# HyperJournal

A tamper-proof publishing system

# HyperJournal

How to build a tamper-proof publishing system

# Errata

http://github.com/emilbayes/jsconfau-2016

Doctors have the hippocratic oath

Software developers have “Don’t roll your own crypto”

# The narcissistic part 💅🏼

# @emilbayes

CommodiTrader
Copenhagen 🇩🇰

Just another hermite
from the internet 🐚📡

# What’s up with HyperJournal?

Provide an easy to use, cryptographically secure, integrated newsroom for distributed media organisations

# Motivation

Working with Syrian journalists in exile

# SNP

تمدن
جريدة سياسية ثقافية منوعة

زيتون

سوريتنا

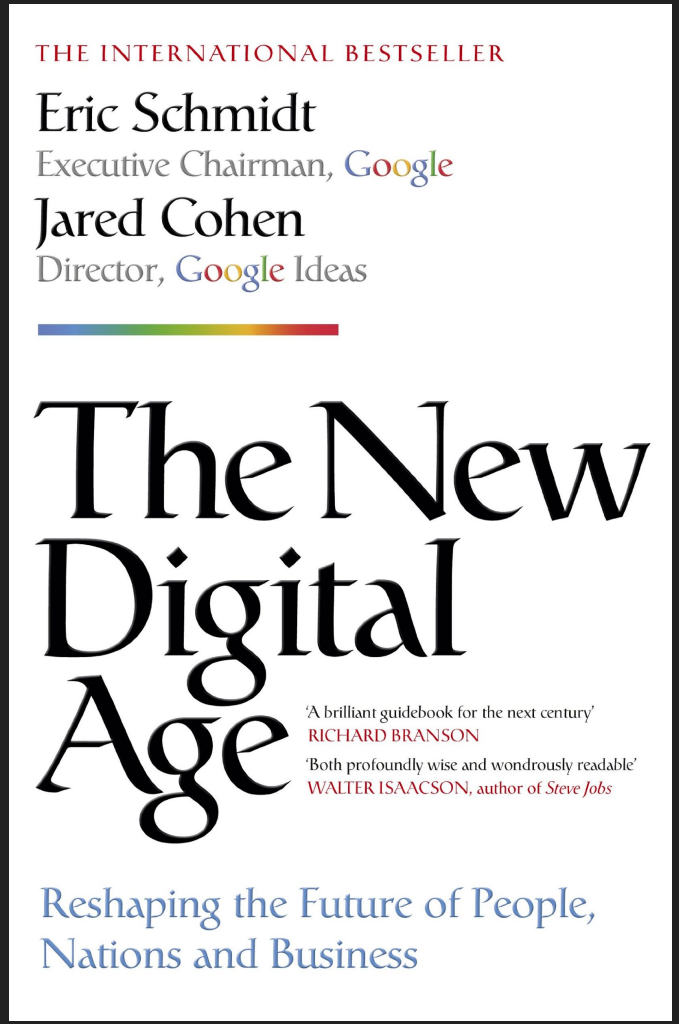
صدى الشام
سياسية . اجتماعية . منوعة

عنب بلدي
enab baladi
من كرم الثورة

عين المدينة

كلنا سوريون
www.allsyrian.net

# Main ideas


THE INTERNATIONAL BESTSELLER
Eric Schmidt
Executive Chairman, Google
Jared Cohen
Director, Google Ideas
The New Digital Age
'A brilliant guidebook for the next century'
RICHARD BRANSON
'Both profoundly wise and wondrously readable'
WALTER ISAACSON, author of Steve Jobs
Reshaping the Future of People, Nations and Business

Connectivity gives new hope to freedom of speech. Adversaries can no longer stop the flow of information.

“Imagine an international NGO whose mission it is to facilitate confidential reporting”

The NGO outside the country, so they don’t have to answer to local legislation.

“In order to protect the identities of journalists [...] every reporter is [...] registered in the system with a unique code” - We’ll call this a public key

Journalists and editors communicate via their anonymous handles and might never meet (except once).

Which problem did we want to solve again?

# Tamper-proof publishing

(over insecure channels)

An encryption function $E(m) = c$

A decryption function $D(c) = m$

Parameters - some public, some secret, eg. key material

Our first cipher! 🔓

Requires the computational power of a determined 4th grader to break

Adequate for sending notes in class

Adequate for “encrypting” christmas shopping lists

Adequate for commanding legions across the Roman Empire

# Cæsar Cipher

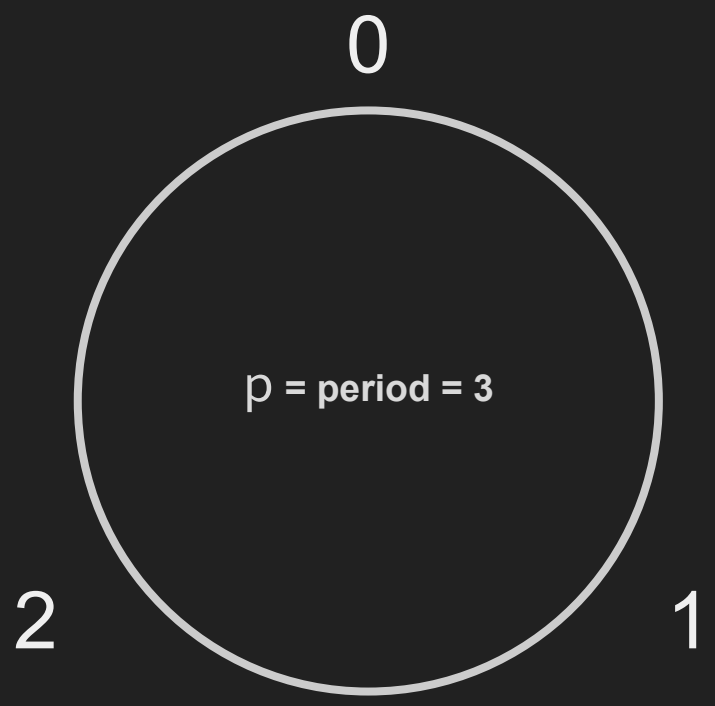
```
var key = 3
function encrypt (plaintext) {
 return plaintext.map(function (byte) {
  return mod(byte + key, 256)
 })
}

function decrypt (ciphertext) {
 return ciphertext.map(function (byte) {
  return mod(byte + key, 256)
 })
}
```

# Quick intro to `mod(n, p)`

Modulo arithmetic, also called clock arithmetic

You use it all the time; degrees, time, GPS

```
mod(n, p) !== (n % p)
```



# Quick intro to Buffers

“Raw” memory access, behaves like an array

Backed by `Uint8Array`

Each “cell” can contain an integer from 0 - 255, eg. one byte

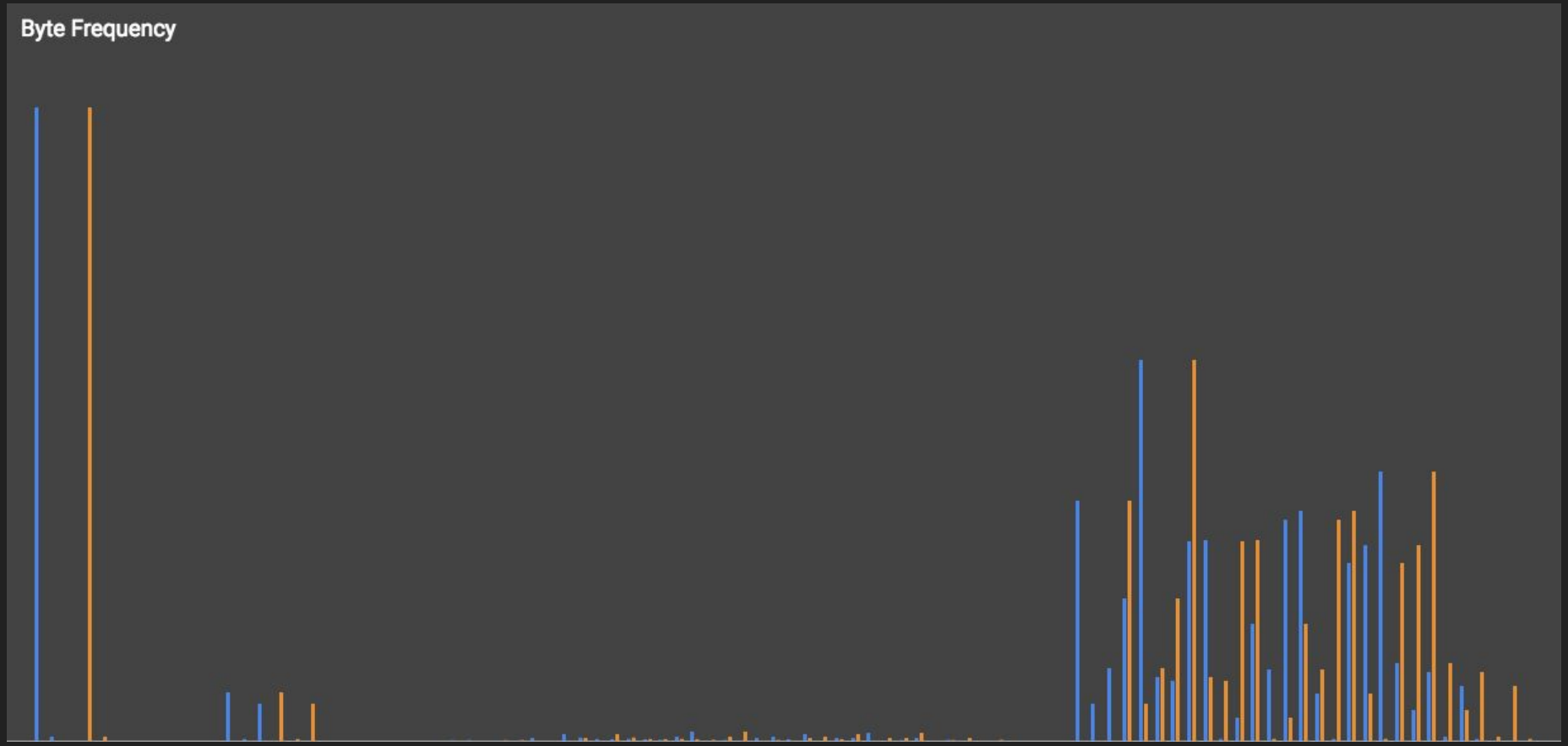
“Default” container when interacting with outside world in Node, eg. net, fs, http etc.

Good for binary data (blobs)

```
Buffer.from('I like cats!')
Buffer.from([0, 1, 2, 3, 4])
Buffer.alloc(32)
```

```
var plaintext = Buffer.from(‘I like cats’)
var ciphertext = encrypt(plaintext)
// <Buffer 6c 23 6f 6c 6e 68 23 67 72 6a 76>
```

```
var fs = require(‘fs’)
var crusoe = fs.readFileSync(‘./crusoe.txt’)
crusoe.length // 655616

var freq = require(‘buffer-byte-frequency’)
freq(ciphertext)
```


Byte Frequency

🔓

# Essentially security by obscurity

# Perfect cipher

# One Time Pad

```
var xor = require('buffer-xor')
var encrypt = oneTimePad
var decrypt = oneTimePad

function oneTimePad(key, text) {
  return xor(key, text)
}
```

Key length ≥ text length

Disposable keys :(

Key must be completely random. Like, very very random!

Discovered 1882 / 1917

```
// require(‘crypto’).randomBytes(11)
var keyA = Buffer.from([0x01, 0x3d, 0xe2, 0xc8, 0x5a,
0x25, 0xd1, 0x45, 0x79, 0x8f, 0xf3])
var keyB = Buffer.from([0x01, 0x3d, 0xe2, 0xc8, 0x5a,
0x25, 0xd1, 0x42, 0x77, 0x9c, 0xf3])

var plaintext = Buffer.from('I like dogs')
var ciphertext = encrypt(keyA, plaintext)


decrypt(keyB, ciphertext) // I like cats
```

Keys are problematic because sharing them requires a secure channel

The problem is bootstrapping trust

Don’t trust people on the internet

What if we could just put a unforgeable signature on the data?

# Hash Functions

# (Cryptographic) Hash Functions

Hashing a blob is like taking it’s fingerprint

```
var crypto = require(‘crypto’)

var hash = crypto.createHash(‘sha256’)

hash.update(Buffer(…))

// …

hash.digest() // Your fingerprint!
```

# Properties and applications

Deterministic

Maps large data to small data (like a fingerprint)

One-way

Collision resistant

ETags (Caching)

Content-addressable storage

Commitment

Password verification

Proof-of-work

Merkle DAGs (Ugh, block chains)

File integrity (!!!)

# MD5

~~MD5~~

SHA-0

~~MD5~~

~~SHA-0~~

SHA-1

~~MD5~~

~~SHA-0~~

~~SHA-1~~

SHA-256 / SHA-512

~~MD5~~

~~SHA-0~~

~~SHA-1~~

SHA-256 / SHA-512

BLAKE2


**Troy Hunt**
@troyhunt

Follow

How to identify a vehicle at risk of collisions:

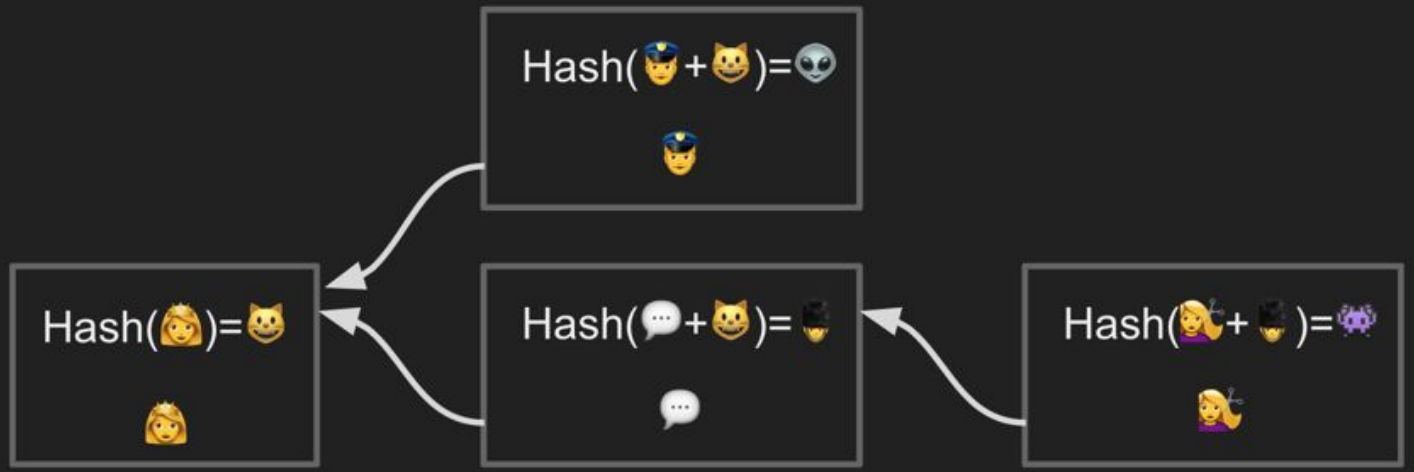
9:01 AM - 6 Apr 2016

2,670 2,800

https://twitter.com/troyhunt/status/717487467876605953

Side-note: Use a proper password hash function (Argon2, scrypt, pbkdf2)

# Digression - Mekle DAG



# Demo

Anyone can compute a hash, so how can we verify where it came from?

# Real-world cryptography

# OpenSSL vs NaCl / libsodium

```
require('sodium-native')
```

# Asymmetric Cryptography 🔐

# Public / Private Key Cryptography

# Identity / Secret Key Cryptography

Have someone’s public key, you can send them secret messages

More importantly, have someone’s public key, you can decrypt their private key messages

If everyone can decrypt what you have said, why even encrypt in the first place?

# Signatures! 🔏

# What’s a signature?

# Non-repudation

# How does a signature work?

```
var privateKey = Buffer(…)
var data = Buffer(…)

var dataHash = hash(data)
var signature = encrypt(privateKey, dataHash)
```

**Jonathan Zdziarski**
@JZdziarski

Follow

PKI / PGP Primer:
🔑 Public Key
🗝 Private Key
📝 Message

📝+🔑 = 🔒✉️ Encrypted
🔒✉️+🗝 = 🔓📝 Decrypted
📝+🗝 = 🔏✉️ Signed
🔏✉️ + 🔑 = 👤 Authenticated

11:44 PM - 13 Jul 2016

3,537 4,296

https://twitter.com/JZdziarski/status/753223642297892864

# How do we get a key pair?

```
var sodium = require(‘sodium-native’)

var identity =
Buffer.alloc(sodium.crypto_sign_PUBLICKEYBYTES)
var privateKey =
Buffer.alloc(sodium.crypto_sign_SECRETKEYBYTES)

sodium.crypto_sign_keypair(identity, privateKey)

// PROFIT!
```

Identity is 32 bytes
Private key is 32 bytes

You can’t remember
32 bytes

44a606679ddeb141b346

bc600650786f7d380e11

14dbd009b066ff8ed921

30b8

What if we could store key pairs in our BRAINS

```
var hash = require('mindvault')
var generatePassphrase = require('eff-diceware-passphrase')
var sodium = require(‘sodium-native’)

var appId = Buffer.alloc(hash.APP_ID_BYTES).fill('mindvault')
var salt = Buffer.from('john@example.com')
var passphrase = Buffer.from(generatePassphrase.entropy(100).join('
'))

var seed = hash(passphrase, salt, appId)

var identity = Buffer.alloc(sodium.crypto_sign_PUBLICKEYBYTES)
var privateKey = Buffer.alloc(sodium.crypto_sign_SECRETKEYBYTES)

sodium.crypto_sign_seed_keypair(identity, privateKey, seed)
```

```
var message = new Buffer('Hello, World!')
var signature = alloc(sodium.crypto_sign_BYTES)

sodium.crypto_sign_detached(signature, message, privateKey)

// … Some time later, maybe some other machine

var isValid = sodium.crypto_sign_verify_detached(signature, message, publicKey)
```

**Jonathan Zdziarski**
@JZdziarski

Follow

PKI / PGP Primer:
🔑 Public Key
🗝 Private Key
📝 Message

📝+🔑 = 🔒✉️ Encrypted
🔒✉️+🗝 = 🔓📝 Decrypted
📝+🗝 = 🔏✉️ Signed
🔏✉️ + 🔑 = 👤 Authenticated

11:44 PM - 13 Jul 2016

3,537 4,296

https://twitter.com/JZdziarski/status/753223642297892864

Now we have tamper-proof publishing!

# Open Problems

(in HyperJournal)

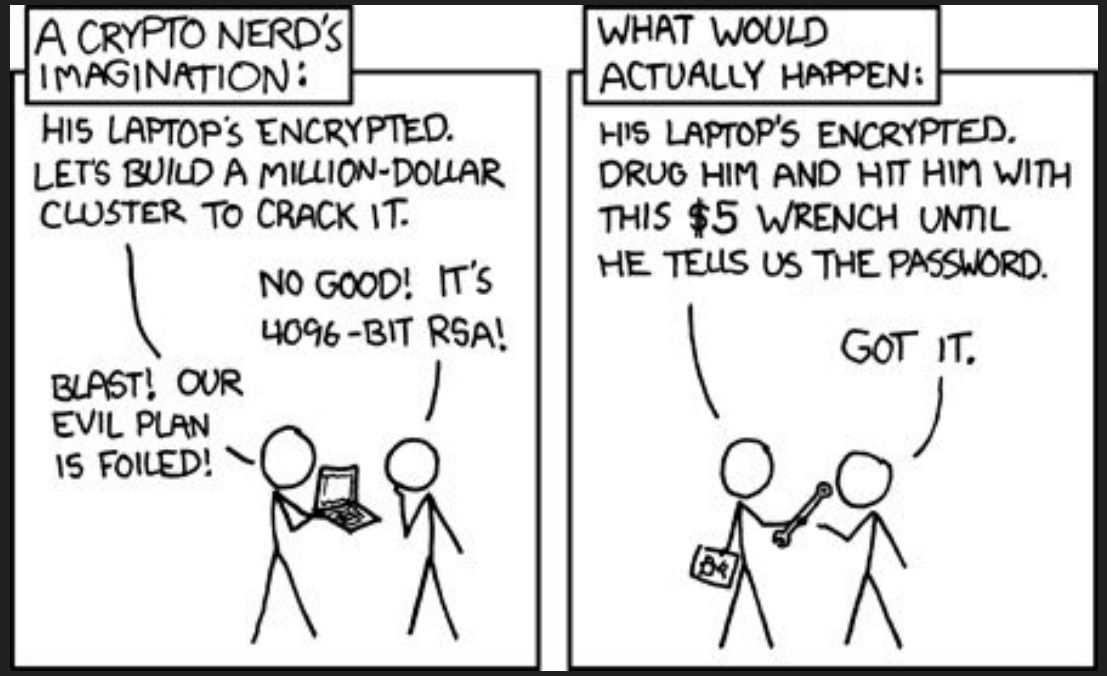
# Delivering content to readers, securely

Secure communication (actual encryption, onion routing?)

DDoS
Fake SSL Certs
DNS Poisoning

# The REALLY bad guys

The REALLY bad guys (ie. your crypto might be unbreakable, but your knee caps aren’t)




https://xkcd.com/538/

# Interesting Projects

Merkle Tree

https://github.com/ipfs/ipfs

https://github.com/mafintosh/hypercore

https://github.com/mafintosh/hyperdrive

https://github.com/datproject/dat

Merkle DAG

https://github.com/mafintosh/hyperlog

https://github.com/ssbc/secure-scuttlebutt

https://github.com/digidem/osm-p2p

Bittorrent

https://github.com/feross/webtorrent

https://github.com/mmckegg/ferment