Distributed Trust and Blockchains — Date: *24 Aug 2019*

Instructor: *Sujit Prakash Gujar* — Scribes: Pragun Saxena, Shubham Raj, Vivek Anand

Lecture 1: Digital Signatures

1 Recap

Previously we looked at hash functions.

Definition 1 (Hash Function)**.** *Functions which are used to map data of arbitrary size onto a data of fixed size.*

A desirable hash function has following properties :-

- It should compress the data.
- It should be deterministic i.e. given a input its hash must exist.
- It should be efficiently computed.
- Even a small change in message should result in large change in hash , basically hash value should be random.
- Pre-Image resistance - Should be difficult to invert.
- 2nd pre-Image resistance - given a message x difficult to compute y such that $H(x) = H(y)$ and $x \neq$y.
- Collision resistant - difficult to find two messages x and y, $x \neq$y such that $H(x) = H(y)$.

The following are applications of hash function :-

1. Message digest- use of hash value for verifying integrity of a file as even a small change in a file will change the hash value completely.
2. Commitment- It allows users to commit to a value while keeping that value hidden from others, with ability to verify their commitment to the value in future.
   The Scheme consist of Two Algorithms -
   - com := commit($key||x$) - the commit function takes message and a random key as input to return a commitment. A random key is appended to message x so that no one else can estimate hash of message x if values of x are limited and less.
   - validate(key, x, com) - the verify message takes key, message and commitment as input and returns true if com == commit($key||x$) and false otherwise.
3. Hash pointers and Merkel Trees -

   Definition 2 (Hash Pointer)**.** *A Hash pointer is basically a pointer to some data along with hash of that data.*

   A useful data structure build on hash pointers is Merkel Tree(A binary tree with hash pointers). In Merkel Tree, Data is stored in Leaf nodes and these nodes are grouped in pairs and hash of each of these nodes is stored in parent node. The parent nodes are in turn grouped in pairs and their hashes stored one level up the tree.

2 Introduction

In the Lecture we looked at digital signatures in depth which is a digital analog of a handwritten signature on paper.

Definition 3 (Digital Signature)**.** *A mathematical technique used to validate and authenticate the integrity of a message, document or digital doc. It is transmitted along the document to verify its contents and sender's identity.*

A desirable digital signature has following properties :-

1. Only you can make your signature i.e. it should be impossible to forge it.
2. Anyone who sees a signature should be able to verify that it's valid.
3. Signing authority should not be able to claim it did not sign.
4. Signature should be tied to a particular document so that your signature can not be used to indicate your agreement to a different document.

A digital signature scheme consist of following three algorithms :-

- (sk, pk) := generateKeys(keysize) - the function here generate two keys of given keysize. the secret key is sk and is used to sign messages and pk is public verification key given to everybody.
- sig := sign(sk, message) - this method takes sk and message as input and outputs a signature for message under sk.
- isValid := verify(pk, message, sig) - this method takes public key, message and singature and input and returns a bool variabel isValid, that will be true if sig is a valid signature for the message and false otherwise.

We are gonna look at two cryptographic algorithms in depth for digital signatures

1. RSA encryption
2. ElGamal encryption

2.1 RSA Algorithm for Digital Signature

RSA involves two keys public and private. The keys are generated in following way :-

1. Choose two different large prime numbers.
2. Calculate $n = p * q$.
3. Calculate totient: $\phi(n) = (p - 1) * (q - 1)$.
4. Select a large number e randomly and that will be your private key.
5. Calculate d such that $ed \equiv 1 \pmod{\phi(n)}$. This is your public key.

It is difficult to find e given d because of factorization problem.

For digitally signing the document follow the following steps-

1. Generate the hash value of the message. $H(m)$.
2. Digitally sign the message using your private key. $s \equiv m^e \pmod{n}$.
3. Now encrypt the message using receiver's public key(d'). $c \equiv s^{d'} \pmod{n}$.
   Now the message is ready to be sent.

4. On receiving the receiver will first apply his private key(e') to retrieve digitally signed message.
   $s \equiv c^{e\prime} \pmod n \Rightarrow s \equiv s^{e^{'} d^{'}} \pmod n \Rightarrow s \equiv s \pmod n$.

5. Now the receiver will verify the digital signature using sender's public key.
   verify(s, m, d) $\Rightarrow$ if $s^d \equiv m \pmod n$ return true else return false.

After decryption receiver can verify the message and sender.

Why is it important to take hash of message first?
Suppose an attacker is trying to get signatures on some documents of his choice. So initially he sends two messages m_1 and m_2 to get signed by you
So the corresponding signatures are :-

- $s_1 = m_1^e \pmod n$.
- $s_2 = m_2^e \pmod n$.

Now attacker can form a new message m_3 such that $m_3 \equiv m_1 m_2 \pmod n$.
So the signature for message m_3 will be:
$m_3 \equiv (m_1 m_2)^e \pmod n \Rightarrow m_3 \equiv s_1 s_2 \pmod n$ and the attacker knows s_1 and s_2, thus he can easily copy our digital signature.
However, if we use hash of message instead of message then
$s_1 \equiv (H(m_1))^e \pmod n$
$s_2 \equiv (H(m_2))^e \pmod n$
$s_3 \equiv (H(m_1 m_2))^e \pmod n$
and since $H(m_1 m_2) \neq H(m_1)H(m_2)$ so $s_3 \neq s_1 s_2 \pmod n$.

2.2 Elgamal Algorithm for Digital Signature

Like RSA, ElGamal also involves two keys public and private.The keys are generated in following way:-

1. Select a large prime p and generator g such that g is generator of group Z_p^* under multiplication.
2. Now calculate $h \equiv g^x \pmod p$.
3. Tuple (h, p, g) is your public key and (x,p,g) is your private key.

It is difficult to find x given h, g and p such that $g^x \equiv h \pmod p$ because of discrete log problem.

The signing protocol of ElGamal is as follows:-

1. Choose a large prime number p and select g(which is generator of group Z_p^*).
2. Select a number k randomly and calculate y such that $r \equiv g^k \pmod p$.
3. Then calculate s such that $s \equiv k^{-1}(m - x * r) \pmod n$ where m is the message to be sent and x is part of private key.
4. Return (r,s).

Now for Verification process we need to verify(m, r ,s , h, p, g) as follows :-

1. we know that $s \equiv k^{-1}(m - x * r) \pmod p$
   $\Rightarrow sk \equiv m - x * r \pmod p$
   $\Rightarrow m \equiv ks + xr \pmod p$
   $\Rightarrow g^m \equiv g^{ks+xr} \pmod p$
   $\Rightarrow g^m \equiv g^{ks} * g^{xr} \pmod p$
   and since $r \equiv g^k \pmod p$ and $h \equiv g^x \pmod p$
   . $\Rightarrow g^m \equiv (r)^s * (h)^r \pmod p$

2. so if $g^m \equiv h^r * r^s \pmod{p}$ return true else return false.

Do we similarly need to compute hash of message first as we did in RSA?
Suppose attacker has two messages and there digital signatures as follows:

- (m_1, r_1, s_1)
- (m_2, r_2, s_2)

Can signature on message m_3 such that $m_3 = m_1 * m_2$ be $(r_1 r_2, s_1 s_2)$?
No it is not possible as we choose k randomly and thus it is not possible that k of r_1 and r_2 will be same as k of $r_1 * r_2$ and therefore in the equation where we replace g^{ks} to r^s, the operation will not be valid i.e
$g^{m_1 m_2} \neq h^{r_1 r_2} * (r_1 * r_2)^{s_1 s_2}$.
This attack may not be possible but, other types of attacks are possible so it is better to still use hash of the message.

2.3 DSA and ECDSA

DSA and ECDSA(Elliptic Curve DSA) are standardized signing algorithms.
- For DSA we use two prime such that:-

p(1024 bit) and q(168 bit)
such that Z_q^* is subgroup of Z_p^*.
For this algorithm first take mod with q then with p.

DSA is difficult to solve because of discrete log problem.
- For ECDSA take a prime p and use the ellipse equation as

$$x^2 * a^{-2} + y^2 * b^{-2} = 1 \pmod{p}$$

Find all such(x,y) pairs for given a,b and collection of such pairs forms a group.
Use generator of that group, g^*

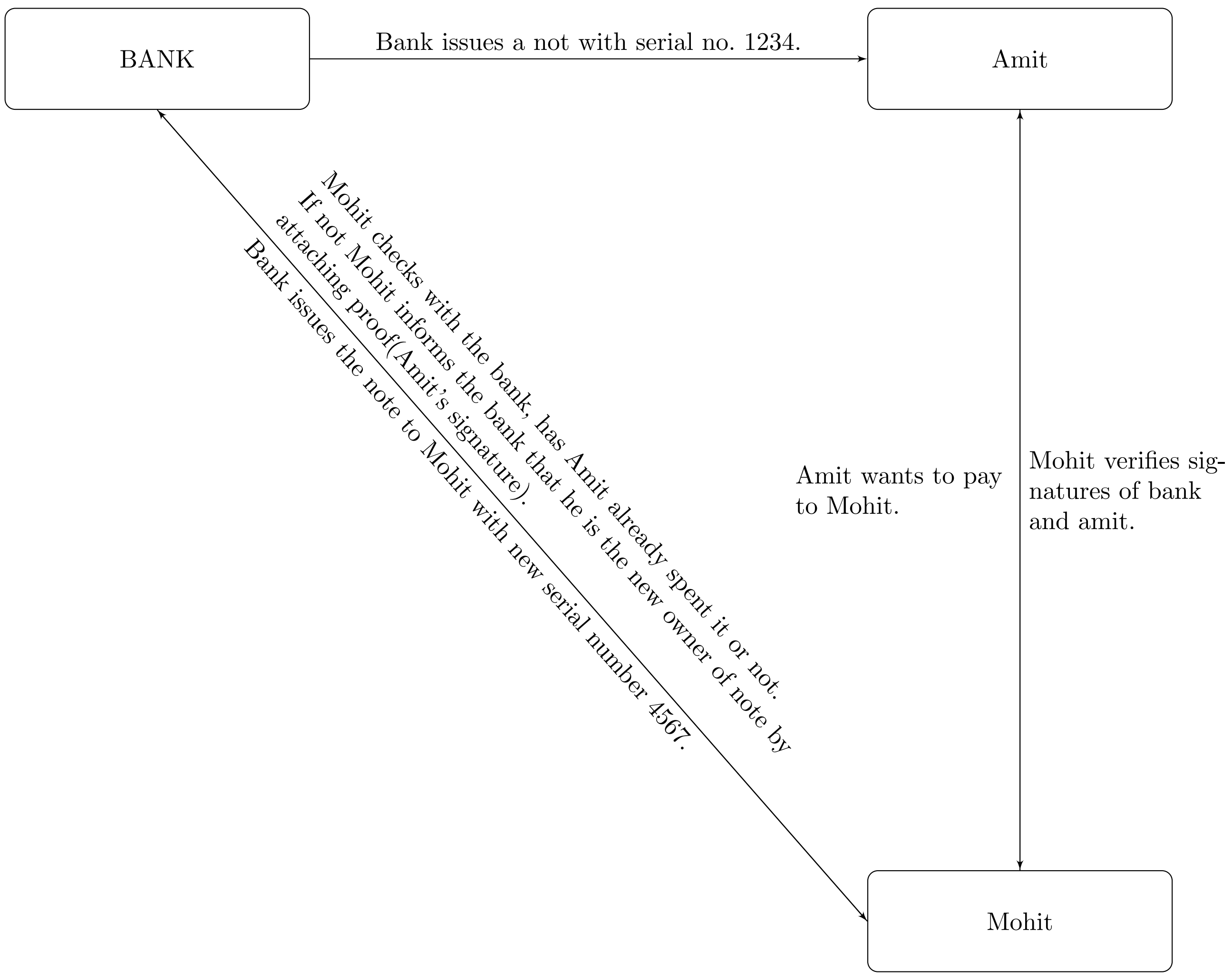
ECDSA is difficult to solve because of elliptic curve discrete problem

2.4 First Attempt of Digital Cash

requirements of Digital Cash are :-

- There should be provision of CreateCoins
- User's digital signatures should be present on Coins owned by User
- There should be a provision of transfer of ownership

Lets look at an example :-


BANK
Bank issues a not with serial no. 1234.
Amit
Mohit checks with the bank, has Amit already spent it or not.
If not Mohit informs the bank that he is the new owner of note by
attaching proof(Amit's signature).
Bank issues the note to Mohit with new serial number 4567.
Amit wants to pay to Mohit.
Mohit verifies signatures of bank and amit.
Mohit

An example of digicash system.

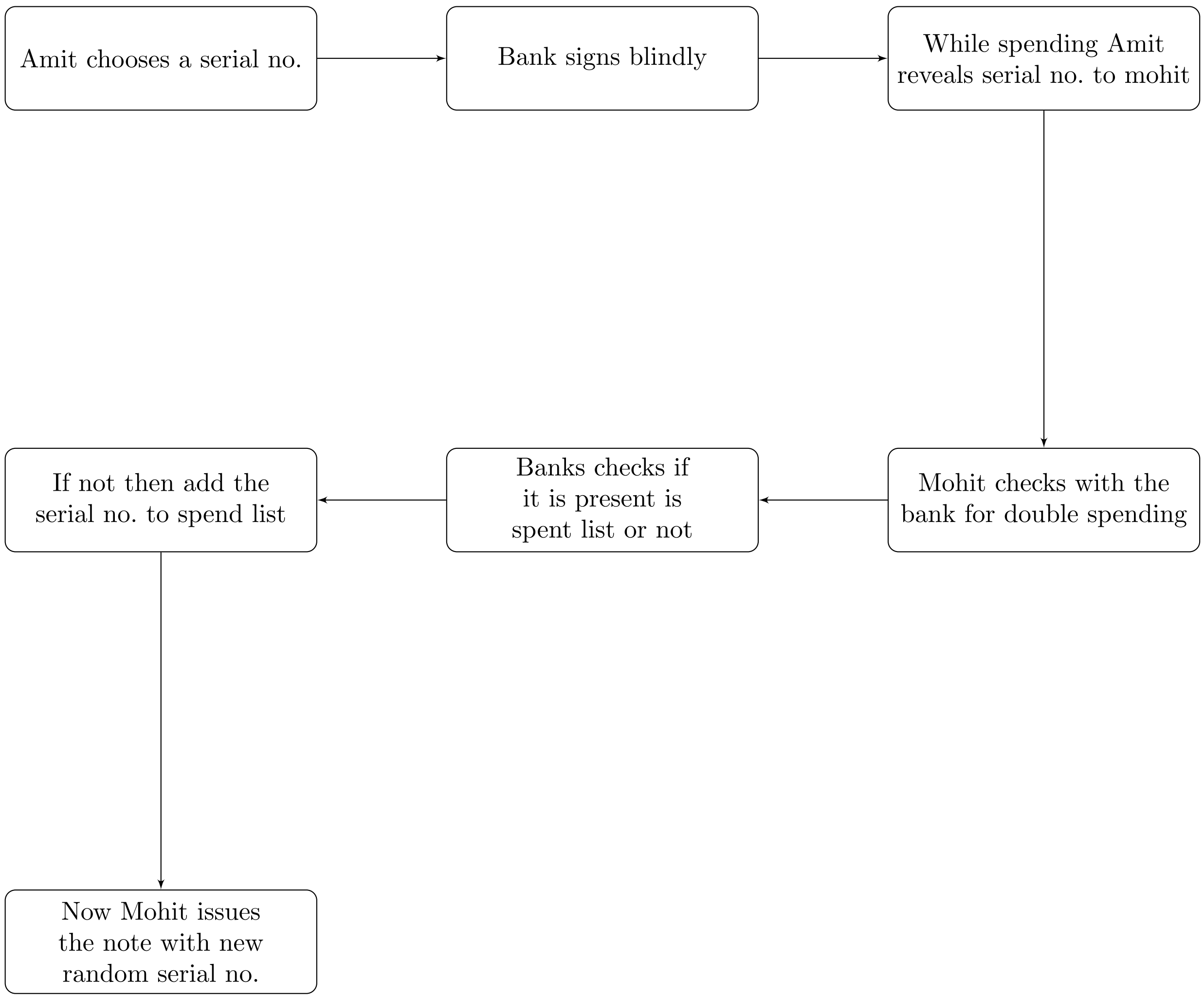
The demerits of this type of system are as follows:-

- Spending is not anonymous. Bank has details of every individual
- The system is centralized.
- Bank must remain online all the time.

To solve the issue of anonymity we have Blind Signatures. In blind signatures the bank would blindly sign the note without knowing who owns but would maintain a spend list in order to avoid double spending
In blind signature:-

- Note with serial no. ___ signed blindly by bank
- Amit chooses a serial no. 1234 and signs to it.
- Amit's signature and serial no. will be unique and bank doesn't know what serial no. is on the note nor Amit can now change the serial no.

A Flowchart to explain the process -


Amit chooses a serial no.
Bank signs blindly
While spending Amit reveals serial no. to mohit
Mohit checks with the bank for double spending
Banks checks if it is present is spent list or not
If not then add the serial no. to spend list
Now Mohit issues the note with new random serial no.

Problems with this type of system :-

- It is possible that some other person also chooses the same random number without spending the note. So no one will know that there are multiple notes with same serial no. However, probability of this is quite negligible and if we suppose that serial no. is 128 bit then only after 2^{64} generations we can say that probability of generating same random number is high.