Understanding Bitcoin Transactions and Privacy

While Bitcoin is often perceived as anonymous, its underlying technology, the blockchain, is fundamentally transparent. This document will explain how Bitcoin transactions work, how they can be traced, the different types of addresses, the mechanics of wallet generation, and various privacy-enhancing techniques like CoinJoin.

What Does "Ledger" Mean in Bitcoin?

In the context of Bitcoin, a **ledger** refers to the **blockchain** itself. It functions as a public, distributed, and immutable digital record of all Bitcoin transactions.

- **Public:** Every transaction is visible to anyone on the network.
- **Distributed:** The ledger is not stored in one central location but is replicated and maintained across thousands of computers (nodes) globally. This decentralization ensures security and censorship resistance.
- **Immutable:** Once a transaction is validated and added to a block on the blockchain, it cannot be altered or removed, guaranteeing the integrity of the transaction history.

This continuous and verifiable record is the foundation upon which the tracing of Bitcoin funds is built.

Prefixes for Bitcoin Addresses

Bitcoin addresses are alphanumeric strings used to send and receive Bitcoin. Different prefixes indicate different address formats, each with specific characteristics related to security, efficiency, and compatibility.

- **1 (P2PKH - Pay-to-Pubkey Hash)**
  - **Description:** The original Bitcoin address format, introduced in 2009. These addresses are still widely supported but are less efficient in terms of transaction size (leading to slightly higher fees) compared to newer formats. They are also case-sensitive.
- **3 (P2SH - Pay-to-Script Hash)**
  - **Description:** Introduced in 2012, these addresses offer more flexibility and support complex functionalities like multi-signature (multisig) transactions (requiring multiple private keys to authorize) and wrapped SegWit (Segregated Witness) transactions. They are case-sensitive.
- **bc1q (Bech32 - Native SegWit)**
  - **Description:** Introduced with SegWit in 2016, these are the **native SegWit** addresses.
  - **Characteristics:**
    - **More efficient:** Transactions using Bech32 addresses are generally smaller, resulting in lower transaction fees.
    - **Improved error detection:** They employ a robust checksum mechanism that significantly reduces the chance of typos leading to lost funds.
    - **Case-insensitive:** Easier to read and share without worrying about

capitalization errors.

- **bc1p (Bech32m - Taproot)**
  - **Description:** The newest address format, introduced with the Taproot upgrade in November 2021.
  - **Characteristics:**
    - A variant of Bech32, also case-insensitive with strong error detection.
    - Enables the benefits of Taproot, including Schnorr signatures and MAST (Merkelized Abstract Syntax Trees), leading to enhanced privacy and efficiency for complex transactions.

How Error Detection Works in Bech32

Bech32 (and Bech32m) addresses incorporate a highly robust error detection mechanism based on a **BCH code (Bose-Chaudhuri-Hocquenghem code)**.

1. **Polynomial Representation:** The entire address (including the human-readable prefix and the data part) is treated as a mathematical polynomial.
2. **Checksum Calculation:** This polynomial is divided by a specific "generator polynomial." The remainder of this division forms the checksum.
3. **Appending Checksum:** This checksum is appended to the data part of the address.
4. **Verification:** When a wallet or network node receives a Bech32 address, it performs the same calculation. If the remainder of the division is zero, the address is valid. Any other remainder indicates an error.

This system is designed to detect almost all single-character errors, two-character errors, and transposition errors, significantly reducing the risk of sending funds to a non-existent or unintended address due to a typo. It immediately flags an invalid address rather than allowing funds to be sent to a subtly corrupted but still valid address.

Wallet Clustering: Elaborated

Wallet clustering is a fundamental blockchain analytics technique used to overcome Bitcoin's pseudonymity by grouping addresses that are likely controlled by the same entity or individual.

- **The Problem:** An entity might use numerous unique Bitcoin addresses, making it difficult to discern their overall financial activity by looking at individual addresses alone.
- The Solution: Clustering Heuristics (Rules of Thumb):
  Blockchain analytics tools employ various rules and assumptions to identify and group these addresses:
  1. **Common Input Ownership Heuristic:** If multiple Bitcoin addresses are used as inputs in a single transaction, it's highly probable that all those input addresses are controlled by the same owner. This is because to spend multiple unspent transaction outputs (UTXOs) in one transaction, the same private key(s) must authorize them.
  2. **Change Address Detection:** When a Bitcoin transaction occurs, any "change" from the spent UTXOs is typically sent back to a new address controlled by the sender. Analysts look for patterns (e.g., one output being a specific odd amount while others are round numbers, or an output going to a previously unseen address that is immediately spent in a subsequent transaction) to identify these change addresses

and link them to the same wallet.
3. **Known Entity Labeling:** If an address or a cluster of addresses interacts with a known entity (e.g., a major cryptocurrency exchange, a darknet marketplace, a gambling site, a sanctioned entity, or a known scam address), this information is used to label the cluster. Analytics firms maintain vast databases of such affiliations.
4. **Deposit Address Re-use/Association:** Some services (like exchanges) issue unique deposit addresses. If a user consistently sends funds from the same set of private addresses to a single deposit address, those private addresses can be grouped.
5. **Timing Analysis:** While less direct for clustering, simultaneous transactions or very close transaction times can sometimes suggest common control, especially in complex obfuscation schemes.

- **Why is Wallet Clustering Crucial for Tracing?**
  - **Reveals the True Scale:** It transforms a view of thousands of disparate addresses into a manageable number of clusters, each representing a single entity's consolidated financial activity.
  - **Simplifies Investigations:** It allows investigators to focus on the overall flow of funds rather than getting lost in individual transactions.
  - **Enables Attribution:** By identifying clusters that interact with known entities (especially regulated exchanges that collect KYC data), law enforcement can bridge on-chain data to real-world identities.
  - **Identifies Illicit Activity:** Clustering helps detect funds flowing to and from known illicit services (e.g., ransomware, darknet markets, terrorist financing).
- Cryptographic Knowledge of Wallet Ownership:
  Cryptographically, you cannot directly know if two seemingly random Bitcoin addresses belong to the same wallet unless you possess the master seed/private key that generated them. Each address is derived from a unique private key. However, as described above, blockchain analysis relies on heuristics and statistical analysis to make highly probable inferences about address ownership, even without cryptographic proof.

How the Master Phrase (Seed Phrase) Works and How Addresses Are Generated

This system provides a user-friendly way to back up and restore an entire Bitcoin wallet.

- **The Master Phrase (Seed Phrase / Recovery Phrase / Mnemonic Phrase):**
  - A sequence of 12 or 24 common words (e.g., "alpha," "bravo," "charlie").
  - It is the **master key** to your entire wallet, used to derive all your private keys and public addresses.
  - It's designed to be human-readable and easy to write down and store securely.
- **Hierarchical Deterministic (HD) Wallets:**
  - Modern Bitcoin wallets use the HD wallet standard (BIP32). This means that from a single master seed (derived from your master phrase), an infinite number of private and public key pairs (and thus addresses) can be **deterministically generated**.
  - "Deterministic" means the same seed phrase will always produce the exact same

sequence of keys and addresses.
    - "Hierarchical" refers to a tree-like structure, allowing for organized generation of keys (e.g., for different accounts or purposes).
- **How Addresses are Generated from the Master Phrase:**
    1. **Mnemonic Phrase to Seed (BIP39):** Your 12 or 24 words are converted into a 512-bit binary **seed** using a cryptographic function (PBKDF2-HMAC-SHA512), optionally incorporating a user-defined passphrase for added security.
    2. **Seed to Master Private Key (BIP32):** The 512-bit seed is used to generate a **master private key** (256 bits) and a **master chain code** (256 bits), which are the root of your entire key derivation tree.
    3. **Master Private Key to Child Private Keys:** Using the master private key, chain code, and a deterministic algorithm, the wallet generates an unlimited number of unique **child private keys** according to a specified **derivation path**.
    4. **Private Key to Public Key:** Each private key is used to generate its corresponding **public key** via elliptic curve cryptography (ECC). This is a one-way mathematical function.
    5. **Public Key to Bitcoin Address:** The public key is then hashed (e.g., using SHA256 then RIPEMD160) and encoded into one of the various Bitcoin address formats (1, 3, bc1q, bc1p), including a checksum for error detection.

Derivation Path

A **derivation path** is a structured hierarchy that specifies how keys and addresses are generated from a master seed in an HD wallet. It acts like a file path, indicating the exact "location" of a specific key and address within the vast tree of possible keys.

- **Format:** Typically represented as: m / purpose' / coin_type' / account' / change / address_index
    - **m**: Represents the master key.
    - **purpose'**: (The apostrophe ' denotes a "hardened" derivation, increasing security). This specifies the wallet standard (e.g., 44' for P2PKH, 84' for Native SegWit, 86' for Taproot).
    - **coin_type'**: Differentiates cryptocurrencies (e.g., 0' for Bitcoin, 60' for Ethereum).
    - **account'**: Allows for multiple "accounts" within a single wallet (e.g., 0' for the first account).
    - **change**: Separates external (receiving) addresses (0) from internal (change) addresses (1).
    - **address_index**: The sequential index for each address generated within that specific account and change type (e.g., 0 for the first address, 1 for the second).
- **Why is it important for compatibility?** Different wallet software might use different default derivation paths. Knowing or selecting the correct derivation path is crucial when restoring a wallet from a seed phrase on a different wallet application to ensure all funds are correctly located.
- **Wallet Iteration to Find Funds:** When restoring a wallet, the software doesn't know which specific addresses you've used. It systematically generates addresses from the seed using

common derivation paths, then queries the blockchain for transaction history for each generated address. Wallets typically use a "gap limit" (e.g., 20 unused addresses) to stop scanning after a certain number of empty addresses, assuming all active addresses have been found.

Script in the Context of a Bitcoin Transaction

Bitcoin transactions utilize a simple, stack-based, non-Turing complete programming language called **Bitcoin Script**. This script defines the conditions that must be met for Bitcoin to be spent.

- **How it Works (High-Level):**
    1. **Stack:** Data is processed on a Last-In, First-Out (LIFO) stack.
    2. **Opcodes (Instructions):** Script consists of various operations (opcodes) that manipulate data on the stack.
    3. **Execution:** When a transaction is validated, two main script components are concatenated and executed:
        - **scriptPubKey (Locking Script):** Part of a transaction's output. It's a "lock" dictating conditions for future spending of those coins (e.g., requiring a specific public key and signature).
        - **scriptSig (Unlocking Script / Witness Data):** Part of a transaction's input. It's the "key" providing data to satisfy the scriptPubKey's conditions (e.g., the sender's public key and digital signature).
    4. **Validation:** The transaction is valid if the combined script execution results in "True" (a non-zero value on top of the stack).
- **Key Types of Instructions (Opcodes):**
    - **Push Data:** Opcodes to push data (public keys, signatures) onto the stack.
    - **Cryptographic:** Perform operations like OP_CHECKSIG (verifies a digital signature) and hashing functions (OP_HASH160).
    - **Control Flow:** OP_IF, OP_ELSE, OP_ENDIF for conditional execution.
    - **Arithmetic:** Basic math operations (many are disabled for security).
    - **Time Lock:** OP_CHECKLOCKTIMEVERIFY (CLTV) to lock funds until a certain time or block height.

Bitcoin Script's simplicity ensures predictability and security, enabling standard payments, multi-signature transactions, and other conditional spending rules.

Schnorr Signatures and Multisig Indistinguishability (and Taproot Script Revelation)

Prior to Taproot, multi-signature (multisig) transactions were visibly different from single-signature transactions on the blockchain, revealing the complexity of the spending condition and potentially reducing privacy.

- **The Problem Before Taproot (ECDSA Signatures):** Traditional ECDSA signatures (used for 1 and 3 addresses) explicitly revealed the multiple public keys and signatures involved in a multisig setup, making it easy for observers to identify and differentiate such transactions.

- How Schnorr Signatures with Taproot achieve Indistinguishability:
  Schnorr signatures, introduced with Taproot, possess "linearity," allowing for signature aggregation.
  1. **Key Aggregation:** Multiple signers can combine their individual public keys into a single, aggregated public key. They then collaboratively create a *single* Schnorr signature that is valid for this aggregated key.
  2. **Spending Paths (MAST Integration):** Funds locked to a Taproot (bc1p) address can be spent via a "key path" or a "script path." The key path is the preferred and most private method.

  When funds are spent via the **key path**, the blockchain only records the aggregated public key and a single aggregated Schnorr signature. **This makes multisignature transactions cryptographically indistinguishable from simple single-signature transactions** to an external observer. This significantly enhances privacy, improves fungibility, and makes it harder for blockchain analysts to link specific entities based on complex transaction patterns.
- Taproot Script Revelation:
  For Taproot outputs, the actual script (or specific spending conditions within a MAST) is not revealed on the blockchain until the output is spent.
  - If the funds are spent via the **key path**, no script information is revealed at all. The transaction remains indistinguishable from a standard single-signature spend.
  - If one of the alternative spending conditions (a script within the MAST) is used (the "script path"), then **only that specific script and the necessary Merkle proof** (to demonstrate its inclusion in the original Merkle Root) are revealed on the blockchain as part of the unlocking script. All other unexecuted scripts within the Merkle tree remain private. This "reveal-on-spend" mechanism is a core privacy feature of Taproot.

MAST (Merkelized Abstract Syntax Trees)

MAST is a feature introduced with Taproot that optimizes the handling of complex spending conditions, offering efficiency and privacy improvements.

- **The Problem Before MAST:** For complex Bitcoin scripts with multiple spending conditions (e.g., various participants, time locks, or oracle conditions), the *entire* script had to be revealed on the blockchain upon spending, even if only one condition was met. This resulted in larger transactions and exposed unnecessary information.
- **How MAST Works:**
  1. **Condition Breakdown:** Each possible spending condition (a small Bitcoin script) is treated as a separate "leaf" in a Merkle tree.
  2. **Hashing:** Each script leaf is hashed, and these hashes are then combined and re-hashed in pairs, forming a tree structure that culminates in a single **Merkle Root**.
  3. **Inclusion in Output:** This compact Merkle Root is included as part of the Taproot output key in the transaction.
  4. **Conditional Revelation:** When the funds are spent, only two things are revealed on the blockchain:

        - The *specific script condition* that was met.
        - The **Merkle proof** (a small set of hashes from the other branches in the tree) that cryptographically proves this specific script is indeed part of the original, larger Merkle Root.
- **Benefits of MAST:**
    - **Privacy:** Only the executed spending condition is revealed; all unexecuted conditions remain hidden, significantly enhancing privacy for complex smart contracts. This is because the full script is only revealed at the moment it's needed for a script-path spend.
    - **Efficiency:** Revealing only a small portion of the script and its Merkle proof results in smaller transaction sizes and lower fees.
    - **Scalability:** Reduced transaction size contributes to overall blockchain scalability.

Mixer (Tumbler) and CoinJoin

These are privacy-enhancing techniques designed to obscure the link between Bitcoin transaction inputs and outputs.

- **Mixer (Tumbler):**
    - **Concept:** A centralized service that pools cryptocurrency from many users and then redistributes an equivalent amount (minus a fee) to new addresses provided by those users.
    - **How it works:** Users deposit coins to the mixer, which shuffles them in a large pool, often with delays and internal transactions, before sending "new" coins to the users' withdrawal addresses.
    - **Risks:**
        - **Custodial Risk:** Users must trust the third-party mixer, risking loss of funds if the mixer is a scam, hacked, or legally compelled to reveal user data.
        - **Logging:** Centralized mixers may keep records, undermining privacy.
        - **Tainted Funds:** Users might receive coins previously involved in illicit activities.
- **CoinJoin:**
    - **Concept:** A **decentralized, collaborative transaction** where multiple users combine their inputs into a single, large transaction with multiple outputs of the *same value*. The goal is to make it impossible to determine which input corresponds to which output.
    - **How it works:** Participants agree to a CoinJoin, contribute their inputs, and provide new output addresses. A coordinator bundles these inputs and outputs into one transaction, which each participant signs only for their own inputs. The transaction is then broadcast.
    - **Key Difference from Mixers:** CoinJoin is **non-custodial**. Participants retain control of their private keys throughout the process, eliminating counterparty risk. The privacy relies on cryptographic plausible deniability within the mixed transaction.
- **Safety and Privacy Trade-offs: Mixers vs. CoinJoin:**
    - **Safety: CoinJoin is significantly safer** due to its non-custodial nature. Mixers carry substantial custodial risk.

- **Privacy:** CoinJoin generally offers stronger, cryptographically guaranteed privacy. While mixers *can* provide privacy, it relies entirely on trusting a third party not to log data or be compromised. CoinJoin is decentralized and resistant to single points of failure.
- **Convenience:** Centralized mixers are often easier to use for less technical users, but this convenience comes at a high cost of trust and security. CoinJoin requires more user understanding and coordination.
- **Conclusion:** For privacy and security, CoinJoin is almost always preferable to centralized mixers, despite potentially being less convenient.

How BIP39 Works (Mnemonic Code for Generating Deterministic Keys)

BIP39 is a standard that defines how a human-readable sequence of words (mnemonic seed phrase) is generated from random data, which then serves as a backup to deterministically derive all private keys and addresses in a wallet.

1. **Generate Randomness (Entropy):** The wallet generates a highly random sequence of bits (e.g., 128, 192, or 256 bits), which determines the length of the seed phrase (12, 18, or 24 words).
2. **Add a Checksum:** A small portion of the hash of the random data is appended as a checksum. This helps detect typos if the phrase is manually entered later.
3. **Divide into 11-bit Chunks:** The combined random bits and checksum bits are divided into 11-bit segments.
4. **Map to a Wordlist:** Each 11-bit chunk corresponds to a unique word in a standardized list of 2048 words (BIP39 wordlist).
5. **Result: The Mnemonic Seed Phrase:** The sequence of these words forms the seed phrase.
6. **Mnemonic to Seed:** When restoring a wallet, the seed phrase (and optional passphrase) is put through a key derivation function (PBKDF2-HMAC-SHA512) to produce a 512-bit binary **seed**. This seed is then used by BIP32 to generate all subsequent keys and addresses.

BIP39 provides a user-friendly and standardized way to back up and recover Bitcoin wallets across different software and hardware, emphasizing security through truly random initial generation and a robust checksum.

Example: Master Phrase to Bitcoin Address Derivation (Conceptual)

Let's illustrate the flow from a hypothetical (and **never-to-be-used-in-real-life**) 12-word BIP39 seed phrase to a Native SegWit (bc1q) address.

Hypothetical Seed Phrase: alpha bravo charlie delta echo foxtrot golf hotel india juliett kilo lima

1. **Mnemonic Phrase to Seed (BIP39):**
   - The 12 words (and an optional passphrase) are processed through BIP39's PBKDF2 function.
   - **Output:** A 512-bit binary **seed**. (e.g., 0110101010101010111100001010101010... - full binary string is too long to display).
2. **Seed to Master Private Key (BIP32):**
   - The 512-bit seed is used to generate a **master private key** (256 bits) and a **master**

chain code (256 bits).
    - **Output:**
        - Master Private Key (e.g., xprv... - a long alphanumeric string)
        - Master Chain Code (e.g., abcdef123456... - a long hexadecimal string)
3. **Master Private Key to Child Private Key (using Derivation Path - BIP84 for Native SegWit):**
    - We apply the standard Native SegWit derivation path for the first external address: m/84'/0'/0'/0/0
    - The wallet deterministically applies cryptographic functions using the master private key, chain code, and each path component.
    - **Output:** A specific **child private key** corresponding to m/84'/0'/0'/0/0. (e.g., L1T... - a different long alphanumeric string).
4. **Child Private Key to Public Key:**
    - The child private key is put through Elliptic Curve Cryptography (ECC).
    - **Output:** A corresponding **public key**. (e.g., 02... or 03... - a 33-byte hexadecimal string for a compressed public key).
5. **Public Key to Bech32 (Native SegWit) Address:**
    - The public key is first hashed (SHA256, then RIPEMD160) to get a 20-byte PubKeyHash.
    - This PubKeyHash forms the "witness program" for SegWit v0.
    - Finally, this witness program is encoded into the Bech32 format, including the bc1q prefix and the error-detecting checksum.
    - **Output:** A valid Native SegWit Bitcoin address (e.g., bc1q... - a long alphanumeric string).

This multi-step, deterministic process ensures that anyone with your master seed phrase can regenerate your entire wallet and access your funds.

- Wallet **nucleus market**: https://www.walletexplorer.com/wallet/NucleusMarket
- Wallet **055adaf7db**: https://www.walletexplorer.com/wallet/055adaf7db2e665f
- Wallet **22c116503b**: https://www.walletexplorer.com/wallet/22c116503b92d979
- State wallet **ca37dd0805**: https://www.walletexplorer.com/wallet/ca37dd08056c192a (address: bc1qm8955nwl67lw96ew03g9a0f2adl5nt2uqp79my)