Proof of reserve. ​TransparentExchange claims that it controls at least 500,000 BTC and wants to prove this to its customers. To do this it publishes a list of addresses that have a total 128 balance of 500,000 BTC. It then signs the statement “TransparentExchange controls at least 500,000 BTC” with each of the corresponding private keys, and presents these signatures as proof. What are some ways in which TransparentExchange might be able to produce such a proof even if it doesn’t actually currently control 500,000 BTC? How would you modify the proof to make it harder for the exchange to cheat?

If TransparentExchange does not actually control 500,000 BTC, it could attempt to produce a false proof of reserve by either of the following methods:

Publishing a list of addresses that have a total balance of 500,000 BTC, but not actually controlling the private keys associated with those addresses. This would be a simple deception, as TransparentExchange could publish a list of addresses that they do not actually control, and then create signatures for those addresses using dummy private keys.

Controlling the private keys associated with the addresses, but moving the BTC out of those addresses after producing the proof of reserve. This would be a more sophisticated deception, as it would involve moving the BTC out of the addresses after the proof of reserve was produced, but before customers could verify the proof.

To make the proof of reserve more robust and harder to cheat, TransparentExchange could implement the following modifications:

Use a multi-signature scheme: Instead of signing the statement with each corresponding private key, TransparentExchange could use a multi-signature scheme that requires the signatures of multiple parties to produce a valid proof of reserve. For example, they could use a 2-of-3 multi-signature scheme that requires signatures from TransparentExchange, a trusted third-party auditor, and a trusted custodian.

Use a time-lock mechanism: TransparentExchange could implement a time-lock mechanism that would prevent them from moving the BTC out of the addresses for a certain period of time after producing the proof of reserve. This would allow customers to verify the proof before the BTC could be moved.

Use a verifiable computation scheme: TransparentExchange could use a verifiable computation scheme that allows customers to verify the correctness of the proof without having to trust TransparentExchange. For example, they could use a zero-knowledge proof that proves the sum of the balances of the addresses is at least 500,000 BTC without revealing the individual balances of the addresses.

Proof of liabilities. TransparentExchange implements a Merkle Tree based protocol to prove an upper bound on its total deposits. (Combined with a proof of reserve, this proves that the exchange is solvent.) Every customer is assigned a leaf node containing an ID which is the hash of her username and a value which is her BTC balance. The protocol specifies that TransparentExchange should propagate IDs and values up the tree by the following recursive definition — for any internal node: node.value = node.left\_child.value + node.right\_child.value node.id = Hash(node.left\_child.id ‖ node.right\_child.id ‖ node.value) The exchange publishes the root ID and value, and promises to prove to any customer that her node is included in the tree (by the standard Merkle tree proof of inclusion). The idea is that if the exchange tries to claim a lower total than the actual sum of deposits by leaving some customers out of the tree or by making their node value less than their balance, it will get caught when any of those customers demand a proof of inclusion. 2.1. Why can’t the exchange include fake customers with negative values to lower the total? 2.2. Show an attack on this scheme that would allow the exchange to claim a total less than the actual sum of deposits. 2.3. Fix this scheme so that it is not vulnerable to the attack you identified. 2.4. Ideally, the proof that the exchange provides to a customer shouldn’t leak information about other customers. Does this scheme have this property? If not, how can you fix

2.1. The exchange can't include fake customers with negative values to lower the total because the recursive definition of the Merkle tree only allows for non-negative values. Adding negative values would result in a negative value for the root, which is not allowed.

2.2. An attack on this scheme would involve the exchange leaving out some customers from the tree entirely. For example, the exchange could exclude customers with small balances or those who have not used the exchange recently. This would allow the exchange to claim a total less than the actual sum of deposits.

2.3. To fix this scheme, the exchange could modify the recursive definition of the Merkle tree to include all customers, regardless of their balance or activity level. Specifically, the exchange could set the value of leaf nodes to be the maximum of their balance and a certain threshold value (e.g., 0.001 BTC). This would ensure that all customers are included in the tree and that the total is not understated.

2.4. This scheme does not have the property of not leaking information about other customers. When a customer requests a proof of inclusion, the exchange must reveal the path from the root to the customer's leaf node, which would reveal the IDs and values of all the nodes on that path. To fix this, the exchange could use a zero-knowledge proof to prove the inclusion of a customer's leaf node without revealing any information about other nodes in the tree. Alternatively, the exchange could use a different data structure, such as a hash chain or a hash-linked list, that does not reveal the entire structure of the tree.

Multi-signature wallet 4.1. BitCorp has just noticed that Mallory has compromised one of their servers holding their Bitcoin private keys. Luckily, they are using a 2-of-3 multi-signature wallet, so Mallory has learnt only one of the three sets of keys. The other two sets of keys are on different servers that Mallory cannot access. How do they re-secure their wallet and effectively revoke the information that Mallory has learned? 4.2. If BitCorp uses a 2-out-of-2 instead of a 2-out-3 wallet, what steps can they take in advance so that they can recover even in the event of one of their servers getting broken into (and Mallory not just learning but also potentially deleting the key material on that server)?

4.1. To re-secure their wallet and effectively revoke the information that Mallory has learned, BitCorp should first generate a new set of keys on a separate, secure server. They should then create a new 2-of-3 multi-signature wallet using the three new sets of keys (including the new one). They can then transfer all of the Bitcoin from the old wallet to the new one, using the remaining two sets of keys. This will ensure that Mallory cannot access the funds in the new wallet, even if she still has access to the compromised server holding one of the old keys.

4.2. If BitCorp uses a 2-out-of-2 wallet, they can use a technique called "key sharding" to recover in the event of a server breach. Key sharding involves splitting each private key into multiple pieces, such that each piece is useless on its own but the entire key can be reconstructed from a sufficient number of pieces. For example, each private key could be split into 10 pieces, and any 7 pieces would be required to reconstruct the full key. BitCorp could then distribute these key pieces across multiple servers, so that no single server has access to the full key. In the event that one server is breached and the key pieces on that server are potentially deleted, BitCorp can still recover the full key by collecting enough key pieces from the remaining servers.

MINING POOL

Mining pools are groups of miners who work together to increase their chances of finding a block and earning the associated rewards. Instead of each miner working independently, they combine their computing power and split the rewards based on their contribution to the pool.

A mining pool typically has a pool manager who is responsible for managing the pool's operations, including distributing rewards, maintaining the pool's infrastructure, and managing communication among the pool's members. The pool manager may also take a small fee or percentage of the rewards as compensation for their services.

In the context of cryptocurrency mining, a 51% mining pool refers to a mining pool that controls more than 50% of the total computing power (hash rate) of the network. This can be problematic because it gives the pool significant control over the network and the ability to potentially conduct a "51% attack". In a 51% attack, the pool could potentially rewrite the blockchain and manipulate transactions in their favor, such as double-spending or excluding transactions.

To mitigate the risk of a 51% attack, many blockchain networks implement measures to prevent any one entity from controlling the majority of the network's hash rate. For example, some networks have implemented consensus algorithms that are resistant to attacks from large mining pools, while others have implemented measures to encourage decentralization and discourage large mining operations.