

Question 1

2 / 2 pts

What is the purpose of the “approve” function in an ERC20?



Without “approve()” contracts would not be able to transfer ERC-20 tokens (only EOAs would be able to transfer ERC-20s)



You cannot send an ERC-20 to an address that has not been “approved” and this prevents you from mistakenly sending ERC-20 tokens to contracts that cannot handle them



If you approve an address, it can spend tokens on your behalf, which makes it easier for the recipient to know the transaction has gone through



Using “approve” and then “transferFrom” results in lower gas fees

Since ERC-20s do not notify their recipient, it can be challenging for a contract to know that it has received tokens. Thus most contracts rely on the transferFrom function to “pull” tokens from a depositor. This requires an approval first.

Incorrect

Question 2

0 / 2 pts

Satoshi's analysis that showed that an attacker would need at least 50% of the hash power to attack the Bitcoin blockchain was flawed because...

☐

It did not take into account the special-purpose hardware (ASICs) that would dramatically increase the efficiency of Bitcoin mining

☐

It incorrectly assumed that an attacker would publish blocks as soon as it found them

☒

It did not take into account the prevalence of mining pools

☐

It incorrectly assumed that the only way to solve the PoW hash puzzle was to choose nonces randomly, when in fact attackers use more sophisticated techniques to mine blocks

The selfish mining attack showed that by strategically withholding blocks, an attacker could force the honest miners to waste their hash power by mining on tips that would eventually be forked out ("uncled").

Question 3

2 / 2 pts

If you send tokens from Ethereum to an optimistic rollup like Optimism or Arbitrum, then you want to withdraw your tokens, there is often a long (e.g. 7-day waiting period). Why is this?

- ☐ The rollup operators want to discourage liquidity from leaving the rollup
- ☒ The rollup contract on the L1 cannot be sure the rollup transaction will not be reverted until the fraud-proof window has expired
- ☐ The L2 sequencer cannot be sure the rollup transaction will not be reverted until the fraud-proof window has expired
- ☐ To save on L1 gas costs, the rollup sequencer only posts batches of transactions to the L1 on a weekly basis

Notice that (3) is false because the Rollup sequencer knows the state is valid as soon as it posts it to the L1. But the L1 does not want to execute the transaction, so the L1 contract must wait until the fraud-proof window expires before it can accept the transaction.

Incorrect

Question 4

0 / 2 pts

If you have USDC on Ethereum, and you send it over the Avalanche bridge you will receive USDC.e on the Avalanche C-chain. If Circle calls the "blacklist()" function on the (Ethereum) USDC contract and provides your Ethereum address, what will happen to your bridged USDC.e on the Avalanche C-chain?

☐

Your USDC.e can be transferred normally on Avalanche, and you will be able to send it back over the bridge to reclaim your USDC on Ethereum but only if you send it back to a different address

☐

You will not be able to transfer your USDC.e on Avalanche

☐

If you used the same address on Ethereum and Avalanche, you will not be able to transfer your USDC.e on Avalanche, but if you used a different address, you will still be able to transfer your USDC.e on Avalanche

☒

Your USDC.e can be transferred normally on Avalanche, but you will not be able to send it back over the bridge to reclaim your USDC on Ethereum.

When Circle blacklists your Ethereum address, you will not be able to transfer your USDC on Ethereum, but this will not have any effect on your USDC.e on Avalanche.

Question 5

2 / 2 pts

In Uniswap v2, if you are a liquidity provider who has provided 25% of the liquidity to the pool, then you will receive 25% of the fees generated from trades.

☒ True

☐ False

Question 6

2 / 2 pts

Here is an example function from an ERC20 contract:

```
function transfer(address recipient, uint amount) external returns (bool) {  
    balanceOf[msg.sender] -= amount;  
    balanceOf[recipient] += amount;  
    emit Transfer(msg.sender, recipient, amount);  
    return true;  
}
```

What would happen if all occurrences of msg.sender were changed to tx.origin?

☐ Nothing, the contract would still behave in the same way

☐ The contract would no longer compile

☐ The contract would compile, but users would not be able to transfer their tokens

☒ The change would introduce a vulnerability that would allow attackers to steal users tokens

Question 7

2 / 2 pts

In Uniswap v3, when you provide liquidity you receive an NFT from the Uniswap contract. This is because...

☐

Uniswap wants to incentivize Liquidity Providers by giving them NFTs they can sell.

☒

The ticks in Uniswap v3 mean that Liquidity Positions are no longer fungible.

☐

Using ERC-721 tokens instead of ERC-20 tokens to represent Liquidity Positions results in lower gas fees.

☐

The ERC-721 token can be redeemed for a pair of Unisocks.

Question 8

2 / 2 pts

If a company creates a new ERC-20 token, MOON, and wants to sell it on chain, it can use the Initial DEX Offering (IDO) approach, by seeding a Uniswap pool with their MOON tokens, and allowing users to buy tokens from the pool. If you wanted to sell your new MOON tokens in this way, you should...

☐

Use Uniswap v2 because you have more flexibility in setting your bonding curve on Uniswap v2

☐

Use Uniswap v2 because fees are lower on Uniswap v2

☒

Use Uniswap v3 because you don't need to provide any additional liquidity beyond your MOON tokens

☐

Use Uniswap v3 because Uniswap v2 doesn't support ERC-20 tokens

In Uniswap v2, you have to deposit an equal value of both tokens into a pool, so if you want to sell MOON tokens for (say) USDC, you have to deposit both MOON and USDC. In Uniswap v3, you can deposit only one token.

Incorrect

Question 9

0 / 2 pts

Lending protocols like Aave use oracles to get real-time price data. If the price oracle reports an anomalously high price for ETH, what would you expect to happen?



Users who put up ETH as collateral to borrow other assets would likely see their positions liquidated



Users who borrowed ETH would likely see their positions liquidated



Users who borrowed ETH would likely see their interest rates increase



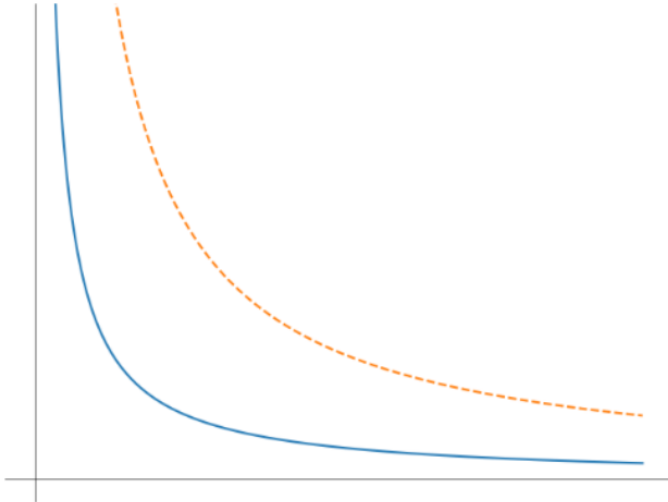
Users who put up ETH as collateral to borrow other assets would likely see their interest rates increase

If the price of ETH jumped (according to the oracle), users who borrowed ETH would likely find that their positions were under-collateralized, and would see their positions liquidated.

Question 10

2 / 2 pts

The plot below shows two potential bonding curves for an $xy = k$ automated market maker (like Uniswap). Which curve would you prefer to trade on, and why?



- ☐ The solid blue curve because the blue curve is closer to the axes, resulting in lower fees
- ☐ The dashed orange curve because the orange curve is further from the axes, resulting in lower fees
- ☐ The solid blue curve because it has more curvature, resulting in less slippage
- ☒ The dashed orange curve because it has less curvature, resulting in less slippage

The dashed orange curve represents a pool with more liquidity, and thus a trade (e.g. a given change in the x axis) will result in a flatter curve (better exchange rate).

Question 11

2 / 2 pts

A common method for creating a Verifiable Random Function is to hash the output of a digital signature scheme. So if (sk, vk) is a key pair for a digital signature scheme, we can define a function: $F_{sk}(x) = H(\text{sign}_{sk}(x))$.

Algorand implements a VRF like this using ED25519, and Ethereum's RANDAO implements a VRF like this using BLS signatures. What would be the problem with implementing this using ECDSA?

☐ ECDSA is too slow

☐

There is no problem – you could securely implement a VRF like this using ECDSA

☐ The output would not be pseudorandom

☒ The ECDSA signature algorithm is not deterministic

Unlike ED25519, ECDSA signatures are randomized, meaning that there are many valid signatures for any given message. This means that the VRF would not be binding. In other words, an adversary could influence the output of the VRF by choosing the ECDSA nonce that yielded the best result.

Question 12

2 / 2 pts

Why does Ethereum use BLS signatures as part of consensus?

- ☐ BLS signatures are quantum resistant
- ☐ It would be insecure to let validators use the same signature scheme for attestations as for their wallets
- ☒ BLS signatures can be efficiently aggregated
- ☐ BLS signatures have fewer patent restrictions than ECDSA signatures

During the Ethereum consensus process, the protocol needs to aggregate signatures from many attestors. BLS allows these signatures to be combined into one aggregate signature, which drastically reduces the amount of storage space necessary to store the signatures from the attestation committee.

Question 13

2 / 2 pts

The following contract implements a simple proof-of-work lottery in Solidity. What is wrong with this contract?

```
contract Lottery {
    bytes32 public target;
    address public winner;

    constructor( bytes32 _target ) {
        target = _target;
    }

    function guess( bytes memory preImage ) public {
        if( keccak256( preImage ) == target ) {
            winner = msg.sender;
        }
    }
}
```

☐

If a user submits a winning guess to the mempool, a bot can "frontrun" their guess

☐

If a user submits a winning guess another user can "backrun" their guess to become the winner

☒

Both option (1) and option (2)

☐

Neither option (1) or (2) – the contract would behave as expected

When a user submits a winning guess, that guess is visible to anyone monitoring the mempool, so an arbitrage bot could see the guess, check that it is a winning guess, then frontrun the transaction (by paying a higher priority fee). A second problem is that the contract does not mark the lottery as having been won, so another user could submit the same guess and their address would be set to be the winner.

Question 14

2 / 2 pts

In Ethereum, if your transaction is included in a block, you have to pay gas fees even if your transaction is reverted. Why is this?

☐

This is a legacy feature from Gavin Wood's first Ethereum client that remains for historical reasons, but is not present in most other blockchains

☐


It was designed to provide extra revenue to block producers when there is a shortage of valid transactions

☐

It was implemented to discourage Priority Gas Auctions (PGAs), where users would make submit multiple conflicting transactions, and only one could succeed

☒

It is used to prevent spam attacks against the network

If users did not have to pay for reverted transactions, they could flood the mempool with transactions that claimed to pay extremely high gas prices, but reverted after a lengthy computation. Block producers have no way to know that a transaction is going to revert without actually executing it (this is similar to the famous [halting problem](#) ) , so miners would spend all their computational resources trying to execute transactions that would eventually fail and could not be included in a block. By forcing the transaction issuer to pay for these failed transactions, this type of spam attack is no longer cost-feasible.

Incorrect

Question 15

0 / 2 pts

The following implements a simple “bank” that allows you to store and withdraw ETH. What is wrong with this contract?

```
contract Bank {
    mapping(address => uint256) public balances;

    function deposit(address _for) public payable {
        balances[_for] += msg.value;
    }

    function withdraw(address from, uint256 amount) public {
        require(balances[from] >= amount, "insufficient balance");
        balances[from] -= amount;
        msg.sender.call{value: amount}("");
    }
}
```

- ☒ The withdraw function is vulnerable to a re-entrancy attack
- ☐ The withdraw function allows you to withdraw more than you deposited
- ☐ Because the deposit function allows you to deposit into other people's accounts, you can reduce their balances by depositing negative amounts
- ☐ The withdraw function never actually sends ETH

The withdraw function withdraws from the address “from” rather than msg.sender, so by calling “withdraw()” you can withdraw ETH that was deposited by other users, and in particular, you could withdraw more than you deposited. The contract is **not** vulnerable to a re-entrancy attack because the balances[from] -= amount; line appears before the ETH is transferred.

Question 16

2 / 2 pts

The Bored Ape Yacht Club contract includes the “withdraw()” function:

```
function withdraw() public onlyOwner {  
    uint balance = address(this).balance;  
    msg.sender.transfer(balance);  
}
```

What is the purpose of this function?

☐ It allows users to claim the Apes they have purchased



It allows the contract owner to claim ETH that was generated from the initial sale of Apes

☐ It allows the contract owner to claim any Apes that were not sold



It allows the contract owner to reclaim Apes that were mistakenly transferred to the contract

The function (1) looks up the ETH balance of the BoredApe contract, then it transfers that balance to the person who called withdraw. The onlyOwner modifier means that only the owner can call this function.

Question 17

2 / 2 pts

What can make ETH a deflationary asset?

- ☒ After EIP-1559 base fees are burned
- ☐ The Ethereum consensus mechanism burns 2 ETH per block
- ☐ When a user loses the private key for their account, the ETH in that account is burned
- ☐ The Ethereum Foundation routinely burns large amounts of ETH to deflate the supply

EIP-1559 introduced a "base fee" that is assessed with each transaction. Unlike priority fees (or gas fees before EIP-1559), the priority fee does not go to the block producer, but instead is burned. If the total burning of base fees exceeds the block rewards, Ethereum can become deflationary.

Question 18

2 / 2 pts

Why don't NFTs enforce royalties on chain?

- ☐ Since Ethereum accounts are pseudonymous, there is no way to know who should pay the royalty fee
- ☐ Buyers won't buy NFTs that require royalty payments
- ☒ The NFT contract cannot determine the sale price
- ☐ The NFT contract cannot determine when it has been sold

The NFT contract keeps a record of ownership, so it can determine when the NFT was transferred, but it cannot determine the sale price, so it cannot calculate a royalty that is as *percentage* of the sale price.

Question 19

2 / 2 pts

Which of the following is true in Ethereum?

- ☐ The amount of gas used to make a specific function call changes based on demand
- ☒ The price of gas changes based on demand
- ☐ Both statements are true

The amount of gas used by a transaction is fairly fixed, and only changes on hard-forks to the protocol. The gas price is set by the user of each transaction, and the average gas price a user pays to get into a block changes based on demand.

Question 20

2 / 2 pts

In Ethereum, every transaction includes a nonce, (different from the nonce used in the Proof of Work) whereas Bitcoin transactions do not. This is because...

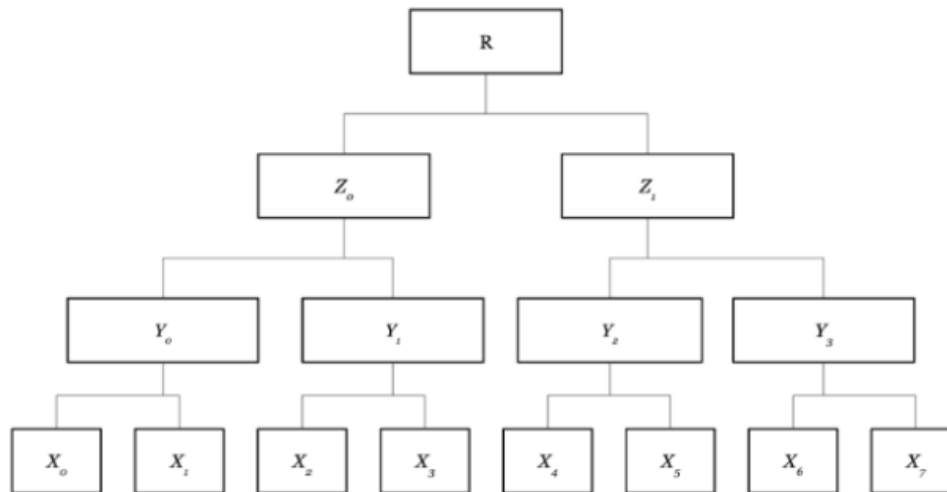
- ☐ Ethereum uses Proof-of-Stake-based consensus whereas Bitcoin uses Proof-of-Work-based consensus
- ☐ Ethereum supports smart contracts whereas Bitcoin does not
- ☒ Ethereum uses the account-balance model, whereas Bitcoin uses the UTXO model
- ☐ Ethereum uses ED25519 signatures, whereas Bitcoin uses ECDSA signatures
- ☐ The nonce provides resistance against quantum attacks

The nonce in Ethereum is designed to prevent “replay attacks” where an attacker resends a signed transaction in hopes that the transaction will be processed twice. For example, if Alice sends Bob 5 ETH, Alice could copy the transaction (along with Bob’s signature) and send it back to a miner in hopes of collecting another 5 ETH from Bob. In Bitcoin, this type of attack is not possible since each transaction references a UTXO, and if a miner sees a replayed transaction, the UTXO input will already have been spent. Ethereum (and other account-balance blockchains) solve this by including a counter (“nonce”) with each transaction, and the miner will reject transactions that have a nonce that has already been seen.

Question 21

2 / 2 pts

Given the Merkle Tree in the diagram below, if you wanted to make an inclusion proof for the element X_2 to a verifier holding the root, R , you would need to provide...



- ☐ X_2, Y_2, Z_1, R
- ☐ X_2, X_3, Y_0, Y_1
- ☒ X_2, X_3, Y_0, Z_1
- ☐ X_1, X_2, Y_0, Z_2
- ☐ X_2, X_3, Y_1, Z_0

Providing X_2, X_3 allows the verifier to compute Y_1 . If you give them Y_0 , then the verifier can compute Z_0 , and if you give them Z_1 , then the verifier can compute R (and compare it to the root value they hold).

Question 22

4 / 4 pts

The Ethereum blockchain uses the RANDAO mechanism to generate randomness to select block producers. Why is it not recommended to use the RANDAO values as a source of on-chain randomness (e.g. for betting games)? Write a short response justifying your answer.

Your Answer:

It's not recommended to use the RANDAO values as a source of on-chain randomness because:

1. Manipulation risk:

Participants can potentially manipulate the process to their advantage by controlling their randomness inputs.

2. Predictability:

If the randomness is partially known or predictable, it can be exploited, reducing the fairness and integrity of the game.

In conclusion, since RANDAO relies on multiple participants to reveal their random values sequentially, using RANDAO for critical randomness like in betting games could skew the final randomness result and lead to unfair outcomes.

Question 23

3.5 / 4 pts

What is the purpose of a "Data Availability Layer" in the context of rollups? Are they necessary to run a rollup? Write a short response justifying your answer.

Your Answer:

The purpose of a 'Data Availability Layer' in the context of rollups is to ensure that all the necessary data for transactions is available, and can be accessed and verified by all participants in the network.

1. Data verification:

It allows users to verify the correctness of the rollup's state transitions without relying solely on the rollup validator.

2. Trust:

It ensures that users do not have to trust a central party for data availability.

They are necessary to run a rollup because, without data availability, the security/trustworthiness/integrity of the rollup cannot be guaranteed, leading to potential fraud or censorship. Therefore, it is necessary to run a rollup effectively and securely.

Your answer is mostly correct but data availability layers are not strictly necessary to run a rollup. They are generally a cost-effective solution but ultimately the data could be stored on the L1 (at a higher cost).

Quiz Score: **41.5** out of 50