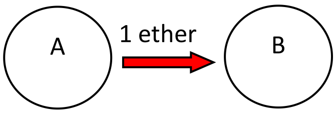
Blind Signature

Xuran Cai, Xueshen Liu
CS639 intro to blockchain
https://github.com/LouisCaixuran/blindSignature

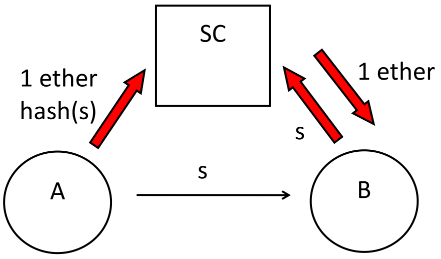
1.Introduction

In the blockchain world, every approved transaction can be found on the main chain to ensure that every approved transaction is valid, so if A want to send some money, let's say, one ether to B, then the most straightforward way to create a transaction on the chain directly.



circle - address/person
red arrow - transaction on chain

However, if A does not want anyone to know he sent money to B, then the connection between A and B is not private, as anyone with the main chain can find the transaction and hence notice the relationship between A and B. So to make it more private, we have to introduce a third party, a smart contract.



circle - address/person
square - address/contract
red arrow - transaction on chain
black arrow- private message between A and B

In this case, A create a transaction identifier string s and deposit one ether to the smart contract with the hash encryption of s. Then, B can withdraw the one ether from the smart contract with s, and the smart contract can verify it by calculating hash(s). As long as no others except A and B know s before the withdrawal request, only A and B can get the withdrawal. In this way, there is no direct transaction between A and B anymore. However, as s is on the chain, and anyone with s can calculate hash(s) themselves, it means anyone can still find the connection between A and B by finding the transaction with message hash(s). To solve this problem, we find a blind signature algorithm with RSA.

2.Algorithm and Process

2.1 RSA

An RSA key pair would contain (P,Q,N,E,D), where P and Q are large prime numbers and N=PQ. It has the following property: for any integer r where r>2 and r<N, if r is coprime to N, then r^(ED) = r (mod N). With this property, by making N and E public and keeping P,Q,D private in the smart contract, we can create the following blind transaction process.

2.2 process

Only the bold step is on the chain, all other steps can be done locally.

1. Person A create an unique transaction identifier string s and calculate the hash encryption hash(s).

2. A decide a blind factor r. As N is public, A can make sure that r and N are coprime.

3. A calculate hash(s)*(r^E) (mod N) locally with public E and N, and take the value as f(hash(s))

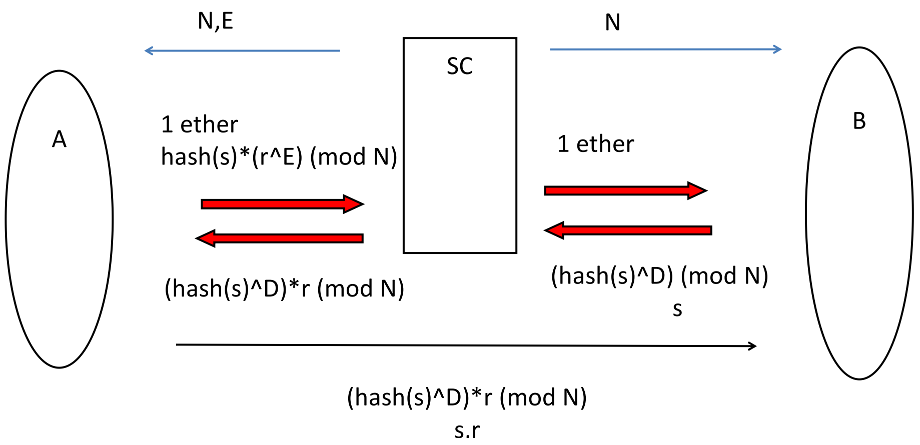
4. A deposit 1 ether and the f(hash(s)) to the smart contract, the contract would return f(hash(s))^D (mod N) back to A as blind key bk.

5. A send s, r, bk to B in private.

6. With N from contract and r from A, B can calculate the reverse of r under mod N, r’.

7. B can calculate key k with bk*r’ (mod N).

8. B send withdraw request to smart contract with k and s. The smart contract would verify hash(s)^D (mod N) = key, then send back 1 ether to B



circle - address/person
square - address/contract
red arrow - transaction on chain
black arrow- private message between A and B
blue arrow- read public value in smart contract

2.3 algorithm and safe analyze

The whole process uses the property of RSA that r^(ED) = r (mod N), hence the blind key bk = f(hash(s))^D = (hash(s)^D)*(r^DE) = (hash(s)^D)*r (mod N), and the key B generate is k = bk*r’ = (hash(s)^D)*r*r’ = hash(s)^D (mod N).

As D is private in the smart contract, B can only generate the key from the blind key given to A, ensuring that B cannot withdraw before A deposits to the smart contract. As s is private before the withdrawal transaction, only A and B can get the one ether back.

As nobody except A and B knows the blind factor r, even the smart contract cannot find the connection between the f(hash(s)) and key, or between the blind key and key, which can ensure that nobody can find the relationship between A and B on the chain during the whole process。

2.4 double spending problem

In the whole process, even the smart contract cannot connect A and B without r, so anyone with the (key,s) pair can withdraw, even when there is no deposit before the withdrawal.

As D is private in the smart contract, key can only be generated from the blind key. Hence it can ensure that before getting the pair (key,s) to withdraw, there must be at least a deposit to the contract, so we do not need to worry about the first withdraw request with each key and s.

So to solve a double spending problem, we add a mapping from s to boolean in the smart contract to ensure that each s can be only used to withdraw once. As long as the s is used, others can no longer use the same (key,s) pair to get money.

3. Conclusion and Limitation

This contract can help A send money to B privately. No other, even the contract itself, can find the connection between person A and person B. However, there are still limitations.

As while depositing, the smart contract does not know s, and there is no way to prevent person C from using the same s to send money to D even after the whole process between A and B is already finished. To make it easier, we added another function in the contract to check if the string s is used. Before making the deposit, C should call the function to check it himself.

However, if A and C use the same s to deposit the contract almost simultaneously, as they may use different blind factors, we cannot prevent this situation. In this situation, only one of B and D can get the withdrawal, and the other one would be stopped due to the double spending detection. Due to the unlinkability of the whole process, this problem is hard to solve.