Cryptographic Deep Dive

Blockchains & Applications

G. Chênevert

February 21, 2024

Cryptographic Deep Dive

Blockchains as protocols

Hash functions

Elliptic curve signatures

Blockchains as protocols

Blockchains may be viewed as a communication protocol among mutually distrustful parties that which to agree on a mutual distributed, shared journalized ledger.

As such, it prevents:

- modification of events in the past (immutability)
- unauthorized additions (certificate signature)
- invalid additions (validation)
- rogue validating nodes (replication)
- incoherent states (consensus mecanism)

Security point of view

From a cryptographic point of view, no exploit is ever absolutely impossible...

But we want to make either *extremely unlikely* or *impractically long*

Example

Trying to decrypt data encrypted using AES with a 128-bit secret key

. . . without knowing the key

Guessing the key

The attacker could pick a random key and decrypt the data with it.

$\implies$ nonzero probability of success!

However: since there are 2^{128} unique keys, the probability of picking the right one on first try is $\frac{1}{2^{128}}$.

You may have a better chance of quantum tunneling through a wall!*

(* should check with a physicist though)

Brute-force attack on the key

The attacker may try all the possible key until they find the right one.

How long would it take?

Assuming we could try 5.6×10^{18} keys per second (current performance of the Bitcoin mining network):

that's $\frac{2^{128}}{5.6 \times 10^{18}} \approx 2^{66}$ seconds

so roughly 128 times the estimated age of the universe (!)

Cryptographic building blocks

Used by all blockchains:

- hash functions
- signatures

Used by some blockchains:

- encryption (to protect payloads)
- zero-knowledge proofs (Monero)
- ...

Cryptographic Deep Dive

Blockchains as protocols

Hash functions

Elliptic curve signatures

Hash functions

Definition

A **cryptographical hash function** is a *collision-resistant* deterministic function H with fixed-size outputs.

Example

SHA256 maps any input to a string of 256 bits $=$ 32 bytes $=$ 64 hex digits

Collision resistance

We want that it's (in practice) impossible to find distinct inputs x and y such that $H(x) = H(y)$.

(It is however always possible, in theory, to find such collisions!)

In particular, it should be hard to solve the **second preimage problem** :

given y, find $x \neq y$ such that $H(x) = H(y)$.

In particular, it should be hard to solve the **preimage problem** :

given h, find x such that $H(x) = h$.

Exercise (hard?)

Can you manufacture collisions for the `FastHash` function from TD1?

Second preimages??

Preimages for arbitrary h???

Authenticated data structures

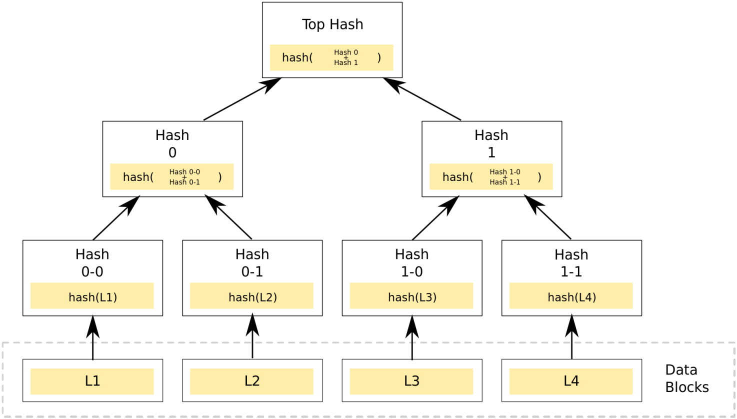
Any hash function can be used to turn an (acyclic) data structure into an authenticated data structure : for every pointer, add the hash of the child element to the parent element.

If it's not possible to find second preimages : the contents of the children cannot be changed without recomputing the hashes in all the parents.

Authenticated linked list : basis for a blockchain

Merkle tree

Authenticated binary tree



Merkle trees in blockchains

- used to efficiently compute the top hash of a list of certificates that summarizes them inside a block

  (adding of verifying an entry costs $\log_2(n)$ hash computations instead of n)

- and to compress the chain into a simplified version by removing everything except the root hash

  (old transactions can still be verified by looking up the relevant branch of the Merkle tree)

Also used e.g. by BitTorrent and Git

Commitments

One-way (preimage resistant) hash functions also provide a way to *commit* data without revealing it

A: I'm thinking about a fruit

B: It is a banana?

A: Yes

B: How can I be sure you didn't change it?!

Commitments

A: I'm thinking about a fruit

(`b493d48364afe44d11c0165cf470a4164d1e2609911ef998be868d46ade3de4e`)

B: It is a banana?

A: Yes

B: Ok that works out

Application : Patents

Anchoring documents to the blockchain to provide proof of anteriority

A: I invented this gizmo two years ago

B: ...

A: Here is a PDF file I wrote to describe it

B: ...

A: The date in the document says 2022

B: ...

A: And I stored its hash in a block at that time

B: Ok I can see that!

Cryptographic Deep Dive

Blockchains as protocols

Hash functions

Elliptic curve signatures

Signature algorithms

Building block of certificates:

- Alice signs a message (transaction) m with her private key k_{priv}

  i.e. computes $s = \text{Sign}(k_{\text{priv}}, m)$ and appends it to m

- Anyone (validating node) can then verify whether the signature is correct using the corresponding public key:

$$\text{Verify}(k_{\text{pub}}, m, s) \in \{\texttt{true}, \texttt{false}\}.$$

Requirements of a signature algorithm

- Correct verification:

$$\text{Verify}(k_{\text{pub}}, m, \text{Sign}(k_{\text{priv}}, m)) = \texttt{true} \qquad \text{for all} \quad m$$

- Non-forgery

  impossible in practice to manufacture a valid signature for a (new) message m without access to k_{priv}

  In particular: not possible to recover k_{priv} from k_{pub}.

- Consequence: non-repudiation

  If Alice keeps k_{priv} private and a valid signature for k_{pub} is encountered, it means without reasonable doubt that she did sign (a signature is binding)

Desirable properties of a signature algorithm

- Efficiency:

  the Sign and Verify algorithms should be reasonably fast

- Signature conciseness:

  the produced signatures s should be reasonably small

- Key conciseness:

  the private and public keys k_{priv} and k_{pub} should not be too large

- Efficient key generation:

  should be easy to come up with new pairs $(k_{\text{priv}}, k_{\text{pub}})$

  e.g. reusable parameters

RSA signatures

- $n = p \cdot q$ product of two (large) distincts prime numbers
- d and e two integers such that $de \underset{\varphi}{\equiv} 1$ where $\varphi = (p-1)(q-1)$
- $k_{\text{priv}} = (d, n)$
- $k_{\text{pub}} = (e, n)$
- $\text{Sign}(k_{\text{priv}}, m) \underset{n}{\equiv} H(m)^d$ where $H = \text{SHA-256}$
- $\text{Verify}(k_{\text{pub}}, m, s) = \left[s^e \underset{n}{\equiv} H(m)\right]$

based on a consequence of Fermat's theorem: if $de \underset{\varphi}{\equiv} 1$ then $h^{de} \underset{n}{\equiv} h$ for all h

The problem with RSA signatures

To be secure by today's standards, n (and d, e) must be at least 3072 bits long

$\implies$ the keys are 768 bytes long

(in particular, the 256-bit hash must be padded to avoid weaknesses)

And tomorrow (5-10 year time window): 15360 bits = 3.8 kB keys :(

Parameter reuse?

Suppose Alice and Bob both use

$$n = \texttt{0x1bb85a72024a0365b78cfd0bdef8a63f943fb291899a91a713}.$$

Bob knows:

$$e_A = \texttt{0x1af2918de90624fd9f7de44c3d654b179a091a9961336d85e1}$$
$$e_B = \texttt{0x45d9a596bdbc35c3c24dbacf68a588bde5cb8538a120ab07f}$$
$$d_B = \texttt{0x94064f06cbaabbd280ec708999254b23d9a5022a0f32526f}$$

Show he's easily able to steal all of Alice's tokens.

Hint: Fermat's trick

Bob knows $c := d_B e_B - 1$ is a multiple of $\varphi = (p-1)(q-1)$.

So: for every integer a we have

$$a^c \underset{n}{\equiv} 1 \qquad \text{(Fermat's theorem)}$$

Write $c = 2^t r$ where r is odd (take out powers of 2) and try different values of a until one is found for which

$$a^r \underset{n}{\not\equiv} \pm 1, \qquad a^{2r} \underset{n}{\equiv} 1.$$

We can then factor the congruence as

$$(a^r - 1)(a^r + 1) \underset{p \cdot q}{\equiv} 0$$

and we find

$$p = \mathrm{GCD}(a^r - 1, n), \qquad q = \mathrm{GCD}(a^r + 1, n).$$

Fermat attack

```python
from math import gcd
from random import randrange

n = 0x1bb85a72024a0365b78cfd0bdef8a63f943fb291899a91a713
eA = 0x1af2918de90624fd9f7de44c3d654b179a091a9961336d85e1
eB = 0x45d9a596bdbc35c3c24dbacf68a588bde5cb8538a120ab07f
dB = 0x94064f06cbaabbd280ec708999254b23d9a5022a0f32526f

# find odd exponent r

r = eB*dB - 1

```

Evaluate

Language: Python

Share

```
prime factors found 0xb35fef3e4e1779ff86d2c3a0b 0x278fc589159cf1718ffb97419

Alice's private key is 0x4dec3a340aa923355f984d369bfe5091ad7e60a7f3e7985c1
```

Timeline of signature algorithms

- 80’s - 90’s: RSA, DSA (integer-based)
- 00’s - 20’s: ECDSA, EdDSA (**elliptic curve**-based)
- 30’s - ??’s: quantum-resistant signatures (lattice or code-based)

Elliptic curve-based signatures:

- 256-bit keys (tomorrow: 512)
- parameter reuse possible

Elliptic curves

Definition

An *elliptic curve* is a plane curve defined by an equation of the form

$$\mathcal{E}:\; y^2 = f(x)$$

where $f(x)$ is a cubic polynomial with distinct roots.

- $f(x) = x^3 + ax + b$ with $4a^3 + 27b^2 \neq 0$: **Weierstrass form**
- $f(x) = x^3 + Ax^2 + x$ with $A^2 \neq 4$: reduced **Montgomery form**

Some famous elliptic curves

Secp256k1 (Bitcoin, Ethereum)

$$y^2 = x^3 + 7$$

Curve25519 (Monero, Zcash, . . .)

$$y^2 = x^3 + 486662x^2 + x$$

Addition on an elliptic curve

Given $P, Q \in \mathcal{E}$, the line through P and Q intersects $\mathcal{E}$ at a third point $R = (x_R, y_R)$.

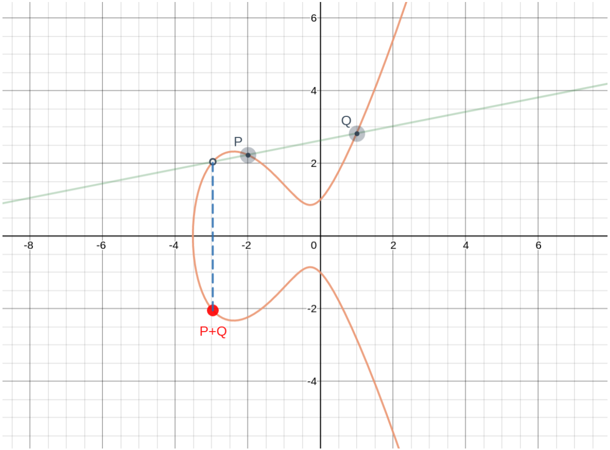
Definition

$$P + Q := (x_R, -y_R)$$

Fact: This makes $\mathcal{E} \cup \{O\}$ into an abelian group

(The *point at infinity* $O = (0, \infty)$ being the neutral element)

Addition on an elliptic curve



Addition formulas

Line through $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$:

$$y = y_P + \lambda(x - x_P) \qquad \text{with} \qquad \lambda = \frac{y_Q - y_P}{x_Q - x_P}.$$

To find the third point, we solve the degree 3 equation:

$$\big(y_P + \lambda(x - x_P)\big)^2 = x^3 + ax + b.$$

Write this as

$$x^3 - \lambda^2 x^2 + \ldots = (x - x_P)(x - x_Q)(x - x_R),$$

so $x_R = \lambda^2 - x_P - x_Q$, and $y_R = y_P + \lambda(x - x_P)$.

Special case of doubling

When $P = Q$: the line through P and Q must be interpreted as the tangent to $\mathcal{E}$ at P.

$$y^2 = x^3 + ax + b$$

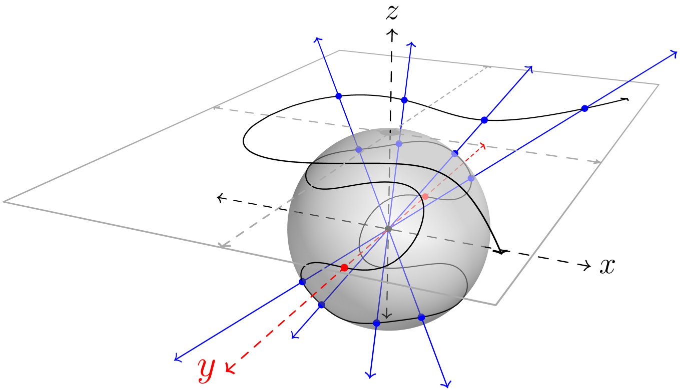
$$2y\frac{\partial y}{\partial x} = 3x^2 + a$$

$$\implies \lambda = \frac{3x_P^2 + a}{2y_P}$$

and the rest is the same.

To make more sense of the point at infinity

We should put the elliptic curve in perspective



(source)

Projective coordinates

We think of the xy-plane as sitting in 3-space as the $z = 1$ plane.

A radial line $(X : Y : Z)$ through (X, Y, Z) intersects $\mathcal{E} \iff (\frac{X}{Z}, \frac{Y}{Z}, 1) \in \mathcal{E}$

$$\frac{Y^2}{Z^2} = \frac{X^3}{Z^3} + a\frac{X^2}{Z^2} + b$$

$$\implies Y^2Z = X^3 + aX^2Z + bZ^3 \qquad \textbf{homogeneous equation}$$

$Z = 0$: find a single line $(0 : 1 : 0) = O$ the point at infinity

Hard problem on an elliptic curve

Given a point $G \in \mathcal{E}$ and n an integer, it is easy (fast) to compute

$$P = nG = \underbrace{G + \cdots + G}_{n} \quad \text{in} \;\; \mathcal{E}.$$

However, given P and G, it is in general difficult (long) to recover n

(**discrete logarithm problem**)

provided we work over a finite field

Elliptic curves over finite fields

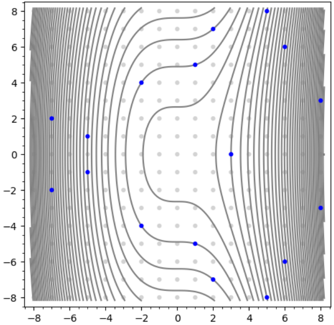
Consider solutions modulo a fixed prime p

$$y^2 \underset{p}{\equiv} x^3 + ax + b$$

$\rightsquigarrow \mathcal{E}(\mathbb{F}_p)$ elliptic curve over the field with p elements

(a finite abelian group)

$$y^2 \underset{17}{\equiv} x^3 + 7$$



Example computations

```
E = EllipticCurve(GF(17),[0,7])

P = E([-5,1])
Q = E([8,3])

P + Q
```

Evaluate

Language: Sage

Share

```
(10 : 15 : 1)
```

Size of $\mathcal{E}$

Theorem (Hasse bound)

$$\#\mathcal{E}(\mathbb{F}_p) = 1 + p + \mathcal{O}(\sqrt{p})$$

hence $\#\mathcal{E}(\mathbb{F}_p) \approx p$.

We use elliptic curves with points G of large additive order $n \approx p$.

Secp256k1 (more precisely)

Weierstrass curve with $a = 0,\ b = 7$

$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$

base point G with

$$x_G = \texttt{79be667e f9dcbbac 55a06295 ce870b07}$$
$$\texttt{029bfcdb 2dce28D9 59f2815b 16f81798}$$
$$y_G = \texttt{483ada77 26a3c465 5da4fbfc 0e1108a8}$$
$$\texttt{fd17b448 a6855419 9c47d08f fb10d4b8}$$

of additive order

`ffffffff ffffffff ffffffff fffffffe baaedce6 af48a03b bfd25e8c d0364141`

Curve25519 (more precisely)

Montgomery curve with $A = 486662$

$p = 2^{255} - 19$

base point G with

$$x_G = 9$$

(y coordinates not needed)

A widely used, efficient *de facto* standard curve since 2013.

ECDSA

Public parameters : an elliptic curve $\mathcal{E}$ with base point G of order n

Private key : a random integer $d \in [\![0, n]\!]$

Public key : $P = dG$

To sign a message m :

- Choose random $k \in [\![0, n]\!]$
- Compute $(x, y) = kG$ on $\mathcal{E}$
- Compute $z \underset{n}{\equiv} k^{-1}(H(m) + xd)$ where H is a suitable hash function
- Signature is the pair $s = (x, z)$

Verification : check whether $x = ((z^{-1}H(m))G + (z^{-1}\ x)P)[1]$

Schnorr signatures

Public parameters : an elliptic curve $\mathcal{E}$ with base point G of order n

Private key : a random integer $d \in [\![0, n]\!]$

Public key : $P = dG$

To sign a message m :

- Choose random $k \in [\![0, n]\!]$
- Compute $t = H(kG \parallel m)$ and send $s = k - dt$ and t.

Verification : check whether $H(sG + tP \parallel m) = t$.

Special case

EdDSA : Edwards-curve Digital Signature Algorithm

A modification of Schorr signatures using *twisted Edwards curves*.

Ed25519 : EdDSA signature using a twisted Edwards curve associated to Curve25519

Project idea : add EC signature support to your blockchain?

(You have all you need here to implement ECDSA)

Thoughts on future security

- Security levels of all cryptographic primitives **will** have to improve over time
- Blockchains are still young procotols that have yet to withstand a major security level upgrade
- Signatures are computed as certificates and blocks are added: suffices to increase security level over time

  (so no major worry about quantum computers and such)
- Exploitable weaknesses in the hash function however would be fatal: two blocks with the same root hash are perfectly interchangeable for all intents and purposes...