Module 4: Concepts of Bitcoin

Text Book : Mastering Bitcoin - Andreas M. Antonopoulos (Chapters 4, 5 & 6)

Priya R L, Lifna C S

Department of Computer Engineering, VESIT, Mumbai

Agenda

Keys, Addresses

- **Public Key Cryptography and Cryptocurrency**
- **Private and Public Keys**
- Bitcoin Addresses -
  - Base58
  - Base58Check Encoding

Wallets

- Nondeterministic (Random) Wallets
- Deterministic (Seeded) Wallets
- HD Wallets (BIP-32/BIP-44)
- Wallet Best Practices
- Using a Bitcoin Wallet

Transactions

- Transaction Outputs and Inputs
- Transaction Fees
- Transaction Scripts and Script Language
- Turing Incompleteness
- Stateless Verification
- Script Construction (Lock + Unlock)
- Pay-to-Public-Key-Hash (P2PKH)
- Bitcoin Addresses, Balances, and Other Abstractions

Key & Addresses

Bitcoin uses elliptic curve multiplication as the basis for its cryptography.

- **public key cryptography** to create a key pair that controls access to bitcoin.
- The **public key** is used to **receive funds**, and
- the **private key** is used to **sign trans- actions to spend the funds**

A **bitcoin wallet** contains a collection of key pairs, each consisting of a private key and a public key.

In most wallet implementations, the private and public keys are stored together as a key pair for convenience. However, the **public key can be calculated from the private key, so storing only the private key is also possible.**

Key & Addresses

The private key (k) is a number, usually picked at random.

- From the private key, we use **elliptic curve multiplication**, a one-way cryptographic function, to generate a public key (K).
- From the public key (K), we use a **one-way cryptographic hash function** to generate a bitcoin address (A)

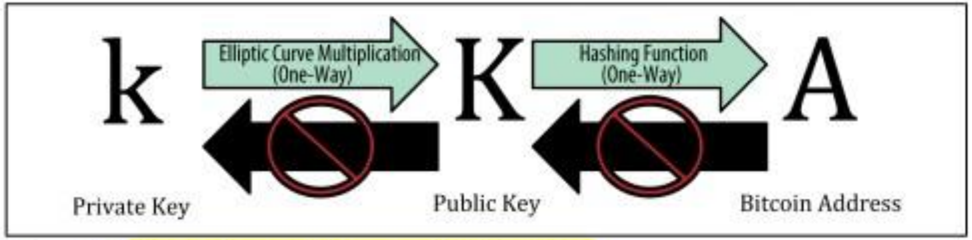


Figure 4-1. Private key, public key, and bitcoin address

Key & Addresses

A bitcoin address is a ***string of digits and characters*** that can be shared with anyone who wants to send you money.

- Addresses **produced from public keys** consist of a string of numbers and letters, beginning with the digit "1."
- appears most commonly in a transaction as the "**recipient**" of the funds.
- The bitcoin address is derived from the public key through the **use of one-way cryptographic hashing.**
- **Secure Hash Algorithm (SHA) specifically SHA256**
- **RACE Integrity Primitives Evaluation Message Digest (RIPEMD), specifically RIPEMD160.**

Key & Addresses

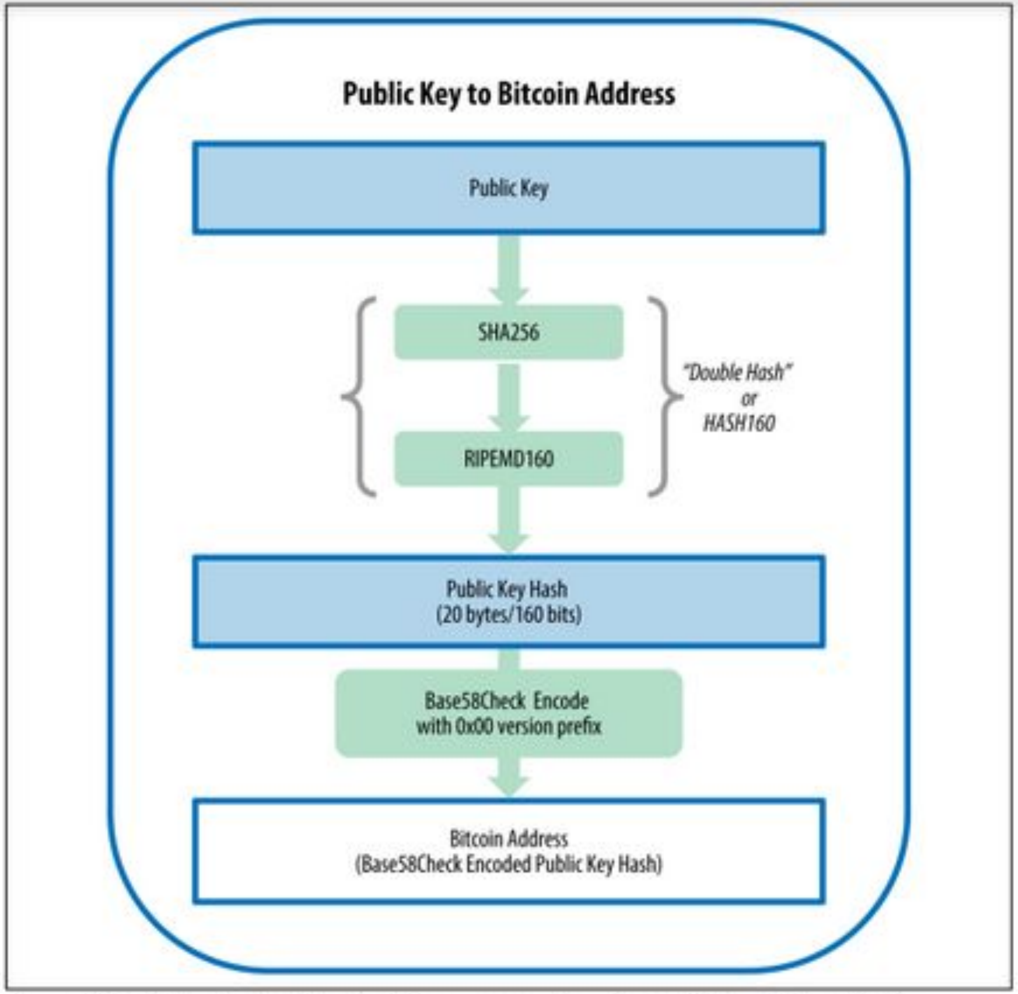


Figure 4-5. Public key to bitcoin address: conversion of a public key into a bitcoin address

Key & Addresses

- Bitcoin addresses are almost always encoded **as "Base58Check"**
    - **uses 58 characters (a Base58 number system)**
    - **checksum to help human readability, avoid ambiguity, and protect against errors in address transcription and entry.**
- Base58Check is also used in many other ways in bitcoin, whenever there is a **need for a user to read and correctly transcribe a number,** such as a bitcoin address, a private key, an encrypted key, or a script hash.

Key & Addresses

- to represent long numbers in a compact way, using fewer symbols, many computer systems use mixed-alphanumeric representations with a base (or radix) higher than 10.

Eg:

- **Decimal system** uses the 10 , numerals 0 through 9
- **Hexadecimal system** uses 16, with the letters A through F as the six additional symbols.
- **Base64** → uses 26 lowercase letters, 26 capital letters, 10 numerals, and 2 more characters such as “+” and “/”
  - **to transmit binary data over text-based media** such as email.
  - to **add binary attachments to email.**

Key & Addresses

Base58 is a t**ext-based binary-encoding format** developed for use in bitcoin and used in many other cryptocurrencies.

- It offers a **balance between compact representation, readability, and error detection and prevention**.
- a subset of Base64, using upper- and lower-case letters and numbers, but omitting some characters that are frequently mistaken for one another and can appear identical when displayed in certain fonts.
- Base58 is a **set of lowercase and capital letters and numbers without the four** (0, O, l, I) just mentioned.

Key & Addresses

Base58Check

- Base58 encoding format, which has a **built-in error-checking code**.
- **To add extra security against typos or transcription errors**
- Its is a **four bytes added to the end of the data** that is being encoded.
- It is **derived from the hash of the encoded data**
- used to detect and prevent transcription and typing errors.
- When presented with **Base58Check code**, the decoding software will calculate the checksum of the data and compare it to the checksum included in the code.
    - If NO match, Base58Check data is invalid.
- This prevents a mistyped bitcoin address from being accepted by the wallet software as a valid destination, an error that would otherwise result in loss of funds.

Key & Addresses

To convert data (a number) into a Base58Check format,

1 . add a prefix to the data, called the "**version byte**," which serves to easily identify the type of data that is encoded.

- Eg: for bitcoin address the prefix is zero (0x00 in hex)
- Encoding a private key is 128 (0x80 in hex).

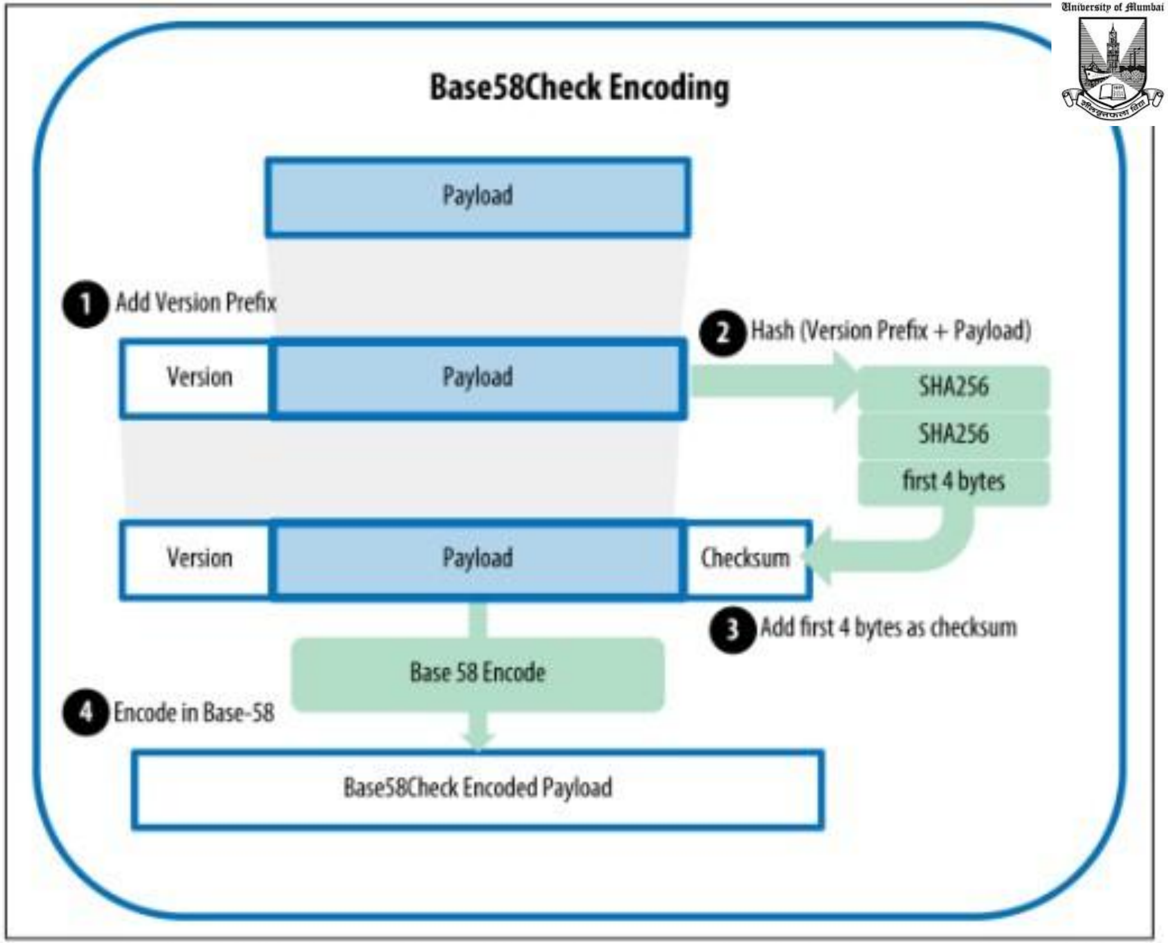


Figure 4-6. Base58Check encoding: a Base58, versioned, and checksummed format for unambiguously encoding bitcoin data

Wallets

- refers to the data structure used to store and manage a user's keys.
- are two primary types of wallets
    - **nondeterministic wallet,**
        - where each key is independently generated from a random number.
        - The keys are not related to each other.
        - This type of wallet is also known as a **JBOK "Just a Bunch Of Keys."**
    - **deterministic wallet**,
        - where all the keys are derived from a single master key, known as the seed.
        - All the keys in this type of wallet are related to each other and can be generated again if one has the original seed.
        - The most commonly used derivation method uses a tree-like structure and is known as a **hierarchical deterministic or HD wallet.**

V.E.S. Since 1962

University of Mumbai

Wallets - **Nondeterministic (Random) Wallets**

- The disadvantage of random keys is that if you generate many of them you must keep copies of all of them, meaning that the wallet must be backed up frequently.
- transaction. Address reuse reduces privacy by associating multiple transactions and addresses with each other.

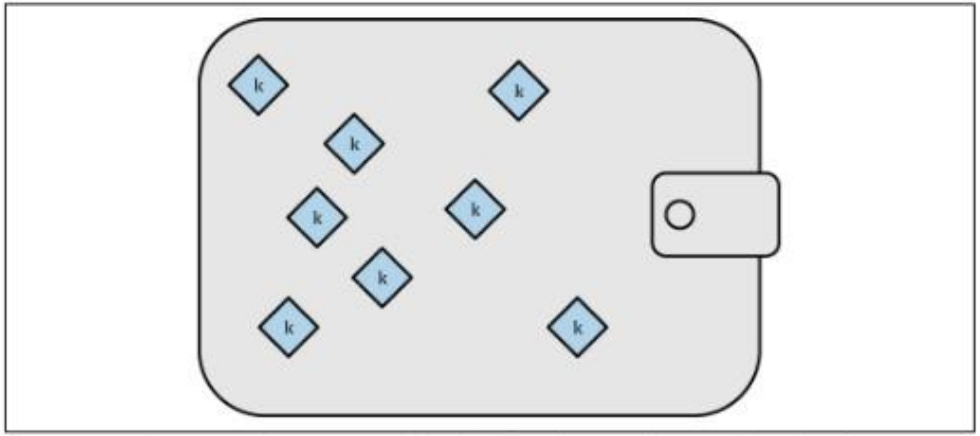
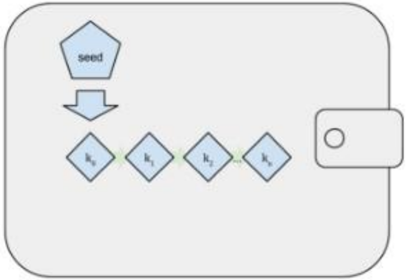


Figure 5-1. Type-0 nondeterministic (random) wallet: a collection of randomly generated keys

Wallets - **Deterministic (Seeded) Wallets**

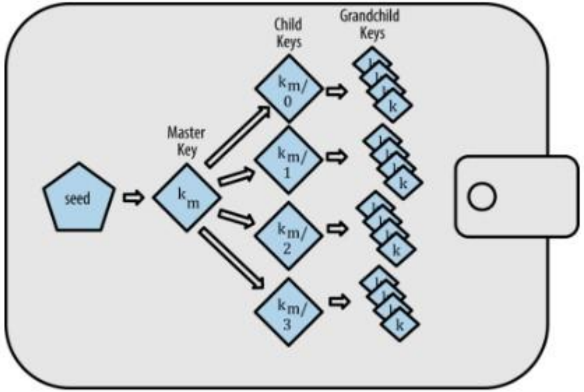
- wallets that contain **private keys that are all derived from a common seed**, through the use of a one-way hash function.
- The seed is a **randomly generated number** that is combined with other data, such as an index number or “chain code” to derive the private keys.
- the seed is sufficient
    - to **recover all the derived keys, and therefore a single backup at creation time is sufficient.**
    - wallet export or import, **allowing for easy migration of all the user’s keys between different wallet implementations.**



Wallets - **Deterministic (Seeded) Wallets**

HD Wallets (BIP-32/BIP-44)

- The most advanced form of deterministic wallets is the HD wallet defined by the BIP-32 standard.
- HD wallets contain keys derived in a tree structure, such that a parent key can derive a sequence of children keys, each of which can derive a sequence of grandchildren keys, and so on, to an infinite depth.



Wallets - **Deterministic (Seeded) Wallets**

- **Bitcoin Improvement Proposal 39 wordlist** (or 'BIP39' for short) is **a standardized set of words for the recovery and backup of a bitcoin or cryptocurrency wallet**.
- Each word in the list is unique within the first four letters of each word, meaning no two words on the list share the same first four letters.

Github : [BIP 39]

Wallets - **Deterministic (Seeded) Wallets**

HD Wallets two major advantages over random (nondeterministic) keys.

1. Tree structure can be used to **express additional organizational meaning**,
   Eg: Branches of keys can also be used in corporate settings, allocating different branches to departments, subsidiaries, specific functions, or accounting categories.
2. **Users can create a sequence of public keys without having access to the corresponding private keys.**
   a. Thus **HD wallets to be used on an insecure server**
   b. **The public keys do not need to be preloaded or derived in advance,**

Wallets

Wallet Best Practices

- certain common industry standards have emerged that make bitcoin wallets broadly interoperable, easy to use, secure, and flexible.
- These common standards are:
  - Mnemonic code words, based on BIP-39
  - HD wallets, based on BIP-32
  - Multipurpose HD wallet structure, based on BIP-43
  - Multicurrency and multiaccount wallets, based on BIP-44
- Eg : software wallets : Breadwallet, Copay, Multibit HD, and Mycelium.
- Eg : Hardware : Keepkey, Ledger, and Trezor.

Wallets

Using a Bitcoin Wallet

Figure 5-4. A Trezor device: a bitcoin HD wallet in hardware

Eg:

1. the device generated a mnemonic and seed from a built-in hardware random number generator. During this initialization phase, the wallet displayed a numbered sequence of words, one by one, on the screen
2. By writing down this mnemonic, we can created a backup that can be used for recovery in the case of loss or damage to the Trezor device.
3. This mnemonic can be used for recovery in a new Trezor or in any one of the many compatible software or hardware wallets.

Note : sequence of words is important,

Transaction Outputs and Inputs

- The **fundamental building block** of a bitcoin transaction is a **transaction output.**
- Transaction outputs are **indivisible chunks of bitcoin currency,** recorded on the blockchain, and recognized as valid by the entire network.
- Bitcoin full nodes track all available and spendable outputs, known as **unspent transaction outputs, or UTXO.**
- The collection of all UTXO is known as the **UTXO set** and currently numbers in the millions of UTXO.
- The UTXO set grows as new UTXO is created and shrinks when UTXO is consumed.
- **Every transaction represents a change (state transition) in the UTXO set**.

Transaction Outputs and Inputs

Every bitcoin transaction creates outputs, which are recorded on the bitcoin ledger.

Transaction outputs consist of two parts:

1. An amount of bitcoin, denominated in satoshis, the smallest bitcoin unit
2. A cryptographic puzzle that determines the conditions required to spend the output

The cryptographic puzzle is also known as a **locking script, a witness script, or a scriptPubKey.**

Transaction Outputs and Inputs

Transaction inputs identify (by reference) which UTXO will be consumed and **provide proof of ownership through an unlocking script.**

The input contains four elements:

1. A transaction ID, referencing the transaction that contains the UTXO being spent
2. An output index (vout), identifying which UTXO from that transaction is referenced (first one is zero)
3. A scriptSig, which satisfies the conditions placed on the UTXO, unlocking it for spending
4. A sequence number

Transaction Fees

- Most transactions include transaction fees, which **compensate the bitcoin miners for securing the network**.
- Fees also serve as a security mechanism themselves, by making it **economically infeasible for attackers to flood the network with transactions.**
- **incentive to include (mine) a transaction into the next block** and also as a **disincentive against abuse of the system by imposing a small coston every transaction.**
- Transaction fees are **collected by the miner who mines the block that records the transaction on the blockchain.**

Transaction Scripts and Script Language

- bitcoin transaction script language, called **Script**, is a Forth-like reverse-polish notation stack-based execution language
- A s**tack-based scripting language** embedded in Bitcoin transactions
- locking script placed on a UTXO and the unlocking script are written in this scripting language.
- Script is a **very simple language**
    - **limited in scope** and
    - **executable on a range of hardware**
    - **requires minimal processing**
    - **use in validating programmable money, this is a deliberate security feature.**

Transaction Scripts and Script Language

- Bitcoin transaction validation is **not based on a static pattern,** but instead is **achieved through the execution of a scripting language.**
- This language **allows for a nearly infinite variety of conditions to be expressed.**
- This is how bitcoin gets the power of “**programmable money.**”

Transaction Scripts and Script Language

<u>**Turing Incompleteness**</u>

- bitcoin transaction script language **contains many operators**
- there **are no loops or complex flow control capabilities other than conditional flow control.**
- This ensures that the language is <u>**not Turing Complete,**</u>
- scripts have **limited complexity and predictable execution times.**
- Script is not a **general-purpose language.**
- These limitations ensure that the **language cannot be used to create an infinite loop or other form of “logic bomb” that could be embedded in a transaction in a way that causes a denial-of-service attack against the bitcoin network**.

Transaction Scripts and Script Language

Turing Incompleteness

- **Every transaction is validated by every full node on the bitcoin network.**
- A **limited language prevents the transaction validation mechanism from being used as a vulnerability.**

Transaction Scripts and Script Language

Stateless Verification

- The bitcoin transaction script language is stateless, in that **there is no state prior to execution of the script, or state saved after execution of the script.**
- Therefore, **all the information needed to execute a script is contained within the script.**
- A **script will predictably execute the same way on any system**.
- A valid transaction is valid for everyone and everyone knows this. **This predictability of outcomes is an essential benefit of the bitcoin system.**

Transaction Scripts and Script Language

<u>Script Construction (Lock + Unlock)</u>

- Bitcoin's transaction **validation engine relies on two types of scripts to validate trans- actions:**
    - a locking script and
    - an unlocking script.

Transaction Scripts and Script Language

Script Construction (Lock + Unlock)

A locking script

- **spending condition placed on an output**:
- it specifies the **conditions that must be met to spend the output in the future.**
- called a **scriptPubKey**, because it usually **contained a public key or bitcoin address (public key hash).**
- referred to as a **witness script / cryptographic puzzle.**

Transaction Scripts and Script Language

Script Construction (Lock + Unlock)

An unlocking script

- A **script that "solves," or satisfies, the conditions placed on an output by a locking script** and **allows the output to be spent**.
- **part of every transaction input.**
- they contain a **digital signature produced by the user's wallet** from his or her private key.
- called **scriptSig,** because as they **contained a digital signature**

Transaction Scripts and Script Language

<u>Script Construction (Lock + Unlock)</u>

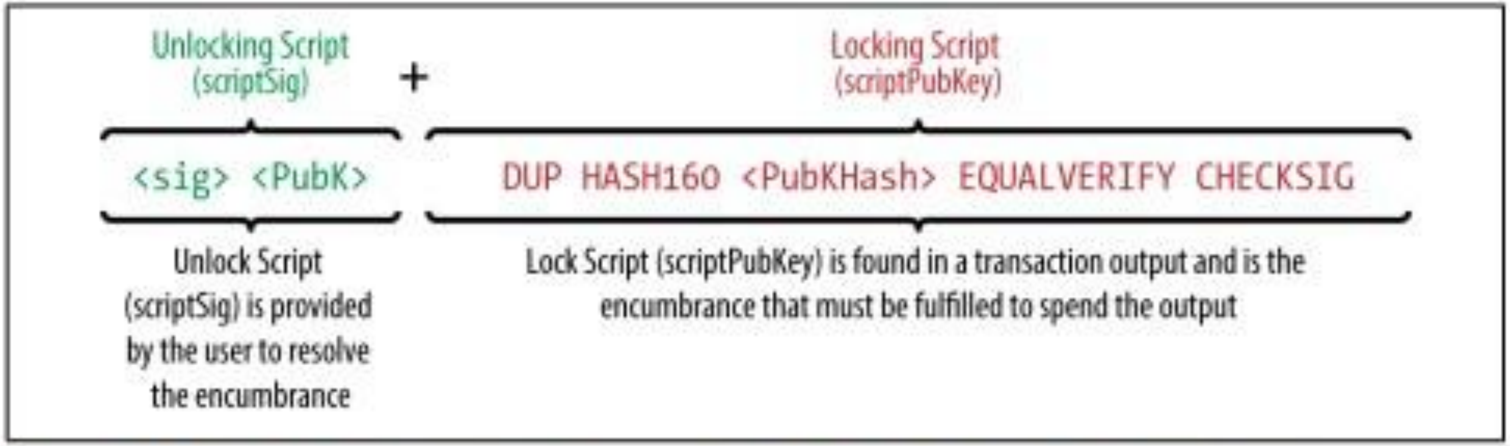
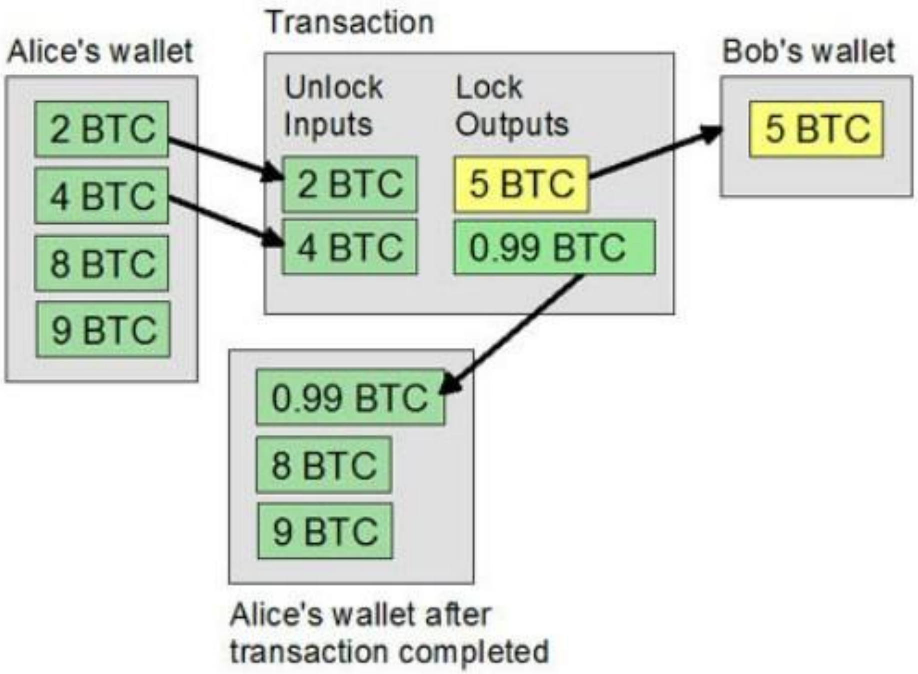


Figure 6-3. Combining scriptSig and scriptPubKey to evaluate a transaction script

Transaction Scripts and Script Language

- When bitcoins are sent to a recipient,
    - Script commands in an **unlocking script (scriptSig)** validate the available bitcoins (UTXOs)
    - Script commands in a **locking script (scriptPubKey)** set the conditions for spending them.

Courtesy : [PC Magazine]

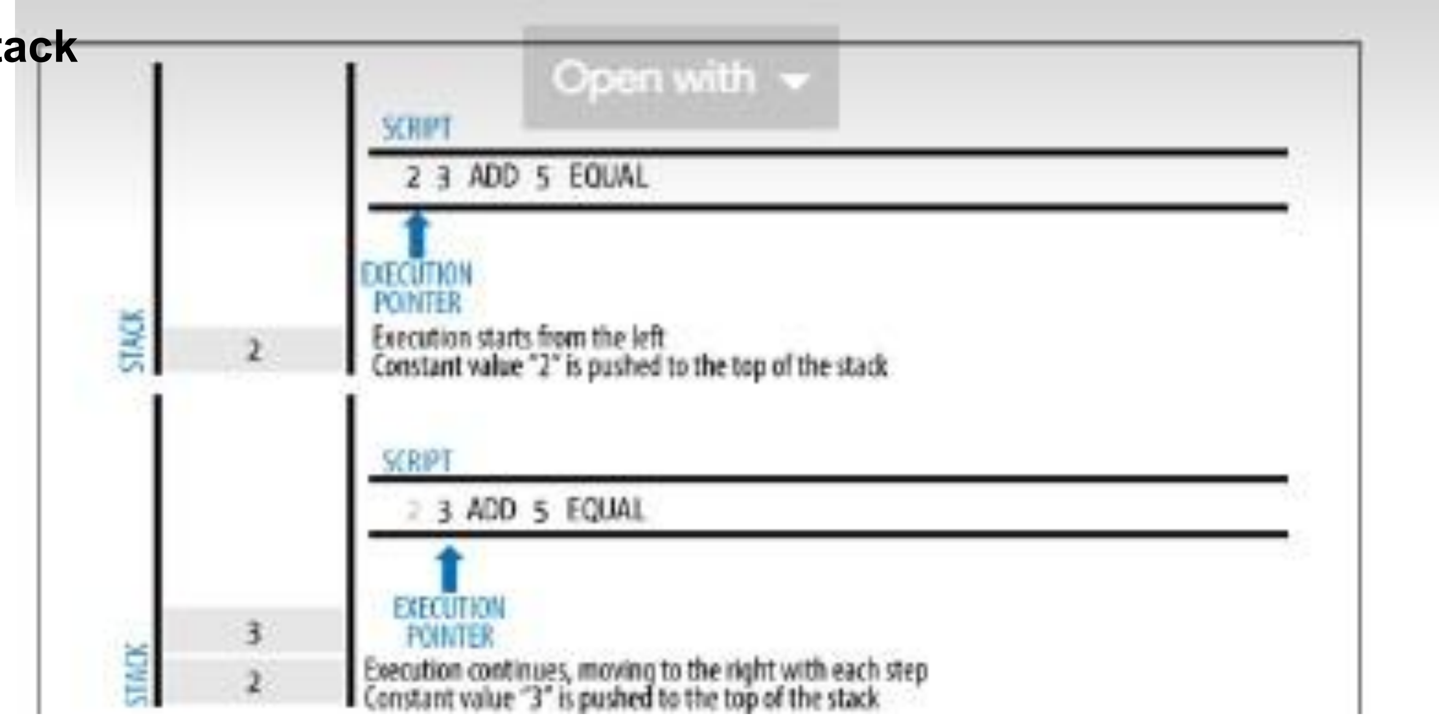


Inputs Are Unlocked and Outputs Are Locked

The unlocking script validates the Bitcoin inputs, and the locking script sets the conditions for spending the transferred coins. For more details, see Bitcoin transaction.

Transaction Scripts and Script Language

script execution stack



Transaction Scripts and Script Language

script execution stack

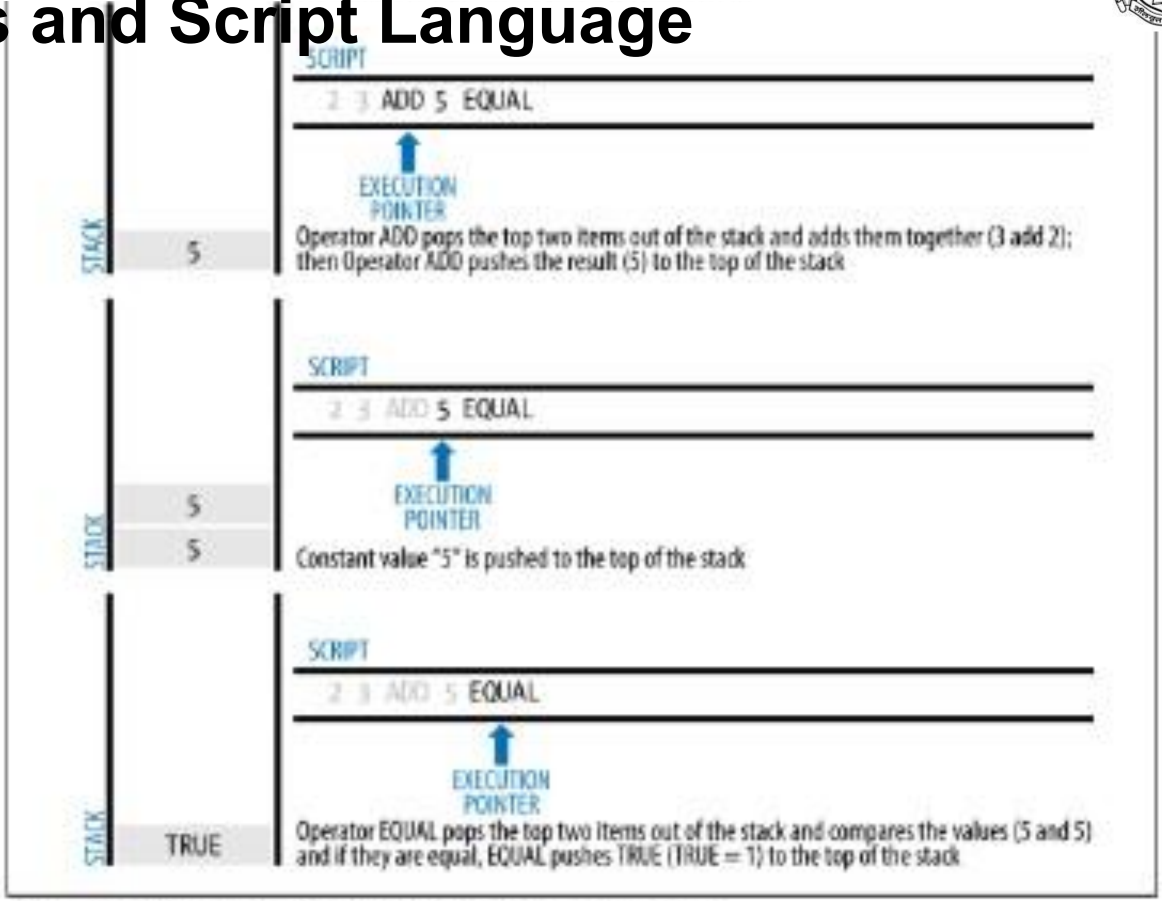


Figure 6-4. Bitcoin's script validation doing simple math

Pay-to-Public-Key-Hash (P2PKH)

- The vast majority of transactions processed on the bitcoin network spend outputs locked with a Pay-to-Public-Key-Hash or "P2PKH" script.
- These **outputs contain a locking script that locks the output to a public key hash**, more commonly known as a bitcoin address.
- An **output locked by a P2PKH script can be unlocked (spent) by presenting a public key and a digital signature created by the corresponding private key**

Pay-to-Public-Key -Hash (P2PKH)

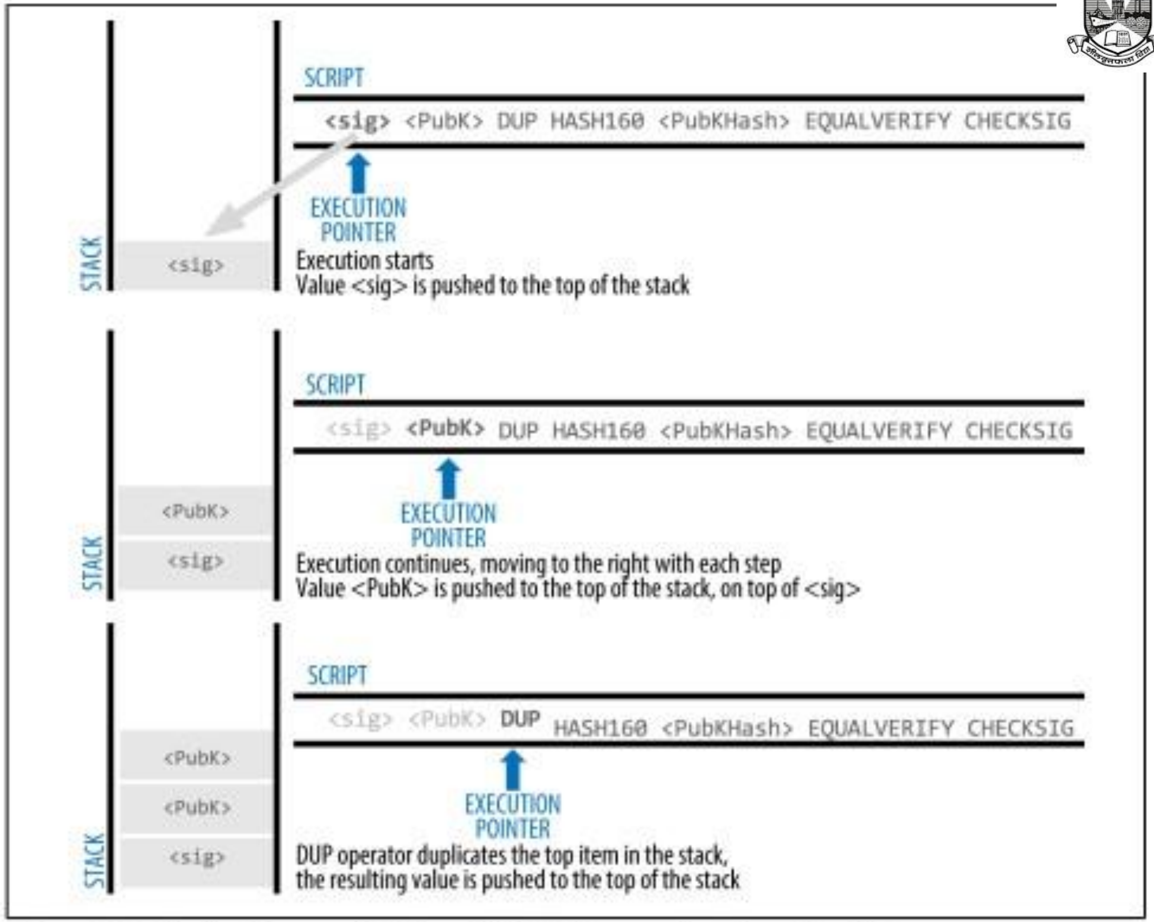


Figure 6-5. Evaluating a script for a P2PKH transaction (part 1 of 2)

Pay-to-Public-Key -Hash (P2PKH)

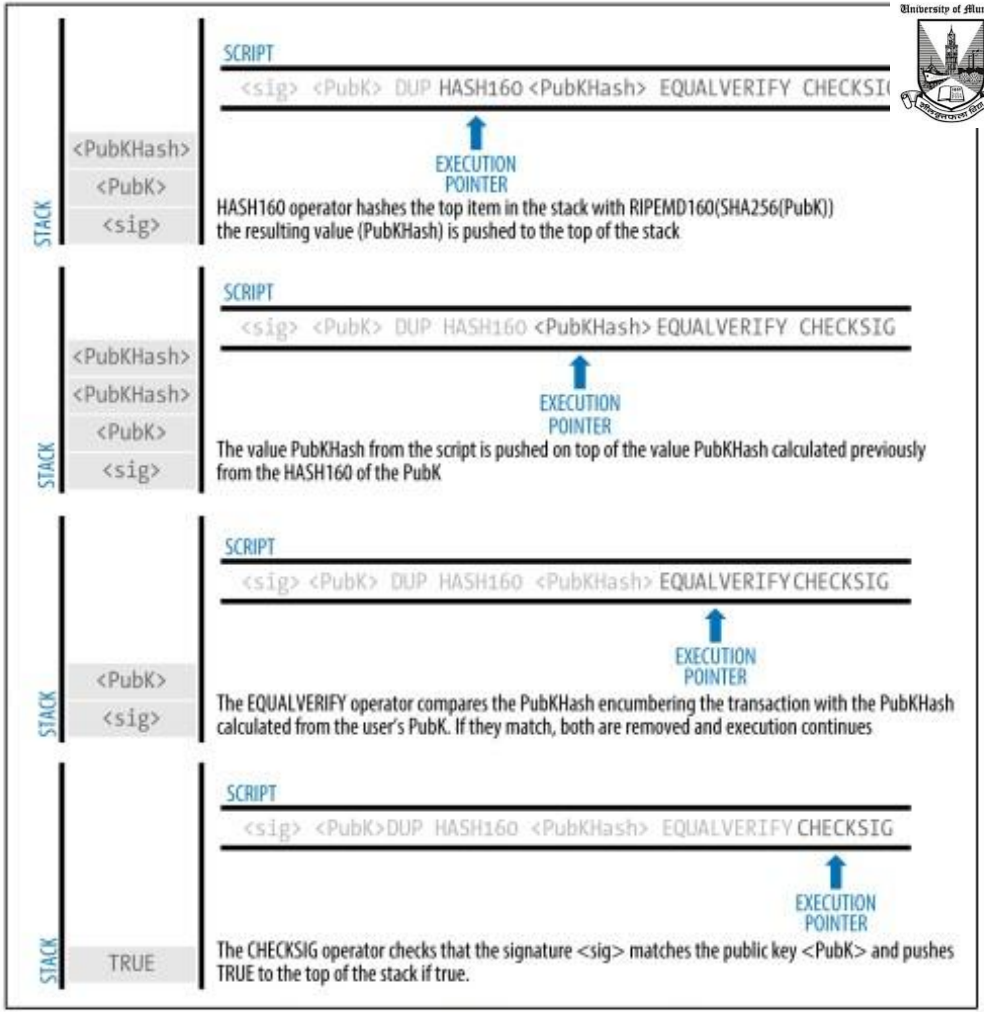


Figure 6-6. Evaluating a script for a P2PKH transaction (part 2 of 2)