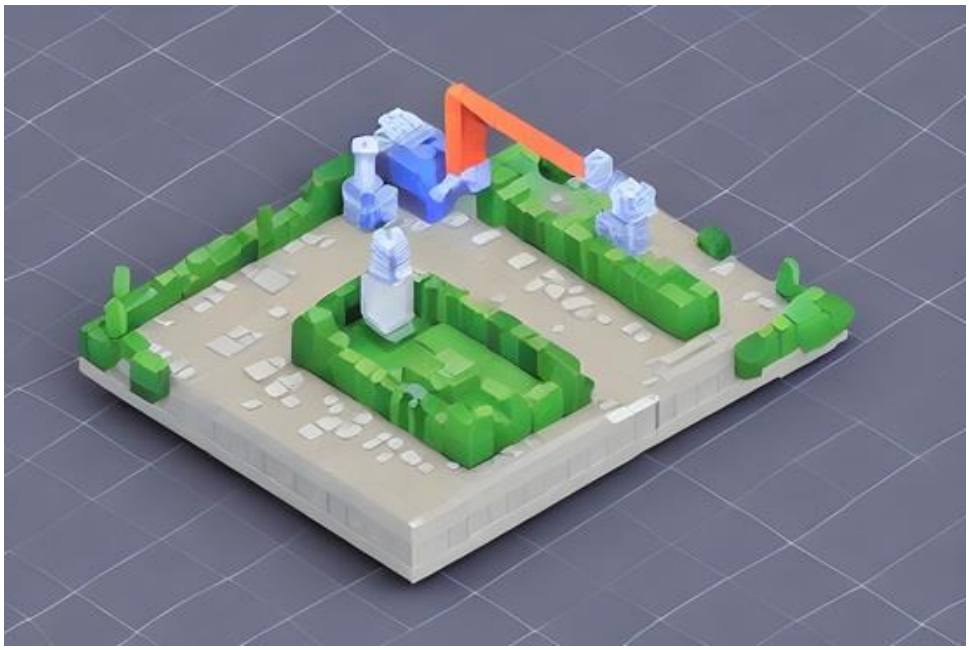
Technical details about consecues algo sha256csm

The SHA-256CSM algorithm is a cryptographic hash function that is used in various blockchain and cryptocurrency technologies, including Bitcoin. Below are the technical details of the SHA-256CSM algorithm:



SHA-256 Overview

SHA-256 (Secure Hash Algorithm 256-bit) is a member of the SHA-2 (Secure Hash Algorithm 2) family of cryptographic hash functions. It generates a fixed-size 256-bit (32-byte) hash value from an input data of arbitrary size. Another various hashing algorithm is SHA-256d, it uses a double bit hashing (512bit) function. SHA-3. The SHA-256ASICBOOST hashing consecues algorithm differs from SHA-256 by adding 4 bytes or 32 bits as a candidate to the root of the genesis block. The MerkleRoot candidate a added as temBh without a nonce:

```
while True:
    blockHeader = ...  # based on various fields
    candidates = dict()  # 4 bytes -> sets of blocks
    for i in range(...):  # Generate the more the merrier
        tempBh = blockHeader.randomizeMerkle()
        sha256Block0 = sha256Expand(tempBh[0 : 64])
        tempBh.midState = sh256Compress(sha256InitVector, sha256Block0)
        candidates[tempBh.merkleRoot[28 : 32]].add(tempBh)

    for i in range(2**32):  # Try nonces
        for key in candidates:
            tempBh = candidates[key][0]
            tempBh.nonce = i
            sha256Block1 = padAndExpand(tempBh[64 : 80])

            for tempBh in candidates[key]:
                singleSha = sh256Compress(tempBh.midState, sha256Block1)
                sha256Block2 = padAndExpand(singleSha)
                doubleSha = sh256Compress(sha256InitVector, sha256Block2)
                if doubleSha < target:
                    miningSuccessful(blockHeader)  # Jackpot!
```

Thus the sha256csm mining technique wins over conventional mining when most candidate groups have more than one candidate, and that the overhead and over tail of generating and sorting candidates exceeds the gains from saving at most one calculation of block expansion per candidate. By adding the initial hash value to the next block together with a timestamp and nonce we generate new blocks in the SHA256csm algorithm differently than other SHA256 algorithm's that changes over block time. It saves overall 30% processor power.

Sha256csm Hashing Algorithm Steps:

- **Initialization:**
  - Initialize hash values:

```
makefile                                                    Copy code

h0 = 0x6a09e667
h1 = 0xbb67ae85
h2 = 0x3c6ef372
h3 = 0xa54ff53a
h4 = 0x510e527f
h5 = 0x9b05688c
h6 = 0x1f83d9ab
h7 = 0x5be0cd19
```

- **Pre-processing:**
  - Append padding bits:
    - Append a single '1' bit
    - Append K '0' bits, where K is the minimum number >= 0 such that the resulting message length in bits is congruent to 448 (mod 512)
    - Append the original length of the message in bits as a 64-bit big-endian integer
- **Processing:**
  - Break the message into 512-bit chunks

    - For each chunk:
        - Create a message schedule of 64 32-bit words
        - Initialize eight working variables with the current hash value
        - Perform 64 rounds of operations
- **Output:**
    - Concatenate the final hash values to produce the 256-bit hash value

SHA-256CSM Specifics:

```
/** serialization implementation */
ADD_SERIALIZE_METHODS;

template <typename Stream, typename Operation>
inline void SerializationOp(Stream& s, Operation ser_action, int nType, int nVersion)
{
    READWRITE(nTransactions);
    READWRITE(vHash);
    std::vector<unsigned char> vBytes;
    if (ser_action.ForRead()) {
        READWRITE(vBytes);
        CPartialMerkleTree& us = *(const_cast<CPartialMerkleTree*>(this));
        us.vBits.resize(vBytes.size() * 8);
        for (unsigned int p = 0; p < us.vBits.size(); p++)
            us.vBits[p] = (vBytes[p / 8] & (1 << (p % 8))) != 0;
        us.fBad = false;
    } else {
        vBytes.resize((vBits.size() + 7) / 8);
        for (unsigned int p = 0; p < vBits.size(); p++)
            vBytes[p / 8] |= vBits[p] << (p % 8);
        READWRITE(vBytes);
    }
}
```

The "CSM" in SHA-256CSM stands for "Consecutive specialized Masternode" This term is related to the Bitcoin mining algorithm, which utilizes the SHA-256 cryptographic hash function for mining new blocks.

In the context of Sha256csm mining:

- **Block Header:**
    - Version (Genesis block)
    - Previous block hash
    - MerkleTree root transactions
    - Timestamp (Time-lock)
    - Difficulty target (multi-signature addresses)
    - Nonce (which miners change to find a valid hash)

- **Mining Processes:**
    - Miners change the nonce value in the block header and calculate the SHA-256 hash of the block header

- The goal is to find a hash that meets the current difficulty target set by the network
- If the hash does not meet the target, miners increment the nonce and try again
- Once a valid hash is found (i.e., the hash is less than the target), the block is considered mined, and the miner broadcasts the block to the Network

Properties:

- **Deterministic:** For a given input, the output (hash) is all ways the same.
- **Pre-image resistant:** Given a hash, it is computationally infeasible to determine the original input data. (Data obfuscation)
- **Collision resistant:** It is computationally infeasible to find two different inputs that produce the same hash output.

Security:

```
uint256 hashMerkleRoot = TraverseAndExtract(nHeight, 0, nBitsUsed, nHashUsed, vMatch);
// verify that no problems occured during the tree traversal
if (fBad)
    return 0;
// verify that all bits were consumed (except for the padding caused by serializing it as a byte sequence)
if ((nBitsUsed + 7) / 8 != (vBits.size() + 7) / 8)
    return 0;
// verify that all hashes were consumed
if (nHashUsed != vHash.size())
    return 0;
return hashMerkleRoot;
```

SHA-256 is considered a secure cryptographic hash function, meaning that it is resistant to various types of attacks, including collision attacks, pre-image attacks, and second pre-image attacks, when used correctly.

Sha256csm has more security features to bid in with. Multi-signature addresses. Time-locked instant transactions, Fixed transaction fee, Proof of stake and network-wide upgrades.

Conclusion:

SHA-256CSM is a variant of the SHA-256 algorithm specifically used in the Bitcoin mining process. It is a hybrid and cryptographic hash function that is deterministic, pre-image resistant, and collision resistant, making it suitable for fast and secure blockchain network and verifying the integrity of data.

Please note that while SHA-256CSM is related to Bitcoin mining, the specific implementation and parameters may vary based on the software or platform using the algorithm.