COMP4901W - Introduction to Blockchain, Cryptocurrencies and Smart Contract Spring 2023

Taught by Amir Gohashady

Notes by Marcus Chan

May 23, 2023

Contents

1 Lecture2

1.1 Properties of hasing:

1. collision-resistant: $h(x) \neq h(y)$ for $x \neq y$
2. hiding: can't find x s.t. $h(x) = y$

1.2 Applications:

1. finding files
2. ledger with pointers
3. commitment scheme

   bidding protocol: for security reasons

   (a) highest bid can be found
   (b) no player can change the bid after seeing others' bid
   (c) auditability (i.e. auditor won't change the deals)

   steps:

   (a) compute $h(b_i + n_i)$ for each player and choose a random number n_i from large domain
   (b) player publishes the hash (commit)
   (c) player publish the bid and n_i for others to hash and verify (reveal)

2 Lecture3

2.1 Merkle tree

Protocol:

1. reclaim once
2. message is short(const)
3. deposit can be taken back
4. message doesnt leak
5. proof p_i is provided and can be decoded

3 Lecture4 - Symmetric Encrpytion

3.1 Definition

let a key $k \subseteq \sum^*$ which is known by both players. A knows

$$ENC_k k \subseteq \sum^*$$

and B knows

$$DEC_k : k \subseteq \sum^*$$

$\forall m \in \sum^* DEC_k(ENC_k(m))$

3.2 Onetime pad

let encoded message e; let original message m;

$$e = m \oplus k.$$

3.3 Security analysis

Combination = $e^{|n|}$ where n = length of key. Problem with multiuse: suppose we have m_0 and m_1, then eavsdropper can do:

$$m_0 \oplus m_1 = k \oplus k \oplus e_0 \oplus e_1 = e_0 \oplus e_1.$$

which in turn some information is leaked $\implies$ not so secured

3.4 Key exchange

Let players p_0 and p_1. Both have their own message m_0 and m_1. Both players then compute $f(m_i)$. Our task is to compute secret k such that it is easy to compute and impossible to compute using individual secrets m_i

3.5 Algorithm: Diffie-Hellman Exchange

1. find a large prime p and $g \in \{0, \ldots, p{-}1\}$ such that $\{g^0, g^1, g^2, \ldots, g^{p-1}\} = \{0, 1, 2, \ldots, p-1\}$

2. p_1 chooses a secret a from $\{0, 1, 2, \ldots, p-1\}$. similarly for p_2
3. p_1 computes $g^a \% p$ and send to p_2. Similarly for p_2
4. p_1 computes $g^b * a = g^{ab}$

4 Lecture 6 - The RSA Cryptosystem

A key pair (e, d) where e is the public key and d is the private key. Choose also n ∈ N. $\forall m Dec_d(Enc_e(m)) = m$

$$Enc_e(m) = m^e \text{mod n} \tag{1}$$

$$Dec_d(m') = (m')^d \text{mod n} \tag{2}$$

$$Dec_d(Enc_e(m)) = m^{ed} = m(\text{mod n}) \tag{3}$$

4.1 Key generation - failed attempt

What happens if n is a prime?

$$\forall m m^{ed} = m \iff \forall m^{ed-1} = 1.$$

Then we choose e,d s.t. $(n-1)|(ed-1)$
$\implies ed - 1 = 0(\text{mod n-1}) \iff ed = 1(\text{mod n-1})$
$\implies d = e^{-1}$

4.1.1 Security analysis:

Eavsdropper can see n, e, m^e, which he can compute $d = e^{-1}$(mod n-1). He can decrpyt the message to be m^{ed}

4.2 Key generation - successful attempt

1. Choose two large prime: p,q and n = p * q
2. Generate d,e s.t. $\forall m m^{ed} = m$ (mod n), which is the same as mod p and mod q. $\iff$ $m^{ed-1} = 1$ (mod p or mod q)
3. Make sure $l = ed - 1$ is the multiple of both $p - 1$ and $q - 1$

$$l = LCM(p-1, q-1).$$

4.2.1 Security analysis

Similar to above, eavsdropper see e,n,m^e, $d = e^{-1}$ (mod l). He cannot find p,q,l = lcm(p-1,q-1), d,m, meaning he cannot decrpyt the message

5 Lecture 7 - Digital Signatures

5.1 Homomorphic property of RSA

$$Enc_e(m_1) * Enc_e(m_2) = Enc_e(m_1 * m_2).$$

5.2 Digital signature

A signature function: sgn_d: $\sum^* \to \sum^*$. A verification function: ver_e: $\sum^* * \sum^* \to \{0,1\}$ or $\overline{ver_e}$: $\sum^* \to \sum^*$ if eavsdropper knows m,e, sgn, ver, he should not be able to compute $sgn_d(m)$ without d

5.3 RSA signature

1. Compute RSA keys: Find two large primes p,q. $n = p * q$. Find e,d, s.t. $\forall m m^{ed} = m$ (mod n-1)
2. Assume everyone knows n and e
3. p_1 sign the message with sgn_d
4. p_2 verify the message with $\overline{ver_e}$ s.t. $\overline{ver_e} = m$

5.3.1 Security analysis

Eavsdropper knows m, m^d, n, e. He cannot forge a signature meaning he doesn't know d s.t. he can compute m'^d Also, even with the homomorphic property, $sgn_d(m_1) * sgn_d(m_2) = sgn_d(m_1 * m_2)$ which doesn't make sense

6 Lecture 8 - Transactions and Double-spending

6.1 Creating a crpytocurrency (public keys as identities)

Required info:

1. sender's identity
2. Recipient's identity
3. Amount
4. Proof of ownership (hash pointer to Tx that paid the sender
5. Proof of consent (signature from the sender)

7 Lecture 9 - Centralized Ledger

Let there is a central bank B to keep track of the history of transactions **done by the users** to prevent the problem of double spending. Let block be a sequence(**Merkle tree**) of transactions of size at most b, which will be **created and published by the bank**. Problem:

1. How to know if b_i was created by the bank? Soln: The bank signs every block
2. How to ensure the bank does not change the history?

7.1 Viewpoint of a node

1. Case 1: Receive a transaction. Steps:
   (a) Verify if it is valid
   (b) input $\geq$ output
   (c) No double spending: input has happened before in the block chain
   (d) Propagate to the neighbours
2. Case 2: Receive a block B_i: Steps:
   (a) Verify signature of the bank
   (b) Validate every transactions in B_i
   (c) If B_i is valid, add it to my blockchain and send to neighbours

7.2 Viewpoint of the bank

1. Maintain a copy of the blockchian
2. Maintain a new block
3. Listen for transactions

8 Lecture 10 - Bitcoin and Proof of Work

8.1 Decentralized ledger

Definition 1 (Decentralization) *Every node has the same permissions*

How to extend the blockchain through **mining** while preserving consensus for honest nodes?

8.2 Proof of work

The first person to solve the puzzle adds the next block Criteria for the puzzle:

1. hard to solve
2. easy to verify
3. impossible to steal

Definition 2 (Nonce) *A number chosen by the miner*

Set the puzzle to be $h(B)$ is small such that for example $h(B) = 00\ldots_{60}\ldots00$. The invert the hash function, we can only try each and every nonce, which has a probability of $(\frac{1}{2})^{60}$

8.3 Viewpoint of nodes

A node keeps tack of both blockchain and mempool(ready to push to blockchain). If the node hears a new transaction, do the following verifications:

1. signatures
2. inputs $\geq$ outputs
3. no double-spending in blockchain $\cup$ memory pool

if the transaction is valid, then send the transaction to all neighbours and add it to the mempool. If a block B is valid, then add the block to the blockchain and clear the transactions in the block and transactions in conflict with B from the memory pool.

Definition 3 (Consensus chain) *The longest chain is the consensus chain because the miners get to choose the chain to extend on. Then, even if forks exist, up to certain point, there will always be a longer chain which becomes the consensus chain.*

9 Lecture 11 - Mining rewards and forks

Definition 4 (Transaction fee) $\sum$ *input* - $\sum$ *output*

Why should a miner mine a non-empty block? Miners gain transaction fees after mining blocks. The difference $\sum$ inputs - $\sum$ outputs is paid to the miner. Thus, if you want to add your transaction earlier, pay more transaction fee to lure miners.

9.1 Problem with forks

The transactions that are not in consensus chains are neglected and miners get no reward. Hence, miners are aware of the existence of forks, and will put the transactions in the chain that is less likely to be a consensus branch. This doesn't create the double-spending problem as the memory pool will only check the same chain. Besides, after being mined, the transactions in conflict will be removed from the memory pool (Lecture 10).

9.2 Double spending attack

Scenario: bit-coin based vending machine

9.2.1 First attack: Immediate double-spending

1. create one transaction T_1, with 1 BTC as the input, 0.99 as the output, and then receive the product

2. create another transaction T_2, with 1 BTC as the input, 0.95 BTC as the output, that gives money to herself, immediately, which creates a fork.

3. if the miner includes T_2 in a block, which the transaction fee is 0.05BTC $>$ 0.01 BTC, then the attack was successful, as the player can get both the money and the product.

Soln: Wait for T_1 to be mined

9.2.2 Second attack: Double-spending in a fork

Goal: Resolve and mine all the honest transactions. Problem: some blocks can be reverted (i.e. T_1) and removed. Soln: Wait for confirmation(6 blocks added after the block). Problem to soln: need to wait for an hour (not friendly to immediate transaction)

9.3 Adjusting difficulty

Let d be the constant that $h(B) < d$ where B is the block. The probability of success is $d/2^{256}$ (note: 256 bits is the max size of each block so as to uphold the latency rate) Also, we can ask the miners to check the timestamp such that if the timestamp is too old, then ignore it. (remark: the more far away, the longer the transaction chain, the harder it is to find a nonce)

Find the average mining time μ. If $\mu > 10$, then decrease d and increase difficulty. Else, increase d and decrease difficulty.

10 Lecture 12 - Centralization and Scripts

10.1 51% attack

Imagine a malicious miner with more than half of the hash power. Then, he can revert all the honest minings that follow the protocol: adding to the consensus branch, by keep on adding to the new fork and lengthen it.

10.2 Scripts

Scripts are to impose certain requirements to choosing blocks Suppose two players: A and B. They have a phone call and 1 BTC for each transaction. A and B do not trust each other. Then, we can use the following protocol, which require signatures from both A and B or signature from A only when block number > C (just to prevent the case where B doesnt sign, and A lost 100-i BTC)

1. create first despoit (input: 100, output:100)
2. for each minute i, A pays i BTC to Bob, 100 - i BTC to herself.
3. B signs the last contract and pending to be mined.

11 Lecture 13 - Two-way Payment Channels

11.1 One-way payment channel

Properties:

1. flow in one direction(A $\rightarrow$ B)
2. B will sign and close the transaction
3. The transaction expires at time t
4. The channel has a fixed capacity
5. only two transactions (base deposit, final transaction) are added to the blockchain. Thus, only two Tx fees are paid for k payments

11.2 Two-way payment channel

Problem of cheating: B published T_{x_i} but $\exists T_{x_j} j > i$ that pays less to the player, meaning the B is not honouring the refund that owes to A. Solution: Change the protocol such that B has to sign the index of every transaction. Thus, cheating now means that B's signature is on an index > current transaction's index

4 channels, 2 each one-way channel Scenario:

1. A pays 75 to B. A pay 75 in c1 (c1's capacity = 25)
2. B pays 80 to A. B refunds 75 in c1, and pay 5 in c2 (c1's capacity = 100, c2's capacity = 95)
3. A pays 50 to B. A refunds 5 in c2, and pay 45 in A(c1's capacity = 55, c2's capacity = 100)

12 Lecture 14 - Escrows and Ethereum-like Smart Contracts

12.1 Escrow

Distrustful situation: A doesnt trust B, then item first. B doesnt trust A, then payment first.

12.1.1 Awful solution

Use a trusted third-party C. Suppose C sides with P $\in \{A, B\}$, then C and P create the contract and each get 50 BTC, sign it and publish it. This leads to the bribery problem.

12.1.2 Another attempt

The money is locked when disagreement happens. Problem: extortion.

12.1.3 Solution: win-win or lose-lose

Change the deposit. If both party agree that the transaction happens, then A pay 200 to B. If both party agree that the transaction did not happen, then both A got refund 100. If disagreement happens, then both do not have any money.

12.2 Script

Why limited?

1. complete and unambiguous semantics (e.g. cpp overflow then compiler decide handling)

2. complete in $O(1)$ time

3. miner white-list script

Transactions can deploy the program (smart contract) and invoke functions of a previously-deployed smart contract (pointer to the contract, function name, parameters) Every contract can receive and manage money.

13 Lecture 5 - Basic Number Theory and ElGamal Encryption

13.1 Fermat's little theorem

Theorem 1 *If p is a prime number, then for any integer a, the number $a^p - a$ is an integer multiple of p*

Theorem 2 *If a is not divisible by p, then*

$$a^{p-1} \equiv 1 \, (a \text{ mod } p) \, .$$

Definition 5 (Primitive root) *A primitive root mod n is an integer g such that every integer relatively prime to n is congruent to a power of g mod n*

Example 2.1 (Non-primitive root) *let $p = 5$, $a = 4$.*

$$\begin{aligned} a^0 &= 1 \\ a^1 &= 4 \\ a^2 = 16 &= 1 \\ a^3 &= 4 \\ a^4 &= 1 \end{aligned}$$

Example 2.2 (Primitive root) *let $p = 5$, $a = 3$. There exists a cycle of length $i|(p-1)$.*

$$\begin{aligned} a^0 &= 1 \\ a^1 &= 3 \\ a^2 = 9 &= 4 \\ a^3 = 12 &= 2 \\ a^4 = 6 &= 1 \end{aligned}$$

13.2 Computing Primitive Roots

Task 1 *P is a prime such that $\log p \geq 1024$. Find g such that $\{g^0, g^1, \ldots, g^{p-2}\} = \{1, 2, \ldots, p-1\}$.*

We cannot simply compute $g^0, g^1, g^2, \ldots$ until a cycle is found due to large P. Hence, consider $O(g)$ = length of the cycle of the powers of g = smallest positive i such that $g^i =$ i (mod p) (recall from above examples). This implies that $O(g)|p-1$ (note: O(g) is divisible by p-1). Find all the prime factors of p-1:

$$p - 1 = q_1^{\alpha_1}, q_2^{\alpha_2}, \ldots, q_r^{\alpha_r}.$$

(note: idk why use an equal sign)

Then, we need to prove that p-1 is indeed the minimum prime such that $O(g) = p-1$. If there is a smaller prime that satisfies the above requirement, then it leads to the contradiction setting that $O(g)$ is the smallest positive i, hence $p-1$ is not a primitive root. To do so, we need to consider the cases below:

$$O(g) = p - 1 \tag{4}$$

$$O(g)|\frac{p-1}{q_1} \tag{5}$$

$$\vdots$$

$$O(g)|\frac{p-1}{q_r} \tag{r}$$

To rule out cases (2) to (r), we can simply compute $g^{\frac{p-1}{q_i}}$ (mod p). If it equals to 1, then g is not the smallest positive i but the prime factor q, hence g is not the primitive root.

13.3 Fast modular exponential

Task 2 *Compute a^b mod c*

```
exp(a,b,c): // a^b mod c
  ans = exp(a,b/2);
  ans *= ans;
  ans %= c
  if(b%2 == 1){
    ans *= b;
    ans %=c;
  }
  return ans;
```

(note: a mod c mod c ...mod c = a mod c ?)
Analysis: $O(lgb)$ multiplications

Theorem 3 *There is at least one primitive root g for each prime*

Example 3.1 *Choose g randomly and check if g is a primitive root*

g_i is our random choice within the cycle. Using Example 2.2, the cycle is $1 \to 3 \to 4 \to 2 \to 1$. For instance, take i = 2, then we start at $g^2 = 4$, and take 2 steps at a time. The cycle will be $4 \to 1 \to 4 \to 1 \ldots$.

Theorem 4 *If $gcd(i, p-1) = 1$, where i is power, and p-1 is the length of cycle, then we can see everything in the cycle. (note: proof is below)*

13.4 Modular Multiplicative Inverse

Recall $a^{p-1} = 1$ (mod p). Then,

$$a * a^{p-2} = 1(\text{mod p}) \implies a^{p-2} = \frac{1}{a}(\text{mod p}).$$

So whenever we want to divide by a, we can simply multiply by the **multiplicative inverse** $a^{(p-2)}$. If $gcd(a, n) \neq 1$, then a has no inverse. For example, a = 2, n = 4, there is no b such that 2b = 1 (mod 4) as 2b mod 4 will only result 0 and 2. Let $d = gcd(a, b)$. If $d|a$ and $d|b$, then $d|(a-b)$ (note: not sure why this is mentioned)

If $gcd(a, b) = 1$, then $\exists c, d \in Z$ such that $c * a + d * b = 1$

$$ca + db = 1$$

$$ca = 1(\text{mod b})$$

$$c = a^{-1}(\text{mod b})$$

Hence, we have found our multiplcative inverse.

13.5 El-Gamal Encrpytion

Example 4.1 *El-gamal encrpytion implementation*

1. *A generates p, g, a(secret key),* g^a
2. *B generates p, g, b(secret key),* g^b
3. *B wants to send a message m. He sends* g^b *and* $m + g^{ab}$ *to A.*
4. *A computes* $g^{ab} = (g^b)^a$
5. *A computes* $m = e - g^{ab}$

13.6 Public Key Crpytography

Encrypt with public key, and decrpyt with secret key. (note: more will be covered next lecture)