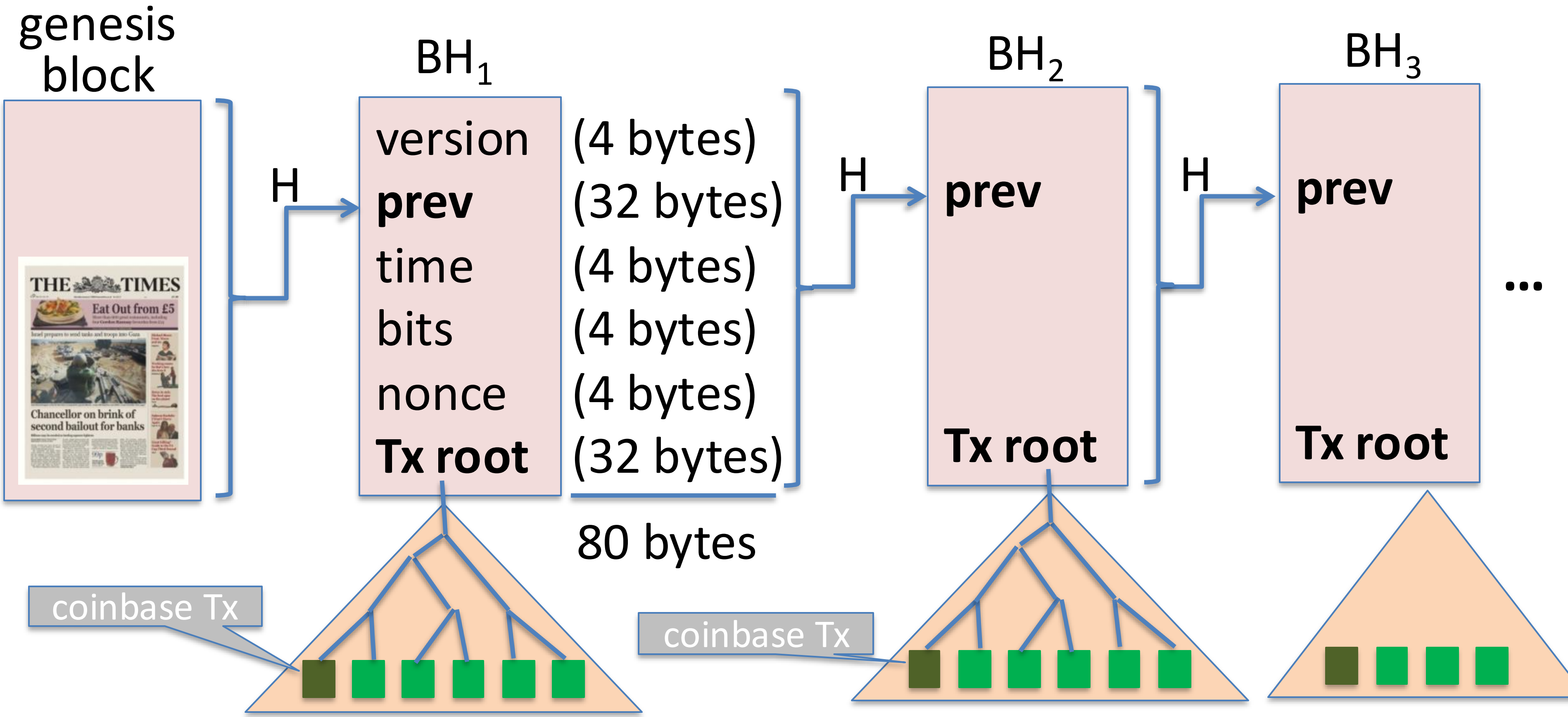
Introduction to Bitcoin

Deian Stefan

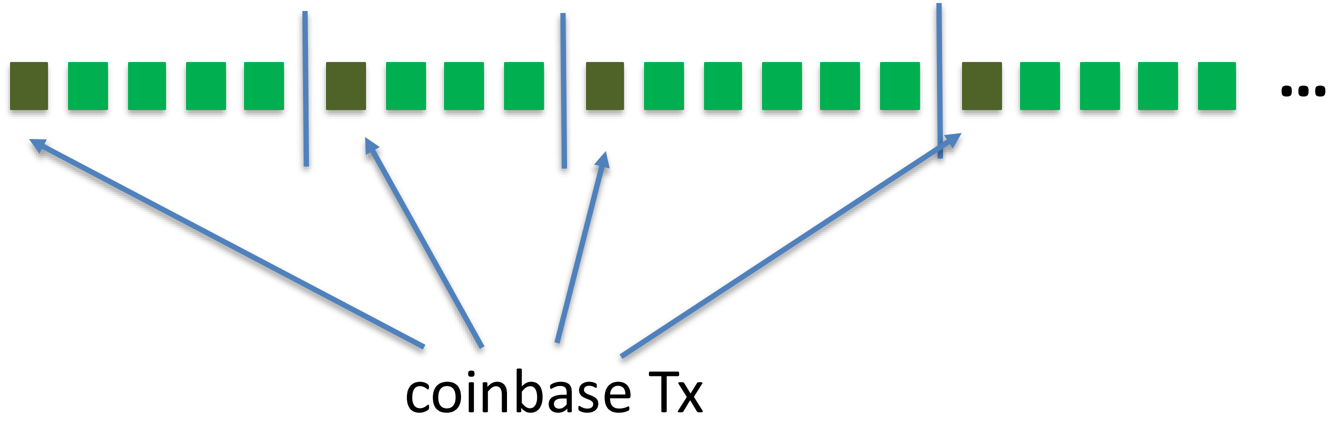
Slides from Dan Boneh

Recap: the Bitcoin blockchain



Tx sequence

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



Tx structure (non-coinbase)

inputs	input[0] input[1] input[2]
outputs	output[0] output[1]
(segwit)	witnesses (part of input)
(4 bytes)	locktime

TxID = H(Tx)
(excluding witnesses)

earliest block # that can include Tx

input:

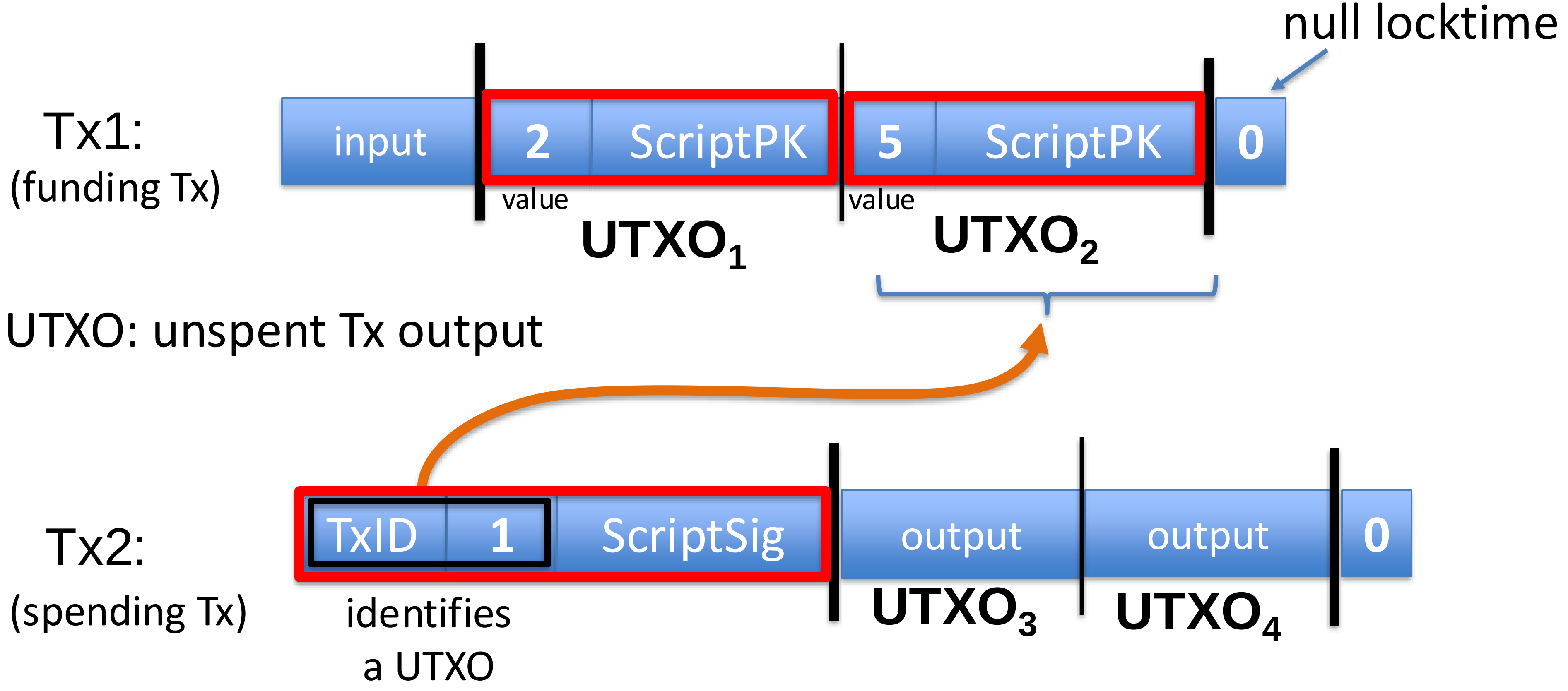
TxID	32 byte hash
out-index	4 byte index
ScriptSig	program
seq	ignore

output:

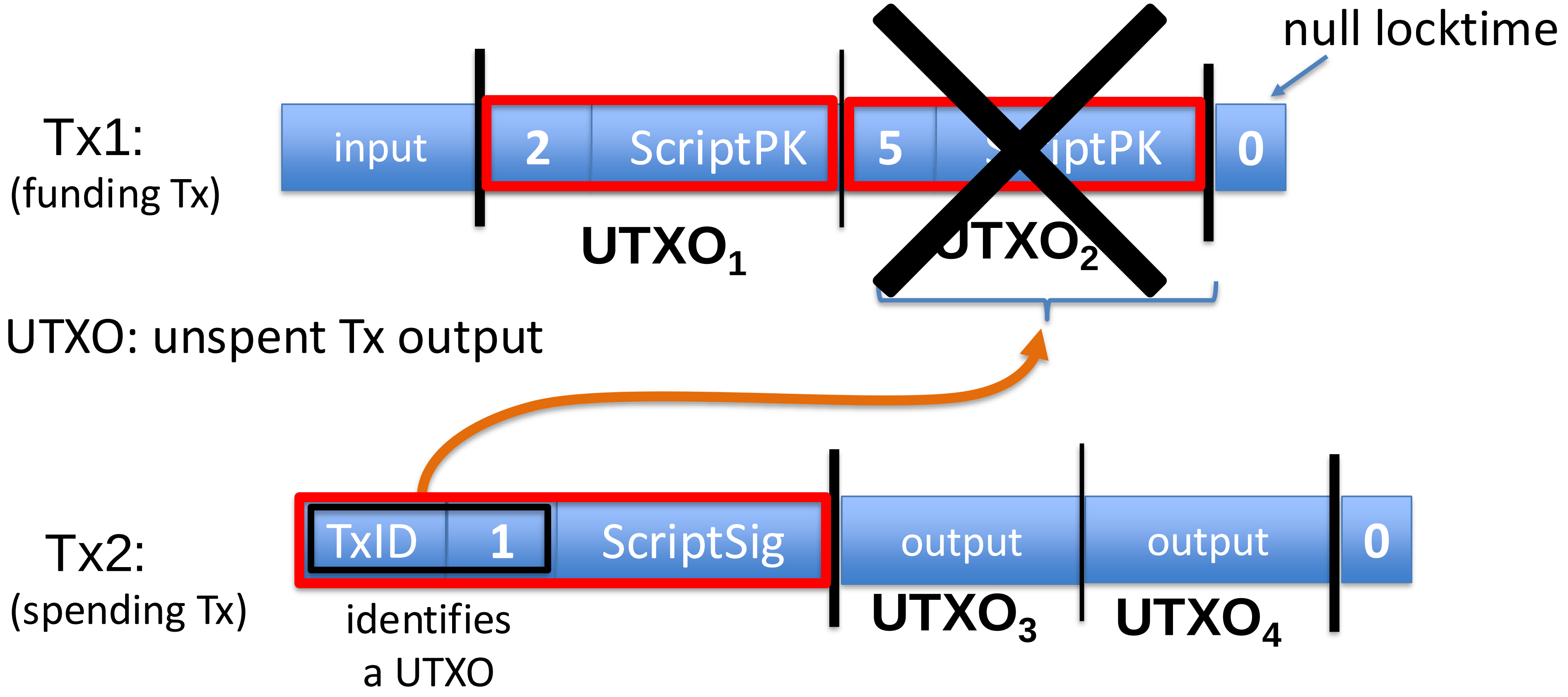
value	8 bytes
ScriptPK	program

#BTC = value/10^8

Example


null locktime
Tx1:
(funding Tx)
input
2
ScriptPK
5
ScriptPK
0
value
value
UTXO$_1$
UTXO$_2$
UTXO: unspent Tx output
Tx2:
(spending Tx)
TxID
1
ScriptSig
output
output
0
UTXO$_3$
UTXO$_4$
identifies
a UTXO

Example



Tx1:
(funding Tx)
input
2
ScriptPK
5
iptPK
0
null locktime
UTXO$_1$
TXO$_2$
UTXO: unspent Tx output
Tx2:
(spending Tx)
TxID
1
ScriptSig
output
output
0
identifies a UTXO
UTXO$_3$
UTXO$_4$

Validating Tx2

Miners check (for each input):

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

1. The program $\boxed{\text{ScriptSig | ScriptPK}}$ returns true
2. $\boxed{\text{TxID | index}}$ is in the current UTXO set

program from spending Tx:
proof that conditions
are met

3. sum input values $\geq$ sum output values

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

Today

Understand:

1. **How we spend coins (bitcoin scripts)**
    - **P2PKH**
    - **P2SH**
    - **more advanced scripts**
2. **How users interface with blockchains (wallets)**

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:

Point to Alice's UTXO

input 7 BTC	5	$ScriptPK_B$	2	$ScriptPK_A$	0

$UTXO_B$ for Bob

$UTXO_A$ for Alice (change)

$ScriptPK_B$: DUP HASH256 **<addr$_B$>** EQVERIFY CHECKSIG

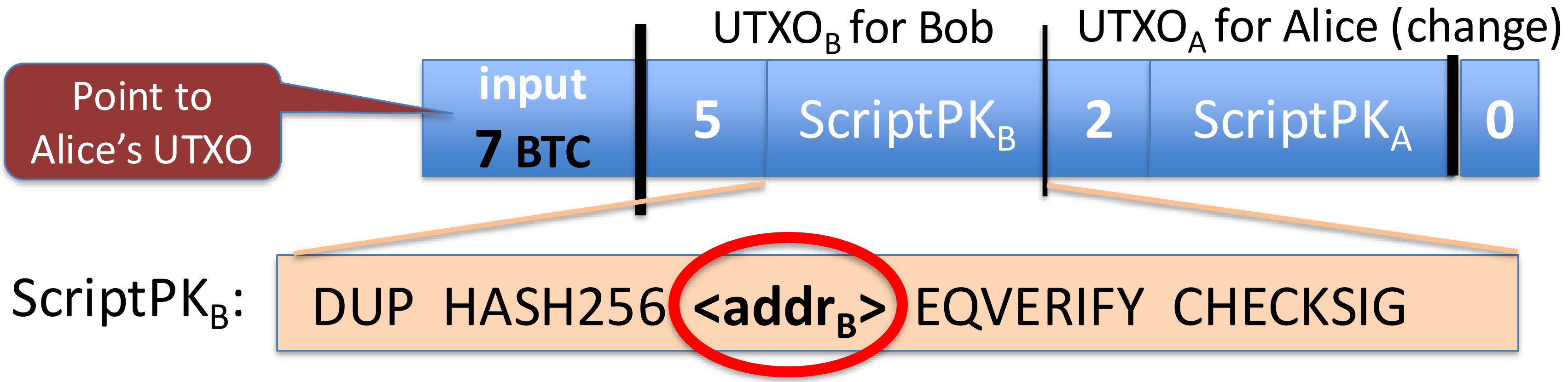
Transaction types: (1) P2PKH

pay to public key hash

"input" contains ScriptSig that authorizes spending Alice's UTXO

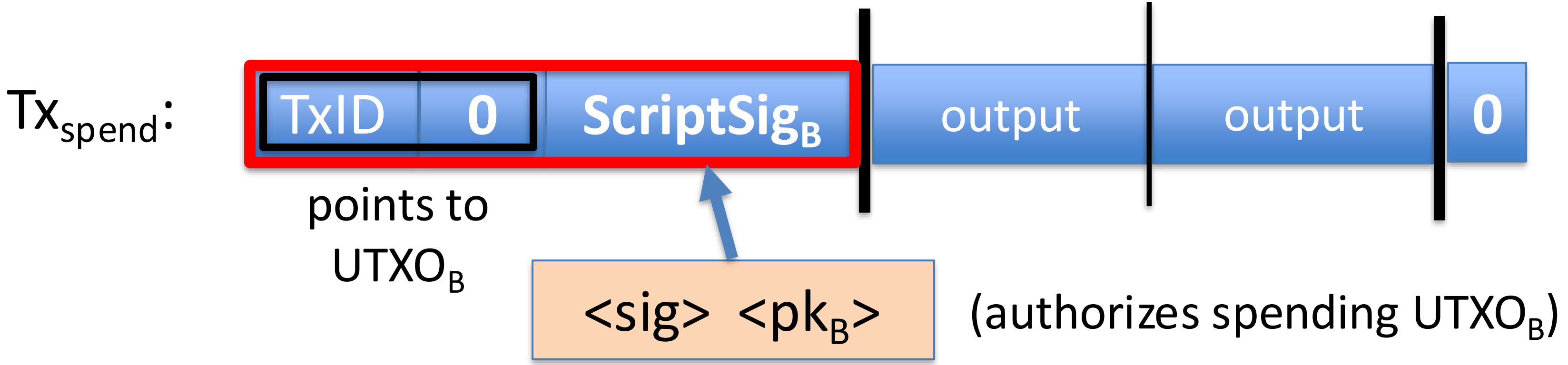
- example: ScriptSig contains Alice's signature on Tx

  $\Longrightarrow$ 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}



<sig> = Sign(sk_B, Tx) where Tx = (Tx_{spend} excluding all ScriptSigs) (SIGHASH_ALL)

Miners validate that $ScriptSig_B$ | $ScriptPK_B$ 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 <$Addr_B$> and steal funds:

  invalidates Alice's signature that created $UTXO_B$

Segregated Witness

ECDSA malleability:

Given (m, sig) anyone can create (m, sig’) with sig ≠ 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)

We’ve actually been looking at P2WPKH

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

Let's payer specify a redeem script (instead of just pkhash)

Usage: payee publishes hash(redeem script) ⟵ Bitcoint addr.

payer sends funds to that address

ScriptPK in UTXO:

ScriptSig to spend:

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

Let's payer specify a redeem script (instead of just pkhash)

Usage: payee publishes hash(redeem script) ⟵ Bitcoint addr.

payer sends funds to that address

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

ScriptSig to spend: $<sig_1> <sig_2> \ldots <sig_n>$ <redeem script>

payer can specify complex conditions for when UTXO can be spent

P2SH

Miner verifies:

(1) <ScriptSig> ScriptPK = true ⟵ payee gave correct script

(2) ScriptSig = true ⟵ script is satisfied

Example P2SH: multisig

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

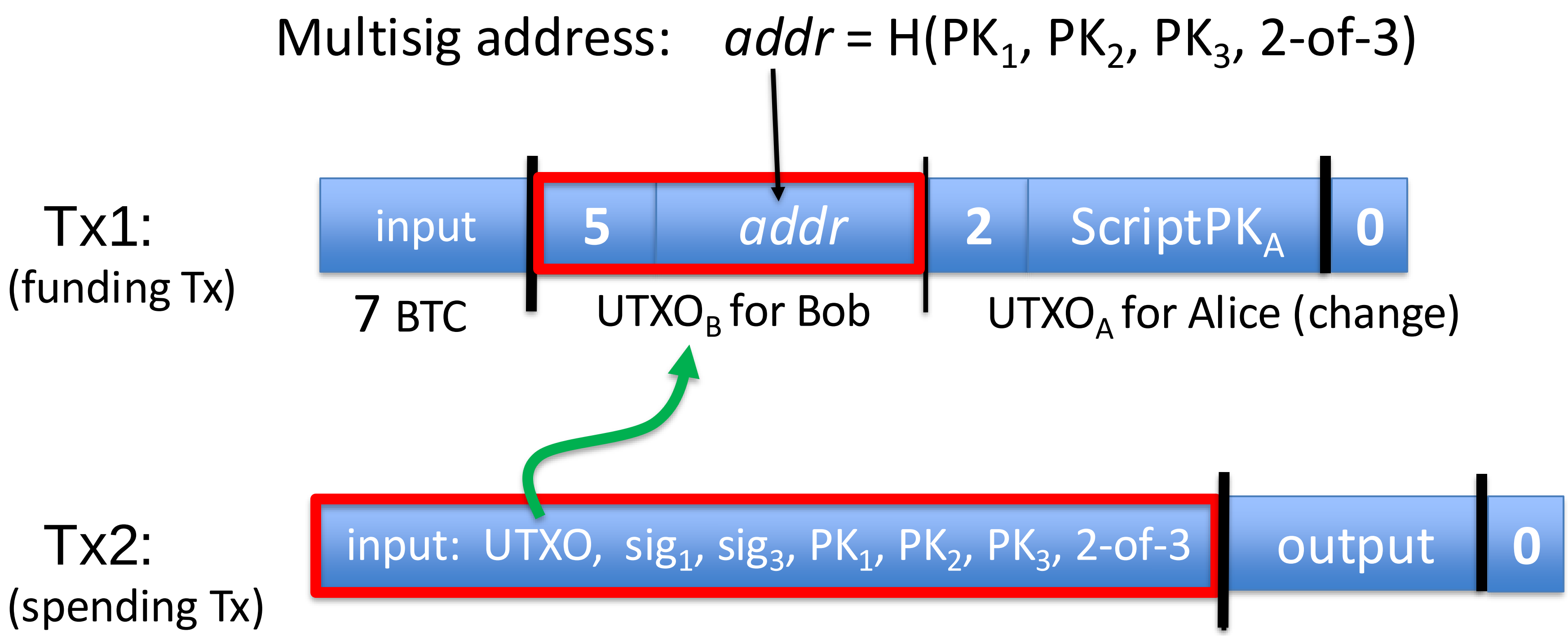
Redeem script for 2-out-of-3: (set by payer)

<2> <PK_1> <PK_2> <PK_3> <3> CHECKMULTISIG

hash gives P2SH address

ScriptSig to spend: (by payee) <0> <sig1> <sig3> <redeem script>

Abstractly …



Multisig address: *addr* = H(PK_1, PK_2, PK_3, 2-of-3)
Tx1:
(funding Tx)
input
5
addr
2
$ScriptPK_A$
0
7 BTC
$UTXO_B$ for Bob
$UTXO_A$ for Alice (change)
Tx2:
(spending Tx)
input: UTXO, sig_1, sig_3, PK_1, PK_2, PK_3, 2-of-3
output
0

Transaction types: (3) P2TR: pay to Taproot

Let’s payer specify complex spending conditions:

A. Public key (similar to P2WPKH) but using Schnorr signatures

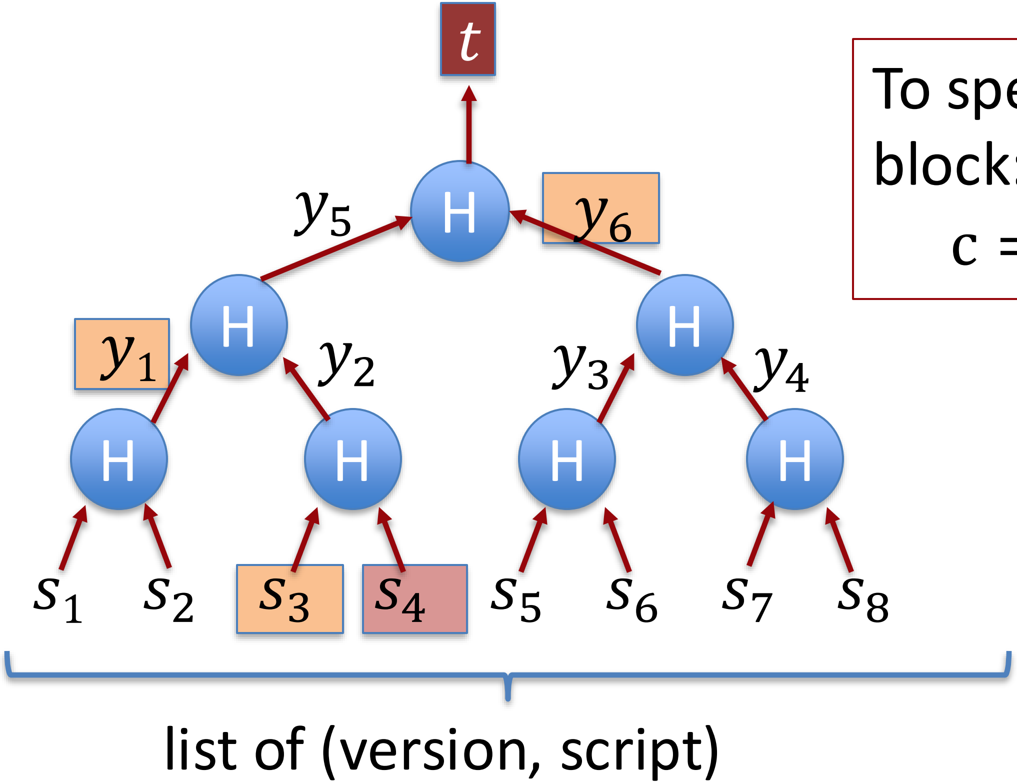
 - Aggregate and threshold signing is easy

B. Some script (similar to P2SH)

 - Can have many script spending conditions

Idea: Can’t distinguish between A or B + don’t need to reveal script until you spend.

Transaction types: (3) P2TR: pay to Taproot



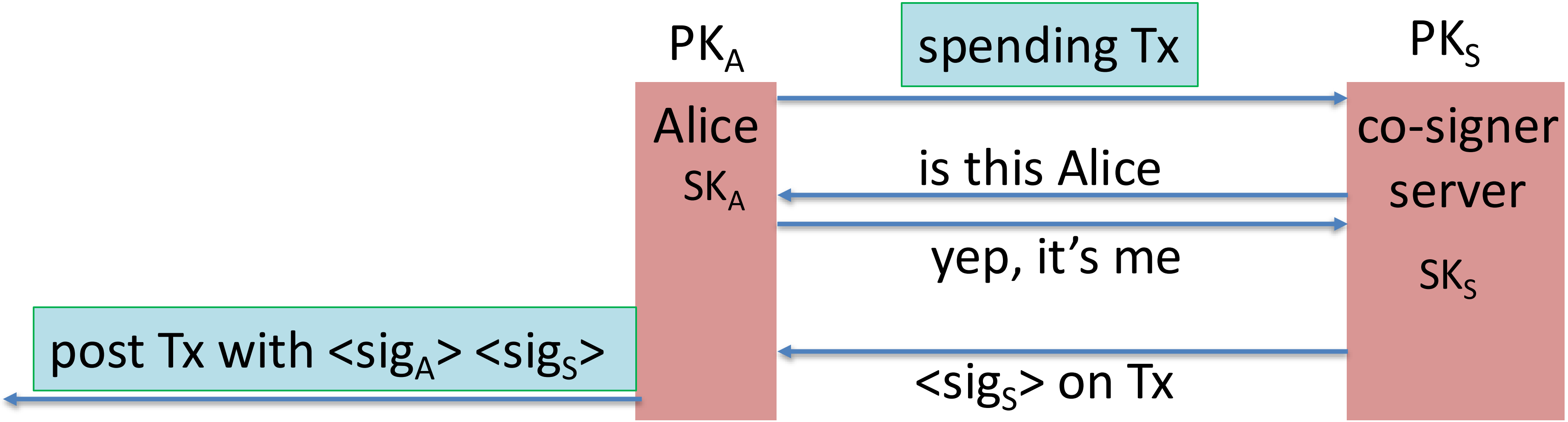
To spend with s_4, provide control block:

$$c = (internal\ pk, s_4, y_1, y_6)$$

Using Bitcoin scripts

Protecting assets with a co-signer

Alice stores her funds in UTXOs for $\boxed{\textit{addr} = \textbf{2-of-2}(\mathbf{PK_A}, \mathbf{PK_S})}$



$\Rightarrow$ theft of Alice's SK_A does not compromise BTC

Escrow service

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

<u>**Goal**</u>: Alice only pays after backpack arrives, but can't not pay

addr = **2-of-3(PK_A, PK_B, PK_J)**

post payment of 0.11Ƀ to *addr*

(creates $UTXO_{ABJ}$)

Alice

PK_A

want backpack for 0.1Ƀ

once see Tx on chain

mail backpack

backpack arrives

send <sig_A> on Tx:

$UTXO_{ABJ} \rightarrow (PK_B{:}0.1,\ PK_A{:}0.01)$

Bob

PK_B

redeem using <sig_A> <sig_B> on Tx

Judge

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: $(\ \mathbf{UTXO_{ABJ}} \rightarrow \mathbf{PK_A}\ ,\ \ \mathbf{sig_A},\ \mathbf{sig_{Judge}}\ )$ [2-out-of-3]

---

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

Bob gets paid with help of Judge and a Tx:

Tx: $(\ \mathbf{UTXO_{ABJ}} \rightarrow \mathbf{PK_B}\ ,\ \ \mathbf{sig_B},\ \mathbf{sig_{Judge}}\ )$ [2-out-of-3]

---

(3) Both are at fault: Judge publishes $<sig_{Judge}>$ on Tx:

Tx: $(\ \mathbf{UTXO_{ABJ}} \rightarrow \mathbf{PK_A: 0.05,\ \ PK_B: 0.05,\ \ PK_J: 0.01}\ )$

Now either Alice or Bob can execute this Tx.

Cross Chain Atomic Swap

Alice has 5 BTC,   Bob has 2 LTC (LiteCoin).   They want to swap.

Want a sequence of Tx on the Bitcoin and Litecoin chains s.t.:

- either success: Alice has 2 LTC and Bob has 5 BTX,
- or failure: no funds move.

Swap cannot get stuck halfway.

<u>Goal</u>: design a sequence of Tx to do this.

solution: programming proj #1 ex 4.

How do users sign anything?

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:

- **custodial cloud wallet** (e.g., Coinbase): like a bank
- **non-custodial cloud wallet** (e.g., Core): … only you access keys
- **non-custodial self hosted** (e.g., MetaMask): keys in browser extension/app
- **consumer hardware wallet** (e.g., Ledger): keys in hw you hold
- **paper**: print all sk on paper
- … lots of things in between (lots of bad ideas)

Simplified Payment Verification (SPV)

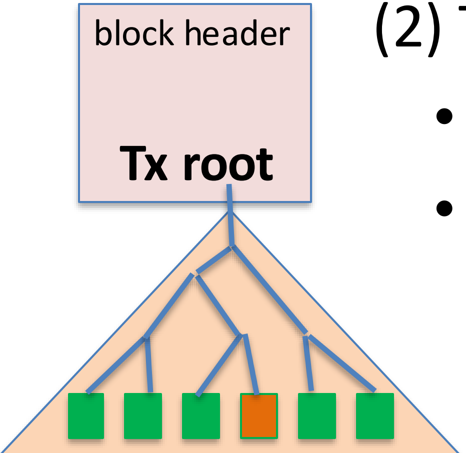
How does a client wallet display Alice's current balances?

- Wallet needs to verify an incoming payment
- **<u>Goal</u>**: do so w/o downloading entire blockchain (500+ GB)

SPV: (1) download all block headers (<100 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?

- Example: download block headers from ten random servers, optionally, also from a trusted full node.

(2) **Privacy**: remote server can test if an *addr* belongs to wallet

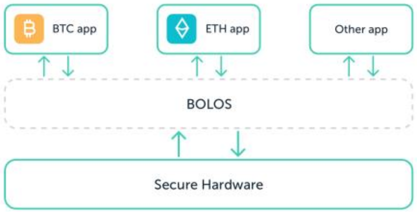
We will see better light client designs later in the course (e.g. Celo)

Hardware wallet: Ledger, Trezor, …

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

Hardware wallet (e.g., Ledger Nano S Plus)

- 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
- screen and buttons to verify and confirm Tx



Hardware wallet: backup

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

ECDSA public key

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

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

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

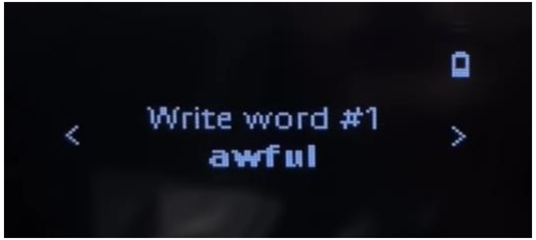
k_0 is stored on HW device and in offline storage (as 24 words)

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

Seed phrases

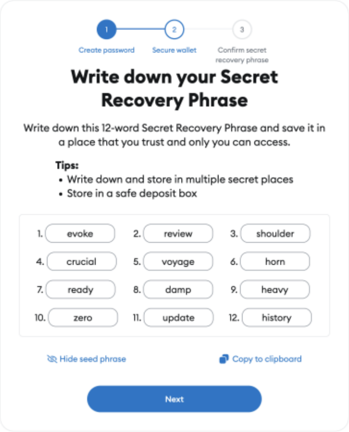
When initializing Ledger:

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

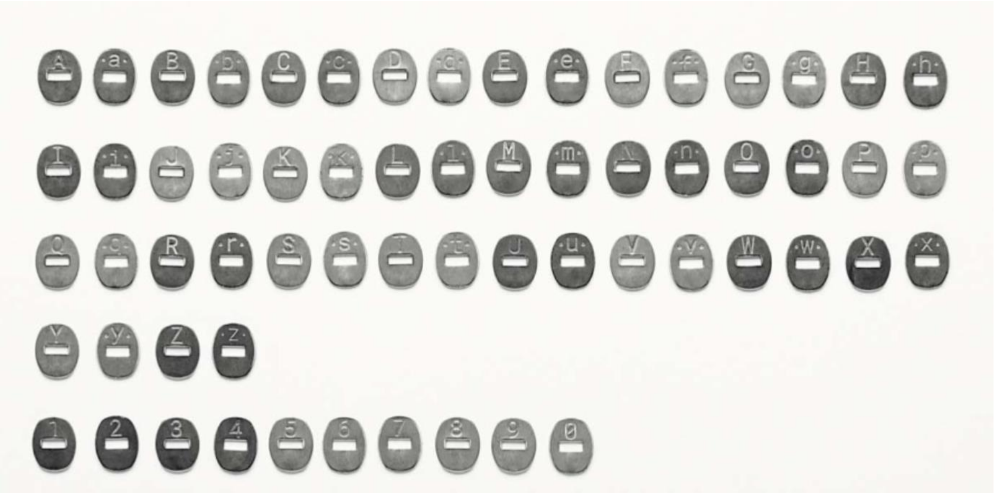


Initializing Metamask is similar:

- user asked to write down 12 words
- user confirms some of those words



Paper is not great



Careful with unused letters …

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

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

for all i=1,2,…: $\begin{cases} sk_i \leftarrow k_1 + \mathrm{HMAC}(k_2, i) \\ pk_i \leftarrow g^{sk_i} = g^{k_1} \cdot g^{HMAC(k_2,i)} = \underbrace{h \cdot g^{HMAC(k_2,i)}}_{\text{computed from } k_{pub}} \end{cases}$

$\boxed{k_{pub} \text{ does not reveal } sk_1, sk_2, \ldots}$

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

k_0: on ledger