Usable Crypto: Introducing miniLock

Nadim Kobeissi
HOPE X, NYC, 2014


2012

"Browsers are an environment that is hostile to cryptography"

* Malleability of the JavaScript runtime.

* The lack of low-level (system-level) programming access.

* DOM-style vulnerabilities (XSS)

"Code Delivery is a Chicken-Egg Problem"

* What prevents web app code from being intercepted and modified by a "man in the middle"?

* Fine, why not use SSL?

Quote from popular anti-JS crypto article

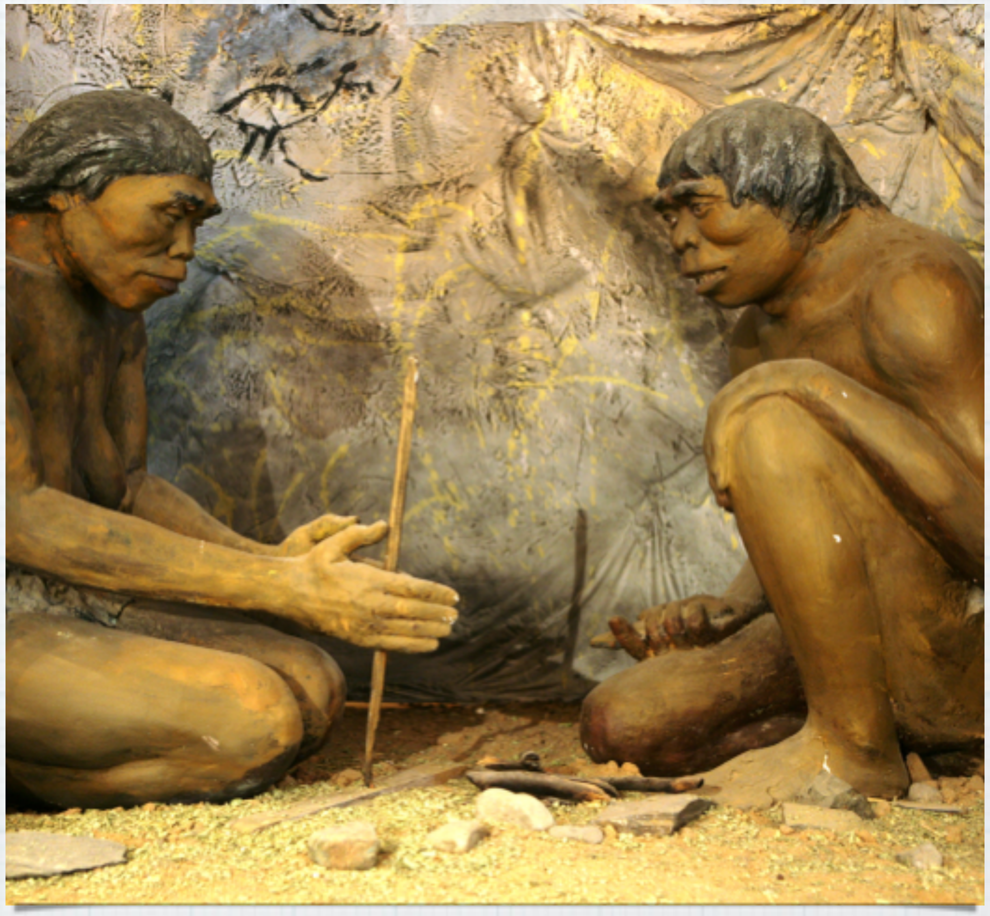
"You can [use SSL]. It's harder than it sounds, but you [can] safely transmit Javascript crypto to a browser using SSL. The problem is, having established a secure channel with SSL, you no longer need Javascript cryptography; you have "real" cryptography."

What the author ignores:
Unlike SSL, JavaScript cryptography
protects data from server access.

Also claims people are using JS Crypto to get around deploying SSL (???)

No One Saw the Value of JavaScript Cryptography

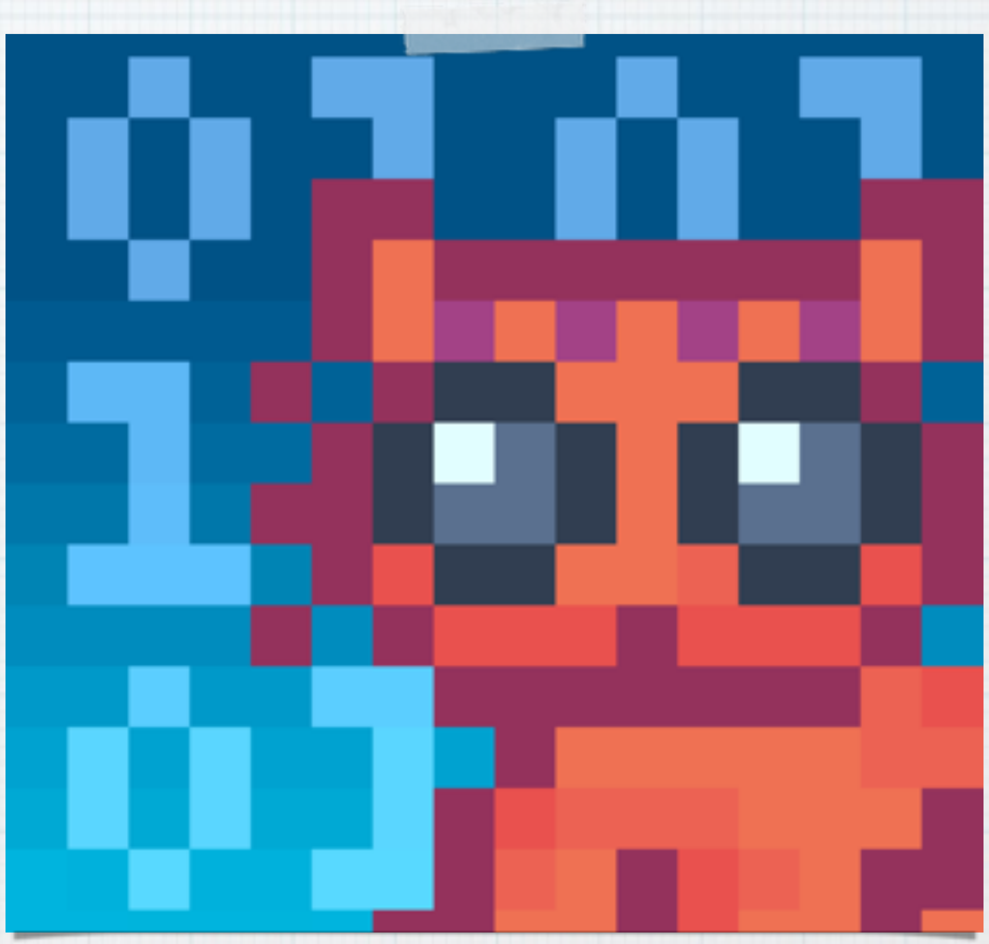
- JS shifting from *language of the web* to *language of everything*
- Making JS crypto real means making crypto work in the world's most accessible language
- Huge privacy/security gains in a usable environment




2013

Cryptocat: Encrypted Chat in the Browser

* Open source app with over 200,000 users.
* Goal: make encrypted chat accessible, **fun**, and easy to use.
* Accessible no matter your background.



Basic Needs

* Secure cryptographic primitives (AES, SHA2, ECDH).
* Secure pseudorandom number generation.
* Secure code delivery.

Cryptographic Primitives

* Public key cryptography and digital signature algorithms depend on numbers much larger than 64-bit floating point.
* Big integers require a third-party library

Cryptographic Primitives

* Some algorithms are computationally expensive (RSA, Diffie-Hellman, DSA…)
* Web workers came to the rescue.

Cryptographic Primitives

- Multiple maturing libraries: SJCL, **Crypto-JS**, OpenPGPJS

- Crypto operations (bitwise, etc.) are surprisingly cleanly writeable in JavaScript.

One Round of AES in JavaScript and C

```
a2 = t0[a>>>24] ^ t1[b>>16 & 0xff] ^ t2[c>>8 & 0xff] ^ t3[d & 0xff] ^ key[4];
b2 = t0[b>>>24] ^ t1[c>>16 & 0xff] ^ t2[d>>8 & 0xff] ^ t3[a & 0xff] ^ key[5];
c2 = t0[c>>>24] ^ t1[d>>16 & 0xff] ^ t2[a>>8 & 0xff] ^ t3[b & 0xff] ^ key[6];
d  = t0[d>>>24] ^ t1[a>>16 & 0xff] ^ t2[b>>8 & 0xff] ^ t3[c & 0xff] ^ key[7];
```

```
t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >> 8) & 0xff] ^ Te3[s3 & 0xff] ^ rk[4];
t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >> 8) & 0xff] ^ Te3[s0 & 0xff] ^ rk[5];
t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(s0 >> 8) & 0xff] ^ Te3[s1 & 0xff] ^ rk[6];
t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(s1 >> 8) & 0xff] ^ Te3[s2 & 0xff] ^ rk[7];
```

JavaScript Cryptography: Example of a Bug

```
 multiParty.genPrivateKey = function() {
-   rand = Cryptocat.randomString(32, 0, 0, 1);
-   myPrivateKey = BigInt.str2bigInt(rand, 10);
+   var rand = Cryptocat.randomString(64, 0, 0, 0, 1);
+   myPrivateKey = BigInt.str2bigInt(rand, 16);
    return myPrivateKey;
```

Thanks, weak typing.

Secure Pseudorandomness

* `Math.random()` relies on guessable entropy sources.
* `window.crypto.getRandomValues()` doesn't.

Code Delivery

* Browser apps. Chrome led a revolution, model adopted by Safari and Opera (Firefox lags behind.)
* Great features: Code signing, enhanced security, protection against XSS and in-line eval.

Code Delivery

* In some cases, signed browser apps have benefits over regular desktop apps!
* Strong separation from system level.
* Chrome: tab CPU sandboxing.


2014
BROS.

W3C Web Crypto API

* Native cryptographic primitives!
* A solid chance to mitigate side-channel attacks such as timing attacks.
* (Disclosure: I'm on that team)

W3C Web Crypto API

* Missing features:
* Modern algorithms (Curve25519)
* Key storage API

Sudden Acceptance of JS Cryptography

* Google publishes browser extension for GPG, own JS cryptography library
* Microsoft publishes Microsoft JavaScript Cryptography Library
* Thai Duong: “Why JavaScript cryptography is useful”

Matthew Green
@matthew_d_green

If you have issues with Javascript crypto, wait til you see C crypto.

2014-06-23, 10:12 AM

77 RETWEETS | 73 FAVORITES

Stéphane Bortzmeyer
@bortzmeyer

Web crypto API (native crypto for JavaScript) is at "
working draft" stage. Comment now! #RMLL2014

2014-07-09, 5:35 AM

4 RETWEETS | 3 FAVORITES

Thomas Noe
@teh2mas

Came back to twitter to follow crypto currency trends. Found lots of talk about JavaScript. When did twitter get so interesting?

2014-07-09, 2:37 PM

Kenn White
@kennwhite

If you haven't seen it, great posts by Nate Lawson & Thai Duong on javascript crypto:
vnhacker.blogspot.com/2014/06/why-ja...
rdist.root.org/2014/06/23/in-...

2014-07-08, 12:09 PM

Juliano Rizzo
@julianor

Thai Duong blog: Why Javascript Crypto is important vnhacker.blogspot.com.ar/2014/06/why-ja... btw js shift left (1<<31)>>31^1= ? #ESP

2014-06-18, 4:34 PM

9 RETWEETS | 16 FAVORITES

azet
@a_z_e_t

javascript crypto you say? good idea you say? vimeo.com/99599725

2014-07-07, 6:30 PM

2 RETWEETS | 5 FAVORITES

Ben Adida
@benadida

Still a lot of JavaScript crypto hating. Cryptographers should better understand defense in depth.

2014-06-23, 10:03 AM

Clipperz
@Clipperz

What do Stanford, Google, and Microsoft have in common? Each has released an open source Javascript crypto library. ow.ly/ylu63

2014-06-23, 6:10 PM

3 RETWEETS | 7 FAVORITES

Remaining Problems Today: Weak Typing

- I want ECMAScript to have optional strong typing.
- `var k = 5`
- `var number(k) = 5`
- Both would work, but the second one can throw a TypeError if you do `k = 5 + ‘Meow’`

What I'm introducing today

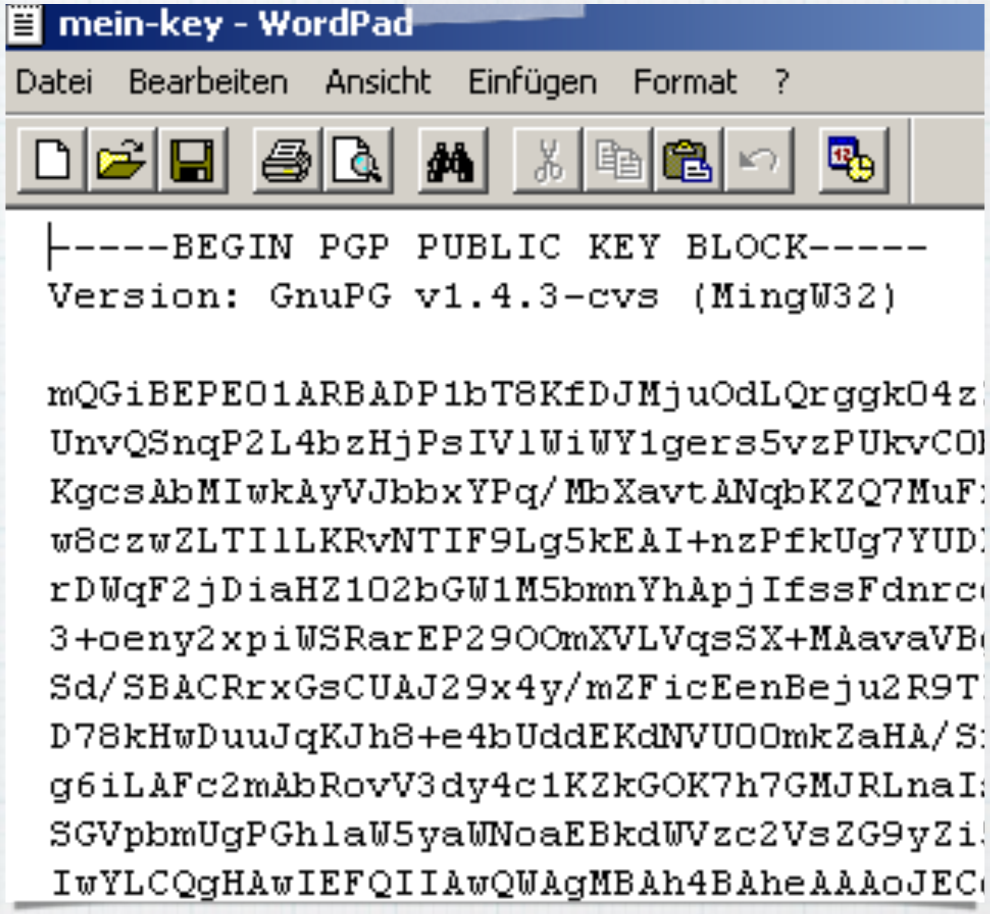
* My new usable encryption software.
* Let's innovate on file encryption and sharing.
* Let's make it universally accessible.

miniLock

File encryption software
that does more with less.

Current status for file encryption

* Main contender: PGP
* Main use case: File attachments
* Classic public key management

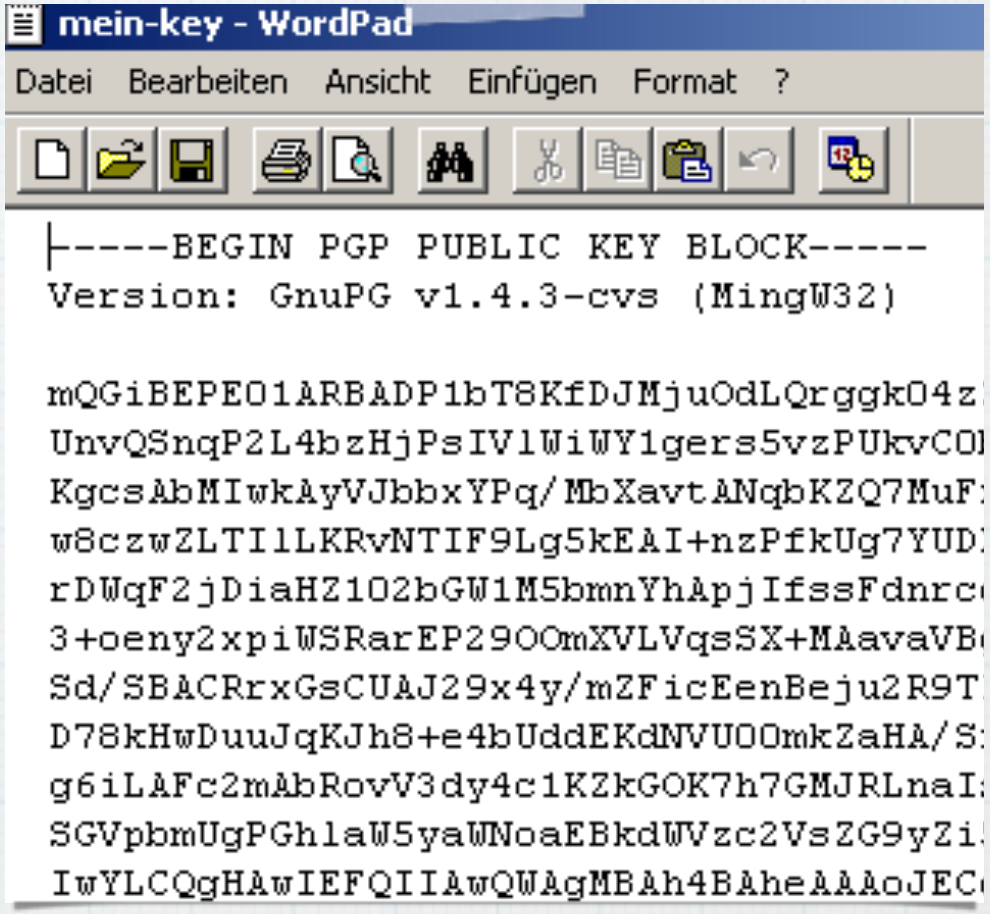


???

TrueCrypt

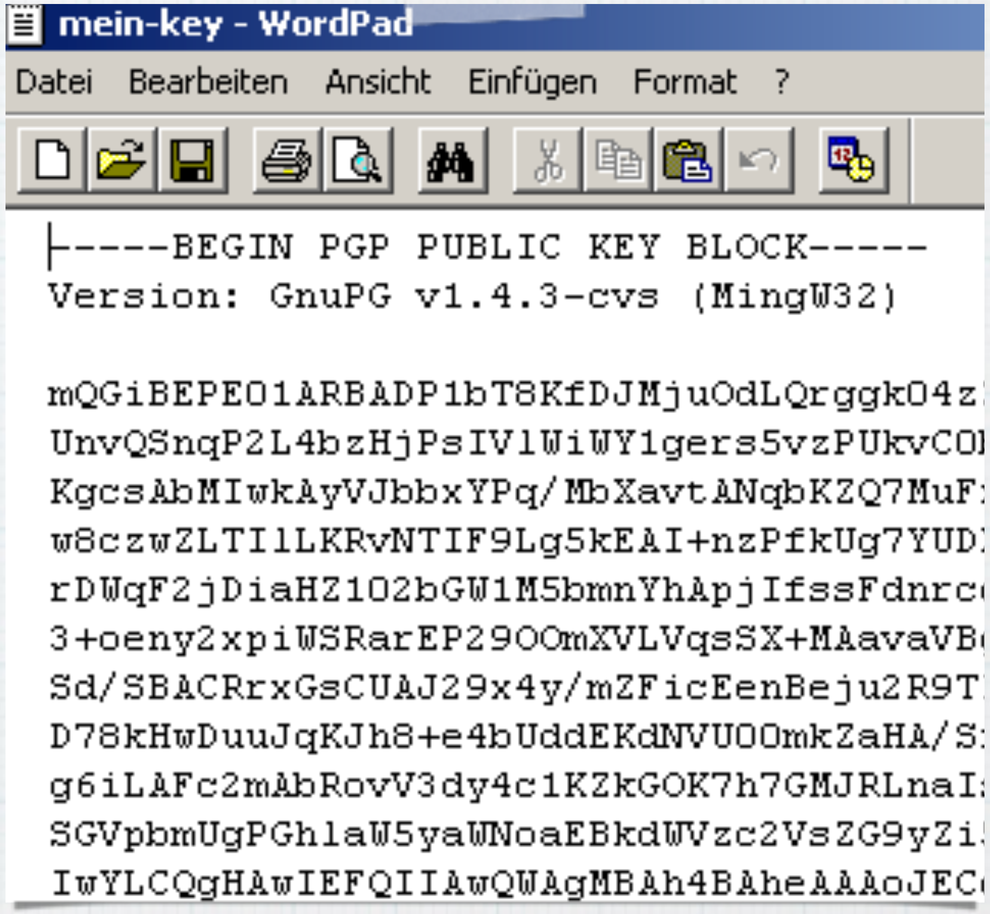
Key management is awful

- Generating keys
- Saving them on disk with passphrase
- Sharing long public keys via email
- Storing other people's keys, authenticating via fingerprints, managing keys



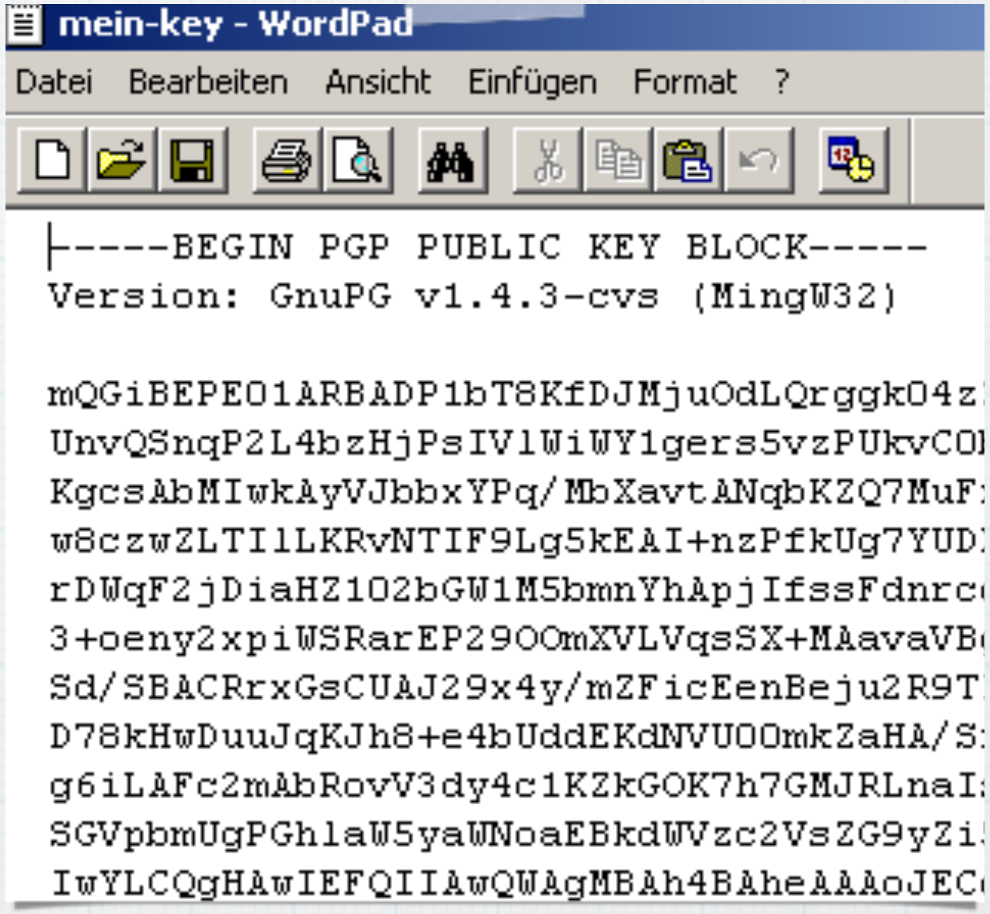
This sucks.

- Generating keys
- Saving them on disk with passphrase
- Sharing long public keys via email
- Storing other people's keys, authenticating via fingerprints, managing keys



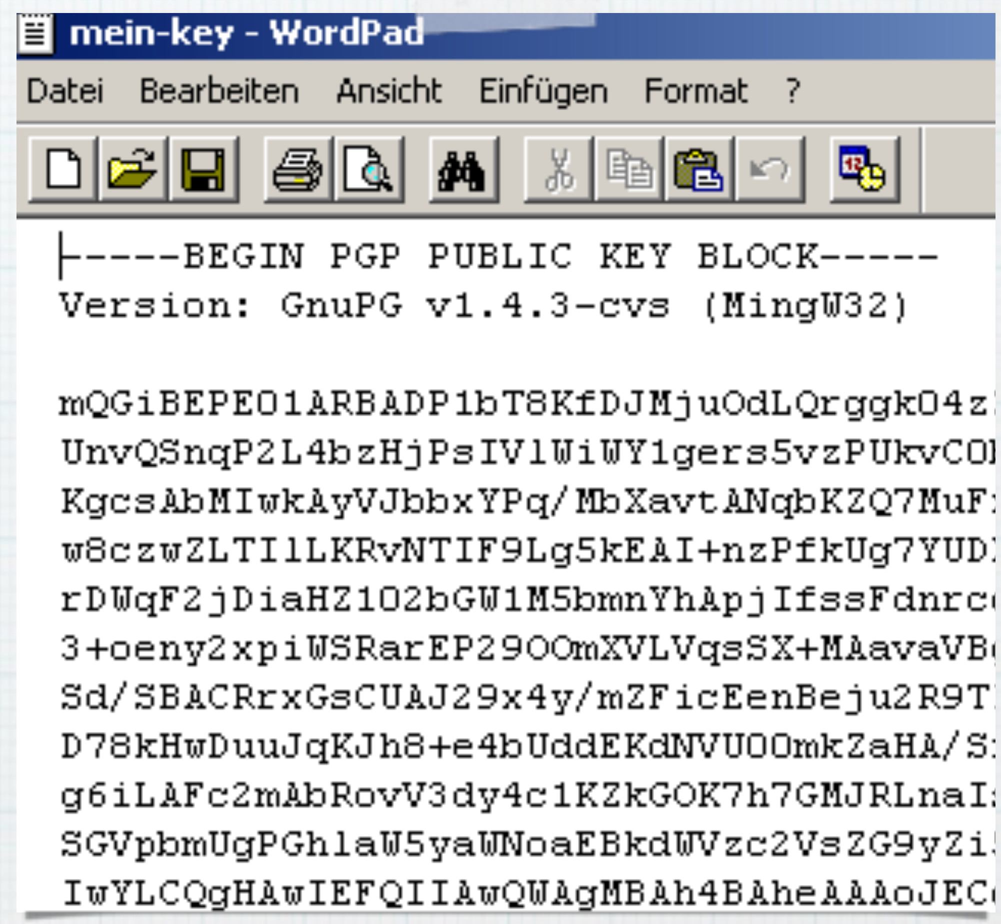
~~This sucks.~~

- Generating keys
- Saving them on disk with passphrase
- Sharing long public keys via email
- Storing other people's keys, authenticating via fingerprints, managing keys



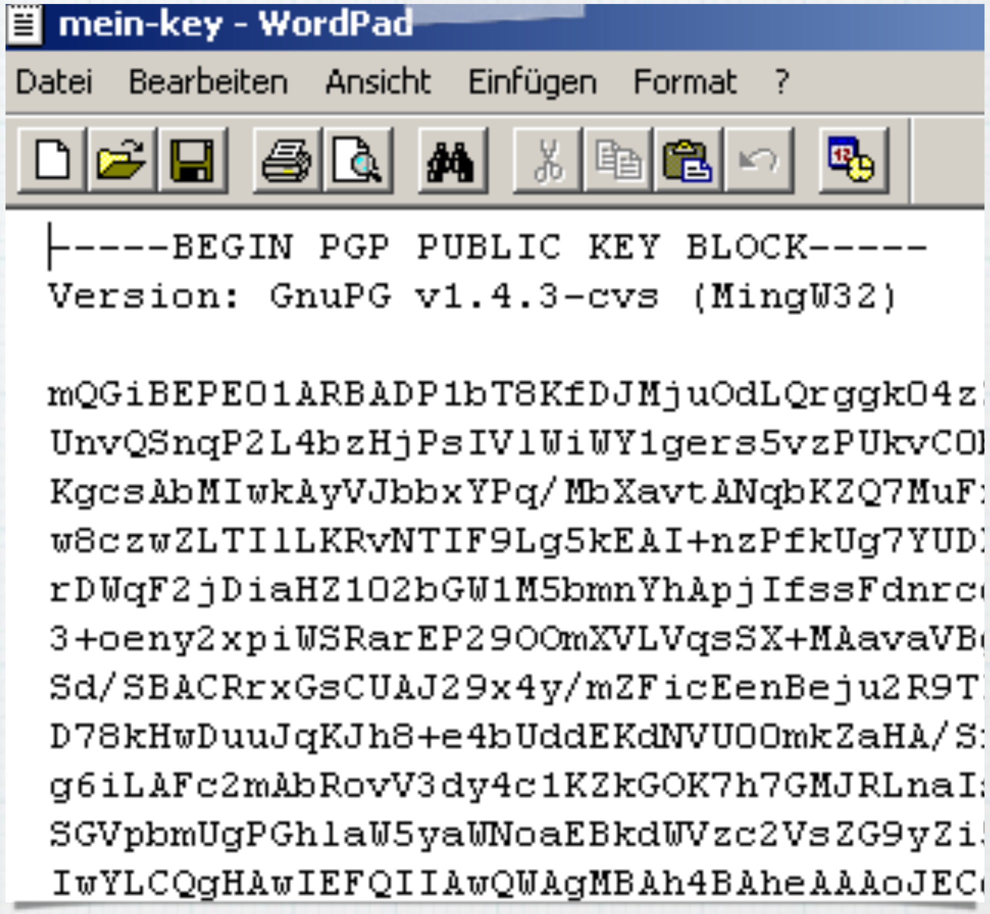
This is not convenient and we can do a lot better.

- Generating keys
- Saving them on disk with passphrase
- Sharing long public keys via email
- Storing other people's keys, authenticating via fingerprints, managing keys



This is not convenient and we can do a lot better.

1. * Generating keys
2. * Saving them on disk with passphrase
3. * Sharing long public keys via email
4. * Storing other people's keys, authenticating via fingerprints, managing keys



No key storage. No key files.

* miniLock asks you for a passphrase and uses it to generate your **key identity.**
* Enter that passphrase on any computer in the world, **obtain the same persistent key identity.**
* Nothing is ever stored anywhere.

miniLock uses **miniLock IDs.**

* miniLock IDs are shareable public keys that are **44 characters** long.
* Here's mine:
  `9LbEGtYBXRf1s0bIyw`
  `qbhty7uA00TF0XdynV`
  `+fIJlDc=`

User flow

* I enter my passphrase on a miniLock-capable computer and get my miniLock ID (always the same).
* I can send files to others using their ID.
* I can receive files sent to my ID.

This sucks less.

* No private key storage or management.
* No managing long key identities of others (miniLock IDs are **tweetable!**)
* miniLock IDs are so small that **they act as their own fingerprint.**

Nice features

* Easy to use interface
* Encrypt files for own use, decrypt later
* Runs on any computer

Nice features

- Send to multiple recipients (almost no performance decrease/file size increase)
- miniLock IDs of recipients are anonymized (even from the recipients)
- Fast!
- Retains filename on decryption

Best of all

* Peer-reviewed design specification

Best of all

* Peer-reviewed design specification

* **Fully audited** (Thanks to Cure53 and OTF)

Unit Test Kit

* Simulates entire user flow with randomized use-cases
* Also can run independent user flow elements atomically

miniLock in your app!

* Highly portable
* Comes with full design documents/spec/tests/reference
* Your app can be miniLock-ready

Everything will be released today

* Right after this talk
* AGPLv3 license
* But first…

How do the internals work?

* Reliance on elliptic curve cryptography (specifically, TweetNaCL)

* Mechanisms to evaluate strength of passphrases/ suggest strong pass phrases

* Scrypt.

TweetNaCL

- *"World's first **auditable** high-security cryptographic library"* — Daniel J. Bernstein
- Tiny, capable, easy to audit (fits in 100 tweets)
- Ported to JS by Dmitry Chestnykh

TweetNaCL

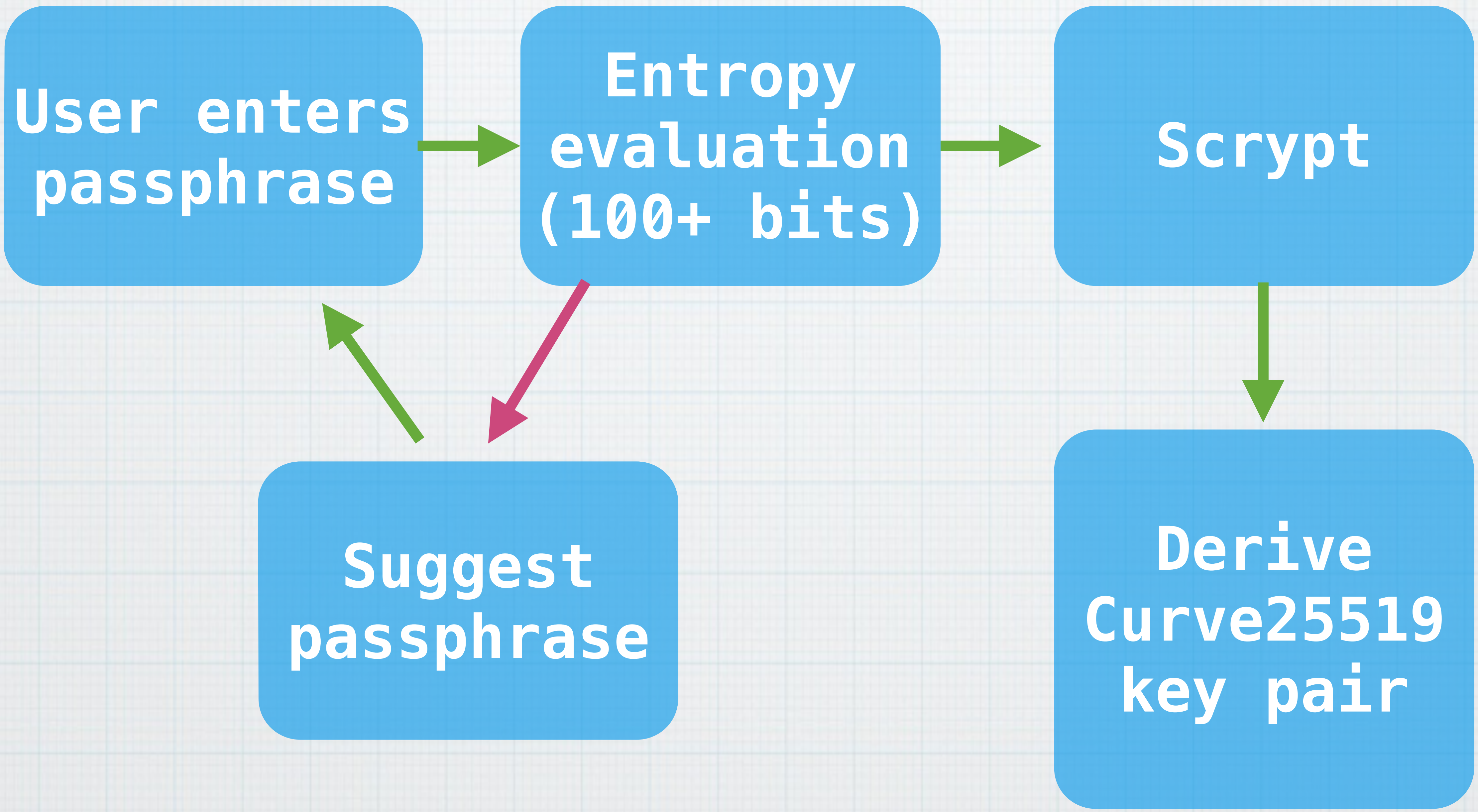
- Offers interface for:
  - Curve25519 (public key generation)
  - Xsalsa20 (Encryption)
  - Poly1305 (Authentication)

Curve25519

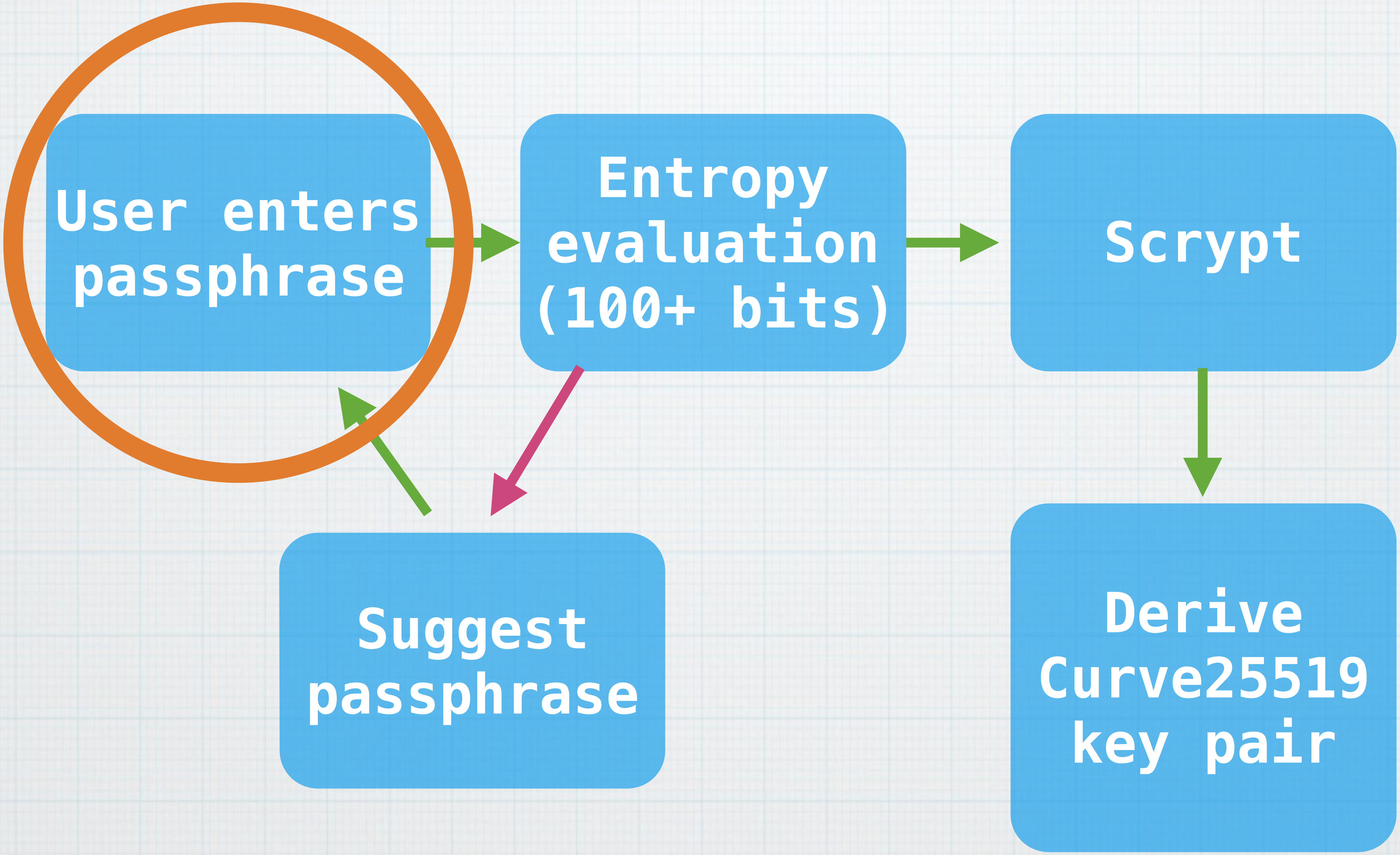
* 32-byte private keys,
32-byte public keys
(tiny!)

* ***Extremely*** fast

Key derivation



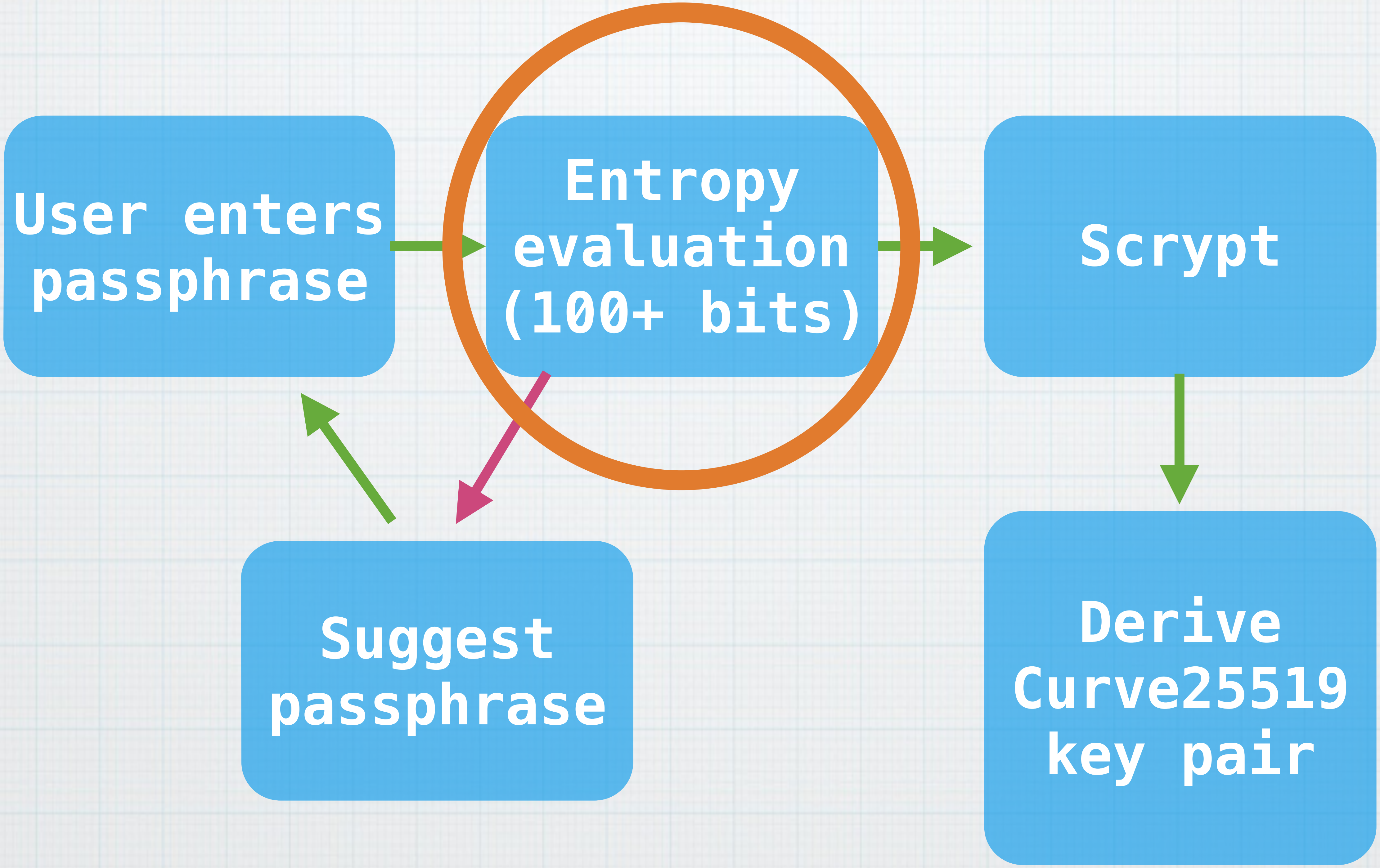
Key derivation



User enters passphrase

* Optimally, we want to map 32 bytes of entropy into the 32-byte Curve25519 private key
* Not practically feasible.

Key derivation



Entropy Evaluation

* We measure user passphrase entropy (using zxcvbn)

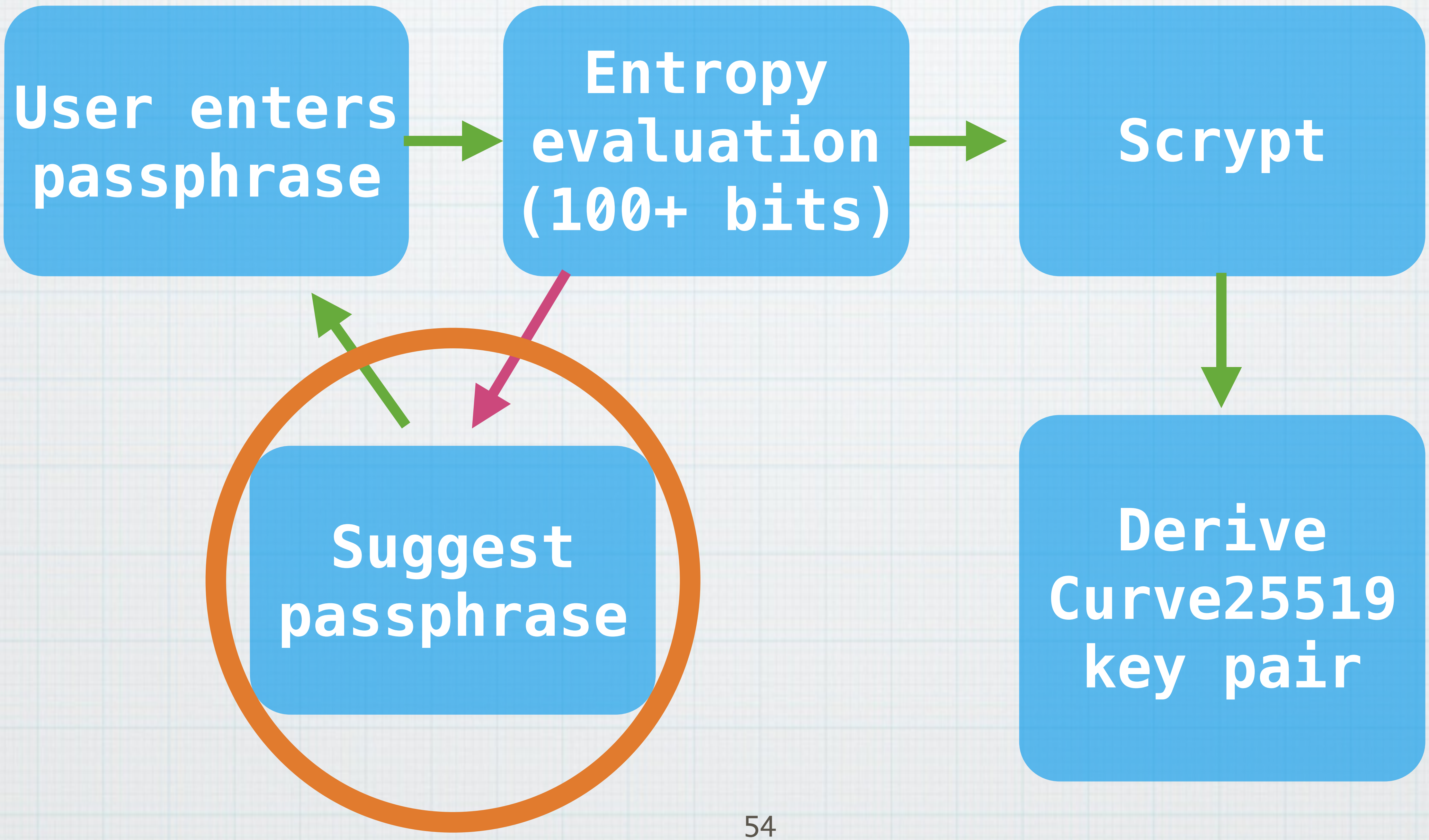
* miniLock suggests *"a long, unique phrase that makes sense only to you."*

Entropy Evaluation

* Less than 100-bit entropy pass phrases are not allowed (**miniLock will refuse to open**)

* Instead, miniLock constructs a suggested passphrase

Key derivation



Entropy Evaluation

* miniLock ships with dictionary of 58,110 most used English words
* 7-word passphrase = $58110^7 \sim= 2^{111}$

Towards reliable storage of 56-bit secrets in human memory

Joseph Bonneau
Princeton University

Stuart Schechter
Microsoft Research

Abstract

Challenging the conventional wisdom that users cannot remember cryptographically-strong secrets, we test the hypothesis that users can learn randomly-assigned 56-bit codes (encoded as either 6 words or 12 characters) through *spaced repetition*. We asked remote research participants to perform a distractor task that required logging into a website 90 times, over up to two weeks, with a password of their choosing. After they entered their chosen password correctly we displayed a short code (4 letters or 2 words, 18.8 bits) that we required them to

continue to be manufactured and deployed. But they are sufficiently pervasive that we must design our protocols around their limitations.)

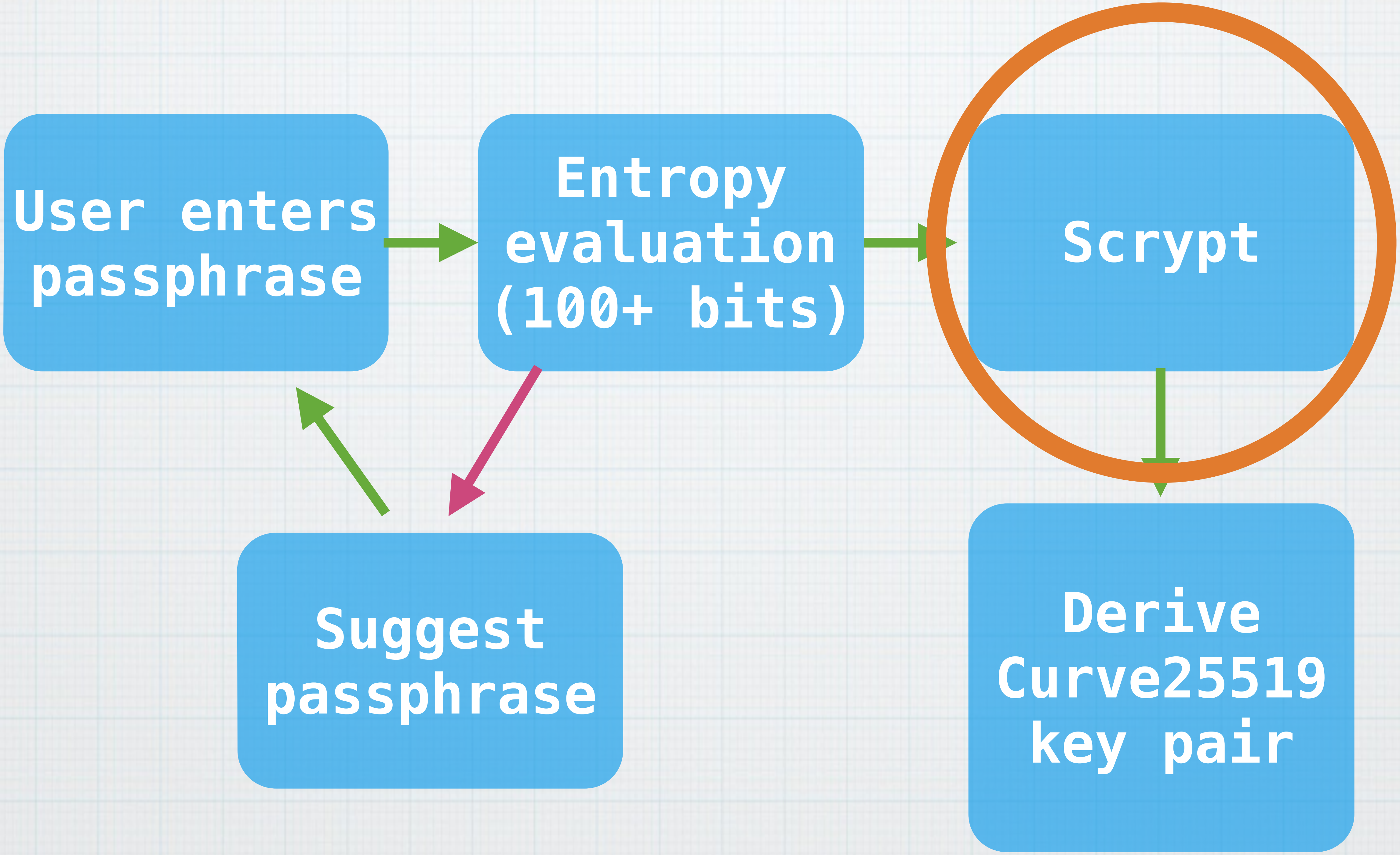
—Kaufman, Perlman and Speciner, 2002 [60]

The dismissal of human memory by the security community reached the point of parody long ago. While assigning random passwords to users was considered standard as recently in the mid-1980s [29], the practice died out in the 90s [4] and NIST guidelines now presume all passwords are user-chosen [35]. Most banks have even given up on expecting customers to memorize random

100+ bits of entropy

* Sufficient our purposes
* We can also work on making it harder to map the keyspace

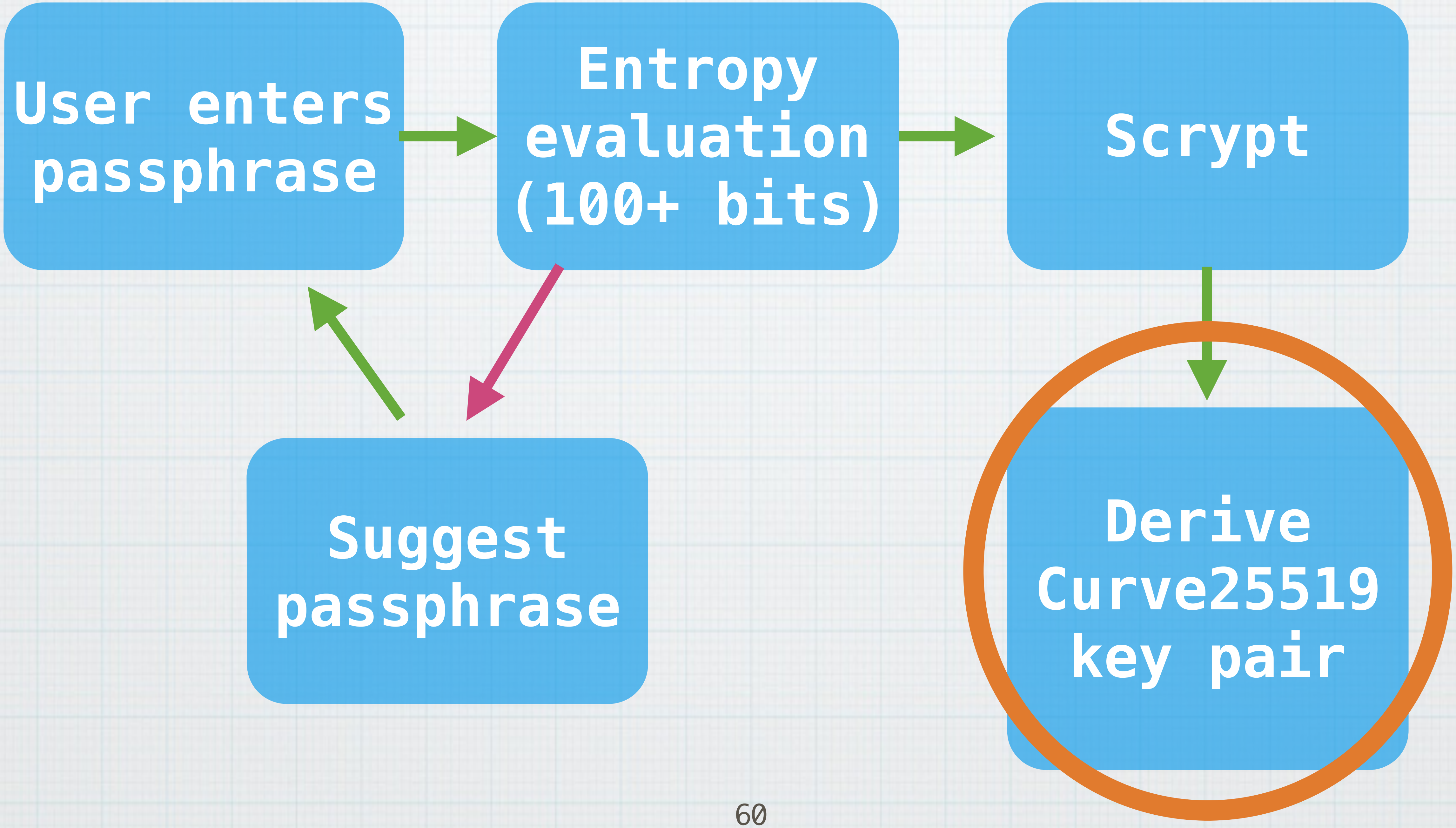
Key derivation



Scrypt

* Provides "memory-hard" key derivation.
* First we derive a SHA-512 hash of the passphrase
* Hash goes through 2^{17} rounds

Key derivation



Key derivation

* scrypt(L) = 32 bytes → Curve25519 private key
* miniLock ID is Base64 encoding of public key

File encryption format

- File is encrypted using a random unique symmetric key
- Symmetric key is encrypted asymmetrically once for each recipient and stored in header
- Both are authenticated encryption

File encryption format

Begin header

Sender ID

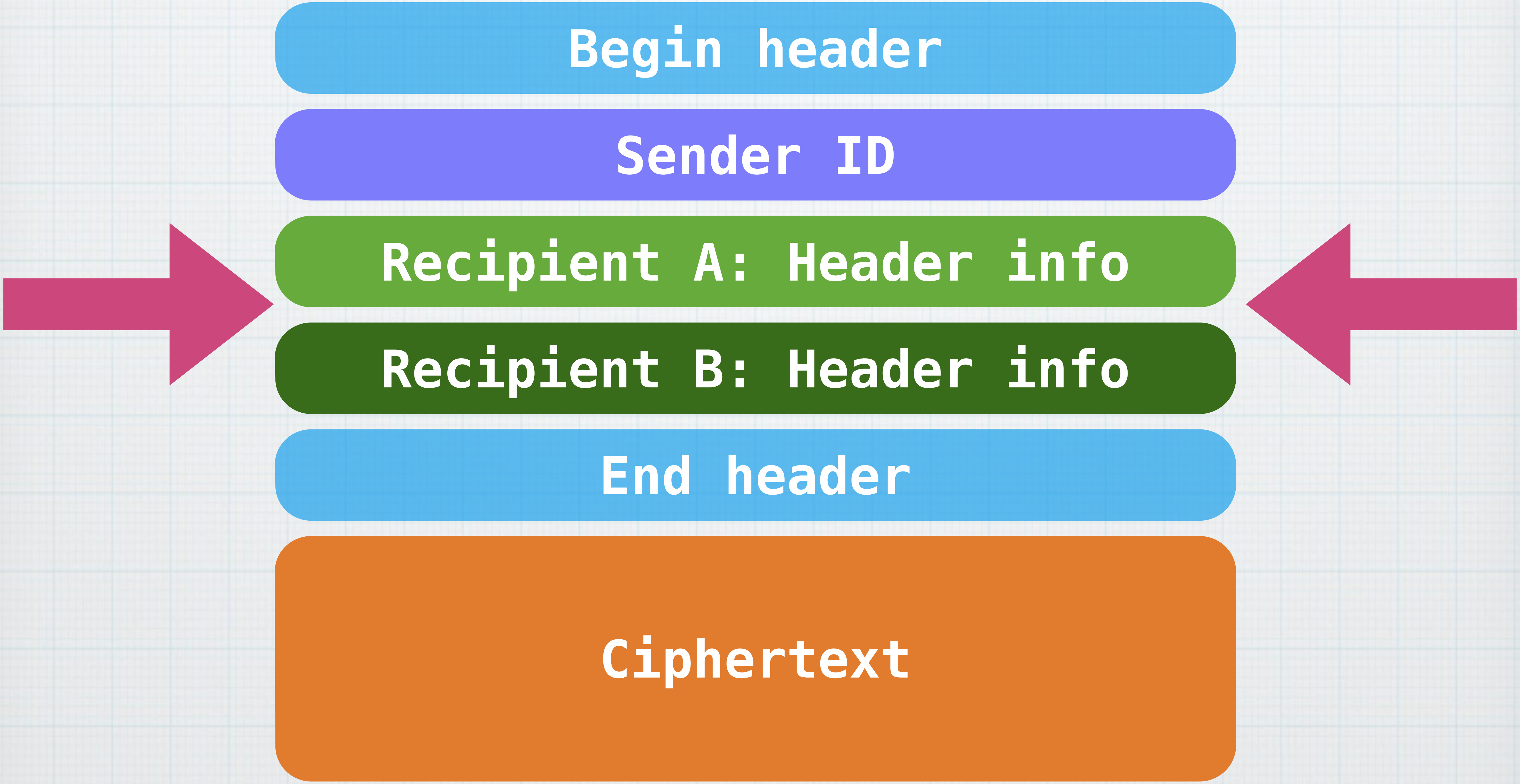
Recipient A: Header info

Recipient B: Header info

End header

Ciphertext

File encryption format



File encryption format

A

File key
File nonce
File name

(Encrypted to Recipient A’s public key)

B

File key
File nonce
File name

(Encrypted to Recipient B’s public key)

Header recipients are anonymized

?

File key
File nonce
File name

(Encrypted to Recipient ?'s public key)

?

File key
File nonce
File name

(Encrypted to Recipient ?'s public key)

Header recipients are anonymized

- Recipient attempts to decrypt every section of the header
- If they obtain an authenticated decryption, they know they are an intended recipient

General usage

* Share your miniLock IDs with friends
* Encrypt any files using friends' miniLock IDs
* Decrypt files sent to you
* Drag and drop simplicity

Demonstration

Release schedule

* miniLock is audited, reviewed software: ready for use
* 2-week test period before "App Store" release

Release schedule

* Will be released as a Chrome app
* Runs on Chrome OS, Windows, Mac, Linux
* 2-week test period before "App Store" release

Thank you!

* Get the code and documentation today

* **http://minilock.io**