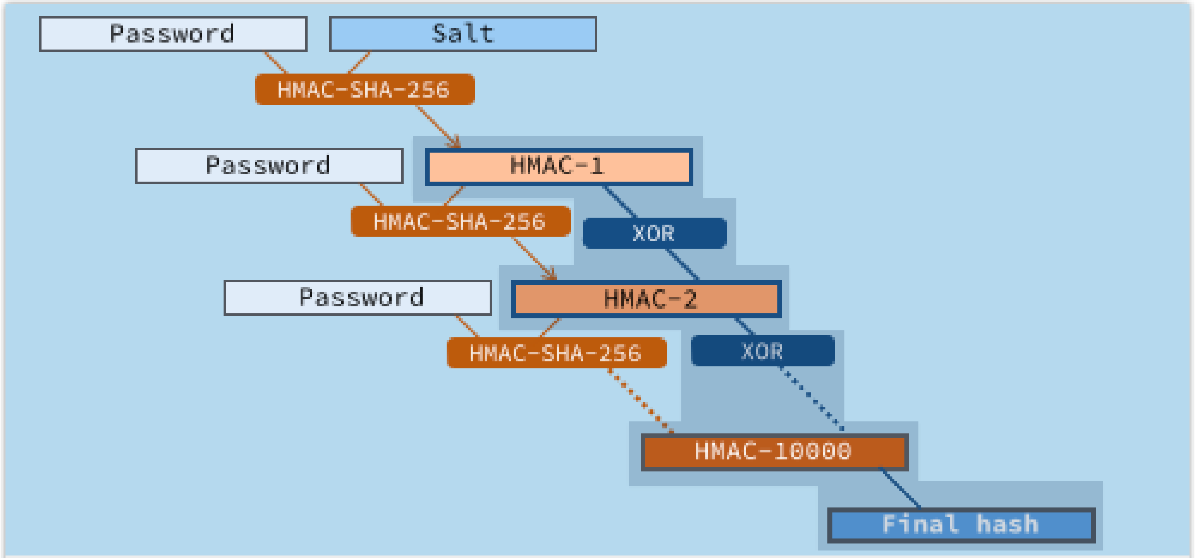
- Verify by direct match (provided_password == expected_password?)
  - Attack: compromise plain passwords on server
- $pwdTag_i$ = SHA-2(“password”)
  - Same passwords from multiple users => same resulting pwdTag
  - Attack: Large pre-computed “rainbow” tables allow for very quick check common passwords
- $pwdTag_i$ = SHA-2(“password” | salt)
  - Use of rainbow tables “prevented” by addition of random (and potentially public) *salt*
  - Attack: dictionary-based brute-force still possible
- $pwdTag_i$ = AES(“password”, 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

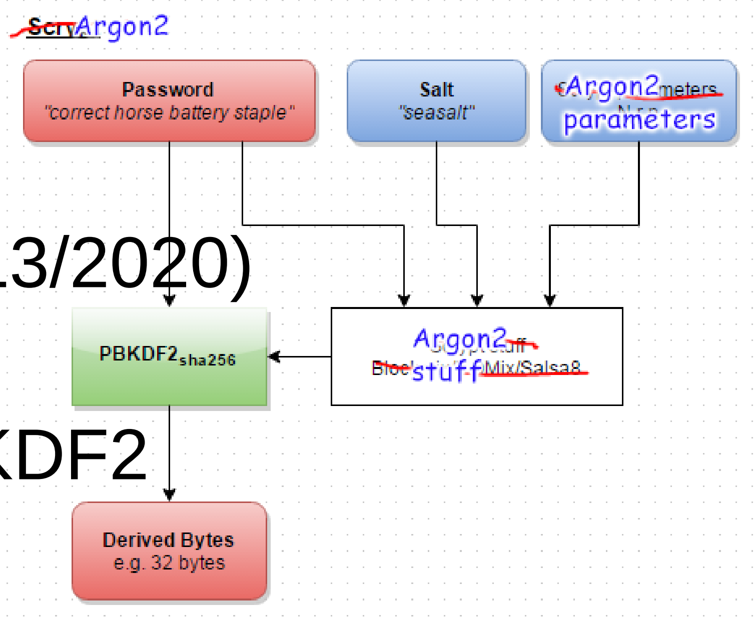


PBKDF2 using XOR to combine 10,000 successive HMAC-SHA-256 outputs into a final hash

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

Argon2 – memory hard function



- Password hashing competition (PHC) winner (2013/2020)
  – Large (configurable) memory size is required
- Memory hard functions are (slowly) replacing PBKDF2
  - Available in OpenSSL since 3.2 (09/2024)
- 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)

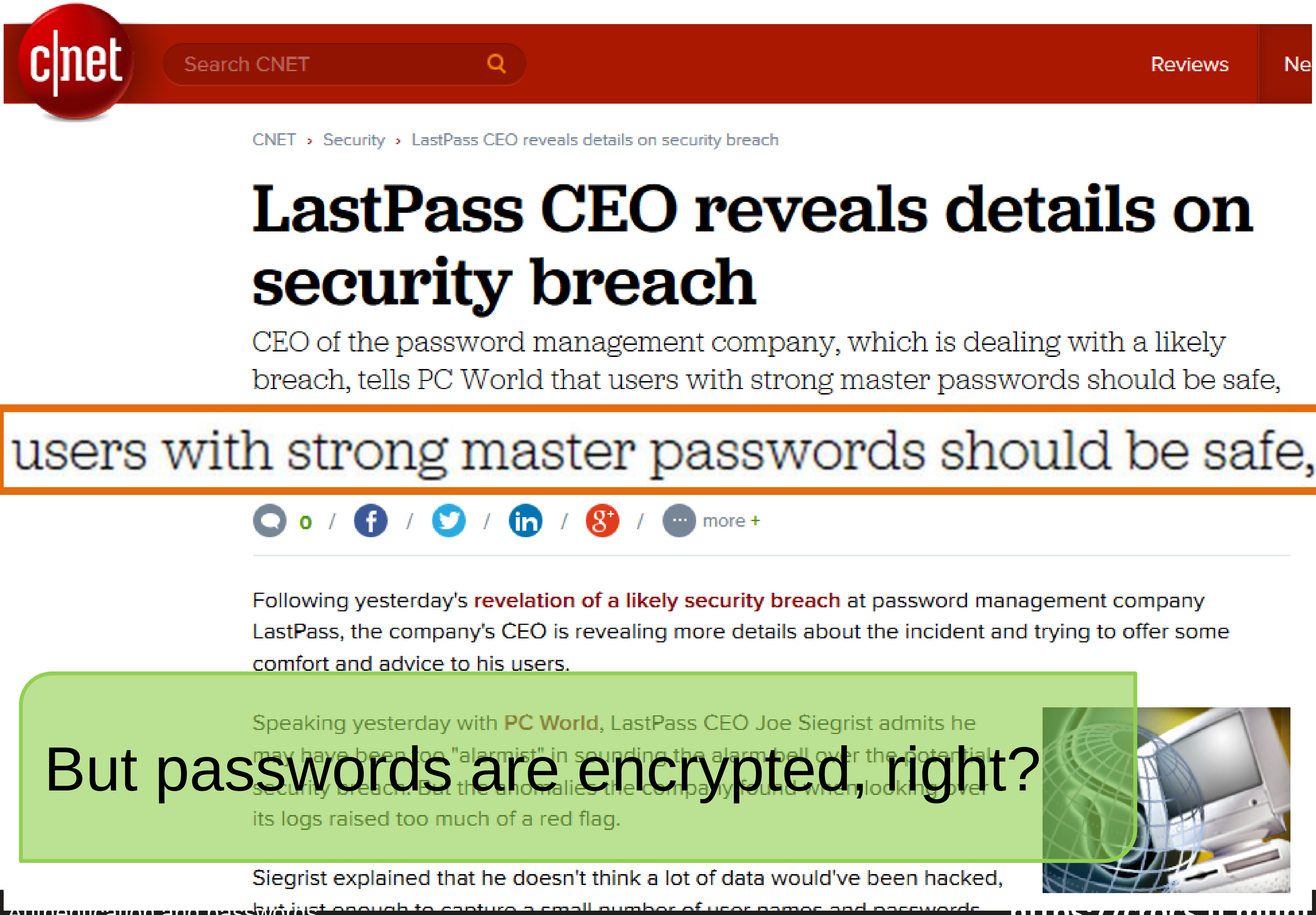
1. Human memory only
2. Write it down on paper
3. Write it into file
4. Use local password manager

Pas$w0rd
Pas$w0rd01
Google:
Sfdlk2c&
Skype:
*(&21mefd
Google:
Sfdlk2c&432mo%
Skype:
*(&21mefd872!&
Google:
Sfdlk2c&432mo%
Skype:
*(&21mefd87 !&

Remote password managers

Google:
Sfdlk2c&432mo%
Skype:
*(&21mefd872!&

KeePass+Dropbox
LastPass
1Password
MozillaSync
Firefox Lockwise
…


Search CNET

Reviews

Ne

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,

0 / / / / / more +

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 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 passwords are encrypted, right?

Devil is in the details

- Is master password strong enough? (>60% users has weak ones)
- 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?

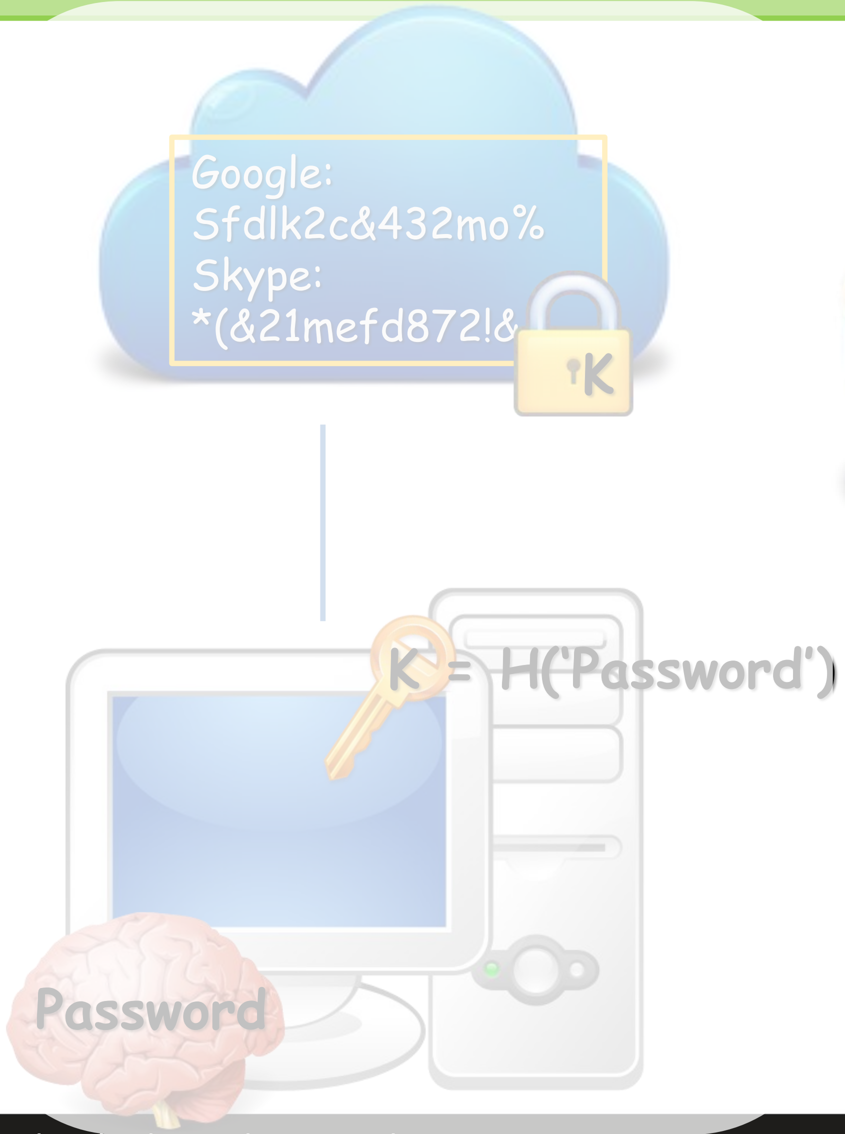
Case study

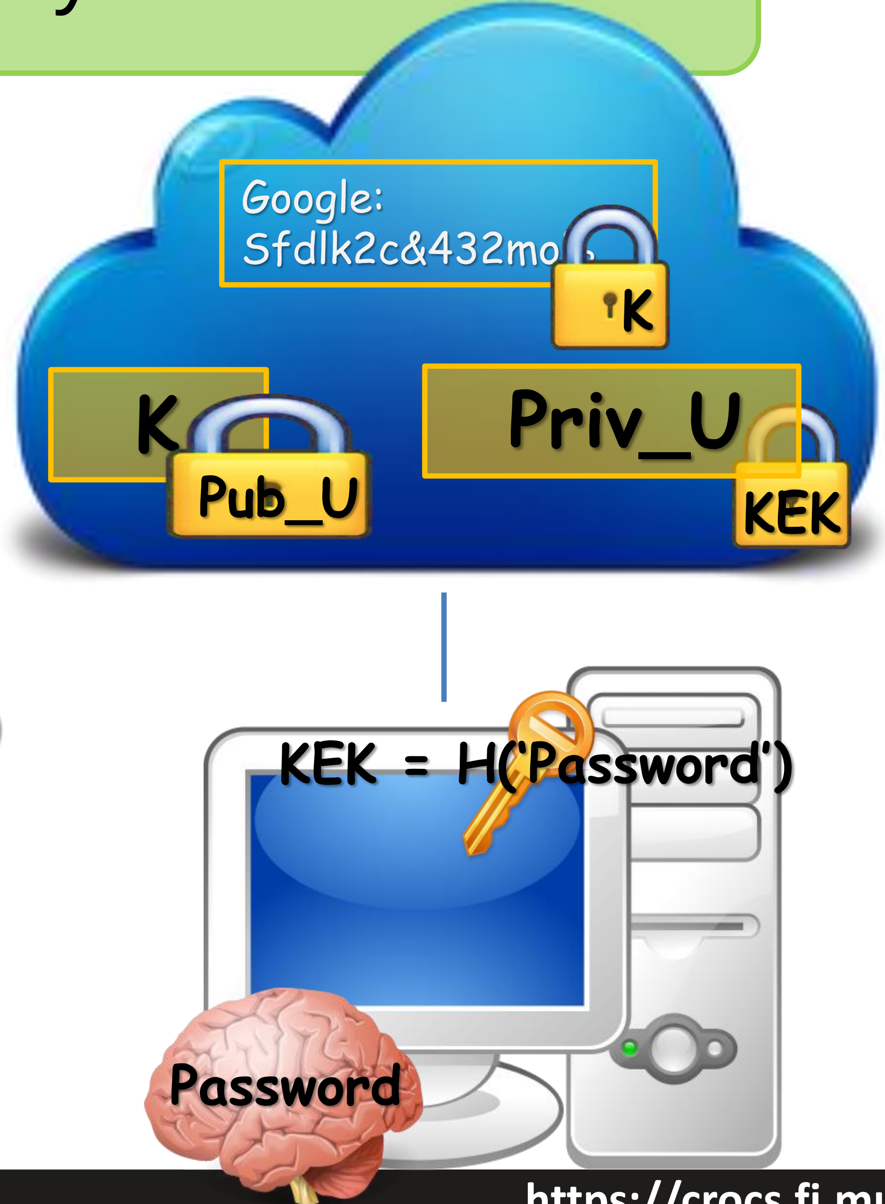
PASSWORD MANAGER FOR MULTIPLE DEVICES

Main security design principles

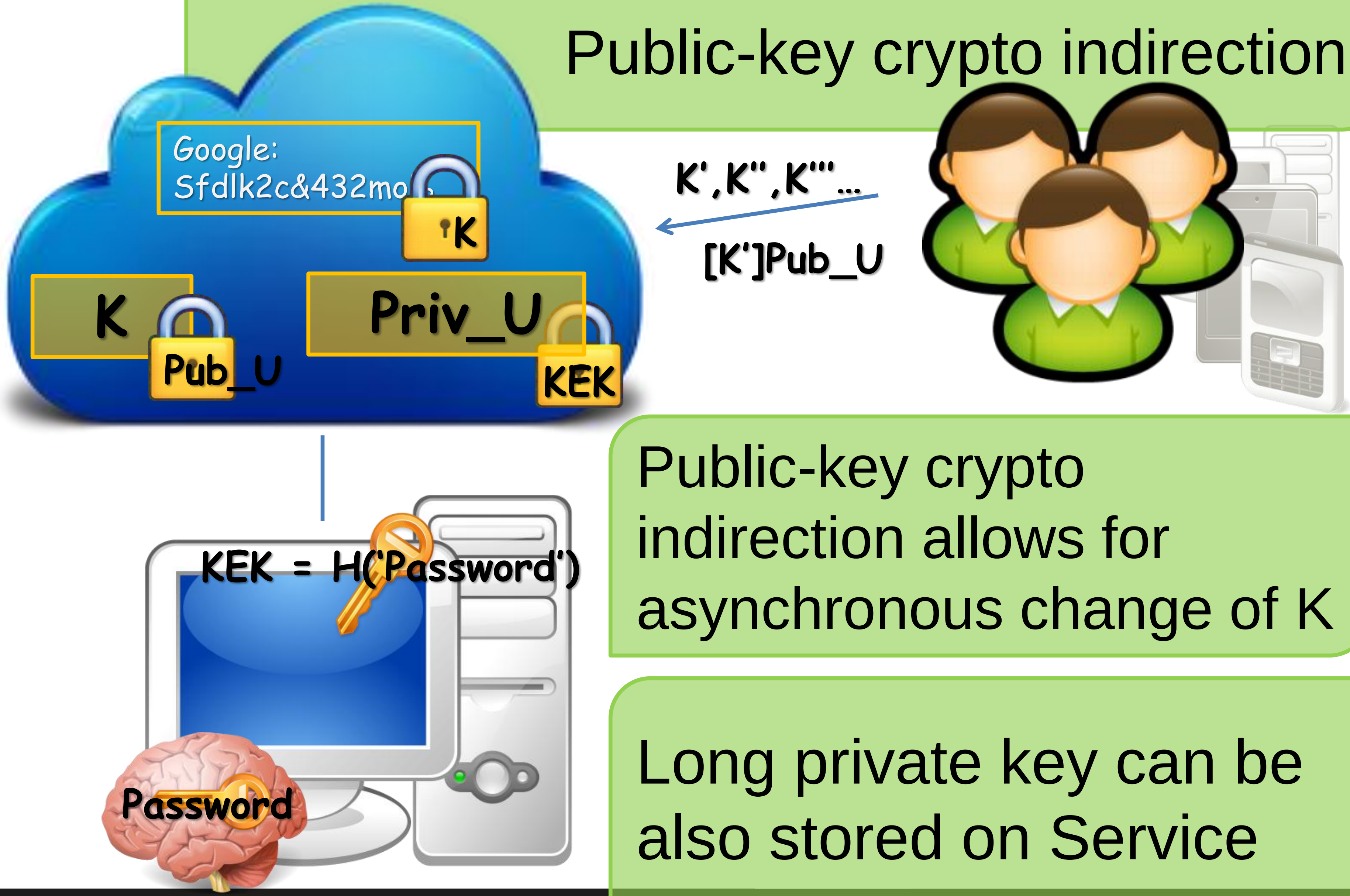
I. Treat storage service as untrusted and perform security sensitive operations on client
II. Make necessary trusted component as small as possible
III. Prevent offline brute-force, but don’t expect strong password from user
   – add entropy from other source
IV. Make transmitted sensitive values short-lived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

Public-key cryptography indirection



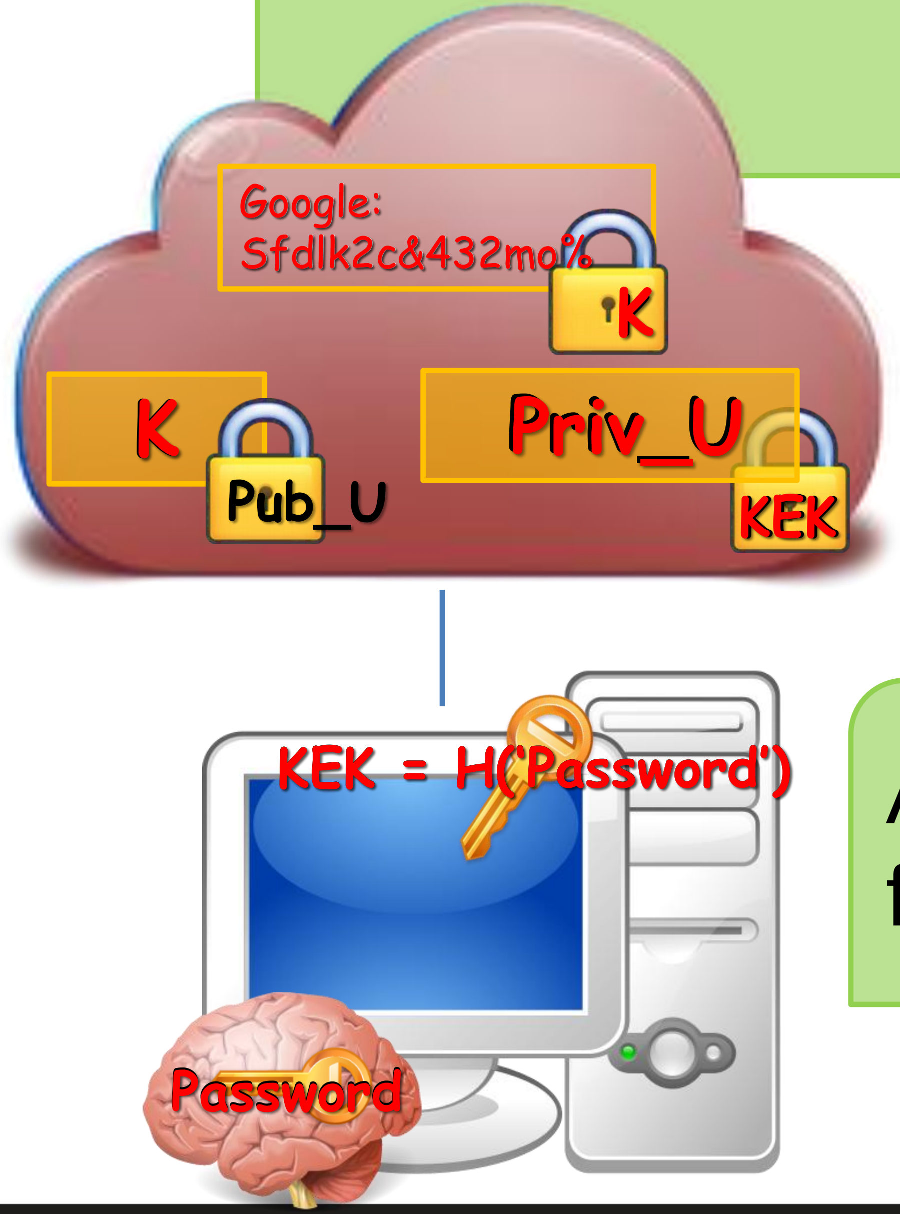


Public-key crypto indirection



Public-key crypto indirection allows for asynchronous change of K

Long private key can be also stored on Service

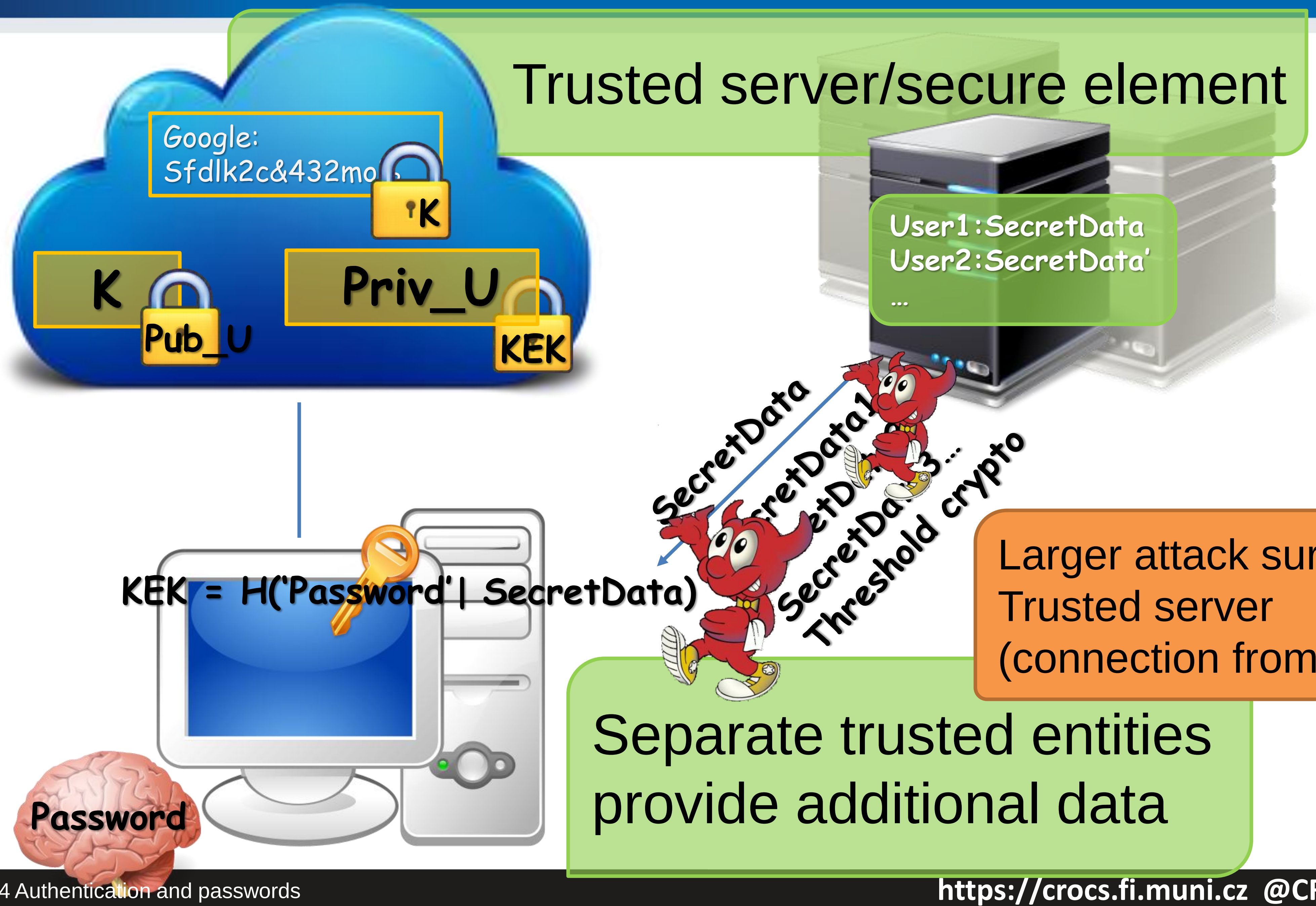


Weak password?

Users tend to have weak passwords…

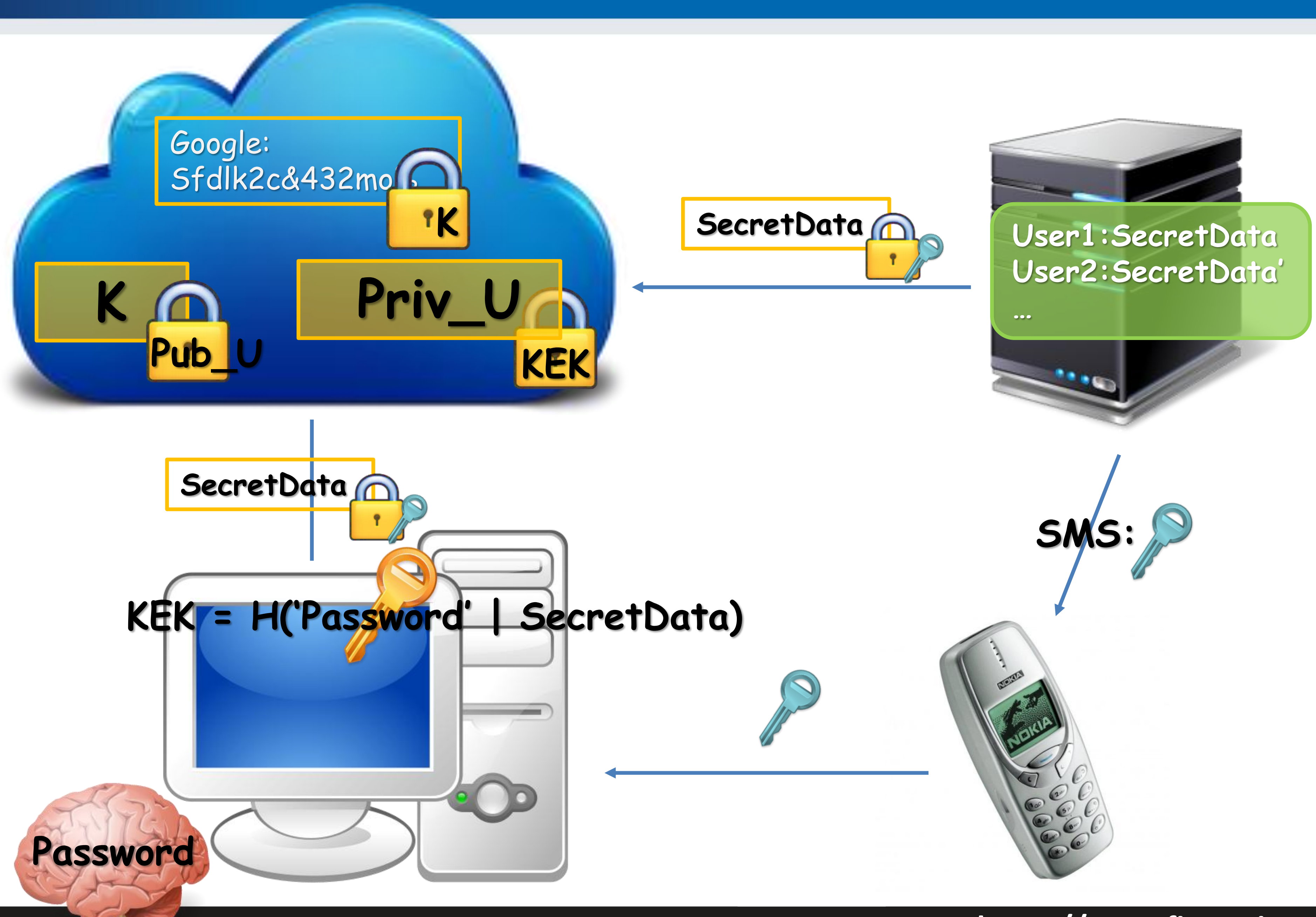
Attacker has motivation for attacking the Service!

Trusted server/secure element

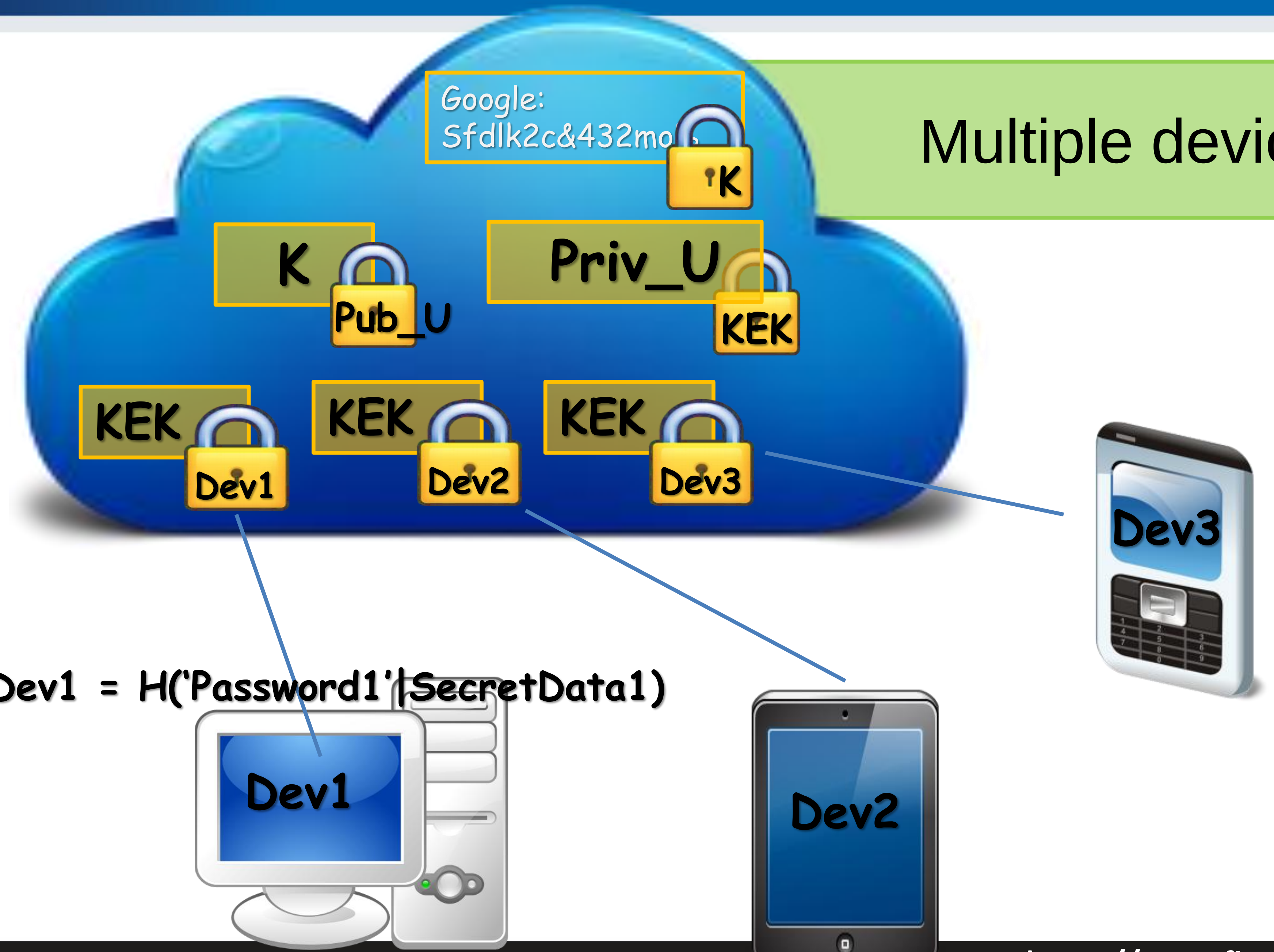


Larger attack surface on Trusted server (connection from world)

Separate trusted entities provide additional data


Google:
Sfdlk2c&432mo
K
K
Pub_U
Priv_U
KEK
SecretData
User1:SecretData
User2:SecretData'
…
SecretData
SMS:
KEK = H('Password' | SecretData)
Password

Multiple devices



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 since 2013 (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

- Basic design similar 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 (=> attacker cannot effectively bruteforce)
- 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
- Google has not publicly disclosed details for its Password Manager
  - (expect something similar, possibly more trusted)

Next week ☺

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

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

=> FIDO2, PASSKEYS

Next week ☺

What about API keys?

- “Unique codes used to authenticate requests between a client (such as an app or website) and an API on server side”

```
curl -H "Authorization: Bearer YOUR_API_KEY" https://api.example.com/data
```

- API key = long ☺, random ☺, static ☹ password
  - Practically impossible to brute-force
  - Can be compromised on client or server side (typically not changed frequently)
- Follow best practices for API keys usage:
  - Do not hardcode them in source code (use environment variables instead)
  - Restrict API key usage (operation, IP range…)
  - Rotate API key value regularly (e.g., every 60 days)
  - Use short-term API keys (similar to OTP, OAuth, JWT tokens…)

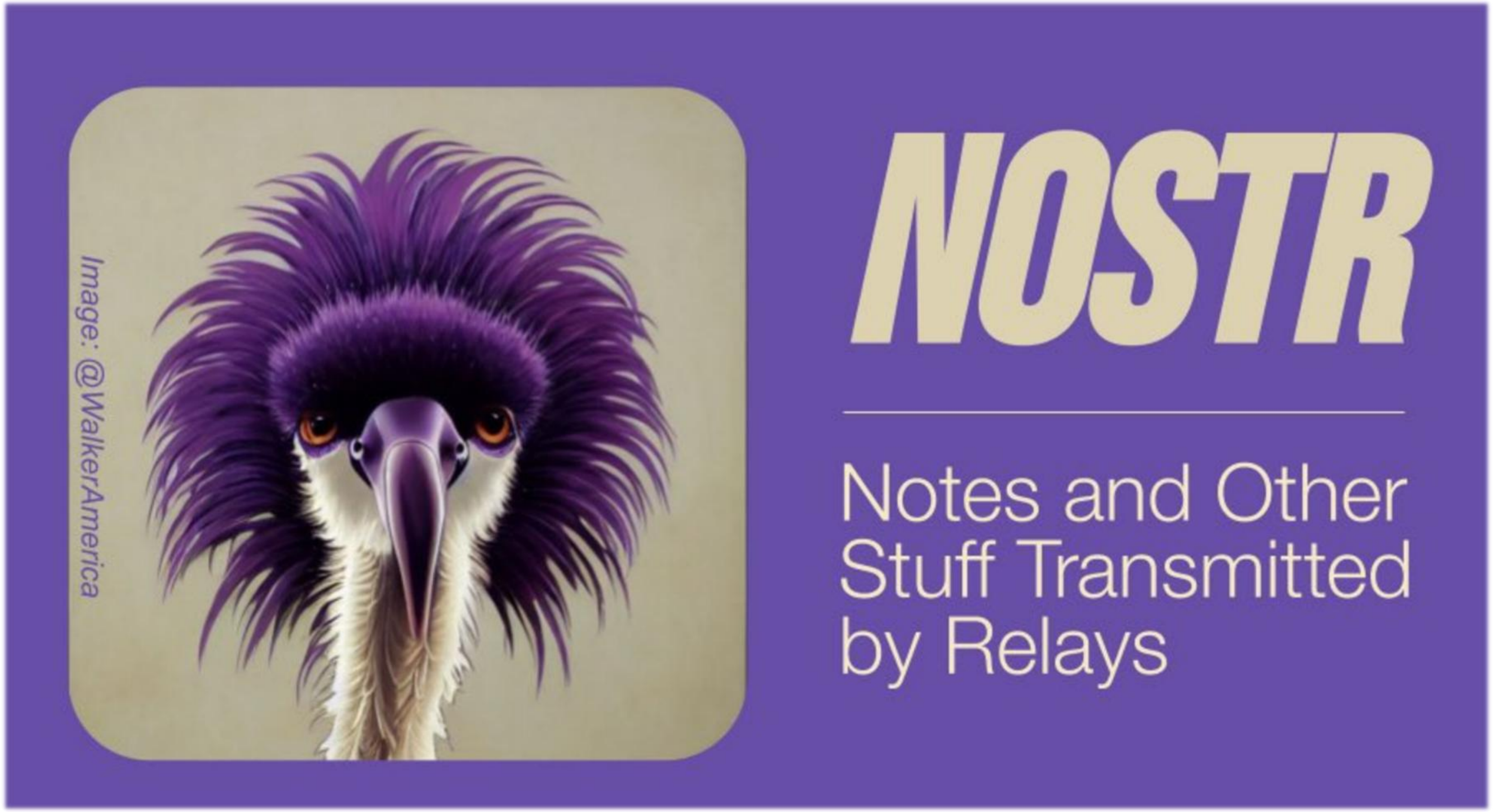
Possible password replacements

- 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

NOSTR

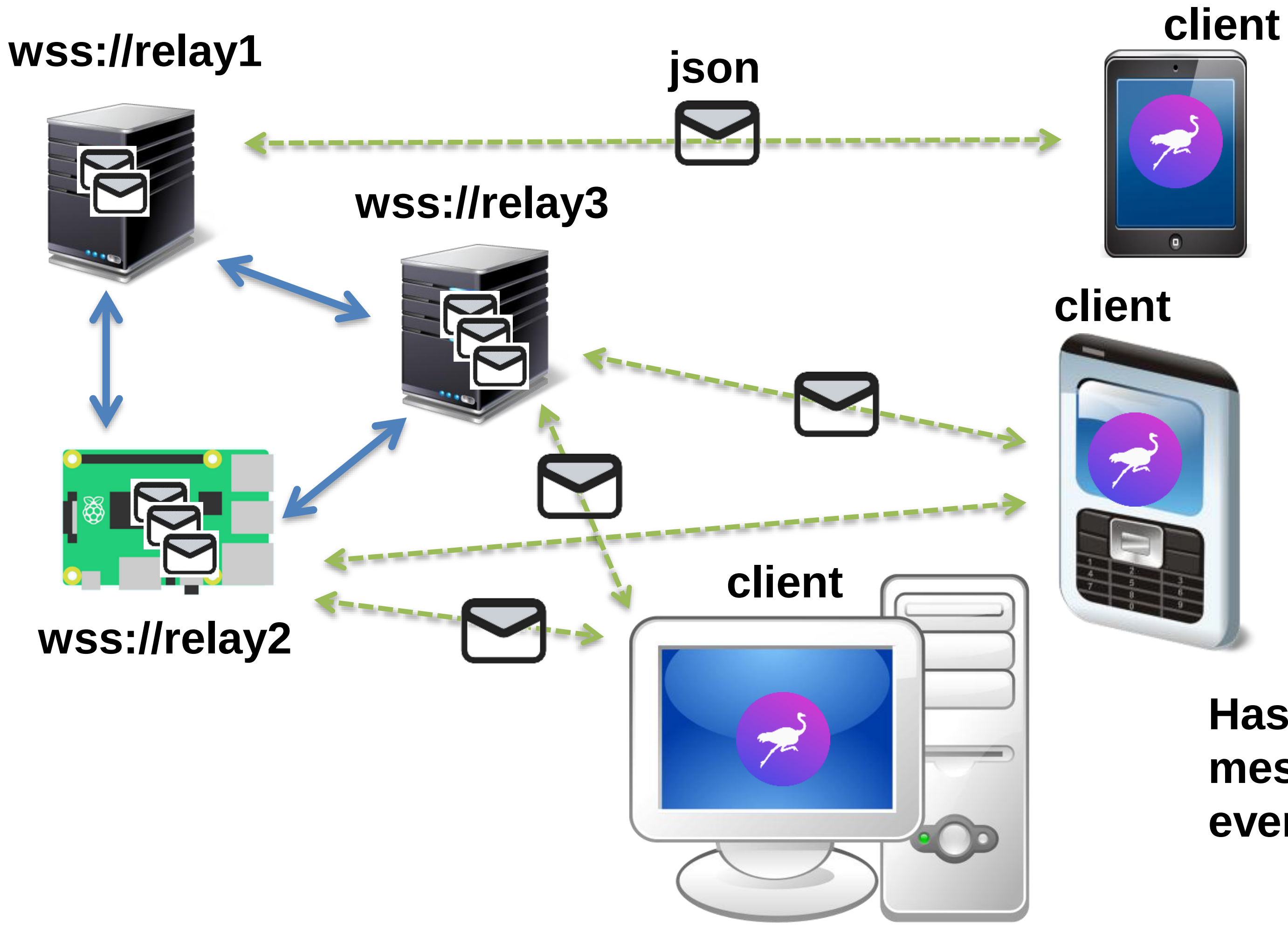


PV204 project – security application interacting with Nostr

Main problems of existing content distribution systems

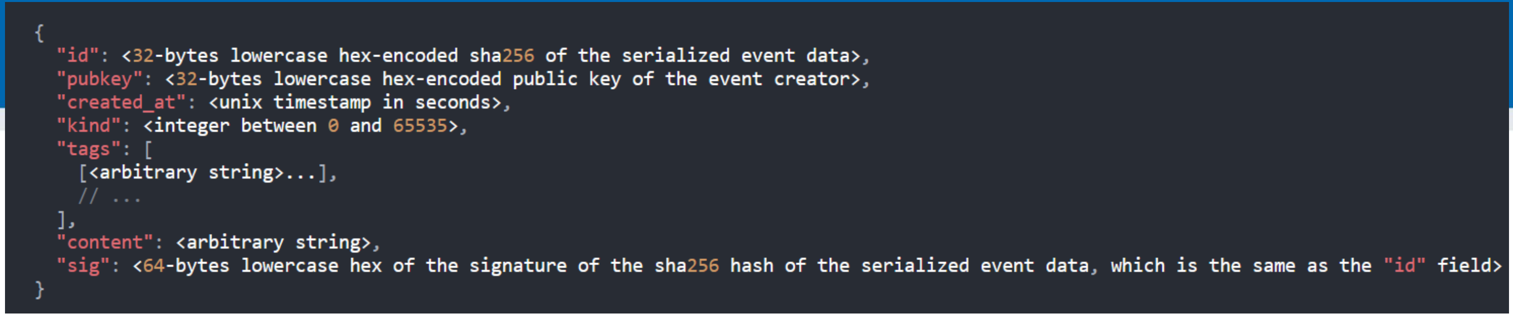
- No strong authenticity of data (only secure channel)
  - TLS ensures secure channel, confidentiality and integrity
  - Missing strong binding of data to its origin (e.g., who posted the picture)
  - Difficult to tell if message/picture/view was not modified later
- Censorship (Great Firewall of China, connectivity drop during protests, curation algorithms, de-platforming…)
  - Complicated societal question, some states/organizations censor more than others
- DNS dependency (centralization)
  - Inability to contact centralized distribution service

NOSTR basic design (relays, clients, events, JSON)



Message → `"content": "Nostr example",`
Type of msg. → `"kind": 1,`
Timestamp → `"created_at": 1736263247,`
Public key → `"pubkey": "4e51ad5b7b852a`
Signature → `"sig": "9354a3fd5628157362`
Optional tags → `"tags": [`
Hash of whole message (used as event identifier) → `"id": "0024b077529655716`

```
{
 "content": "Nostr example",
 "kind": 1,
 "created_at": 1736263247,
 "pubkey": "4e51ad5b7b852a
 "sig": "9354a3fd5628157362
 "tags": [
  [
   "url",
   "https://example.org/prod
  ],
 ]
 "id": "0024b077529655716
}
```



```
{
  "id": <32-bytes lowercase hex-encoded sha256 of the serialized event data>,
  "pubkey": <32-bytes lowercase hex-encoded public key of the event creator>,
  "created_at": <unix timestamp in seconds>,
  "kind": <integer between 0 and 65535>,
  "tags": [
    [<arbitrary string>...],
    // ...
  ],
  "content": <arbitrary string>,
  "sig": <64-bytes lowercase hex of the signature of the sha256 hash of the serialized event data, which is the same as the "id" field>
}
```

Basic NOSTR usage

1. Client A generates own keypair (ECC-256, Schnorr signature secp256k1)
2. Client A creates and signs json message (so called *Event*)
3. Client A sends *Event* to one or more relays
4. Relay(s) store *Event* into its database
5. Client B asks its relay(s) for some Event(s):
   - Events with specific keyword or hashtag
   - Events signed by specific public key (“following” that user)
   - Events by an prioritization algorithm (own algorithms possible)
   - Events encrypted for B’s public key…

- Client B’s application renders Events for its user (human/machine)
  - E.g, microblogging posts or result of payment

Main design goals of NOSTR

1. Decentralization (no single entity controls the network)
2. Censorship Resistance (difficult to completely suppress message)
3. Simplicity (low entry barrier for developers, json + signatures)
4. Interoperability (application-agnostic, different clients and services)
5. User Ownership of Data (identity based on cryptographic keys)
6. Resilience (decentralized infrastructure resilient to relay failures)
7. Privacy (identity given by keys, not real-world identifiers)
8. Extensibility (new features without breaking compatibility)

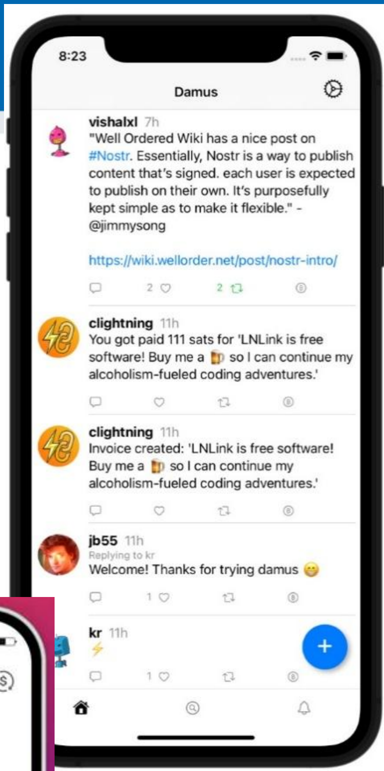
NOT design goals

- No attempt to maintain global synchronized state
  - Clients connected to different relays can see different set of messages
  - (Maintaining global synchronized state in decentralized systems is very expensive)
- No guarantee to keep messages available forever
  - Relays may clear its cache of messages or disappear
  - (User may operate own relay or simply rebroadcast messages)

NOSTR is open communication protocol (like TCP/IP), not some specific application!

Applications build atop NOSTR

- Social Media & Microblogging (like Twitter/X/Bluesky)
- Long-form content (like Medium/Substack)
- Chat & Messaging Apps (like WhatsApp/Telegram)
- Decentralized Finance & Payments
  - – Zaps, Lighting payments…
- Marketplaces & Communities
- Authentication & Login
- Web of Trust formed from "followers" of public keys
- …



vishalxl 7h
"Well Ordered Wiki has a nice post on #Nostr. Essentially, Nostr is a way to publish content that's signed. each user is expected to publish on their own. It's purposefully kept simple as to make it flexible." - @jimmysong

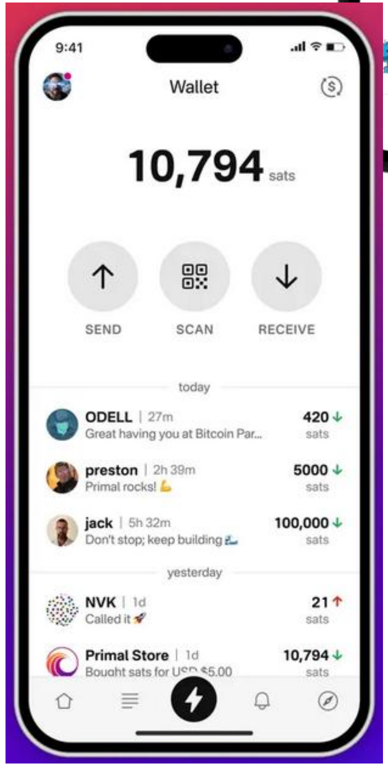
https://wiki.wellorder.net/post/nostr-intro/

clightning 11h
You got paid 111 sats for 'LNLink is free software! Buy me a 🍺 so I can continue my alcoholism-fueled coding adventures.'

clightning 11h
Invoice created: 'LNLink is free software! Buy me a 🍺 so I can continue my alcoholism-fueled coding adventures.'

jb55 11h
Replying to kr
Welcome! Thanks for trying damus 😊

kr 11h
⚡



10,794 sats

today

ODELL | 27m Great having you at Bitcoin Par...	420 ↓ sats
preston | 2h 39m Primal rocks! 💪	5000 ↓ sats
jack | 5h 32m Don't stop; keep building 🏗	100,000 ↓ sats

yesterday

NVK | 1d Called it 🚀	21 ↑ sats
Primal Store | 1d Bought sats for USD $5.00	10,794 ↓ sats

Problems of (truly) open systems

- Spam and Deny of Service attacks
  - Permissionless access allow spamming bots to easily join and send messages
  - Cheap to create new identity to circumvent blocking based on blacklists
  - Solution: PoW before posting any message (can be Relay specific)
  - Solution: Reputation system based on (pseudo) identities (whitelist)
- Incentives to run relays
  - Maintaining relay (server) has some cost (bandwidth, storage, CPU)
- Coordination of development to maintain interoperability
  - Nostr Implementation Possibilities (NIPs) https://nips.nostr.com/
- Algorithms to sort and prioritize messages (e.g., in microblogging)

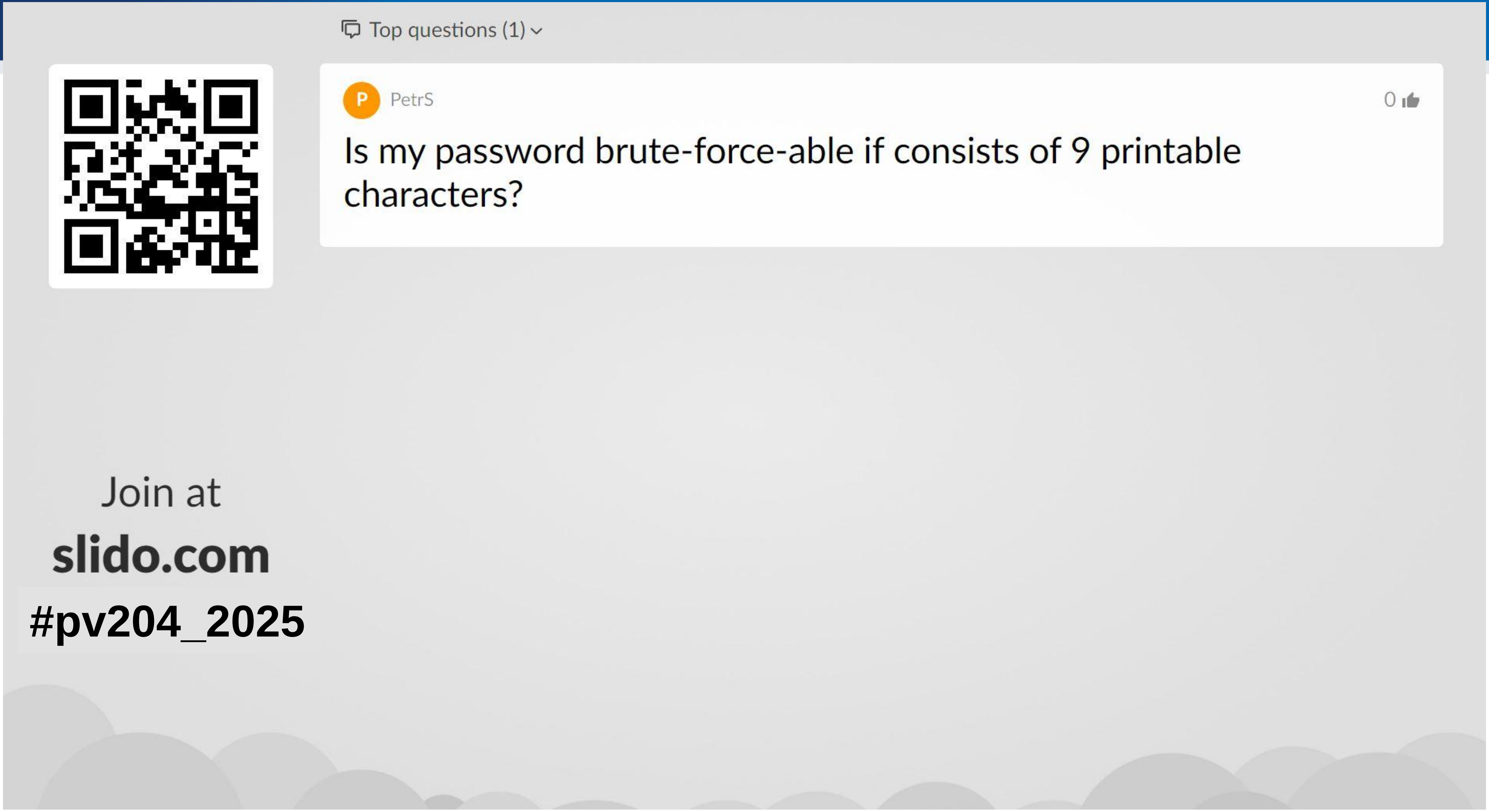
Technical resources for NOSTR

- Nostr Implementation Possibilities (NIPs)
  – https://github.com/nostr-protocol/nips
- Existing implementations
  – https://github.com/aljazceru/awesome-nostr/blob/main/README.md

- Try it!
  – Mobile apps: Damus, Primal, Amethyst…
  – Web clients: https://nostrudel.ninja/#/ -> Sign in -> Sign up -> store nsec

Summary

- Simple passwords have numerous issues, but are hard to be replaced
  - But gradually, replacement is happening (one-time passwords, tokens…)
- Major server-side breaches now very common
- Important to use passwords securely (implementation guidelines)
- Password manager with synchronization over multiple devices is not straightforward, but doable (e.g., Apple's iCloud Keychain)
- Open protocol NOSTR, data signed, keypair instead of identity
- 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_2025