SNICKER

SIMPLE NON-INTERACTIVE COINJOIN WITH KEYS FOR ENCRYPTION REUSED

SNICKER (BIP Draft, 2020)
Adam Gibson (u/waxwing)

SNICKER_BIP_draft.mediawiki

Raw

```
BIP: ??
Layer: Applications
Title: SNICKER - Simple Non-Interactive Coinjoin with Keys for Encryption Reused
Author: Adam Gibson <AdamISZ@protonmail.com>
Comments-Summary: No comments yet.
Comments-URI: -
Status: Proposed
Type: Informational
Created: -
License: BSD-2-Clause
```

https://gist.github.com/AdamISZ/2c13fb5819bd469ca318156e2cf25d79

Last week – Knapsack Coinjoin

Summary of Knapsack CoinJoin idea:

We can allow for users to mix **arbitrary values** if we further **break down the outputs** in a tactical way. By considering the ways that a CoinJoined transaction can be broken down, we can optimize the break down of outputs by **increasing the number possible ways a CoinJoined transaction can be interpreted** by an outsider. Anonymity of coins and their origins is achieved by having a multiplicity of possible links between any two coins in this scheme.

Wasabi Research Club

- January 6th, 2020 – Knapsack CoinJoin
- January 13th, 2020 – SNICKER
- January 20th, 2020 - TBD

https://github.com/zkSNACKs/WasabiResearchClub

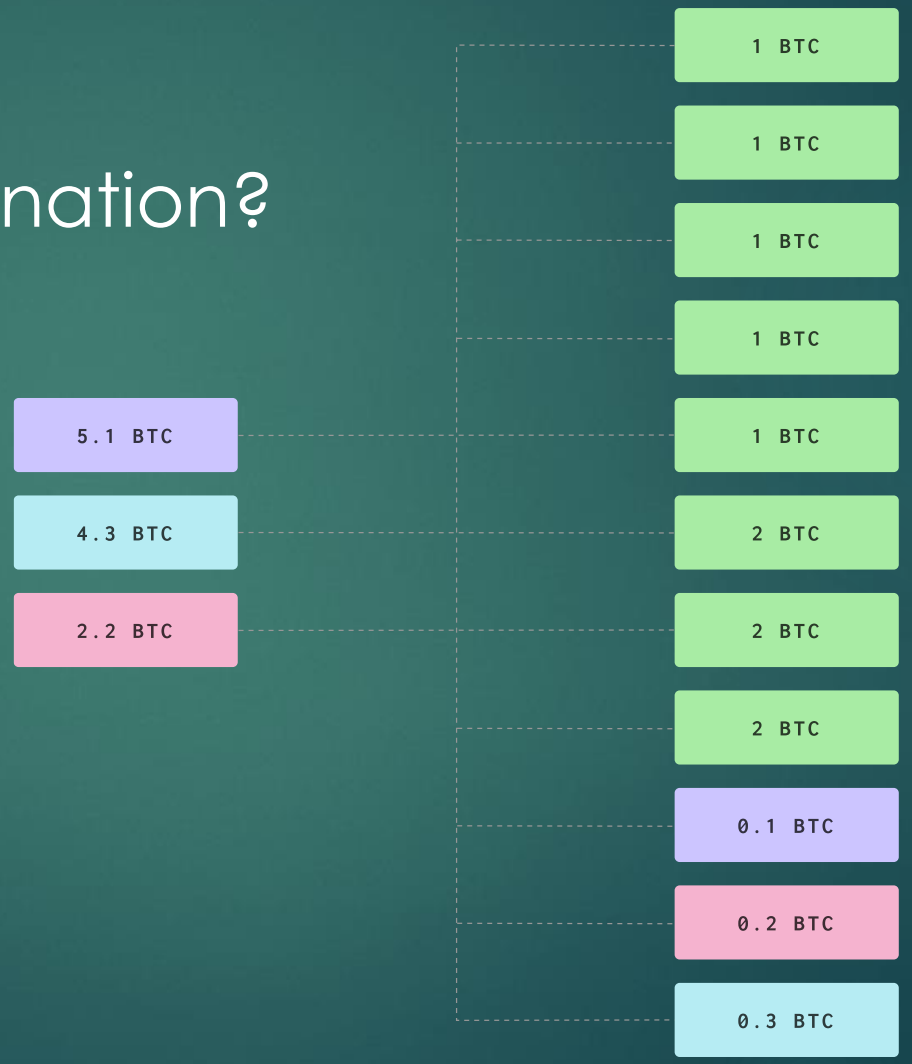
Problem with current CoinJoins

- Interactive (requires a server)
  - Could reduce the privacy of users
  - Difficult to coordinate many participants
  - Fragile to attack (central point of failure)

Could a CoinJoin be done *without* Coordination?

What requires coordination?

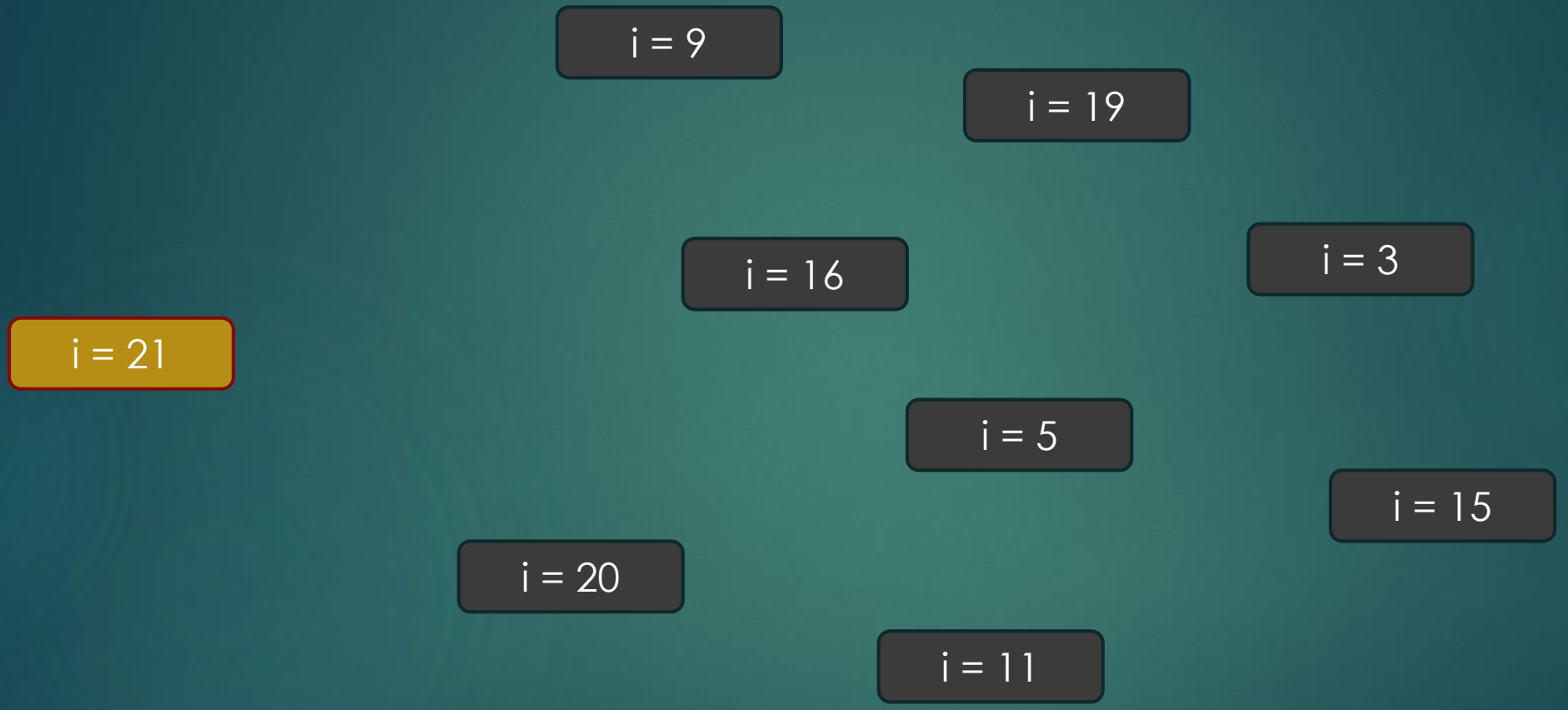
- Inputs
- Outputs
- Signatures



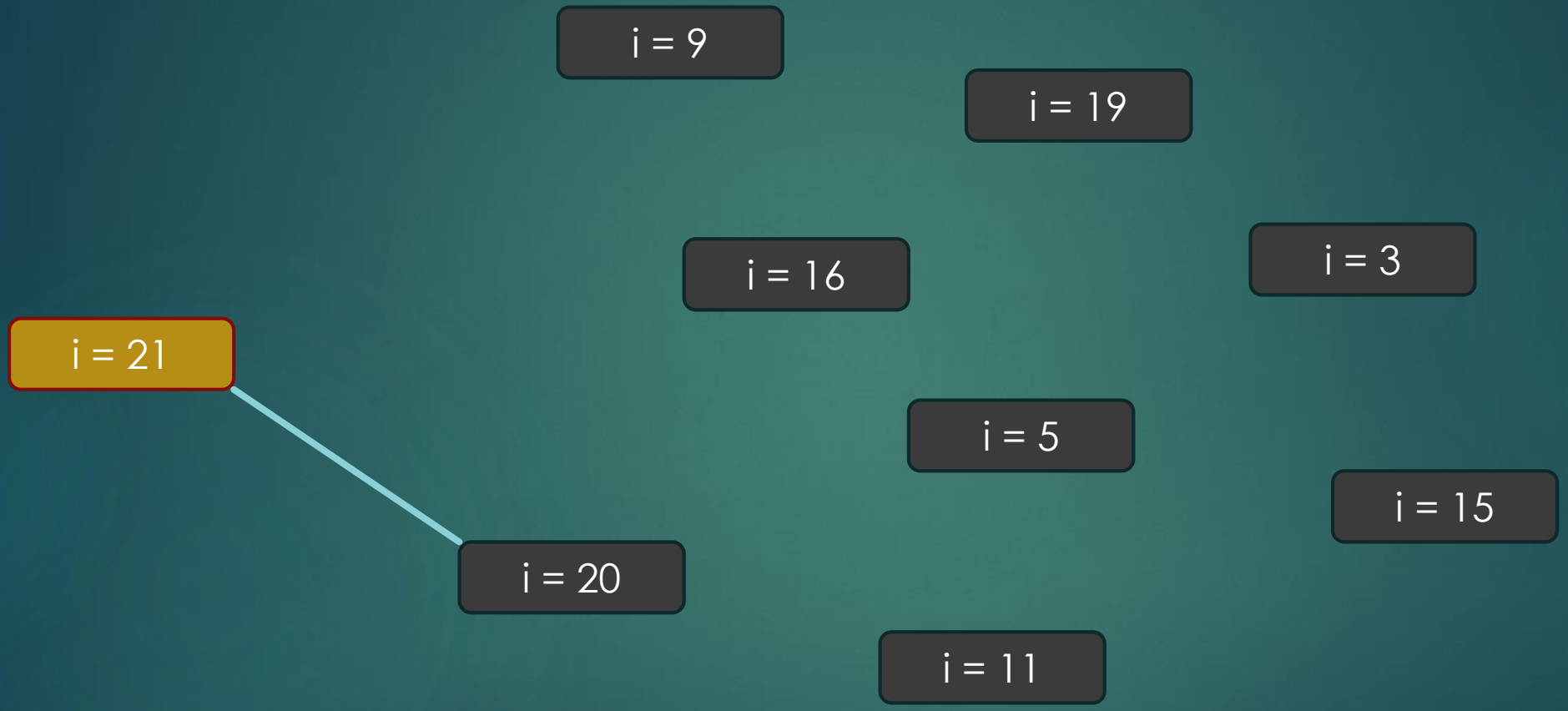
Part 1 – Inputs

- A **proposer** knows their own input, but they don't know the input of a potential CoinJoin buddy.
- So the proposer simply guesses which UTXOs on the blockchain might be the most likely to be interested.
  - Perhaps UTXOs involved in Joinmarket type transactions
  - Or UTXOs at random that are recent
  - Later UTXOs from previous SNICKER Coinjoins

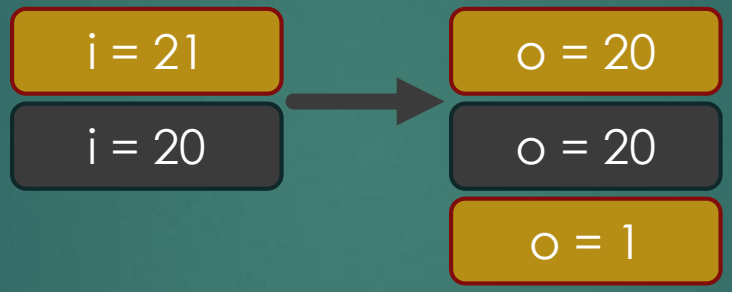
Part 1 – Inputs



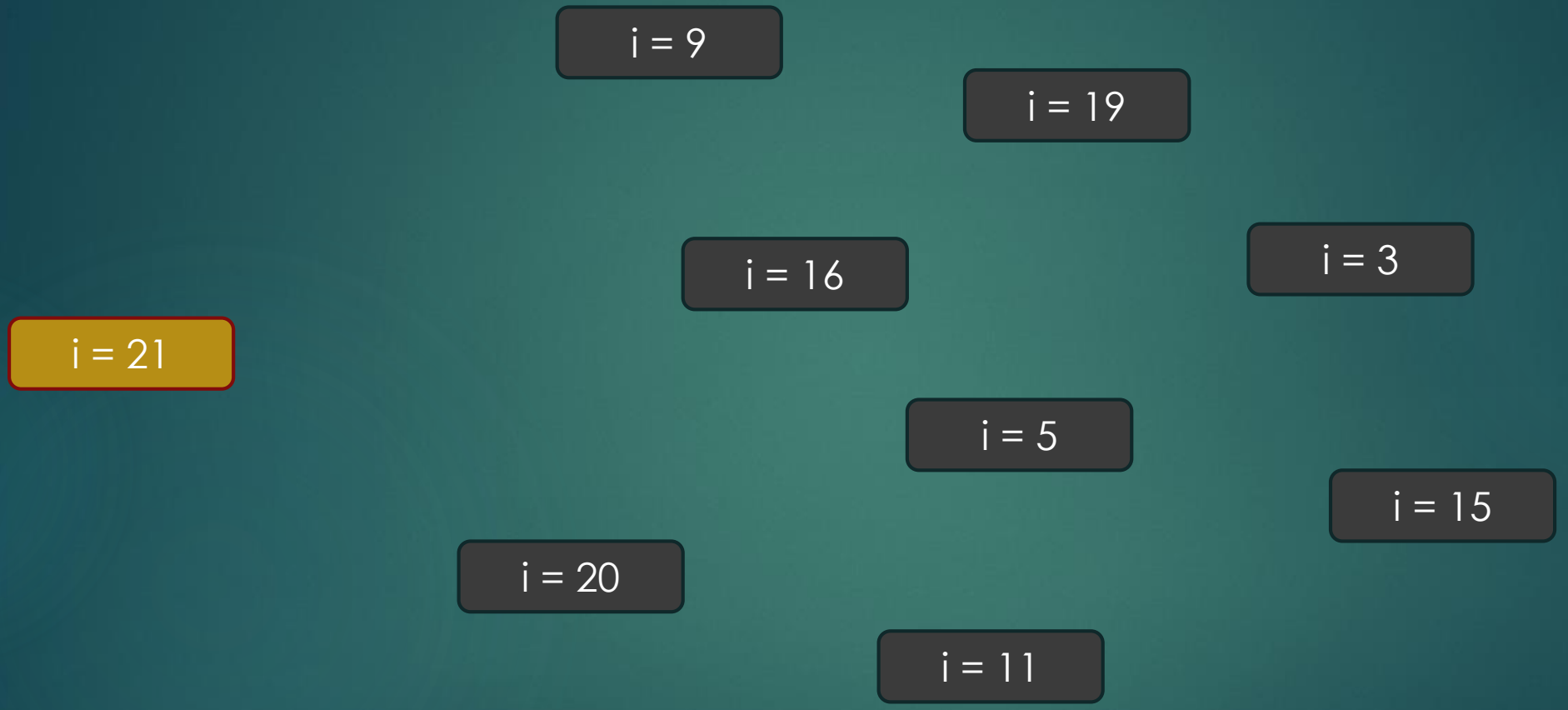
Part 1 – Inputs



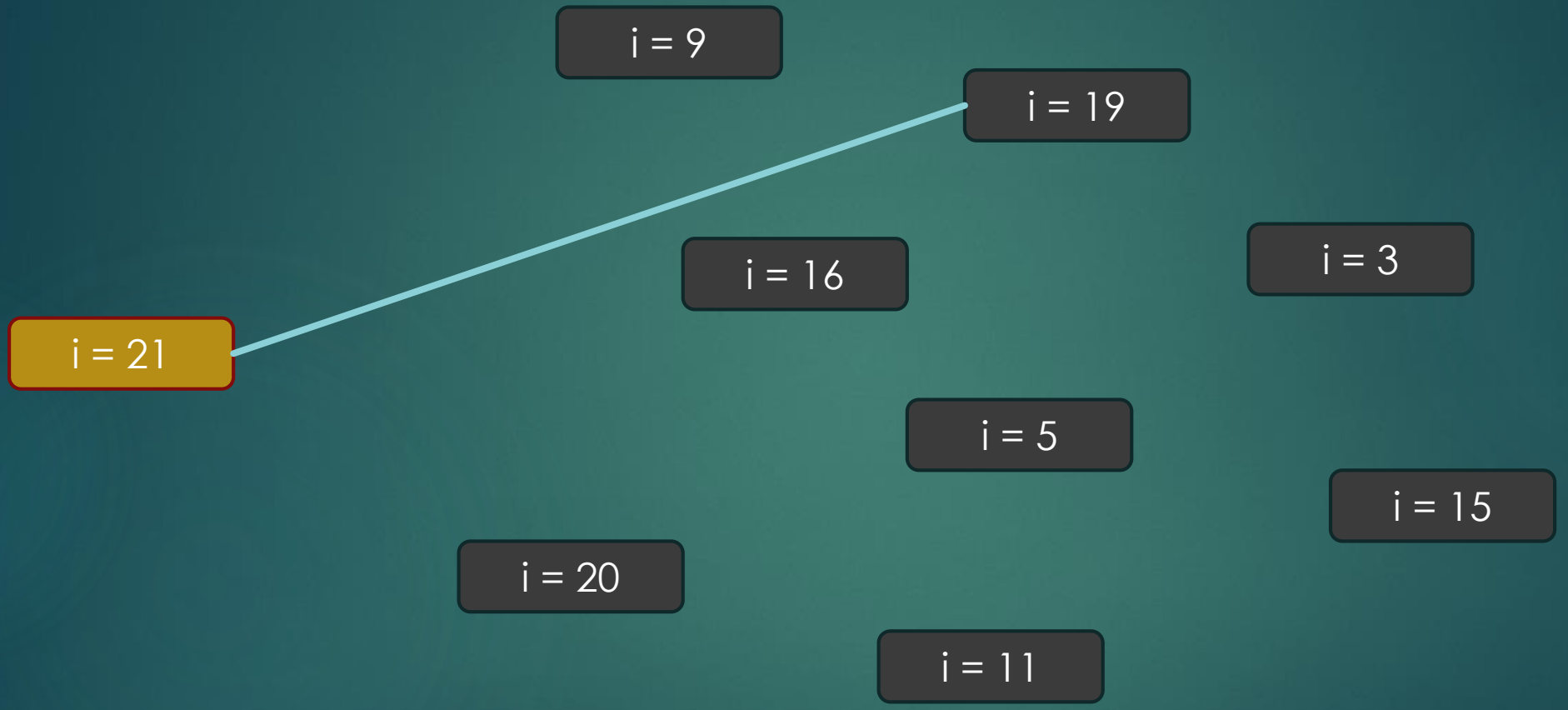
Part 1 – Inputs



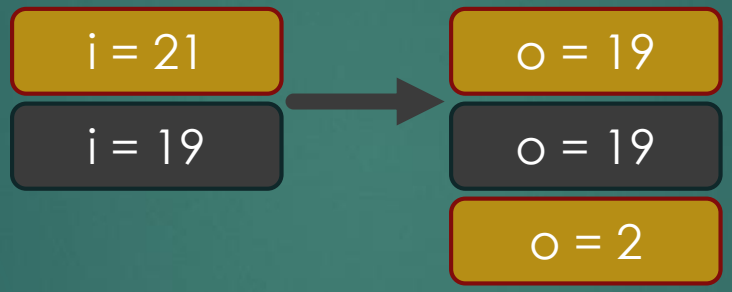
Part 1 – Inputs



Part 1 – Inputs



Part 1 – Inputs

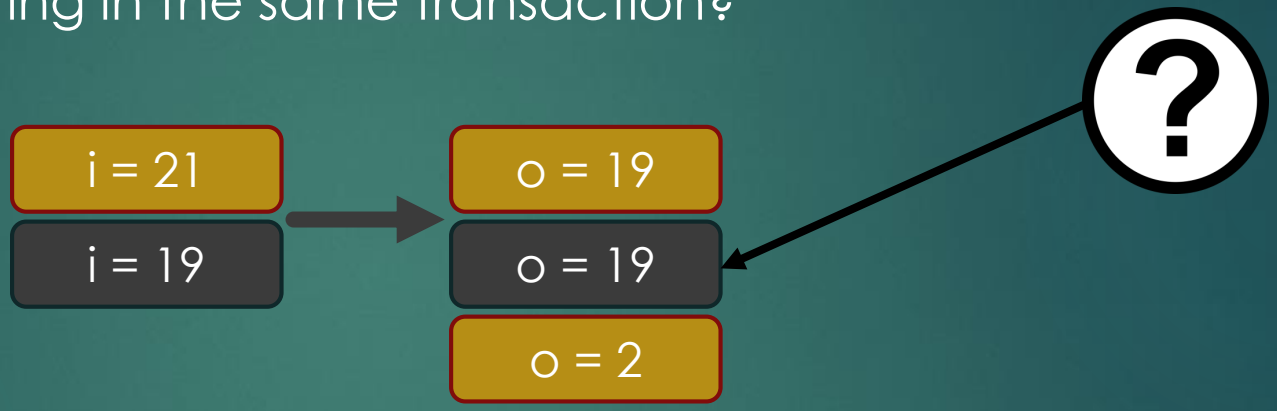


Part 2 – Outputs

- A **proposer** knows their own address, and they know their own **unused** fresh addresses.
- But how will a proposer figure out a fresh address that belongs to the other party that is participating in the same transaction?
  - They aren't communicating!

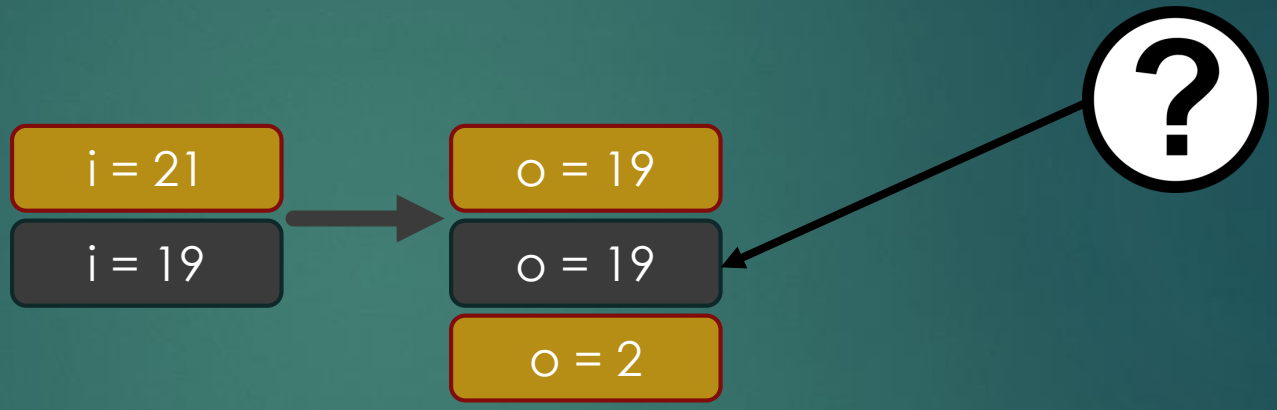
Part 2 – Outputs

How will a proposer figure out a fresh address that belongs to the other party that is participating in the same transaction?



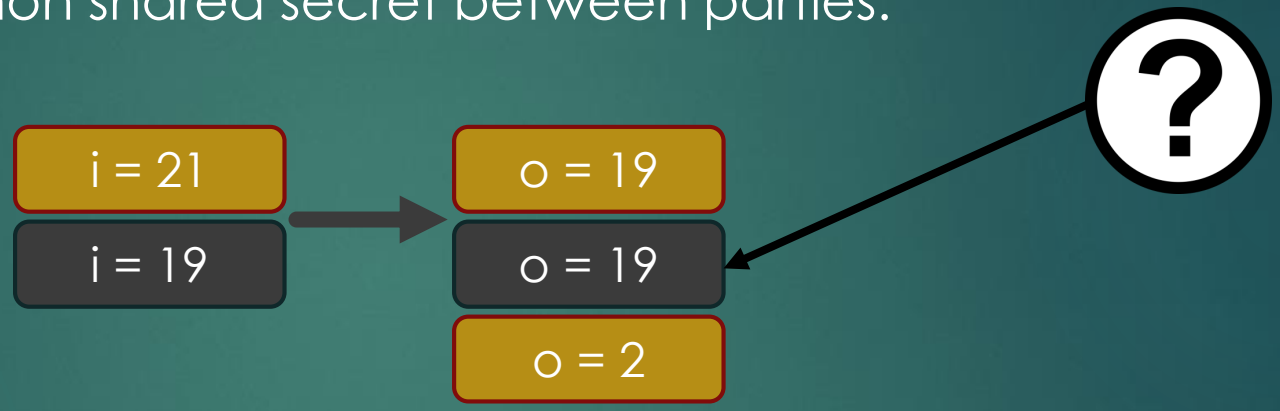
Part 2 – Outputs

Naïve Solution – Just copy the same address used by that UTXO!



Part 2 – Outputs

Smart Solution – Use an address that is tweaked from the public key of the UTXO, by a common shared secret between parties.



Part 2 – Outputs

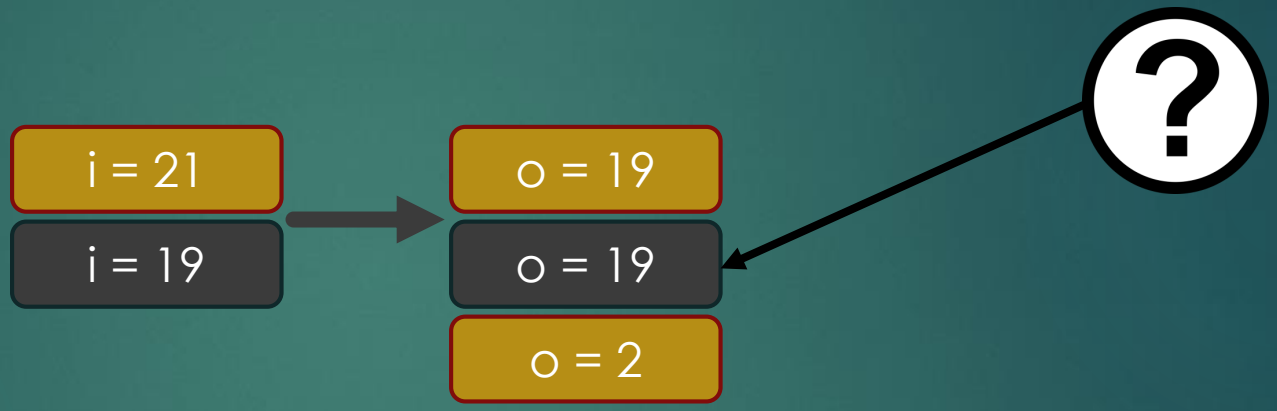
Smart Solution – Use an address that is tweaked from the public key of the UTXO, by a common shared secret between parties.

- All you know about the mystery UTXO is that it possesses the private key to the public key that holds the funds.
- But the **public key is not explicitly written** in the UTXO! Only the hash of the public key is offered.
- Unless… the individual with the UTXO has **spent from that address in the past** – in which case the signature would reveal the public key.
- Thus the proposer must only consider UTXOs which belong to addresses that have been re-used at least once (SNICKER V. 0)

Part 2 – Outputs

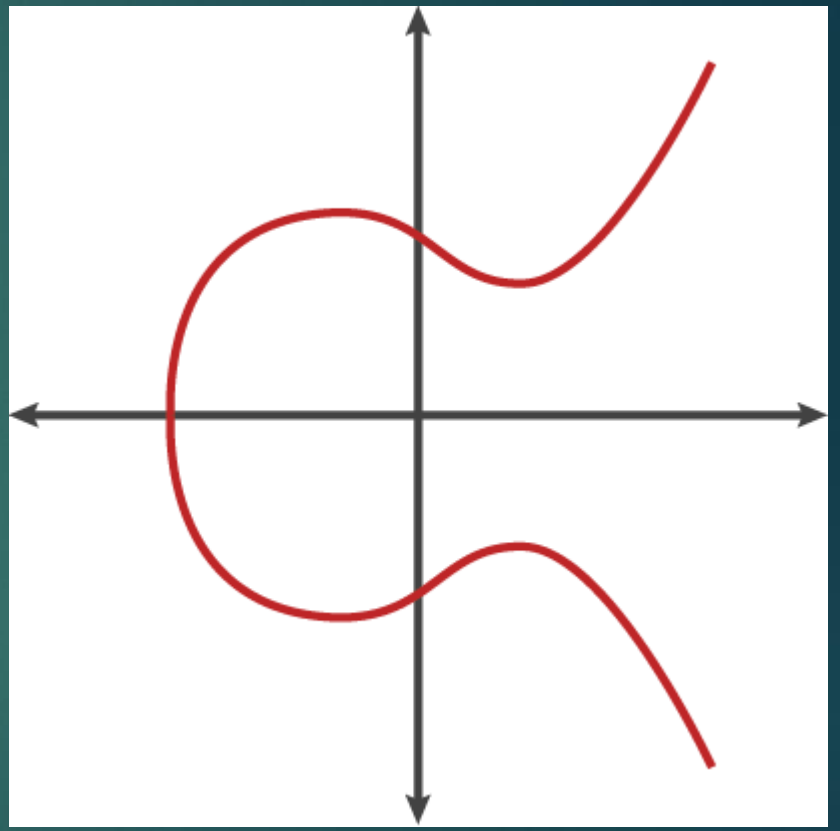
But how will we teak the public key of the receiver?

With a shared secret!



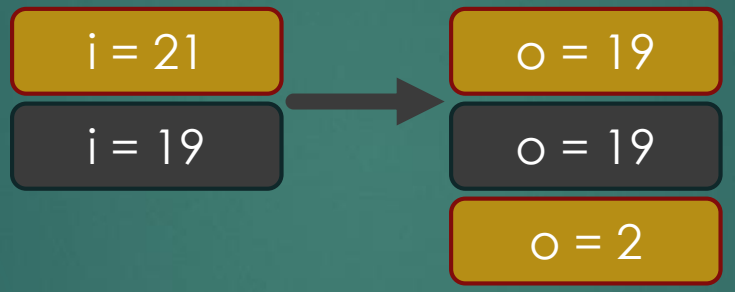
Elliptic Curve Primitives, Diffie Hellman

- $p \cdot G = P$
- $k \cdot G = K$
- $p \cdot K = p \cdot (k \cdot G) = k \cdot (p \cdot G) = k \cdot P$
- $p \cdot K = S$
- $k \cdot P = S$
- Diffie Hellman Key Exchange (S)



Part 2 – Outputs

Let's take the proposer and receiver public key and use it to create a shared secret (c), then let's shift the receiver's public key by c to create a fresh address!



$$(k + c) \cdot G = c \cdot K = K''$$

Part 2 – Outputs

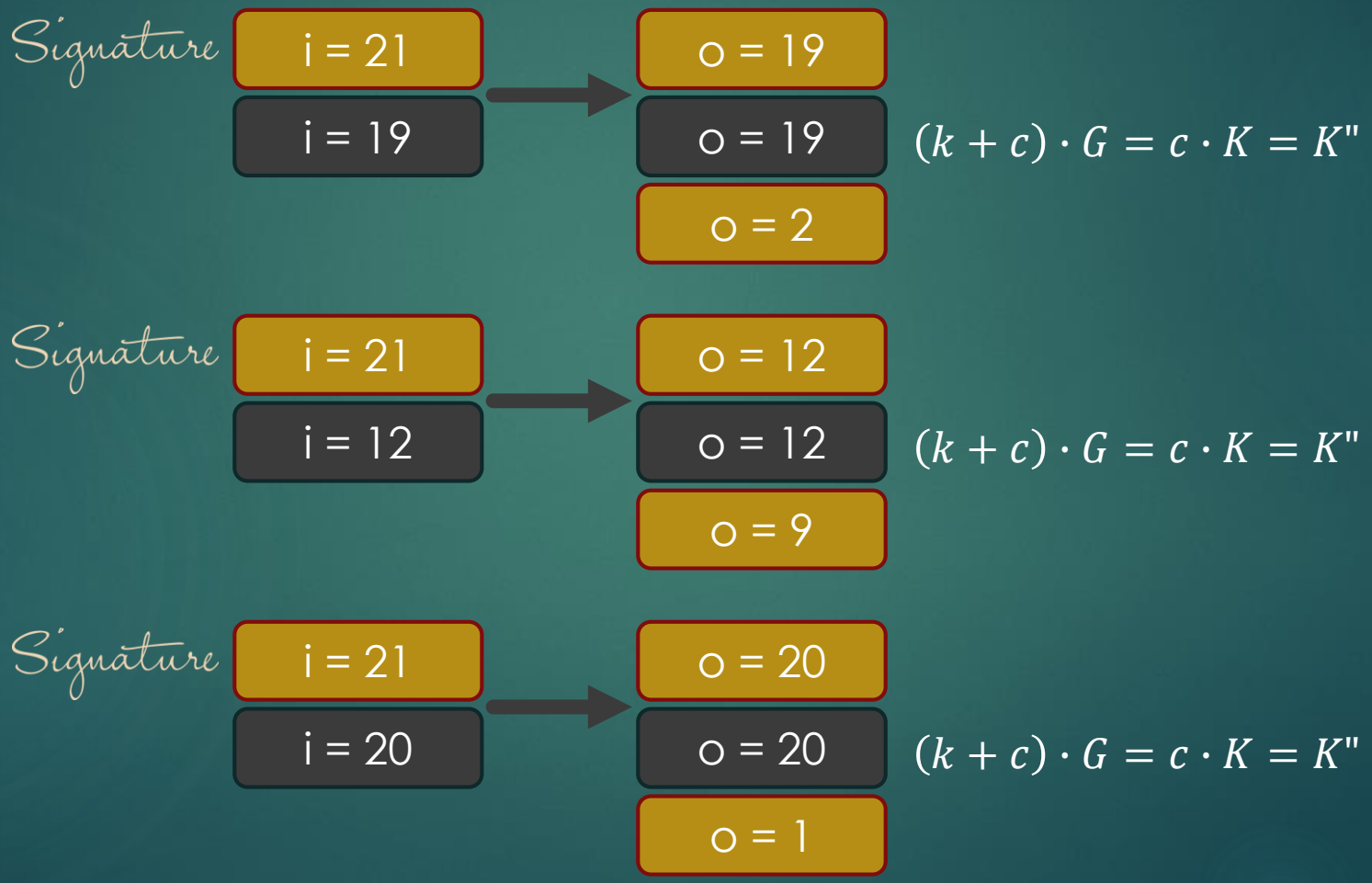
So far

1. Selected a bunch of candidate UTXOs
    1. Re-used addresses
    2. Amount smaller than ours
2. Create recipient address with DH tweak (c)
3. But what about signatures?

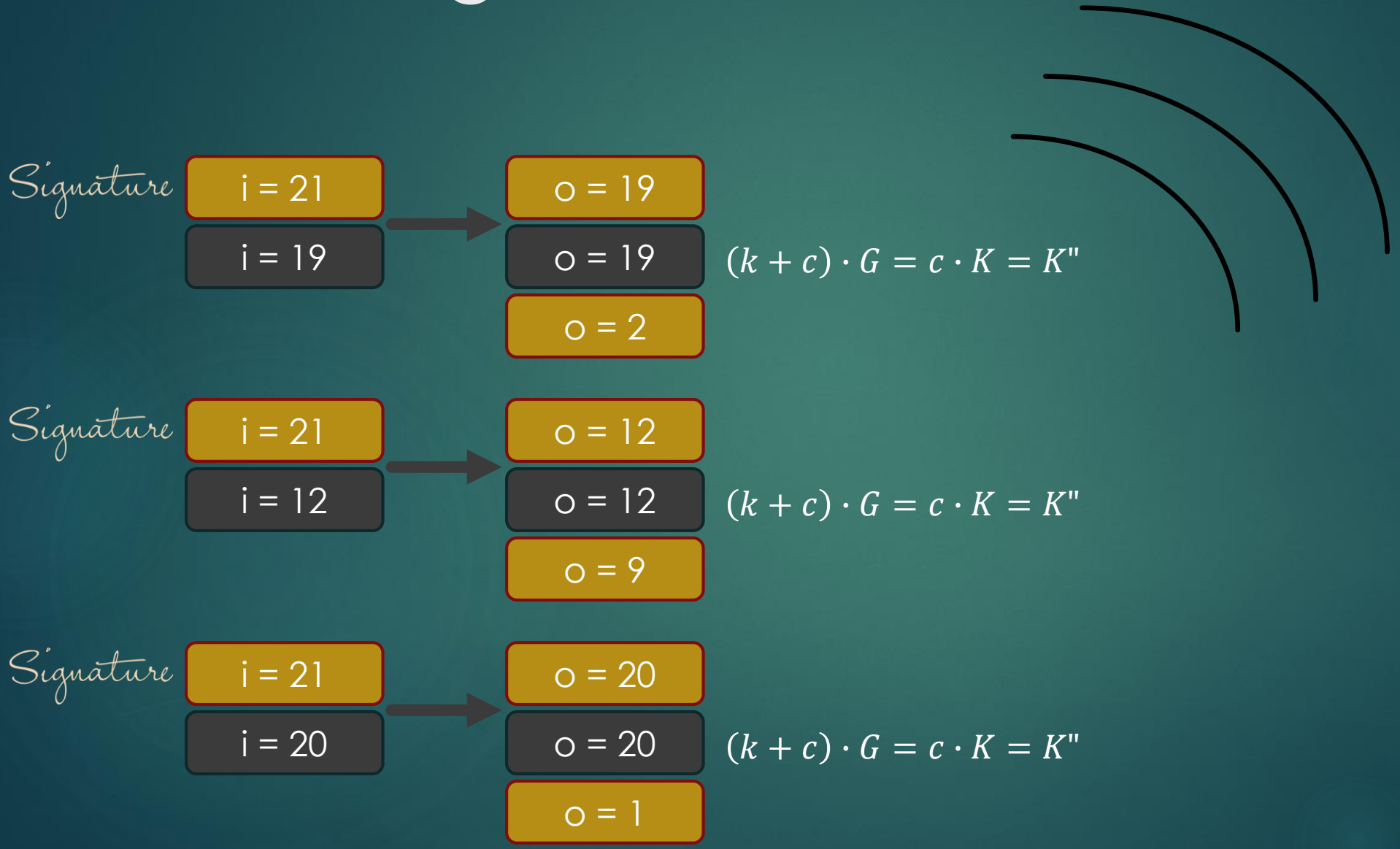
Part 3 – Signatures

- A **proposer** can sign their own input, and can validate that all of the information in the CoinJoin is valid. How can the proposer get the signature of the participant?
- A proposer could first sign the CoinJoin and then present the partially signed bitcoin transaction (PSBT) to the receiver.
  - Perhaps there is a message board
  - Or in a more P2P fashion (Directly send PSBT clusters to active users)

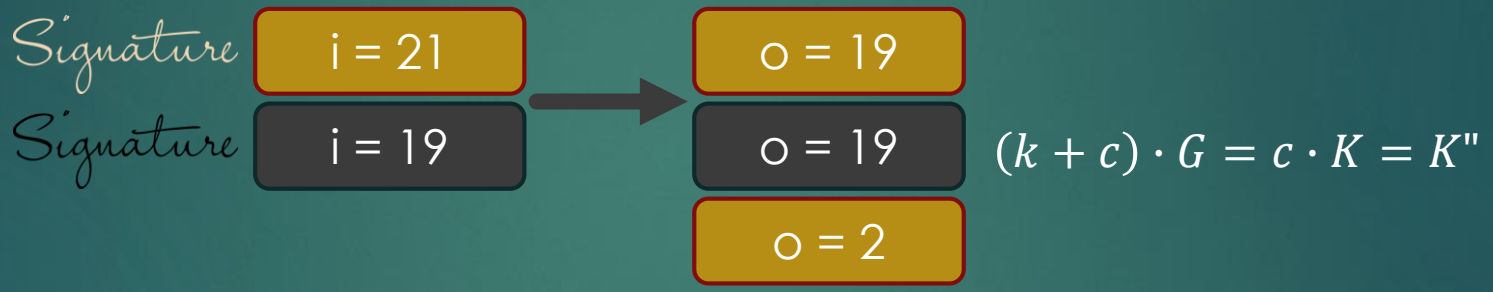
Part 3 – Signatures



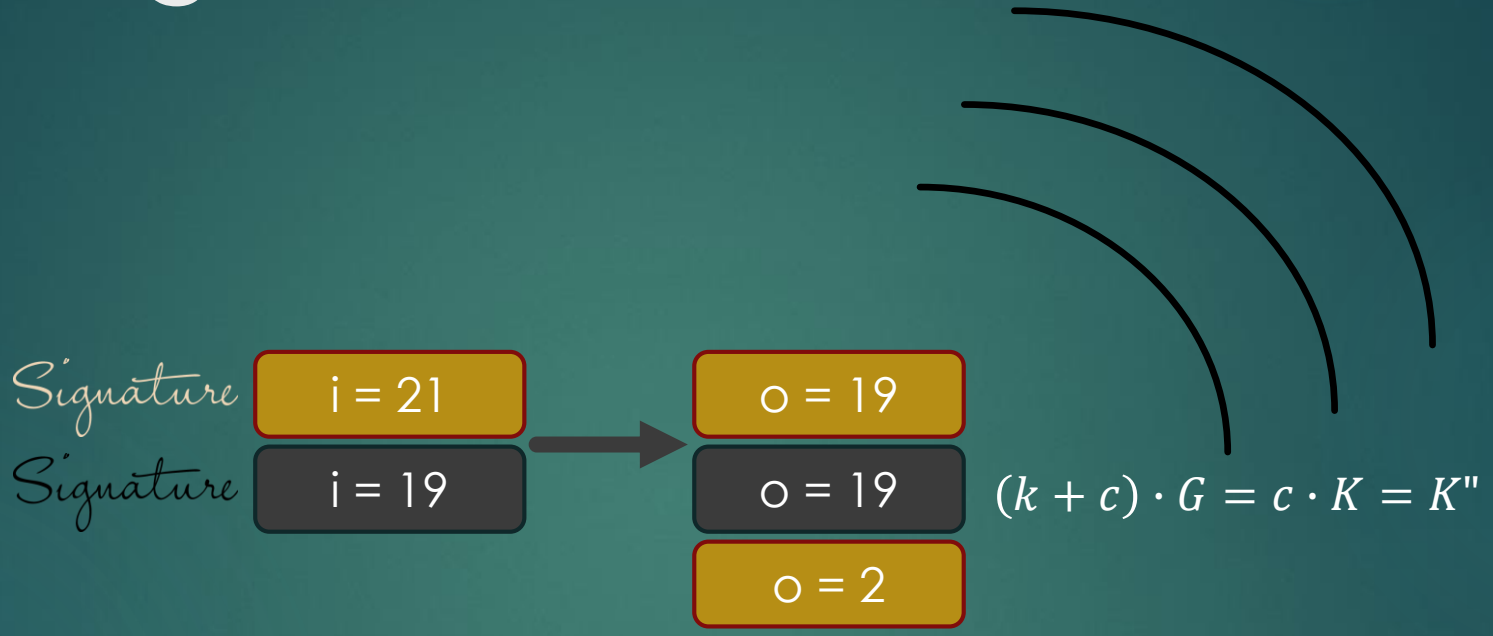
Part 3 – Signatures



Part 3 – Signatures



Part 3 – Signatures



Part 3 – Signatures

- A **receiver** can verify the contents of the PSBT, and can validate that the Diffie Hellman tweaked public key.
- Then the receiver may choose to sign and broadcast
  - Note that the UTXO may or may not already be consumed by another UTXO that was selected by the proposer, but this doesn't matter as there is no loss of funds.
  - Further, observe that if the PSBT is not encrypted against the public key of the UTXOs it attempts to join, a malicious listener could intercept a PSBT and map inputs to outputs.

SNICKER V. 0 Summary

1. A **proposer** selects potential UTXOs from **re-used addresses** that have values smaller than his own.
2. The proposer then constructs outputs by tweaking the **receivers** public key with a **DH key exchange**.
3. The proposer then broadcasts an **encrypted PSBT** to **public forum** or to peers directly.
4. The receiver can **decrypt the PSBT** and chose to **sign and broadcast.**

SNICKER V. 0 Summary

1. A **proposer** selects potential UTXOs from **re-used addresses** that have values smaller than his own.
2. The proposer then constructs outputs by tweaking the **receivers** public key with a **DH key exchange**.
3. The proposer then broadcasts an **encrypted PSBT** to **public forum** or to peers directly.
4. The receiver can **decrypt the PSBT** and chose to **sign and broadcast.**

SNICKER V. 0 Summary (Note)

- SNICKER is a deterministic protocol, and a user restoring their wallet with their seed words should be able to find all of the SNICKER outputs.
- Non-interactive! The receiver only participates at the very last stage, having had no prior interactions with the proposer.

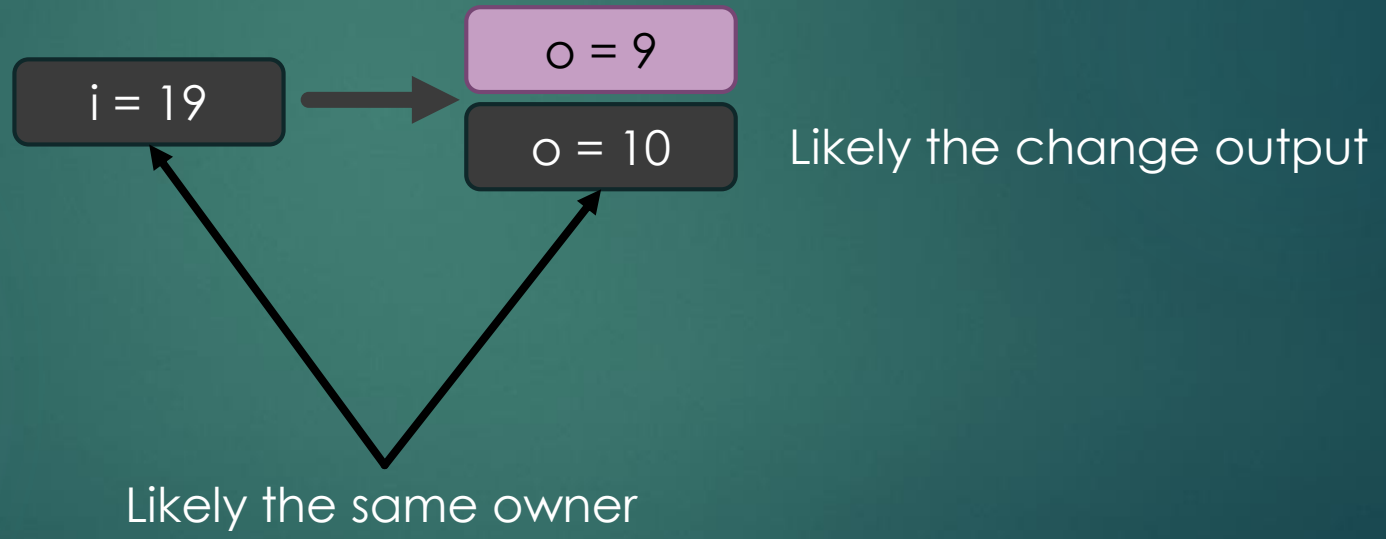
SNICKER V. 1

How can we implement SNICKER without requiring address re-use? That is, without requiring the receiver to reveal their pubkey?

SNICKER V. 1

Solution – the **proposer** can **guess co-ownership** of used addresses and UTXOs sitting in un-used address.

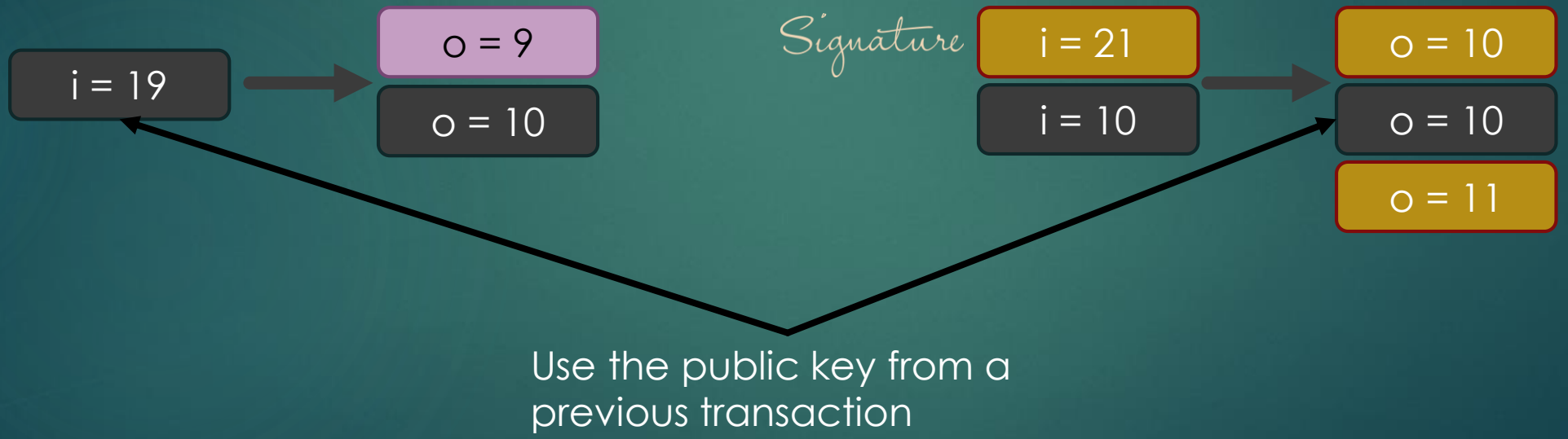
For example:



SNICKER V. 1

Solution – the **proposer** can **guess co-ownership** of used addresses and UTXOs sitting in un-used address.

For example:



SNICKER Summary

Non-interactive CoinJoins are possible between two participants by leveraging:

- Either re-used addresses or co-ownership
- A public board for posting PSBT

Discussion Time

SNICKER

SIMPLE NON-INTERACTIVE COINJOIN WITH KEYS FOR ENCRYPTION REUSED