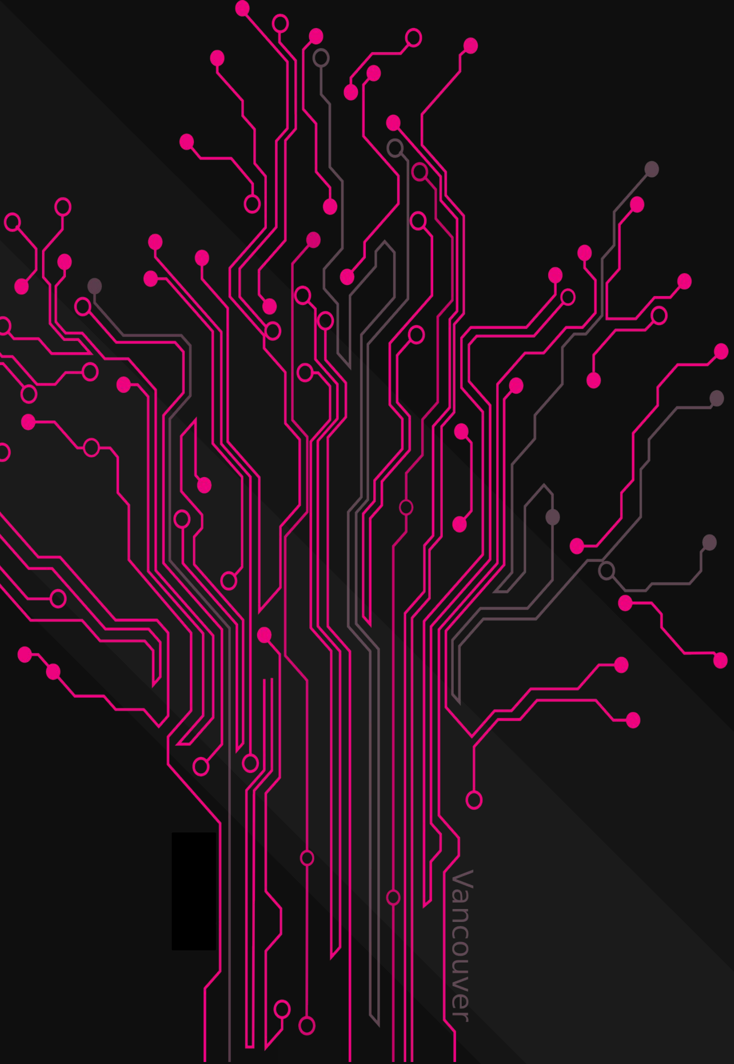


Cryptography 101

Brodie McRae
CISO, Axiom Zen
@brodieve

Who is Brodie McRae?

- Vancouver born+raised
- Hacking software checks to play games for free
- LAMP stack developer before serialization was cool *("PHP3 is cool, but 4's gonna be mint!")*
- Basically it was this or jail
- Currently head of security at Dapper Labs

Who is Brodie McRae?

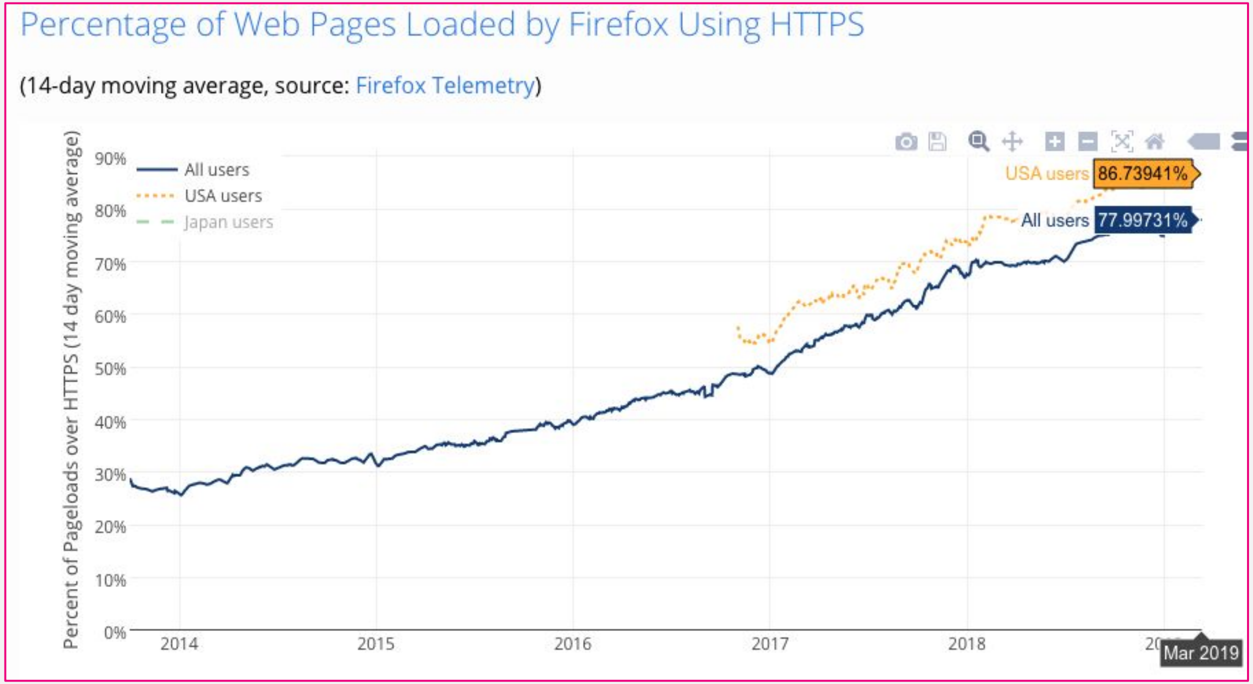
My company built a blockchain called Flow. We *literally* rolled our own crypto.

Don’t roll your own crypto.

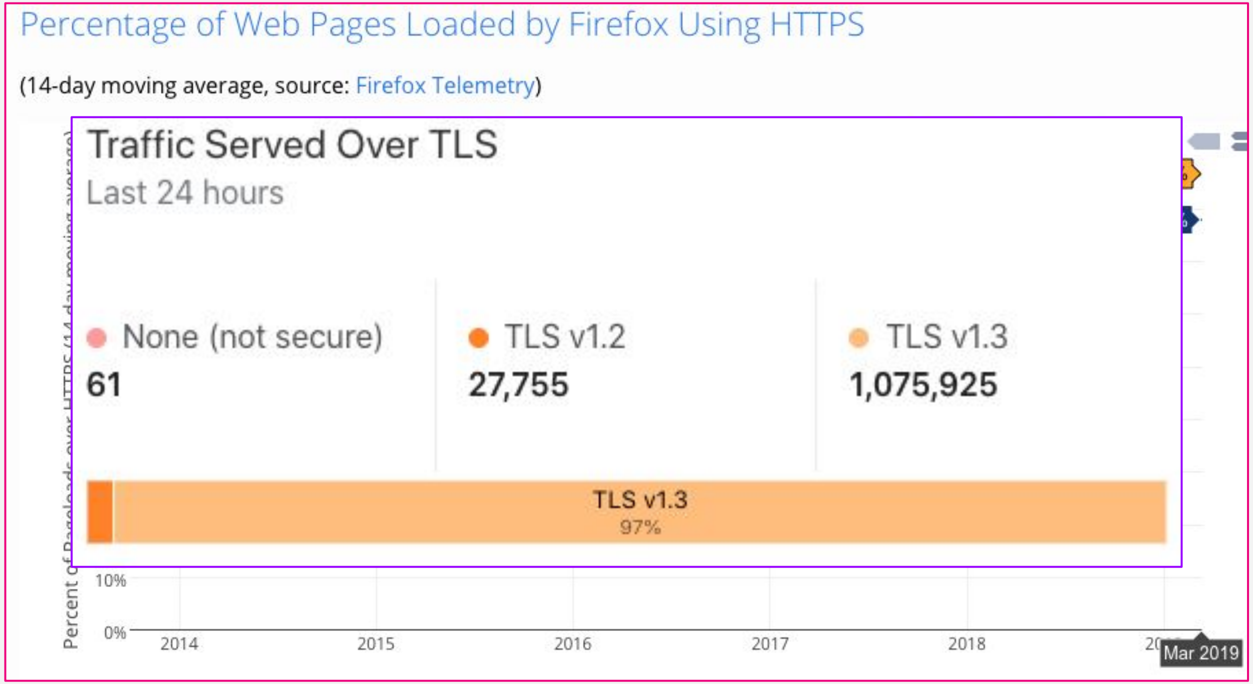
What is cryptography?

Secret messages using math / algorithms.

Today, cryptography is used to control who can see certain information, and also guarantee the *authenticity* of it.



In 2020, it underpins basically everything.



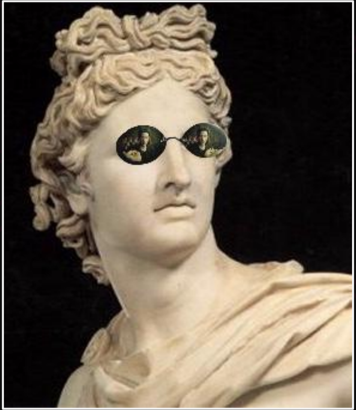
In 2020, it underpins basically everything.

What is cryptography?

The word cryptography comes from Greek,

crypto secret

graphy writing



Kryptos, Greek God of block ciphers

Terms

We need to lay out some terms.

We really do.

Terms

Data

1s and 0s. Even this text you're reading is fundamentally just 1s and 0s.

"*bits.*"

Terms

Cleartext

- Normal, meaningful *data* that a computer or person can understand on its own.
- May not be human readable
  - Binary program code
  - ‘Encoded’ (e.g., base64)

Terms

Encoding

Converting data from one representation into some other representation using an algorithm (set of rules and steps).

Easily reversible.

Encoding

Binary 01100010 01100001 01100011 01101111 01101110

ASCII bacon

Base64 YmFjb24=

Binary	Oct	Dec	Hex	Glyph		
				'63	'65	'67
110 0000	140	96	60		@	`
110 0001	141	97	61		a	
110 0010	142	98	62		b	
110 0011	143	99	63		c	
110 0100	144	100	64		d	

Why encoding?

- Humans are bad at reading binary.
- Different computers have different architectures and capabilities, so common languages help.

Email encoding example

```
From: Brodie McRae <bronie@mylittlepony.hasbro.com>
To: Alex <alexs@finewineburg.io>
Subject: Super sweet pic of nothing
Content-Type: multipart/mixed; boundary=45eg2c1aa958146c04054e41653a

--45eg2c1aa958146c04054e41653a
Content-Type: text/plain; charset=UTF-8; format=flowed; delsp=yes

Yo here is some pretty sweet nothingness can you dig it.
--
--45eg2c1aa958146c04054e41653a
Content-Type: image/png; name="nada.png"
Content-Disposition: attachment; filename="nada.png"
Content-Transfer-Encoding: base64

iVBORw0KGgoAAAANSUhEUgAAAAEAAAABCAQAAAC1HAwCAAAAC0lEQVR42mNkYAAAAAYAAjCB0C8A
AAAASUVORK5CYII=
--45eg2c1aa958146c04054e41653a--
```

JWT encoding example

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Imt0bTc5MHIiLCJpYXQiOjE1MTYyMzkwMjJ9.L7602mf70gDFYZzhF9PhKWQFnjIw8P2K-GFDxJjLeiw

```
echo
"eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Imt0bTc5MHIiLCJpYXQiOjE1MTYyMzkwMjJ9" |
base64 -D
{"sub":"1234567890","name":"ktm790r","iat":1516239022}
```

Terms

Ciphertext

- *Data* that cannot be understood by a human or computer without additional information (a *secret*).
- If it's well done, it appears to be random.
- Usually not text.

Terms

Encryption

Using a secret to convert cleartext into ciphertext data that, without the secret, is meaningless.

Encryption

Using a secret to convert cleartext into ciphertext data that, without the secret, is meaningless.

"bacon" encrypt("bacon", secret)

"1☃️9!^%"

"bacon" decrypt("1☃️9!^%", secret)

Encryption

Using a secret to convert cleartext into ciphertext data that, without the secret, is meaningless.

Secret: “shift 13 places in the alphabet for each word character”

cleartext

“flat dow truth earth”

ciphertext

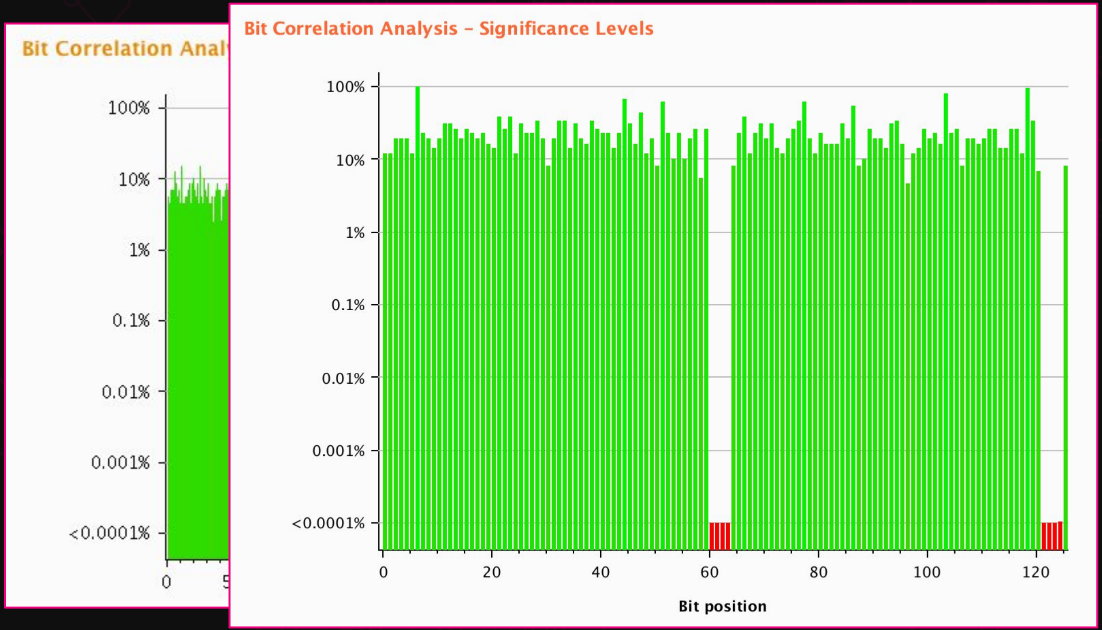
“syng qbj gehgu rnegu”

```
encrypt(cleartext, secret)
    = “syng qbj gehgu rnegu”

decrypt(ciphertext, secret)
    = “flat dow truth earth”
```

Encryption

In practice, encrypted data should appear highly random.



Encoding vs Encryption

Encoding takes source data and applies steps (public algorithm) to change its form.

Binary	Oct	Dec	Hex	Glyph		
				'63	'65	'67
110 0000	140	96	60		@	`
110 0001	141	97	61		a	
110 0010	142	98	62		b	
110 0011	143	99	63		c	
110 0100	144	100	64		d	

Encryption requires the source data and some *secret* info (e.g., a digital key and/or algorithm)

"shift 13 places in the alphabet for each word character"

Encoding + Encryption

PGP encryption often combines the two, encoding its encrypted data into base64:

```
$ head -n 5 test.pgp
-----BEGIN PGP MESSAGE-----

hQIMAwo87e15Vh9UAQ/6Awtm9T2LFyqxtvPJXrzEpM/1J7VLhAG6SvmGMIPuN30b
JlLYnBlvfmMj+olZbmMjiwKDgPvOr4a7QRH8nrnQs2qmIVdUy/UNptuiNtiop8MZ
+3ZPESsZs+CNa7mr4wHuoZtwJ6tk++ObCxW7mqY1s+OaofP4MBgzSYbAPBOJ6VCX
```

Encryption 101

Caesar's dead, his knowledge lost, so how does encryption work today?

"Using a secret to convert cleartext into ciphertext data that, without the secret, is meaningless."

..but how?

Terms

XOR – "Exclusive OR"

A logical operator that takes two inputs (A,B):

A		B		
0	$\oplus$	0	=	0
0	$\oplus$	1	=	1
1	$\oplus$	0	=	1
1	$\oplus$	1	=	0

"A or B, but not both."

Bitwise XOR

You can XOR two bytes of data, but it is a “bitwise” operation”:

$$
\begin{array}{r}
\oplus \; 01001011 \\
10001010 \\
\hline
\oplus \; 11000001 \\
10001010 \\
\hline
01001011
\end{array}
$$

One of the special properties of XOR is that it’s reversible:

$$A \oplus B \oplus B = A$$

$$(\; B \oplus B = 0$$

$$A \oplus 0 = A \;)$$

“Perfect” One Time Pad

A one time pad is considered “perfect” if random and used only once. $|K| > |m|$; Shannon’s theorems

cleartext

01100110011001010111001001101110011000010110111001100100

⊕

OTP “key”

01100010011000010110001101101111011011100110000101101001

=

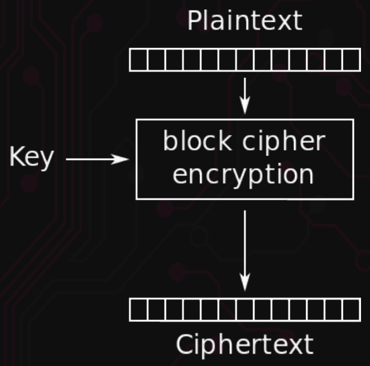
ciphertext

00000100000001000001000100000001000011110000111100001101

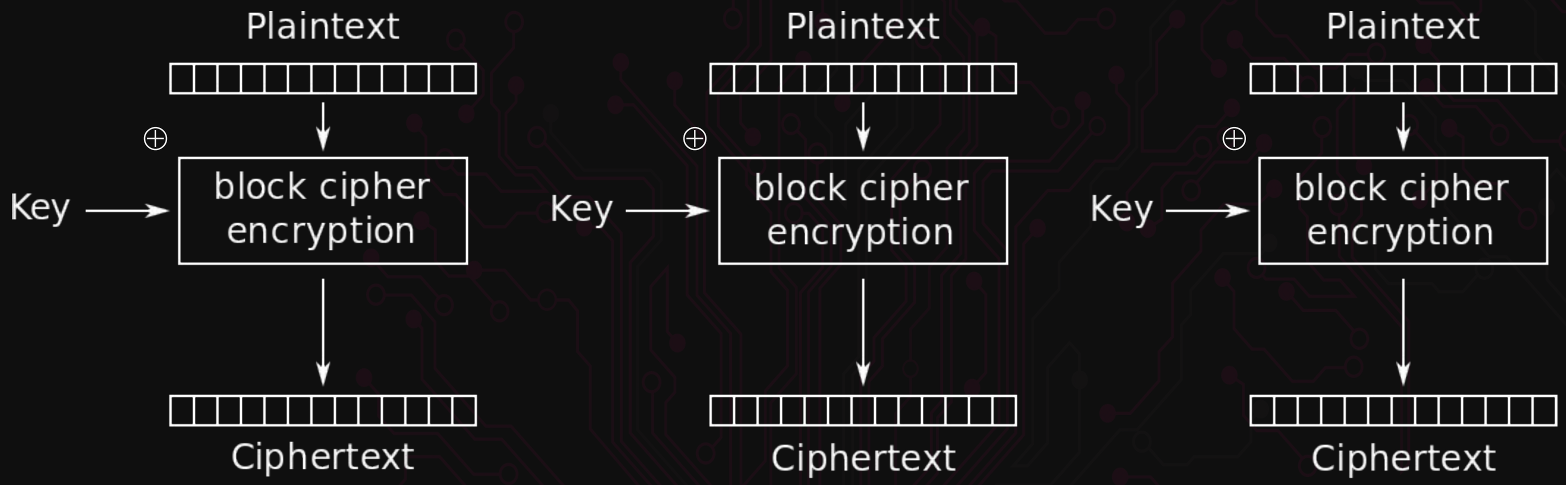
“Less than perfect” Encryption

“How do I encrypt 10GB of data with a 256-bit key?”

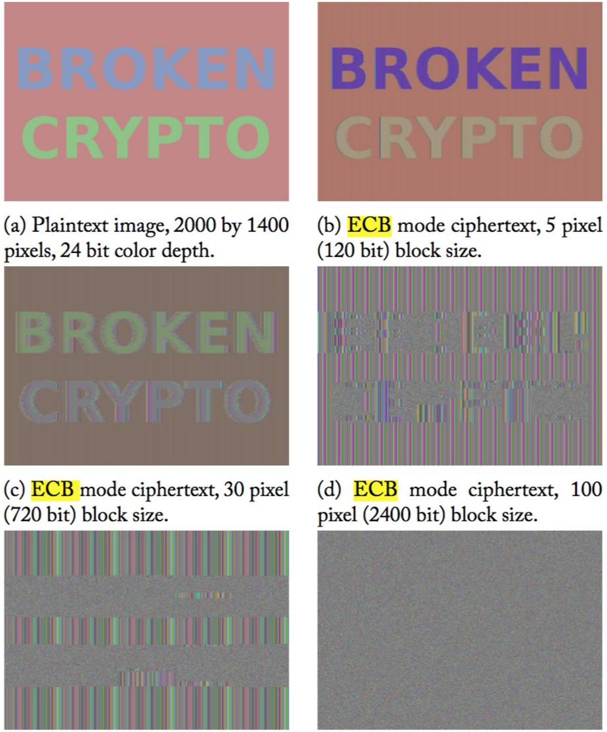
- ECB
- CBC
- CTR



ECB (Electronic Code Book)



ECB's determinism:



(a) Plaintext image, 2000 by 1400 pixels, 24 bit color depth.

(b) ECB mode ciphertext, 5 pixel (120 bit) block size.

(c) ECB mode ciphertext, 30 pixel (720 bit) block size.

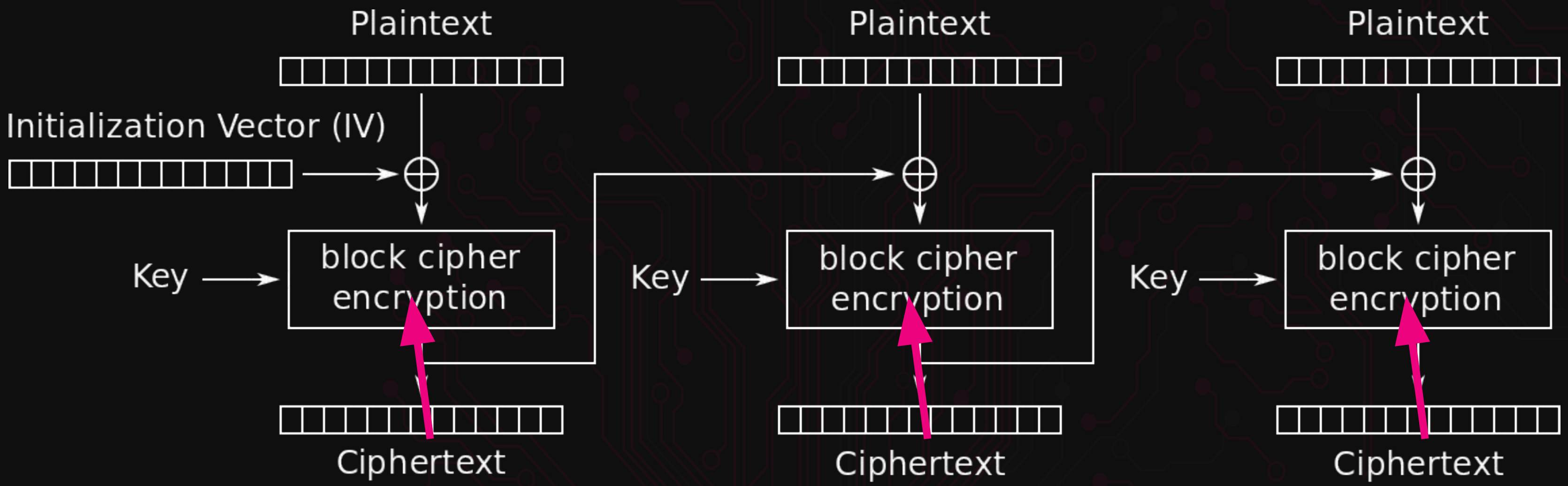
(d) ECB mode ciphertext, 100 pixel (2400 bit) block size.

(e) ECB mode ciphertext, 400 pixel (9600 bit) block size.

(f) Ciphertext under idealized encryption.

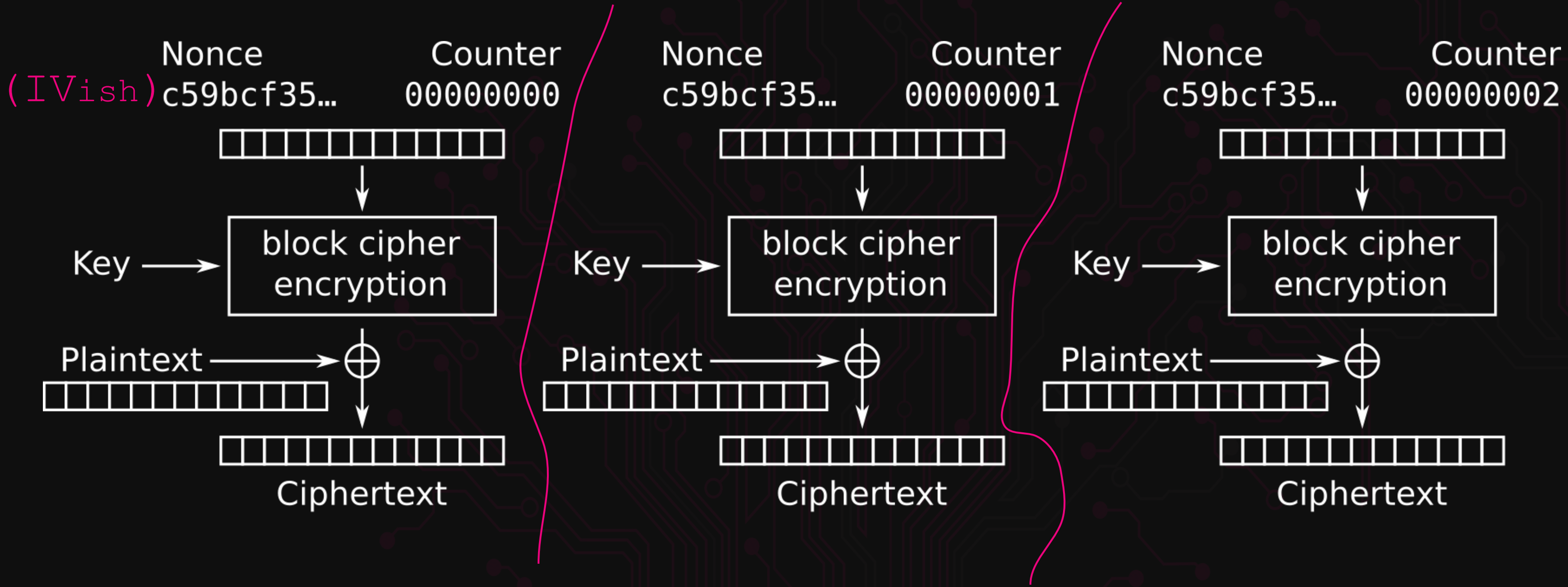
Lvh (crypto101)

CBC (Cipher Block Chaining) mode



Decrypt

CTR (e.g., GCM) Counter Mode



Terms

Hashing

Converting any piece of data, using an algorithm, into a typically much smaller - but “unique” - identifier.

Terms

Hashing

Think fingerprinting:

- Algorithm: taking someone's fingerprint
- Input: actual data (real person w/ finger)
- Output: unique identifier called a hash, or a digest (fingerprint image)

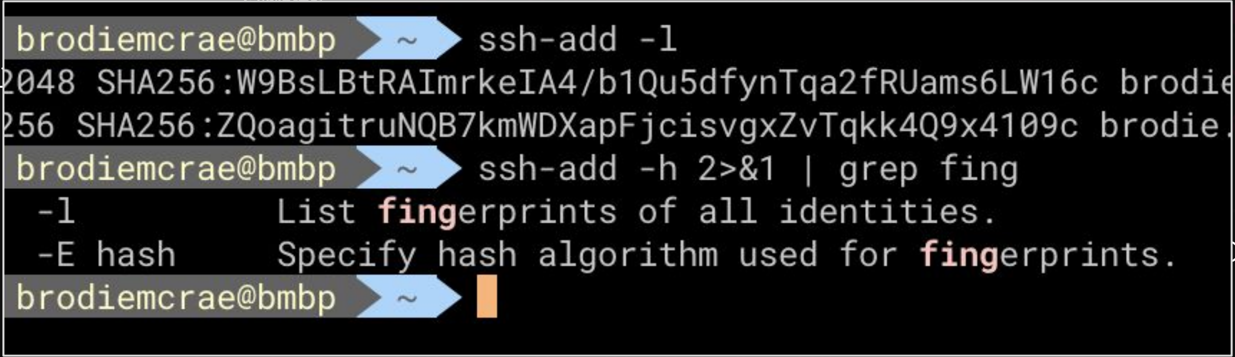
Analogy is somewhat flawed because the output should never resemble (or give any information about) any part of the input.

Terms

Hashing

Th

-
-
- or a



Analogy is somewhat flawed because the output should never resemble (or give any information about) any part of the input.

Hashing - SHA1 example

Converting any piece of data, using an algorithm, into a typically much smaller - but “unique” - identifier.

```
$ echo “OWASP2019” | shasum
8ae44490361a675416ab94c7a35e1456e32f9601
$ echo “OWASP2020” | shasum
a70065412784c4630153c3e66cf73bc65bd19dc3
$ echo “OWASP2020.” | shasum
0550ae6fb0547d362cb562bf8bdd551c7f8b413c
```

Hashing - SHA1 example

"unique" refers to *collision resistance*

Should be *extremely* hard to find more than one input that results in the same output.. Like, so hard it would take Google's idle compute *weeks* to find one.

Small changes to input should make significant, cascading changes to the output.. But make sure your padded input and output block *sizes* are different.

Hashing - SHA1 example

What about output size > input size, like these?

“OWASP2020”	fae3ed2ad31b7cf577932318c6732dce28cdea6c
“OWASP2021”	a70065412784c4630153c3e66cf73bc65bd19dc3
“.”	a5d5b61aa8a61b7d9d765e1daf971a9a578f1cfa

Block padding

Hashing - SHA1 example

What about output size > input size, like these?

“OWASP2020”	fae3ed2ad31b7cf577932318c6732dce28cdea6c
“OWASP2021”	a70065412784c4630153c3e66cf73bc65bd19dc3
“.”	a5d5b61aa8a61b7d9d765e1daf971a9a578f1cfa

Block padding

Hashing - SHA1 example

What is this output, anyway?

"OWASP2999"	6a652a0badd7706dea07361cdccdfba9a36b0615
"OWASP3000"	70809fa061004b0297ca7f7503347cb005c9cb94
"OWASP3001"	0550ae6fb0547d362cb562bf8bdd551c7f8b413c

Hashing - SHA1 example

What is this output, anyway? Encoded as hex:

“OWASP2999” 6a 65 2a 0b ad d7 70 6d ea 07
36 1c dc cd fb a9 a3 6b 06 15

SHA-1 output is 160 bits 0110101001100101 …
20 bytes
40 hex char 6a 65 …

Hashing vs Encryption

Use cases

- Hashing is used to validate data
  - Message integrity - paired with original value
  - Passwords - don't need original value

- Encryption is used to keep data secret
  - Keep data safe in transit
  - Stored data that is stolen cannot be read

Hashing + Encryption

- Protect encrypted data from being tampered with

- **Encrypt-then-Auth**
- **Auth-then-Encrypt**
- Which is best? Arguments for both

- Bonus term: MAC "message auth codes"

Data in Transit

Secure communication: establishing a temporary channel for comms.

Often “signed” keys for auth, then *shared* keys for the session.

Data at Rest

Secure storage. To use an analogy:

transit Armored truck moving valuables between banks

rest Storing valuables in a safe

Key Exchange

So, how can two parties set up a secure channel when someone in the middle is listening to everything?

It turns out, there are novel ways to exchange secrets through/despite an intermediary.

Key Exchange in Abstract

Say Alice wants to send a super secret message to Bob in a secure channel.

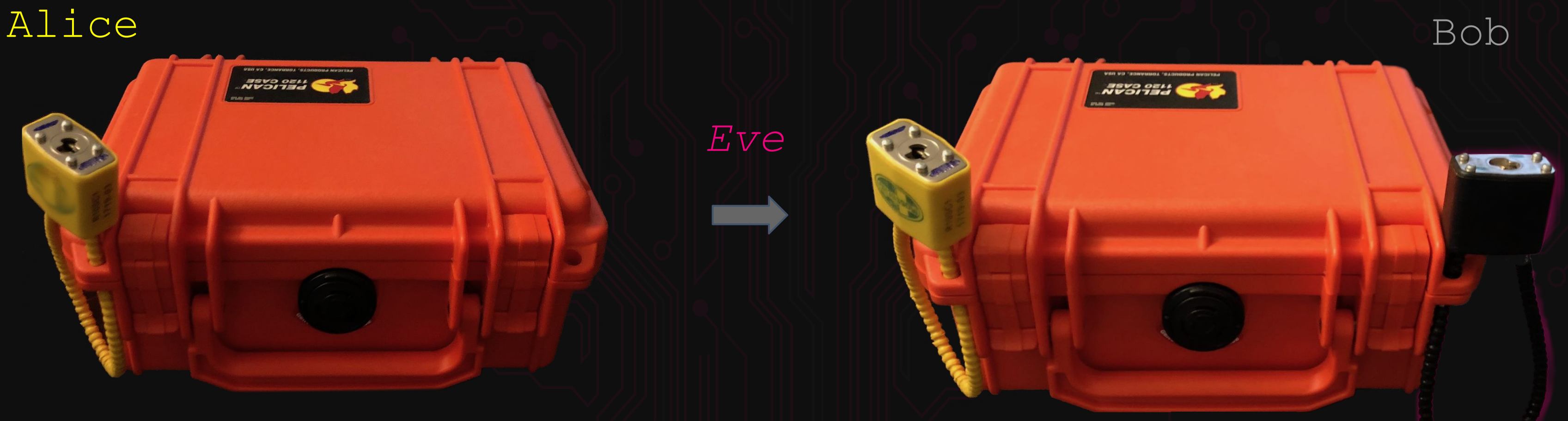
Key Exchange in Abstract

Alice seals her heartwarming words with her lock.

Alice

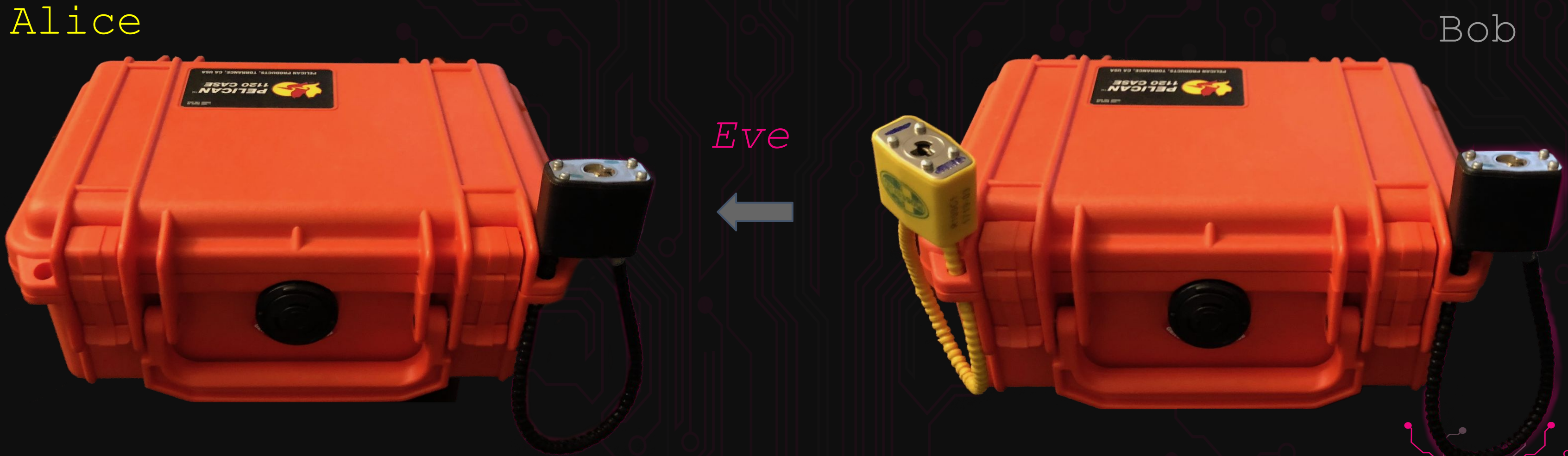
Key Exchange in Abstract

Bob can receive the sealed message, and apply his own lock.



Key Exchange in Abstract

Bob sends the message back, and Alice removes her lock.



Key Exchange in Abstract

Finally, Alice sends the message back to Bob, who removes his lock:

Bob

Bob's heart is thusly warmed.

Key Exchange: Diffie-Hellman

The preeminent example of key exchanges.

To explain DH, we need to touch on something.
Something dark. We need more math.

DH 101: Modular arithmetic 1/3

Modulus (think remainders)

24hrs to 12am/pm:

1619 hrs mod 1200

= 419 (pm)

DH 101: Modular arithmetic 2/3

Pop quiz: what's a logarithm?

Inverse of an exponent

- Exponentiation

$$10^3 = 1000$$

- Logarithm

$$\log_{10}(1000) = 3$$

Given a base (10) and an exponent (3), it's really easy to compute a result (1000) *modulo some prime number*.

Given a modulo prime result of some known base and *secret* exponent, it's extremely hard to determine the original exponent.

It can be said that this problem cannot be solved in polynomial time.

DH 101: Modular arithmetic 3/3

Logarithms

- Original calc

  $10^3 \bmod 13 = 12$

- Logarithm

  $\log_{10}(?) = ?$

Crux: What possible exponents, mod13, have a remainder of 12?

This is called the *discrete logarithm problem*.

For a large prime modulus value, the inverse computation is prohibitively expensive. Hard to calculate, easy to verify. This property forms the basis of cryptography.

Given a base (10) and an exponent (3), it's really easy to compute a result (1000) *modulo some prime number*.

Given a modulo prime result of some known base and *secret* exponent, it's extremely hard to determine the original exponent.

It can be said that this problem cannot be solved in polynomial time.

Diffie Hellman Walkthrough

Wikipedia kinda says it best. Alice and Bob want to share a secret. They agree *publicly* to use a prime number, 23, and a base, 5.

Alice chooses a secret value: 6
Bob chooses a secret value: 15

Alice calculates 5^6 **mod** 23 = 8
Bob calculates 5^15 **mod** 23 = 19

Alice and Bob share 8 and 19 with each other.

Diffie Hellman Walkthrough

Public prime, 23, and base, 5

Alice's secret: 6

Bob's secret: 15

Step 1:

Alice: 5^6 **mod** 23 = 8

Bob: 5^15 **mod** 23 = 19

Alice and Bob exchange 8 and 19, publicly, then:

Step 2:

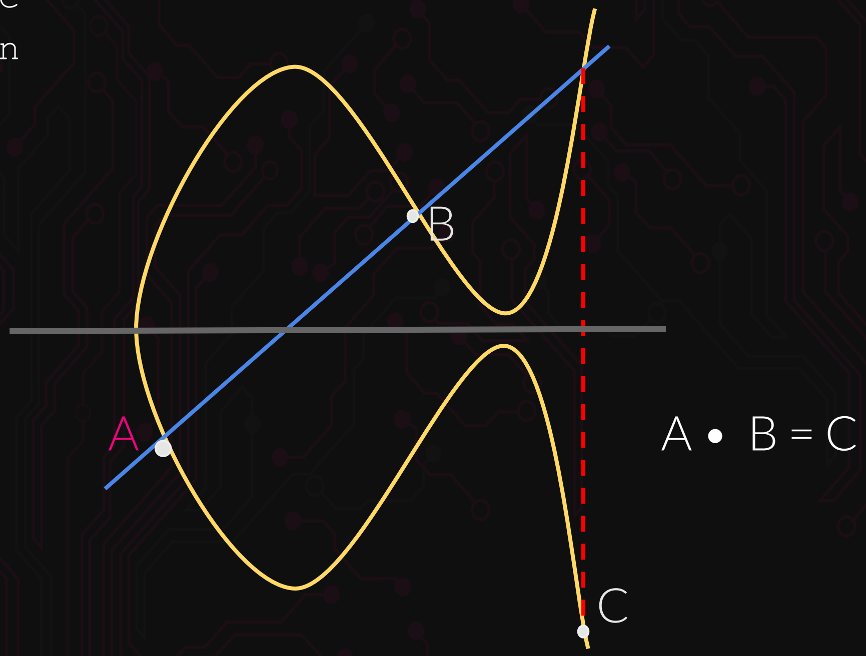
Alice: 19^6 **mod** 23 = 2

Bob: 8^15 **mod** 23 = 2

$$\left(5^{a} \bmod 23\right)^{b} \bmod 23$$
$$=\left(5^{b} \bmod 23\right)^{a} \bmod 23$$

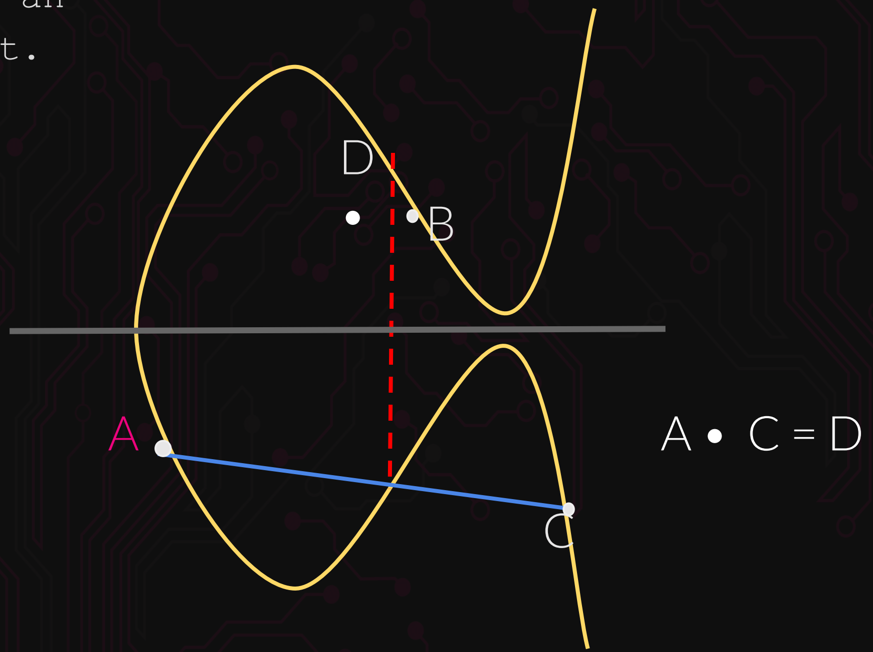
Elliptic Curves

A line through an elliptic curve will intersect with the curve in three places.



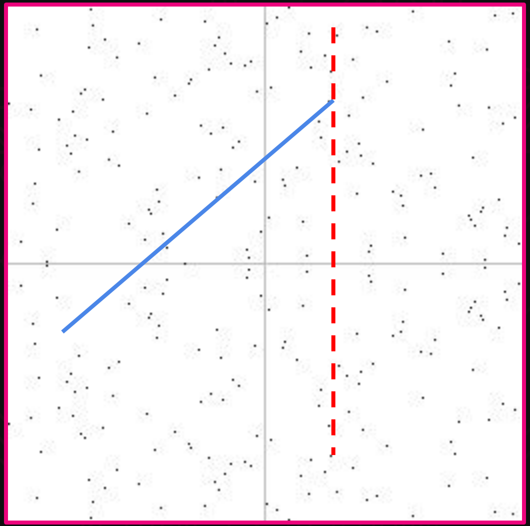
Elliptic Curves

The result - C - can be used as an input into finding another point.



Elliptic Curves

Typical "curve," like P-256:



I told you, sexy

Elliptic Curves

OK, but why curves?

e.g., logjam – “export grade” DH

Equivalent strength SSH Keys:

RSA: 2048 bits

ECDSA: 256 bits

Elliptic Curves

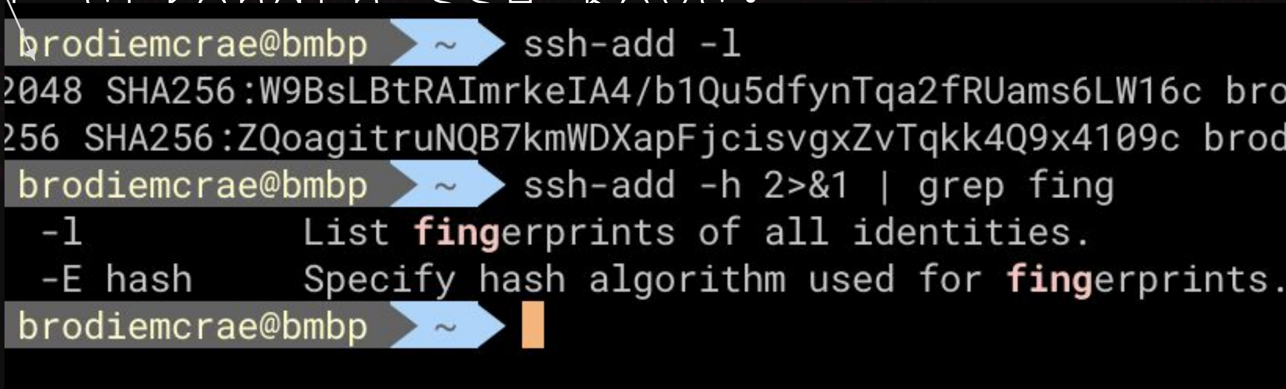
OK, but why curves?

e.g., logjam – “export grade” DH

Equivalent strength SSH Keys:

RSA:

ECDSA:



Elliptic Curves

Curves are used for encryption and for DH key exchanges because secret starting points are ***hard*** to derive.

X25519 is the DH exchange based on my fave, **Curve25519**

(So-named because it uses Prime $p = 2^{255} - 19$)

Curve25519

- Standardized in TLS 1.3
- DNSCrypt
- OpenSSH
- Signal
- AirPlay
- iOS
- Android
- OpenBSD
- Tor

P-256 and others like secp256k1

- Ethereum
- Bitcoin
- Government backdoors *(prolly?)*

Protip: Just use whatever Microsoft backs

The Register®
Biting the hand that feeds IT

NTRE SOFTWARE **SECURITY** DEVOPS BUSINESS PERSONAL TECH SCIENCE

Security

Microsoft throws crypto foes an untouchable elliptic curveball

Redmond's new, free, crypto library dubbed FourQ leaves P-256 swinging and missing

Kidding aside I think cloudflare rolled a sweet Go implementation, 3x + perf .. so ..

Symmetric vs. Asymmetric

- Symmetric: same key is used to encrypt and decrypt (in transit, the temporal session key. at rest, typically a fixed encryption key)

- Asymmetric: private key used to decrypt/sign, public key used to encrypt to person that holds private key, and validate messages *from* said keyholder.

TLS and HTTPS

Common misconception that TLS uses private/public keypair for session encryption. Good implementations authenticate with these keys and then negotiate (and frequently cycle) throwaway session keys. Related: "Perfect forward secrecy"

Typical:

RSA 2048-bit server key, signed by a CA

AES 256-bit session key for cipherstream

“cipher suite” deconstruction

ECDHE-RSA-AES128-GCM-SHA256

SHA 256 *HMAC*

Galois Counter Mode

AES cipher

RSA key+signature

Ephemeral

Diffie-Hellman

Elliptic Curve

TLS: browsers, websockets..

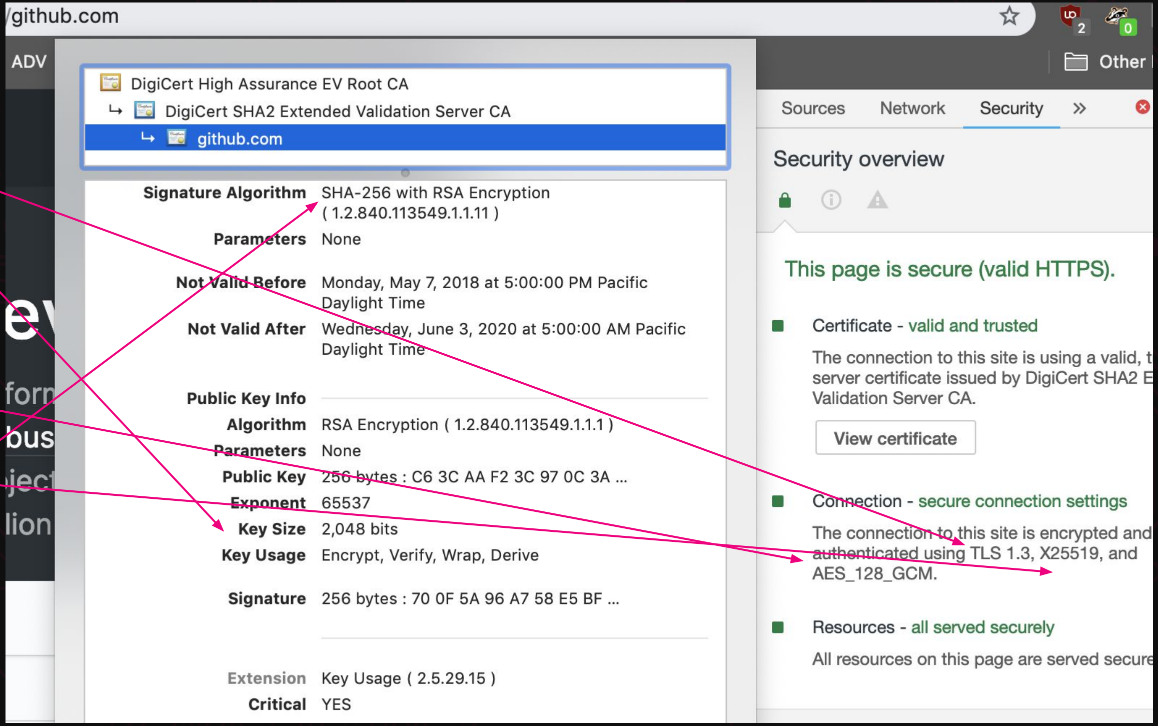
Protocol: TLS 1.3

Private Key: RSA 2048 bit

Encryption: AES w/Counter mode

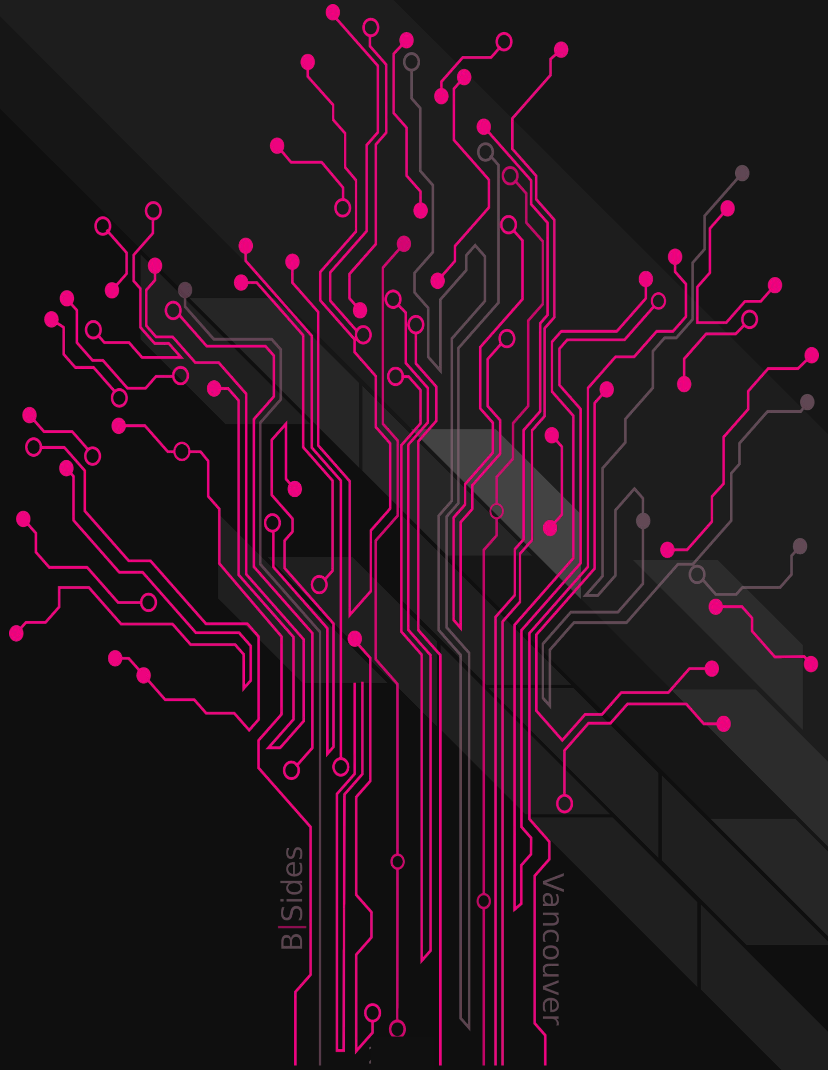
Exchange: X25519

Signatures: SHA-256



Questions

This slide is here because of my unbending optimism



Bonus reading

- Dive deeper into applied crypto
  - Crypto101 (lvh) - bit deeper
    - crypto101.io
  - Graduate applied cryptography - lot deeper
    - crypto.stanford.edu/~dabo/cryptobook/
  - Nigel Smart's UMD intro crypto
  - I've heard coursera.org/learn/crypto is good?

- Guidelines
  - safecurves.cr.yp.to
  - cipherli.st
    (reference TLS config guides)
  - github.com/ssllabs/research