Signatures and identities - #03

Blockchain and Distributed Ledger technologies

SAPIENZA
UNIVERSITÀ DI ROMA

Massimo La Morgia
massimo.lamorgia@uniroma1.it

Digital signature

A digital signature is supposed to be the digital analog to a handwritten signature on paper.

We desire two properties from digital signatures:

- only you can make your signature, but anyone who sees it can verify that it's valid.
- The signature have to be tied to a particular document so that the signature cannot be used to indicate your agreement or endorsement of a different document.

Digital signature schem

(s_k, p_k) := generateKeys(keysize)

Generate a keypair (s_k, p_k) where p_k is the public verification key and s_k have to be kept secret and used to sign messages

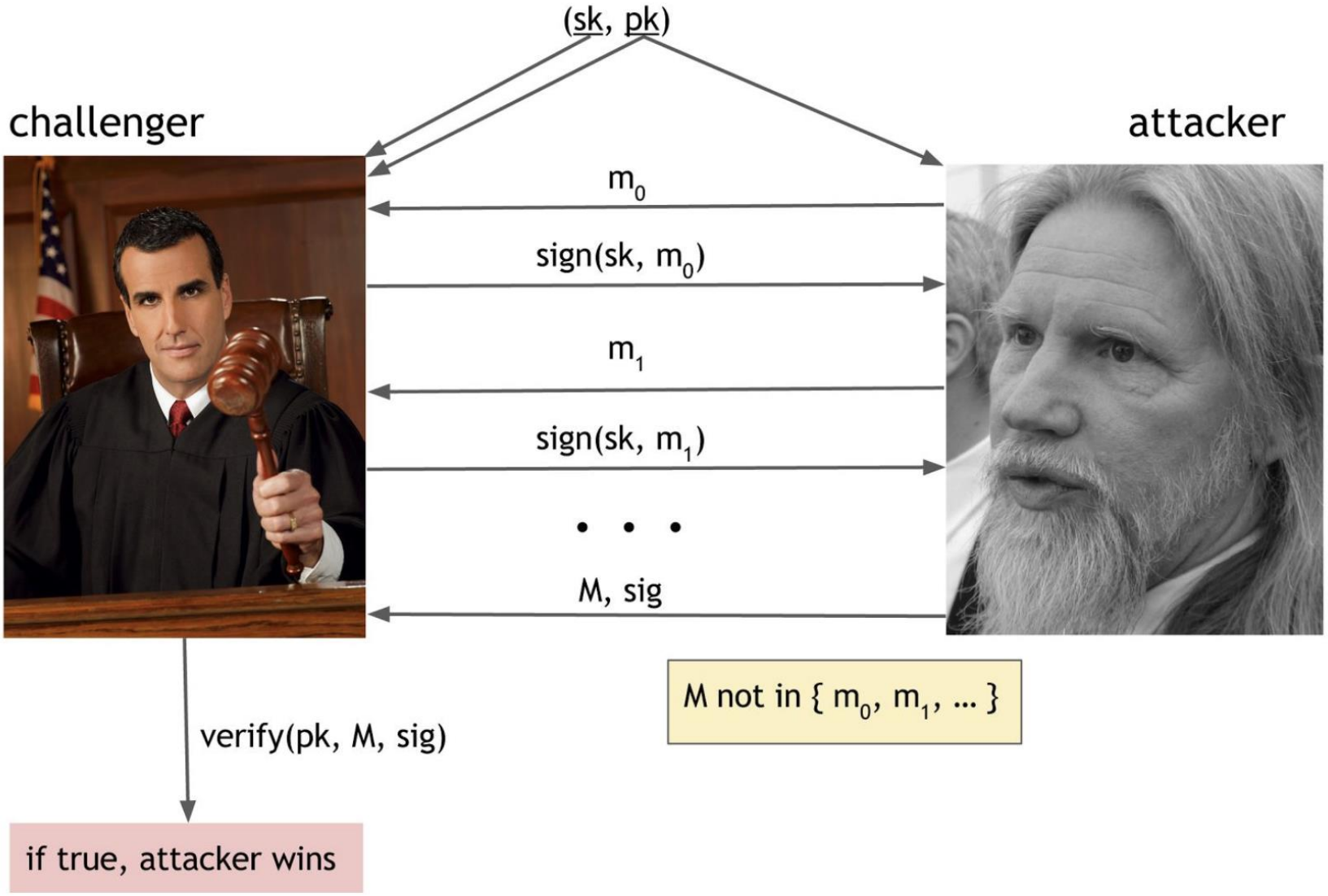
sig := sign(s_k, message)

Sign takes a message and a secret key, as input and outputs a signature for message under s_k

isValid := verify(p_k , message , sig)

It returns a boolean value, isValid , that will be true if sig is a valid signature for message under public key p_k , and false otherwise.

Unforgeability



Identies on bitcoin

Bitcoin uses a particular digital signature scheme that's called the Elliptic Curve Digital Signature Algorithm (ECDSA).

Elliptic curve used "secp256k1"

It produces:

Private key: 256 bits
Public key, uncompressed: 512 bits
Public key, compressed: 257 bits
Message to be signed: 256 bits
Signature: 512 bits

Bitcoin address

There is no need of centralized entities to register your username into the system

It is possible to generate fresh identities at any time and make as many identities as you desire.

Addresses are used when you want to send bitcoins to someone else.

Public key can be very long and impractical.

The shortest version of Bitcoin public keys known to the developers of early Bitcoin were 65 bytes, the equivalent of 130 characters when written in hexadecimal.

How can the public key be shortened?

Hash and the commitment properties is the solution.

Bitcoin uses a concatenation of SHA-256 (256 bit 32 byte) and RIPEMD-160 (160-bit 20Byte) to create address. In this way the final Bitcoin address is of 20 Byte. This operation is also called HASH-160

Then, to make the address more human friendly it is encoded with Base58Check

Identities on bitcoin

Base58 alphabet ” 123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz”

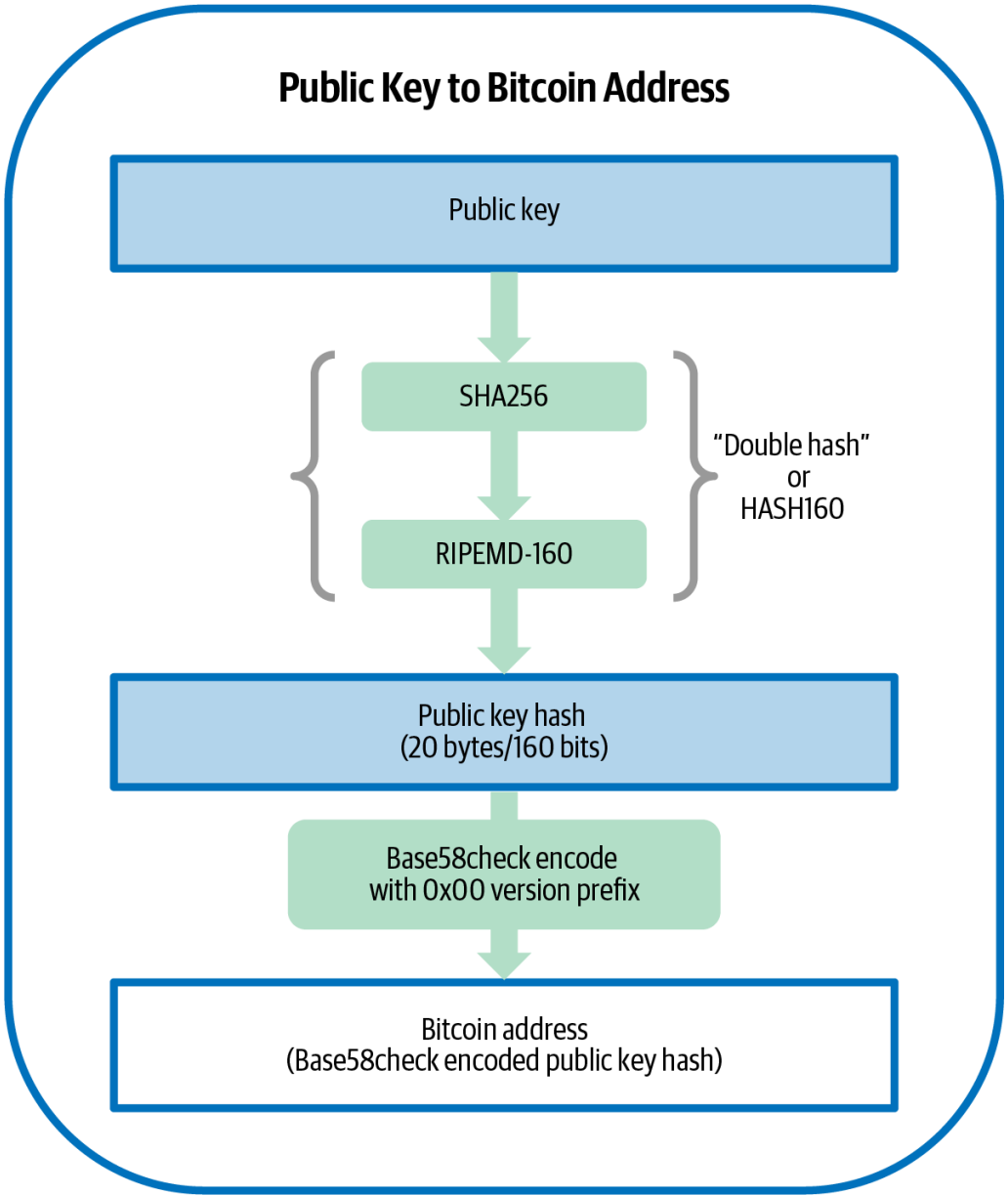
it is a set of lowercase and capital letters and numbers without the four (0, O, I, l)

To add extra security against typos or transcription errors, base58check includes a checksum encoded in the base58 alphabet. The checksum is an additional 4 bytes added to the end of the data that is being encoded.

The checksum, namely Hash-256, is computed by applying 2 times Sha-256 on the data and taking the first 4 bytes.

Type	Version prefix (hex)	Base58 result prefix
Address for pay to public key hash (P2PKH)	0x00	1
Address for pay to script hash (P2SH)	0x05	3
Testnet Address for P2PKH	0x6F	m or n
Testnet Address for P2SH	0xC4	2
Private Key WIF	0x80	5, K, or L
BIP32 Extended Public Key	0x0488B21E	xpub

Identies on bitcoin



Identies on bitcoin

Public key
04f6fc82a2b56b6f8f6ec3370979030f4cb2184f348801f1a27ad9170245e6d44cc3616e5 75f7f92192aa5ef76db7c48a4a3ac392df791c07fab617cce010779af

Sha-256(P_k)
84ea101e376f57e8a22c1a0750b450a6f6b6741419cdd416c4a3436ed6236ca5

RIPEMD-160(Sha-256(P_k))
01c1ec28a2425ba0134b0e07a2d058e576f98bcf

HASH-256(Prefix || RIPEMD-160(Sha-256(P_k)))	
00	01c1ec28a2425ba0134b0e07a2d058e576f98bcf
8525e124a2323332a2502224867663a9f1297c0cc037408fc003fff8628f56f6	

Prefix
00

HASH-160
01c1ec28a2425ba0134b0e07a2d058e576f98bcf

checksum
8525e124

Base58 encoding
1AHzATL6mKFNr5jFQmnonWcc3KzzQdg4B

Practical task -1

Objective

This task is designed to illustrate the computational difficulty of mining a block by having you complete a similar process: generating a custom Bitcoin address (a "Vanity Address") with specific character requirements. Although you won't mine an actual block, this exercise will give you hands-on experience with the complexity involved in customizing cryptographic addresses.

Task Requirements

Generate 5 addresses with the following character pattern criteria:

The second character of the address must match the first character of your first name.

The second and third characters of the address must match the first and second characters of your first name, respectively.

The second, third, and fourth characters of the address must match the first, second, and third characters of your first name, respectively.

Continue this pattern for the remaining addresses.

Example

If your name is "Massimo":

Address 1: 1MmSHj1NUeJV7SxA3t4n3qSSCo7h4H4RMb
Address 2: 1Mao6grNr9tsmynAdkyWUZH97pwbdRp1mx
Address 3: 1MascdKmMbA8Ra1ncRdVpSD8TVEJuqq3VV
Address 4: 1MassMGJnvmLvrrZsrazn3RLUPXdMwTxBE
Address 5: 1MassiGZ8yo9c5Hd56pcuMAHZaz7noqjY

Note: If your first name has fewer than 5 characters, you may concatenate it with your surname.

Tips

To print the public key the Vanitygen tool requires to run it with right flag (-v).
Command example: ./vanitygen++ -v 1Massi

Practical task - 1

Tips
To print the public key the Vanitygen tool requires to run it with right flag (-v).
Command example: ./vanitygen++ -v 1Massi

Suggested Tools

You can use any open-source software or write your own code to generate these addresses. A recommended tool is Vanitygen++ (https://github.com/10gic/vanitygen-plusplus), but feel free to explore other options.

Important Details

Bitcoin addresses exclude the following characters: 0 (zero), O (uppercase o), I (uppercase I), and l (lowercase L).

Ensure all generated Bitcoin addresses are of type P2PKH, starting with "1."

To complete this task, please submit:

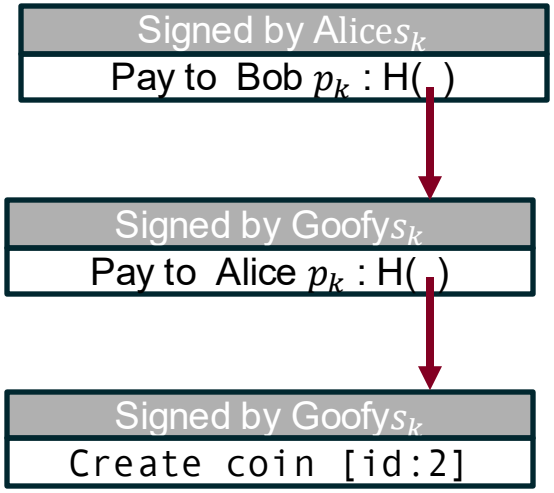
The 5 generated Bitcoin addresses
The associated public keys for each address
The software or method you used to generate the addresses
Challenge
The student who generates a vanity address with the highest number of matching characters (beyond the requirements) will earn an additional point.

GoofyCoin

● Goofy can create new coins by simply signing a statement that he's making a new coin with a unique coin ID.

● Whoever owns a coin can pass it on to someone else by signing a statement that says, "Pass on this coin to X" (where X is specified as a public key)

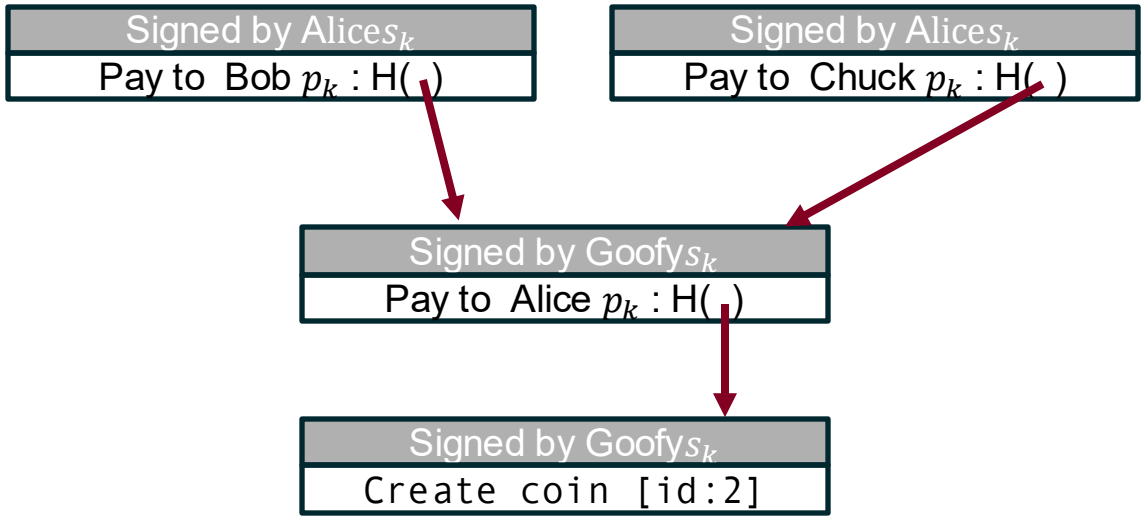
● Anyone can verify the validity of a coin by following the chain of hash pointers back to its creation by Goofy, verifying all of the signatures along the way.



GoofyCoin - Problems

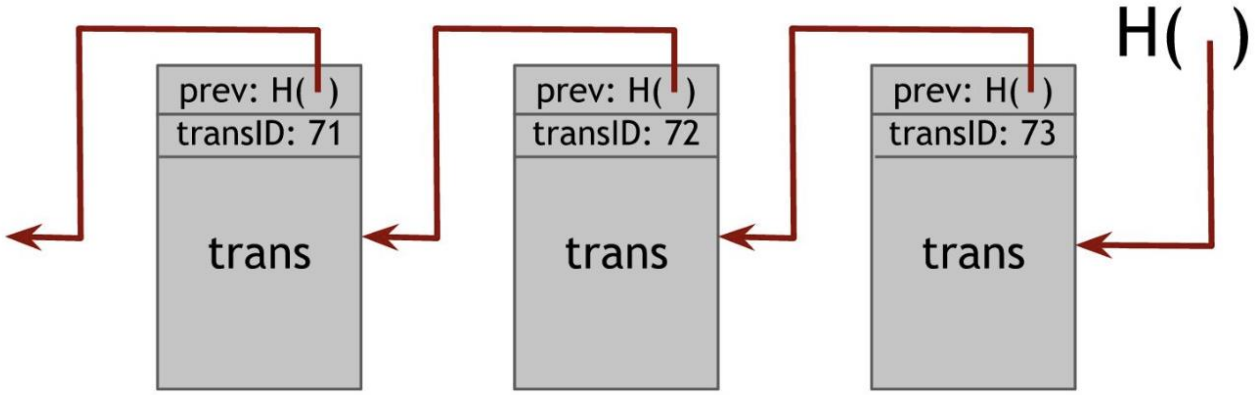
Double Spending

Let's say Alice passed her coin on to Bob by sending her signed statement to Bob but didn't tell anyone else. She could create another signed statement that pays the very same coin to Chuck. To Chuck, it would appear that it is perfectly valid transaction, and now he's the owner of the coin. Bob and Chuck would both have valid-looking claims to be the owner of this coin.



ScroogeCoin

A public append only-leger



Rules:

- The consumed coins are valid, that is, they really were created in previous transactions.
- Only scrooge can create new coins
- The consumed coins were not already consumed in some previous transaction. That is, that this is not a double-spend.
- The total value of the coins that come out of this transaction is equal to the total value of the coins that went in.
- The transaction is validly signed by the owners of all of the consumed coins.
- Scrooge makes sure that he doesn't endorse a transaction that attempts to double-spend an already spent coin.

ScroogeCoin

Rules:

- The consumed coins are valid, that is, they really were created in previous transactions.
- Only scrooge can create new coins
- The consumed coins were not already consumed in some previous transaction. That is, that this is not a double-spend.
- The total value of the coins that come out of this transaction is equal to the total value of the
- coins that went in.
- a transaction only counts if it is in the block chain signed by Scrooge.
- The transaction is validly signed by the owners of all of the consumed coins.

TransID:15 type:Createcoins		
Coins created		
num	value	recipient
0	2	0x0A
1	3	0x0B
2	1	0x0C

TransID: 17 type:PayCoins		
Consumed coinIDs: 15(15), 16(2), 14(2)		
Coins created		
num	value	recipient
0	2	0x0A
1	3	0x0B
2	1	0x0C
signature		

ScroogeCoin – Problems

Can he fake other people's signatures?

Can he create as many coins as he wants?

Can he refuse to publish some transactions?

Can he shut down the blockchain?

ScroogeCoin – Problems