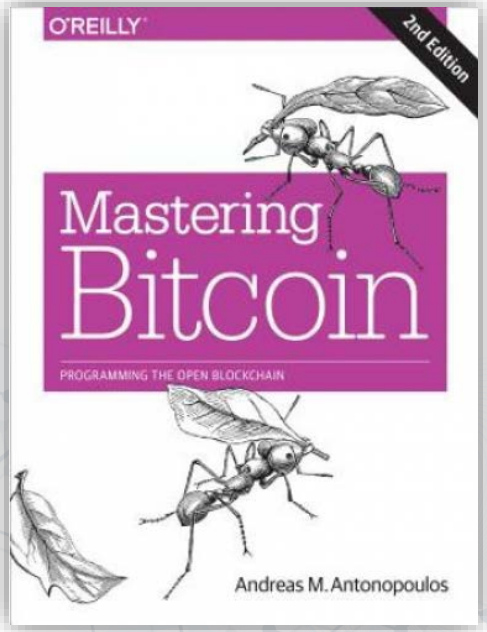
GIST

# Goal of this lecture note

- Mastering Bitcoin
- Elliptic Curve Signatures
- Bitcoin Addresses
- Unspent Transaction Outputs (UTXOs)

# 1 Mastering Bitcoin




Refer to M.B. for materials:

1. Elliptic Curve Signatures
2. Transactions
3. Scripts
4. OP Codes
5. Example Scripts
6. Smart Contracts

『Mastering Bitcoin』, Antonopoulos, Andreas M., O'Reilly Media, 교보문고 제공

GIST

# 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithms
  - Additions and multiplications on some curves.
  - Fifteen curves defined in a NIST standard.
  - But Bitcoin uses the curves def'd in `Secp256k1`.
  - Asymmetric cryptography, pub and priv keys.
  - A public key is used to give a Bitcoin address.
  - A private key is to sign the transfer of right.

[http://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm]
http://www.secg.org/sec2-v2.pdf

GIST

## 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithm
  - Public domain info

1. Use a designated hash function $H(*)$
2. A curve is collection of the roots of $y^2 = x^3 + ax + b$ over a finite field $F(p)$ with prime $p$.
3. $G = (x, y)$, a point on the curve.
4. $n$ the multiplicative order of $G$.

[http://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm]

GIST

## 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithm
  - Key Generation

Out: $k$ (private key), $K$ (public key)

1. Select an integer $k$ in [0, n-1].
2. Compute $K = k\,G$.
3. $K$ and $G$ ~ points on the curve
4. The key-pair is $(k, K)$.

Results: Alice's pair $(k_A, K_A)$ and Bob's pair $(k_B, K_B)$.

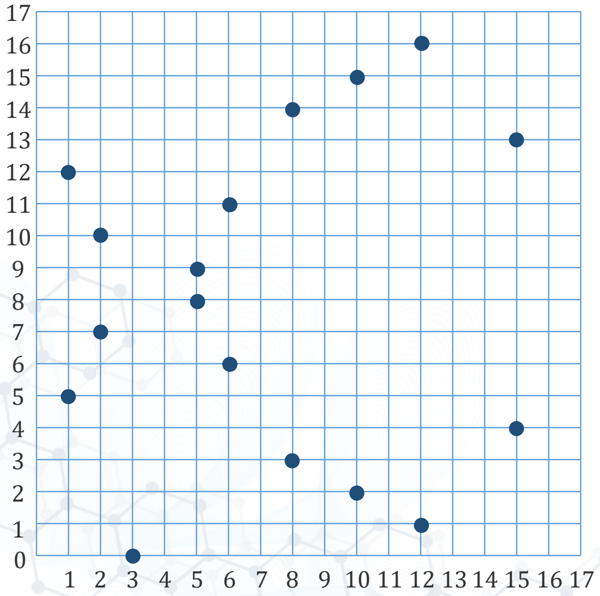
It is an asymmetric cryptography.

[http://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm]

GIST

## 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithm
  - Elliptic Curve




- The points are the roots $(x, y)$ of the curve equation defined by:

$$y^2 = x^3 + 7 \text{ mod } 17$$

Figure 4-3. Elliptic Curve Cryptography: Visualizing an elliptic curve over $F(p)$, with $p = 17$

GIST

## 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithm
  - How many points are on the curve?
    - Observation:
      - For each $x$, there are 0, 1, or 2 possible $y$-point(s).
      - There are total 17 $(x,\ y)$-points.
    - Facts:
      - The set of finite points on the curve forms a *group* which is closed under a binary operation.

GIST

## 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithm
  - Addition of any two points on elliptic curve
    - There are three cases:
      Case 1) Adding two points where $x_1$ neq to $x_2$:

$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$$
$$(\ (x_2 - x_1) \cdot m\ )\ \bmod\ p\ =\ 1$$
$$s = (y_2 - y_1) \cdot m$$
$$x_3 = (\ s^2 - x_1 - x_2\ )\ \bmod\ p$$
$$y_3 = (\ s \cdot (x_1 - x_3) - y_1\ )\ \bmod\ p$$

GIST

## 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithm
  - Addition of any two points on elliptic curve
    - There are three cases:
      Case 2) Adding two points where $x_1 = x_2$ and $y_1 = y_2$

$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$$

$$(\, 2y_1 \cdot m \,) \bmod p = 1$$

$$s = (3x_1^{\,2} + a) \cdot m$$

$$x_3 = (\, s^2 - x_1 - x_2 \,) \bmod p$$

$$y_3 = (\, s \cdot (x_1 - x_3) - y_1 \,) \bmod p$$

GIST

## 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithm
  - Addition of any two points on elliptic curve
    - There are three cases:
      Case 3) Adding two points where $x_1 = x_2$ and $y_1 \neq y_2$

$$(x_1, y_1) + (x_1, y_2) = O$$

The identity element

GIST

# 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithm
  - Table of point additions for $y^2 = x^3 + 7$ mod 17

| + | ∞ | (1,5) | (1,12) | (2,7) | (2,10) | (3,0) | (5,8) | (5,9) | (6,6) | (6,11) | (8,3) | (8,14) | (10,2) | (10,15) | (12,1) | (12,16) | (15,4) | (15,13) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ∞ | ∞ | (1,5) | (1,12) | (2,7) | (2,10) | (3,0) | (5,8) | (5,9) | (6,6) | (6,11) | (8,3) | (8,14) | (10,2) | (10,15) | (12,1) | (12,16) | (15,4) | (15,13) |
| (1,5) | (1,5) | (2,10) | ∞ | (1,12) | (5,9) | (15,13) | (2,7) | (12,1) | (8,14) | (6,6) | (6,11) | (10,15) | (8,3) | (15,4) | (12,16) | (5,8) | (3,0) | (10,2) |
| (1,12) | (1,12) | ∞ | (2,7) | (5,8) | (1,5) | (15,4) | (12,16) | (2,10) | (6,11) | (8,3) | (10,2) | (6,6) | (15,13) | (8,14) | (5,9) | (12,1) | (10,15) | (3,0) |
| (2,7) | (2,7) | (1,12) | (5,8) | (12,16) | ∞ | (10,15) | (12,1) | (1,5) | (8,3) | (10,2) | (15,13) | (6,11) | (3,0) | (6,6) | (2,10) | (5,9) | (8,14) | (15,4) |
| (2,10) | (2,10) | (5,9) | (1,5) | ∞ | (12,1) | (10,2) | (1,12) | (12,16) | (10,15) | (8,14) | (6,6) | (15,4) | (6,11) | (3,0) | (5,8) | (2,7) | (15,13) | (8,3) |
| (3,0) | (3,0) | (15,13) | (15,4) | (10,15) | (10,2) | ∞ | (8,14) | (8,3) | (12,16) | (12,1) | (5,9) | (5,8) | (2,10) | (2,7) | (6,11) | (6,6) | (1,12) | (1,5) |
| (5,8) | (5,8) | (2,7) | (12,16) | (12,1) | (1,12) | (8,14) | (5,9) | ∞ | (10,2) | (15,13) | (3,0) | (8,3) | (15,4) | (6,11) | (1,5) | (2,10) | (6,6) | (10,15) |
| (5,9) | (5,9) | (12,1) | (2,10) | (1,5) | (12,16) | (8,3) | ∞ | (5,8) | (15,4) | (10,15) | (8,14) | (3,0) | (6,6) | (15,13) | (2,7) | (1,12) | (10,2) | (6,11) |
| (6,6) | (6,6) | (8,14) | (6,11) | (8,3) | (10,15) | (12,16) | (10,2) | (15,4) | (1,5) | ∞ | (1,12) | (2,10) | (2,7) | (5,9) | (3,0) | (15,13) | (12,1) | (5,8) |
| (6,11) | (6,11) | (6,6) | (8,3) | (10,2) | (8,14) | (12,1) | (15,13) | (10,15) | ∞ | (1,12) | (2,7) | (1,5) | (5,8) | (2,10) | (15,4) | (3,0) | (5,9) | (12,16) |
| (8,3) | (8,3) | (6,11) | (10,2) | (15,13) | (6,6) | (5,9) | (3,0) | (8,14) | (1,12) | (2,7) | (5,8) | ∞ | (12,16) | (1,5) | (10,15) | (15,4) | (2,10) | (12,1) |
| (8,14) | (8,14) | (10,15) | (6,6) | (6,11) | (15,4) | (5,8) | (8,3) | (3,0) | (2,10) | (1,5) | ∞ | (5,9) | (1,12) | (12,1) | (15,13) | (10,2) | (12,16) | (2,7) |
| (10,2) | (10,2) | (8,3) | (15,13) | (3,0) | (6,11) | (2,10) | (15,4) | (6,6) | (2,7) | (5,8) | (12,16) | (1,12) | (12,1) | ∞ | (8,14) | (10,15) | (1,5) | (5,9) |
| (10,15) | (10,15) | (15,4) | (8,14) | (6,6) | (3,0) | (2,7) | (6,11) | (15,13) | (5,9) | (2,10) | (1,5) | (12,1) | ∞ | (12,16) | (10,2) | (8,3) | (5,8) | (1,12) |
| (12,1) | (12,1) | (12,16) | (5,9) | (2,10) | (5,8) | (6,11) | (1,5) | (2,7) | (3,0) | (15,4) | (10,15) | (15,13) | (8,14) | (10,2) | (1,12) | ∞ | (8,3) | (6,6) |
| (12,16) | (12,16) | (5,8) | (12,1) | (5,9) | (2,7) | (6,6) | (2,10) | (1,12) | (15,13) | (3,0) | (15,4) | (10,2) | (10,15) | (8,3) | ∞ | (1,5) | (6,11) | (8,14) |
| (15,4) | (15,4) | (3,0) | (10,15) | (8,14) | (15,13) | (1,12) | (6,6) | (10,2) | (12,1) | (5,9) | (2,10) | (12,16) | (1,5) | (5,8) | (8,3) | (6,11) | (2,7) | ∞ |
| (15,13) | (15,13) | (10,2) | (3,0) | (15,4) | (8,3) | (1,5) | (10,15) | (6,11) | (5,8) | (12,16) | (12,1) | (2,7) | (5,9) | (1,12) | (6,6) | (8,14) | ∞ | (2,10) |

[https://graui.de/code/elliptic2/]

GIST

## 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithm
  - Example to find a point on a curve
    - Let $p$ = 17.
    - Let the curve be $y^2 = x^3 + 7 \text{ mod } 17$.
    - Find a point on the curve

      Let $x = 3$. Then $y = ?$

      $y^2 = 27 + 7 = 34 = 0$

      $y^2 = 0$

      $y = 0$
    - Thus, (3, 0) is a point on the curve.

GIST

## 2 Elliptic Curve Signatures

Anaconda Powershell Prompt

```
>>>
>>> p = 17
>>> x = 3
>>> y_square = (x**3 + 7)%p
>>> y_square
0
>>>
```

GIST

## 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithm
  - Example to find a point on a curve
    - Let us continue to find another point.
    - This time, let us start with an $y$ element.
    - Let $y = 12$ and find $x$.

$$y^2 = 12^2$$
$$= 144 - \text{floor}(144/17)\text{x}17$$
$$= 8$$
$$x^3 + 7 = 8$$
$$x^3 = 1$$
$$x = 1$$

- Thus, (1, 12) is a point on the curve.

## 2 Elliptic Curve Signatures

Anaconda Powershell Prompt

```
>>> p = 17
>>> y = 12
>>> y_square = y**2
>>> y_square
144
>>> y_square = y_square%p
>>> y_square
8
>>> x_3rd_power = (y_square - 7)%p
>>> x_3rd_power
1
>>>
```

GIST

## 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithm
  - Let us add two points.

Given two points $(x_1, y_1) = (3, 0)$ and $(x_2, y_2) = (1, 12)$.

Find $(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$.

Note this is Case 1.

$$( (1-3) \cdot m ) \% 17 = 1$$

$$m = 8$$

$$s = (y_2 - y_1) \cdot m = (12-0) \cdot 8 = 96 = 11$$

$$x_3 = s^2 - x_1 - x_2 = s^2 - 3 - 1 = 121 - 4 = 117\%17 = 15$$

$$y_3 = s \cdot (x_1 - x_3) - y_1 = 11 \cdot (3-15) - 0 = -132 = -132\%17 = 4$$

$$(x_3, y_3) = (15, 4)$$

# 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithm
  - Let us add two points.

Given two points $(x_1, y_1) = (6,11)$ and $(x_2, y_2) = (6,11)$.

Find $(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$.

Note this is Case 2.

$$( 2 \cdot 11 \cdot m ) \,\%\, 17 = 1$$

$$m = 7$$

$$s = (3x_1^{\ 2} + a) \cdot m = (3 \cdot 6^2 + 0) \cdot 7 = 756 = 8$$

$$x_3 = s^2 - x_1 - x_2 = 8^2 - 6 - 6 = 52\%17 = 1$$

$$y_3 = s \cdot (x_1 - x_3) - y_1 = 8 \cdot (6 - 1) - 11 = 29\%17 = 12$$

$$(x_3, y_3) = (1,12)$$

GIST

## 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithm
  - Let us add two points.

Given two points $(x_1, y_1) = (10, 2)$ and $(x_2, y_2) = (10, 15)$.

Find $(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$.

Note this is Case 3.

$$(10, 2) + (10, 15) = O$$

The identity element

GIST

## 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithm
  - A scalar multiplication example
    - Take any point $P = (x, y)$ on the curve and multiply it by a scalar $k$.
    - The resulting point can be obtained by adding $P$ $k$ times, i.e.,

$$kP = P + P + \ldots + P$$

# 2 Elliptic Curve Signatures

- We may use Python for computations.
  - A point $P(x, y)$ is point on the the `secp256k1` curve.
  - You can check our results using Python.

Anaconda Powershell Prompt

```
(base) PS C:₩Users₩Heung-No Lee> python
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> p = 17
>>> x = 1
>>> y = 5
>>> (x**3+7-y**2)%p
0
>>>
```

GIST

## 2 Elliptic Curve Signatures

- We may use Python libraries at github.
  - One example is https://github.com/vbuterin/pybitcointools.
  - It offers `pybitcointools` library which allows us to generate and display keys and addresses.
  - The other one is at https://github.com/warner/python-ecdsa which offers ECDSA implementation in Python.

GIST

## 2 Elliptic Curve Signatures

- From private key $k$, obtain public key by $K = k*G$.
  - A 256 bit string is shown as 64 hexadecimal string.

$k=$ 1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD

$G = (x, y) =$ (55066263022277343669578718895168534326250603453777594175500187360389116729240,
32670510020758816978083085130507043184471273380659243275938904335757337482424)

  - Multiply the private key $k$ with the generator point $G$ to obtain the public key $K$.

$K=$ 1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD $*G$

$K=$ (x, y)

where,

$x=$ F028892BAD...DC341A

$y=$ 07CF33DA18...505BDB

GIST

## 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithm
  - We now know how to generate keys.
  - Next is how to sign and validate it.

http://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm

## 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithm
  - SignGenerate

**In** $m$ the message, Alice's private key $k_A$

**Out** Alice' signature $(r, s)$

1. Calculate the message hash $e=H(m)$
2. Let $z$ be the $L_n$ leftmost bits of $e$ where $L_n$ is the bit length of the group order $n$
3. Select an integer d from $[1, n-1]$
4. Calculate the curve point $(x_1, y_1)=dG$
5. Calculate $r=x_1 \bmod n$, If $r=0$, go to step 3
6. Calculate $s= k_A^{-1}(z+rk_A) \bmod n$, If $s=0$, go to step 3
7. The signature is the pair $(r, s)$

GIST

# 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithm
  - IsSignatureValid

**In** $m$ a message, Alice's signature $(r, s)$, and $K_A$
**Out** Valid or invalid

1. Verify if $K_A$ is a valid curve point as follows:
   1. Check to see if $K_A$ is not equal to the identity element $O$
   2. Check to see if $K_A$ lies on the curve
   3. Check that $n \times K_A = O$
2. Verify that $r$ and $s$ are integers in $[1, n-1]$
   If not, the signature is invalid
3. Calculate the message hash $e = H(m)$

[출처: http://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm]

GIST

# 2 Elliptic Curve Signatures

- Elliptic Curve Digital Signature Algorithm
  - IsSignatureValid

**In** $m$ the message, Alice's signature $(r, s)$, and $K_A$
**Out** Valid or invalid

4. Let $z$ be the $L_n$ leftmost bits of $e$ where $L_n$ is the bit length of the group order $n$
5. Calculate $w = s^{-1} \bmod n$
6. Calculate $u_1$=z $w$ mod $n$ and $u_2$=$r$ *$w$ mod $n$
7. Calculate the curve point $(x_1, y_1) = u_1 * G + u_2 * Q_A$
   If $x_1, y_1 = 0$, then the signature is invalid
8. The signature is valid if $r = x_1 \bmod n$, invalid otherwise

[출처: http://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm]

# 3 Bitcoin Addresses

- An example Bitcoin Address is *1thMjrt546nngXqyPEz532S8fLwbozud8.*
  - BTCs belong to a Bitcoin address.
  - We aim to know how they are generated.
  - An address is generated from a public key.
  - It goes through several mappings such as `SHA256`, `RIPEMD160`, and `Base58Check`.

GIST

# 3 Bitcoin Addresses

- Making a Bitcoin address from a public key
  - Private key $k$ (32 bytes)
  - Public key $K = G * k$
    - Uncompressed one is 65 bytes $(0x04 + x + y)$.
    - Compressed one is 33 bytes
      (0x02+ $x$, use 02 for even $y$; 0x03+$x$ *for odd* $y$).
  - Public Key Hash = RIPEMD160(SHA256($K$))
    - 160 bit (20 byte)
  - Base58Str
    = Base58Check(PKH + 4Byte_checksum)

Ex 1PMycacnJaSqwwJqjawXBErnLsZ7RkXUAs

GIST

## 3 Bitcoin Addresses

- What is Base58Check and why?
  - Base58Check is mapping a PKH into a more readable format.
  - Base58 is similar to Base64 but with 6 characters removed.
  - Base64 uses A-Z, a-z, 0-9, + and /.
  - Removed are +, /, 0, O, I and l.
  - These symbols are prone to confusion.
  - A Bitcoin address is of between 27 and 34 characters long!

GIST

# 3 Bitcoin Addresses

- Base58 Value-to-Character Mapping Table

| Value | Character | Value | Character | Value | Character | Value | Character |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 2 | 3 | 3 | 4 |
| 4 | 5 | 5 | 6 | 6 | 7 | 7 | 8 |
| 8 | 9 | 9 | A | 10 | B | 11 | C |
| 12 | D | 13 | E | 14 | F | 15 | G |
| 16 | H | 17 | J | 18 | K | 19 | L |
| 20 | M | 21 | N | 22 | P | 23 | Q |
| 24 | R | 25 | S | 26 | T | 27 | U |
| 28 | V | 29 | W | 30 | X | 31 | Y |
| 32 | Z | 33 | a | 34 | b | 35 | c |
| 36 | d | 37 | e | 38 | f | 39 | g |
| 40 | h | 41 | i | 42 | j | 43 | k |
| 44 | m | 45 | n | 46 | o | 47 | p |
| 48 | q | 49 | r | 50 | s | 51 | t |
| 52 | u | 53 | v | 54 | w | 55 | x |
| 56 | y | 57 | z | | | | |

GIST

## 3 Bitcoin Addresses

- Example of Base58Check Mapping

$$
\begin{aligned}
12437_{10} &= 3\text{x}58^2 + 40\text{x}58^1 + 25 \\
&= 3\ 40\ 25_{58} \\
&= 4hS_{58}
\end{aligned}
$$

GIST

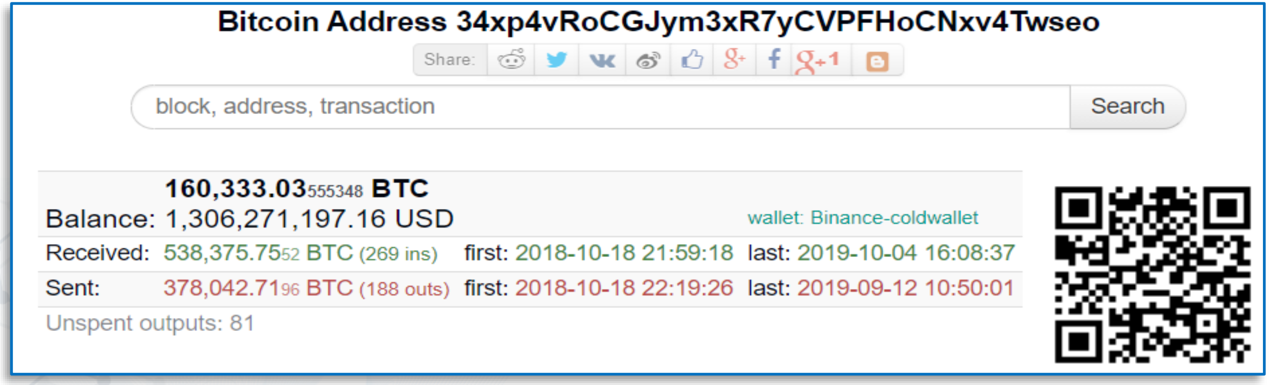
## 3 Bitcoin Addresses

- A version prefix is appended to Base58Str
  - Table 4-1. Version Prefixes

| Type | Version prefix (hex) | Base-58 prefix |
| --- | --- | --- |
| Bitcoin Address | 0×00 | 1 |
| Pay-to-Script-Hash Address | 0×05 | 3 |
| Bitcoin Testnet Address | 0×6F | m or n |
| Private Key WIF | 0×80 | 5, K or L |
| BIP38 Encrypted Private Key | 0×0142 | 6P |
| BIP32 Extended Public Key | 0×0488B21E | xpub |

# 3 Bitcoin Addresses

- The richest Bitcoin address on 2019/10/14 is
  **34xp4vRoCGJym3xR7yCVPFHoCNxv4Twseo**
- It holds 160,333.03 BTCs.




https://bitinfocharts.com/bitcoin/address/34xp4vRoCGJym3xR7yCVPFHoCNxv4Twseo

GIST

# 4 Unspent Transaction Outputs (UTXOs)

- UTXO is an unspent transaction output.
- Given an address, one can obtain all the UTXOs belonging to that address by going through the ledger.
- We are interested in

  *Creating, signing and submitting Transactions based on UTXOs.*

GIST

# 4 Unspent Transaction Outputs (UTXOs)

- How to obtain UTXOs?
  - When you download/install Bitcoin core, you run the Bitcoin client.
  - Mastering Bitcoin has a detailed procedure for installation (see Ch.3)
  - One can use the Bitcoin client to find all the UTXOs.
  - The command `listunspent` can list out all UTXOs which belong to address.
  - Once UTXOs are figured out, they can be spent.

『Mastering Bitcoin』, Antonopoulos, Andreas M., O'Reilly Media

GIST

# 4 Unspent Transaction Outputs (UTXOs)

- UTXOs
  - First, use the `listunspent` command to show all the unspent confirmed outputs to each address in our wallet.

```
$ bitcoin-cli listunspent

[
    {
        "txid" : "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
        "vout" : 0,
        "address" : "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
        "account" : "",
        "scriptPubKey" : "76a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac",
        "amount" : 0.05000000,
        "confirmations" : 7
    }
]
```

GIST

## 4 Unspent Transaction Outputs (UTXOs)

- UTXOs
  - When you want to spend an `UTXO`, you make a transaction in which an `UTXO` is used as an input by referring to the previous `txid` and `vout` index.
  - You need to create a new transaction that will spend the 0th `vout` of the `txid` `9ca8f0...` as its input and assign it to a new output address.

GIST

# 4 Unspent Transaction Outputs (UTXOs)

- Closer look at a UTXO with `txid 9ca8…, vout0`
  - Use the `gettxout` command.
  - Transaction outputs are always referenced by `txid` and `vout`, and they are the parameters we pass to `gettxout`.

GIST

# 4 Unspent Transaction Outputs (UTXOs)

- Closer look at `txid 9ca8... vout0`

```
$ bitcoin-cli gettxout 9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3 0
```


```
{
    "bestblock" : "0000000000000001405ce69bd4ceebcdfdb537749cebe89d371eb37e13899fd9",
    "confirmations" : 7,
    "value" : 0.05000000,
    "scriptPubKey" : {
        "asm" : "OP_DUP OP_HASH160 07bdb518fa2e6089fd810235cf1100c9c13d1fd2\
         OP_EQUALVERIFY OP_CHECKSIG",
        "hex" : "76a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac",
        "reqSigs" : 1,
        "type" : "pubkeyhash",
        "addresses" : [
            "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL"
        ]
    },
    "version" : 1,
    "coinbase" : false
}
```

GIST

## 4 Unspent Transaction Outputs (UTXOs)

- Closer look at `txid 9ca8…, vout0`
  - What we see above is the output that has 0.05 BTC to our address `1hvz....`
  - To spend this output we shall create a new transaction.
  - For this, we need to get an address to which we will send the money:

# 4 Unspent Transaction Outputs (UTXOs)

- Making a new transaction
  - There is a Bitcoin client command `createrawtransaction`.
  - It can be used to generate a raw transaction.
  - Suppose you want to make a new transaction
    - A payment of 0.030 BTC to a recipient with address 1LTz9…1cP.
    - A change of 0.015 BTC is given back to an address of yours, 1Bts8…2Ps.
    - The rest, 0.050 – 0.030 – 0.015 = 0.005 BTC, is given to miners as TX fee.

GIST

# 4 Unspent Transaction Outputs (UTXOs)

**TXID 7957a35…f18**

| In0: | | Vout0: | |
|---|---|---|---|
| TXID 9ca8…ae3 | → | ScriptPK1 | 0.030 BTC |
| vout 0 | | Vout1: | |
| Sign | | ScriptPK2 | 0.015 BTC |
| 0.050 BTC | | | |

GIST

# 4 Unspent Transaction Outputs (UTXOs)

- Each TX is locked. To unlock, you need the private key.
  - 시간 1: A's Signature (Key)→ B (Locked to B) 2BTC.
  - 시간 2: B's Signature (Key)→ C (Locked to C) 1BTC.
  - 시간 3: C's Signature (Key)→ D (Locked to D) 0.5BTC.

GIST

# 4 Unspent Transaction Outputs (UTXOs)

- Making a new transaction
  - Inputs given to `createrawtransaction` include:
    - UTXO's TXID vout 0
    - 1LTz9⋯1cP 0.030 BTC
    - 1Bts8⋯2Ps 0.015 BTC
  - Then, a chuck of script code is generated.