BITCOIN MECHANICS & OPTIMIZATIONS:

A TECHNICAL OVERVIEW

SISHIR GIRI & HAENA LEE

LECTURE OVERVIEW

01 Cryptographic Hash Functions

02 A Tamper-Evident Database

03 SIGS, ECDSA, AND ADDRESSES

04 BITCOIN SCRIPT

INTRODUCTIONS

INTRODUCING YOUR LECTURERS

Haena Lee

Education

Sishir Giri

Consulting

CRYPTOGRAPHIC HASH FUNCTIONS

CRYPTOGRAPHIC HASH FUNCTIONS

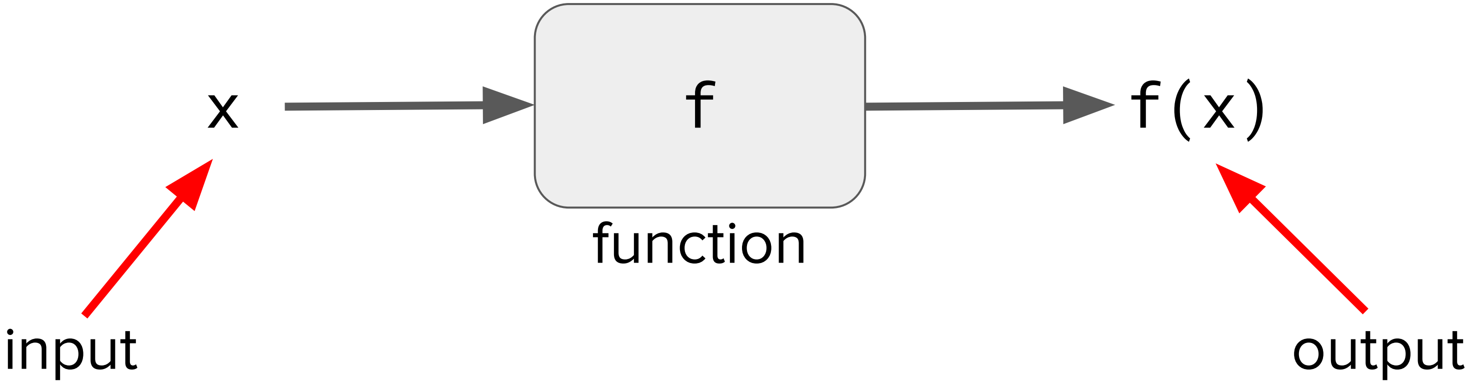
How do we ensure trust in communication in a trustless environment?

⇒ With **cryptographic hash functions**

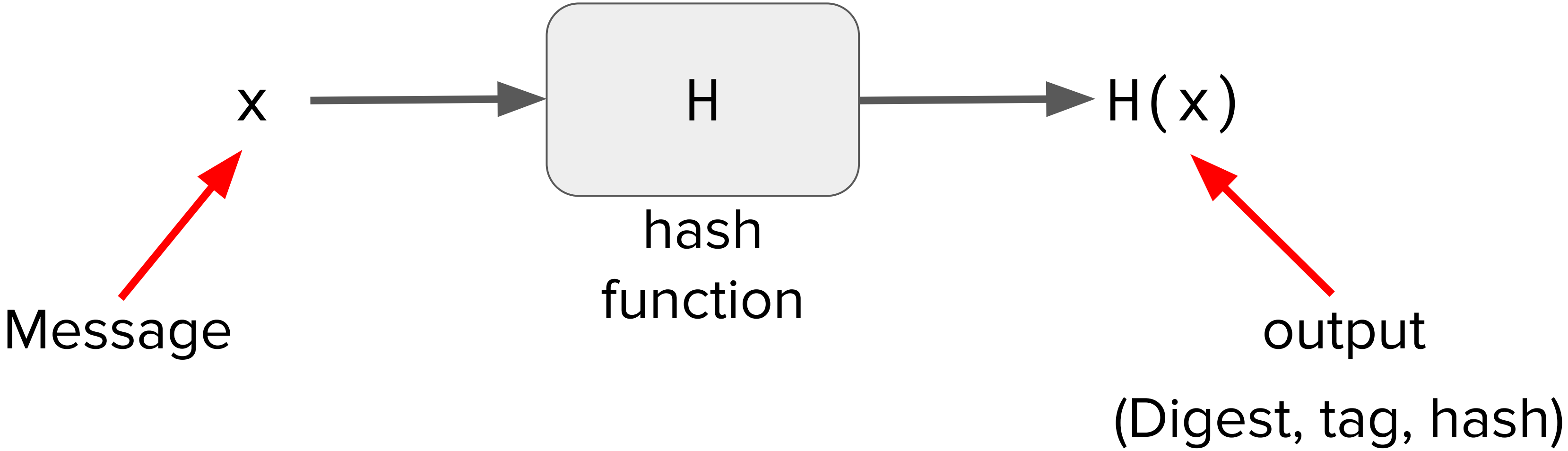
Image source: https://spiritegg.com/wp-content/uploads/2016/03/63180952_fingerprint_types624.jpg

USED HIGHLY IN DIGITAL SIGNATURES

CRYPTOGRAPHIC HASH FUNCTIONS

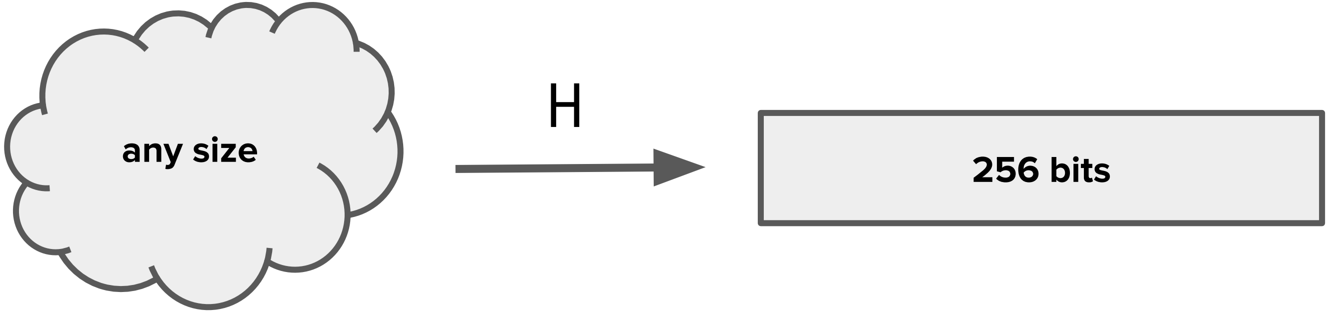


CRYPTOGRAPHIC HASH FUNCTIONS



WHAT IS A HASH FUNCTION?

CRYPTOGRAPHIC HASH FUNCTIONS



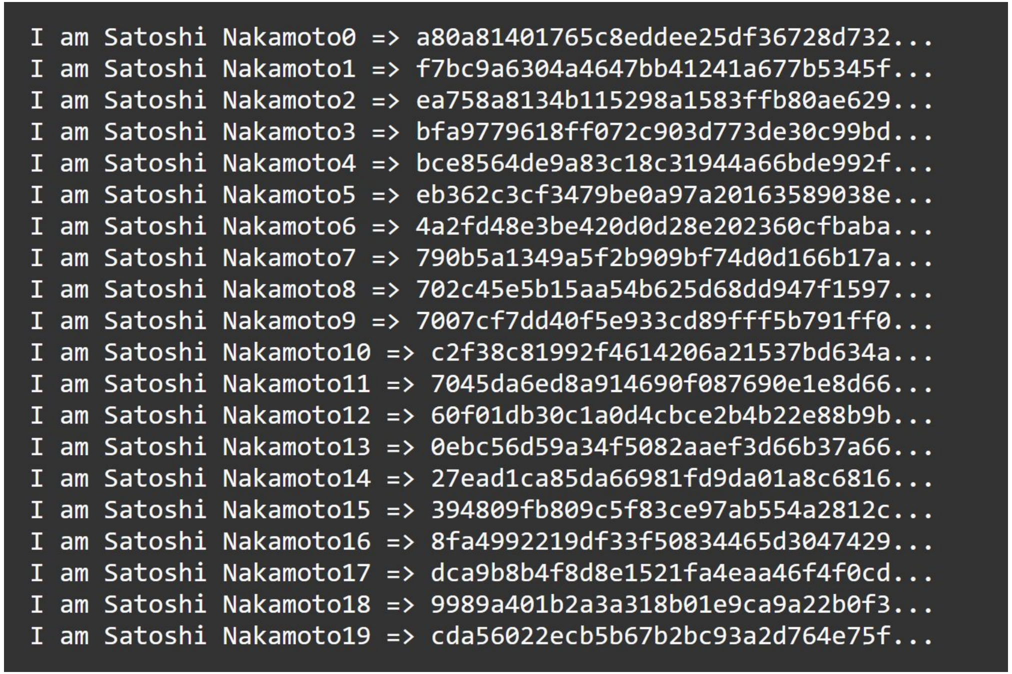
CRYPTOGRAPHIC HASH FUNCTIONS

Cryptographic hash function:

A hash function with three special properties:

- Computationally Efficient
- Collision resistance
- Hide information

The equivalent of **mathematical fingerprints/identifiers**



```
I am Satoshi Nakamoto0 => a80a81401765c8eddee25df36728d732...
I am Satoshi Nakamoto1 => f7bc9a6304a4647bb41241a677b5345f...
I am Satoshi Nakamoto2 => ea758a8134b115298a1583ffb80ae629...
I am Satoshi Nakamoto3 => bfa9779618ff072c903d773de30c99bd...
I am Satoshi Nakamoto4 => bce8564de9a83c18c31944a66bde992f...
I am Satoshi Nakamoto5 => eb362c3cf3479be0a97a20163589038e...
I am Satoshi Nakamoto6 => 4a2fd48e3be420d0d28e202360cfbaba...
I am Satoshi Nakamoto7 => 790b5a1349a5f2b909bf74d0d166b17a...
I am Satoshi Nakamoto8 => 702c45e5b15aa54b625d68dd947f1597...
I am Satoshi Nakamoto9 => 7007cf7dd40f5e933cd89fff5b791ff0...
I am Satoshi Nakamoto10 => c2f38c81992f4614206a21537bd634a...
I am Satoshi Nakamoto11 => 7045da6ed8a914690f087690e1e8d66...
I am Satoshi Nakamoto12 => 60f01db30c1a0d4cbce2b4b22e88b9b...
I am Satoshi Nakamoto13 => 0ebc56d59a34f5082aaef3d66b37a66...
I am Satoshi Nakamoto14 => 27ead1ca85da66981fd9da01a8c6816...
I am Satoshi Nakamoto15 => 394809fb809c5f83ce97ab554a2812c...
I am Satoshi Nakamoto16 => 8fa4992219df33f50834465d3047429...
I am Satoshi Nakamoto17 => dca9b8b4f8d8e1521fa4eaa46f4f0cd...
I am Satoshi Nakamoto18 => 9989a401b2a3a318b01e9ca9a22b0f3...
I am Satoshi Nakamoto19 => cda56022ecb5b67b2bc93a2d764e75f...
```

Image source:
http://chimera.labs.oreilly.com/books/1234000001802/ch08.html#_proof_of_work_algorithm

COMPUTATIONALLY EFFICIENT

CRYPTOGRAPHIC HASH FUNCTIONS

Computationally efficient:

Set of computation to get a digest/hash should not take a long time

Fingerprint analogy:

Whose fingerprint is this?

CRYPTOGRAPHIC HASH FUNCTIONS

Collision Resistance:

It should be hard to find two inputs that maps the same output/hash/Digest. **Output should look random**

<u>Fingerprint analogy:</u>

Can you find two random people with the same fingerprint?

CRYPTOGRAPHIC HASH FUNCTIONS

Hide information:

Given the output, it should be hard to find anything interesting about the input. Ex: even or odd Number

Fingerprint analogy:

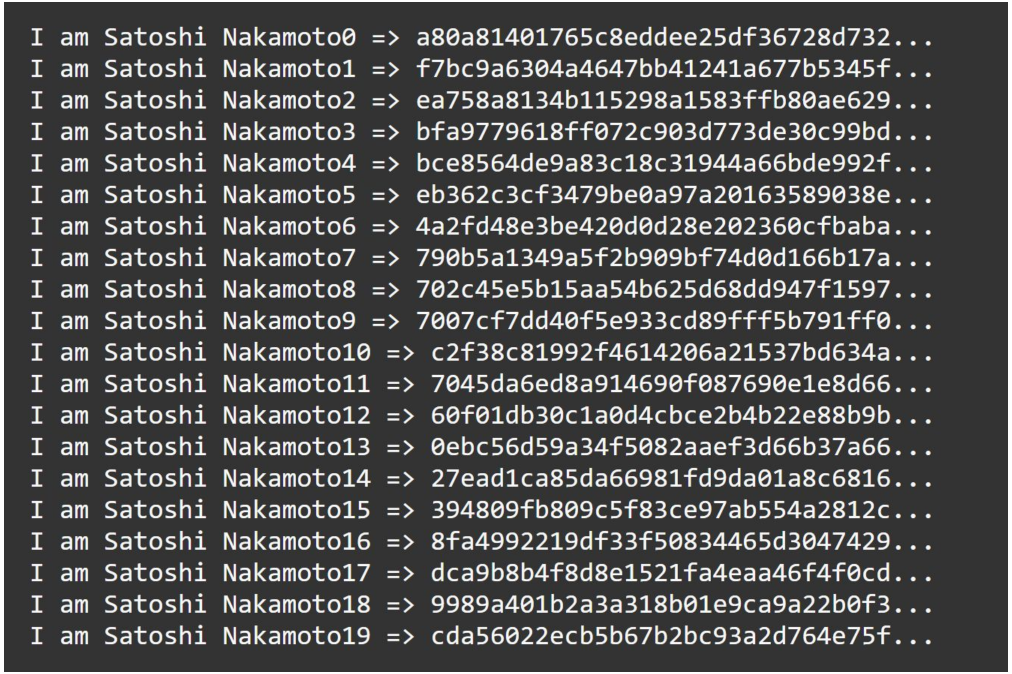
Can you find someone with the same fingerprint as you?

AVALANCHE EFFECT

CRYPTOGRAPHIC HASH FUNCTIONS

Avalanche effect: a small change in the input produces a pseudorandom change in the output

- Often a significant difference from the first output
- Prevents “hot or cold” game with inputs to produce or predict outputs



```
I am Satoshi Nakamoto0 => a80a81401765c8eddee25df36728d732...
I am Satoshi Nakamoto1 => f7bc9a6304a4647bb41241a677b5345f...
I am Satoshi Nakamoto2 => ea758a8134b115298a1583ffb80ae629...
I am Satoshi Nakamoto3 => bfa9779618ff072c903d773de30c99bd...
I am Satoshi Nakamoto4 => bce8564de9a83c18c31944a66bde992f...
I am Satoshi Nakamoto5 => eb362c3cf3479be0a97a20163589038e...
I am Satoshi Nakamoto6 => 4a2fd48e3be420d0d28e202360cfbaba...
I am Satoshi Nakamoto7 => 790b5a1349a5f2b909bf74d0d166b17a...
I am Satoshi Nakamoto8 => 702c45e5b15aa54b625d68dd947f1597...
I am Satoshi Nakamoto9 => 7007cf7dd40f5e933cd89fff5b791ff0...
I am Satoshi Nakamoto10 => c2f38c81992f4614206a21537bd634a...
I am Satoshi Nakamoto11 => 7045da6ed8a914690f087690e1e8d66...
I am Satoshi Nakamoto12 => 60f01db30c1a0d4cbce2b4b22e88b9b...
I am Satoshi Nakamoto13 => 0ebc56d59a34f5082aaef3d66b37a66...
I am Satoshi Nakamoto14 => 27ead1ca85da66981fd9da01a8c6816...
I am Satoshi Nakamoto15 => 394809fb809c5f83ce97ab554a2812c...
I am Satoshi Nakamoto16 => 8fa4992219df33f50834465d3047429...
I am Satoshi Nakamoto17 => dca9b8b4f8d8e1521fa4eaa46f4f0cd...
I am Satoshi Nakamoto18 => 9989a401b2a3a318b01e9ca9a22b0f3...
I am Satoshi Nakamoto19 => cda56022ecb5b67b2bc93a2d764e75f...
```

CRYPTOGRAPHIC HASH FUNCTIONS

SHA-256: A cryptographic hash function designed by the NSA

Bitcoin uses **SHA-256^2** (“`SHA-256 squared`”), meaning that `H(x)` actually means `SHA256(SHA256(x))`

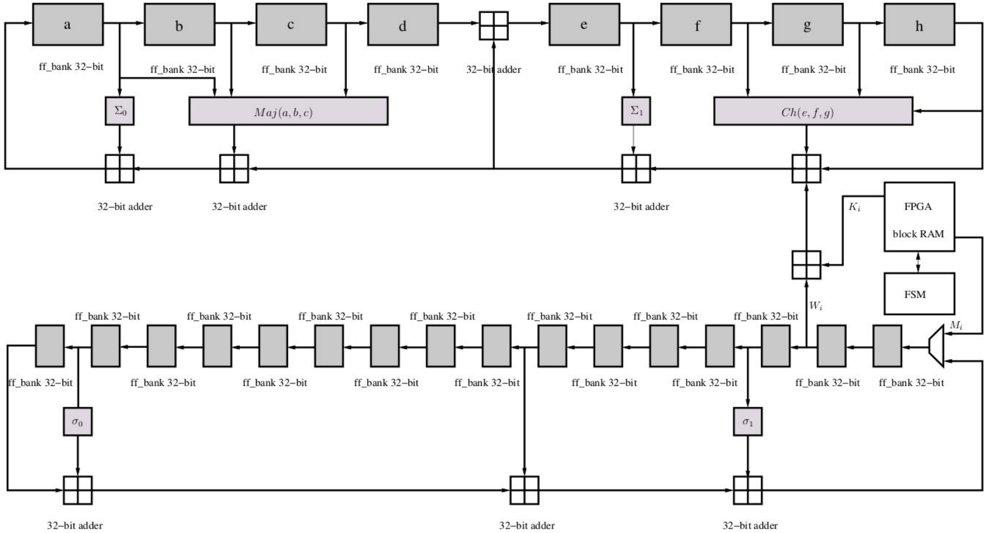
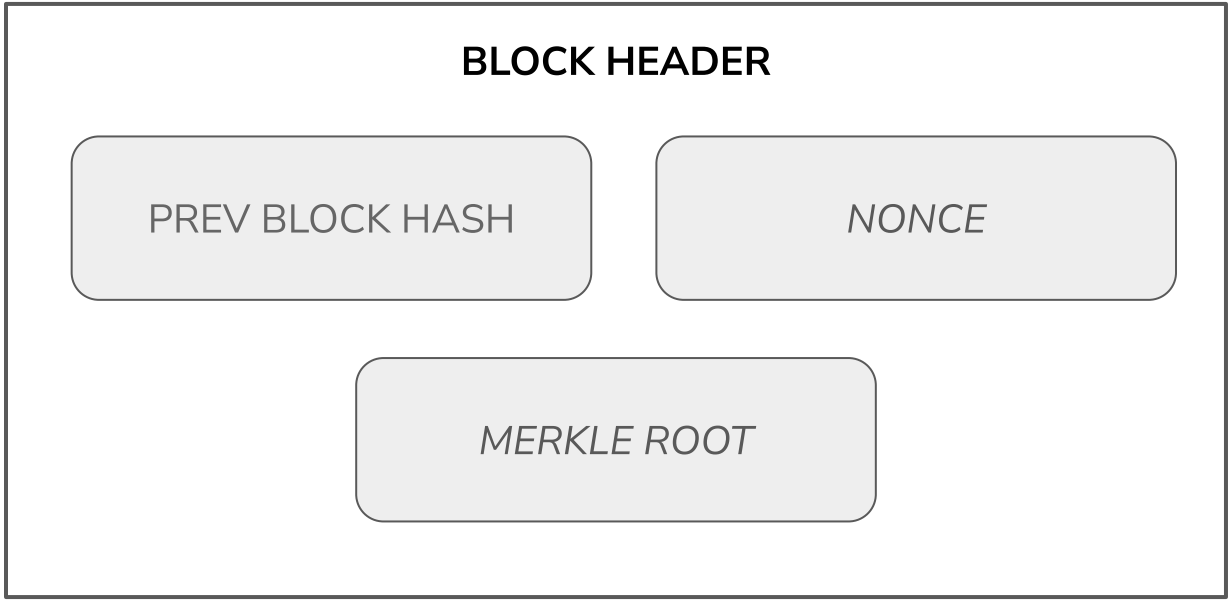


Image source: https://opencores.org/usercontent,img,1375985843

QUESTIONS?

A TAMPER EVIDENT DATABASE

A TAMPER-EVIDENT DATABASE



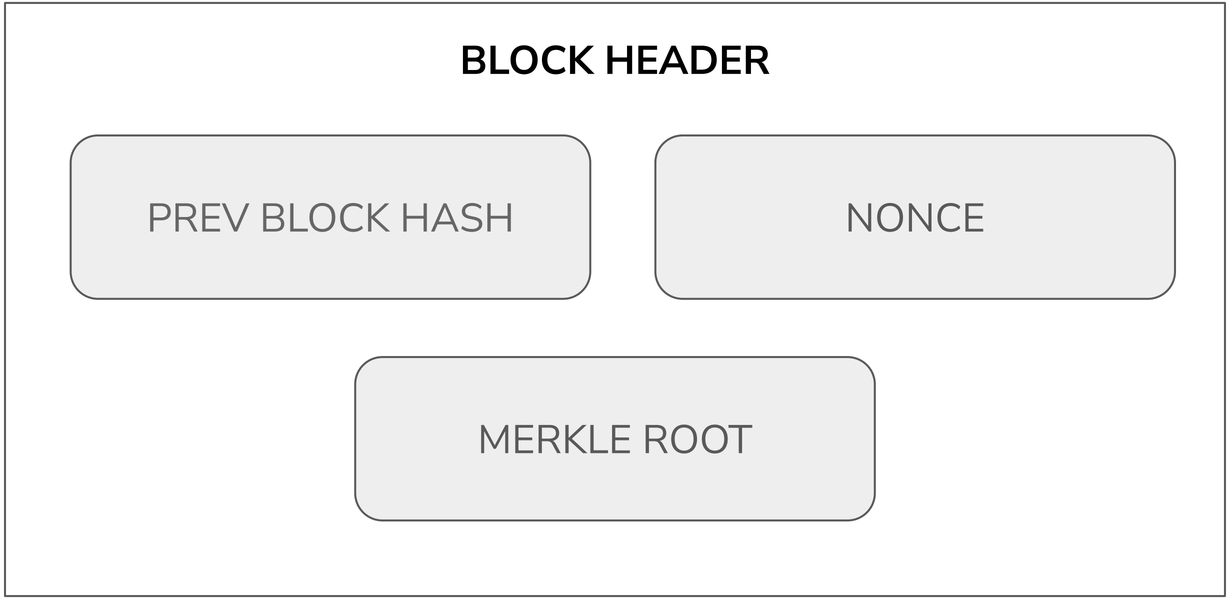
BLOCK HEADER

PREV BLOCK HASH

NONCE

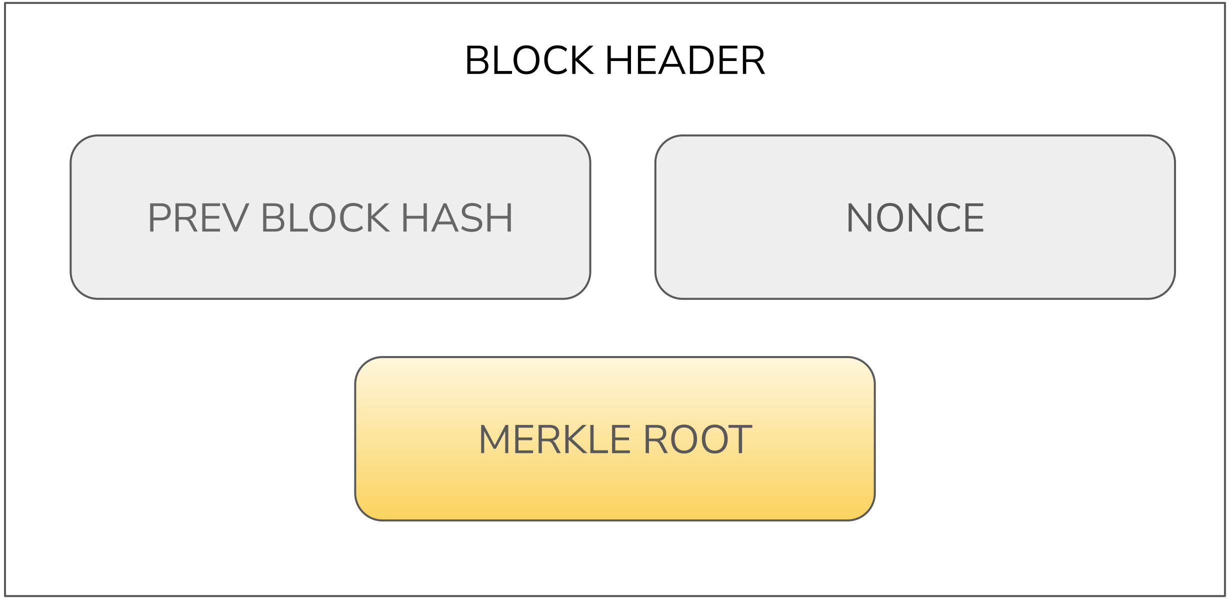
MERKLE ROOT

A TAMPER-EVIDENT DATABASE



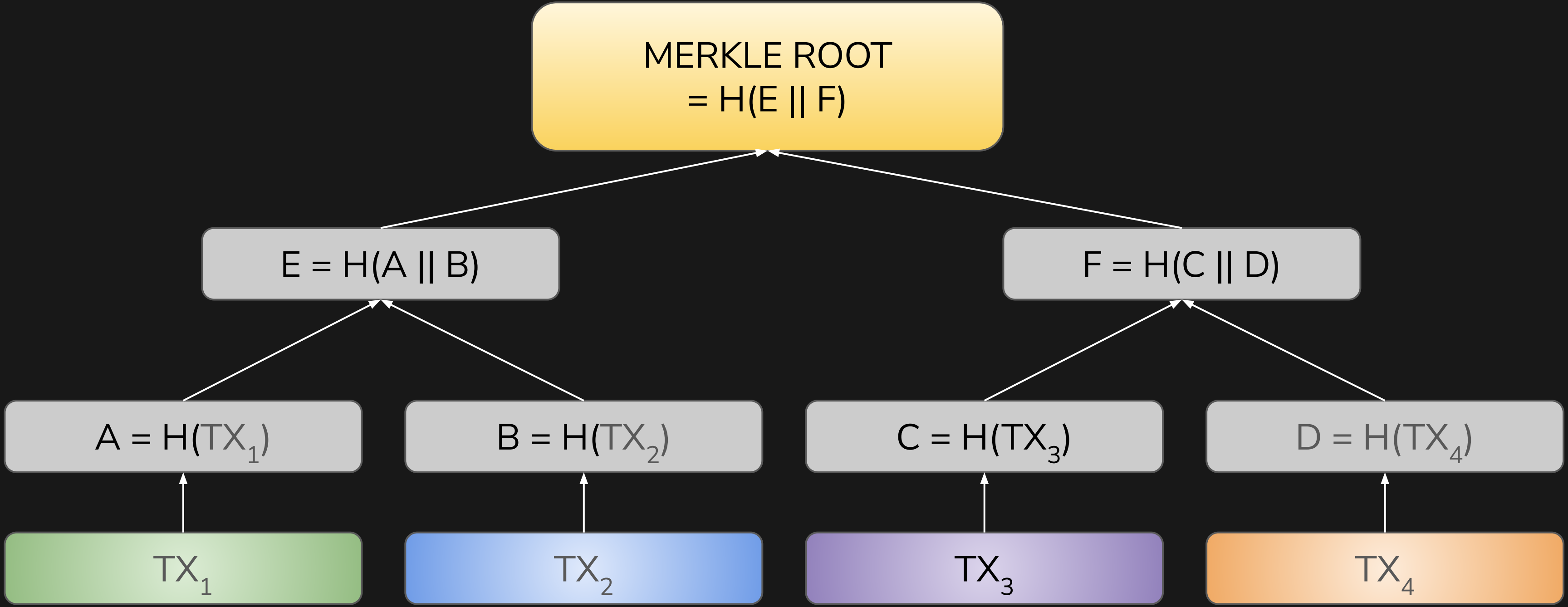
```
blockID = H(blockHeader) = H(prevBlockHash || merkleRoot || nonce)
```

A TAMPER-EVIDENT DATABASE



MERKLE ROOT

A TAMPER-EVIDENT DATABASE

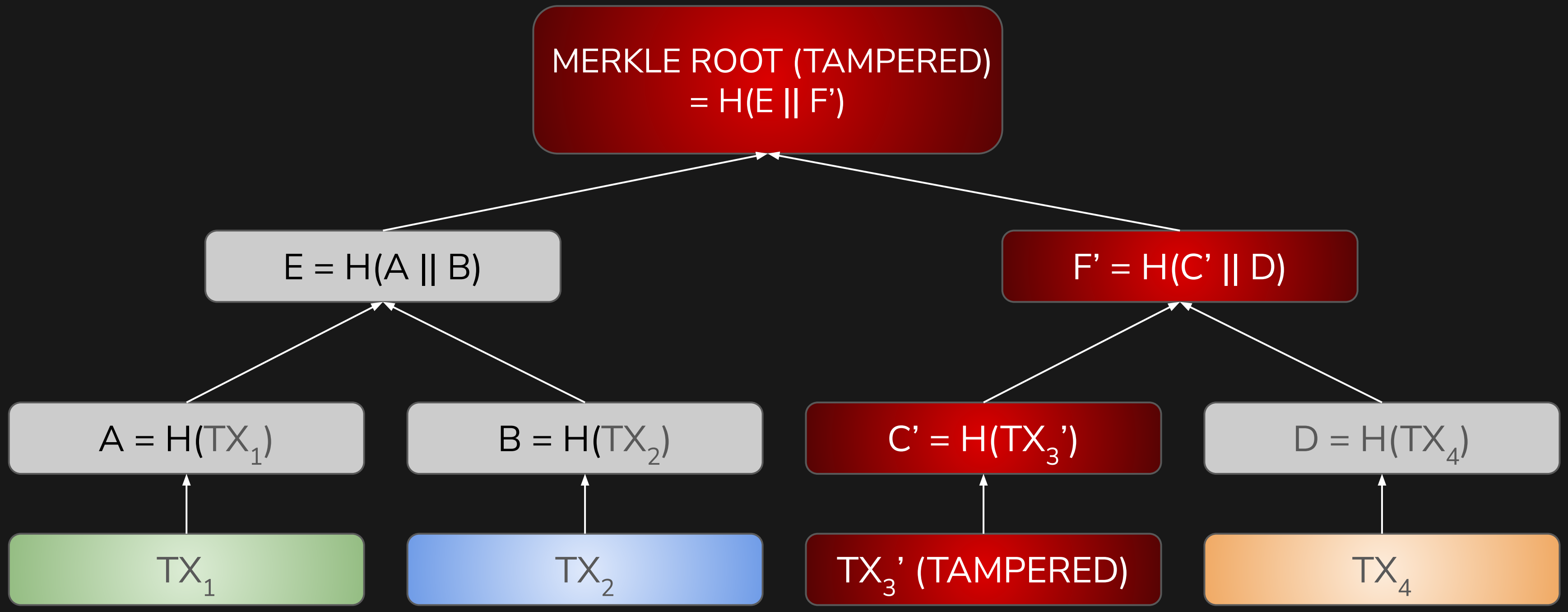


AUTHOR: NADIR AKHTAR

BLOCKCHAIN AT BERKELEY

MERKLE ROOT

A TAMPER-EVIDENT DATABASE

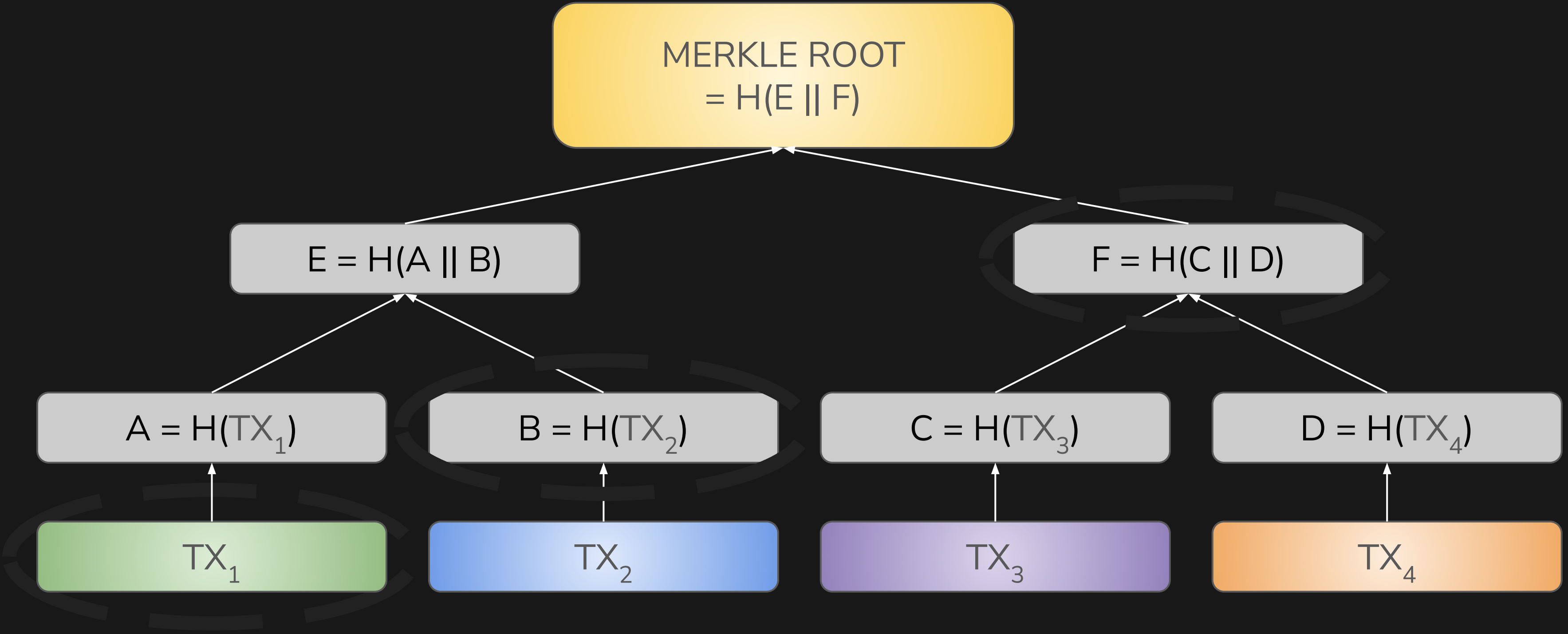


AUTHOR: NADIR AKHTAR

BLOCKCHAIN AT BERKELEY

MERKLE BRANCH AND PROOF OF INCLUSION

A TAMPER-EVIDENT DATABASE

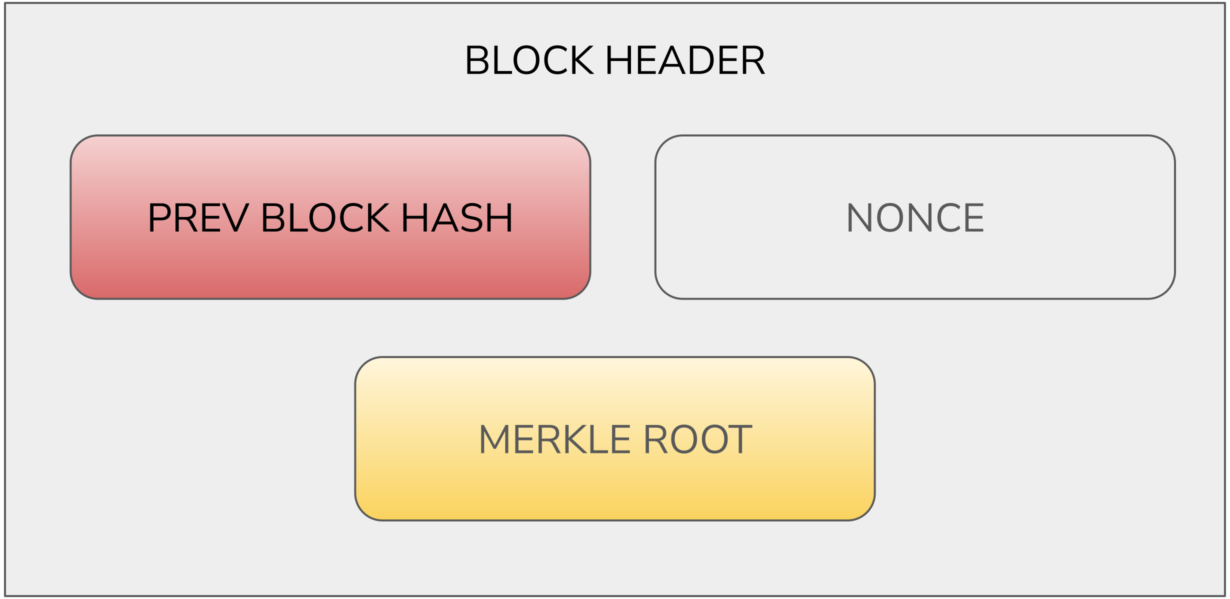


AUTHOR: NADIR AKHTAR

BLOCKCHAIN AT BERKELEY

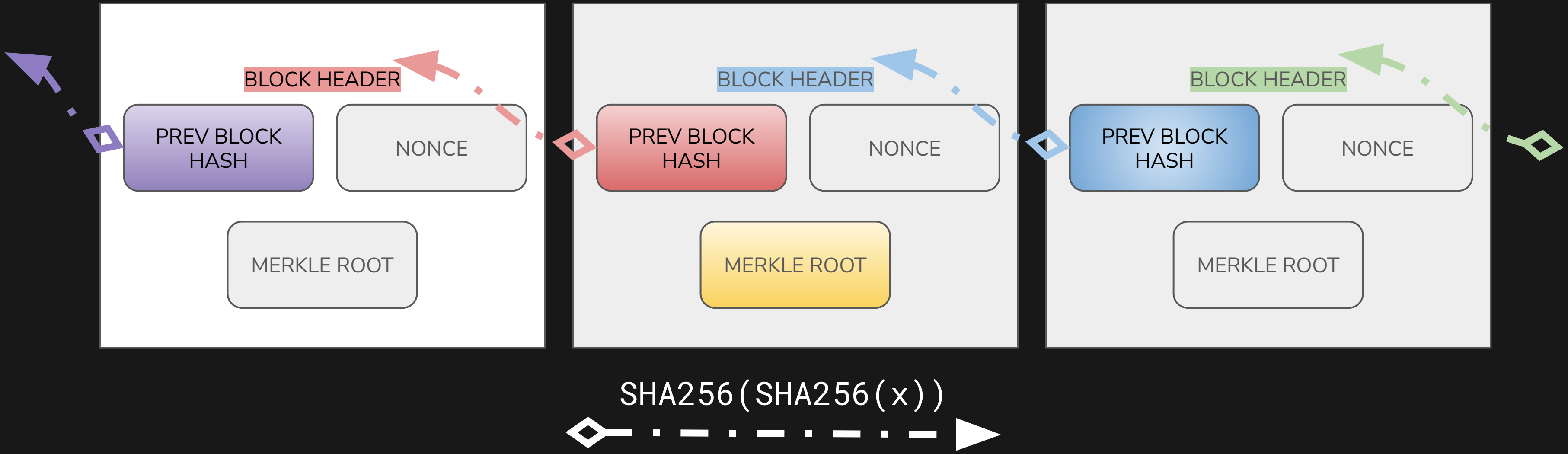
QUESTIONS?

A TAMPER-EVIDENT DATABASE



PROTECTING THE CHAIN

A TAMPER-EVIDENT DATABASE



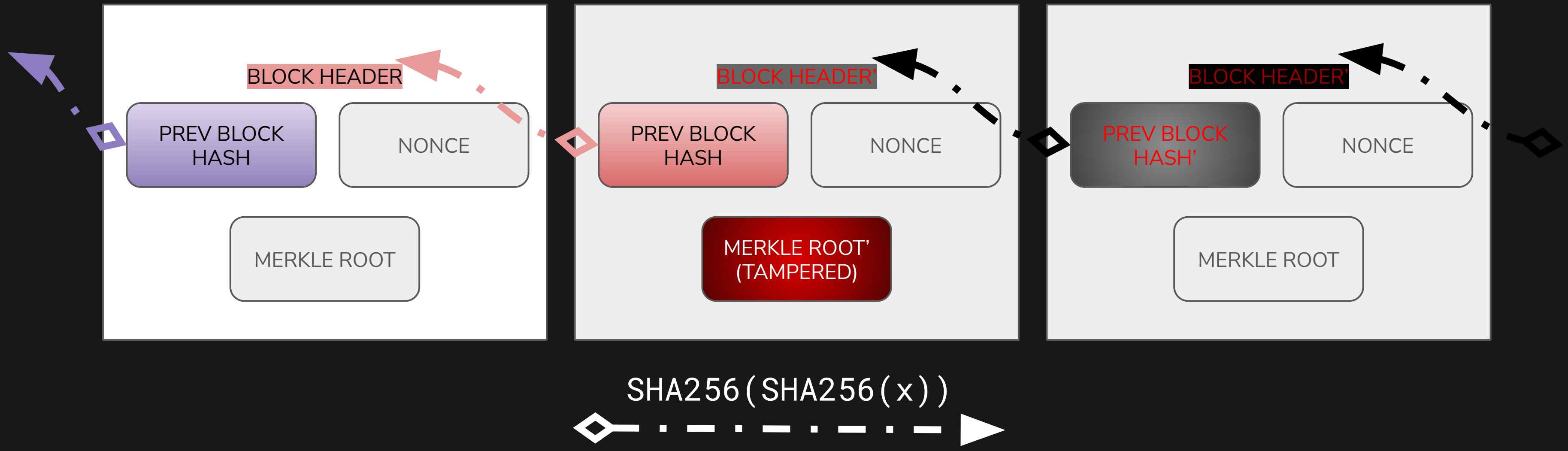
```
prevBlockHash = H(prevBlockHash || merkleRoot || nonce)
```

AUTHOR: NADIR AKHTAR

BLOCKCHAIN AT BERKELEY

PROTECTING THE CHAIN

A TAMPER-EVIDENT DATABASE



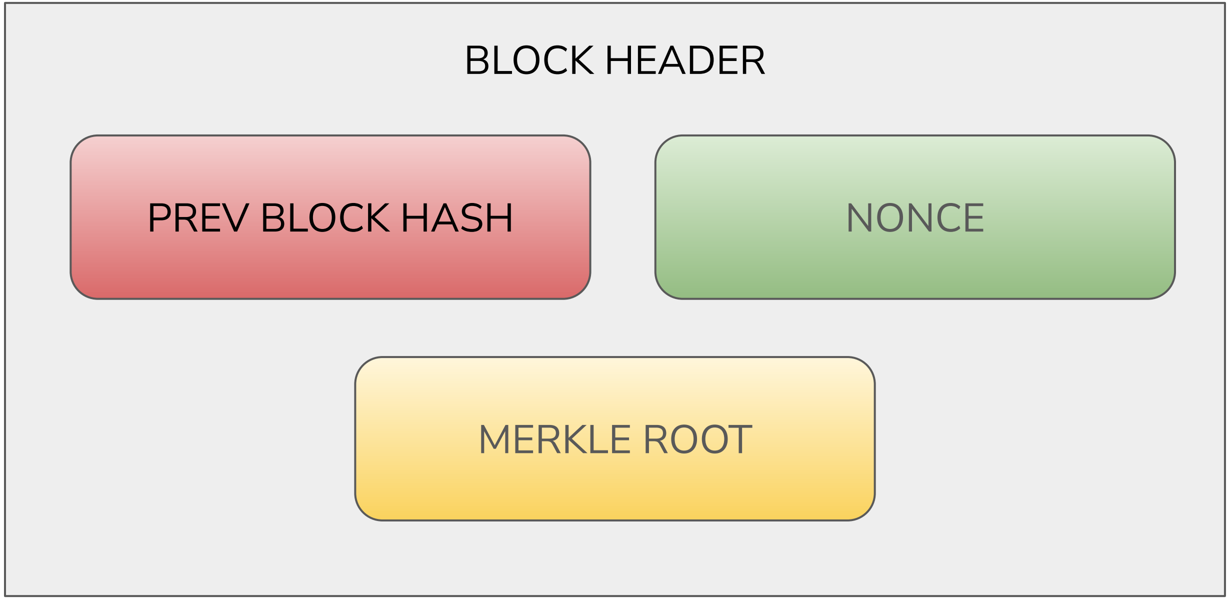
```
prevBlockHash = H(prevBlockHash || merkleRoot || nonce)
```

AUTHOR: NADIR AKHTAR

BLOCKCHAIN AT BERKELEY

QUESTIONS?

A TAMPER-EVIDENT DATABASE



A TAMPER-EVIDENT DATABASE

- Bitcoin's Proof of Work consensus requires miners to solve a computationally difficult puzzle

Hash puzzles need to be:

1. Computationally difficult.
2. Adjustable
3. Easily verifiable.

HASH PUZZLE

A TAMPER-EVIDENT DATABASE

Bitcoin's partial preimage hash puzzle: A problem with a requirement to find a nonce that satisfies the following inequality:

```
H(prevBlockHash || merkleRoot || nonce) < target
```

AUTHOR: NADIR AKHTAR

BLOCKCHAIN AT BERKELEY

HASH PUZZLE

A TAMPER-EVIDENT DATABASE

Bitcoin’s partial preimage hash puzzle:

```
H(prevBlockHash || merkleRoot || nonce) <
0x0000ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

AUTHOR: JUSTIN YU

BLOCKCHAIN
AT BERKELEY

HASH PUZZLE

A TAMPER-EVIDENT DATABASE

```
H(prevBlockHash || merkleRoot || nonce)
```

↓

```
H(“Hello, world!0”)
```

```
0x1312af178c253f84028d480a6adc1e25e81caa44c749ec81976192e2ec934c64
```

<

```
0x0000ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

HASH PUZZLE

A TAMPER-EVIDENT DATABASE

```
H(prevBlockHash || merkleRoot || nonce)
```

↓

```
H(“Hello, world!1”)
```

```
0xe9afc424b79e4f6ab42d99c81156d3a17228d6e1eef4139be78e948a9332a7d8
<
0x0000ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

HASH PUZZLE

A TAMPER-EVIDENT DATABASE

```
H(prevBlockHash || merkleRoot || nonce)
```

↓

```
H(“Hello, world!4250”)
```

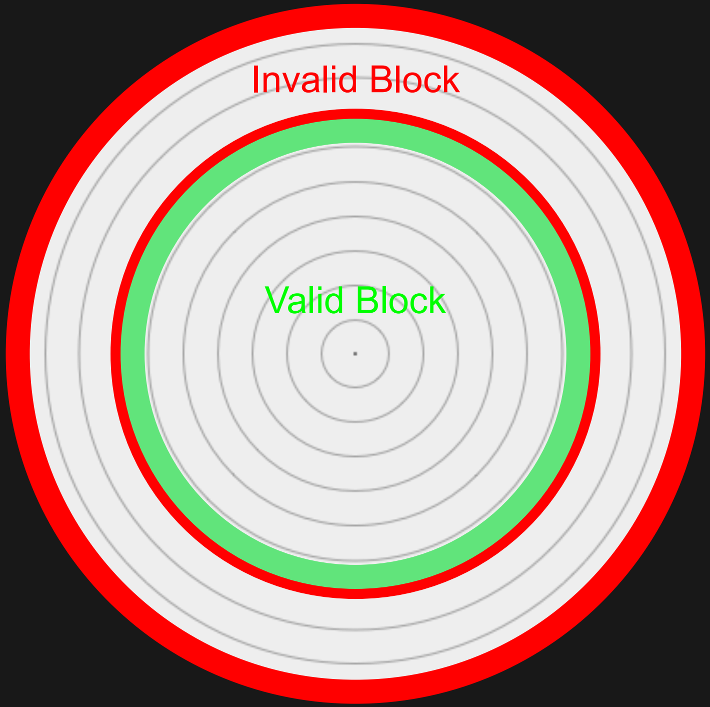
```
0x0000c3af42fc31103f1fdc0151fa747ff87349a4714df7cc52ea464e12dcd4e9
<
0x0000ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

Solved!

MINING

A TAMPER-EVIDENT DATABASE

- **Mining** is like throwing darts at a target while blindfolded:
  - Equal likelihood of hitting any part of the target
  - Faster throwers ⇒ more hits / second
- Miners look for a hash below an algorithmically decided target



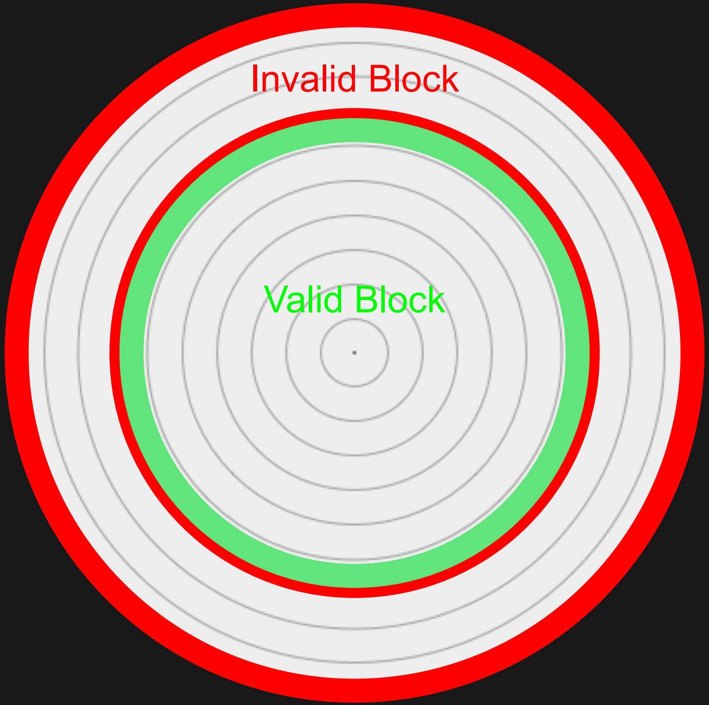
AUTHOR: NADIR AKHTAR

```
H(prevBlockHash || merkleRoot || nonce) < target
```

BLOCKCHAIN AT BERKELEY

BLOCK DIFFICULTY

A TAMPER-EVIDENT DATABASE



Difficulty: A representation of the expected number of computations required to find a block

- Implemented as requirement of leading number of 0s
- Adjusts with global hashrate
- Adjusts every 2016 blocks (~2 weeks)

AUTHOR: NADIR AKHTAR

```
H(prevBlockHash || merkleRoot || nonce) < target
```

BLOCKCHAIN AT BERKELEY

ADJUSTING DIFFICULTY

A TAMPER-EVIDENT DATABASE

```
H(prevBlockHash || merkleRoot || nonce) <
```

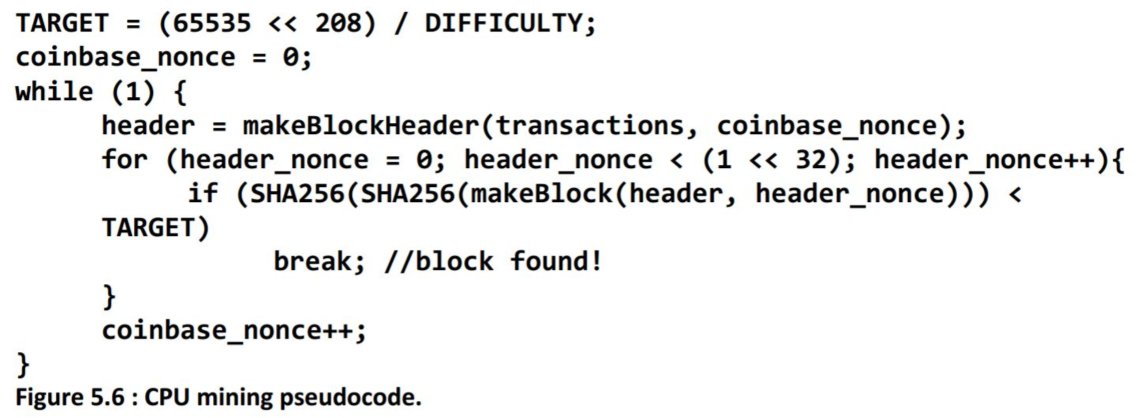
```
a.  0x0000ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff

b.  0x000000000000ffffffffffffffffffffffffffffffffffffffffffffffffffff

c.  0x00000000ffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

MINING PSEUDOCODE

A TAMPER-EVIDENT DATABASE



```
TARGET = (65535 << 208) / DIFFICULTY;
coinbase_nonce = 0;
while (1) {
        header = makeBlockHeader(transactions, coinbase_nonce);
        for (header_nonce = 0; header_nonce < (1 << 32); header_nonce++){
                if (SHA256(SHA256(makeBlock(header, header_nonce))) <
        TARGET)
                        break; //block found!
        }
        coinbase_nonce++;
}
```

Figure 5.6 : CPU mining pseudocode.

Source: (from Princeton Textbook, 5.2)

EXAMPLE BLOCK

A TAMPER-EVIDENT DATABASE

REAL BITCOIN BLOCK EXAMPLE

https://www.blockchain.com/explorer

Source:
https://blockchain.info/block/00000000000000000013942c4215cd92306bbce769cfcb349d0b42f031c994eb

QUESTIONS?

BREAK SECTION

BLOCKCHAIN

AT BERKELEY

BLOCKCHAIN
AT BERKELEY

SIGS, ECDSA, AND ADDRESSES

INTRO TO DIGITAL SIGNATURE ALGORITHMS

Dilemma:

When sending transactions to other users, we want 2 seemingly contradictory things to happen:

1. Tie user identity to a transaction
2. Have no sensitive identifiable characteristics associated with a particular transaction

INTRO TO DIGITAL SIGNATURE ALGORITHMS

Public Key Cryptography: a cryptographic system that allows for secure dissemination of identity and authentication of valid messages.

INTRO TO DIGITAL SIGNATURE ALGORITHMS

1. **Public Key:** information about a user that can be distributed widely

2. **Private Key:** sensitive information about a user that should be only known by the user

ECDSA (ELLIPTIC CURVE DSA)

- **<u>Elliptic Curve Digital Signature Algorithm (ECDSA):</u>**
    - the algorithm the Bitcoin network uses to generate public keys and verify transactions.
    - a variant of standard DSA but with elliptic curves

SIGNATURE GENERATION AND VERIFICATION

ECDSA IN ACTION

ALICE

BOB

Private Key

Alice uses a random number generator to create a private key

AUTHOR: PHILIP HAYES
UPDATED: HAENA LEE

Disclaimer: this is not the complete story of what happens when making a transaction on the Bitcoin network. This is a general representation of transaction verification using ECDSA.

SIGNATURE GENERATION AND VERIFICATION

ECDSA IN ACTION

ALICE

BOB

Private Key

Public Key

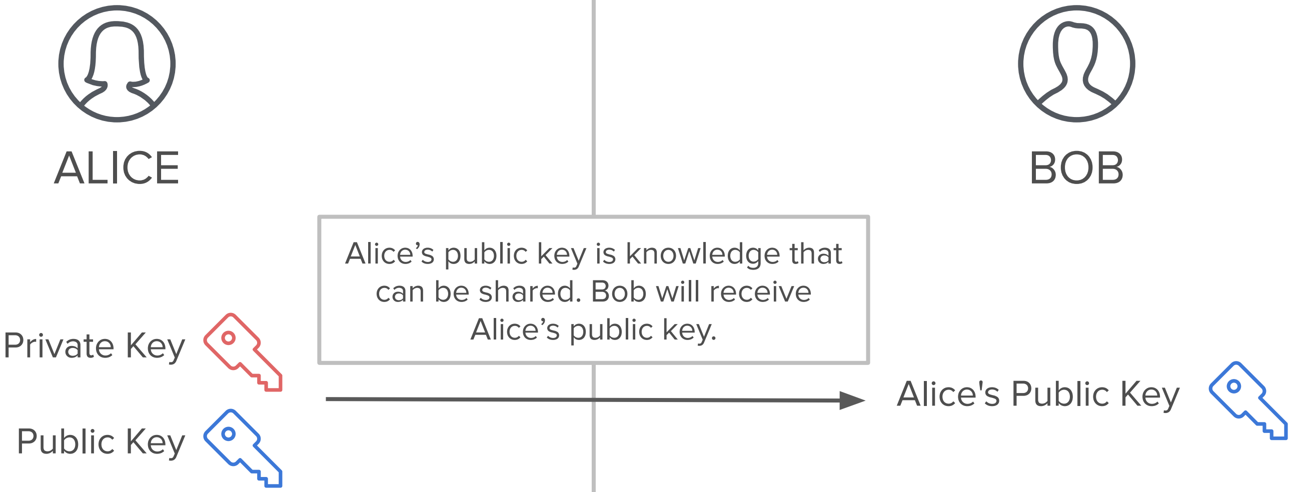
Alice uses ECDSA to calculate public key

Disclaimer: this is not the complete story of what happens when making a transaction on the Bitcoin network. This is a general representation of transaction verification using ECDSA.

AUTHOR: PHILIP HAYES
UPDATED: HAENA LEE

SIGNATURE GENERATION AND VERIFICATION

ECDSA IN ACTION



ALICE

BOB

Private Key

Public Key

Alice’s public key is knowledge that can be shared. Bob will receive Alice’s public key.

Alice's Public Key

AUTHOR: PHILIP HAYES
UPDATED: HAENA LEE

SIGNATURE GENERATION AND VERIFICATION

ECDSA IN ACTION

ALICE

Private Key

Public Key

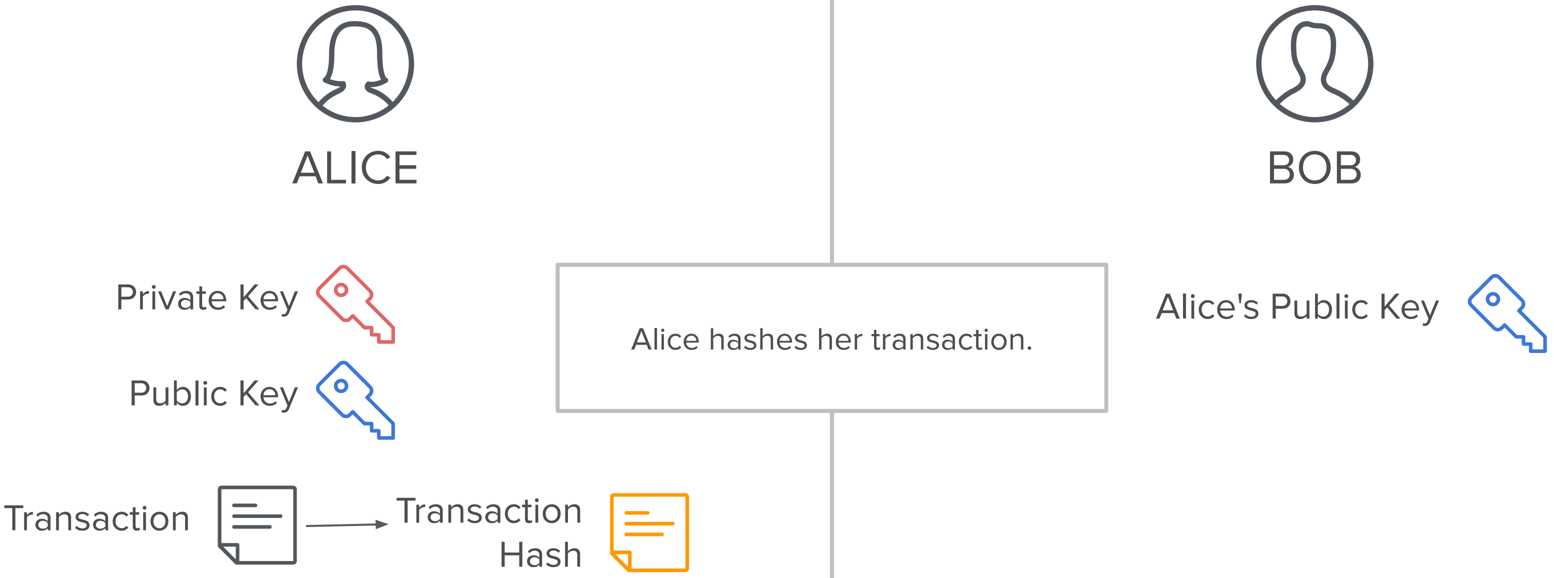
Transaction

Alice has some transaction to send to Bob.

BOB

Alice's Public Key

ECDSA IN ACTION



Alice hashes her transaction.

AUTHOR: PHILIP HAYES
UPDATED: HAENA LEE

SIGNATURE GENERATION AND VERIFICATION

ECDSA IN ACTION

ALICE

Private Key 🔑

Public Key 🔑

Transaction 📄

Transaction Hash 📄

Signature 🏷 = 🔑 + 📄

BOB

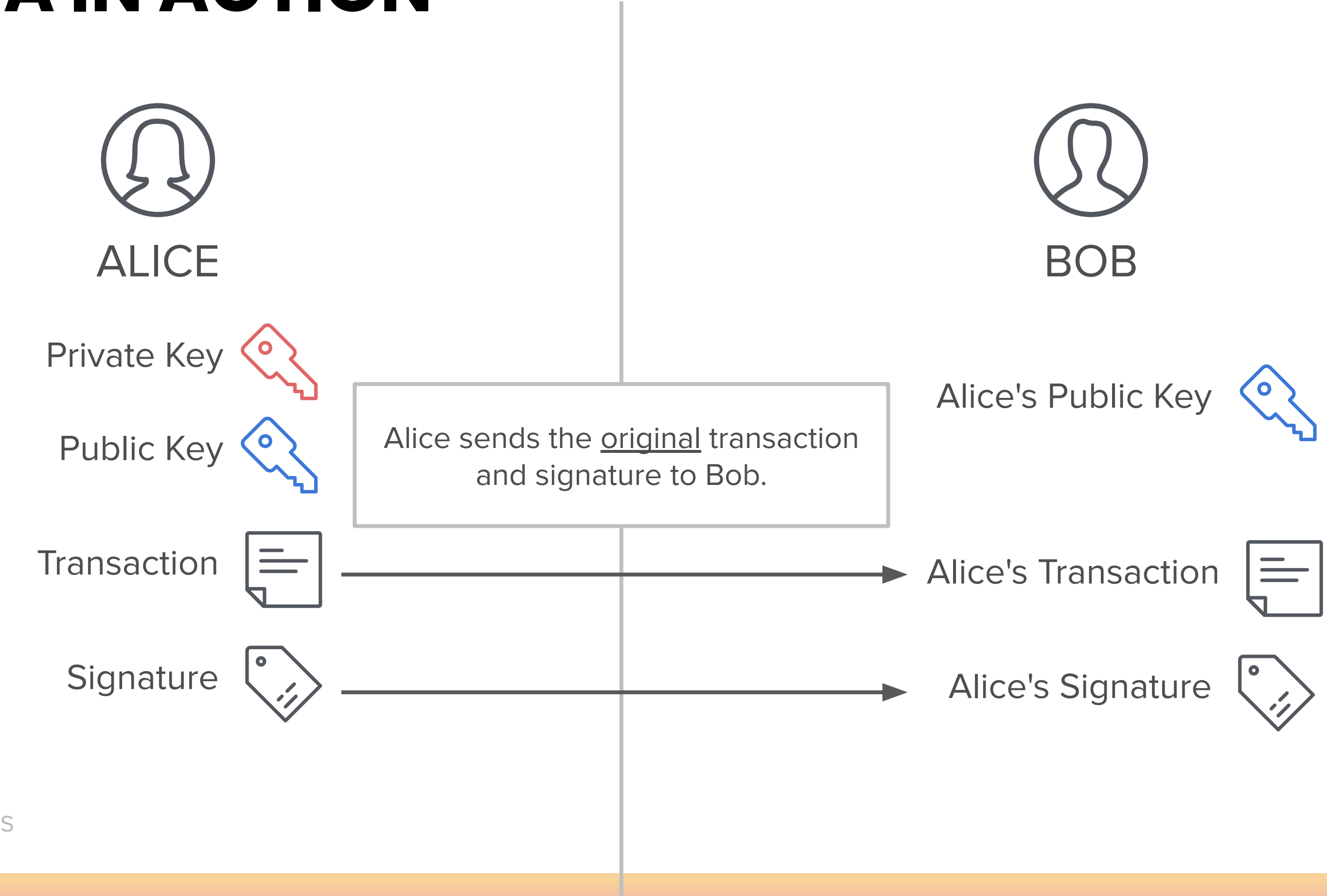
Alice's Public Key 🔑

Alice produces a signature using the hash, private key, and elliptic curve.

AUTHOR: PHILIP HAYES
UPDATED: HAENA LEE

SIGNATURE GENERATION AND VERIFICATION

ECDSA IN ACTION

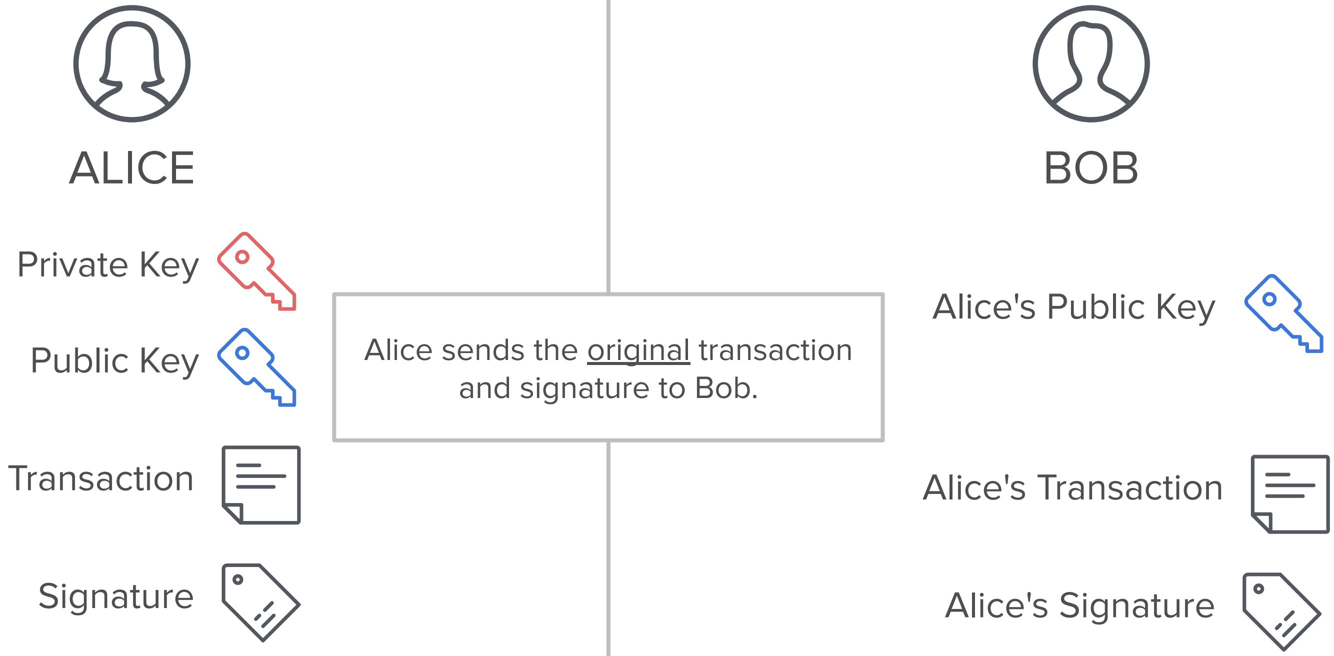


Alice sends the original transaction and signature to Bob.

AUTHOR: PHILIP HAYES
UPDATED: HAENA LEE

SIGNATURE GENERATION AND VERIFICATION

ECDSA IN ACTION



ALICE

Private Key

Public Key

Transaction

Signature

Alice sends the original transaction and signature to Bob.

BOB

Alice's Public Key

Alice's Transaction

Alice's Signature

AUTHOR: PHILIP HAYES
UPDATED: HAENA LEE

SIGNATURE GENERATION AND VERIFICATION

ECDSA IN ACTION

ALICE

Private Key

Public Key

Transaction

Signature

Bob will be able to verify if Alice created the signature.

BOB

Alice's Public Key

Alice's Transaction

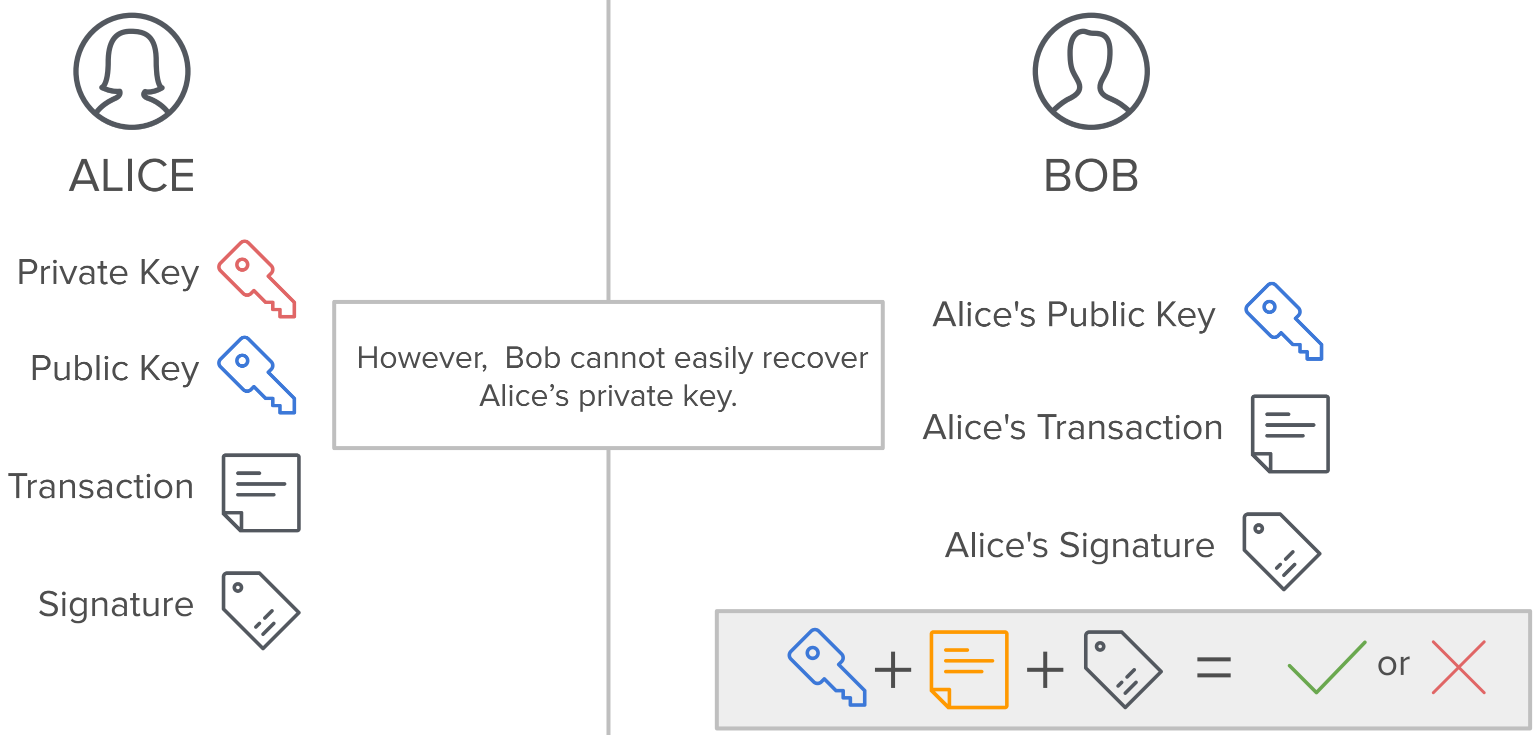
Alice's Signature

🔑 + 📄 + 🏷 = ✓ or ✗

AUTHOR: PHILIP HAYES
UPDATED: HAENA LEE

SIGNATURE GENERATION AND VERIFICATION

ECDSA IN ACTION



However, Bob cannot easily recover Alice’s private key.

SIGNATURE GENERATION AND VERIFICATION

ECDSA IN ACTION

ALICE

Private Key

Public Key

Transaction

Signature

EVE

Eve’s Private Key

Eve’s Public Key

BOB

Alice's Public Key

AUTHOR: GILLIAN CHU
UPDATED: HAENA LEE

SIGNATURE GENERATION AND VERIFICATION

ECDSA IN ACTION

ALICE

Private Key

Public Key

Transaction

Signature

EVE

Eve’s Private Key

Eve’s Public Key

private key + message = signature

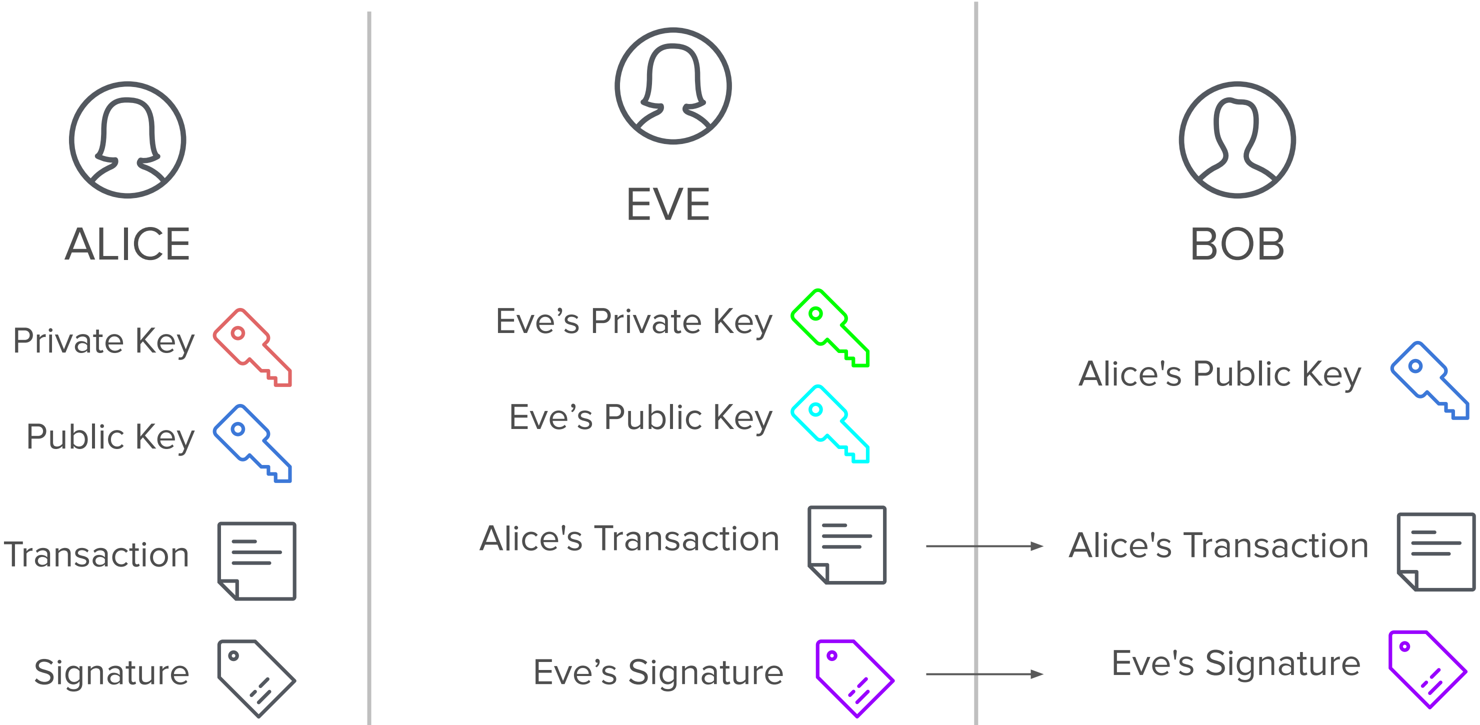
BOB

Alice's Public Key

AUTHOR: GILLIAN CHU
UPDATED: HAENA LEE

SIGNATURE GENERATION AND VERIFICATION

ECDSA IN ACTION



ALICE

Private Key

Public Key

Transaction

Signature

EVE

Eve’s Private Key

Eve’s Public Key

Alice's Transaction

Eve’s Signature

BOB

Alice's Public Key

Alice's Transaction

Eve's Signature

AUTHOR: GILLIAN CHU
UPDATED: HAENA LEE

SIGNATURE GENERATION AND VERIFICATION

ECDSA IN ACTION

ALICE

Private Key

Public Key

Transaction

Signature

EVE

Eve’s Private Key

Eve’s Public Key

Alice's Transaction

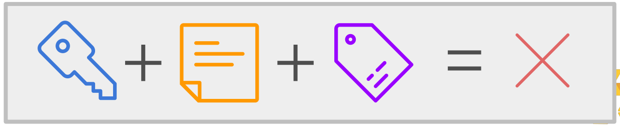
Eve’s Signature

BOB

Alice's Public Key

Alice's Transaction

Eve's Signature



AUTHOR: GILLIAN CHU
UPDATED: HAENA LEE

GUARANTEED PROPERTIES

ECDSA SUMMARY

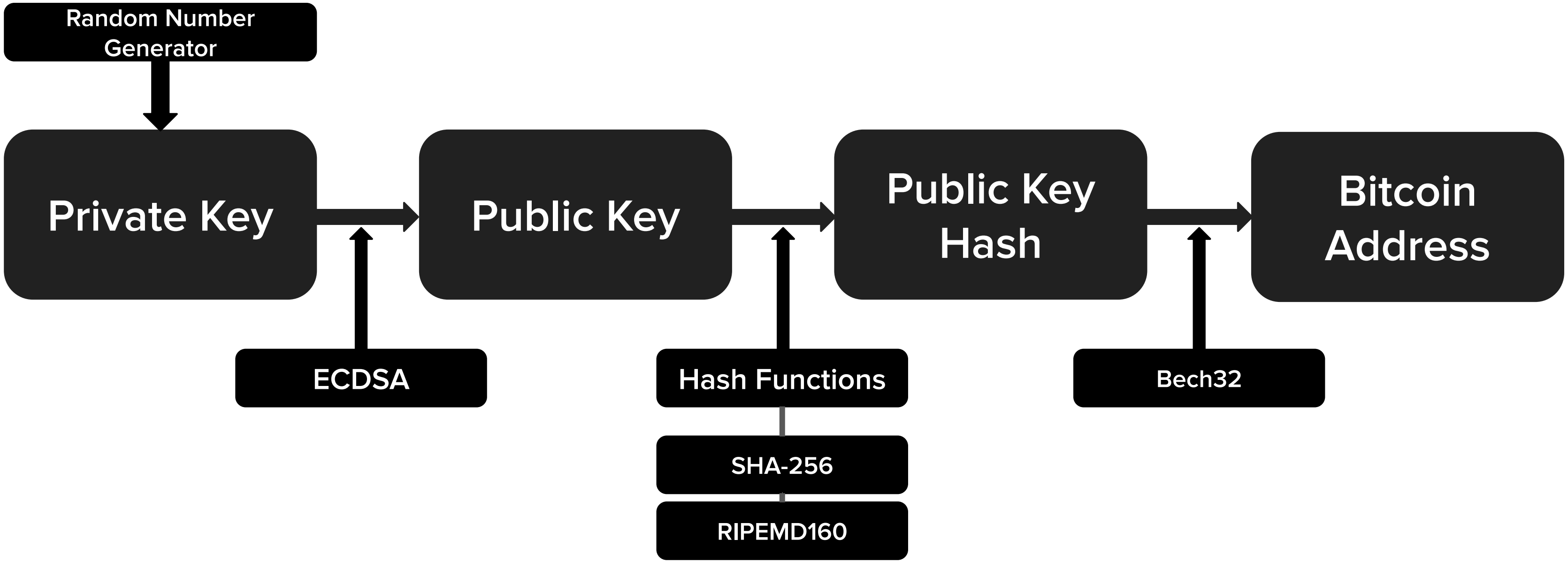
Recipients given the (message, signature) pair should be able to verify:

- **Message Origin**: original sender (owner of private key) has authorized this message/transaction
- **Non-repudiation**: original sender (owner of private key) cannot backtrack
- **Message Integrity**: transaction cannot have been modified since sending

QUESTIONS?

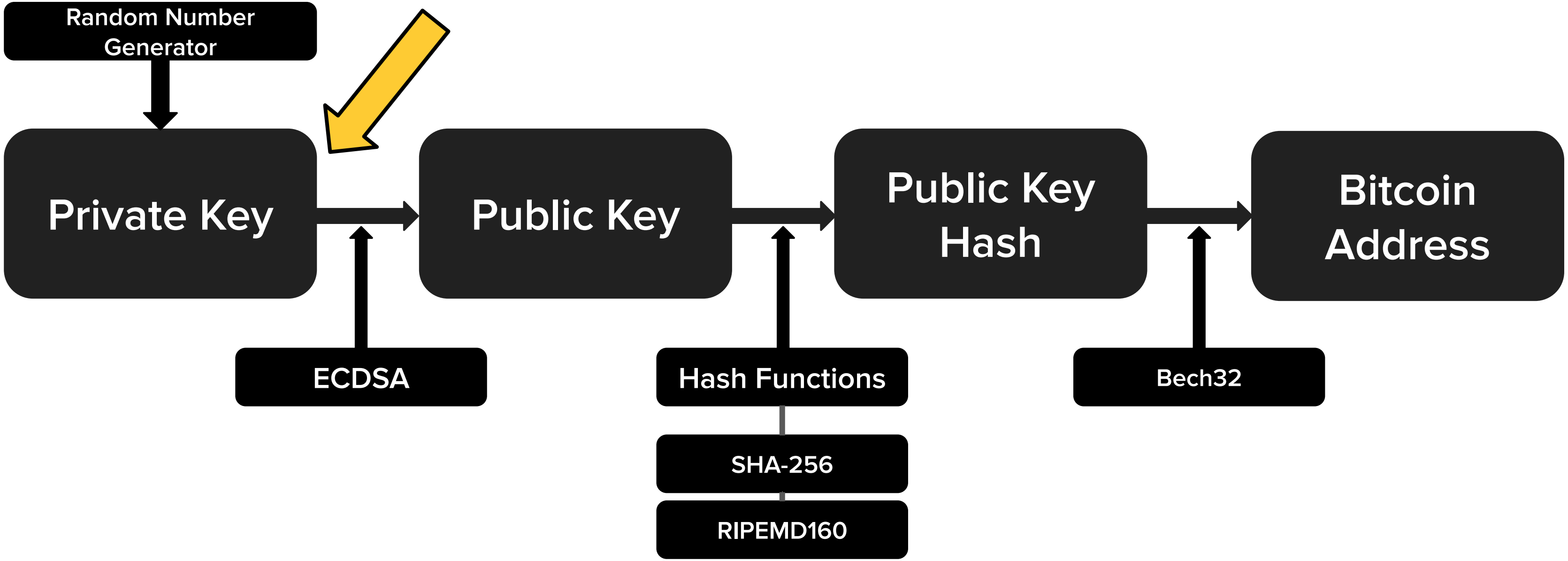
LAYERS AND LAYERS UPON MYSTERY

CONVERSION SUMMARY



LAYERS AND LAYERS UPON MYSTERY

CONVERSION SUMMARY



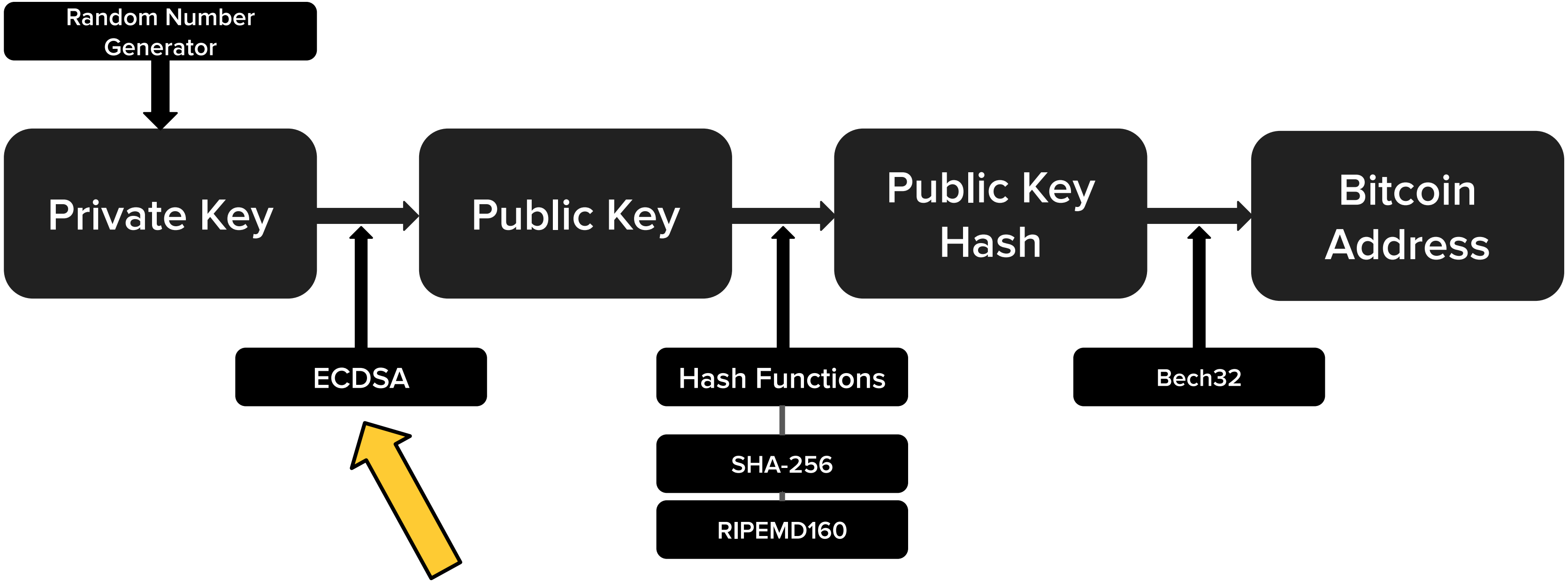
RANDOM NUMBER GENERATORS ARE NOT CREATED THE SAME

PRIVATE KEY GENERATION

- Private Keys on Bitcoin: 256-bit unsigned integers
- Private keys on Bitcoin are **not** generated using regular random number generators.
- Chances of two different individuals generating the same private key (collision) is extremely low.
  - ~2^{256} unique private keys
  - Chances of collision: $(1/2^{256}) * (1/2^{256})$ = really, really low

LAYERS AND LAYERS UPON MYSTERY

CONVERSION SUMMARY

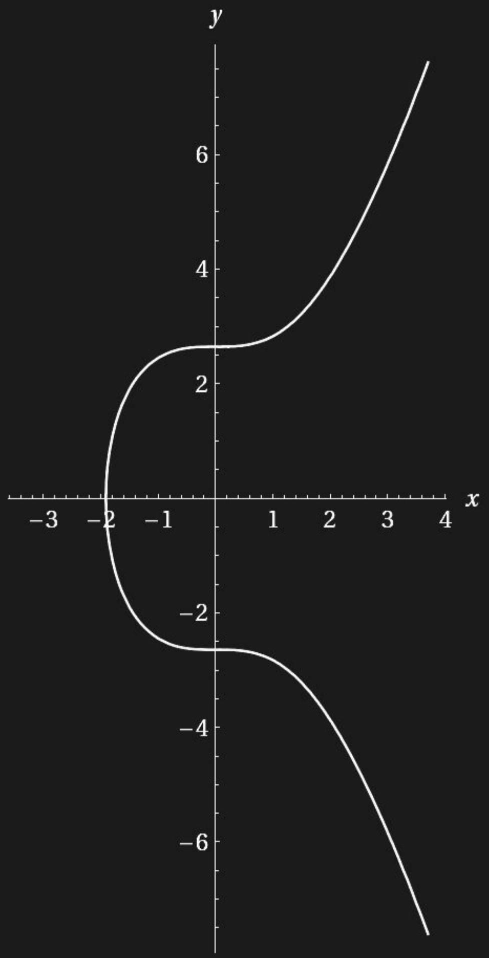


ELLIPTIC CURVES (SECP256K1)

PUBLIC KEY GENERATION

- Bitcoin uses **ECDSA** (Elliptic Curve Digital Signature Algorithm) to produce public keys
- The Elliptic Curve is defined by some mathematical function
  - **Bitcoin's Elliptic Curve**:

    secp256k1 : $Y^2 = (X^3 + 7)$ over ($\mathbf{F}_p$)

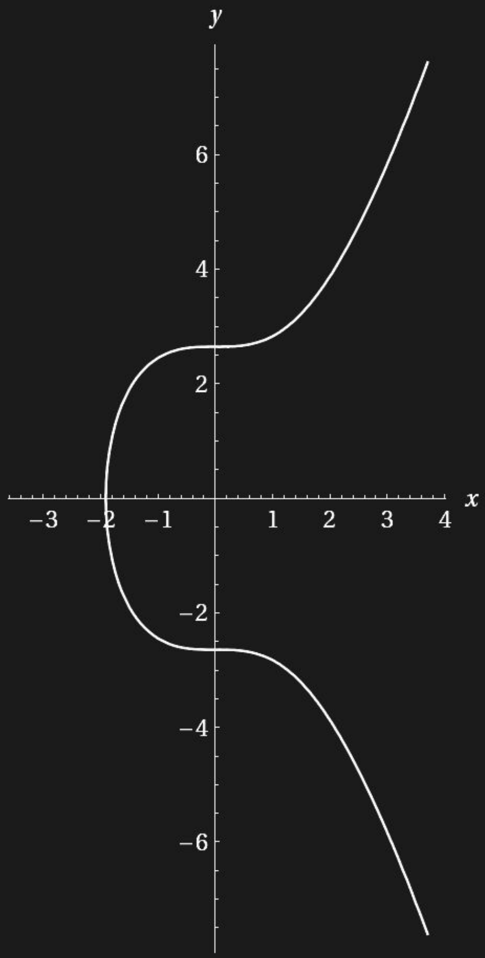


y^2 = x^3 + 7 | Computed by Wolfram|Alpha

ELLIPTIC CURVE SCALAR MULTIPLICATION

PUBLIC KEY GENERATION

- Using a private key as an input, we can generate a public key by performing point multiplication/elliptic curve scalar multiplication.
    - Key thing to note here: point multiplication is a trapdoor function.
    - This means calculating the public key is a one-way function.



y^2 = x^3 + 7 | Computed by Wolfram|Alpha

TIME COMPLEXITY

PUBLIC KEY GENERATION

Input: public key
Output: corresponding private key

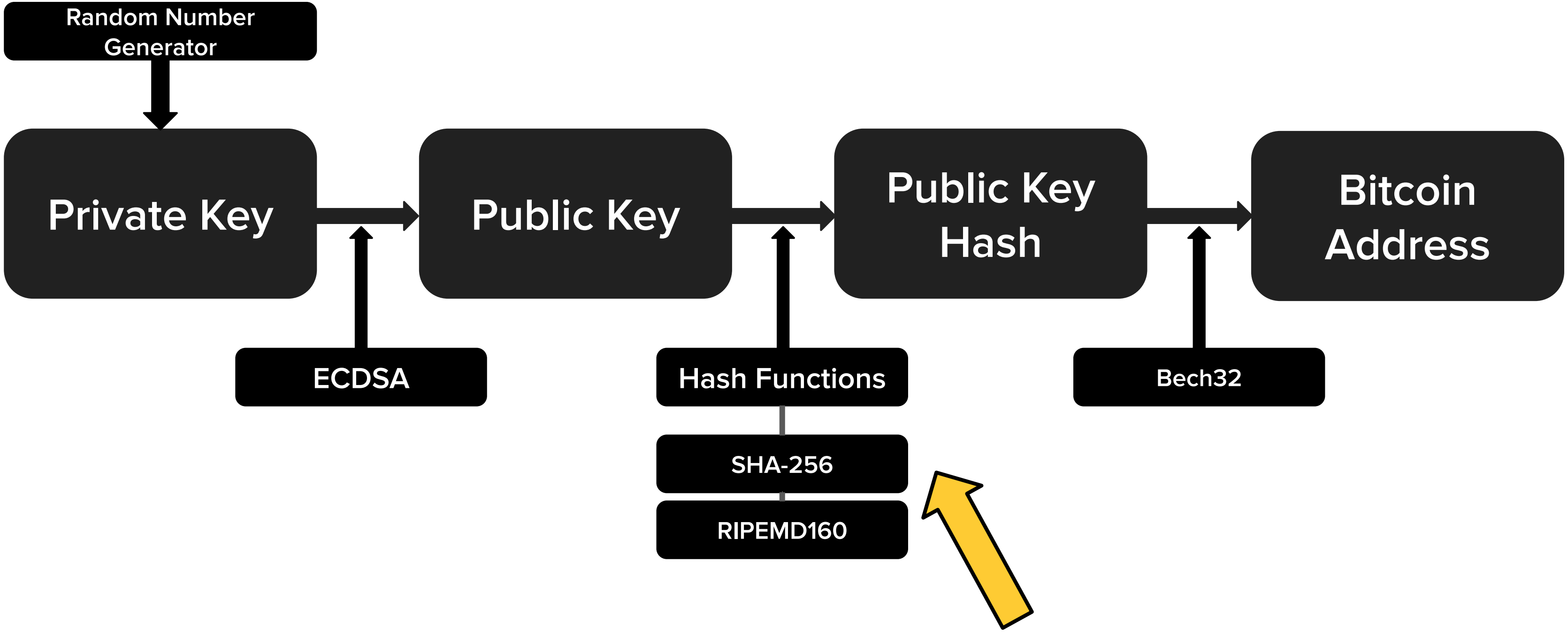
256 bit private key, takes O(sqrt(n)) operations to crack
15 * pow(2,40) hashes per second on the ENTIRE Bitcoin network

pow(2,128) / (15 * pow(2,40)) / 3600 / 24 / 365.25
= 0.653799211229596e18

650 million billion years

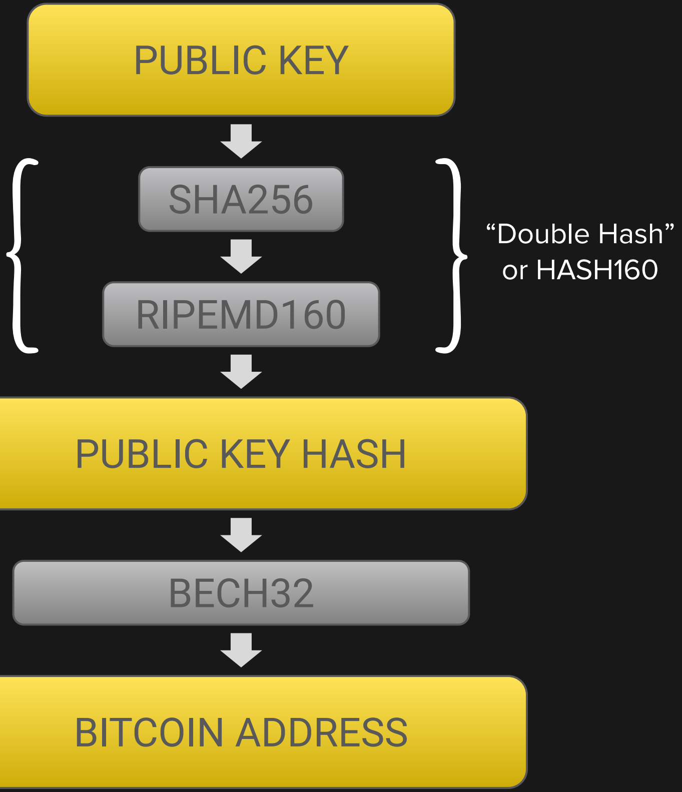
LAYERS AND LAYERS UPON MYSTERY

CONVERSION SUMMARY



PUBKEY TO PUBKEY HASH

PUBLIC KEY TO BITCOIN ADDRESS



```
PUBKEYHASH = RIPEMD160(SHA256(K))
```

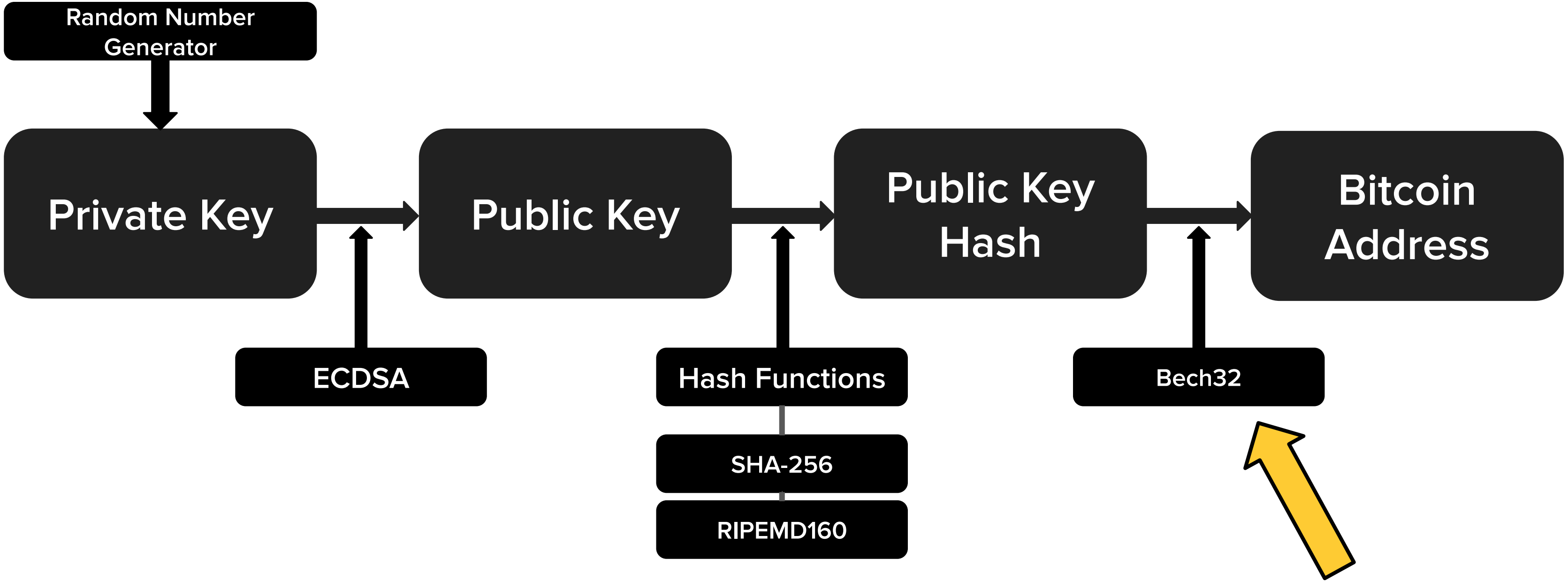
- SHA-256 (Secure Hashing Algorithm)
  - Used extensively in bitcoin scripts and mining
- RIPEMD (RACE Integrity Primitives Evaluation Message Digest)
  - Produces 160-bit (20-byte) number

AUTHOR: GLORIA ZHAO
UPDATED: GILLIAN CHU

BLOCKCHAIN AT BERKELEY

LAYERS AND LAYERS UPON MYSTERY

CONVERSION SUMMARY



QUESTIONS?

BITCOIN SCRIPT

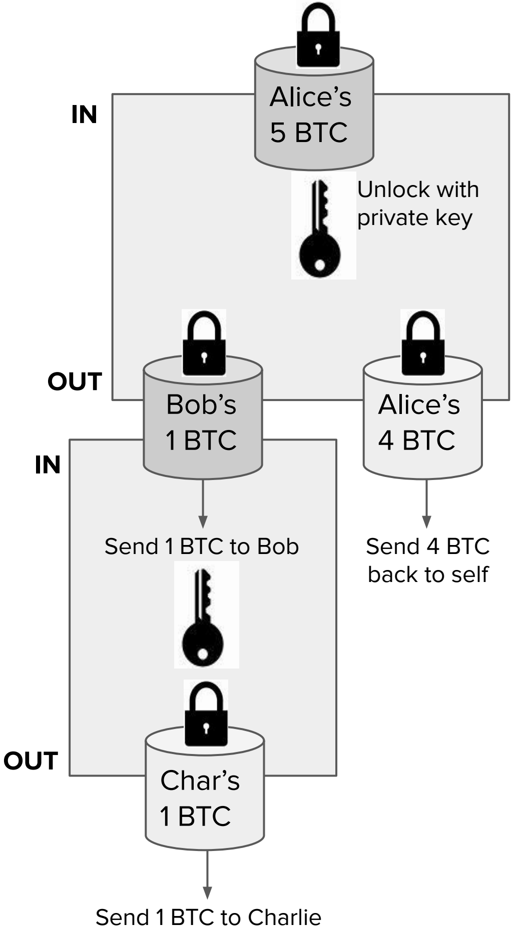
REMEMBER THE UTXO MODEL?

BITCOIN SCRIPT

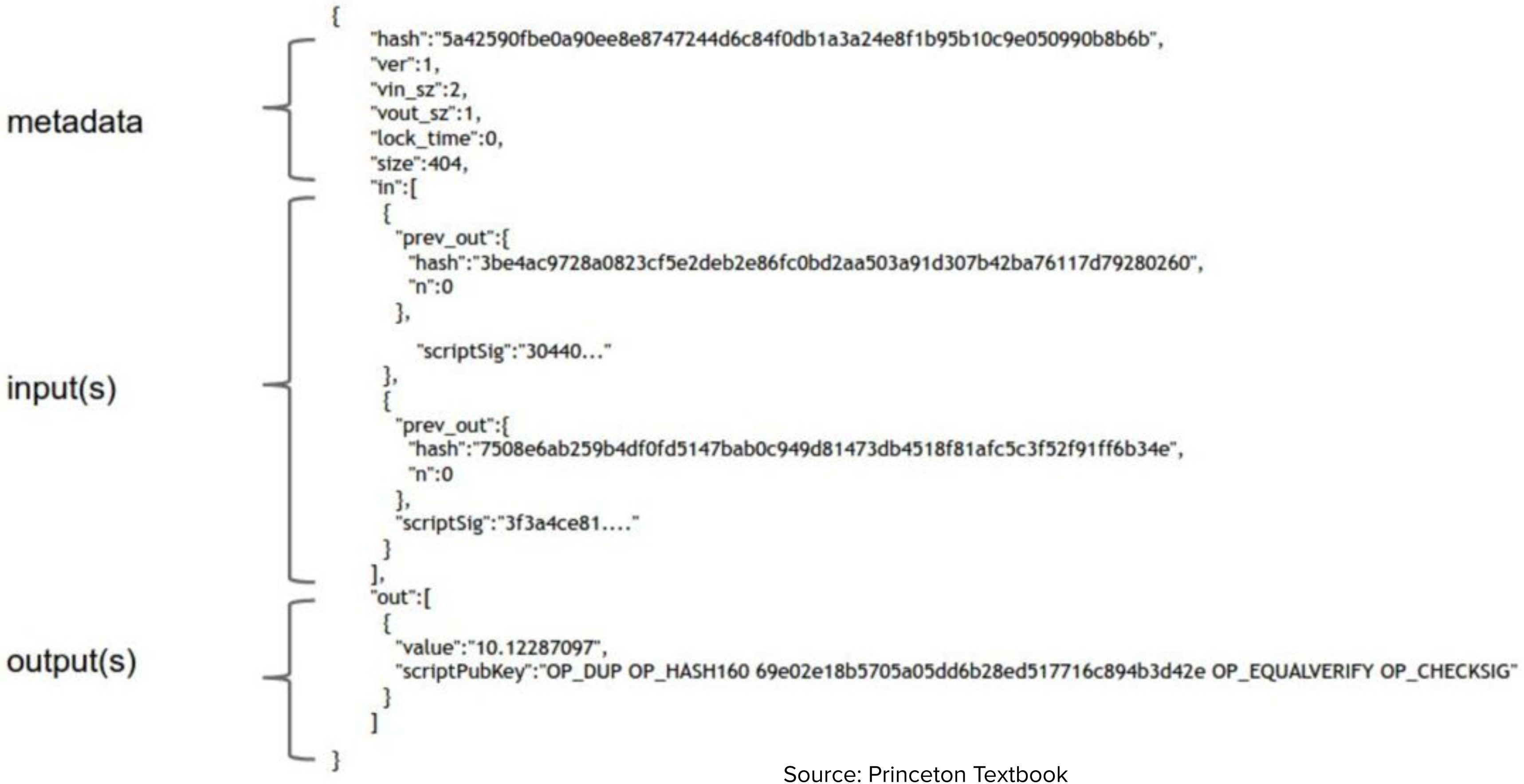
Reminders:

- Bitcoin uses a UTXO model
- Transactions map inputs to outputs,

- Transactions contain signature of owner of funds
- Spending Bitcoin is **redeeming** previous transaction outputs



CONTENTS OF A TRANSACTION


metadata
input(s)
output(s)

```
{
  "hash":"5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
  "ver":1,
  "vin_sz":2,
  "vout_sz":1,
  "lock_time":0,
  "size":404,
  "in":[
    {
     "prev_out":{
       "hash":"3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
       "n":0
     },

        "scriptSig":"30440..."
    },
    {
     "prev_out":{
       "hash":"7508e6ab259b4df0fd5147bab0c949d81473db4518f81afc5c3f52f91ff6b34e",
       "n":0
     },
     "scriptSig":"3f3a4ce81...."
    }
  ],
  "out":[
    {
     "value":"10.12287097",
     "scriptPubKey":"OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
    }
  ]
}
```

Source: Princeton Textbook

CONTENTS OF A TRANSACTION – METADATA

```
{
  “hash”: “5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b”,
  “ver”: 1,
  “vin_sz”: 2,
  “vout_sz”: 1,
  “lock_time”: 0,
  “size”: 404,
  “in”: [
        {
              “prev_out”: {
                    “hash”: “3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260”,
                    “n”: 0
              },
                    “scriptSig”: “30440…”
        },
        {
              “prev_out”: {
                    “hash”: “7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e”,
                    “n”: 0
              },
              “scriptSig”: “3f3a4ce81….”
        }
  ],
  “out”: [
        {
              “value”: 10.12287097”,
              “scriptPubKey”: “OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG”
        }
  ]
}
```

hash or “ID”
of this transaction

Source: Princeton Textbook

CONTENTS OF A TRANSACTION – METADATA

```
{
“hash”: “5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b”,
“ver”: 1,
“vin_sz”: 2,
“vout_sz”: 1,
“lock_time”: 0,
“size”: 404,
“in”: [
        {
                “prev_out”: {
                        “hash”: “3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260”,
                        “n”: 0
                },
                        “scriptSig”: “30440…”
        },
        {
                “prev_out”: {
                        “hash”: “7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e”,
                        “n”: 0
                },
                “scriptSig”: “3f3a4ce81….”
        }
],
“out”: [
        {
                “value”: 10.12287097”,
                “scriptPubKey”: “OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY
OP_CHECKSIG”
        }
]
```

size (number) of inputs

size (number) of outputs

hash or “ID”
of this transaction

Source: Princeton Textbook

CONTENTS OF A TRANSACTION – METADATA

```
{
  "hash": "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
  "ver": 1,
  "vin_sz": 2,
  "vout_sz": 1,
  "lock_time": 0,
  "size": 404,
  "in": [
          {
                    "prev_out": {
                              "hash": "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
                              "n": 0
                    },
                              "scriptSig": "30440…"
          },
          {
                    "prev_out": {
                              "hash": "7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e",
                              "n": 0
                    },
                    "scriptSig": "3f3a4ce81…."
          }
  ],
  "out": [
          {
                    "value": 10.12287097",
                    "scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
          }
  ]
}
```

version

size (number) of inputs

size (number) of outputs

lock time (useful for scripting)

size of transaction

hash or "ID"
of this transaction

Source: Princeton Textbook

CONTENTS OF A TRANSACTION – INPUTS

```
“hash”: “5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b”,
    “ver”: 1,
    “vin_sz”: 2,
    “vout_sz”: 1,
  “in”: [
      {
            “prev_out”: {
                  “hash”: “3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260”,
                  “n”: 0
            },
                  “scriptSig”: “30440…”
      },
      {
            “prev_out”: {
                  “hash”: “7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e”,
                  “n”: 0
            },
            “scriptSig”: “3f3a4ce81….”
      }
  ],
```

remember these?

Source: Princeton Textbook

CONTENTS OF A TRANSACTION – INPUTS

```
"hash": "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
    "ver": 1,
    "vin_sz": 2,
    "vout_sz": 1,
"in": [
    {
        "prev_out": {
            "hash": "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
            "n": 0
        },
            "scriptSig": "30440…"
    },
    {
        "prev_out": {
            "hash": "7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e",
            "n": 0
        },
        "scriptSig": "3f3a4ce81…."
    }
],
```

remember these?

input 1:

input 2:

ID of previous transactions being referenced

Source: Princeton Textbook

CONTENTS OF A TRANSACTION – INPUTS

```
"hash": "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
    "ver": 1,
    "vin_sz": 2,
    "vout_sz": 1,
  "in": [
     {
          "prev_out": {
input 1:        "hash": "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
                "n": 0
          },
                "scriptSig": "30440…"
     },
     {
          "prev_out": {
input 2:        "hash": "7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e",
                "n": 0
          },
          "scriptSig": "3f3a4ce81…."
     }
  ],
```

remember these?

index of input in previous transaction

ID of previous transactions being referenced

index of input in previous transaction

Source: Princeton Textbook

CONTENTS OF A TRANSACTION – INPUTS

remember these?

```
"in": [
    {
input 1:   "prev_out": {
               "hash": "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
               "n": 0
           },
               "scriptSig": "30440…"
    },
    {
input 2:   "prev_out": {
               "hash": "7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e",
               "n": 0
           },
           "scriptSig": "3f3a4ce81…."
    }
],
```

signature used to redeem previous transaction output

signature used to redeem previous transaction output

Source: Princeton Textbook

CONTENTS OF A TRANSACTION – OUTPUTS

```
{
    “hash”: “5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b”,
    “ver”: 1,
    “vin_sz”: 2,
    “vout_sz”: 1,
    “lock_time”: 0,
    “size”: 404,
    “in”: [
          {
                    “prev_out”: {
                              “hash”: “3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260”,
                              “n”: 0
                    },
                              “scriptSig”: “30440…”
          },
          {
                    “prev_out”: {
                              “hash”: “7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e”,
                              “n”: 0
                    },
                    “scriptSig”: “3f3a4ce81….”
          }
    ],

    “out”: [
          {
                    “value”: 10.12287097”,
                    “scriptPubKey”: “OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG”
          }
    ]
}
```

output amount (how much BTC is being sent)

type of script

output script

Source: Princeton Textbook

SOME NOTES

BITCOIN SCRIPT

Output “addresses” are actually scripts.

“scriptPubKey”: “OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG”

➔ This particular Output Script: “This amount can be redeemed by the **public key** that hashes to address X, plus a **signature** from the owner of that public key”

- Inputs and outputs through scripting allows for future extensibility of Bitcoin.

.

- **Script or “Bitcoin Scripting Language”**: Language built specifically for Bitcoin
  - Stack based
  - Simple, **not turing complete** (no loops)

AUTHOR: MAX FANG
EDITED BY: GLORIA ZHAO

Source: Princeton textbook

QUESTIONS?