**Chapter 4 Problems:**

**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?

**Ways TransparentExchange Might Cheat:**

1. **Borrowed Bitcoins:** They could temporarily borrow bitcoins from another entity to show control over 500,000 BTC.
2. **Shared Private Keys:** They might use addresses they don’t fully own but share private keys with others to create the signatures.
3. **Collaborators:** They might collaborate with other exchanges or entities to gain temporary access to private keys for proving ownership.

**How to Prevent Cheating:**

1. **Time-Locked Proofs:** Use time-locked transactions to ensure the exchange cannot just borrow bitcoins temporarily.
2. **Proof of Exclusive Control:** Require the exchange to demonstrate ongoing control over the bitcoins, such as moving small amounts during a verification period.
3. **Third-Party Audits:** Involve trusted third-party auditors to validate ownership and control over the claimed funds.
4. **Continuous Monitoring:** Implement regular and random checks over time to ensure consistent control of the stated bitcoins.

**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

it?

**2.1 Why can’t the exchange include fake customers with negative values to lower the total?**

* **Reason:** Negative values in the Merkle Tree would make the node.value invalid because the sum of values propagating up the tree would be incorrect. During the proof of inclusion, the incorrect propagation would be evident when verifying the consistency of the tree structure.

**2.2 Attack on the Scheme:**

* **Attack Method:** The exchange could split a legitimate customer's balance across multiple leaf nodes. For example, if a customer has 10 BTC, the exchange could create two leaf nodes with 5 BTC each and only prove one of them. This would allow the exchange to claim a smaller total while still satisfying the proof of inclusion for the legitimate customer.

**2.3 Fix to Prevent the Attack:**

1. **Unique User ID Enforcement:** Ensure each customer's balance is assigned to exactly one unique leaf node. Use a cryptographic commitment (e.g., Hash(username || balance)) to enforce this.
2. **Verification of Unique Leaves:** Require customers to verify that their full balance is included in exactly one leaf. Implement cross-checking to ensure no duplication or partial inclusion.
3. **Auditor Validation:** Use an independent third party to audit the entire tree and verify no customer’s balance is split across multiple nodes.

**2.4 Does the Proof Leak Information?**

* **Issue:** Yes, the proof leaks information about neighboring nodes (e.g., their IDs and balances). This is because the proof of inclusion includes siblings and intermediate nodes to verify the path to the root.

**Fix to Prevent Information Leakage:**

1. **Zero-Knowledge Proofs:** Use a zero-knowledge proof (ZKP) system where the exchange proves the correctness of the total without revealing individual values or intermediate nodes.
2. **Homomorphic Encryption:** Encrypt customer balances in such a way that the tree can still sum values correctly, but individual balances remain hidden.
3. **Commitment Schemes:** Use a commitment scheme where each leaf contains only a commitment to the customer’s balance, rather than the balance itself. Only the customer can verify their balance with a secret key.

**Transaction fees.**

**3.1.** Alice has a large number of coins each of small value *v* , which she would like to

combine into one coin. She constructs a transaction to do this, but finds that the

transaction fee she’d have to spend equals the sum of her coin values. Based on this

information (and the default transaction fee policy specified in slide 50), estimate *v* .

**3.2.** Can Alice somehow consolidate her coins without incurring any transaction fee under

the default policy?

**3.3.** Compared to a fee structure that doesn’t factor the age of the inputs into the

transaction fee, what effect might the current default fee structure have on the

behavior of users and services?

**3.1 Estimate vvv:**

* **Default Fee Policy:** Transaction fees depend on the size of the transaction in bytes. A typical transaction has a fee of 0.00001 BTC per byte.
* If the transaction fee equals the total value of Alice’s coins, then fee per byte×transaction size in bytes=sum of coin values\text{fee per byte} \times \text{transaction size in bytes} = \text{sum of coin values}fee per byte×transaction size in bytes=sum of coin values.
* Each coin creates an input, and the number of inputs determines the transaction size.
* vvv, the value of each small coin, is roughly equal to the cost of the transaction divided by the number of inputs:  
  v≈fee per byte×bytes per inputv \approx \text{fee per byte} \times \text{bytes per input}v≈fee per byte×bytes per input.  
  Assuming bytes per input≈148\text{bytes per input} \approx 148bytes per input≈148:  
  v≈0.00001×148=0.00148 BTCv \approx 0.00001 \times 148 = 0.00148 \, \text{BTC}v≈0.00001×148=0.00148BTC.

**3.2 Can Alice consolidate her coins without incurring a transaction fee?**

* **Under Default Policy:**
  + If the combined age of the inputs is high enough (i.e., the coins have been untouched for a long time), the transaction might qualify for a reduced or waived fee.
  + Alice can wait for her coins to age sufficiently to reduce the transaction fee.
  + She could also attempt to mine her own transaction by running a node to include the transaction in a block without paying fees. This requires mining resources.

**3.3 Effect of Age-Based Fee Structure:**

1. **Encourages Holding Coins:**
   * Users are incentivized to hold their coins for longer periods since older coins reduce fees.
   * This may reduce transaction frequency for small-value coins.
2. **Promotes Consolidation Early:**
   * Users with many small-value coins may consolidate them periodically to avoid high fees later.
3. **Impact on Services:**
   * Services dealing with frequent small payments (e.g., faucets, microtransactions) might face higher operational costs due to higher fees for newer coins.
   * They may pass these costs to users or switch to alternative fee structures.
4. **Discourages Dust Creation:**
   * Small-value coins ("dust") become less usable due to high transaction fees, discouraging the creation of tiny unspent outputs.

**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

129

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 Re-securing a 2-of-3 Multi-Signature Wallet:**

* **Revoke the Compromised Key:** Create a new 2-of-3 wallet and transfer all funds from the old wallet to the new one.
* **Remove Compromised Key:** Stop using the compromised key and ensure it's destroyed or no longer accessible.

**4.2 Recovery Steps for a 2-out-of-2 Wallet (Server Break-In):**

* **Store One Key Offline:** Keep one private key in a secure, offline location (e.g., hardware wallet or paper wallet).
* **Backup Both Keys:** Ensure both private keys are securely backed up in separate, safe locations.
* **Key Management:** Use methods like key splitting or a trusted third party to hold a backup key.
* **Monitor for Suspicious Activity:** Set up alerts to track any unauthorized access or actions on the wallet.

**5. Exchange rate**

**5.1.** Speculate about why buying bitcoins in person is generally more expensive than

buying from an online exchange.

**5.2.** Moore and Christin observe that security breaches and other failures of exchanges

have little impact on the Bitcoin exchange rate. Speculate on why this might be.

**5.1 Why Buying Bitcoins In Person is More Expensive:**

* **Higher Risk:** In-person transactions involve physical security risks, so sellers may charge a premium for the added risk.
* **Convenience:** In-person trades offer instant settlement, so they may charge more for the convenience and immediacy.
* **Limited Availability:** There are fewer in-person buyers and sellers, which can drive up prices due to limited supply and higher demand.

**5.2 Why Exchange Failures Don't Affect Bitcoin's Exchange Rate:**

* **Decentralized Nature:** Bitcoin is not tied to any central exchange; it's decentralized, so failures of exchanges don't directly affect its value.
* **Global Market:** Bitcoin's price is influenced by many markets and exchanges, so a failure at one doesn't impact the overall market significantly.
* **Perceived Resilience:** Investors may view Bitcoin as a secure, stable asset, regardless of issues with individual platforms.

**6. Payments.** A Bitcoin payment service might receive thousands of payments from various users

near-simultaneously. How can it tell whether a particular user Alice who logged into the

payment service website and initiated the payment protocol actually made a payment or not?

**6. How Can a Payment Service Confirm Alice Made a Payment:**

* **Transaction Verification:** The service can check the Bitcoin blockchain to see if the transaction from Alice’s wallet address is confirmed.
* **Transaction ID:** Once Alice initiates a payment, the service can look for the specific transaction ID to confirm the payment has occurred.
* **Payment Protocol:** Alice's action in the protocol, such as signing the transaction, can provide proof of her intent to pay.

**7. BitcoinLotto:** Suppose the nation of Bitcoinia has decided to convert its national lottery to use

Bitcoin. A trusted scratch-off ticket printing factory exists and will not keep records of any

values printed. Bitcoinia proposes a simple design: a weekly run of tickets is printed with an

address holding the jackpot on each ticket. This allows everybody to verify the jackpot exists.

The winning ticket contains the correct private key under the scratch material.

**7.1.** What might happen if the winner finds the ticket on Monday and immediately claims

the jackpot? Can you modify your design to ensure this won’t be an issue?

**7.2.** Some tickets inevitably get lost or destroyed. So you’d like to modify the design to roll

forward any unclaimed jackpot from Week *n* to the winner in Week *n+1* . Can you

propose a design that works, without letting the lottery administrators embezzle

funds? Also make sure that the Week *n* winner can’t simply wait until the beginning of

Week *n+1* to attempt to double their winnings

**7.1 Issue with Immediate Jackpot Claim:**

* **Problem:** If the winner immediately claims the jackpot, they might reveal the private key to the address before everyone can verify the jackpot’s existence.
* **Solution:** **Time Lock:** Implement a time lock that prevents claiming the jackpot until a set period has passed (e.g., one week after ticket issuance). This ensures no one can claim prematurely.

**7.2 Rolling Forward Unclaimed Jackpot Design:**

* **Problem:** Allowing winners to wait until the next week could lead to abuse, such as waiting to claim after seeing who wins Week n+1.
* **Solution:** **Escrow System:** The unclaimed jackpot can be stored in an escrow account controlled by a smart contract. The contract can automatically transfer the funds to the winner of the next week, with rules preventing waiting to double winnings. The contract ensures only one valid claim can be made each week.