Blockchain

1. *What is Blockchain*?

   Blockchain is a Distributed Database.

   It is:

   - Append only
   - Transparent
   - Incorruptible (under mild assumptions)
   - Secure
   - Time-stamped
   - With Distributed Consensus

2. *Distributed Consensus*

Financial Arrangements

Barter

Simple enough: If A has b but wants a and B has a but wants b then the two can swap with each other. What is A has c and wants a, but B has a but wants b? We look for C who has b and wants c, and then we can arrange for an exchange.

The issue:

Getting the people to get together and arrange an exchange.

Solution:

1. Cash
2. Credit

Credit Make the transaction, be in debt until repayment after.

Cash Denominating some cash value to all goods, and using cash to buy and sell.

Probability

- Basic Rules
- Conditional Probability
- Independence
- Mutually Exclusive

- Birthday Paradox

  Central Idea: 1-P(no common birthdays), (1-k/d) < exp(-k/d)

- Random Variables
  - Binomial Random Variable:

    P(X=k) = nCk p^k (1-p)^(n-k)

  - Geometric Random Variable: P(X=k) = p(1-p)^(k-1)
  - Poisson Random Variable: (lambda.t)^n exp(-lambda.t)/n!

- Expectation E(X) = Sum(x__i.p__i)
  - Binomial: np
  - Geometric: 1/p
  - Poisson: lambda.t

Number Theory

Group

- (G, *): closure, identity, inverse, associative
- Generator g
- Zn = {x: x = N mod n}
- (Zn, +) is a group
- Zn - {0] = Zn*
- (Zp,) is a group
- phi(n): totient function

  Number of co-primes captured by n

  - phi(p) = p-1
  - phi(pq) = (p-1)(q-1)

Public Key Crypto

RSA

Take: p, q, n = pq, phi(n) = (p-1)(q-1)

select e and d such that ed = 1 mod phi(n)

Public: (e, n)

Encryption c = m^e mod n

Decryption $m = c^d \bmod n$

El Gamal

Take: Z*p, Generator g, random x.

Calculate: h st $h = g^x \bmod p$

Public: (h, p)

Encryption: Take random y

$s = h^y \bmod p$

$c1 = g^y \bmod p$

$c2 = ms \bmod p$

Decryption: $inv(c1^x).c2 \bmod p$

Cryptographic Hash Functions

Properties:

- Deterministic
- Efficient to compute
- Pre-image resistance
- Second pre-image resistance
- Collision resistance
- Small change in the input should modify hash extensively
- Fixed side output, for input of any size

SHA-256

Merkle-Damgard transform is used by SHA-256 to keep it to 256 bits.

Padding is 10 * | len

Commitments

Committing

Have a message to commit, and a nonce.

com = commit(message, nonce)

Verifying

The message and nonce(key) are revealed. Verified as:

message == verify(com, message, nonce)

Digital Signatures

Properties:

- Analogous to Physical Signatures
- Should not be possible to forge onto other documents
- Signer should not be able to deny signing

We have sign(), verify(), keygen(). keygen gives us sk(secret key) and pk(public key)

keygen should be random. sign should be deterministic

Signing

sig = sign(sk, message)

Verifying

verify(pk, sig, message) == true

RSA

- p, q, n=pq, phi(n), e, d st ed=1 mod phi(n)
- d is private and e is public. d is used to sign, e to verify

Sign-Verify

- sig: m^d mod n
- verify: s^e == m mod n

El-Gamal

- p, g, x (random), h (from x,g,p).
-

DSA