EE 465: Cryptocurrency and Blockchain Technologies (Autumn 2019)
Instructor: Saravanan Vijayakumaran
Indian Institute of Technology Bombay

Midsem Exam : 15 points September 17, 2019

1. (a) ($1\frac{1}{2}$ points) Enumerate all the points of the elliptic curve $Y^2 = X^3 + 2$ over $\mathbb{F}_7$.
   (b) ($1\frac{1}{2}$ points) For each point P on the above elliptic curve, calculate the point $2P$.

2. (3 points) Suppose in the fictional country Sokovia it is illegal to own Bitcoin, with a punishment of 10 years imprisonment. Alice and Bob are residents of Sokovia who work in the same company. Bob has friends in USA who own Bitcoin. Bob has physical access to Alice's workspace in their office. Bob wants to get Alice into trouble using the illegality of owning Bitcoin. How can Bob get Alice jailed?

3. (3 points) In the Bitcoin network, miners decide which transactions to include in their candidate blocks based on the fee rate of the transactions. The fee rate of a transaction is defined as the transaction fees paid by the transaction divided by its size in bytes, with units satoshis per byte. Transactions with higher fee rate are included in blocks before transactions with lower fees. Thus the transaction fees in the Bitcoin network depends on the transaction queue. If there are more transactions waiting to be included in the next block than what can fit in a block, users who are in a hurry will pay a higher fee rate to ensure that their transaction is included quickly.

   If a malicious party is willing to lose money, then it can easily increase the fees paid by other users. It can create two addresses and broadcast several transactions which just transfer bitcoins back and forth between these two addresses. Each such transaction will result in a transaction fee being paid to the miners. This strategy will eventually result in the malicious party losing all the bitcoins it started with. But the increase in the transaction queue will result in higher fees for the rest of the users.

   Without losing money, how can the Bitcoin miners collude[1] to earn more fees from the users?

4. Let E be an elliptic curve of prime order L. Assume that the discrete logarithm problem is hard in the group E. Let G and H be generators of the group E such that the discrete logarithm of H with respect to G is not known. Let $C(x, a) = xG + aH$ be a Pedersen commitment to the amount $a \in \mathbb{Z}_L$ with blinding factor $x \in \mathbb{Z}_L$.
   (a) (1 point) Suppose you created a Pedersen commitment C to an amount v. Without revealing the blinding factor x used to create C, how can you prove to someone that C is in fact a commitment to the amount v and not to some amount $v' \neq v$?
   (b) (2 points) Suppose C is a Pedersen commitment to an amount v in the range $R = \{100, 101, \ldots, 105\}$, i.e. v is one of these six integers. The blinding factor x used to create C is known to you. Without revealing x, how can you prove to someone that C is in fact a commitment to an amount v in the range R?

5. (3 points) An auction house decides to use the following Ethereum smart contract to auction a painting which has a starting price of \$100 million. The duration of the auction is going to be 12 hours. Let us assume that the dollar price of ether remains constant for the duration of the auction. A fair auction will result in the painting being sold to the highest bidder. How can a malicious bidder prevent a fair auction? ***Hint:*** *Miners decide which transactions are included in blocks.*

   In the following program, lines beginning with two or three slashes are comments.

```
pragma solidity >=0.4.22 <0.7.0;

contract SimpleAuction {
    address payable public beneficiary;
    uint public auctionEndTime;

    // Current state of the auction.
    address public highestBidder;
    uint public highestBid;

    // Allowed withdrawals of previous bids
    mapping(address => uint) pendingReturns;

    bool ended;


```

[1] *Meaning of collude*: to do something secret or illegal with another person, company, etc. in order to deceive people.

```
    // Create a simple auction with `_biddingTime` seconds bidding time on behalf
    // of the beneficiary address `_beneficiary`.
    constructor(
        uint _biddingTime,
        address payable _beneficiary
    ) public {
        beneficiary = _beneficiary;
        auctionEndTime = now + _biddingTime;
        ended = false;
    }

    /// Bid on the auction with the value sent together with this transaction.
    /// The value will only be refunded if the auction is not won.
    function bid() public payable {
        // No arguments are necessary, all information is already part of
        // the transaction.

        // Revert the call if the bidding period is over.
        require(
            now <= auctionEndTime,
            "Auction already ended."
        );

        // If the bid is not higher, send the money back.
        require(
            msg.value > highestBid,
            "There already is a higher bid."
        );

        if (highestBid != 0) {
            // Sending back the money by simply using highestBidder.send(highestBid)
            // is a security risk because it could execute an untrusted contract.
            // It is always safer to let the recipients withdraw their money
            // themselves.
            pendingReturns[highestBidder] += highestBid;
        }
        highestBidder = msg.sender;
        highestBid = msg.value;
    }

    /// Withdraw a bid that was overbid.
    function withdraw() public returns (bool) {
        uint amount = pendingReturns[msg.sender];
        if (amount > 0) {
            // It is important to set this to zero because the recipient
            // can call this function again as part of the receiving call
            // before `send` returns.
            pendingReturns[msg.sender] = 0;

            if (!msg.sender.send(amount)) {
                // No need to call throw here, just reset the amount owing
                pendingReturns[msg.sender] = amount;
                return false;
            }
        }
        return true;
    }

    /// End the auction and send the highest bid to the beneficiary.
    function auctionEnd() public {

        require(now >= auctionEndTime, "Auction not yet ended.");
        require(!ended, "auctionEnd has already been called.");

        ended = true;

        beneficiary.transfer(highestBid);
    }
}
```