How secure is your master password?

Current password managers, and a proposal for password protected secret sharing

Ulrich Haböck

Kompetenzzentrum für IT-Security, FH Campus Wien

October 13, 2018

ulrich.haboeck@fh-campuswien.ac.at
PGP-key: 48F796E247BEEDE8

- PhD in Mathematics (University Vienna)
- Post-doc TU Vienna
- since 2013 at the Competence Center for IT-Security, FH Campus Wien
- Focus: applied crypto, in particular PETs

This talk:

- The basics of brute force, the problem with password complexity
- How secure are current password managers against brute force?
- Proposal of *SpreadPass*, a password manager based on PPSS

What is NOT covered are issues of secure software design:

- secure storage of secrets (device keys, e.g.), are they kept unencrypted in memory? What about key loggers?
- implementation mistakes (badly implemented authorization, e.g.) and attacks at software level (phishing via auto fill-in, etc.)

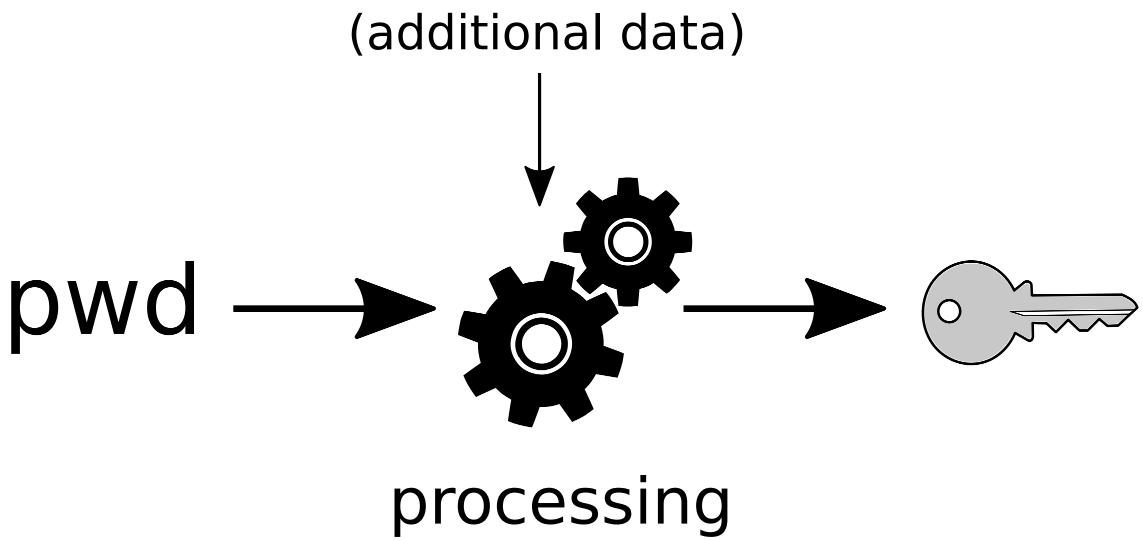
How are password vaults secured?

How are password vaults secured?

In the *password-only-model* (no 2F, token): vault secured with *Master Password pwd*:

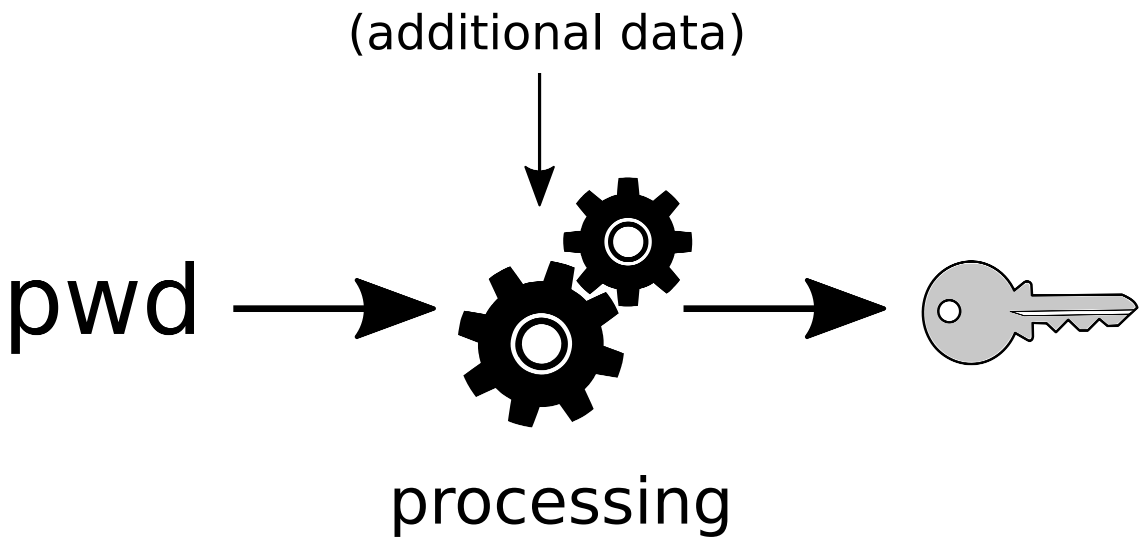
How are password vaults secured?

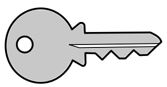
In the *password-only-model* (no 2F, token): vault secured with *Master Password pwd*:

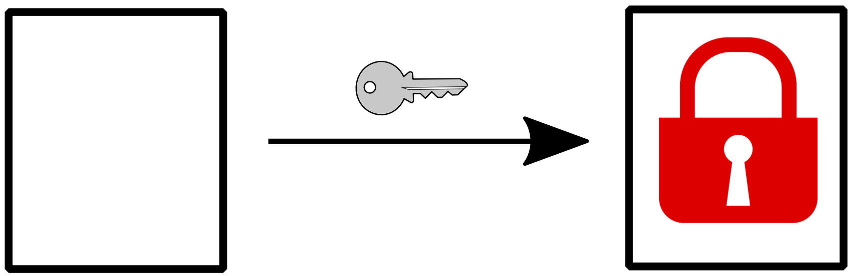


How are password vaults secured?

In the *password-only-model* (no 2F, token): vault secured with *Master Password pwd*:



 is used to encrypt your vault:

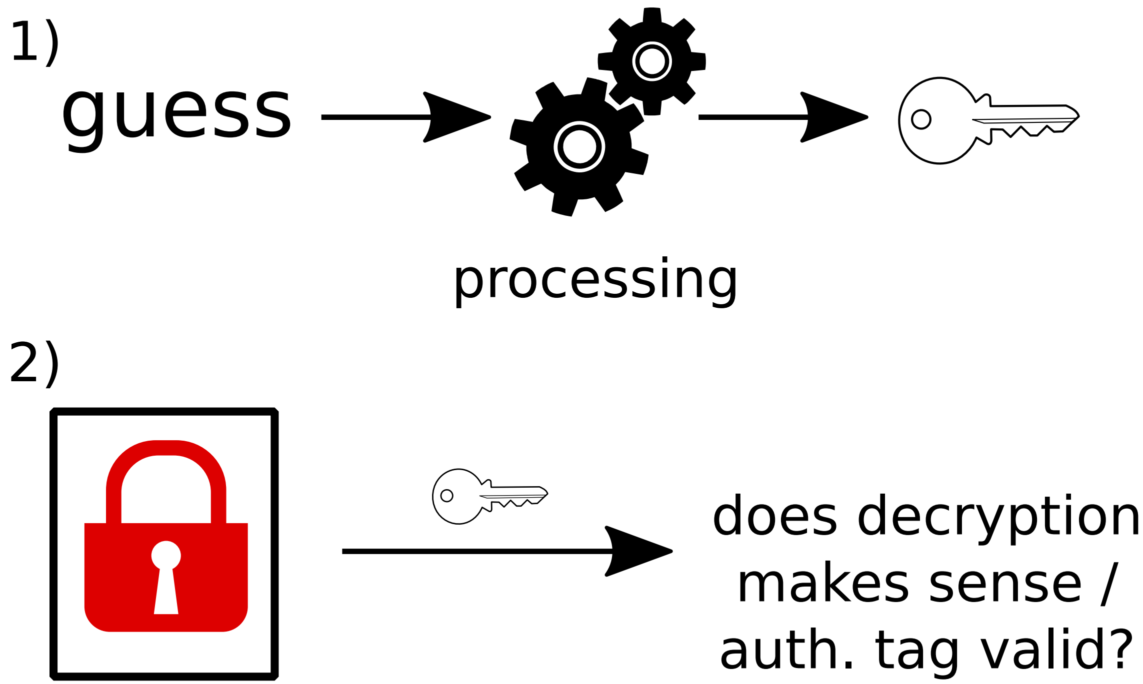


What's the problem?

Vault itself is a *point of risk*, i.e. it offers **possibility for verification** if a pwd guess is correct or not:

What's the problem?

Vault itself is a *point of risk*, i.e. it offers **possibility for verification** if a pwd guess is correct or not:



Brute force

= trying all guesses from a certain *password space*.

Brute force

= trying all guesses from a certain *password space*.

- combinatorial search space, e.g. all 12 latin character strings satisfying some regex: „*brute force*"
- search space = dictionary, i.e. file from password breaches: „*dictionary attack*"

Brute force

= trying all guesses from a certain *password space*.

- combinatorial search space, e.g. all 12 latin character strings satisfying some regex: „*brute force*”
- search space = dictionary, i.e. file from password breaches: „*dictionary attack*”

Of course, there are combinations / variants

Brute Force

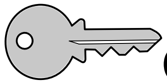
Password complexity

Password managers

SpreadPass Project

Brute force

Which search space sizes are doable? Depends on

- computational power
- pwd $\rightarrow$  derivation (how much time/space consuming)

Brute force

For example, suppose you can calculate 2^{75} Hashes in 10 Minutes (bitcoin mining network) and that your key is a **1,000,000** times iterated hash of *pwd*.

Brute force

For example, suppose you can calculate 2^{75} Hashes in 10 Minutes (bitcoin mining network) and that your key is a **1,000,000** times iterated hash of *pwd*.

- That are

$$\approx \frac{2^{75}}{2^{20}} = 2^{55} \text{ guesses in 10 min.}$$

Brute force

For example, suppose you can calculate 2^{75} Hashes in 10 Minutes (bitcoin mining network) and that your key is a **1,000,000** times iterated hash of *pwd*.

- That are

$$\approx \frac{2^{75}}{2^{20}} = 2^{55} \text{ guesses in 10 min.}$$

- Covers a search space of all pwd's with

$$^{55}/_{6} > 9 \text{ base64 chars.}$$

Brute force

Reference: 2^{75} Hashes in 10 Minutes, **300,000** times iterated hash

pwd len	chars [a-z,A-Z,0-9]	+10	+ 66
8	1.4 s	3.4 s	5.7 min
10	1.11 h	4.9 h	65 d
12	178 d	2.9 y	*
14	*	*	*

* means > 100 years

Brute force

Reference: 2^{75} Hashes in 10 Minutes, **5,000** times iterated hash

pwd len	chars: [a-z,A-Z,0-9]	+10	+ 66
8	< 1 s	< 1 s	5.7 s
10	1.11 min	4.9 min	1 d
12	2.9 h	17.8 h	48 y
14	31 y	*	*

* means > 100 years

How secure is a password?

How secure is a password?

Hard to estimate.

How secure is a password?

Hard to estimate.

1. Depends on how you choose. If you sample base64 chars uniformly, 17 chars = 102 bit, 21 chars = 128 bit search space
   Problem: human choice != uniform

How secure is a password?

Hard to estimate.

1. Depends on how you choose. If you sample base64 chars uniformly, 17 chars = 102 bit, 21 chars = 128 bit search space
   Problem: human choice != uniform
2. Depends on what the attackers know about the way you choose: wordlists, weighted or Markov models, etc. reduce search space a lot!

How secure is a password?

Forget (commonly used) entropy notion for passwords.

1. Is based on unrealistic model assumptions.

How secure is a password?

Forget (commonly used) entropy notion for passwords.

1. Is based on unrealistic model assumptions.
2. in particular: it does not take into account apriori knowledge of the attacker.

How secure is a password?

For example: consider Silvie, who supposedly chooses her password according to the following rule:

- she takes her favorite Elvis song ($\approx$ 784 songs),

How secure is a password?

For example: consider Silvie, who supposedly chooses her password according to the following rule:

- she takes her favorite Elvis song ($\approx$ 784 songs),
- takes only the first letter of a passage of $8 - 12$ consecutive words (there are $<$ 200 words per song, thus max. 200 passages).

How secure is a password?

For example: consider Silvie, who supposedly chooses her password according to the following rule:

- she takes her favorite Elvis song ($\approx$ 784 songs),
- takes only the first letter of a passage of $8-12$ consecutive words (there are < 200 words per song, thus max. 200 passages).

How secure is a password?

For example: consider Silvie, who supposedly chooses her password according to the following rule:

- she takes her favorite Elvis song ($\approx$ 784 songs),
- takes only the first letter of a passage of $8 - 12$ consecutive words (there are $<$ 200 words per song, thus max. 200 passages).

That is only a **19.6 bit search space**!

How secure is a password?

- If Silvie randomly (optimistic!) upper/lower cases the letters: max. 2^{12} combinations per passage, i.e. **31.6 bit search space**.

How secure is a password?

- If Silvie randomly (optimistic!) upper/lower cases the letters: max. 2^{12} combinations per passage, i.e. **31.6 bit search space**.
- if she inserts two numbers at random (!) position between the letters: max. $11 \cdot 10 = 55$ position choices, $5,500$ combinations, i.e. **42 bit search space**.

How secure is a password?

"Average human password $\approx$ 40 bit search space"

citation...

Fazit

- Always hard to measure the security of usable = humanly rememberable pwd's

Fazit

- Always hard to measure the security of usable = humanly rememberable pwd's
- Even when you choose pwd in a smart way: psychology can reduce search space significantly!

Password managers

How secure are current password managers against brute force?

Password managers

How secure are current password managers against brute force?

- Attacks on pwd "no problem", when password vault is only locally. . . if you can trust your device.

Password managers

How secure are current password managers against brute force?

- Attacks on pwd "no problem", when password vault is only locally. . . if you can trust your device.
- But what about **device loss**?

Password managers

How secure are current password managers against brute force?

- Attacks on pwd "no problem", when password vault is only locally. . . if you can trust your device.
- But what about **device loss**?
- Even more delicate: what about cloud-based **password syncing**?

Google sync

Google sync

Chrome/Chromium browser:

- pwds locally secured with the system pwd,

Google sync

Chrome/Chromium browser:

- pwds locally secured with the system pwd,
- synced passwords per default **plaintext** for Google, opt in to encrypt synced passwords with separate pwd

Google sync

Chrome/Chromium browser:

- pwds locally secured with the system pwd,
- synced passwords per default **plaintext** for Google, opt in to encrypt synced passwords with separate pwd
- at least Chromium: key derviation **only** PBKDF-HMAC-SHA1 with **1,003** iterations, **fixed salt**(!)

Google sync

Reference:
https://bugs.chromium.org/p/chromium/issues/detail?id=820976
(Mar 12 2018)

Security: Chrome Sync passphrase is far too easy to bruteforce

Reported by wladi...@palant.de, Mar 12 2018

Ba

```
VULNERABILITY DETAILS
Documentation https://support.google.com/chrome/answer/165139#passphrase states the following:

> With a passphrase, you can use Google's cloud to store and sync your Chrome data without letting Google read it.

That's not currently true. The encryption key for the data is derived from the passphrase using PBKDF2-HMAC-SHA1 using a fixed s
and merely 1003 iterations. This doesn't provide significant protection against bruteforcing the passphrase. Worse yet, if somebo
manages to access encrypted data from multiple user accounts, the fixed salt allows bruteforcing all of them at the same time.

The sequence in code is the following:

* SyncEncryptionHandlerImpl::SetCustomPassphrase(passphrase), https://cs.chromium.org/chromium/src/components/sync/engine_impl/s
ncryption_handler_impl.cc?l=1110&rcl=bddb211b216ed6844cb64a7ec51b069e0ac044b5
* Cryptographer::AddKey({"localhost", "dummy", passphrase}), https://cs.chromium.org/chromium/src/components/sync/base/cryptogra
cc?l=160&rcl=bddb211b216ed6844cb64a7ec51b069e0ac044b5
* Nigori::InitByDerivation("localhost", "dummy", passphrase), https://cs.chromium.org/chromium/src/components/sync/base/nigori.c
5&rcl=bddb211b216ed6844cb64a7ec51b069e0ac044b5
```

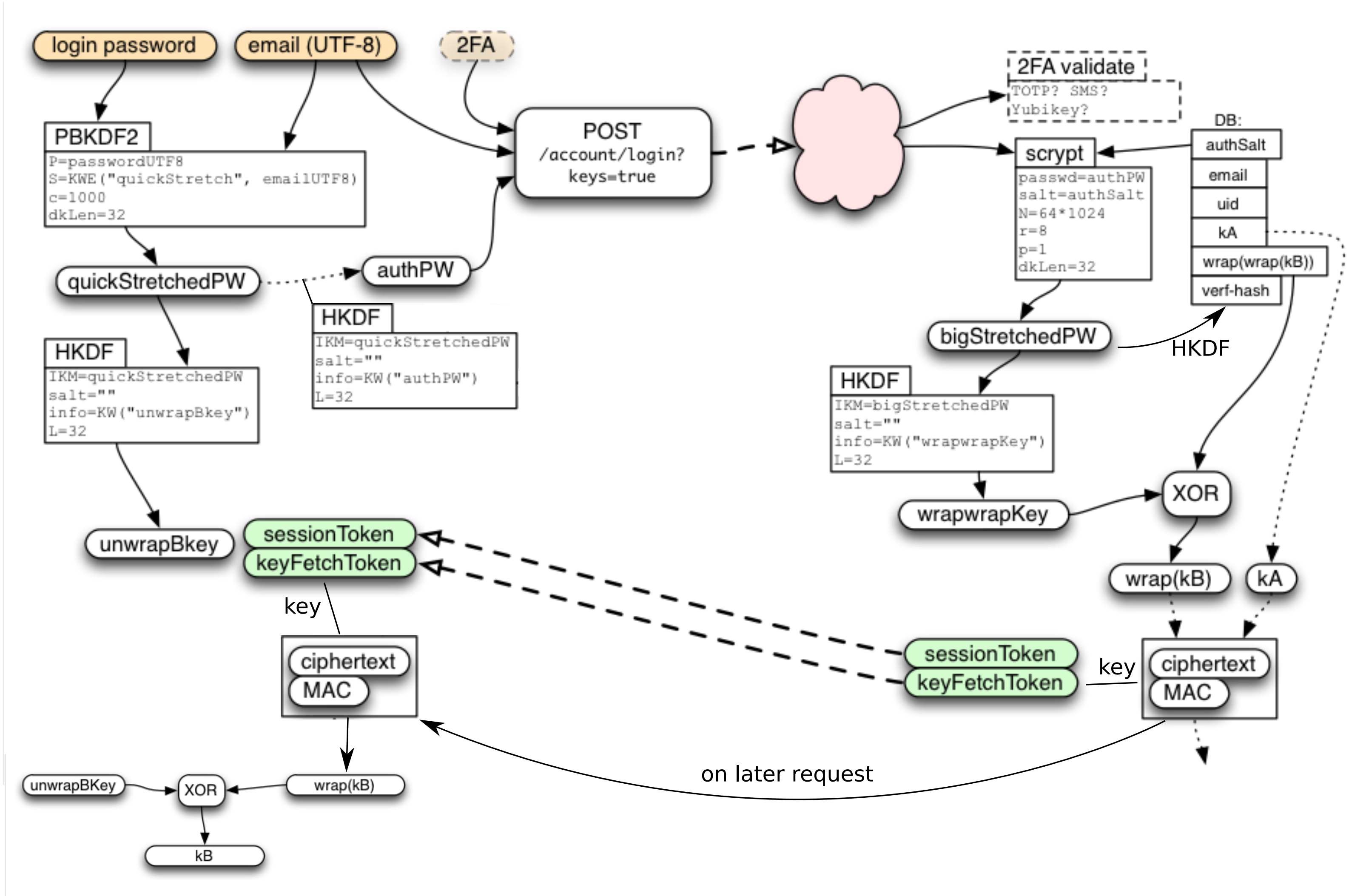
Firefox sync

Firefox onepw protocol[1]:

[1] https://github.com/mozilla/fxa-auth-server/wiki/onepw-protocol

Firefox sync

Firefox onepw protocol[1]:



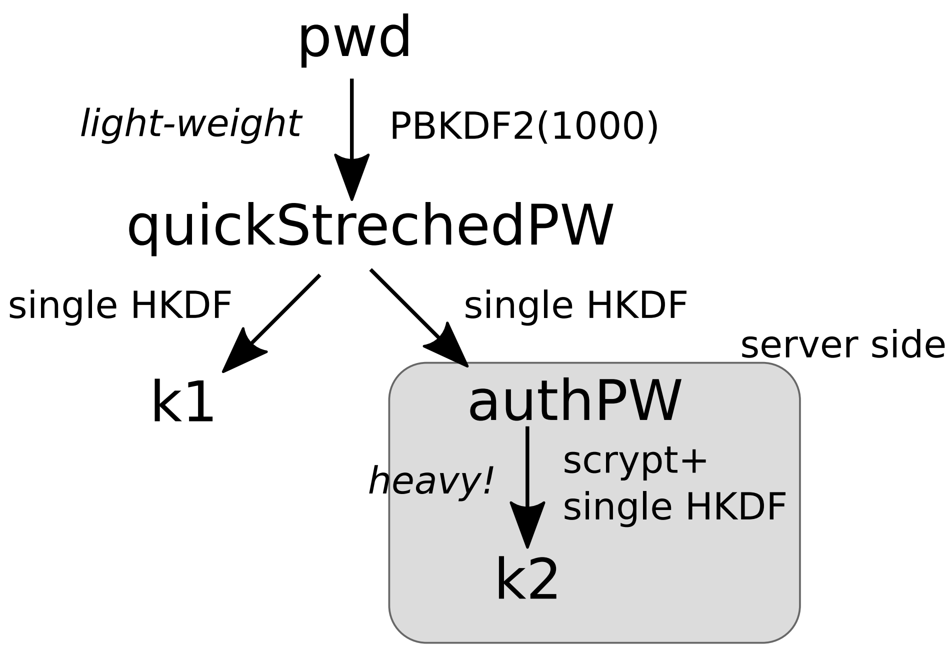
[1] https://github.com/mozilla/fxa-auth-server/wiki/onepw-protocol

Firefox sync

Balanced key derivation:

$$kB = D_{k_1}\left(D_{k_2}(\text{server stored ciphertext})\right)$$

with



Firefox sync

How secure is your pwd:

Firefox sync

How secure is your pwd:

- If your pwd is properly chosen, secure against server breaches (hard to brute force scrypt)

Firefox sync

How secure is your pwd:

- If your pwd is properly chosen, secure against server breaches (hard to brute force scrypt)

Firefox sync

How secure is your pwd:

- If your pwd is properly chosen, secure against server breaches (hard to brute force scrypt)

BUT

- At each login, server learns a light-weight derivative of pwd. (PBKDF(1000)+single HKDF).

Firefox sync

How secure is your pwd:

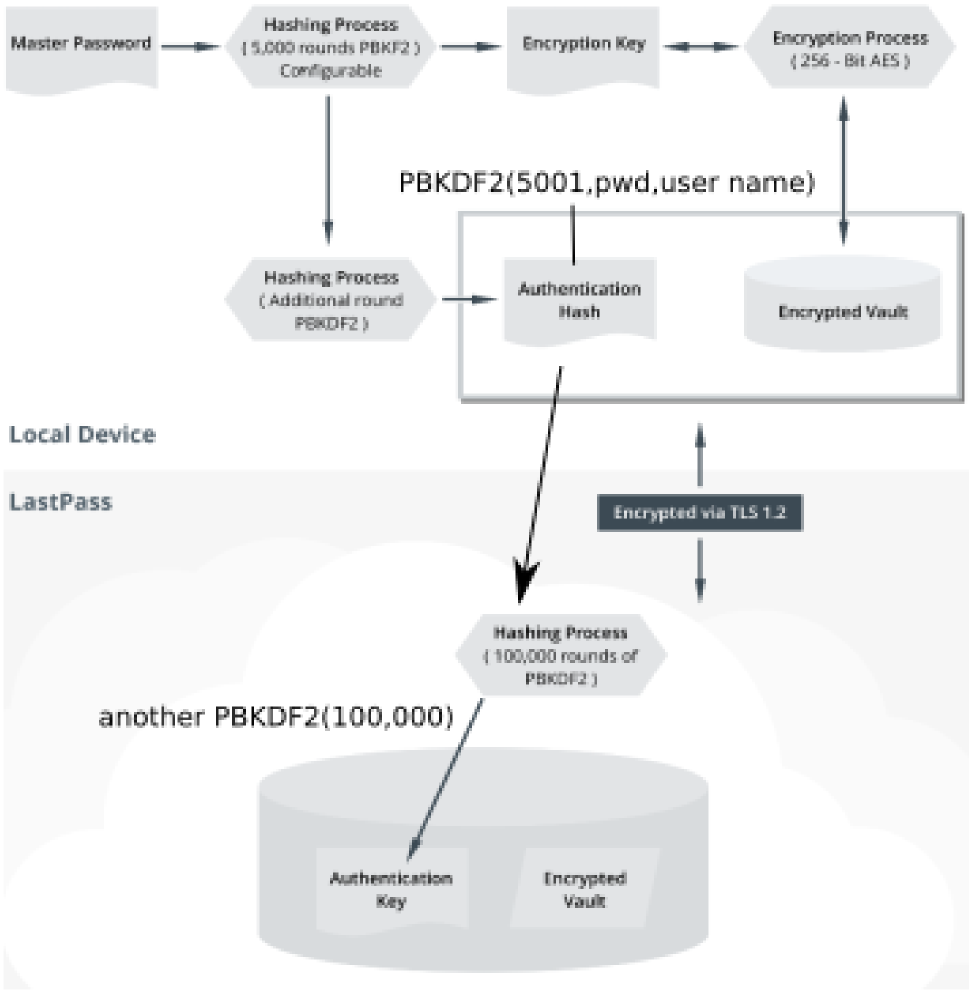
- If your pwd is properly chosen, secure against server breaches (hard to brute force scrypt)

BUT

- At each login, server learns a light-weight derivative of pwd. (PBKDF(1000)+single HKDF).
- All salts are known to the server, thus brute force on pwd doable! No better than Chromium!

Lastpass

As in Firefox sync, balanced key derivation[2]:



[2] LastPass technical white paper

Lastpass

Security of your pwd:

- Server part of key derivation is not very heavy (PBKDF2(100,000)), hard to estimate security against server breaches.
- As for Firefox and Google: server learns a light-weight derivative of pwd (PBDKF2(5,001)).

Lastpass

Security of your pwd:

- Server part of key derivation is not very heavy (PBKDF2(100,000)), hard to estimate security against server breaches.
- As for Firefox and Google: server learns a light-weight derivative of pwd (PBDKF2(5,001)).

Pro:

- user can adjust number of client side iterations (why not more iterations by default?)
- supports many 2F token

Keepass

Keepass

Per-se client only solution, not much to say:

- KeePass 1.x and 2.x: AES256-KDF $6,000$ iterations ($\approx$ PBKDF2(6,000))
- KeePass 2.x only: Argon2 with reasonable parameters

Keepass

Per-se client only solution, not much to say:

- KeePass 1.x and 2.x: AES256-KDF $6,000$ iterations ($\approx$ PBKDF2(6,000))
- KeePass 2.x only: Argon2 with reasonable parameters

Pro:

- optional key file as 2F.

BUT

- If pwd only: be careful when syncing KeePass vault via cloud!

Dashlane

[3] https://www.dashlane.com/download/Dashlane_SecurityWhitePaper_July2018_rev2.pdf

Dashlane

Security white paper[3] is misleading:

1. General Security Principles

a. Protection of User Data in Dashlane

Protection of user data in Dashlane relies on 4 separate secrets:

- The **User Master Password**
  - It is never stored on Dashlane servers, nor are any of its derivatives (including hashes)
  - By default, it is not stored locally on disk on any of the user's devices; we simply use it to (de)cipher the local files containing the user data
  - It is stored locally upon user request when enabling the feature "Remember my Master Password"
  - In addition, the user's Master Password never transmits over the internet, nor do any of its derivatives (including hashes)

[3] https://www.dashlane.com/download/Dashlane_SecurityWhitePaper_July2018_rev2.pdf

As a matter of fact,

- Data at rest, locally or synced, is (encrypt + MAC)ed with pwd, thus *any synced data's MAC gives a derivative of the master password*

As a matter of fact,

- Data at rest, locally or synced, is (encrypt + MAC)ed with pwd, thus *any synced data's MAC gives a derivative of the master password*
- KDF is Argon2d (iterations=3, memory=32M, parallelisation=2), client side only. Better than KeePass, but still beware of weak pwd!

iOS Keychain Sync

iOS Keychain Sync

Uses (hybrid) asymmetric encryption:

- Each device D has it's own key pair $(\mathrm{pk}_D, \mathrm{sk}_D)$, private key is never sent to the cloud.

iOS Keychain Sync

Uses (hybrid) asymmetric encryption:

- Each device D has it's own key pair $(\mathrm{pk}_D, \mathrm{sk}_D)$, private key is never sent to the cloud.
- new device's public key is added to "singed syncing circle" via a double signed (with device secret key and iCloud account pwd) request.

iOS Keychain Sync

Uses (hybrid) asymmetric encryption:

- Each device D has it's own key pair $(\mathrm{pk}_D, \mathrm{sk}_D)$, private key is never sent to the cloud.
- new device's public key is added to "singed syncing circle" via a double signed (with device secret key and iCloud account pwd) request.
- Password vault's encryption key is encrypted with all the public keys from the signed syncing circle.

iOS Keychain Sync

Security:

- iCloud pwd is known to the server, but

iOS Keychain Sync

Security:

- iCloud pwd is known to the server, but
- all passwords are key-encrypted, not pwd-encrypted: there is no (feasible) way to decrypt without the device key...

iOS Keychain Sync

Security:

- iCloud pwd is known to the server, but
- all passwords are key-encrypted, not pwd-encrypted: there is no (feasible) way to decrypt without the device key...

iOS Keychain Sync

Security:

- iCloud pwd is known to the server, but
- all passwords are key-encrypted, not pwd-encrypted: there is no (feasible) way to decrypt without the device key...

...unless...

...unless you are incautiously using *iOS Keychain recovery* to backup your vault encryption key k.

iOS Keychain recovery

Per default vault key k is secured by a 4–6 digit code (iOS security code or device code) and stored on an HSM provided by Apple.

iOS Keychain recovery

Per default vault key k is secured by a 4–6 digit code (iOS security code or device code) and stored on an HSM provided by Apple.

- client encrypts k with the weak 4-6 digit code AND the public key of the HSM (key escrow record)

iOS Keychain recovery

Per default vault key k is secured by a 4–6 digit code (iOS security code or device code) and stored on an HSM provided by Apple.

- client encrypts k with the weak 4-6 digit code AND the public key of the HSM (key escrow record)
- to obtain the code-encrypted key, the client performs a PAKE protocol (SRP) with the HSM to prove knowledge of the six digit code.

iOS Keychain recovery

Security:

- Thx to PAKE, no (feasible) derivation of the 4-6 digit code can be captured between Client and HSM: no way for an eavesdropper to offline-guess the weak code!

iOS Keychain recovery

Security:

- Thx to PAKE, no (feasible) derivation of the 4-6 digit code can be captured between Client and HSM: no way for an eavesdropper to offline-guess the weak code!

iOS Keychain recovery

Security:

- Thx to PAKE, no (feasible) derivation of the 4-6 digit code can be captured between Client and HSM: no way for an eavesdropper to offline-guess the weak code!

BUT

- Usability is put over security: you need to trust that the software of Apples HSM is not malicious.

iOS Keychain recovery

Pro:

- alternative to 4-6 digit passcode: separate longer code, user-defined or device generated random code.

Again, be careful when using user-defined code: you have to trust Apple that it does not perform brute force on that code.

1Password

As iOS Keychain, uses asymm. crypto (hybrid encryption). A user has a

- $(\mathrm{pk}_U, \mathrm{sk}_U)$, a secret account key ($\approx$128 Bit) and his pwd.

1Password

As iOS Keychain, uses asymm. crypto (hybrid encryption). A user has a

- $(\mathrm{pk}_U, \mathrm{sk}_U)$, a secret account key ($\approx$128 Bit) and his pwd.

1Password

As iOS Keychain, uses asymm. crypto (hybrid encryption). A user has a

- $(\mathrm{pk}_U, \mathrm{sk}_U)$, a secret account key ($\approx$128 Bit) and his pwd.

Public keys are used for password sharing.

1Password

- pwd and account key are used to encrypt sk_U: *master unlock key (MUK)* is derived via PBKDF2(100,000) + single HKDF
  Note: MUK-encrypted sk_U is sent to server.
- similar derivation (different salt) of the secret used for authentication.
- — linking new devices: account key transported out of band (QR, e.g.) for authentication, MUK-encrypted sk_U is fetched from server.

1Password

Security of *pwd*:

- server has no information to brute force the pwd: the user's secret account key is kept from server.

1Password

Security of *pwd*:

- server has no information to brute force the pwd: the user's secret account key is kept from server.
- key back up is per default secure

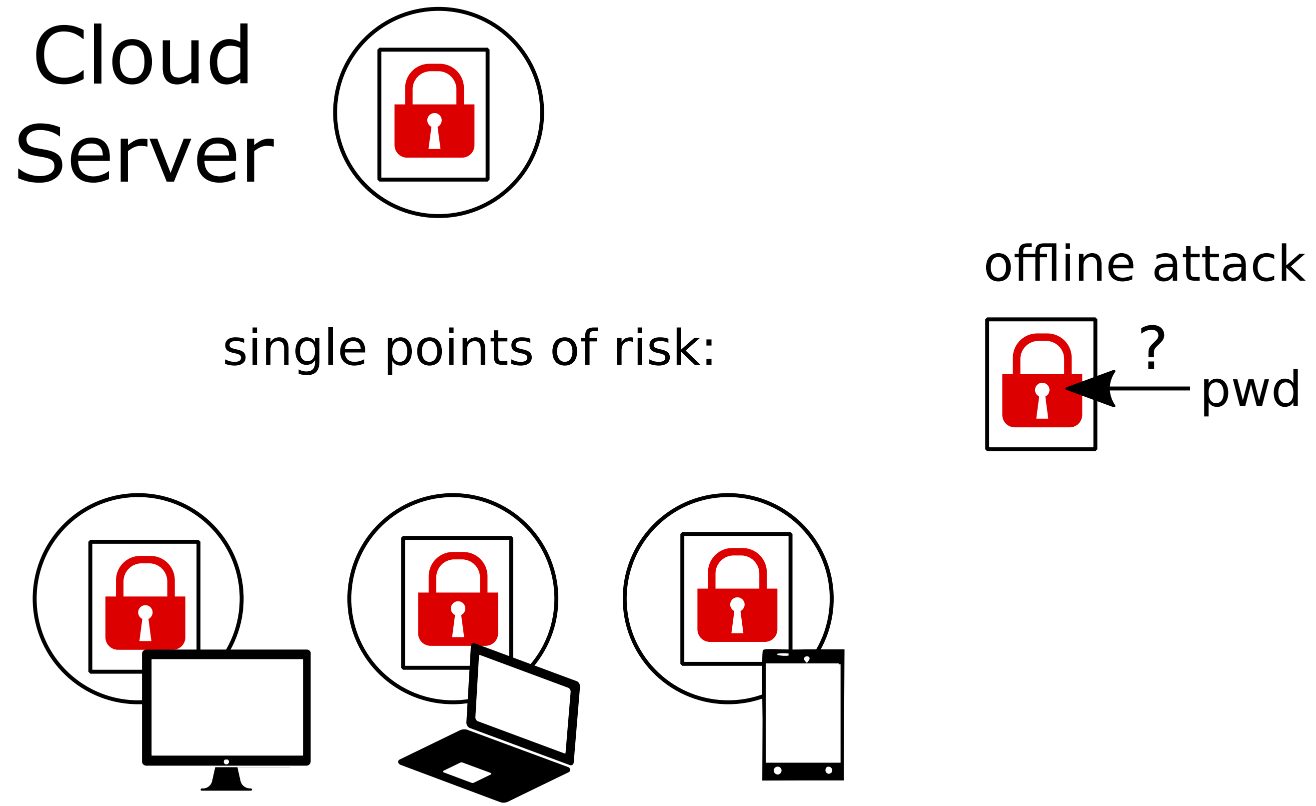
Fazit

Fazit

There are few well-designed and usable password managers.

Fazit

Client AND (in most products) server are *point of risk* for pwd brute force



Can't we do better?

SpreadPass Project

SpreadPass Project

Cloud-based, open source password manager resistant against offline attacks:

- Using a PAKE variant of traditional (threshold) secret sharing: PPSS

SpreadPass Project

Cloud-based, open source password manager resistant against offline attacks:

- Using a PAKE variant of traditional (threshold) secret sharing: PPSS
- One needs the information of threshold many servers AND a user device to mount a brute force on *pwd*.

SpreadPass Project

Cloud-based, open source password manager resistant against offline attacks:

- Using a PAKE variant of traditional (threshold) secret sharing: PPSS
- One needs the information of threshold many servers AND a user device to mount a brute force on *pwd*.

SpreadPass Project

Cloud-based, open source password manager resistant against offline attacks:

- Using a PAKE variant of traditional (threshold) secret sharing: PPSS
- One needs the information of threshold many servers AND a user device to mount a brute force on *pwd*.

Caveat: no offline mode, only meant for online use

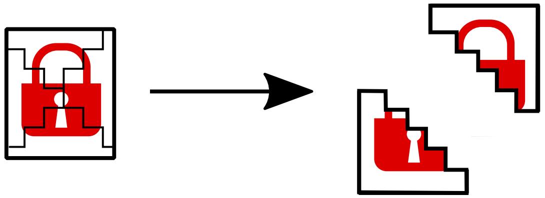
Cryptographic protocols are based on

S. Jarecki, H. Krawczyk, et al., *Highly-Efficient and Composable Password-Protected Secret Sharing* (Or: How to Protect Your Bitcoin Wallet Online), IEEE European Symposium on Security and Privacy 2016,

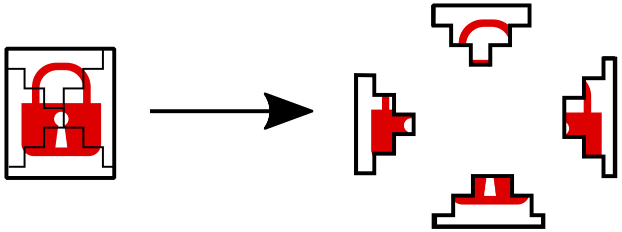
extended to serve functionality of a standard password manager.

Secret sharing

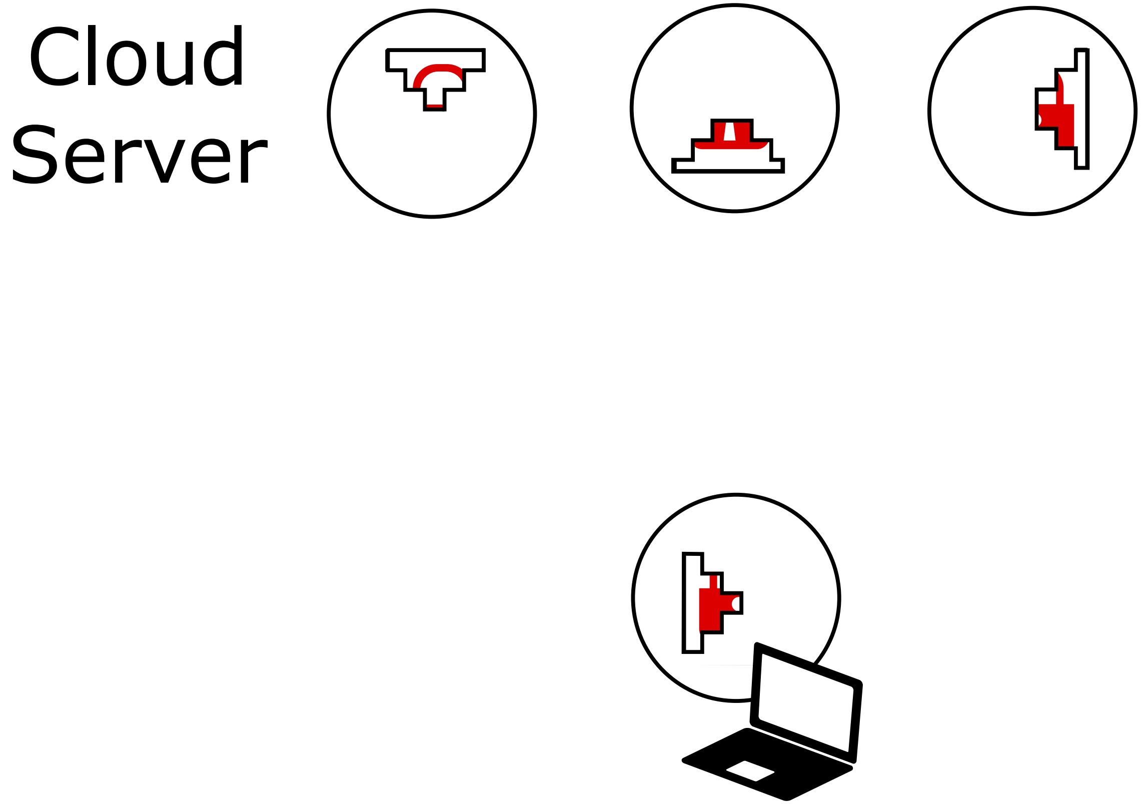
Secret sharing



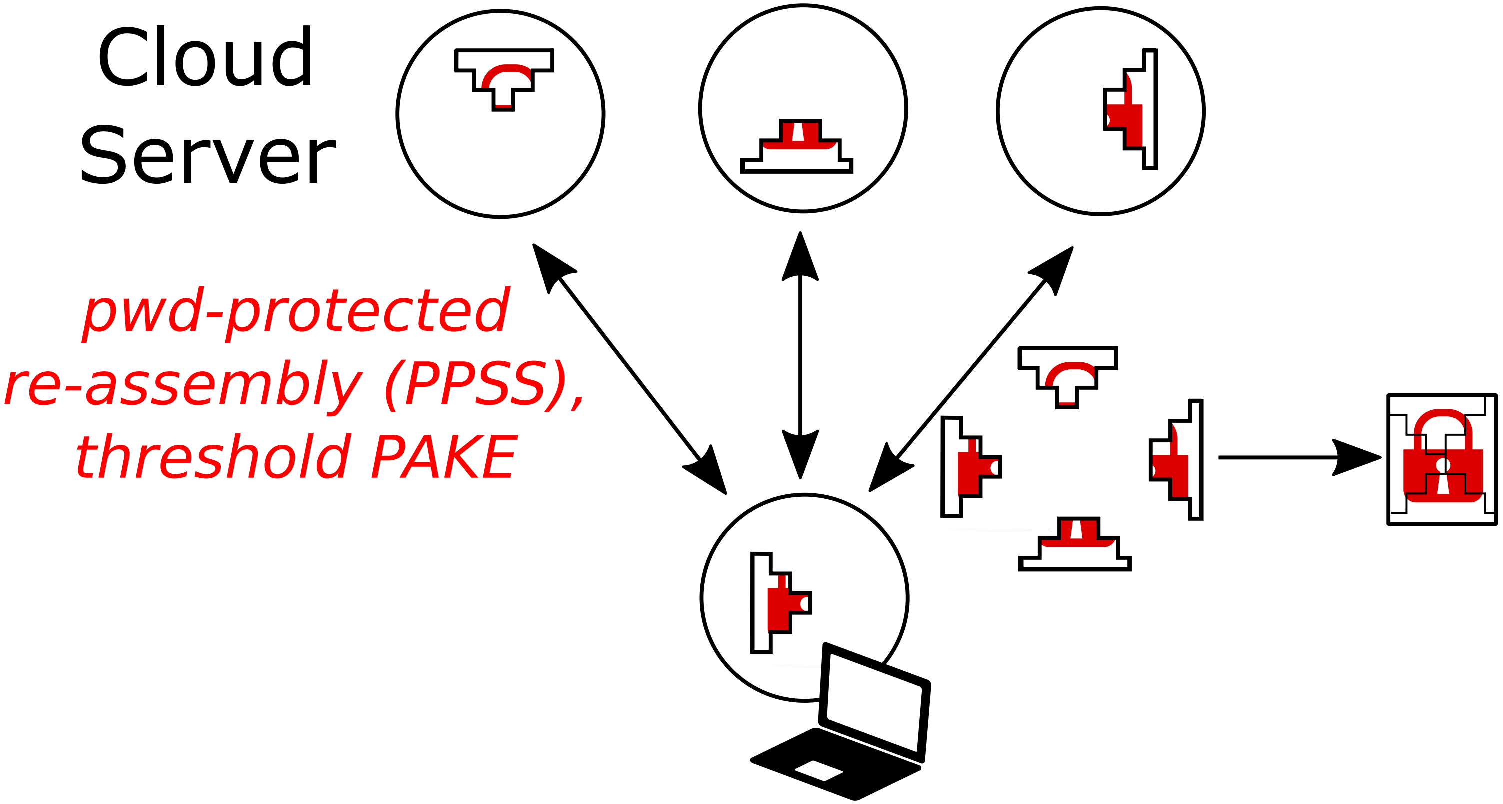
Secret sharing



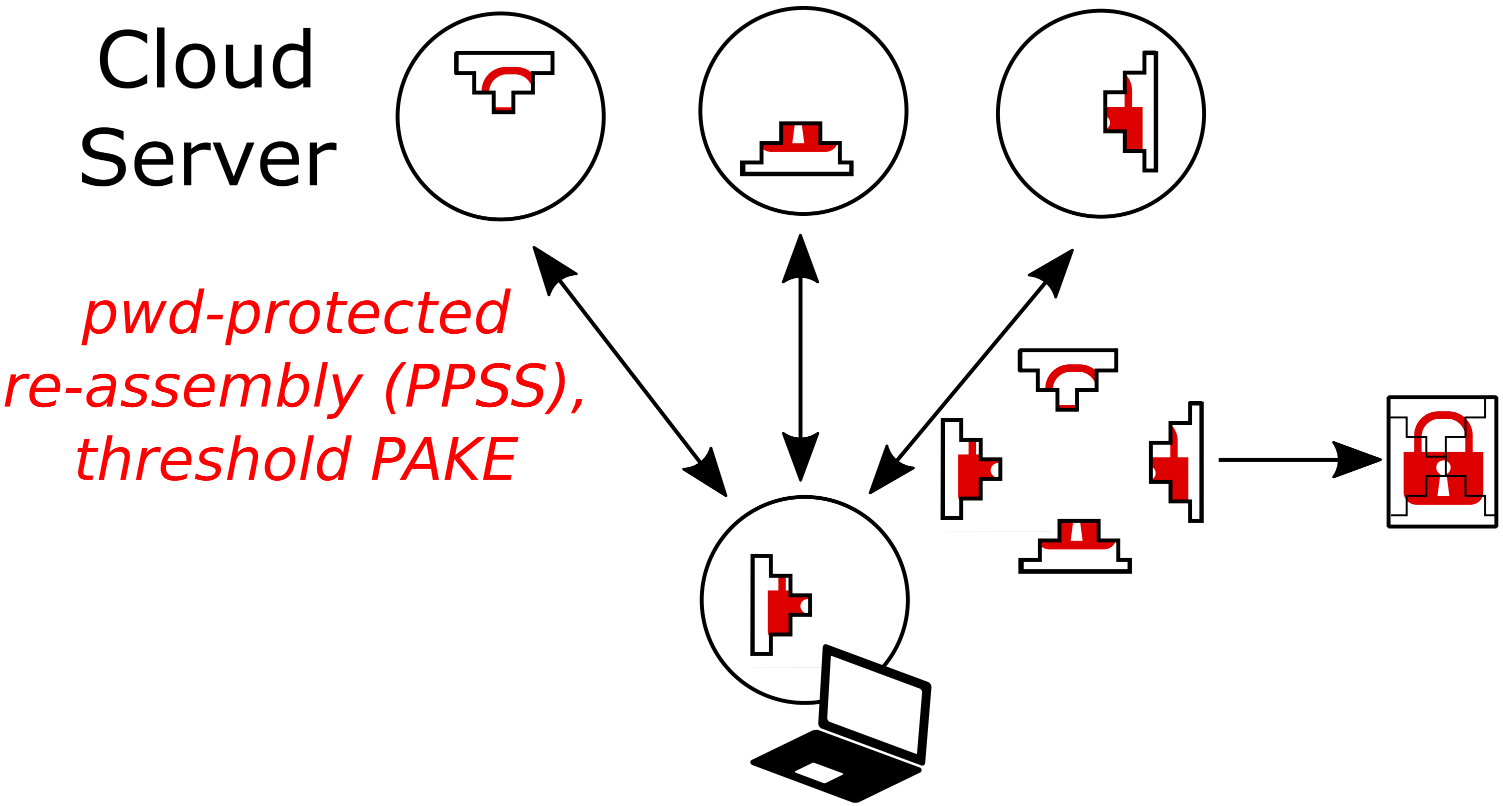
Secret sharing



Secret sharing



Secret sharing



Offline pwd attack: need to control/break into all (or a threshold number of) server plus the client device.

Project description

Master student of mine will implement protoype password manager

- prototype server plus client(s), e.g. browser add-on, mobile app

Project description

Master student of mine will implement protoype password manager

- prototype server plus client(s), e.g. browser add-on, mobile app
- device managment: secure add/delete devices

Project description

Master student of mine will implement protoype password manager

- prototype server plus client(s), e.g. browser add-on, mobile app
- device managment: secure add/delete devices
- server management: add/delete server

Project description

Master student of mine will implement protoype password manager

- prototype server plus client(s), e.g. browser add-on, mobile app
- device managment: secure add/delete devices
- server management: add/delete server
- other functionalities for improving user experience

Project description

But we still look for somebody who is ready to

- implement backend crypto lib

Project description

But we still look for somebody who is ready to

- implement backend crypto lib

Project description

But we still look for somebody who is ready to

- implement backend crypto lib

Note: All protocols do exist, there is no need to work on mathematical level.

Project aims

- open source prototype, well documented

Project aims

- open source prototype, well documented
- a well written white paper + conference publication

Project aims

- open source prototype, well documented
- a well written white paper + conference publication
- etc...

Contact: ulrich.haboeck@fh-campuswien.ac.at

Contact: ulrich.haboeck@fh-campuswien.ac.at

Thank you!–