Q. What is a double spending problem? Show the problem of double spending in Chaumian e-cash with multiple transactions.

https://bitcoin.stackexchange.com/questions/9544/how-does-chaum-style-e-cash-work-all-the-wiki-links-are-broken

We covered Chaumian e-cash in the class, you may also get help from this link about working of Chaumian e-cash

The double-spending problem is a fundamental issue in digital currencies, where a user tries to spend the same electronic currency (e-cash) multiple times, essentially creating counterfeit copies. This problem undermines the integrity and trustworthiness of digital currencies.

In Chaumian e-cash, which is based on the work of David Chaum, the double-spending problem can be illustrated as follows, using a simplified example:

1. Initial Transaction: Alice creates a digital token representing e-cash and sends it to Bob as payment for a product or service. This initial transaction is recorded on the e-cash ledger, showing that Alice has spent the e-cash to Bob.

2. Double Spending Attempt: However, Alice, being dishonest, wants to double-spend the same e-cash. She quickly creates a copy of the e-cash token she sent to Bob.

3. Second Transaction: Alice sends the copy of the e-cash token to Carol as payment for something else before Bob or the network can detect the double spending.

Now, there are two transactions on the e-cash ledger:

- Transaction 1: Alice -> Bob
- Transaction 2: Alice -> Carol

This situation creates a problem because Alice has effectively spent the same e-cash token twice. If both Bob and Carol believe they have received valid e-cash from Alice, the double-spending problem becomes apparent. This compromises the trust and reliability of the e-cash system.

To address the double-spending problem in Chaumian e-cash and similar systems, cryptographic techniques and protocols are used. In Chaumian e-cash, blind signatures and protocols for ensuring unlinkability between tokens are employed. Here's how it works:

1. Blind Signatures: When Alice initially receives e-cash from the bank, she requests a blind signature, which allows her to hide the specific serial number of the e-cash token. This blind signature is unique to Alice's token but doesn't reveal which one it is.

2. Token Redemption: When Alice spends her e-cash, she provides the blinded token to the recipient (e.g., Bob). Bob can verify the validity of the e-cash by checking the bank's signature without learning the serial number of the token.

3. Prevention of Double Spending: Since the bank keeps track of the serial numbers it has issued, if Alice tries to spend the same e-cash token twice, the bank will detect the double spending because it has already issued a valid blind signature for that token.

In this way, Chaumian e-cash prevents double spending by ensuring that the bank can detect and reject attempts to spend the same token multiple times while preserving the privacy of individual transactions.

Q. What is a hash function? What is the size of input and output to a SHA-256 algorithm? What are the properties of SHA-256. Explain with examples using the hash of different numbers that how SHA 256 is

a. pre-image resistance

b. collision resistance

c. avalanche effect

d. use https://emn178.github.io/online-tools/sha256.html to compute hash for demonstration

A hash function is a mathematical function that takes an input (or "message") and returns a fixed-size string of bytes, which is typically a hexadecimal number. Hash functions are widely used in computer science and cryptography for various purposes, such as data integrity verification, password storage, and digital signatures.

SHA-256, which stands for Secure Hash Algorithm 256-bit, is a widely used cryptographic hash function that belongs to the SHA-2 family of hash functions. It produces a 256-bit (32-byte) hash value as its output. This means that regardless of the size of the input data, SHA-256 will always generate a 256-bit hash value.

Properties of SHA-256:

a. Pre-image Resistance: Pre-image resistance means that it should be computationally infeasible to reverse the hash function and find the original input from its hash value. In other words, given a hash value, it should be extremely difficult to determine the input that produced that hash.

Example: Let's say we have a hash value, `H`, produced by SHA-256 from some input data, `X`. It should be computationally infeasible to find `X` given `H`.

b. Collision Resistance: Collision resistance means that it should be computationally infeasible to find two different inputs that produce the same hash value. In other words, it should be highly improbable that two different inputs will produce identical hash values.

Example: Given SHA-256 hash values, `H1` and `H2`, it should be computationally infeasible to find two different inputs, `X1` and `X2`, such that SHA-256(X1) = H1 and SHA-256(X2) = H2, and X1 ≠ X2.

c. Avalanche Effect: The avalanche effect means that a small change in the input data should result in a significantly different hash value. Even a single-bit change in the input should produce a hash that looks completely different.

Example: Let's consider two similar inputs, `X` and `X'`, where `X'` is identical to `X` except for one bit. If you hash both `X` and `X'` using SHA-256, the resulting hash values, `H` and `H'`, should appear completely unrelated, even though the inputs are nearly identical.

Example of SHA-256:

Suppose we have two inputs:

- Input 1: "Hello, World!"
- Input 2: "Hello, World!!"

We'll compute the SHA-256 hashes for these inputs:

- SHA-256("Hello, World!") = `6dcd4ce23d88e2ee9568ba546c007cde6ac81e2a2dcbbc0e2f07e0c12c0e8159`

- SHA-256("Hello, World!!") = `96d95b2c6eebbe85e6b9f48967b6eddb44aeb35fc5f3a424661b628d66b2ce33`

You can see that even though the inputs are very similar, the resulting hash values are entirely different, demonstrating the avalanche effect. Additionally, it's computationally infeasible to find an input that produces a specific SHA-256 hash value, illustrating pre-image resistance. Finally, it's extremely unlikely that two different inputs would produce the same hash value, showcasing collision resistance.