1Password Internals

How does this thing actually work, anyway?

BSides Delaware, November 2018

David Schuetz

Background

- Passwords are a pain in the neck!
  - They need to be strong
  - And unique
- Which means we can't remember them
- And let’s not even mention 2FA
- So we need password managers

Why 1Password?

- Well designed system
- Lots of great features
- Very transparent
  - Reasonably well documented
  - Some parts vague or incomplete
- With help, I think I’ve got it mostly nailed down
  - (I’m kind of obsessive about crypto puzzles)

Why am I here?

- Lots of complicated technology today
- We sort of “assume” these Black Boxes are safe
- But how do we know for sure?

- Need to really understand to assess risk
- Documentation is great
- But must “Verify”

- Best way to know you’ve understood how something works:
  - Teach it to someone else!

Topics for Today

- Login and encryption (local and server)
- Sharing vaults
- Handling multiple accounts
- Recovering from lost password

- High-level “easy to understand”
- Including a few deep technical details
  - Build your own!

Not a topic: comparing to other tools

- I've used 1Password for almost 10 years. We use it at work.
- I’m told there are other password managers out there.
- I’m told some of them are pretty good.
- I’m not going to compare 1Password to any of them.

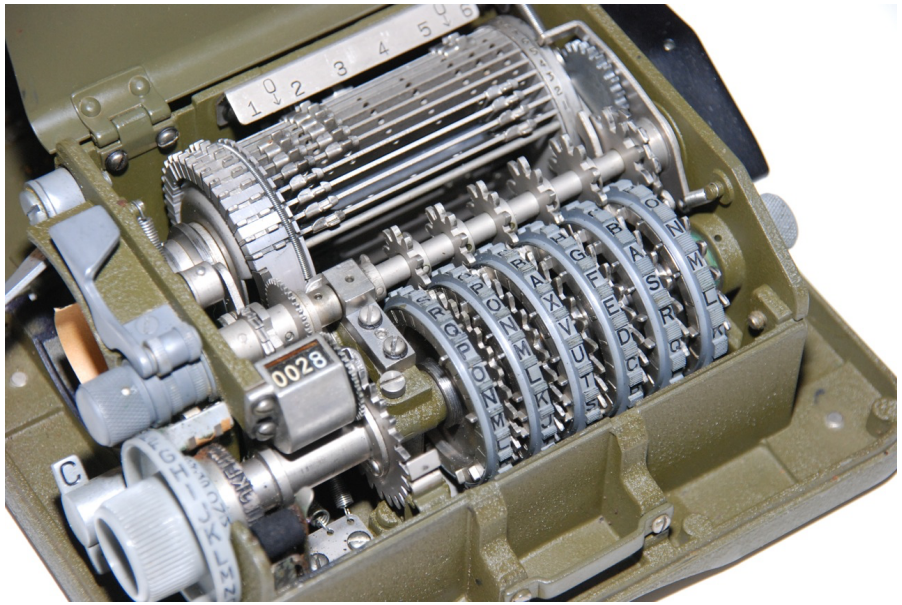
- Love to see more “How things work” talks

1Password Nomenclature

- Account
  - Group of vaults – like your “Expel” account, your “Home” account
- Vault
  - Collection of items (passwords, notes)
- Master Password
  - How you unlock 1Password
- Secret Key
  - An account-specific code
  - That "A3-abcdef-123456-blah-blah-blah" thing

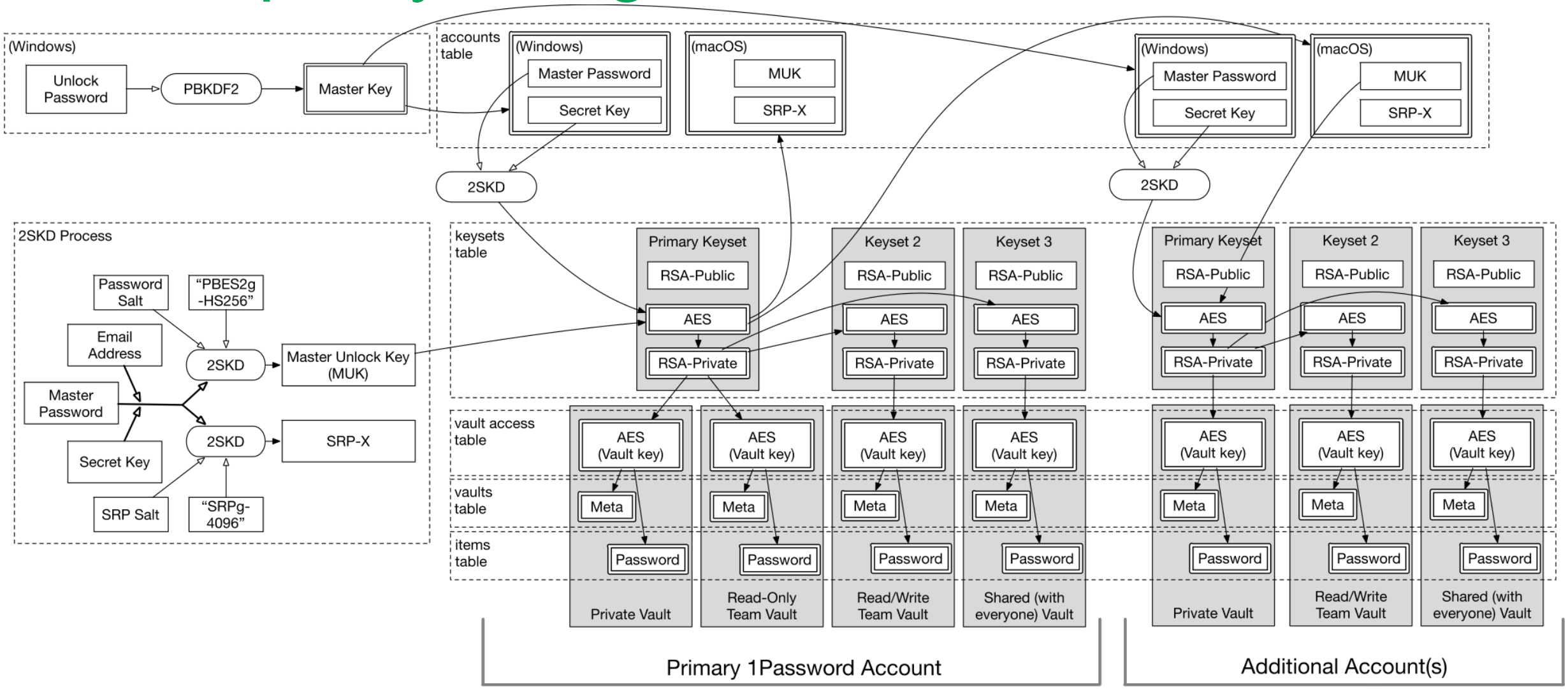
Cryptography

- Fear not! We won't get too technical
- Some useful terms:
  - Symmetric Key – AES
  - Public and Private Keys – RSA
  - Hash Functions – SHA, HMAC, etc.
  - Key Derivation Functions (HKDF, PBKDF2)
- If already understand these – great!
- If not, just treat them as "black boxes"
  - We'll show how they work in context
  - It's the larger picture that's important



Let’s Go!

It’s all pretty straightforward…

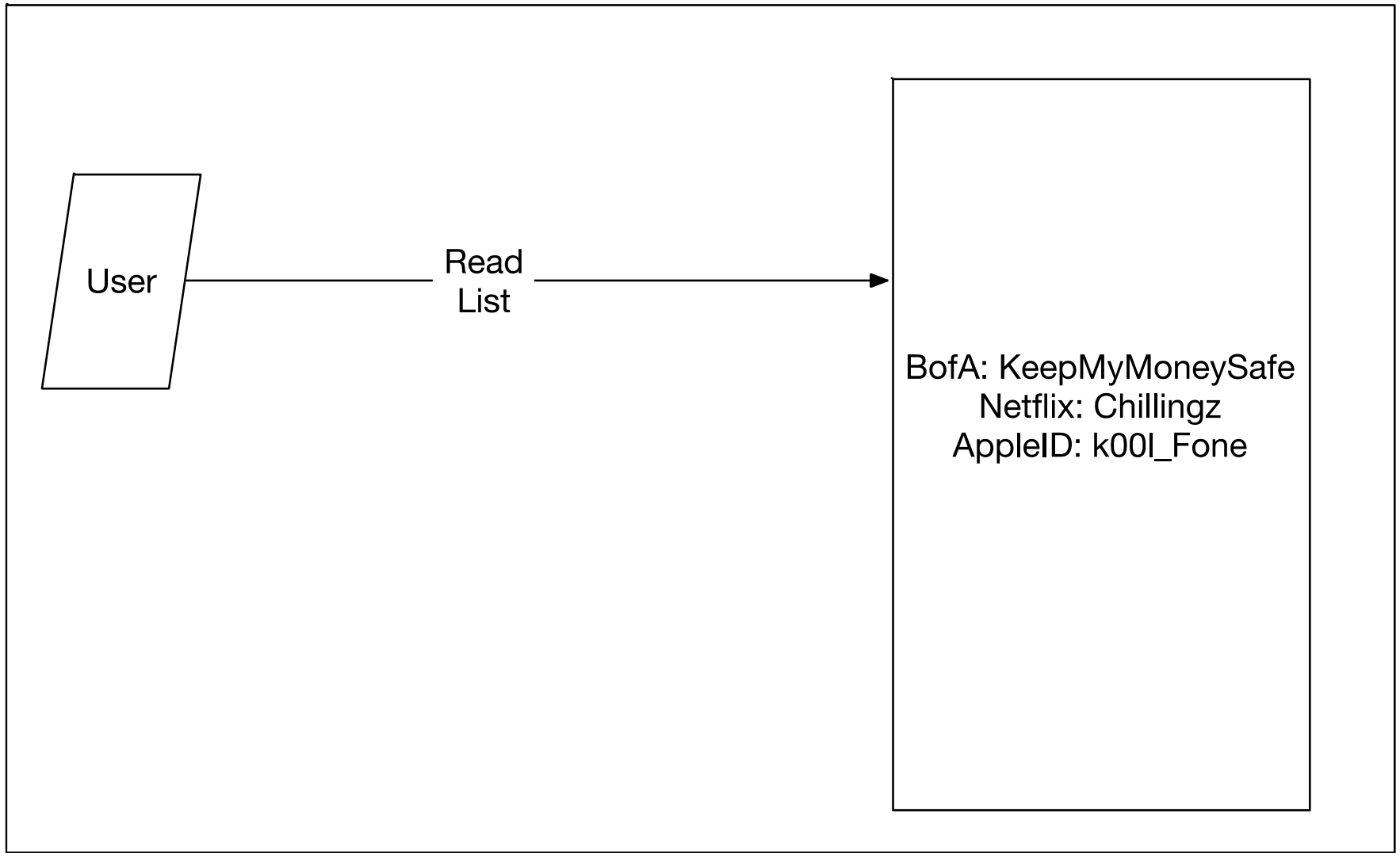


Does it need to be that complicated?

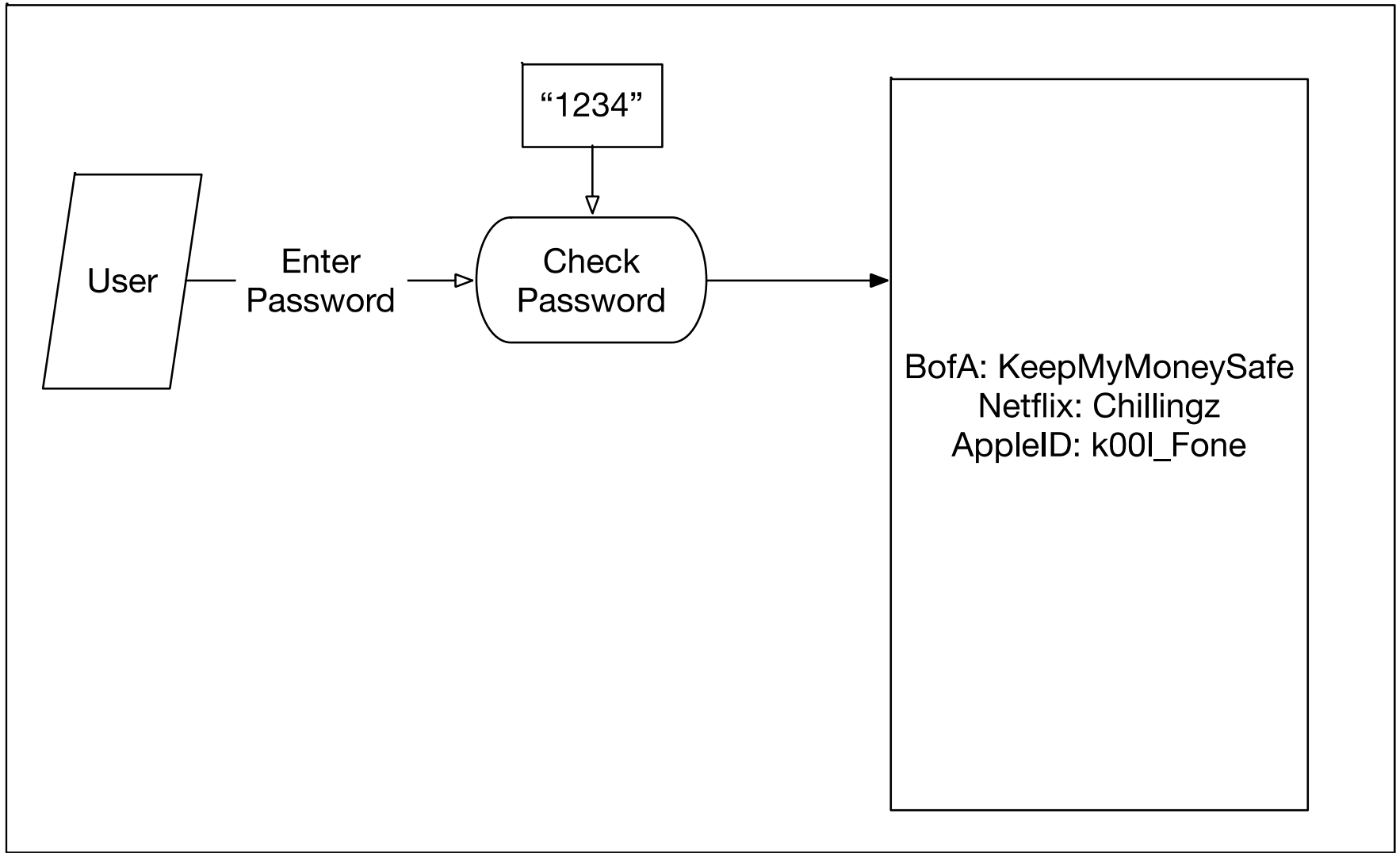
Actually, yes.

Let’s build a simple password manager together

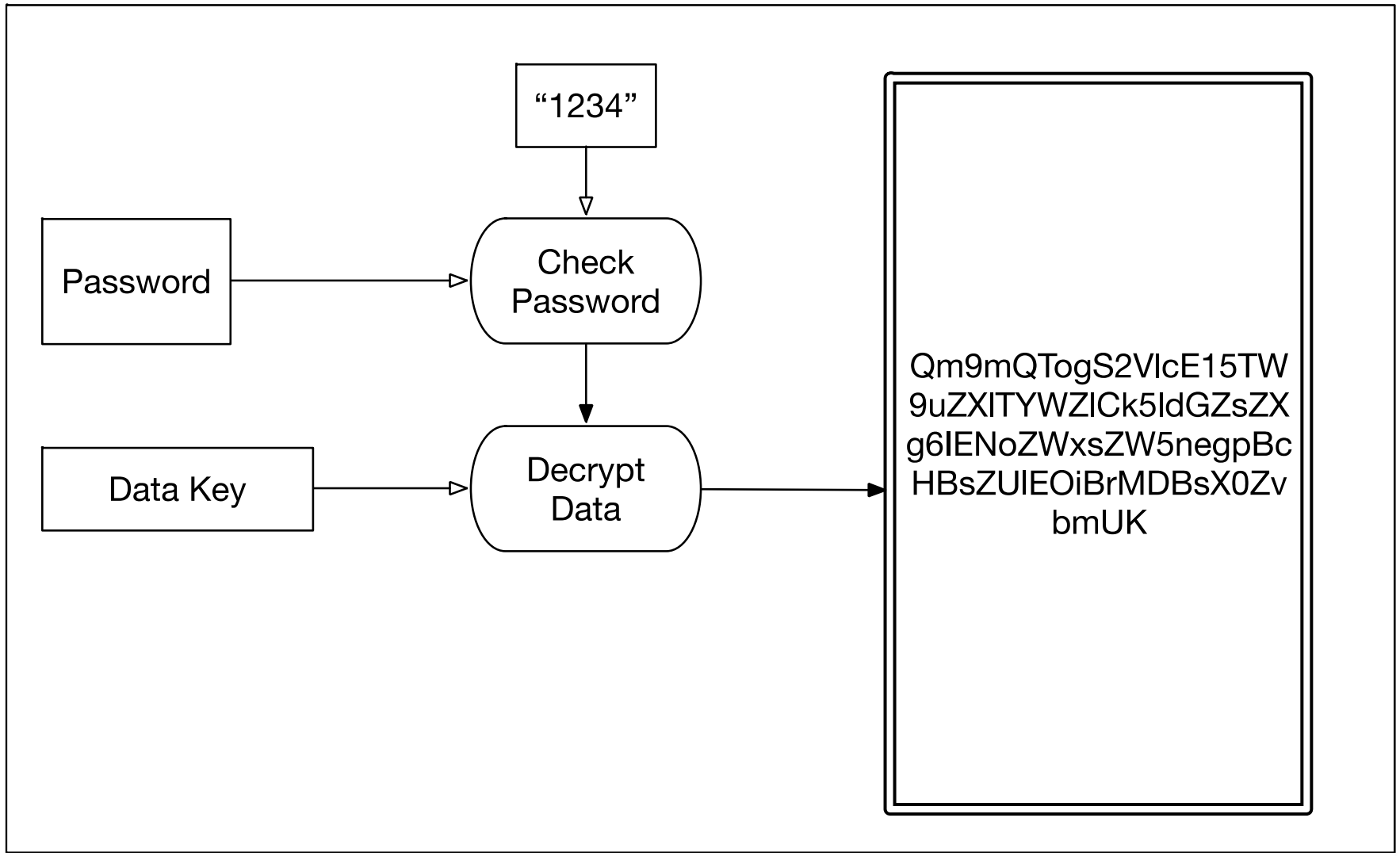
Simple List


User
Read
List
BofA: KeepMyMoneySafe
Netflix: Chillingz
AppleID: k00l_Fone

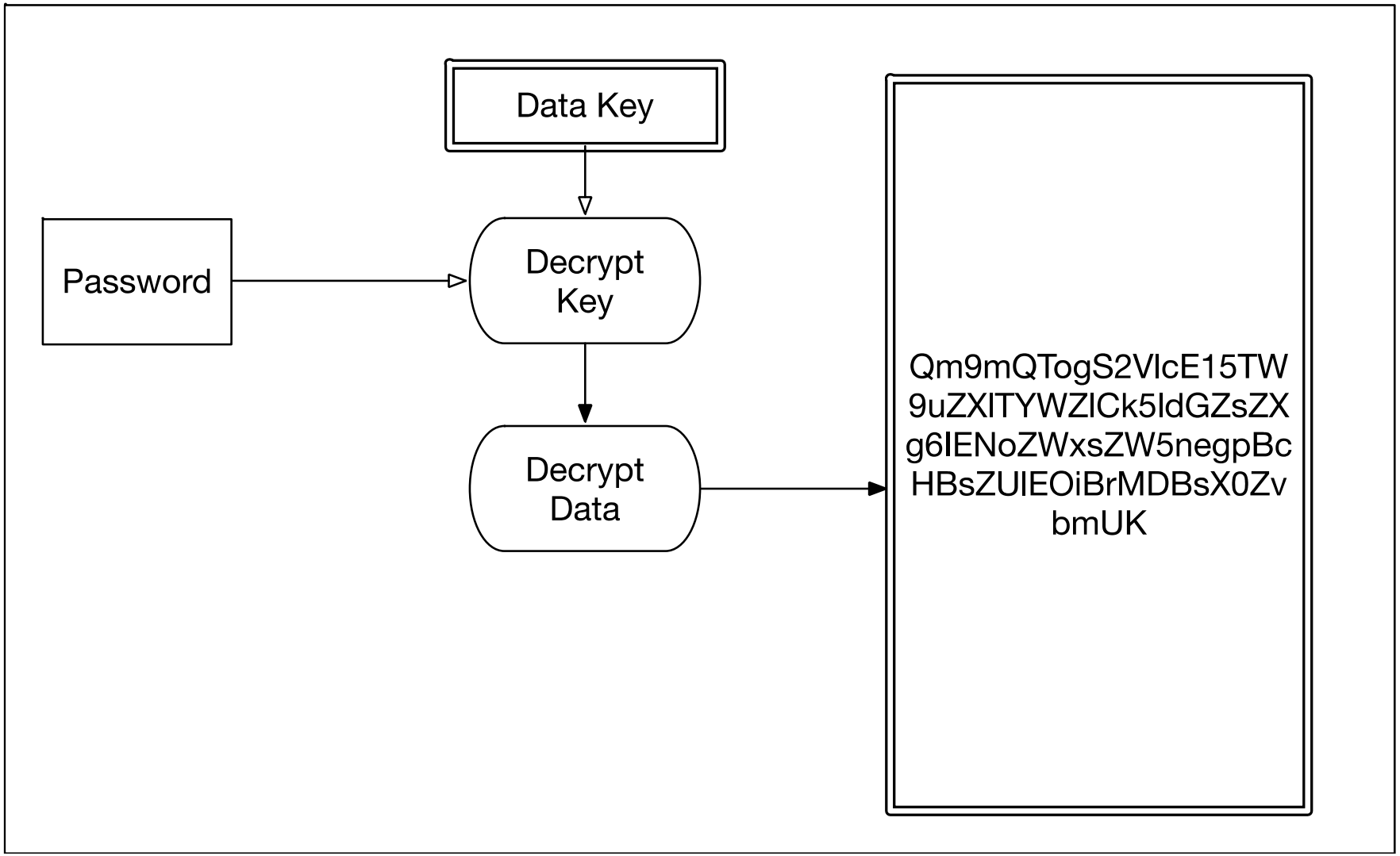
Passcode as a gatekeeper



Encrypt the data



Encrypt the key



Passwords make bad keys

- Just like with password brute forcing
  - Pick a password
  - Decrypt
  - Did it work? No, try another one
- Entropy – measures how “big” the password is
  - “password” – 8 lowercase letters – not a lot of entropy
  - Measured in bits…it’s about 38 bits
- Good encryption requires 128-256 bits
- How do we “stretch” a password into a strong key?

Hash Function

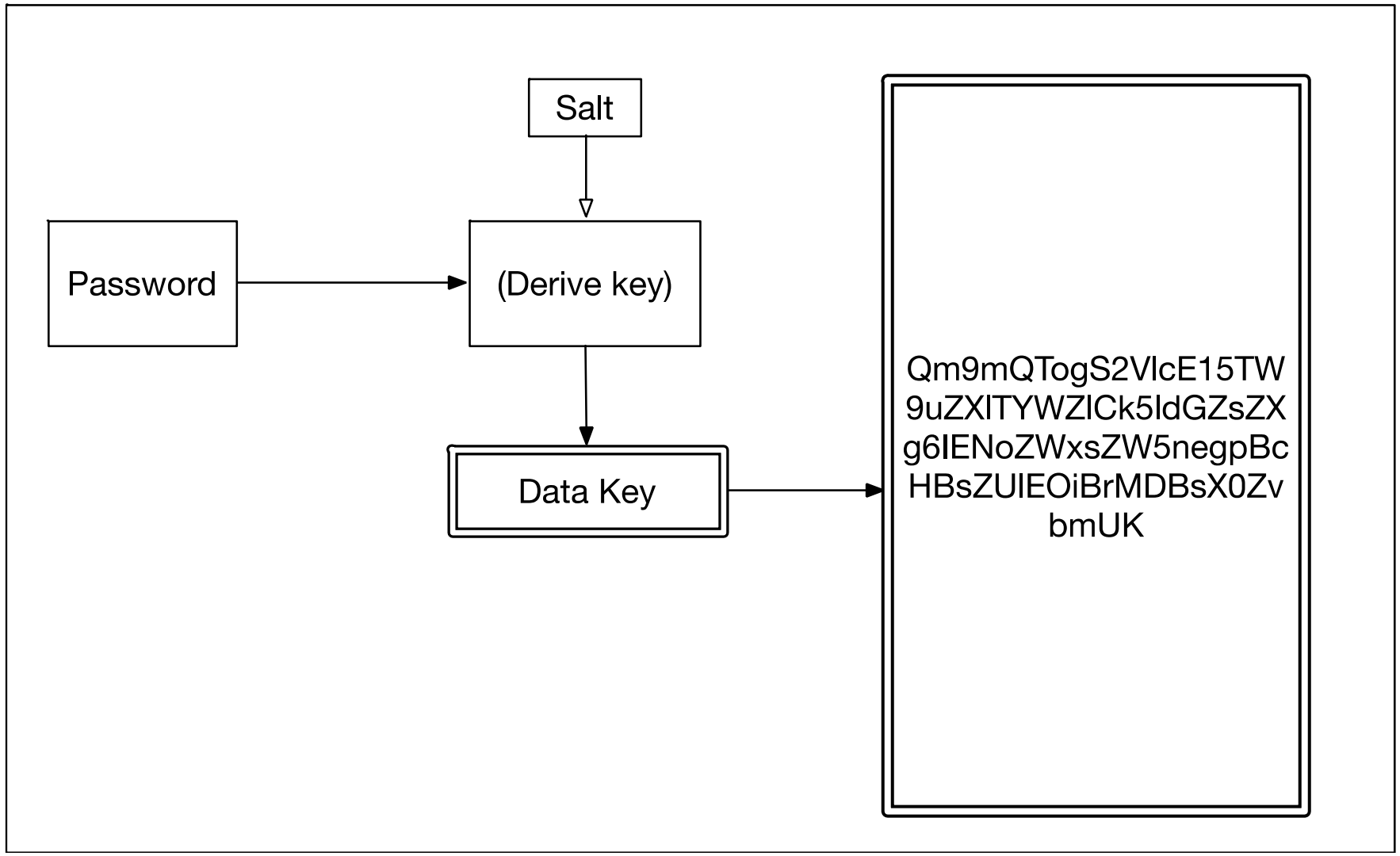
- Converts arbitrary input into a fixed string of bits
  - Random – output totally unlike input
  - Consistent – same input always generates the same output
  - Irreversible – impossible to take hash and go back to original
  - Divergent – hashes of similar texts should vary widely

First Hash	`password0`	`305e4f55ce823e111a46a9d500bcb86c`
Second Hash	`password1`	`7c6a180b36896a0a8c02787eeafb0e4c`
Common Letters	`password.`	`.........8..................c`

Avoiding Duplicate Keys

- Hashes are consistent
  - Two users with the same password
  - Both will have the same key
- Add a “salt”
  - Random string added to the password
  - Not secret – stored alongside the hash or key
  - (needed to regenerate the same result later)

Now what does it look like?



What about server logins?

- Password cracking is still a risk
- Compromised server (external or insider)
  - Crack passwords
  - Decrypt vaults
  - PROFIT!
- So make it 2-factor! (duh!)
  - Can’t use a dynamic token
  - Because every 30 seconds the key changes
  - (also duh)

Introducing the Secret Key

- Attackers need both Master Password and Secret Key
- Looks like this:
  - A3-ASWWYB-798JRY-LJVD4-23DC2-86TVM-H43EB
- Provides just under 129 bits of entropy

How to mix it in?

- Literally “add” the values together
  - Simple and reasonably fast
  - Could end up with a result longer than you need

```
  F0B79179
+ 83233D84
----------
  173DACEFD
```

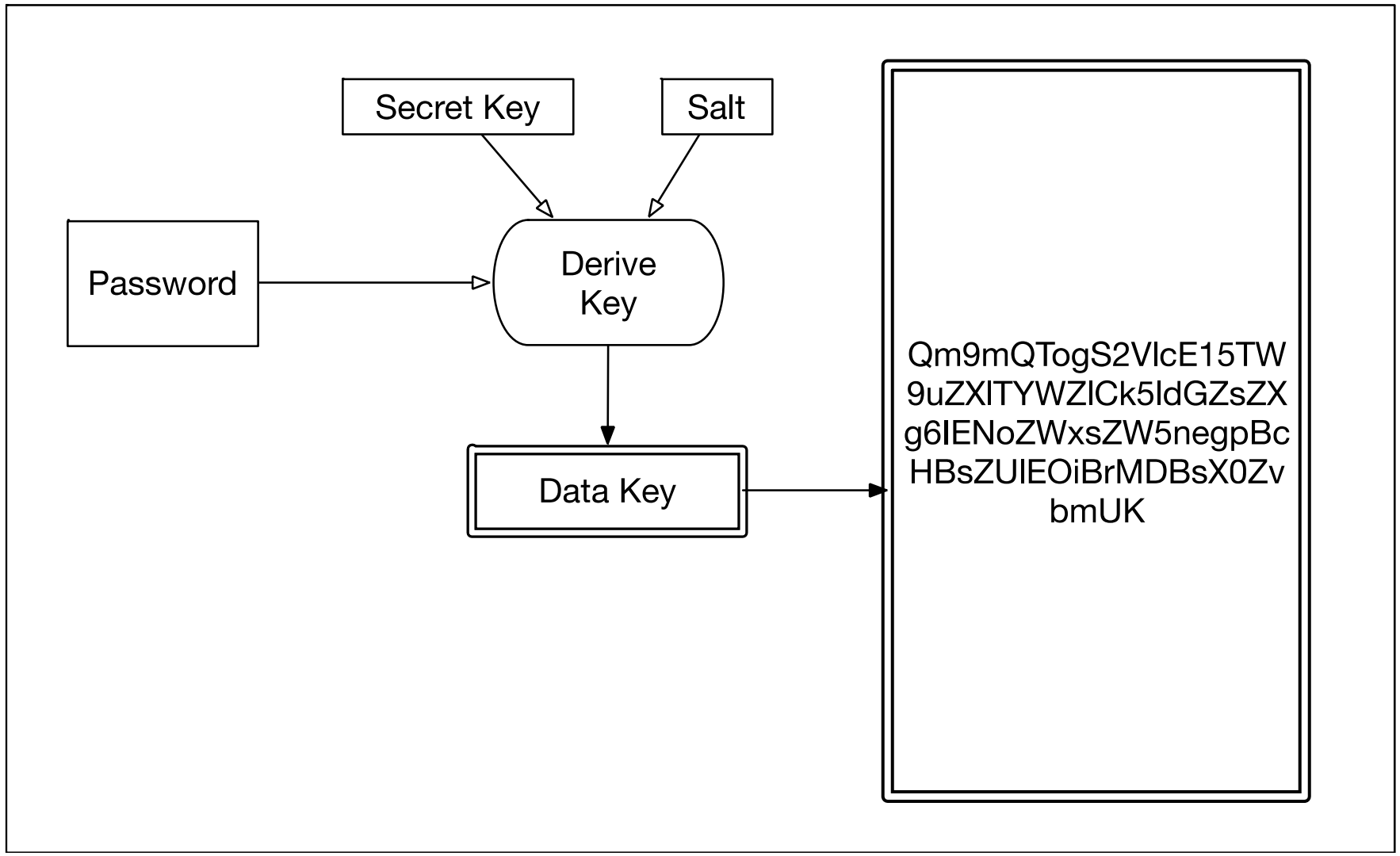
- String them together one after the other
  - Super fast
  - Makes the key much longer than you need

```
F0B7917983233D84
```

- Can also use a logical XOR operation
  - Super fast
  - Leaves the key length exactly the same as before

```
  F0B79179
^ 83233D84
----------
  7394ACFD
```

And now…

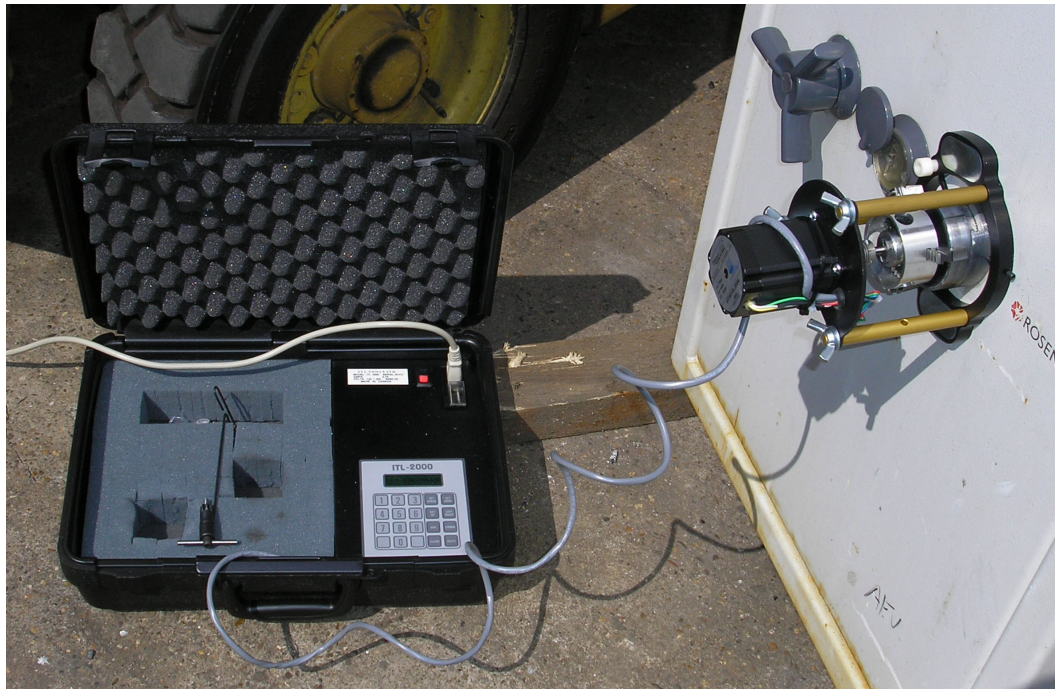

Secret Key
Salt
Password
Derive Key
Data Key
Qm9mQTogS2VlcE15TW
9uZXlTYWZlCk5ldGZsZX
g6IENoZWxsZW5negpBc
HBsZUlEOiBrMDBsX0Zv
bmUK

None of this is good enough

- We’ve gone from a short password
- To a 256-bit key

- This is great!

- We can still guess the password



How do you attack it?

- Have the password and salt, but not the secret key
  - Can derive first result
  - But must brute force the secret key
- Have the secret key and salt, but not password
  - Brute force password
- But won’t that take a long time?
  - Maybe not. The key derivation is actually incredibly fast.
- We beat this by slowing it down. A lot.
  - Repeating the process 100,000 times is (currently) a good start
- Also recommend a strong password

Strong password? How strong?

- Agilebits' recommendation is 4 word passphrase
- Currently hosting a password cracking challenge
  - 3 words (selected from list of 18,000+)
  - Running for six months (since May 3)
  - ***Only one*** of the challenges have been cracked so far
- 4 or 5 words should be more than sufficient (for now)

splendor excel rarefy

update-clown-squid-bedpost

glassy ubiquity absence

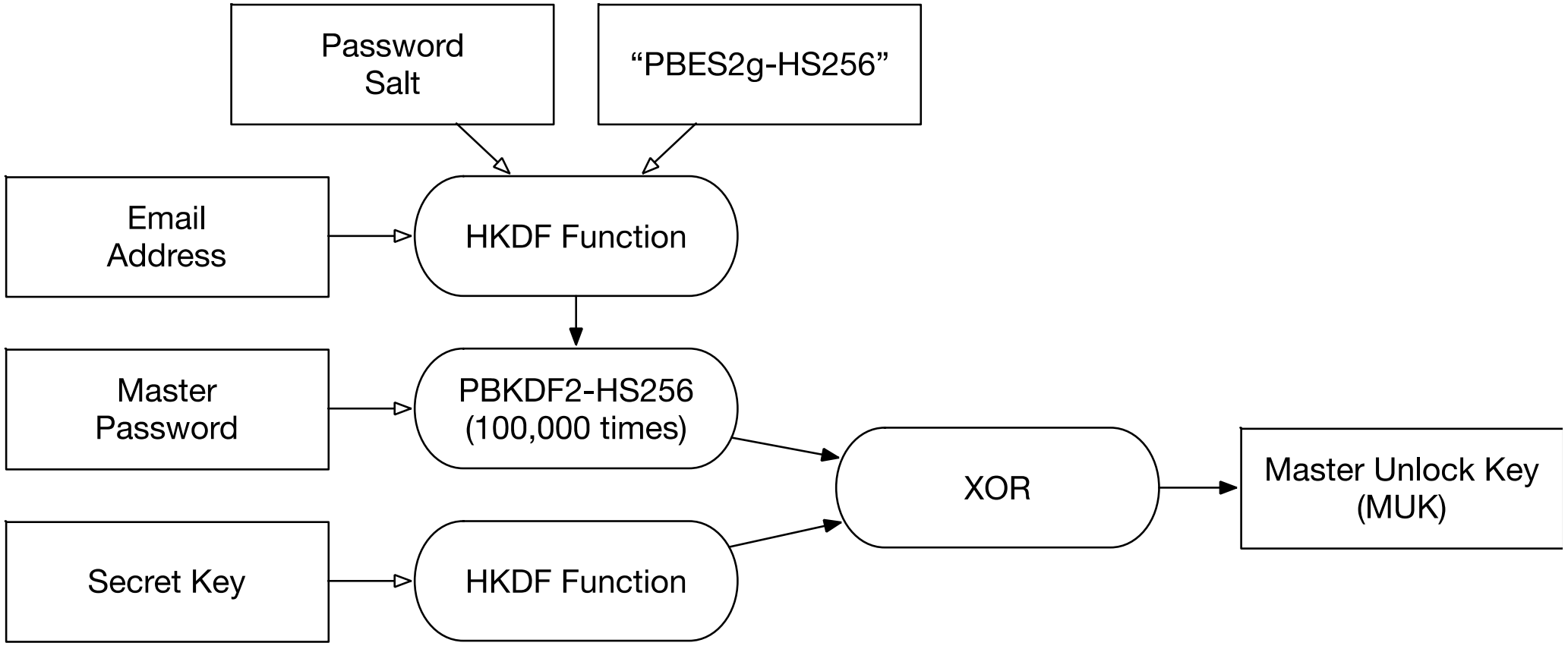
Pull it all together…

Combine everything

- So far we have:
  - Master Password
  - Salt
  - Secret Key
  - Email
- Mix them all together
- Produce final unlocking key

- Called “Two-Secret Key Derivation"

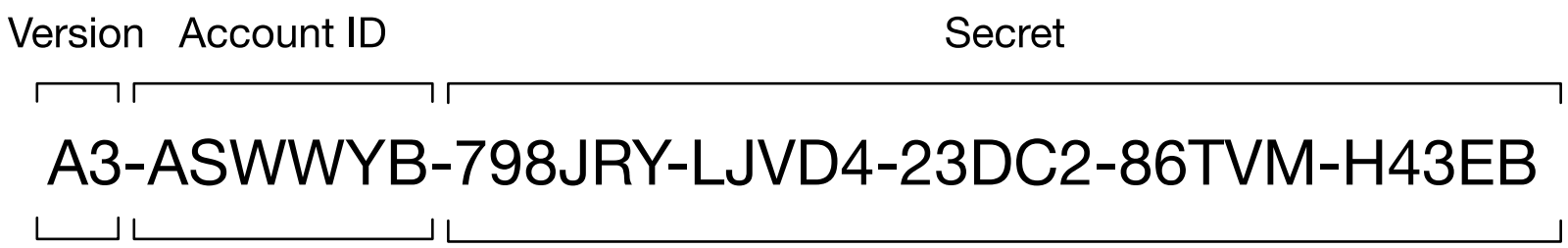
Two-Secret Key Derivation (2SKD) Process



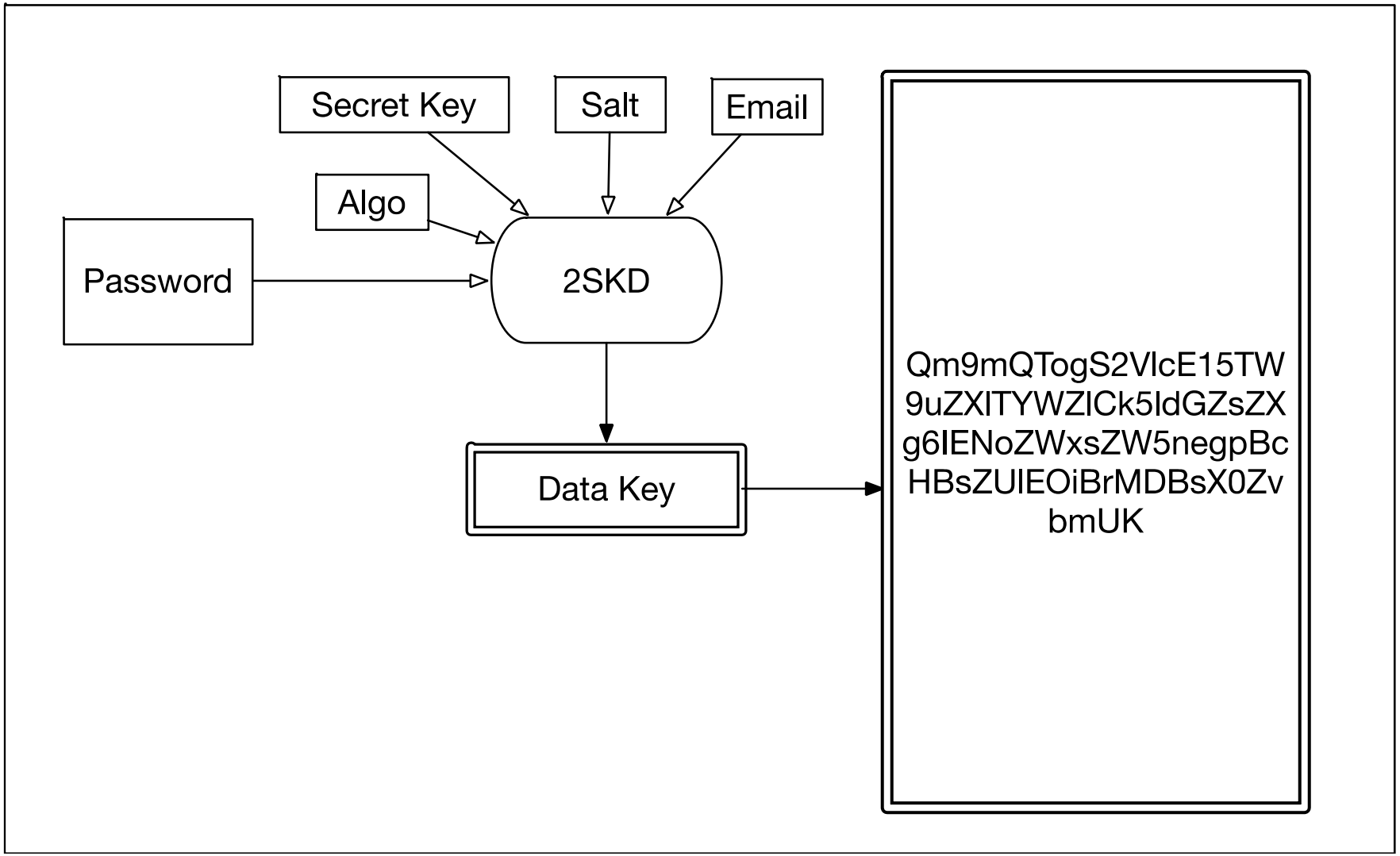
HKDF Step

- HMAC Key Derivation Function – RFC 5869
- Takes three parameters – Key, Salt, Info
- Different data used for each element of 2SKD

Element	Key	Salt	Info
PBKDF2 Salt	Password Salt	User Email	“PBES2g-HS256”
Secret Key mix-in	Secret	Account ID	Version



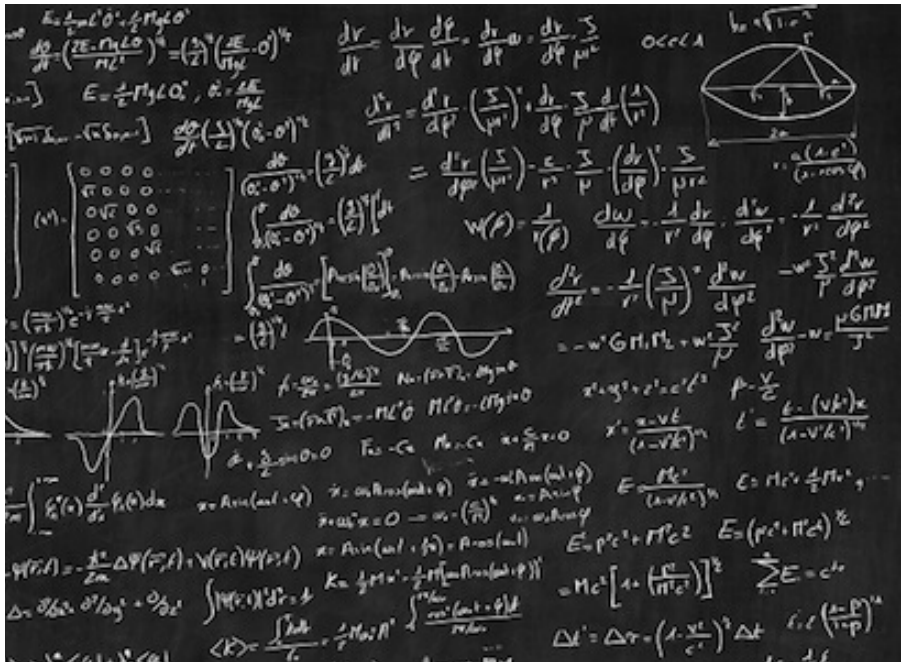
So our system now looks like…



What about logging into the server?

- Don’t we need to send the password to the server?
- Which puts it at risk from…all kinds of attackers?

Relax, we've got math!



- Secure Remote Password protocol:
  - Five specialized functions
  - Password: X
  - Verifier: $V = f_0(X)$
- Client creates the Verifier at setup
  - And sends it to the server
- Verifier is easy to compute, but difficult to reverse
- Only the client ever sees your password

Secure Remote Password Protocol

(Somewhat Simplified)

Client	Server
A = f_1(*password*, random data 1)	B = f_2(*verifier*, random data 2)
(send A to server)	*(send B to client)*
$K_1 = f_3$(A, B, data_1)	$K_2 = f_4$(A, B, data_2)
Does $K_1 = K_2$?	

- If I have the verifier:
  - Can't reverse to password
  - Can't calculate K_2 because I can't build A without the password

The actual math (in case you're curious)

a & b – random numbers at client and server

g, k – special constants known to both

x – user's password (expressed as a number)

v – verifier for the password, stored on the server

$v = g^x$

Client	Server
$A = g^a$ - send to server	$B = kv + g^b$ – send to client
$u = hash(A, B)$	$u = hash(A, B)$
$(B - kg^x)^{(a + ux)}$	$(Av^u)^b$
$(kv + g^b - kg^x)^{(a + ux)}$	$(g^a v^u)^b$
$(kg^x + g^b - kg^x)^{(a+ux)}$	$(g^a(g^x)^u)^b$
$(g^b)^{(a+ux)}$	$(g^{(a+ux)})^b$
$K_1 = hash((g^b)^{(a+ux)})$	$K_2 = hash(g^{(a+ux)})^b$

Again, though, we want a strong password

- We have the Master Password
- We also have the Secret Key
- Makes a REALLY STRONG password.
  - 256-bits
  - Equivalent to 39 characters of A-Za-z0-9special (96 letter alphabet)
  - 2iZmlarN|<+jup$k8f2BJ\'H`7#;O.ncTeH!pOJv

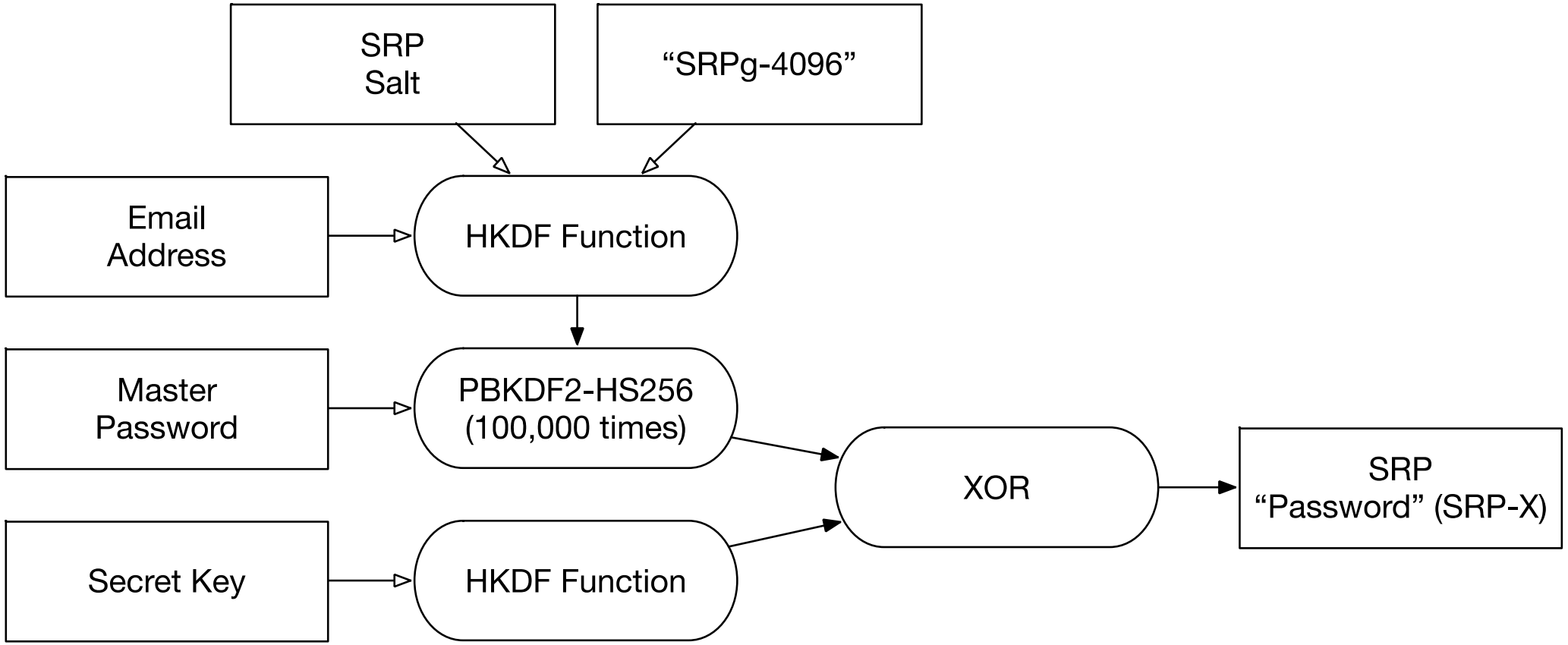
- So why not just re-use the MUK?

We shouldn’t reuse passwords!

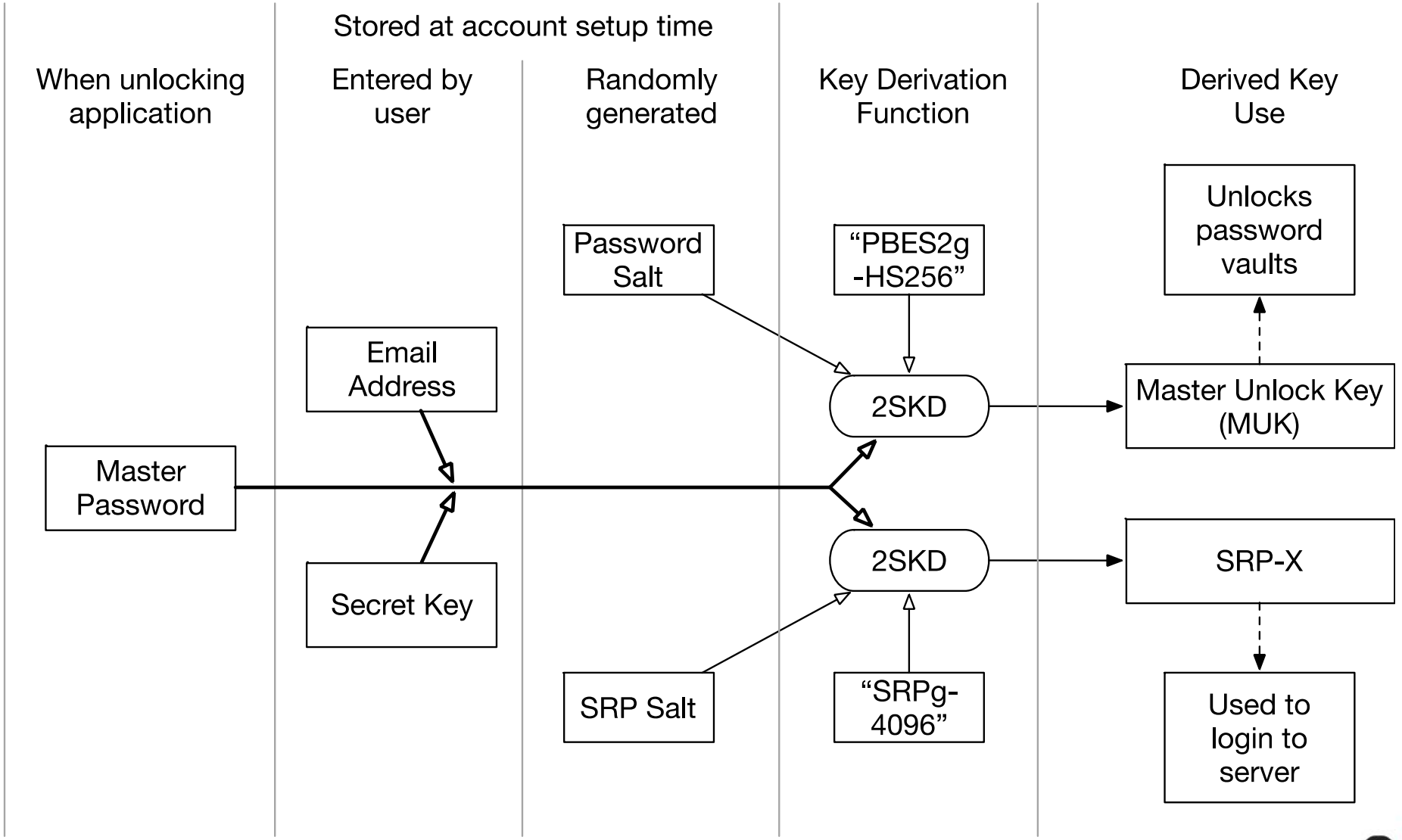
- Good catch.

- Do exactly the same thing as the Master Unlock Key
- But change the salt & a couple other parameters
- Just as strong, but safely different from encryption key

SRP-X Derivation



Summary:


Stored at account setup time
When unlocking application
Entered by user
Randomly generated
Key Derivation Function
Derived Key Use
Password Salt
"PBES2g -HS256"
Unlocks password vaults
Email Address
2SKD
Master Unlock Key (MUK)
Master Password
2SKD
SRP-X
Secret Key
SRP Salt
"SRPg-4096"
Used to login to server

What can SRP get you? (Web Features)

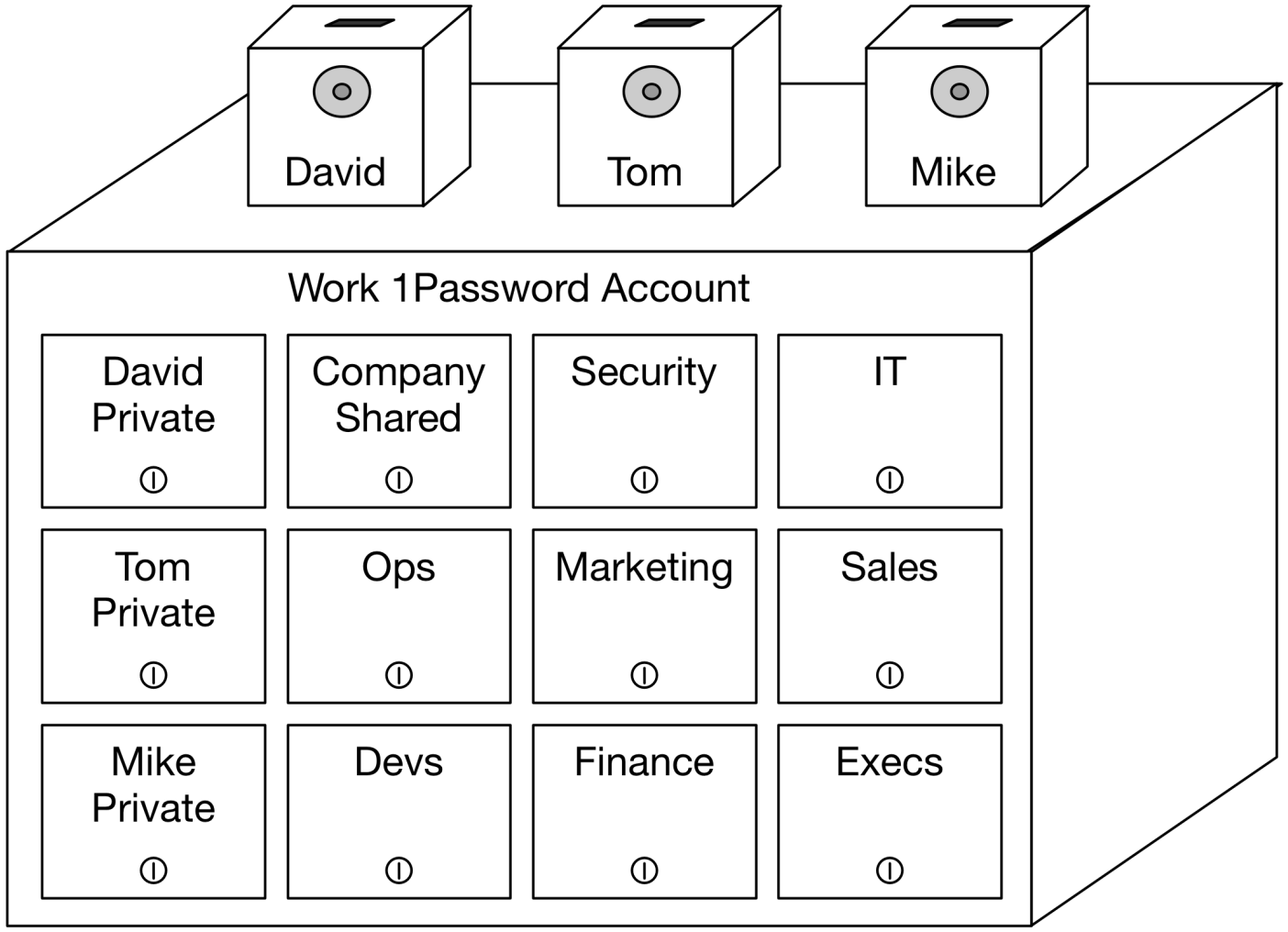
- Several features only in web interface
  - Manage vault access
  - Change account password
  - Billing, etc.
- SRP key can't be used to decrypt vault
  - Even when logged in, 1Password servers can’t read your data
- Vaults can only be decrypted at the client
  - Web client builds MUK locally
  - Decrypts encrypted items inside the browser

Organizing and Sharing Passwords

What if I want to share passwords?

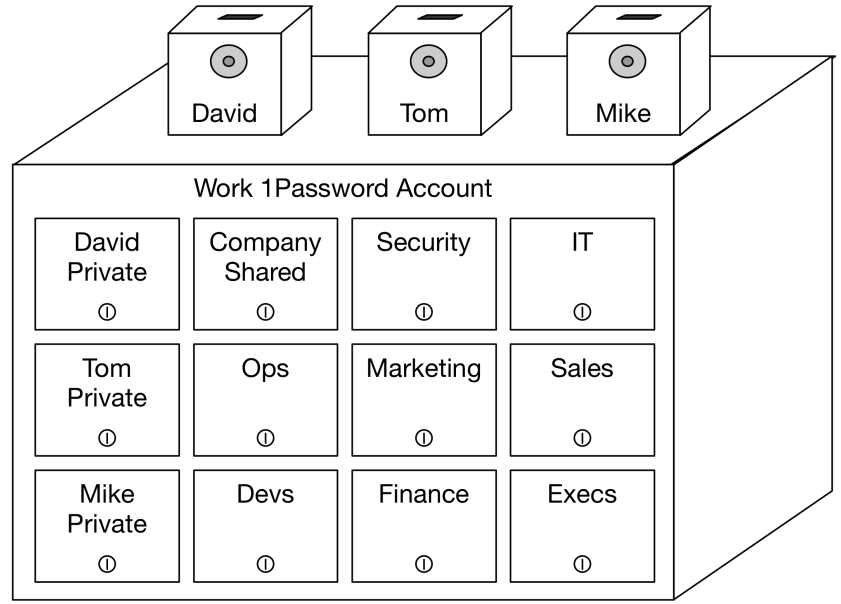
- Probably shouldn’t just share your whole password list
- Best to create different lists
  - One for yourself
  - One for your main work team
  - One for the company as a whole
  - …etc.

Imagine a wall full of small drawers…



Imagine a wall full of small drawers

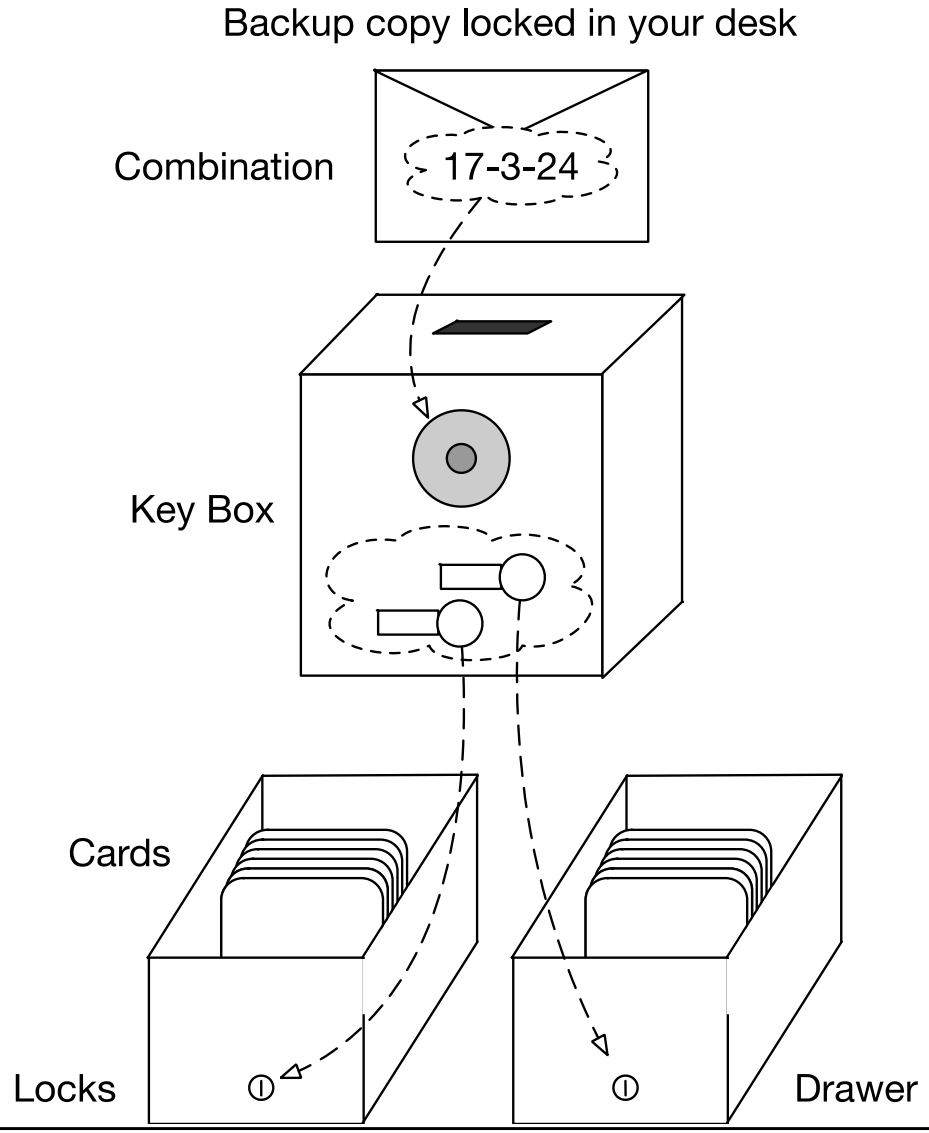
- Passwords stored in locked drawers
- People given drawer keys based on need
- Store their keys in personal key boxes
- A combination unlocks the box
- The combination is sealed in an envelope
  - (as a backup)
- And the envelopes are locked in your desk



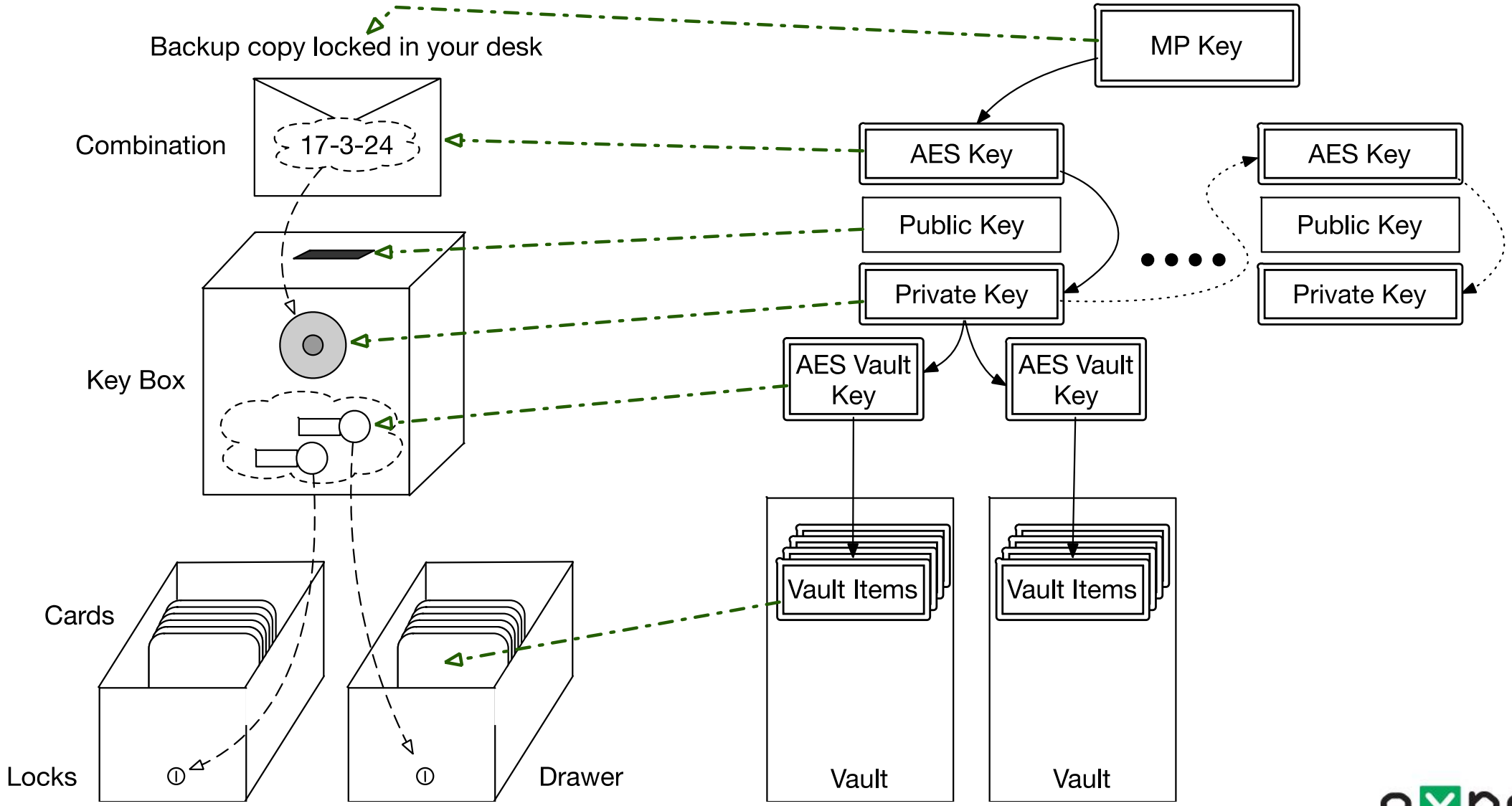
If I create a new vault?

- I make copies of the key for everyone who needs it
- Go to each person’s desk, and find their box
- Then slip the key through a slot in the top
- (I can add it but can't open the box to get other keys)

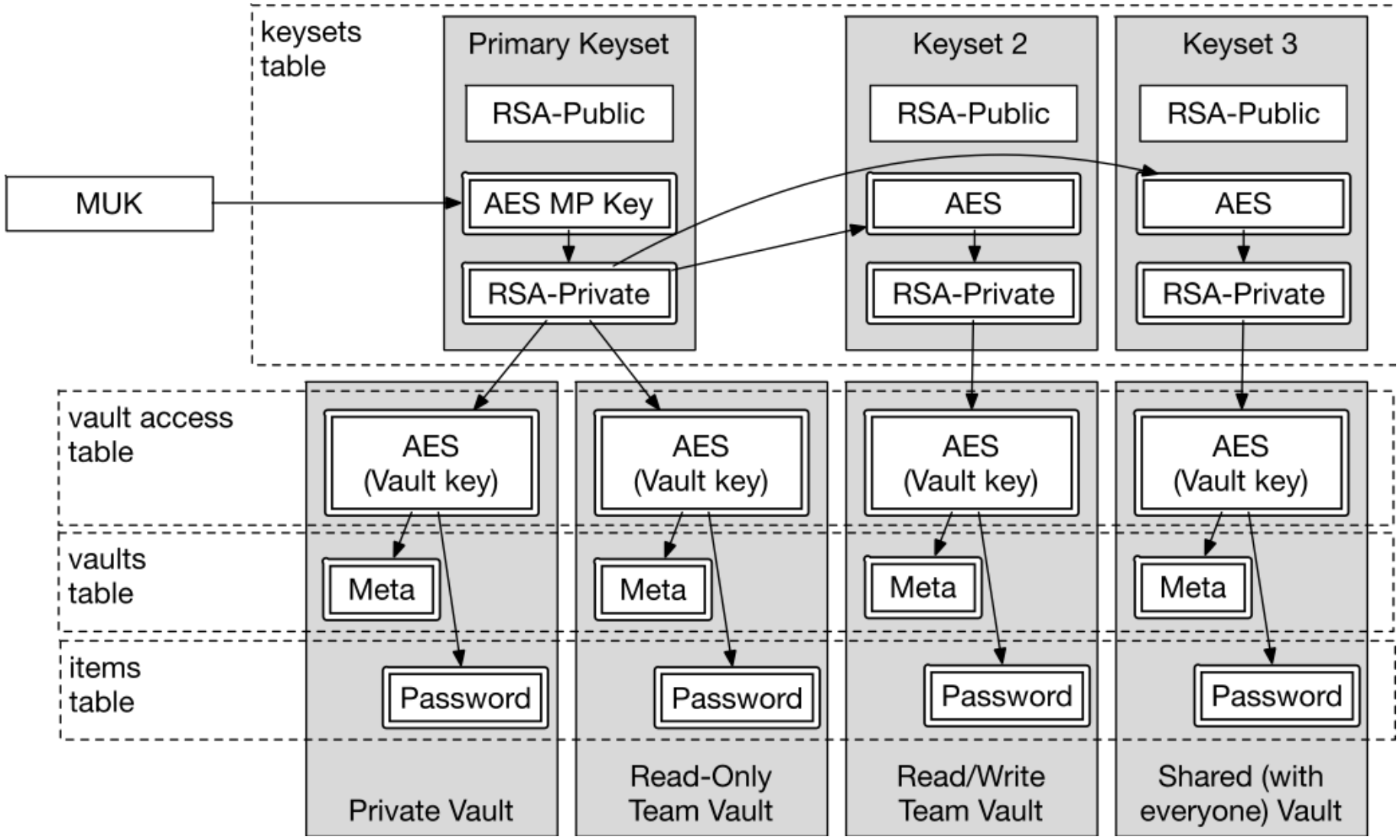
So it works like this:



But in reality, it's this:



Which ultimately looks like this…



What if I forget my password?

- Administrators are part of a "recovery group”
  - Or “Organizer” for 1Password Family account
- They have a keybox with all vault keys in it
  - Only exists on the server
  - Each item encrypted with Recovery Group public key
- When a user resets their account
  - The admin gets a “Finalize recovery” prompt
  - Their client:
    - Retrieves and decrypts the user’s encrypted keys
    - Re-encrypts with the user’s public key
    - Sends them to the user
    - Deletes local copy

Unlocking Multiple Accounts

How does it handle multiple accounts?

- One password (ha!)
- Primary account unlocks others
- Delete the primary account, and the next one becomes primary

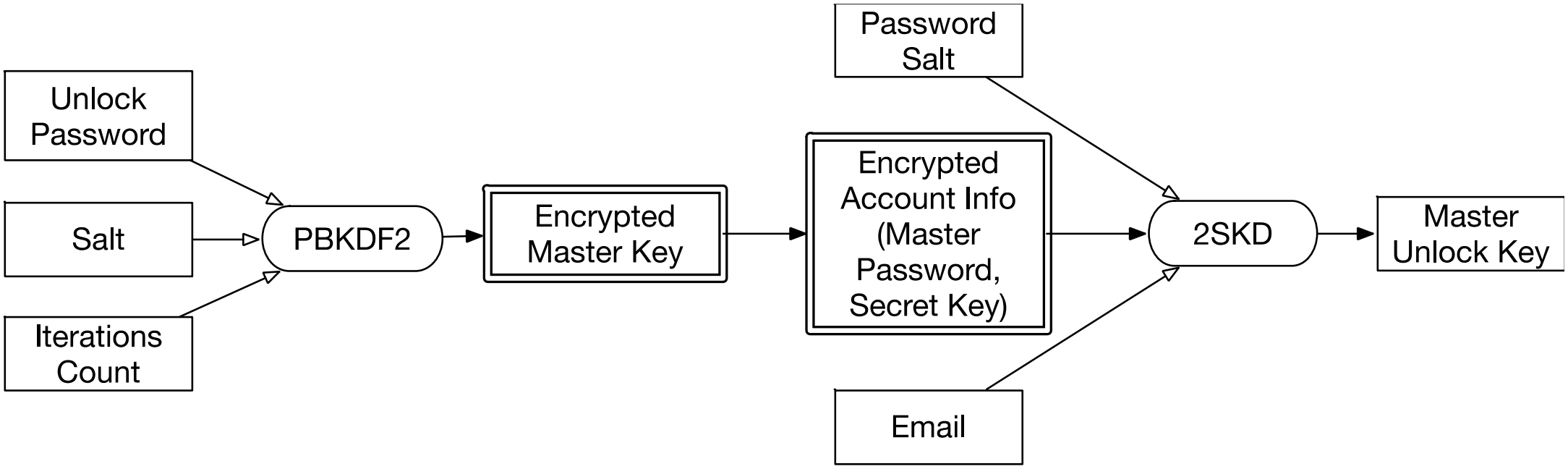
macOS:

- 2SKD derives Master Unlock Key
- MUK decrypts primary account vaults
- MUK decrypts account data for secondary accounts
  - Reveals MUK and SRP-X for each account
  - Each individual MUK decrypts the vaults for its own account

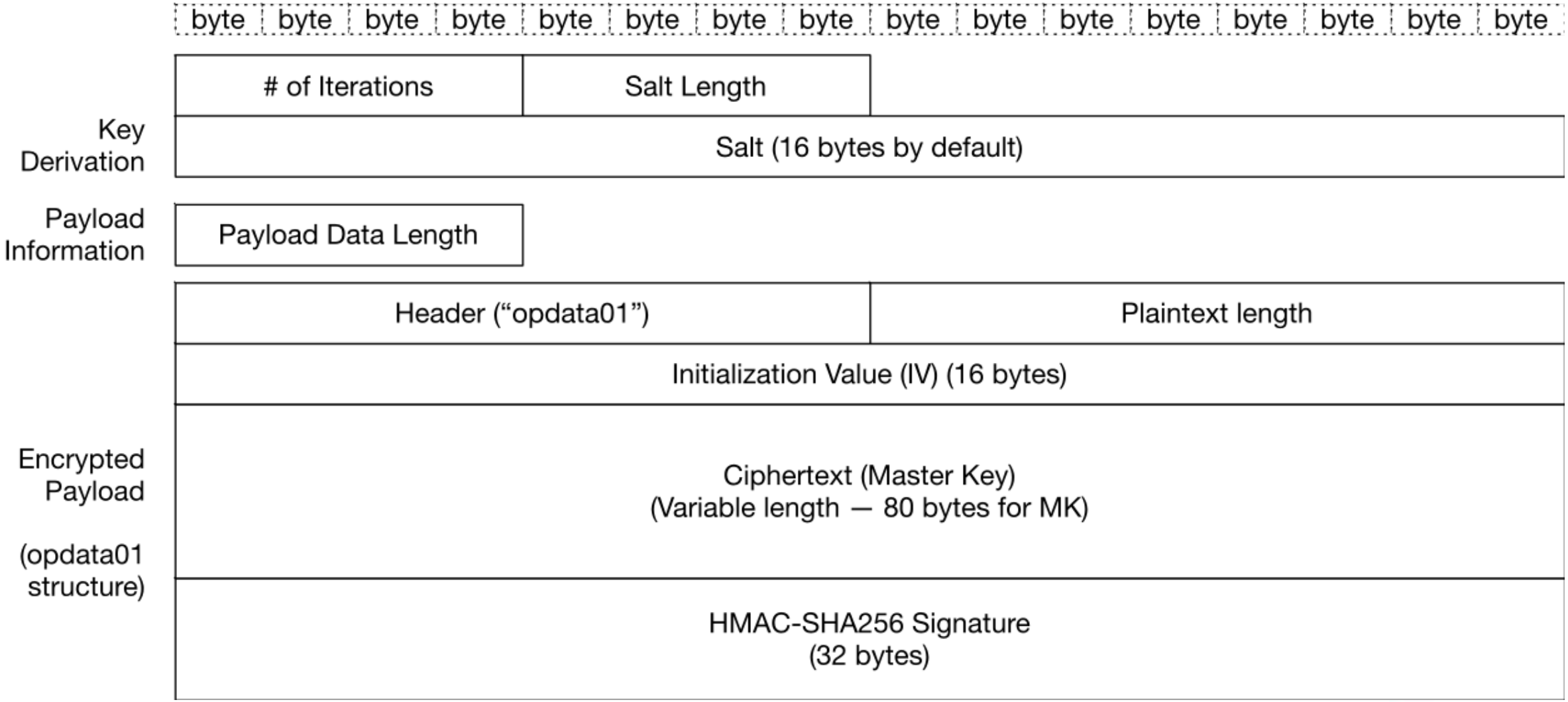
Windows:

- Primary account password decrypts EncryptedMasterKey
- That decrypts account information for each account
  - Master Password (in plaintext), Secret Key, salt, email, etc.
  - Can then derive MUK and SRP for each account
  - …And then decrypts vaults

Encrypted Master Key



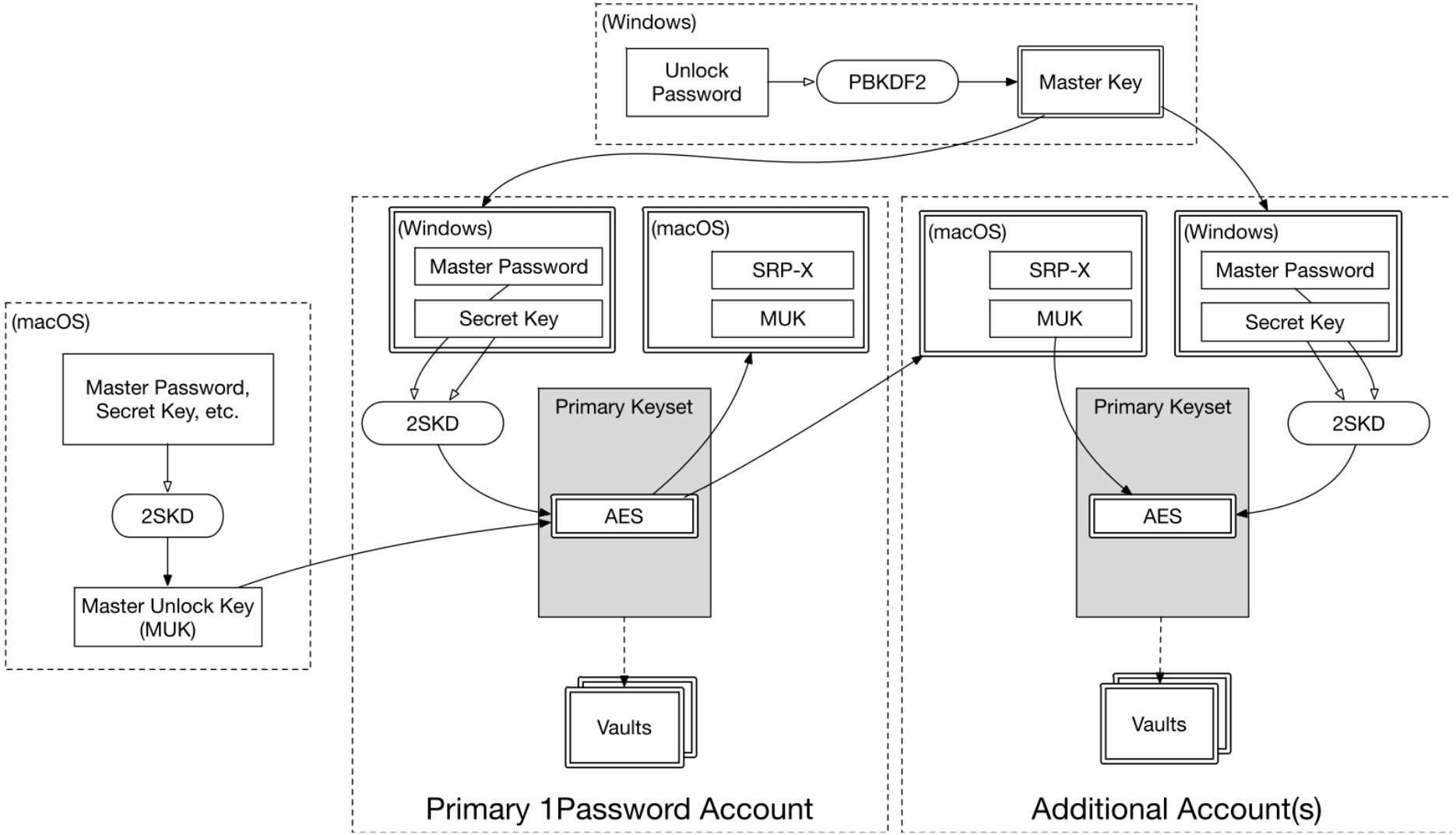
Encrypted Master Key (EMK) Structure



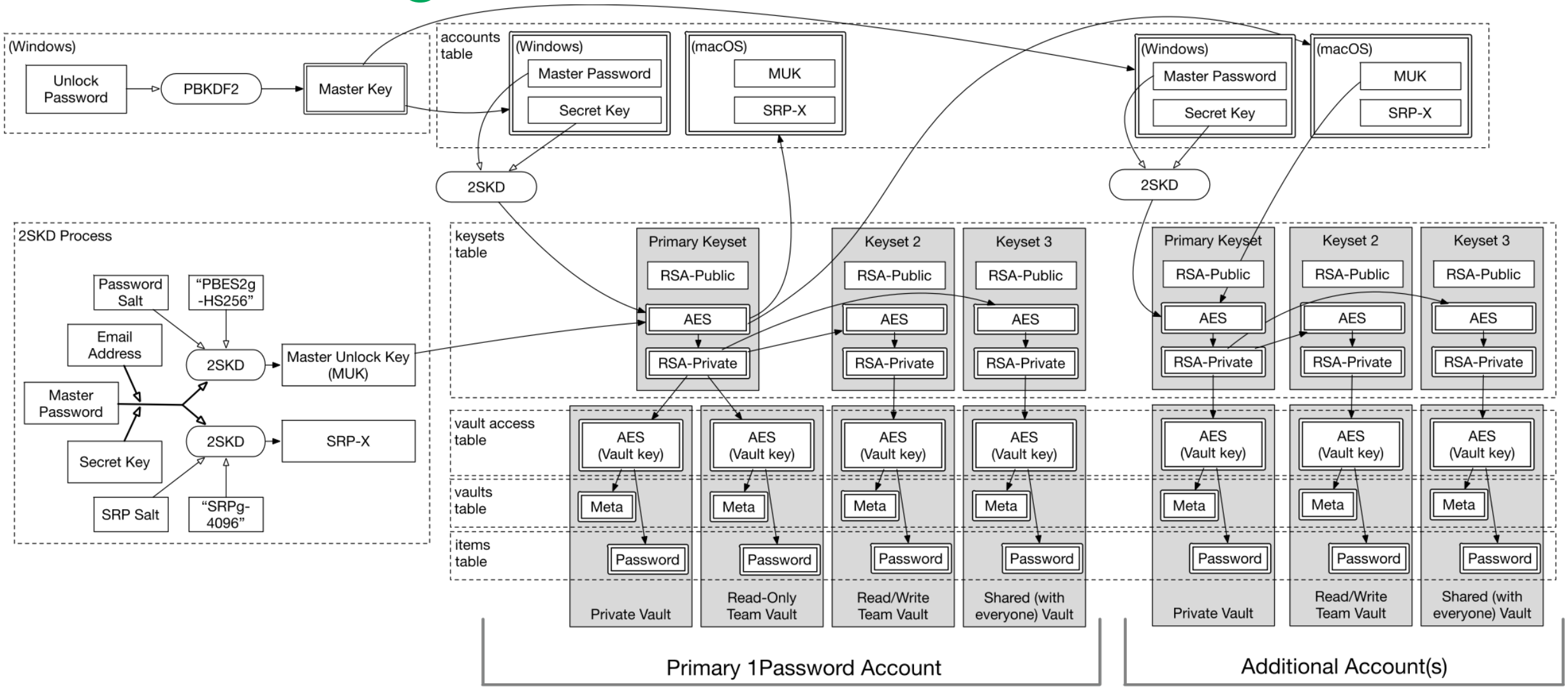
Decrypting EMK

- Derive decryption key from unlock password
  - PBKDF2-SHA512 with given salt
  - Iteration count varies depending on computer speed
    - Updates with each unlock, if necessary
    - Target is to require 1 second to compute
  - Decrypts payload (using AES-CBC)
- Payload contains:
  - Padding (16 bytes), Master Key, and HMAC Signature (32 bytes each)
  - Signature verifies Master Key wasn't tampered with
- Master key decrypts account data

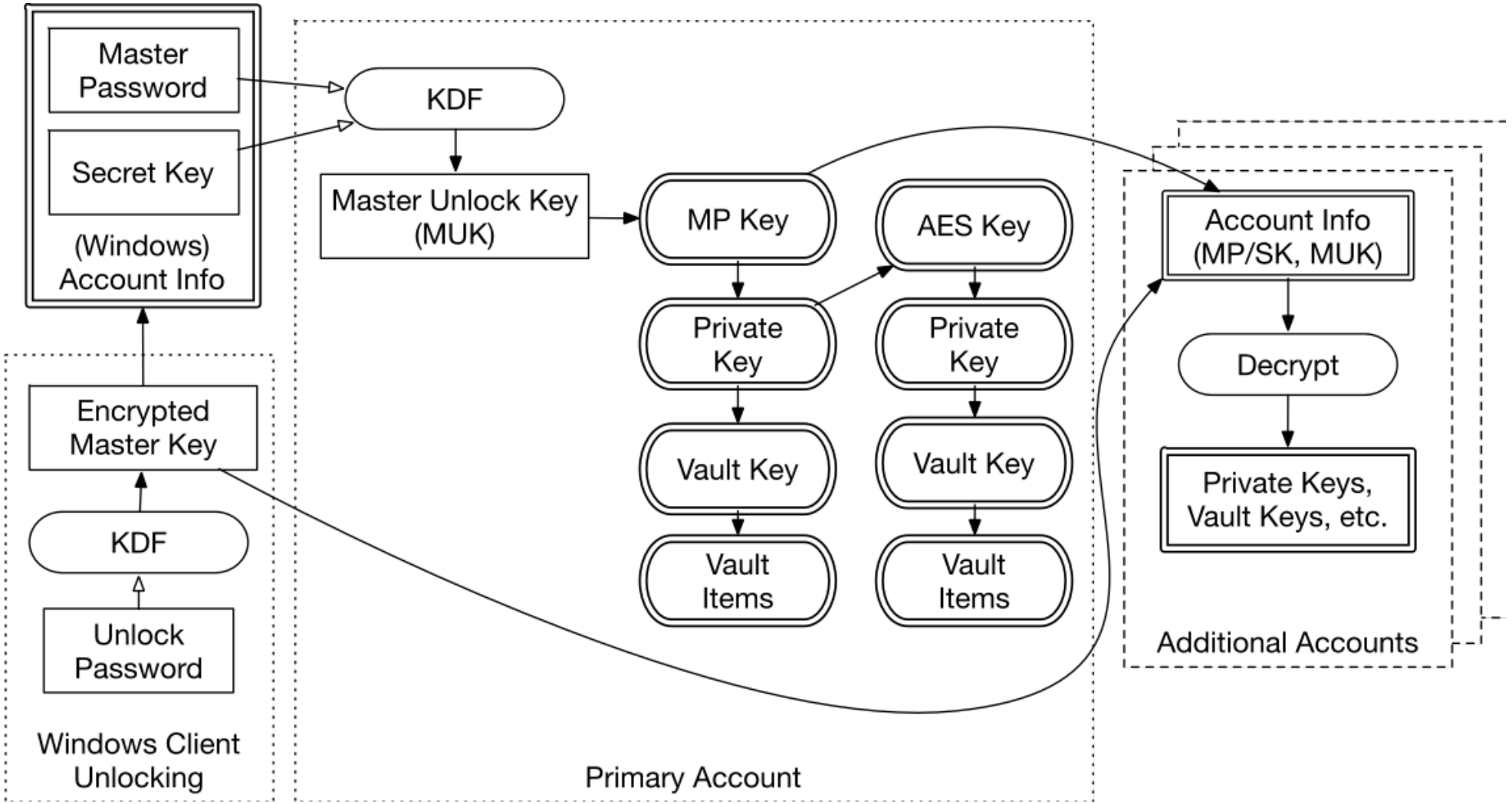
Multiple Accounts, Overview



Which brings us back here…



(better yet, a simplified view)



Where is everything kept?

Client	Vaults	Secret Key	Master Password	Unlock Password
macOS	~/Library	System keychain	User's memory	n/a
Windows	x	Encrypted in vault	Encrypted in vault	User's memory
Web Browser	(n/a – fetched on demand)	Browser local storage (obfuscated)	User's memory	n/a

Wrapping Up

Internal Details

- Secret Key
  - Resilience to password breaches at server
- Master Password
  - Unlocks EVERYTHING
- SRP to server
  - Don’t have to send actual password
- Derived SRP-X
  - And the “password” the server uses is a 256-bit key
- Shared Vaults, Recovery
  - Can share keys w/out knowing recipient’s private keys or password

What'd I miss?

- Watchtower
- Travel mode
- 2FA
- Journal / Backup
- Mobile, Browser, CLI clients
- Browser extensions
- Obfuscated passwords via SMS
- Touch ID

Thanks!

- AgileBits - for being so transparent and open
- AgileBits Engineers - for answering my never-ending stream of questions on the support forums
- Expel - for letting me turn a simple question into this talk
  - (and an absurdly-long set of blog posts)

Further Reading

- Blog:
  - DarthNull.org/series/1password
  - Multi-part series
  - Extensive technical detail, examples, additional topics

- Github
  - GitHub.com/dschuetz/1password
  - Simple example scripts
  - Rough library
  - Test data (and tool to generate it)