Why use SPHINX instead of random other popular password manager?

stf

<2022-07-02 Sat>

Disclaimer

This talk considers only one perspective: the security of your passwords. UI/UX and other fancy features are not in scope and can affect your choice of tool.

Threats & Mitigations

- password recovery attack → High entropy (>80bit) passwords
- password reuse attack → unique passwords
- password db leak → no db-based pwd manager
- phishing attacks → force bind passwords to services
- forgetting master password → analog solutions
- key-sniffing the master password → ¯_(ツ)_/¯

password safe/keepass/keepassX/keepassXC

ancient offline encrypted xml file with structured free-text

keepass et al crypto[1]

- masterkey := sha256(masterpassword || ?keyfile)
- v1: aes-cbc/twofish(aes-kdf(masterkey),database) + sha256(database)
- v2: k := aes-kdf/argon2(id|d)(masterkey) ; ciphertext := aes-cbc/chacha20(k, (database)) + hmac-sha256(k, ciphertext)

In case of password db (which is probably next to the keyfile - if used) leak allows an offline-bruteforce attack against the masterpassword. if you use v1 upgrade asap if possible.

[1] src: https://keepass.info/help/base/security.html

pass / pwd.sh

gpg assymetrically offline encrypted files with free text.

pass / pwd.sh crypto

- KDF is iterated-and-salted S2K(sha1), similar to pbkdf2, which is quite GPU friendly[2]
- Encryption: k = random(), $\text{encrypt}_{\text{asym}}$(pubkey, k) $||$ aes128-cfb/cast5(k,file)

In case of a password db leak also the private gpg key needs to be leaked.

- if the private key is not encrypted then win,
- else offline bruteforce attack against the gpg key

Not only provides access to all passwords, but also to all other cryptograms protected by that key.
If combined with a HW PGP token like a cryptostick, this can be pretty secure though.

[2] `https://crypto.stackexchange.com/a/3255`

passage

like pass, but instead of gpg age, which means nicer crypto algo defaults.

In case of a password db leak also the private gpg key needs to be leaked.

- if the private key is not encrypted then win,
- else offline bruteforce attack against the ~~gpg~~ age key

Not only provides access to all passwords, but also to all other cryptograms protected by that key.

bitwarden

online encrypted files with free text.

bitwarden crypto

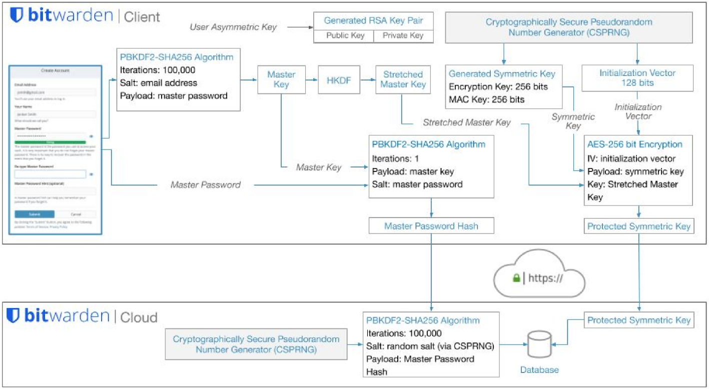

Figure: Bitwarden password hashing, key derivation, and encryption

- items encrypted with aes-cbc.
- javascript crypto.

src: `https://bitwarden.com/images/resources/security-white-paper-download.pdf/`

1password

online encrypted files with free text

1password crypto[3]

Key Derivation

1. p $\leftarrow$ unicode$_{\text{nfkd}}$(trim(password))
2. s $\leftarrow$ HKDF(salt, version, email, 32)
3. $k_m \leftarrow$ PBKDF2-SHA256(p, s, 100000)
4. $k_A \leftarrow$ HKDF(secret-key, version, ID, $||\text{k}_\text{m}||$)
5. $k_m \leftarrow k_m \oplus k_A$
6. priv, pub = rsa-2048-keypair()
7. pk$_{\text{enc}}$ = aes-gcm(k_m, priv)
8. vault$_{\text{key}}$ = random(256b)
9. vk$_{\text{enc}}$ = rsa-oaep(pub, vault$_{\text{key}}$)

step 1-5: for SRP auth key, but with different salt.

[3] src: https://1passwordstatic.com/files/security/1password-white-paper.pdf

1password crypto cont'd

- basically like pass/gpg , but online, aes256gcm instead of aes128cfb, and with a secret key mixed into the master password.
- the secret key mixed into the master password prohibits the 1password server to bruteforce the master password, which is cool.
- since accessing the encrypted keys requires SRP authentication, either the encrypted keys need to be leaked from the client when legitimally authenticated - but then also the master password could be keylogged.
- Auth attempts are rate limited.
- unlike the others, this **seems** to eliminate offline bruteforce-attacks, which is also cool.

Crypto so far

- Home-cooked KDF (aes-kdf, s2k) or pbkdf, maybe argon2
- symmetric encryption: aes-(cbc/cfb/gcm)
- sometimes RSA
- sometimes with an additional secret key mixed into the master key.
- either antique and/or over-engineered

SPHINX

magic silverbullets to the rescue \o/

SPHINX[4] - a password Store that Perfectly Hides from Itself (No eXaggeration)

$$rwd := Hash_{memhard}(pwd || \frac{Hash_{2curve}(pwd) * r * k}{r})$$

[4] https://eprint.iacr.org/2015/1099

Appeal to authority

Designed by Levchin award winner Hugo Krawczyk, who also came up with HMAC, HKDF, OPAQUE, HMQV, SIGMA, UMAC, etc.

SPHINX Benefits

- **information theoretically secure**[5] password store
- manager does not know anything about the password
- manager salt independent from input/output passwords
- can use arbitrary number of "master" passwords
- unless both k and rwd leak only online bruteforce attacks possible.
- KISS: produce only high entropy non-dictionary passwords[6].
- no synching needed

Cons:

- no backups, use password resets or analog means to store rwds for recovery.
- online
- less polished UI

[5] secure against adversaries with unlimited computing resources and time.

[6] we have a mode to set arbitrary max ~40 ascii strings, but use this only if really necessary.

SPHINX ecosystem

- https://github.com/stef/pwdsphinx/blob/master/whitepaper.org
- my server: https://sphinx.ctrlc.hu/
- https://github.com/stef/libsphinx
- https://github.com/stef/pwdsphinx
- https://github.com/stef/websphinx-chrom
- https://github.com/stef/websphinx-firefox
- https://github.com/dnet/androsphinx
- https://github.com/stef/winsphinx
- https://github.com/stef/zphinx-zerver/
- https://github.com/D3vl0per/zphinx-zerver-docker/
- https://github.com/ngi-nix/opaque-sphinx
- soon in a debian-derivative distro of your choice.

Conclusion

if you are using keepass, pass, bitwarden, or similar password managers, you might want to switch[7] or slowly migrate to sphinx to handle your passwords.

[7] importers from these to sphinx are coming real soon now ™

Questions

?