

Bitcoin Scripts and Wallets

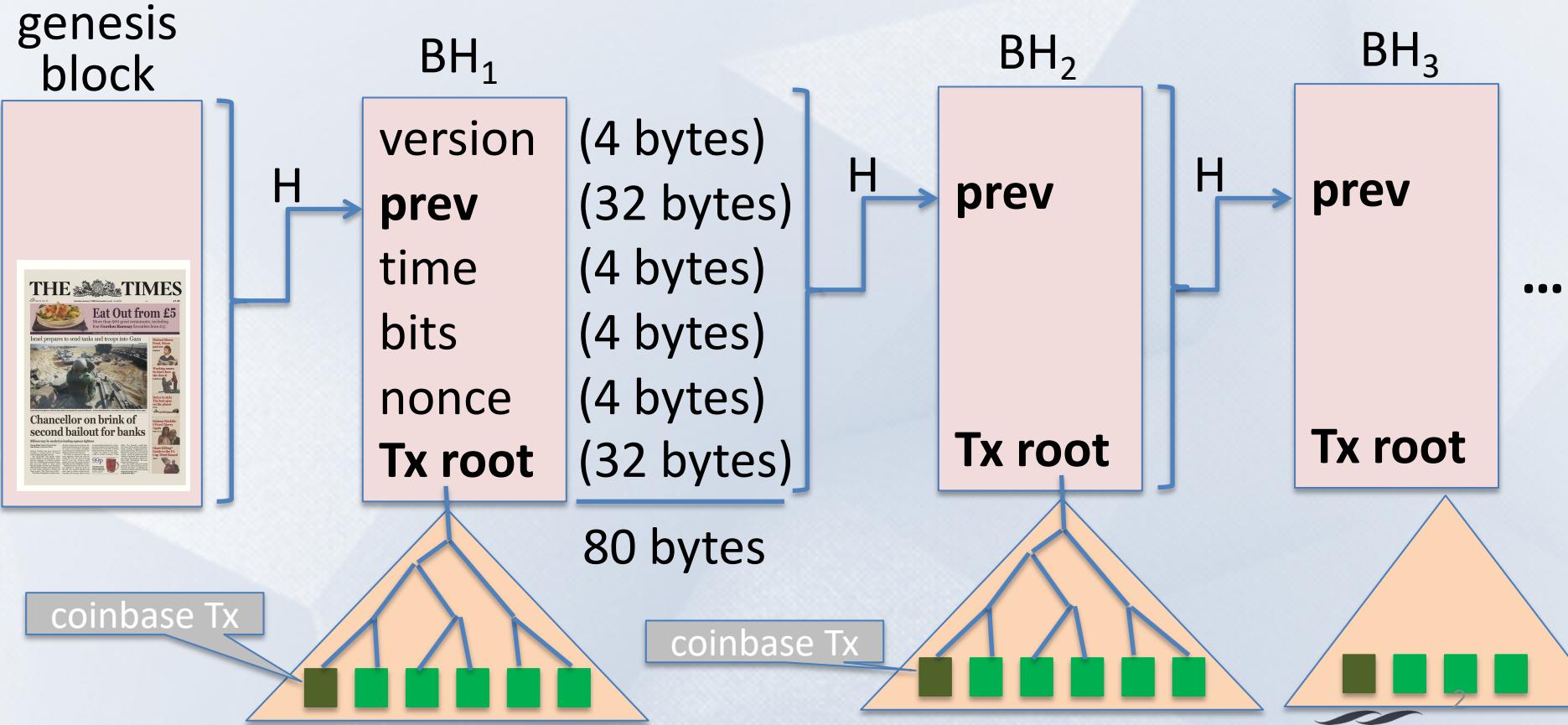
Yu Feng

University of California, Santa Barbara



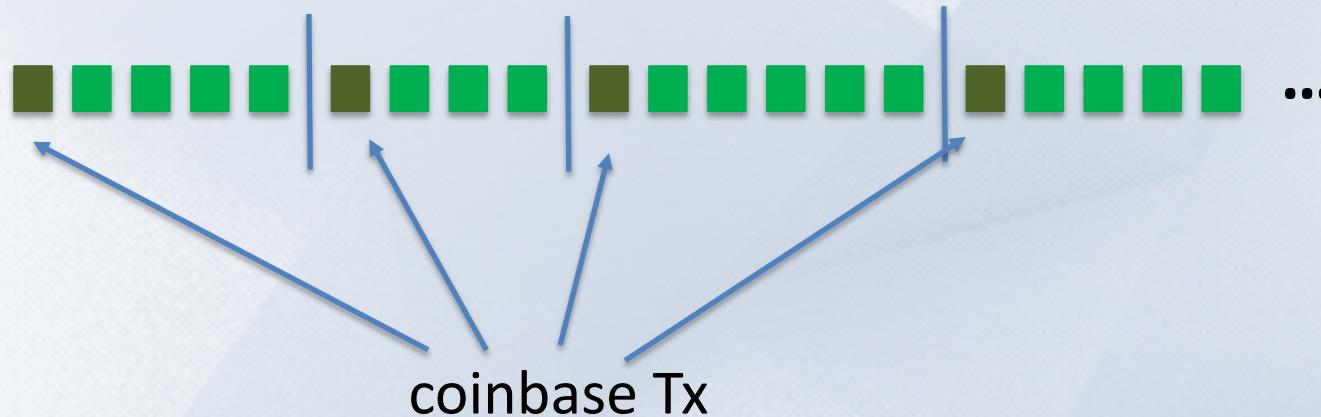
Slides adapted from Stanford CS251

Recap: the Bitcoin blockchain



Tx sequence

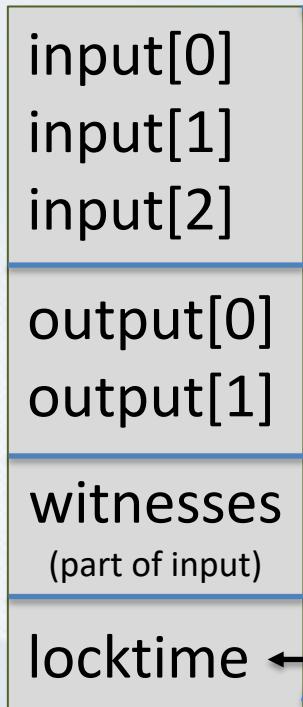
View the blockchain as a sequence of Tx (append-only)



Tx cannot be erased: mistaken Tx \Rightarrow locked or lost of funds

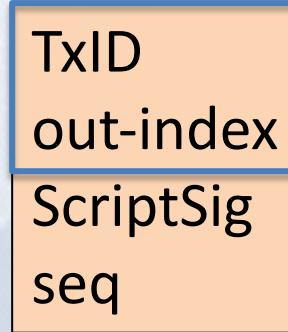
Tx structure (non-coinbase)

inputs
outputs
(segwit)
(4 bytes)



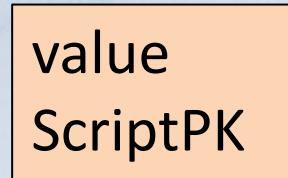
TxID = H(Tx)
(excluding witnesses)

input:



32 byte hash
4 byte index
program
ignore

output:



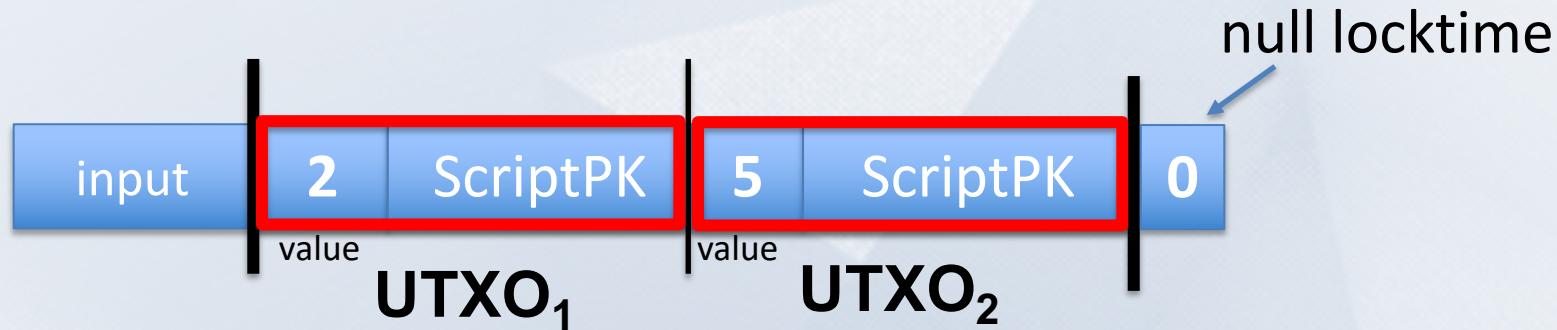
8 bytes
program

earliest block # that can include Tx

#BTC = value/10⁸

Example

Tx1:
(funding Tx)



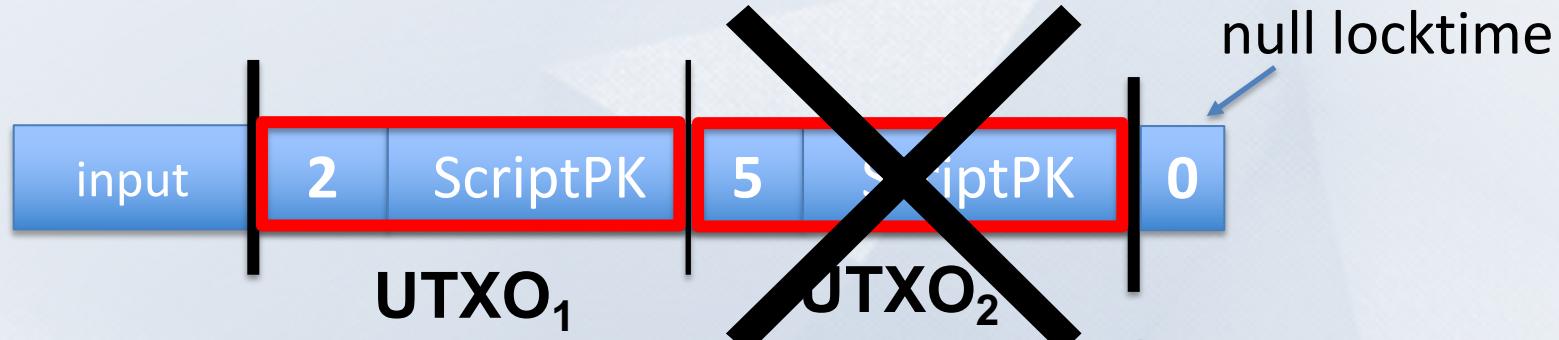
UTXO: unspent Tx output

Tx2:
(spending Tx)



Example

Tx1:
(funding Tx)



UTXO: unspent Tx output

Tx2:
(spending Tx)



Validating Tx2

Miners check (for each input):

1. The program **ScriptSig | ScriptPK** returns true
2. **TxID | index** is in the current UTXO set
3. sum input values \geq sum output values

After Tx2 is posted, miners remove $UTXO_2$ from UTXO set

program from funding Tx:
under what conditions
can UTXO be spent

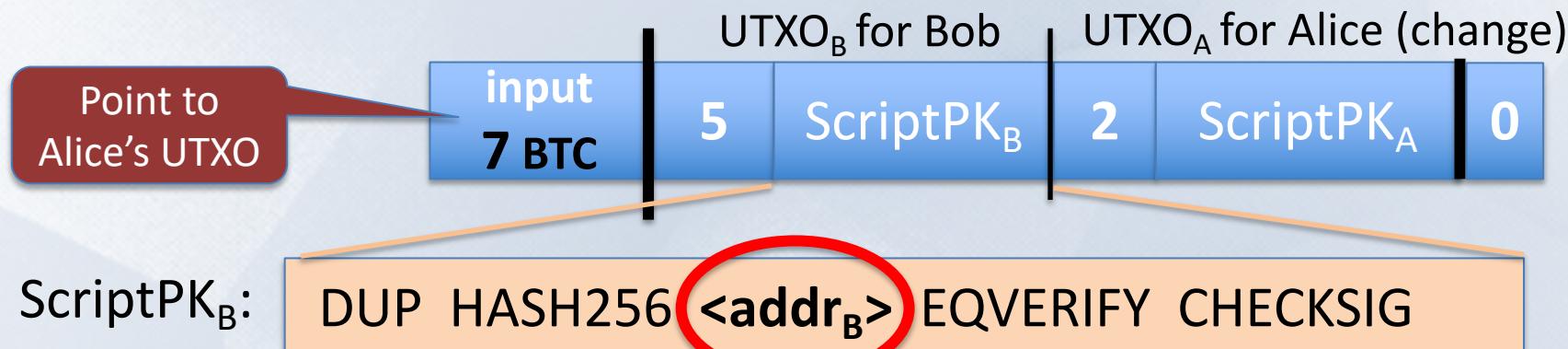
program from spending Tx:
proof that conditions
are met

Transaction types: (1) P2PKH

pay to public key hash

Alice want to pay Bob 5 BTC:

- step 1: Bob generates sig key pair $(pk_B, sk_B) \leftarrow Gen()$
- step 2: Bob computes his Bitcoin address as $addr_B \leftarrow H(pk_B)$
- step 3: Bob sends $addr_B$ to Alice
- step 4: Alice posts Tx:

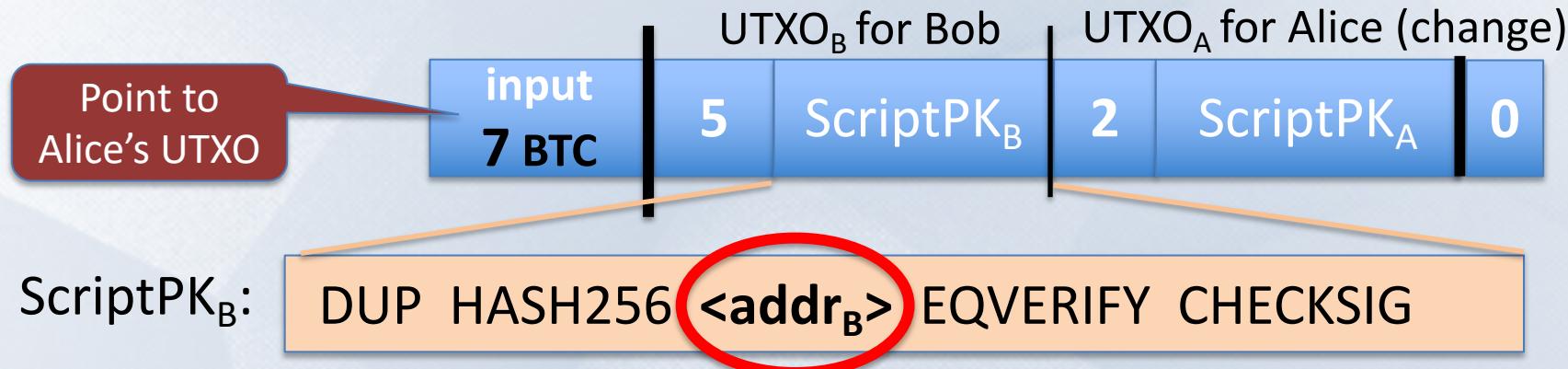


Transaction types: (1) P2PKH

pay to public key hash

“input” contains ScriptSig that authorizes spending Alice’s UTXO

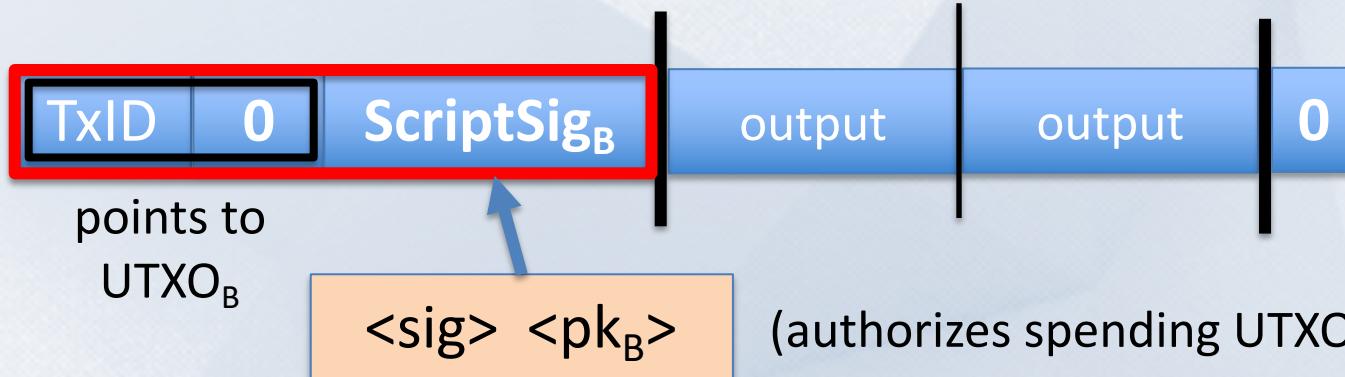
- example: ScriptSig contains Alice’s signature on Tx
⇒ miners cannot change ScriptPK_B (will invalidate Alice’s signature)



Transaction types: (1) P2PKH

Later, when Bob wants to spend his UTXO: create a Tx_{spend}

Tx_{spend} :



$<\text{sig}> = \text{Sign}(\text{sk}_B, \text{Tx})$ where $\text{Tx} = (T_{x_{spend}} \text{ excluding all ScriptSigs})$ (SIGHASH_ALL)

Miners validate that $\text{ScriptSig}_B \mid \text{ScriptPK}_B$ returns true

P2PKH: comments

- Alice specifies recipient's pk in UTXO_B
- Recipient's pk is not revealed until UTXO is spent
(some security against attacks on pk)
- Miner cannot change $\langle \text{Addr}_B \rangle$ and steal funds:
invalidates Alice's signature that created UTXO_B

Segregated Witness

ECDSA malleability:

- Given (m, sig) anyone can create (m, sig') with $\text{sig} \neq \text{sig}'$
- ⇒ miner can change sig in Tx and change TxID = SHA256(Tx)
 - ⇒ Tx issuer cannot tell what TxID is, until Tx is posted
 - ⇒ leads to problems and attacks

Segregated witness: signature is moved to witness field in Tx
TxID = Hash(Tx without witnesses)

Transaction types: (2) P2SH: pay to script hash

(pre SegWit in 2017)

Payer specifies a redeem script (instead of just pkhash)

- Usage:
- (1) Bob publishes hash(redeem script) ← Bitcoind addr.
 - (2) Alice sends funds to that address in funding Tx
 - (3) Bob can spend UTXO if he can satisfy the script

ScriptPK in UTXO: HASH160 <H(redeem script)> EQUAL

ScriptSig to spend: <sig₁> <sig₂> ... <sig_n> <redeem script>

payer can specify complex conditions for when UTXO can be spent

P2SH

Miner verifies:

- (1) <ScriptSig> ScriptPK = true \leftarrow spending Tx gave correct script
- (2) ScriptSig = true \leftarrow script is satisfied

Example P2SH: multisig

Goal: spending a UTXO requires t-out-of-n signatures

Redeem script for 2-out-of-3: (chosen by payer)

<2> <PK₁> <PK₂> <PK₃> <3> CHECKMULTISIG

threshold

hash gives P2SH address

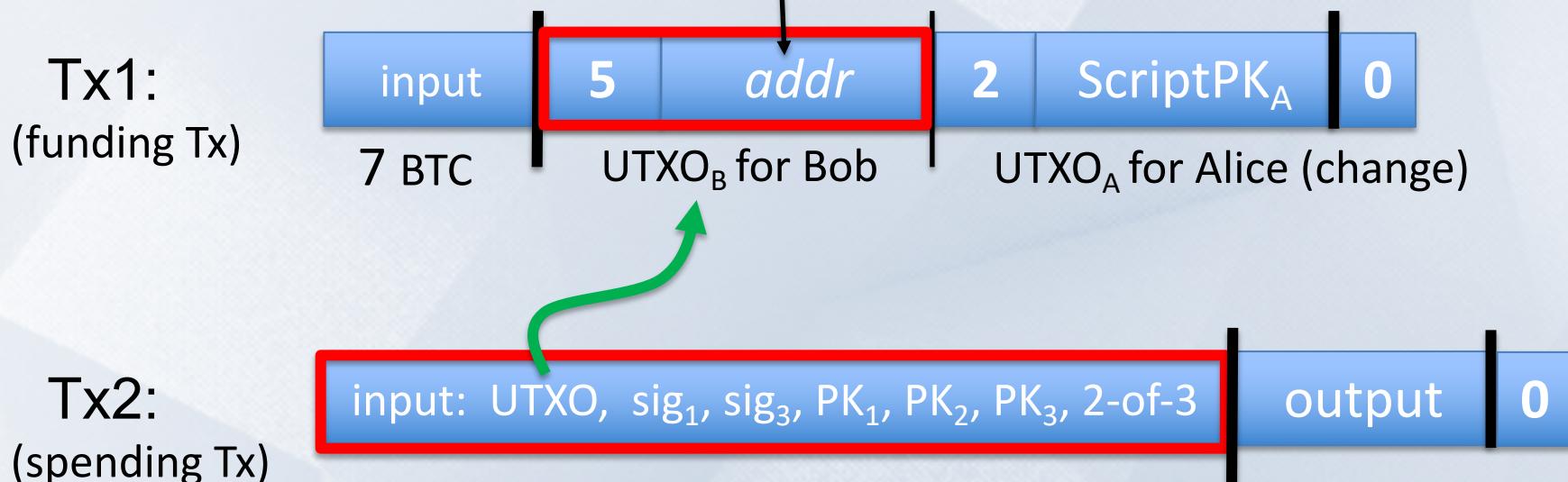
ScriptSig to spend: (by payee)

<0> <sig1> <sig3> <redeem script>

(in the clear)

Abstractly ...

Multisig address: $addr = H(PK_1, PK_2, PK_3, 2\text{-of-}3)$



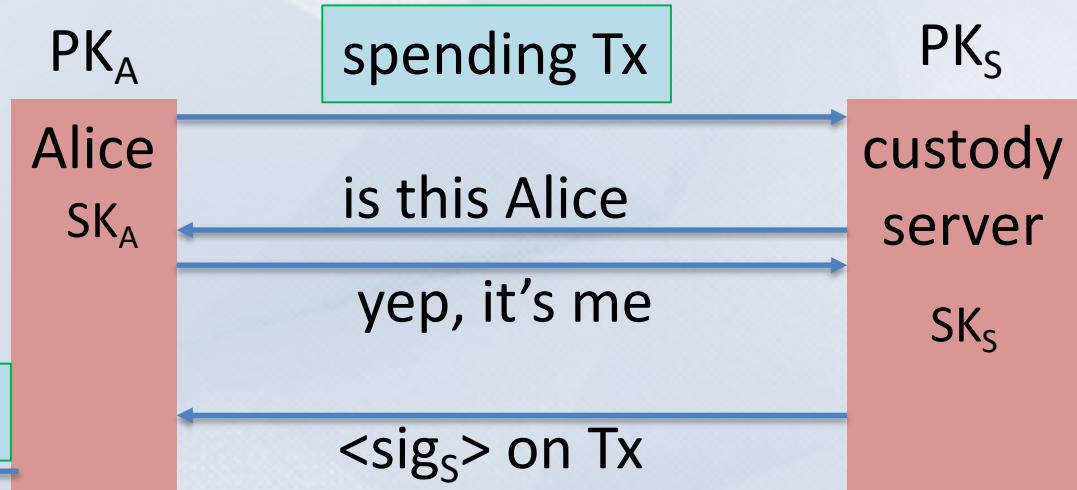
Example Bitcoin scripts



Protecting assets with a co-signatory

Alice stores her funds in UTXOs for

$$\text{addr} = \text{2-of-2}(\text{PK}_A, \text{PK}_S)$$



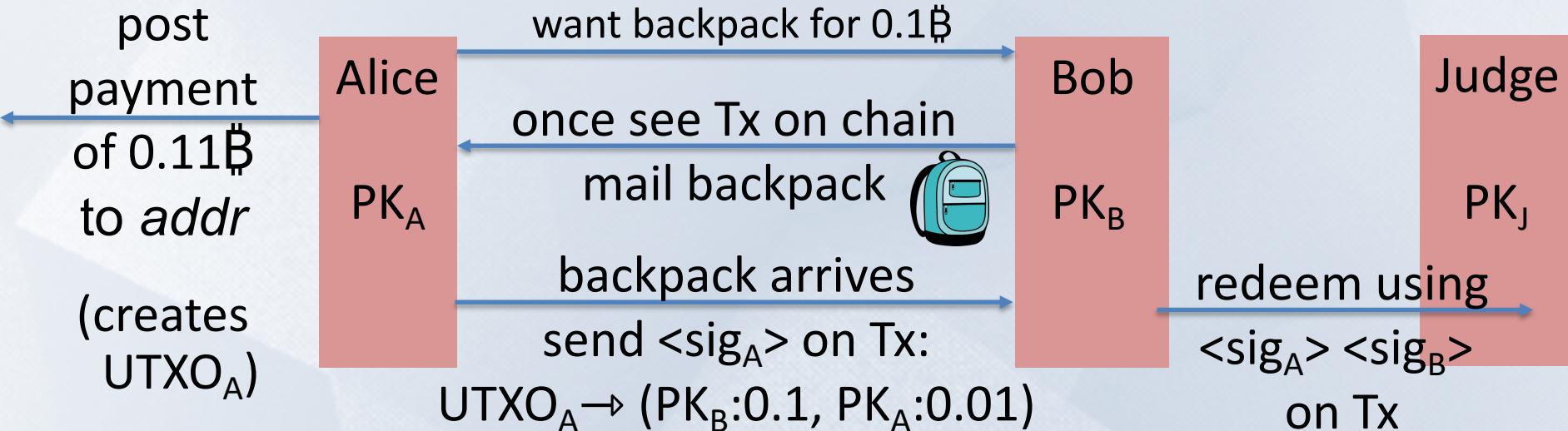
⇒ theft of Alice's SK_A does not compromise BTC

Escrow service

Alice wants to buy a backpack for 0.1฿ from merchant Bob

Goal: Alice only pays after backpack arrives, but can't not pay

$$addr = 2\text{-of-}3(PK_A, PK_B, PK_J)$$



Escrow service: a dispute

(1) Backpack never arrives: (Bob at fault)

Alice gets her funds back with help of Judge and a Tx:

Tx: ($\text{UTXO}_A \rightarrow \text{PK}_A$, sig_A , $\text{sig}_{\text{Judge}}$) [2-out-of-3]

(2) Alice never sends sig_A : (Alice at fault)

Bob gets paid with help of Judge and a Tx:

Tx: ($\text{UTXO}_A \rightarrow \text{PK}_B$, sig_B , $\text{sig}_{\text{Judge}}$) [2-out-of-3]

(3) Both are at fault: Judge publishes $\langle \text{sig}_{\text{Judge}} \rangle$ on Tx:

Tx: ($\text{UTXO}_A \rightarrow \text{PK}_A: 0.05$, $\text{PK}_B: 0.05$, $\text{PK}_J: 0.01$)

Now either Alice or Bob can execute this Tx.

Managing crypto assets: Wallets



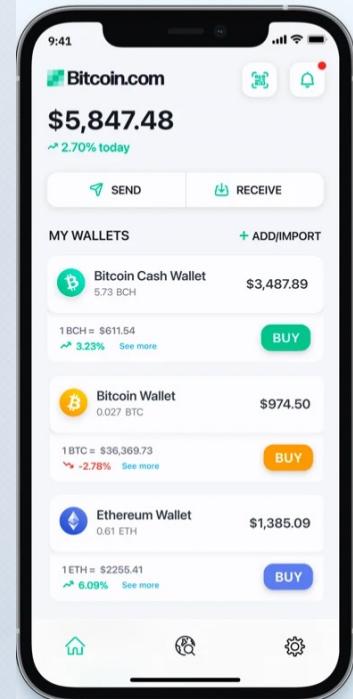
Managing secret keys

Users can have many PK/SK:

- one per Bitcoin address, Ethereum address, ...

Wallets:

- Generates PK/SK, and stores SK,
- Post and verify Tx,
- Show balances



Managing lots of secret keys

Types of wallets:

- **cloud** (e.g., Coinbase): cloud holds secret keys ... like a bank.
- **laptop/phone**: Electrum, MetaMask, ...
- **hardware**: Trezor, Ledger, Keystone, ...
- **paper**: print all sk on paper
- **brain**: memorize sk (bad idea)
- **Hybrid**: non-custodial cloud wallet (using threshold signatures)

client stores
secret keys



Not your keys, not your coins ... but lose key \Rightarrow lose funds

Simplified Payment Verification (SPV)

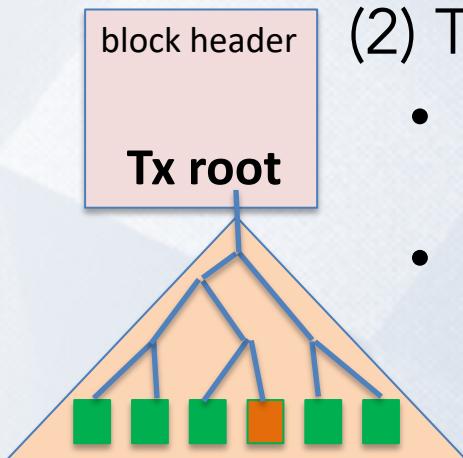
How does a client wallet display Alice's current balances?

- Laptop/phone wallet needs to verify an incoming payment
- Goal: do so w/o downloading entire blockchain (366 GB)

SPV: (1) download all block headers (60 MB)

(2) Tx download:

- wallet → server: list of my wallet addrs (Bloom filter)
- server → wallet: Tx involving addresses + Merkle proof to block header.



Simplified Payment Verification (SPV)

Problems:

- (1) **Security:** are BH the ones on the blockchain? Can server omit Tx?
 - Electrum: download block headers from ten random servers, optionally, also from a trusted full node.
List of servers: electrum.org/#community
- (2) **Privacy:** remote server can test if an *addr* belongs to wallet

We will see better light client designs later in the course (e.g. Celo)

Hardware wallet: Ledger, Trezor, ...

End user can have lots of secret keys. How to store them ???

Hardware wallet (e.g., Ledger Nano X)

- connects to laptop or phone wallet using Bluetooth or USB
- manages many secret keys
 - Bolos OS: each coin type is an app on top of OS
- PIN to unlock HW (up to 48 digits)
- screen and buttons to verify and confirm Tx



Hardware wallet: backup

Lose hardware wallet \Rightarrow loss of funds. What to do?

Idea 1: generate a secret seed $k_0 \in \{0,1\}^{256}$

for $i=1,2,\dots$: $sk_i \leftarrow \text{HMAC}(k_0, i)$, $pk_i \leftarrow g^{sk_i}$

ECDSA public key

pk_1, pk_2, pk_3, \dots : random unlinkable addresses (without k_0)

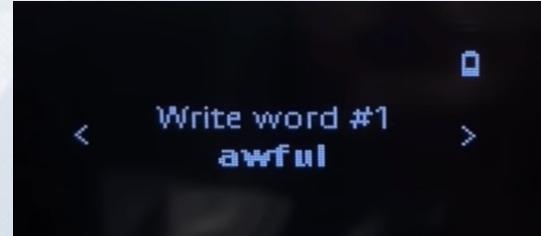
k_0 is stored on HW device and in offline storage (as 24 words)

\Rightarrow in case of loss, buy new device, restore k_0 , recompute keys

On Ledger

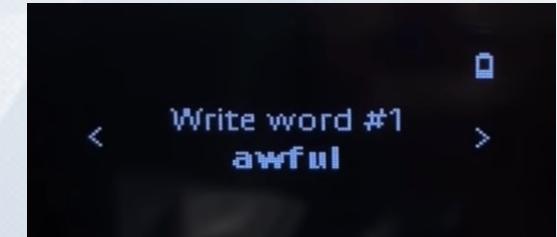
When initializing ledger:

- user asked to write down the 24 words
- each word encodes 11 bits ($24 \times 11 = 268$ bits)
 - list of 2048 words in different languages (BIP 39)



Example: English word list

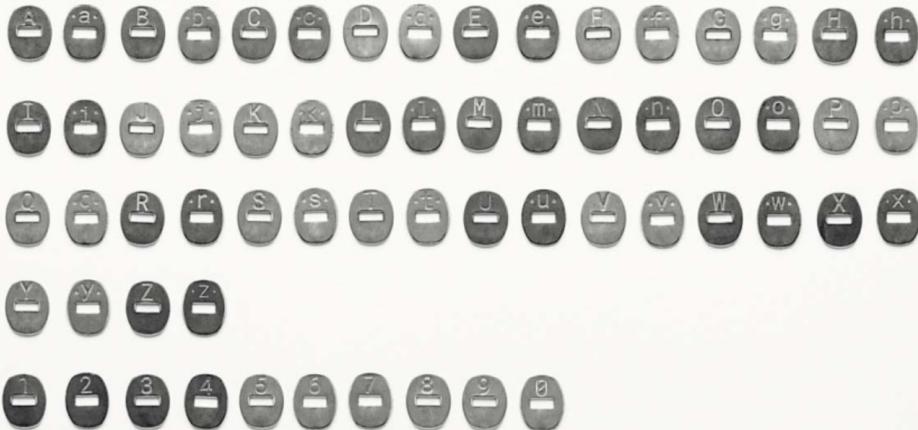
2048 lines (2048 sloc) 12.8 KB	
1	abandon
2	ability
3	able
4	about
5	above
6	absent
7	absorb
8	abstract
9	absurd
10	abuse
:	
2046	zero
2047	zone
2048	zoo



save list of
24 words



Crypto Steel

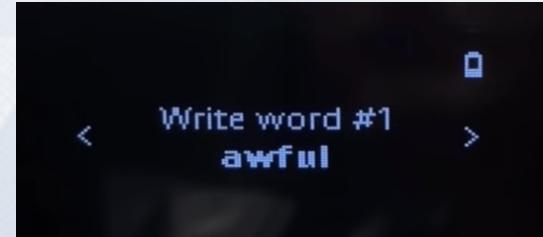


Careful with unused letters ...

On Ledger

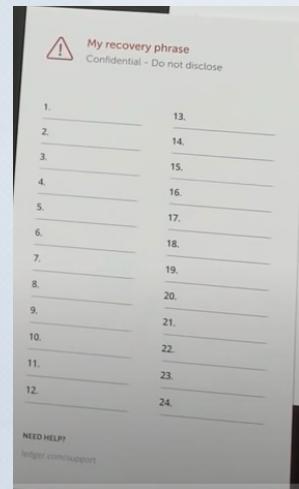
When initializing ledger:

- user asked to write down the 24 words
- each word encodes 11 bits ($24 \times 11 = 268$ bits)
 - list of 2048 words in different languages (BIP 39)



Beware of “pre-initialized HW wallet”

- 2018: funds transferred to wallet promptly stolen



How to securely check balances?

With Idea1: need k_0 just to check my balance:

- k_0 needed to generate my addresses (pk_1, pk_2, pk_3, \dots)
... but k_0 can also be used to spend funds
- Can we check balances without the spending key ??

Goal: two seeds

- k_0 lives on Ledger: can generate all secret keys (and addresses)
- k_{pub} : lives on laptop/phone wallet: can only generate addresses
(for checking balance)

Idea 2: (used in HD wallets)

secret seed: $k_0 \in \{0,1\}^{256}$; $(k_1, k_2) \leftarrow \text{HMAC}(k_0, \text{"init"})$

balance seed: $k_{\text{pub}} = (k_2, h = g^{k_1})$

for all $i=1,2,\dots$: $\begin{cases} sk_i \leftarrow k_1 + \text{HMAC}(k_2, i) \\ pk_i \leftarrow g^{sk_i} = g^{k_1} \cdot g^{\text{HMAC}(k_2, i)} = h \cdot g^{\text{HMAC}(k_2, i)} \end{cases}$

k_{pub} does not reveal sk_1, sk_2, \dots

computed from k_{pub}

k_{pub} : on laptop/phone, generates unlinkable addresses pk_1, pk_2, \dots

k_0 : on ledger

Paper wallet (be careful when generating)



Bitcoin address = $\text{base58}(\text{hash}(\text{PK}))$

signing key (cleartext)

base58 = a-zA-Z0-9 without {0,O,I,1}

Managing crypto assets in the cloud

How exchanges store assets



Hot/cold storage

Coinbase: holds customer assets

Design: 98% of assets (SK) are held in cold storage

cold storage (98%)

$$k_0^{(1)}$$

$$k_0^{(2)}$$

$$k_0^{(3)}$$



$$k_0$$

hot wallet (2%)

h, k_2
used to
verify cold
storage
balances

SK_{hot}
2% of
assets

←
customers
→

t-out-of-n secret sharing of k_0

Problems

Can't prove ownership of assets in cold storage,
without accessing cold storage:

- To prove ownership (e.g., in audit or in a proof of solvency)
- To participate in proof-of-stake consensus

Solutions:

- Keep everything in hot wallet (e.g, Anchorage)
- Proxy keys: keys that prove ownership of assets,
but cannot spend assets

END OF LECTURE

Next lecture: consensus

