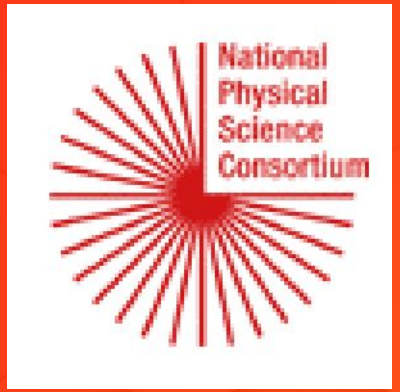
Multi-Factor Key Derivation Function (MFKDF)

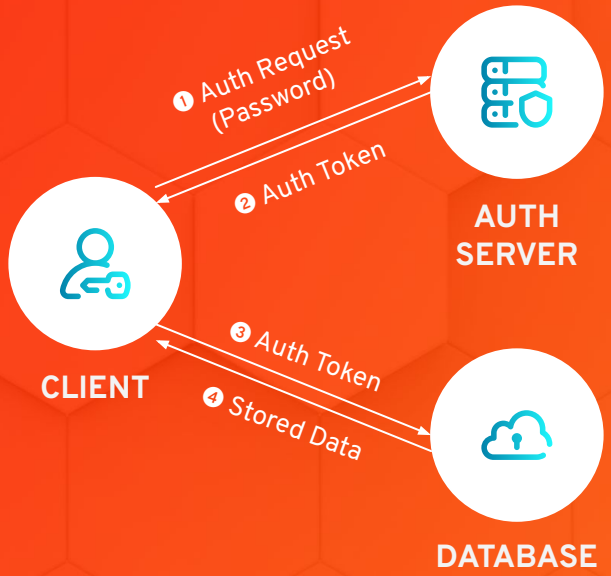
Vivek Nair

Ph.D. Student at UC Berkeley
https://nair.me · vivek@nair.me

Acknowledgments



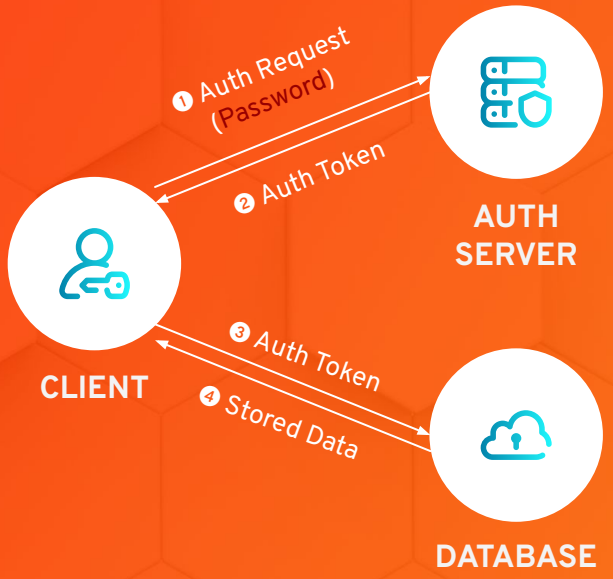
Password Management Service



Password Management Service

Two problems with this architecture:

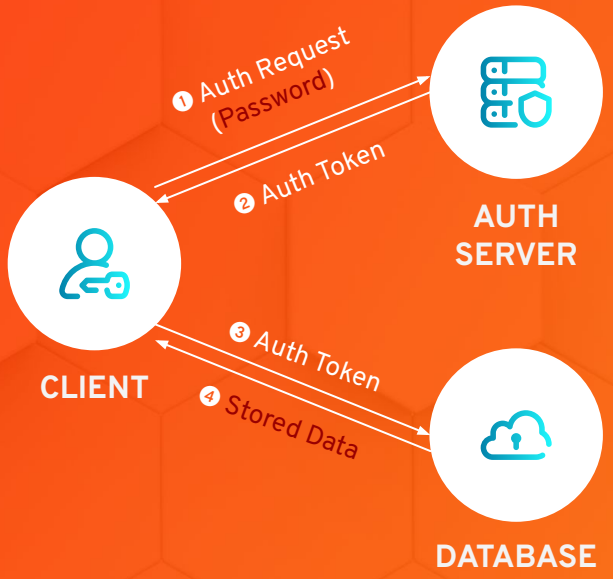
- Passwords are insecure



Password Management Service

Two problems with this architecture:

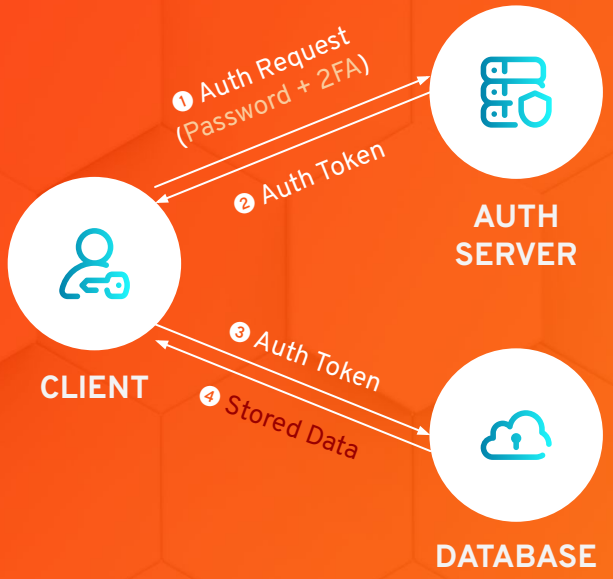
- Passwords are insecure
- Databases are leaky



Password Management Service

Two problems with this architecture:

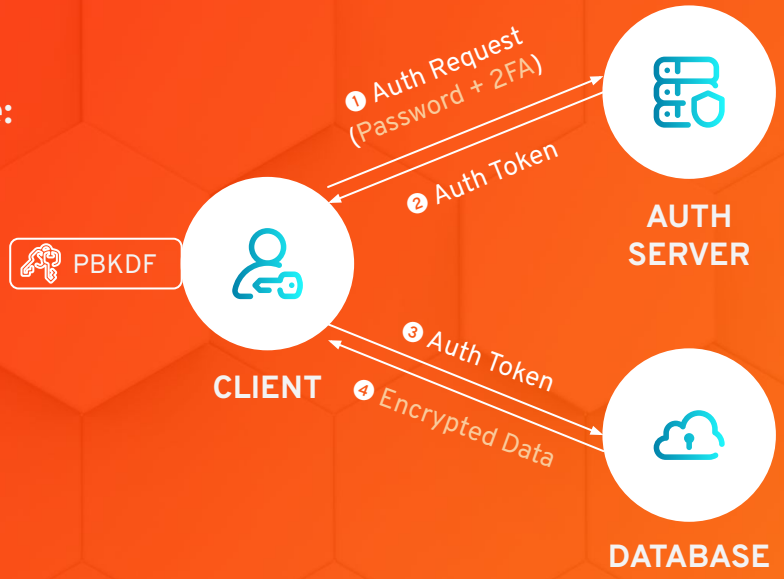
- Passwords are insecure
  - Add MFA!
- Databases are leaky



Password Management Service

Two problems with this architecture:

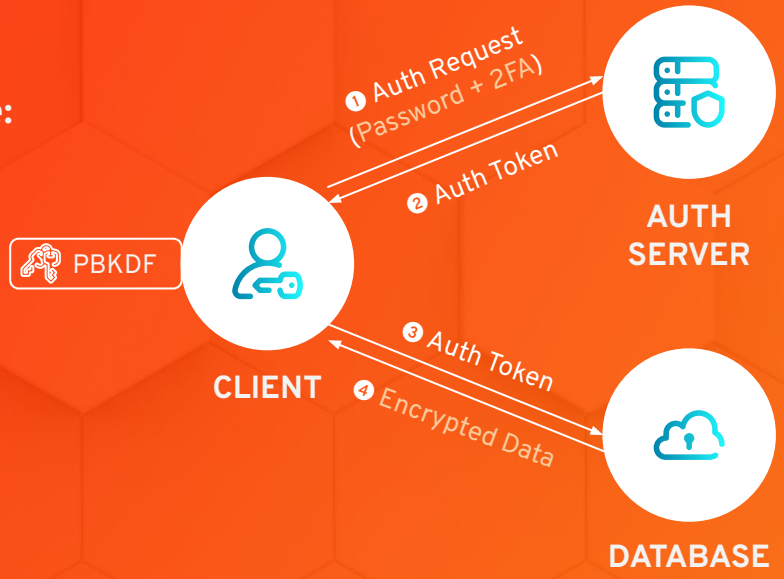
- Passwords are insecure
  - Add MFA!
- Databases are leaky
  - Add PBKDF!



Password Management Service

Two problems with this architecture:

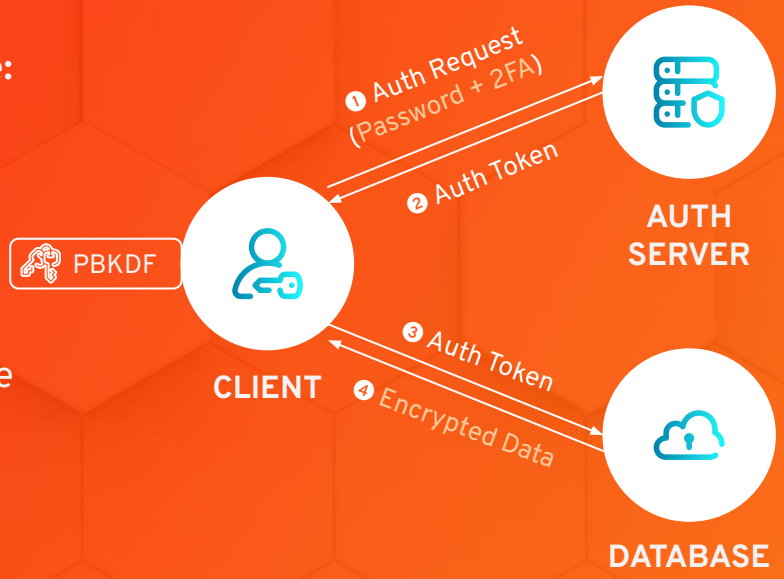
- Passwords are insecure
  - Add MFA!
- Databases are leaky
  - Add PBKDF!



Password Management Service

Two problems with this architecture:

- Passwords are insecure
  - Add MFA!
- Databases are leaky
  - Add PBKDF!
- Can we incorporate MFA into the key derivation function itself?



MULTI-FACTOR KEY DERIVATION

MULTI-FACTOR DERIVED KEY

The **MFKDF** outputs a key as a function of all input factors

FACTOR 01

eg. a **Password**

FACTOR 02

eg. a **TOTP Code**

FACTOR 03

eg. a **U2F Token**

FACTOR 04

eg. **Biometric Data**

FACTOR 01

eg. a **Password**

hunter2

One-Way Function (OWF)

STATIC KEY

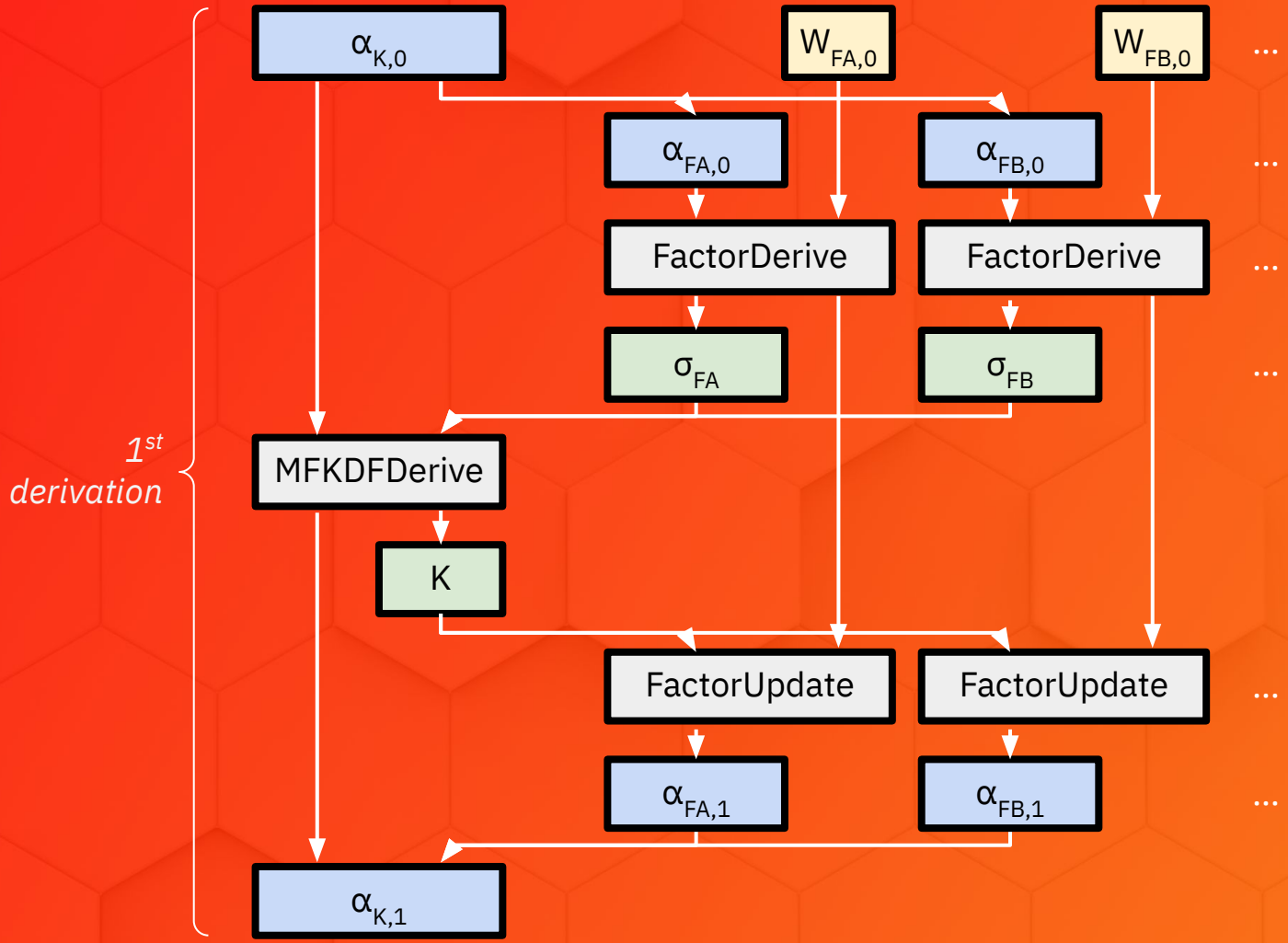
FACTOR 02

eg. a **TOTP Code**

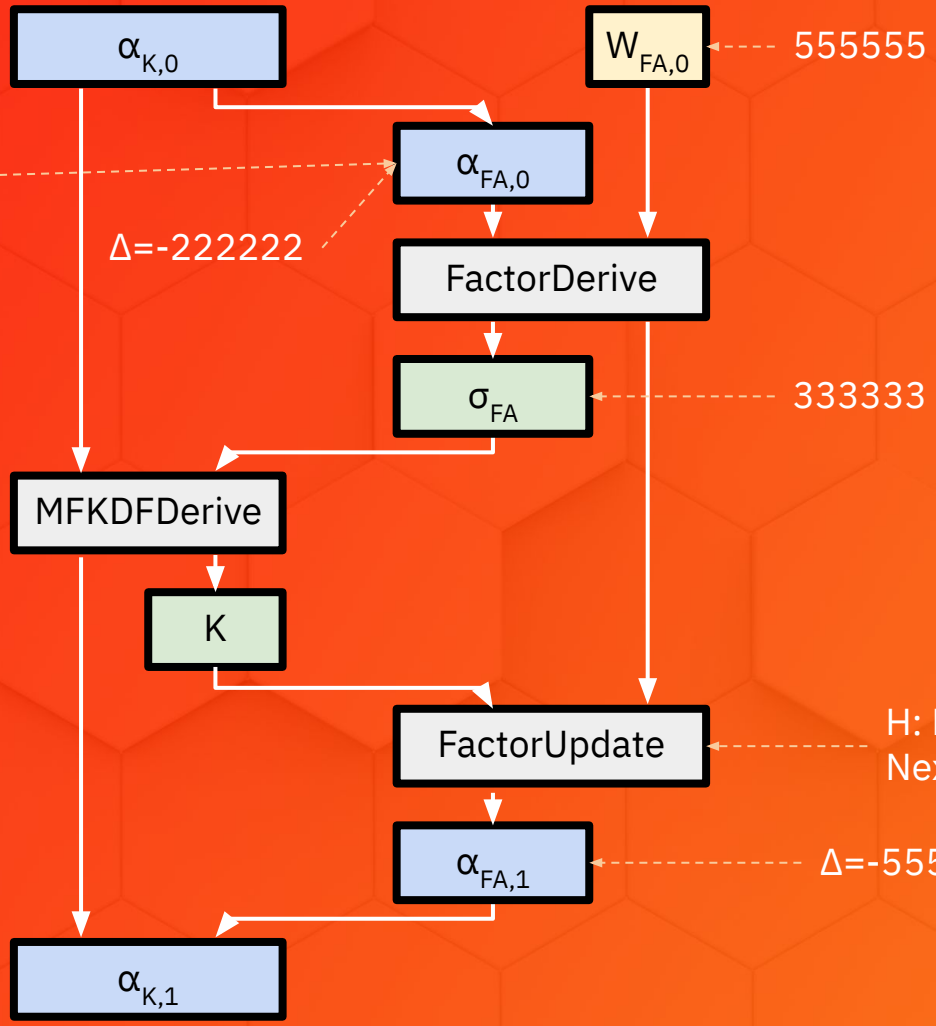
196353
778449
843812
234823
...

???

STATIC KEY


$\alpha_{K,0}$
$W_{FA,0}$
$W_{FB,0}$
...
$\alpha_{FA,0}$
$\alpha_{FB,0}$
...
FactorDerive
FactorDerive
...
σ_{FA}
σ_{FB}
...
MFKDFDerive
K
FactorUpdate
FactorUpdate
...
$\alpha_{FA,1}$
$\alpha_{FB,1}$
...
$\alpha_{K,1}$
1st derivation

K: Derived Key
H: HOTP Key
ct: Enc(H, K)



Other Factors:

Entropy & Brute Force

PBKDF

DK = PBKDF2(PRF, Password, Salt, Rounds, dkLen)

Intentionally inefficient!

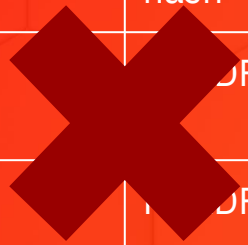
MFKDF

DK = MFAKDF(PRF, [f1,f2,...fn], Rounds, dkLen)
= PBKDF2(PRF, f1 · f2 · f3, Salt, Rounds, dkLen)

Difficulty is on top of all authentication factors!

users

user	hash
user1	DF(password)
user2	DF(password)

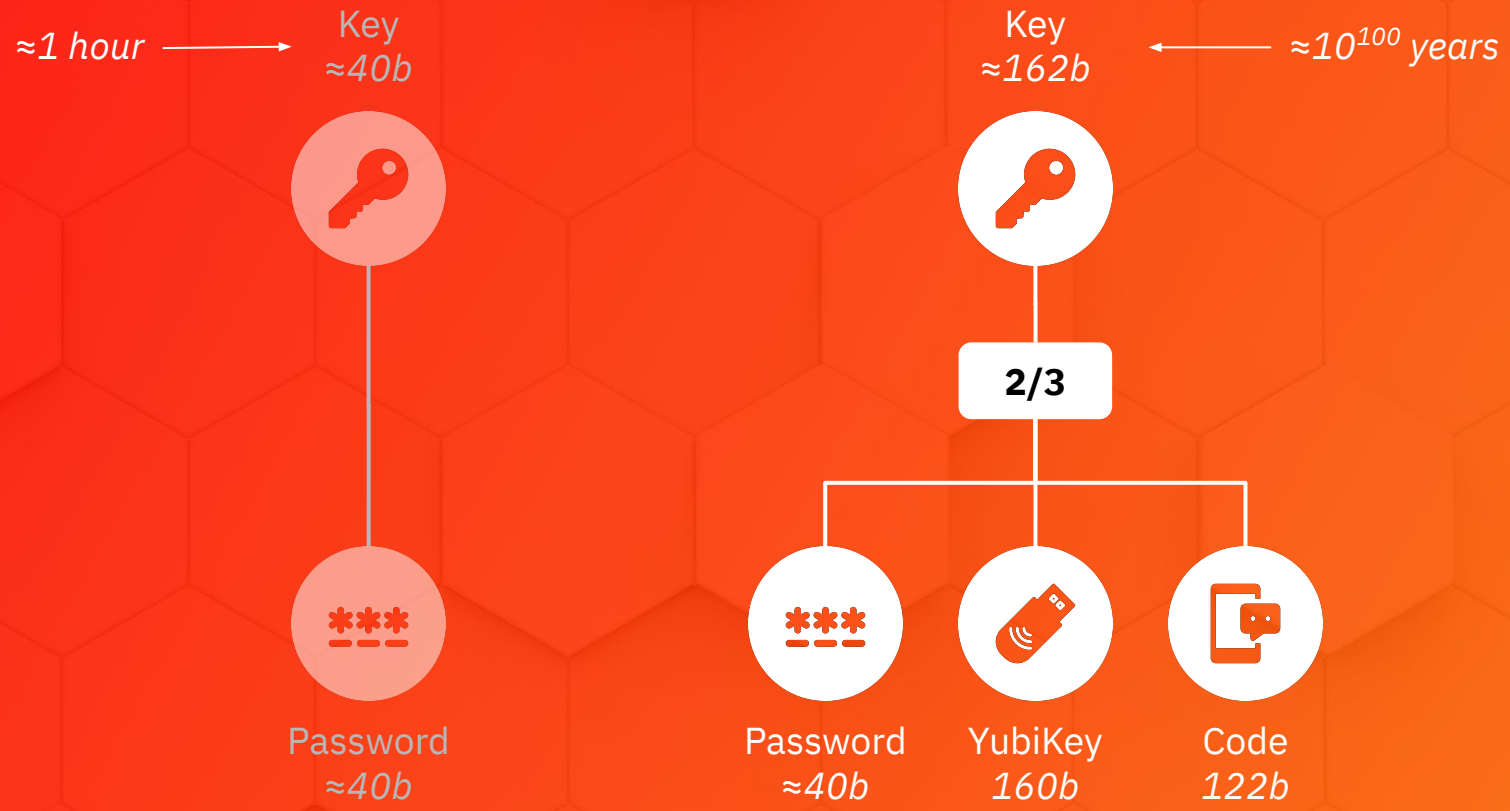


users

user	hash
user1	MFKDF(password, HOTP, ...)
user2	MFKDF(password, HOTP, ...)

MFKDF

MFKDF


≈1 hour
Key
≈40b
***
Password
≈40b
Key
≈162b
≈10^100 years
2/3
***
Password
≈40b
YubiKey
160b
Code
122b

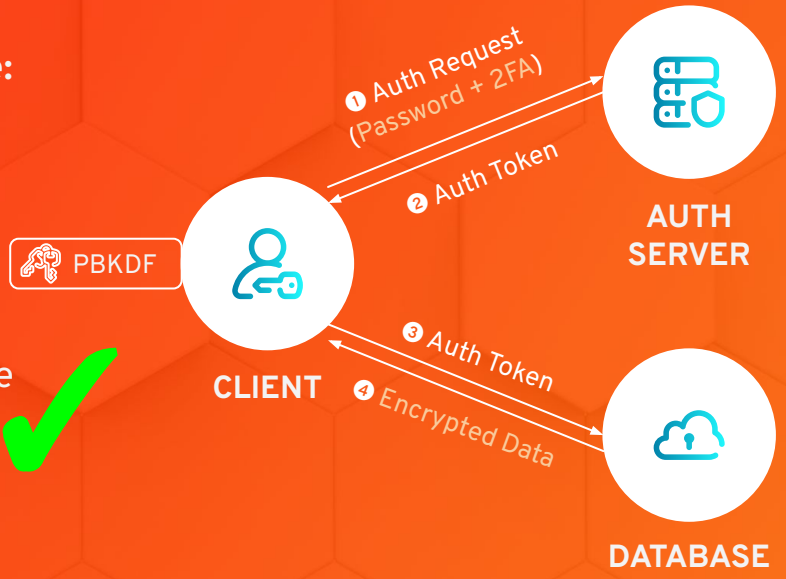
MFKDF

Password Management Service

Two problems with this architecture:

- Passwords are insecure
  - Add MFA!
- Databases are leaky
  - Add PBKDF!

- Can we incorporate MFA into the key derivation function itself? ✓



Password Management Service

NIST SP 800-57: Key Recovery

Data Encryption Key (DEK): Used to encrypt user data

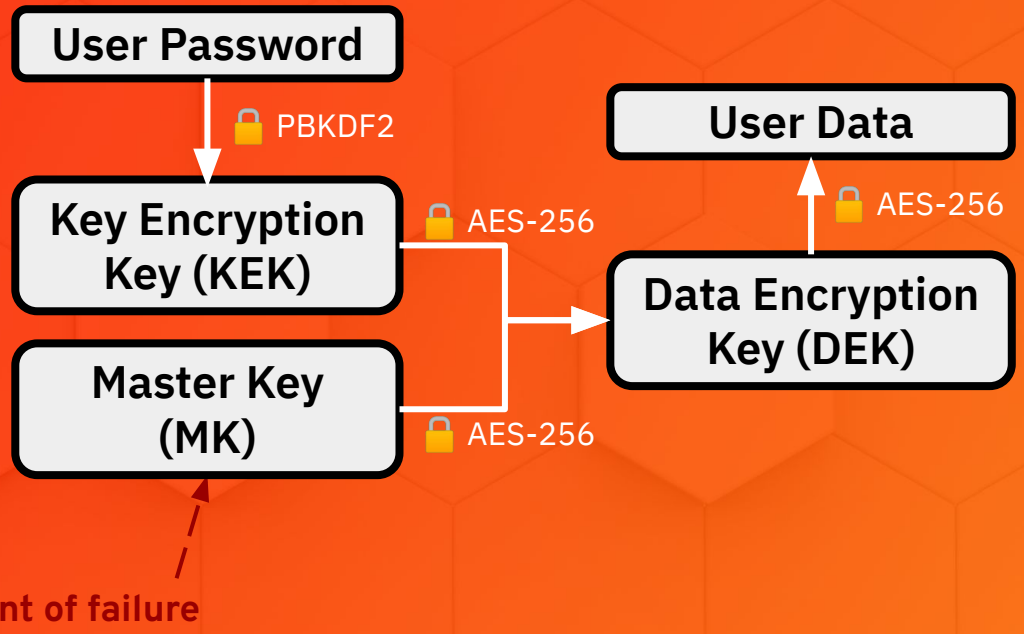
- EncData = Enc(Data, DEK)

Key Encryption Key (KEK): Password-derived key used to encrypt DEK

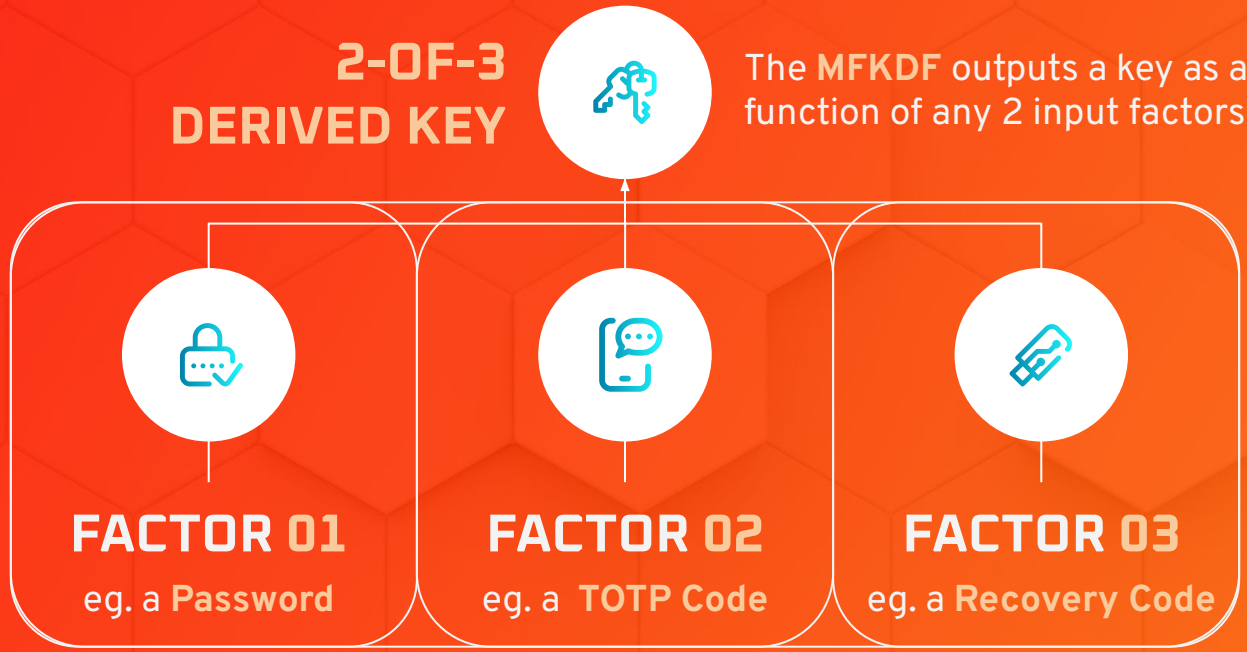
- EncKey = Enc(DEK, KEK)

Master Key (MK): Centrally-stored key used to recover DEK

- RecKey = Enc(DEK, MK)

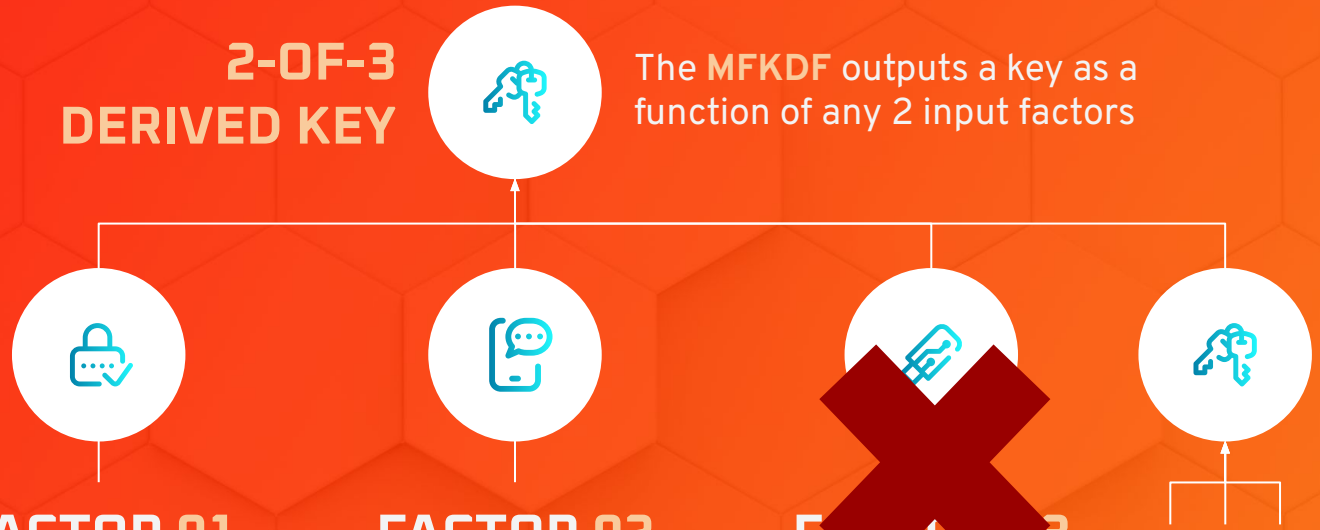


THRESHOLD MULTI-FACTOR KEY DERIVATION



MFKDF

Key Stacking

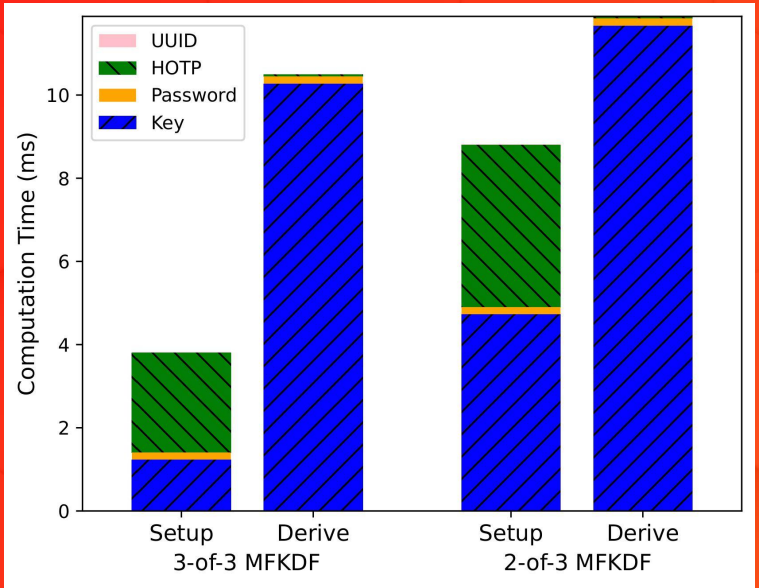


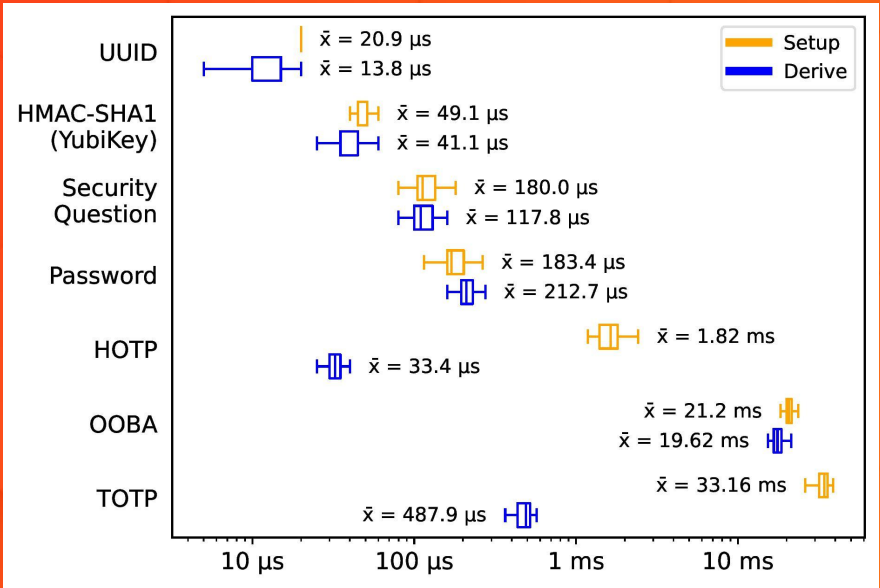
MFKDF


Key
2/3
Password
TOTP
AND
Email
OR
Security Questions
Recovery Code

MFKDF

Performance





mfkdf.com ← pbkdf2.com

MFKDF Docs Tutorials Testing Coverage Demos Videos Get Started

Secure based on argon2id

Fast ≤ 20ms overhead

Transparent fully open-source

Flexible modular design

Go beyond passwords

Most users have notoriously insecure passwords, with up to 81% of them re-using passwords across multiple accounts. MFKDF improves upon password-based key derivation by using all of a user's authentication factors (not just their password) to derive a key. MFKDF supports deriving key material from a variety of common factors, including HOTP, TOTP, and hardware tokens like YubiKey.

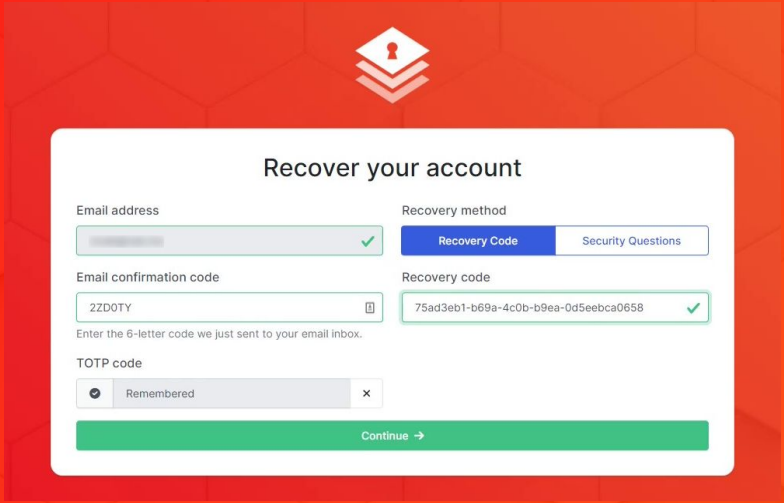
```
const derivedKey = await mfkdf.derive.key(JSON.parse(keyPolicy), {
  password: mfkdf.derive.factors.password('Tr0ub4dour'),
  hotp: mfkdf.derive.factors.hotp(365287),
  recovery: mfkdf.derive.factors.uuid('9b1deb4d-3b7d-4bad-9bdd-2b0d7b3dcb6d')
})

console.log(derivedKey.key.toString('hex')) // -> 34d20ced439ec2f871c96ca377f25771
```

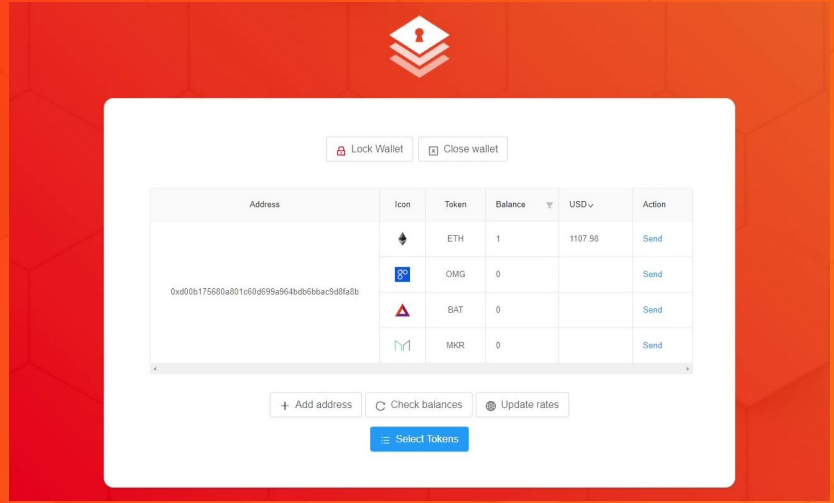
Increased key entropy

All factors must be simultaneously correctly guessed to derive a key using MFKDF, meaning that they can't be individually brute-force attacked. MFKDF keys are thus exponentially harder to crack while remaining just as fast to derive on the fly as password-derived keys for users with the correct credentials.

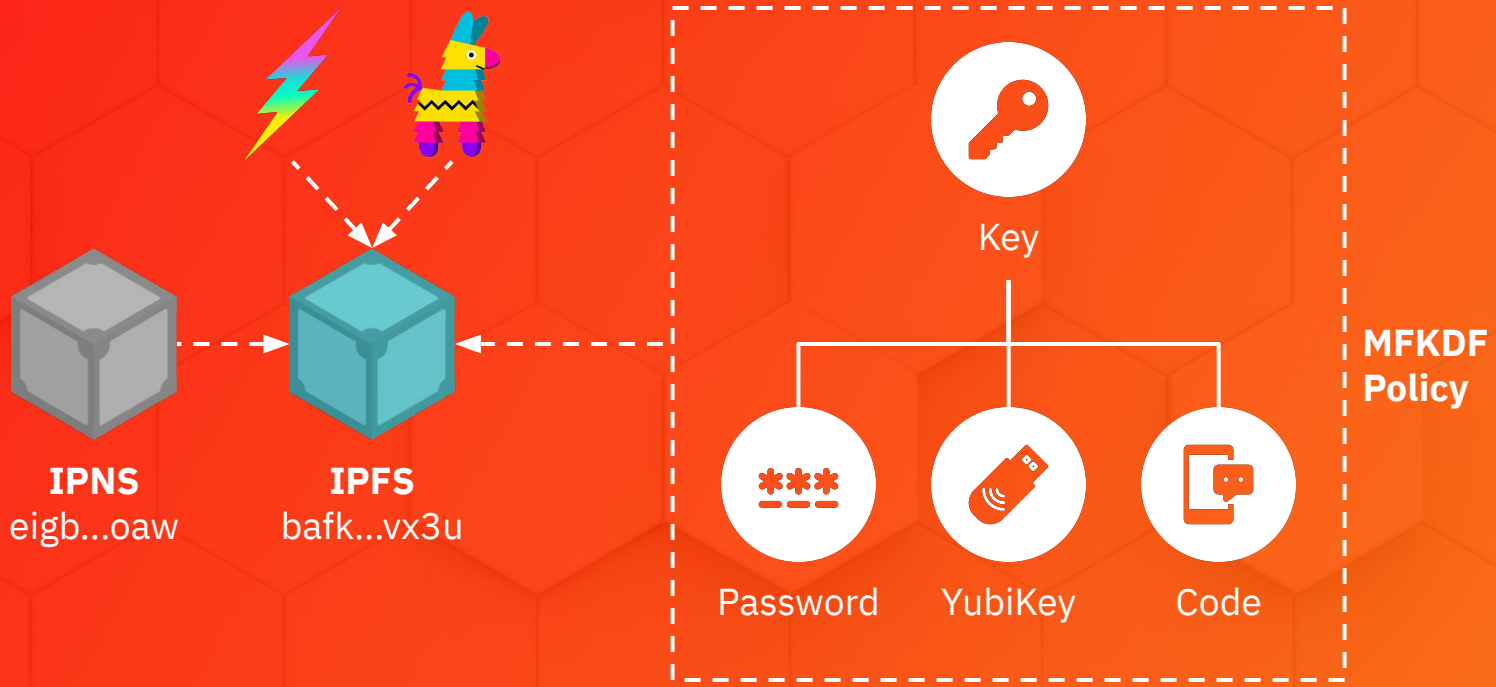
Centralized & Decentralized Demos



https://demo.mfkdf.com



https://wallet.mfkdf.com


Key
MFKDF
Policy
***
Password
YubiKey
Code
IPNS
eigb...oaw
IPFS
bafk...vx3u

MFKDF

Custodial Wallet	Non-custodial Wallet	MFKDF Wallet
✔ Portability	✘ Portability	✔ Portability
✔ Recoverability	✘ Recoverability	✔ Recoverability
✔ MFA	✘ MFA	✔ MFA
✔ Common Factors	✘ Common Factors	✔ Common Factors
✘ Decentralized	✔ Decentralized	✔ Decentralized
✘ Trustless	✔ Trustless	✔ Trustless

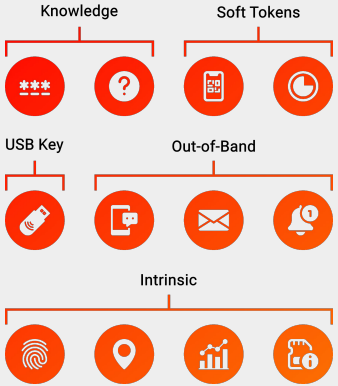
MFKDF

PBKDF2 is also used in...

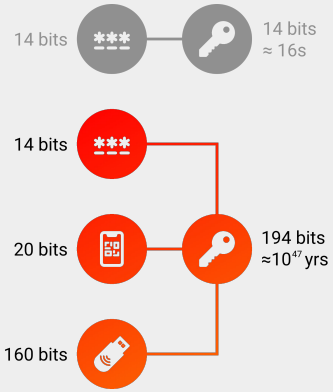
MFKDF Summary

MFKDF

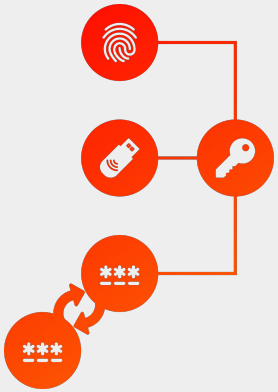
USABILITY & FACTOR COMPATIBILITY



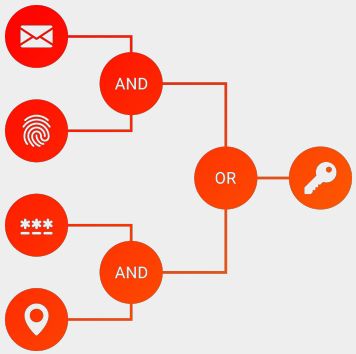
EXPONENTIAL SECURITY



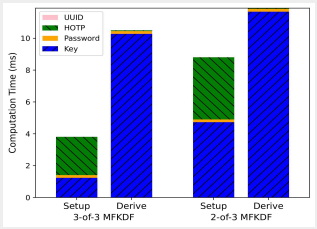
CLIENT-SIDE RECOVERY

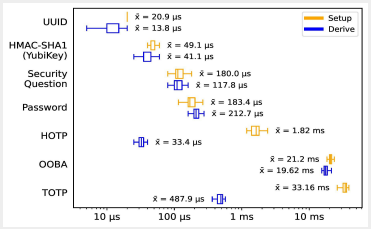


POLICY ENFORCEMENT

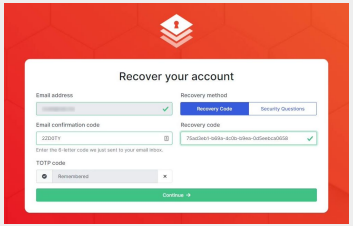


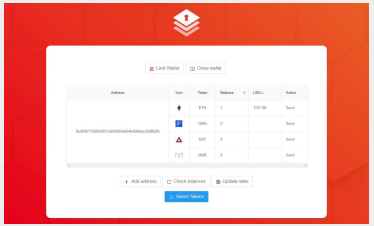
HIGHLY PERFORMANT





NEW & EXISTING APPLICATIONS





Thanks!

https://mfkdf.com

https://arxiv.org/abs/2208.05586

https://github.com/multifactor/mfkdf