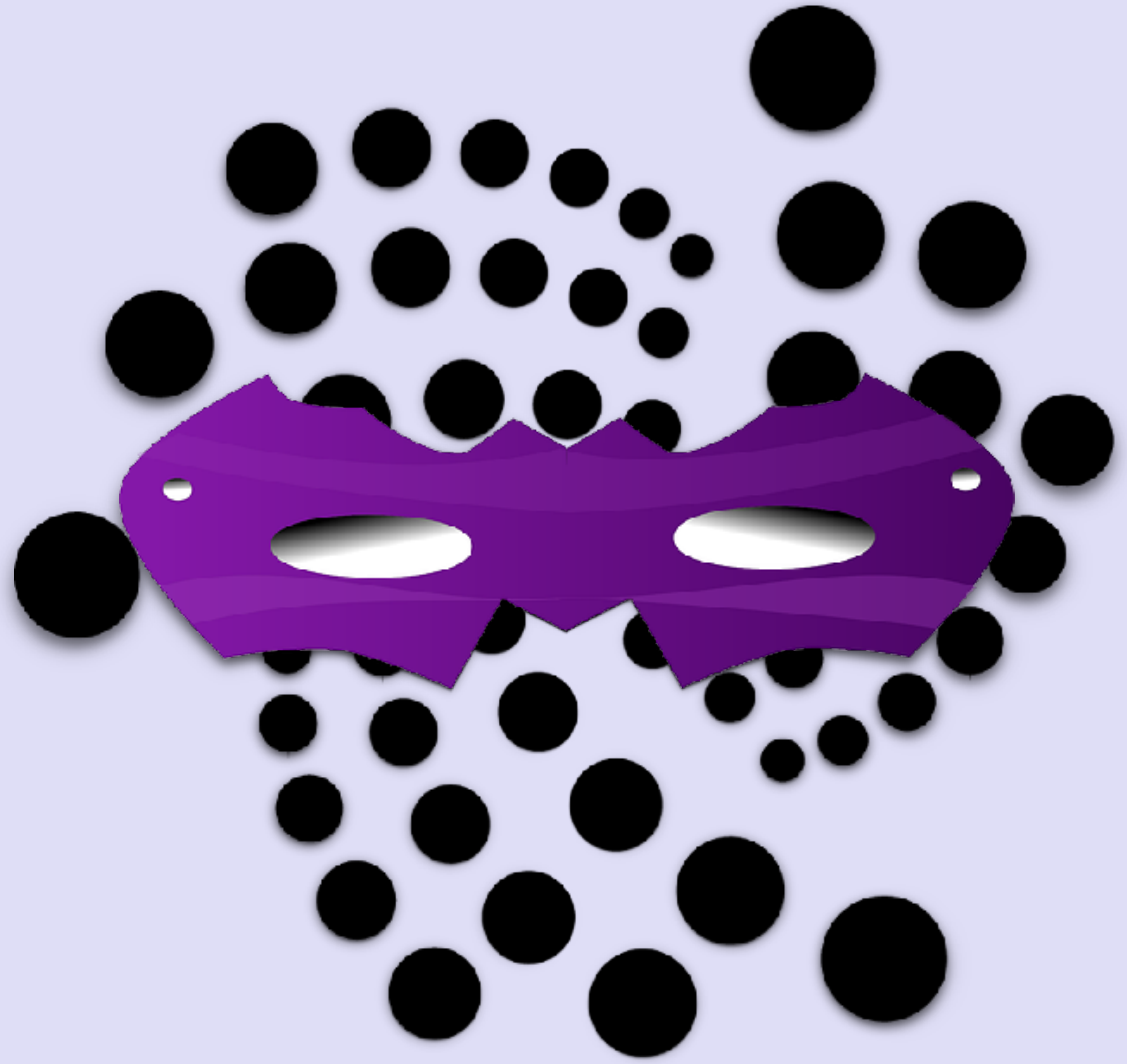
IOTA TUTORIAL 19

Masked Authenticated Messaging



v1.0.1

INTRO

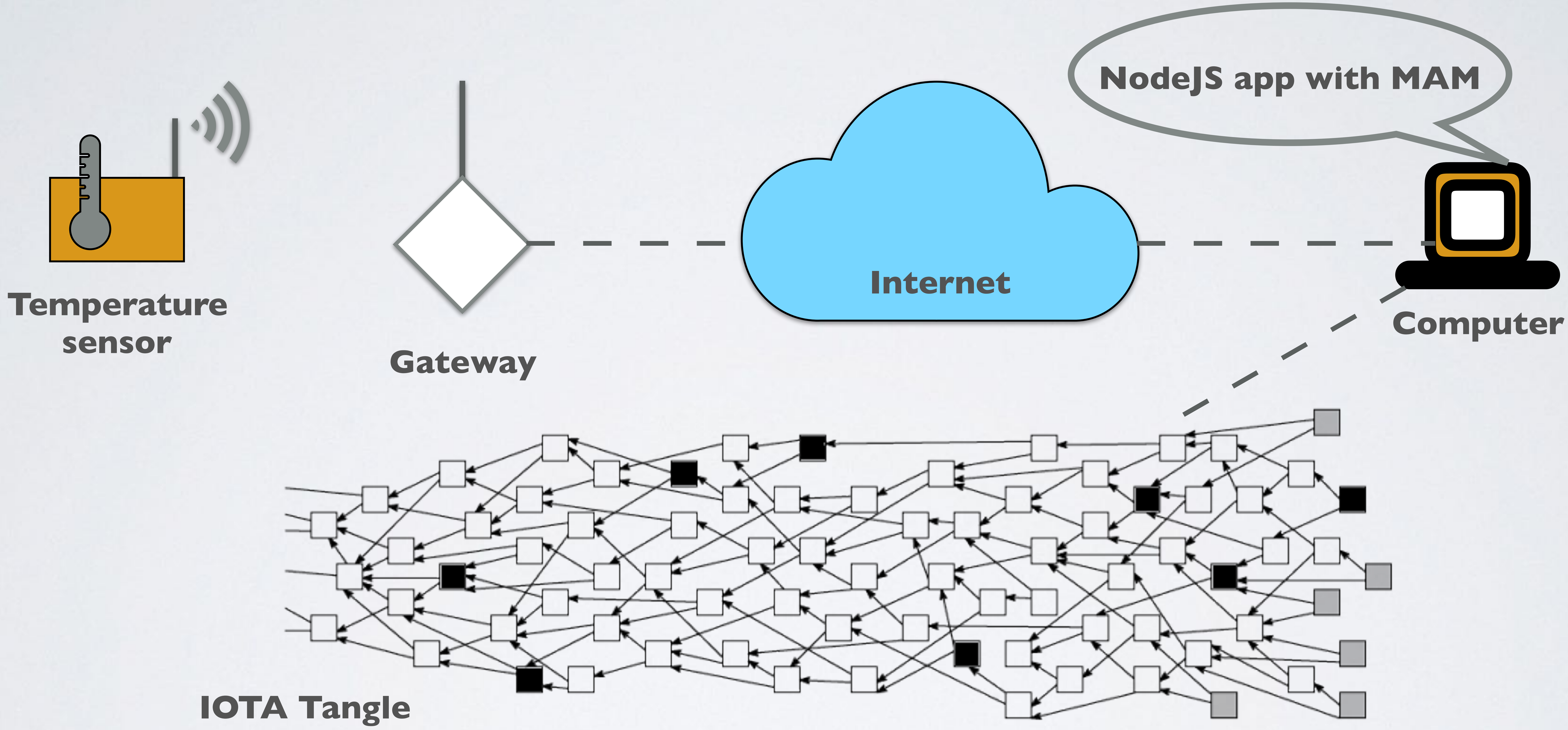
- In this video I will explain what Masked Authenticated Messaging is.

FROM SENSOR TO TANGLE

- IOTA is specific developed for Internet of Things (IoT) or Machine-to-Machine (M2M) economy. IoT devices will generate a huge amount of data which can be stored on the Tangle. These IoT devices can broadcast data using different types of wireless technologies, such as WiFi, BlueTooth, LoRa, ZigBee, etc.

- For example:
  A LoRa (**Lo**ng **Ra**nge) sensor node can measure temperature, humidity, magnetic fields, moisture, etc.
  This LoRa node transmits sensor data to a receiver, also called LoRa gateway.
  The distance between node and gateway can be several kilometres.
  The LoRa gateway is connected to the Internet and sends the data to a server.
  This server is running a NodeJS Masked Authenticated Messaging (MAM) application which in turn sends sensor data to the Tangle.

FROM SENSOR TO TANGLE

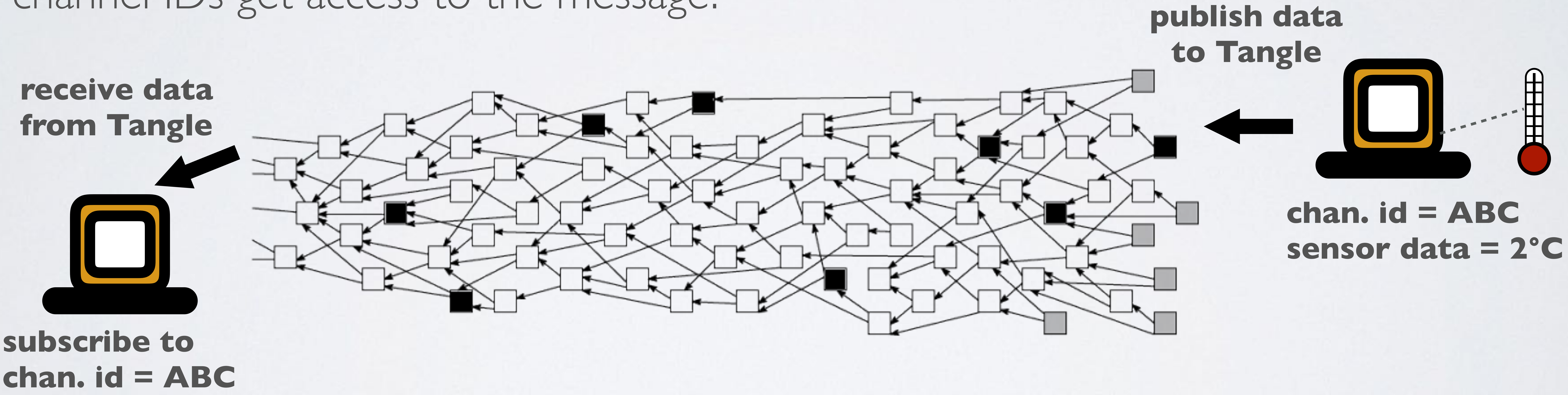


MASKED AUTHENTICATED MESSAGING

- Masked Authenticated Messaging means:
    - The message is encrypted (**Masked**).
    - The message is confirmed to be coming from the device (**Authenticated**).
    - A continuous message stream is created on the Tangle and will carry on until the device stop publishing the data (**Messaging**).
- Masked Authenticated Messaging is a module build on top of IOTA that makes it possible to send messages fully encrypted from authenticated parties.

MASKED AUTHENTICATED MESSAGING

- IOTA Masked Authenticated Messaging (MAM) makes it possible for sensors and other devices to encrypt entire message streams and securely store those in the Tangle each on a separate address. Only authorised parties will be able to read and reconstruct the entire message stream. In essence it works a lot like a radio where only those with the right frequency can listen in. In MAM only those with the right channel IDs get access to the message.



publish data to Tangle

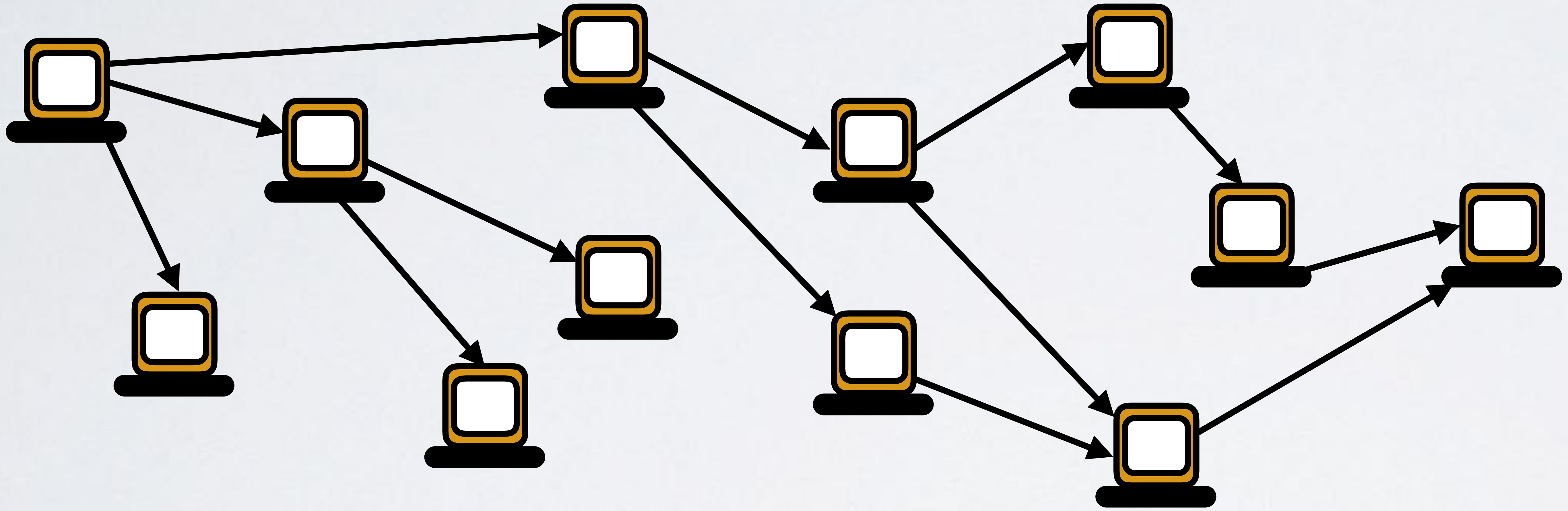
receive data from Tangle

chan. id = ABC
sensor data = 2°C

subscribe to chan. id = ABC

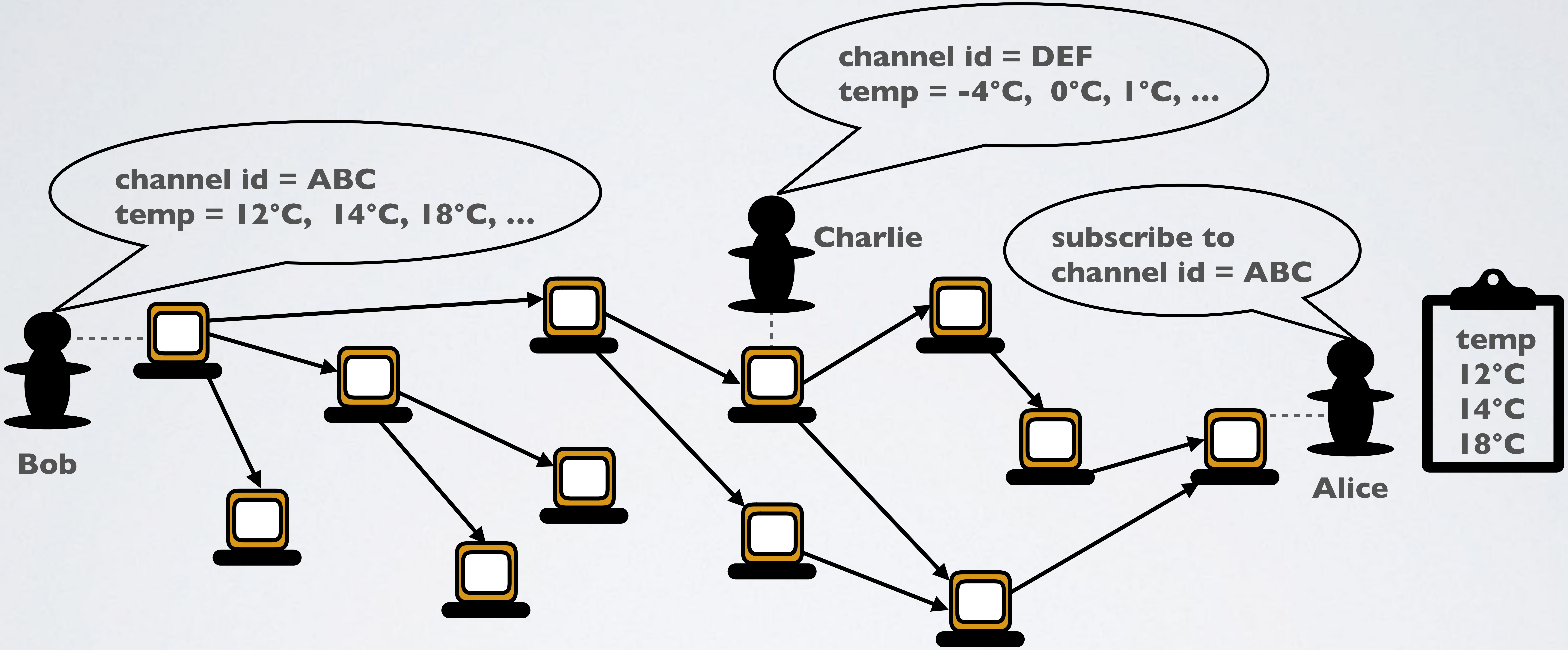
GOSSIP PROTOCOL

- IOTA uses the gossip protocol to propagate messages through the network. Messages are gossiped through the network.



- Bob publishes sensor data to the Tangle under channel id "ABC". Alice is interested in Bob's sensor data and subscribe to his channel id. When a message with channel id "ABC" reaches Alice's node, she will be notified. Messages from Charlie will be ignored.

GOSSIP PROTOCOL



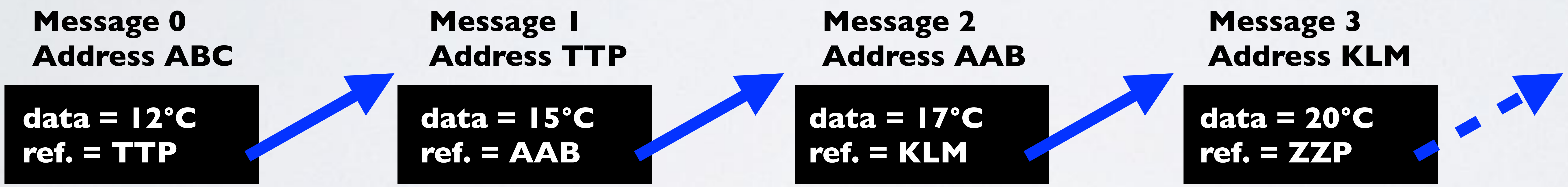
channel id = DEF
temp = -4°C, 0°C, 1°C, ...

channel id = ABC
temp = 12°C, 14°C, 18°C, ...

subscribe to
channel id = ABC

MAM STREAM / MESSAGE CHAIN

- A short simplified explanation how the MAM stream works:
  - In a Masked Authenticated Messaging stream or message chain, every message holds a reference to the next message.



Message 0
Address ABC

data = 12°C
ref. = TTP

Message 1
Address TTP

data = 15°C
ref. = AAB

Message 2
Address AAB

data = 17°C
ref. = KLM

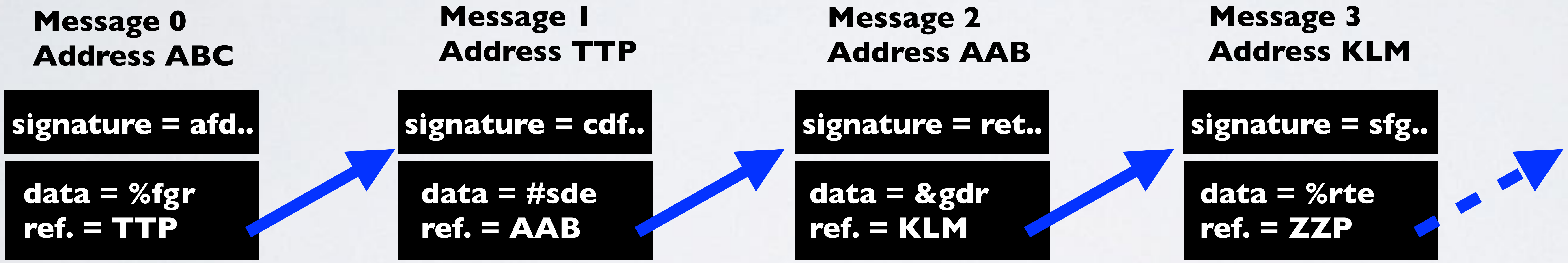
Message 3
Address KLM

data = 20°C
ref. = ZZP

  - The message stream only flows one direction.
    A subscriber with a channel ID has no access to the upstream messages.

MAM STREAM / MESSAGE CHAIN

- In a Masked Authenticated Messaging stream the message is encrypted (masked) and the message also contains a signature.



Message 0
Address ABC

signature = afd..

data = %fgr
ref. = TTP

Message 1
Address TTP

signature = cdf..

data = #sde
ref. = AAB

Message 2
Address AAB

signature = ret..

data = &gdr
ref. = KLM

Message 3
Address KLM

signature = sfg..

data = %rte
ref. = ZZP

- The signature proves that the publisher created the message.

INTERESTING POINTS

- Interesting points to remember:
    - The channel ID is also called the **root**.
      The message is attached to the Tangle using this root.
    - When the message is attached to the Tangle, the transaction does not need to be confirmed.
    - After a snapshot all messages are deleted from the Tangle.
      The messages are still available, if the subscriber is connected to a permanode.

MAM DEMO

- A Masked Authenticated Messaging Demo:
  https://www.mobilefish.com/services/cryptocurrency/mam.html

- This demo uses the MAM Javascript library **mam.web.js** for web applications.
  For nodejs applications use the **mam.node.js** library.
  Both libraries can be found at this location:
  https://github.com/iotaledger/mam.client.js
  The MAM Javascript library publish transactions to the Tangle that contain only messages, with no value.

- *The MAM Javascript library is a work in progress and may have some breaking changes in the future. These will most likely be minor, in addition to extending functionality.*

MAM DEMO

- To install the Masked Authenticated Messaging Demo:
  - View the html source code: https://www.mobilefish.com/services/cryptocurrency/mam.html
  - Read the installation instructions.
- This demo uses the mam.web.js library. https://github.com/iotaledger/mam.client.js/tree/master/lib/mam.web.js

MAM CLIENT JS API

- MAM Client JS API: https://github.com/iotaledger/mam.client.js (mam.web.js)

API	Description
Mam.init(iota, seed, security)	Create and initialise the state object and binds the iota.lib.js library to the mam library.
Mam.changeMode(state, mode, sidekey)	Change the state object channel mode (public, private and restricted) and set the sidekey. Sidekey is used to encode and decode the payload. The sidekey is only used in restricted mode.
Mam.create(state, message)	Create the payload using the state object and message.
Mam.decode(payload, sidekey, root)	Decode the payload using the sidekey and the root.
Mam.attach(payload, address)	Attach the payload to the Tangle.

MAM API

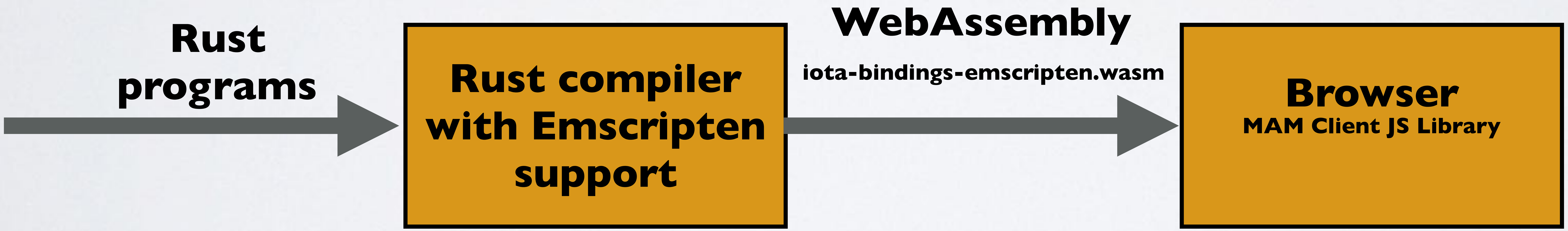
- MAM Client JS API: https://github.com/iotaledger/mam.client.js (mam.web.js)

API	Description
Mam.subscribe(state, channelRoot, channelKey)	Add a subscription to the state object.
Mam.fetch(root, mode, sidekey, callback)	Fetch the message stream sequentially from a known root and optional sidekey.
Mam.fetchSingle(root, mode, sidekey, rounds)	Fetch a single transaction from a known root and optional sidekey.
Mam.listen(channel, callback)	Execute fetch after default 5 seconds.
Mam.getRoot(state)	Get the current root.

IOTA-BINDINGS-EMSCRIPTEN.WASM

- The MAM API (Mam.init, Mam.changeMode, Mam.create, …) can be found in file: https://github.com/iotaledger/mam.client.js/blob/master/src/index.js

- The MAM Client JS Library is a wrapper library which uses the WebAssembly iota-bindings-emscripten.wasm file.
  - WebAssembly is a new binary format for executing code on the web.
  - Rust is a programming language similar to C++.
  - Emscripten is a source-to-source compiler which can also produce WebAssembly.


Rust programs
Rust compiler with Emscripten support
WebAssembly
iota-bindings-emscripten.wasm
Browser
MAM Client JS Library

IOTA-BINDINGS-EMSCRIPTEN.WASM

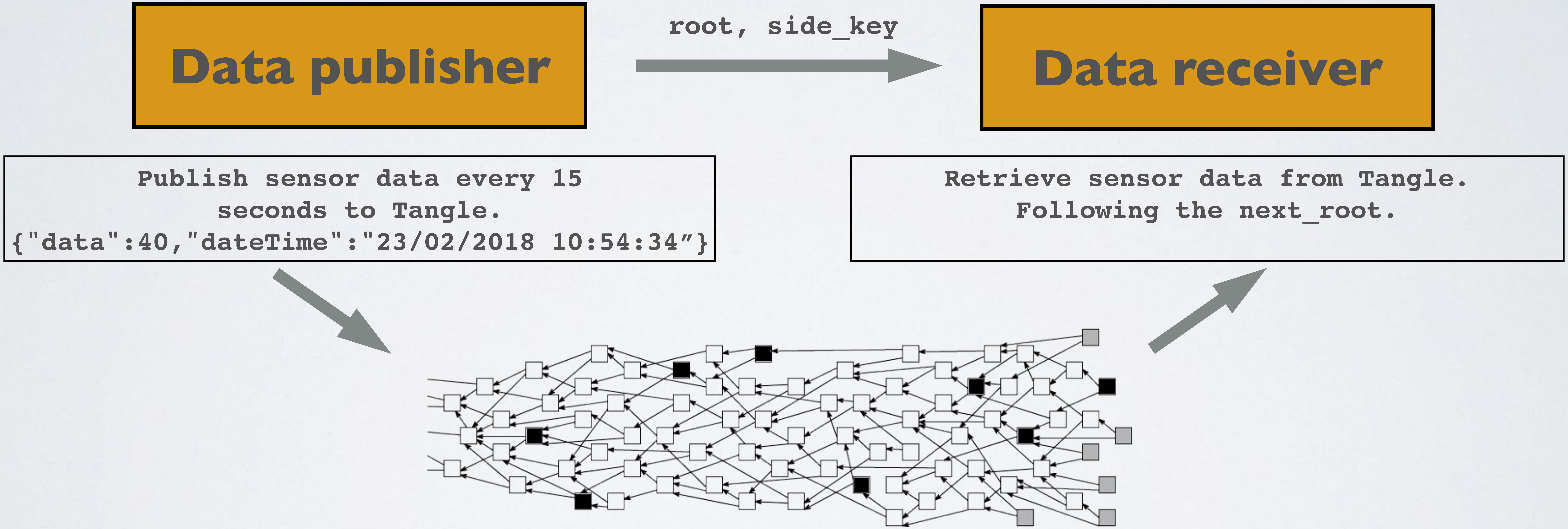
- The MAM Client JS Library uses the following Rust modules:
  https://github.com/iotaledger/MAM
  https://github.com/iotaledger/iota.rs

- The bindings between the MAM Client JS Library and Rust modules can be found in the bindings folders:
  https://github.com/iotaledger/MAM/tree/master/bindings
  https://github.com/iotaledger/iota.rs/tree/master/bindings

- The actual Rust implementations can be found in the other folders:
  https://github.com/iotaledger/MAM
  https://github.com/iotaledger/iota.rs

MAM DEMO

- The Masked Authenticated Messaging Demo: https://www.mobilefish.com/services/cryptocurrency/mam.html



Data publisher

root, side_key

Data receiver

Publish sensor data every 15 seconds to Tangle.
{"data":40,"dateTime":"23/02/2018 10:54:34”}

Retrieve sensor data from Tangle.
Following the next_root.

MASKED AUTHENTICATED MESSAGING OBJECT

- Creating a MAM object using the Mam.create API.

```
{
    "state": {
        "subscribed": [],
        "channel": {
            "side_key": null,
            "mode": "public",
            "next_root": "GNFB...EOAA",
            "security": "2",
            "start": 1,
            "count": 1,
            "next_count": 1,
            "index": 0
        },
        "seed": "OXHU...CMNU"
    },
    "payload": "AHBA...OQLA9",
    "root": "HYKZ...TFHD",
    "address": "HYKZ...TFHD"
}
```

```
const publish = async function(packet) {
    let trytes = iota.utils.toTrytes(JSON.stringify(packet));
    let message = Mam.create(mamState, trytes);

    console.log(JSON.stringify(message,null,″\t"));
    :
}
```

MAM object

MAM OBJECT

Field	Description
side_key	In restricted mode the message is encrypted by the side_key To decrypt, use the same side_key. More info and additional info
mode	Public: Anyone has access to the message. Private: Only the publisher has access to the message. Restricted: Anyone with a side_key has access to the message. More info 1, info 2, and info 3.
next_root	To access the next message in the message chain use the next_root. In private and restricted mode, the next address is calculated as follows: address = hash(next_root) In public mode the address = next_root More info 1, info 2 and reminder of what a key looks like.

MAM OBJECT

Field	Description
security	Security level 1, 2 or 3
start	In the message chain, the Merkle trees uses the same seed. The leaf (= hash(address)) in each Merkle tree is only used once and each leaf has a corresponding key index number. Start refers to the first leaf key index number in the Merkle tree. More info 1 and info 2.
count	The number of leaves in the Merkle tree. More info 1 and info 2.
next_count	When a message is created, always 2 Merkle trees are created: the current Merkle tree and the next Merkle tree. next_count is the number of keys used in the next Merkle tree. More info 1 and info 2.

MAM OBJECT

Field	Description
index	Within the Merkle tree, the index number is used to process each leaf. More info.
seed	In the message chain, the Merkle trees uses the same seed.
payload	Also called masked payload. This payload consist of the actual message, for example sensor data, signature and other information. In restricted mode the actual message and other information are encrypted with the side_key. More info. To decode the payload, goto: https://www.mobilefish.com/services/cryptocurrency/mam.html and select option "Decode payload".

MAM OBJECT

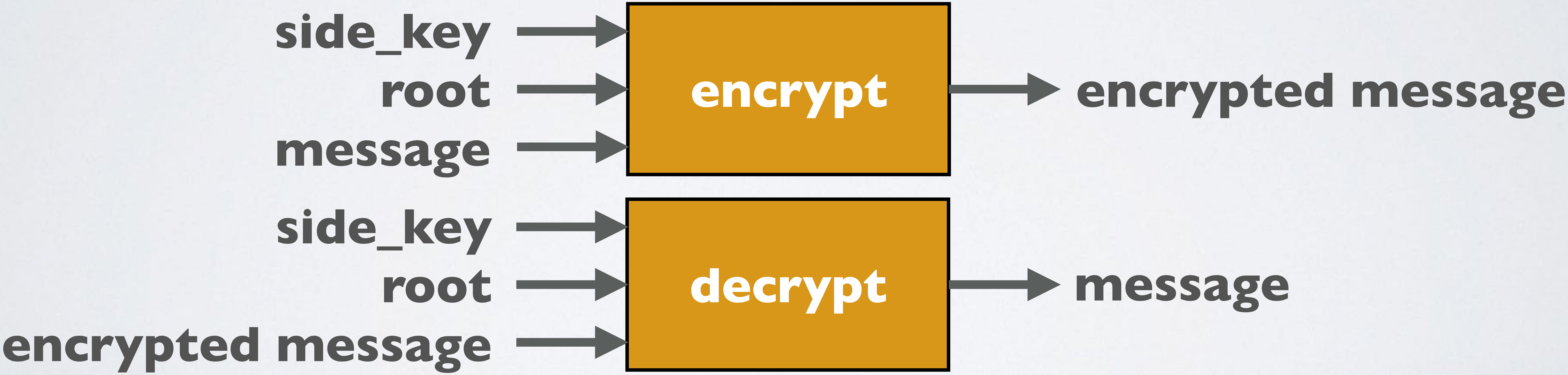
Field	Description
root	The addresses are hashed. These hashed addresses are the leaves and each non-leaf node is a hash of its children. This results in a single hash called the Merkle root. More info 1 and info 2. Each root represents an address where a message can be attached to the Tangle.
address	The address used to attached the message to the Tangle. In private and restricted mode, the address is calculated as follows: address = hash(next_root) In public mode: address = next_root More info.

FIELD: SIDE_KEY

- The side_key is used to encrypt and decrypt the message.
  The side_key is required when using the restricted mode.
  https://github.com/iotaledger/MAM/blob/master/mam/src/mam.rs

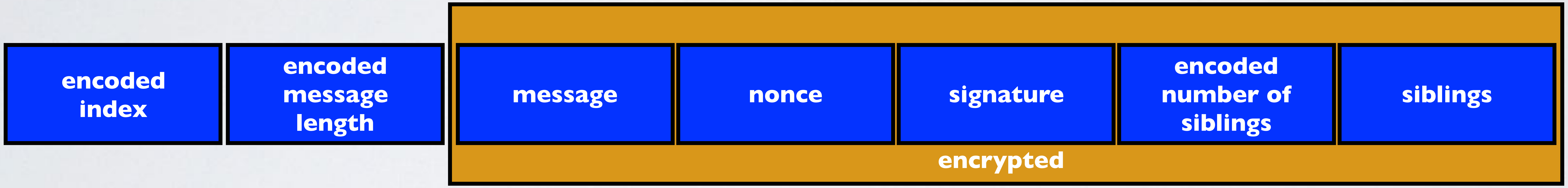
```
pub fn create<C, CB, H>(seed: &[trit], message…)
```

- To encrypt and decrypt message (simplistic explanation):



FIELD: SIDE_KEY

- The side_key is used to encrypt the message, nonce, signature, number of siblings and the siblings. This will be explained in IOTA tutorial 20.



- message = {"payload":"ODGD..GAQD", "next_root":"SJLO..RC9T"}

- The payload contains the actual sensor data converted to trytes:
  {"data":40,"dateTime":"23/02/2018 10:54:34"}

FIELD: MODE

- The publisher publishes messages using the root.
  The publisher can choose the following **channel modes**:

  - **Public: address = next_root**
    Messages can be unwrapped by anybody using the address.

  - **Private: address = hash(next_root)**
    Messages can only be unwrapped if you have the right root, and the root can‘t be deducted from the address due to the hash.

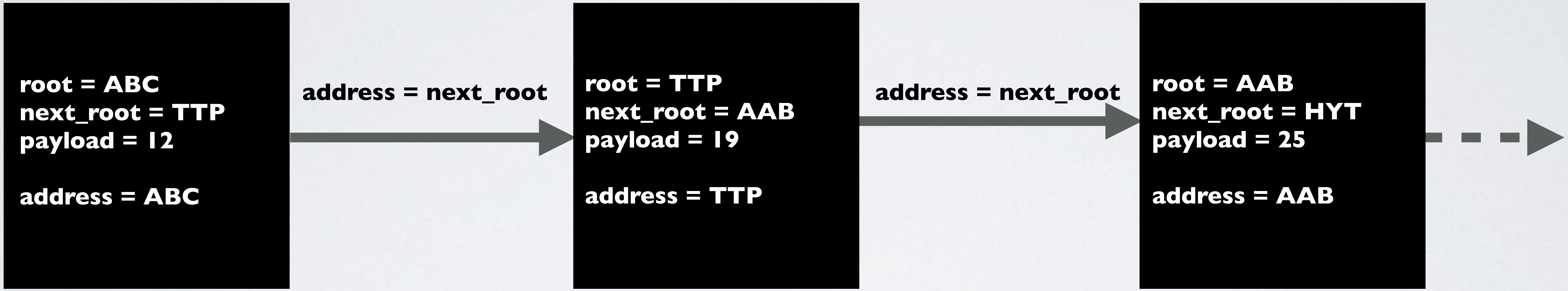
  - **Restricted: address = hash(next_root)**
    Messages can only be unwrapped if you have the right root and side_key.

FIELD: MODE

In all modes: To view the payload you always need the root.

Public mode

If you stumble across address TTP, the payload can be viewed because the root is the same as the address.



root = ABC
next_root = TTP
payload = 12

address = ABC

address = next_root

root = TTP
next_root = AAB
payload = 19

address = TTP

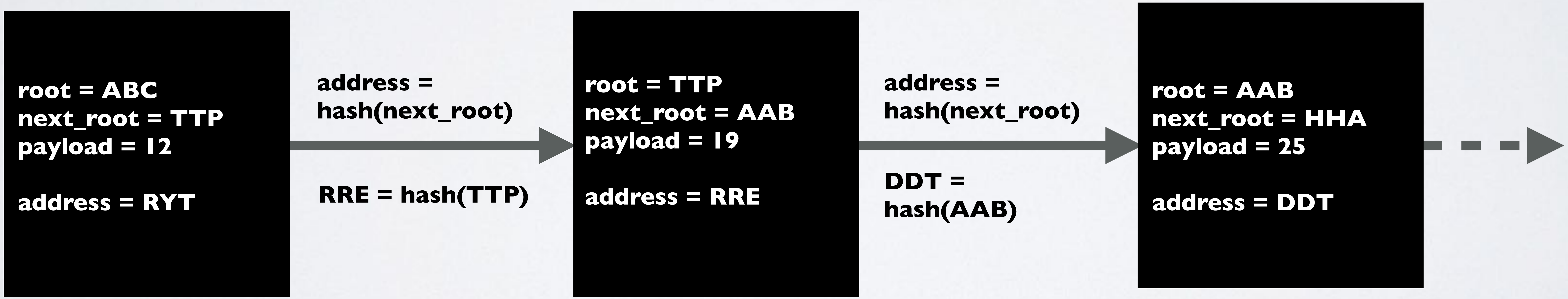
address = next_root

root = AAB
next_root = HYT
payload = 25

address = AAB

Private mode

If you stumble across address RRE, the payload can NOT be viewed because the root is unknown.



root = ABC
next_root = TTP
payload = 12

address = RYT

address = hash(next_root)

RRE = hash(TTP)

root = TTP
next_root = AAB
payload = 19

address = RRE

address = hash(next_root)

DDT = hash(AAB)

root = AAB
next_root = HHA
payload = 25

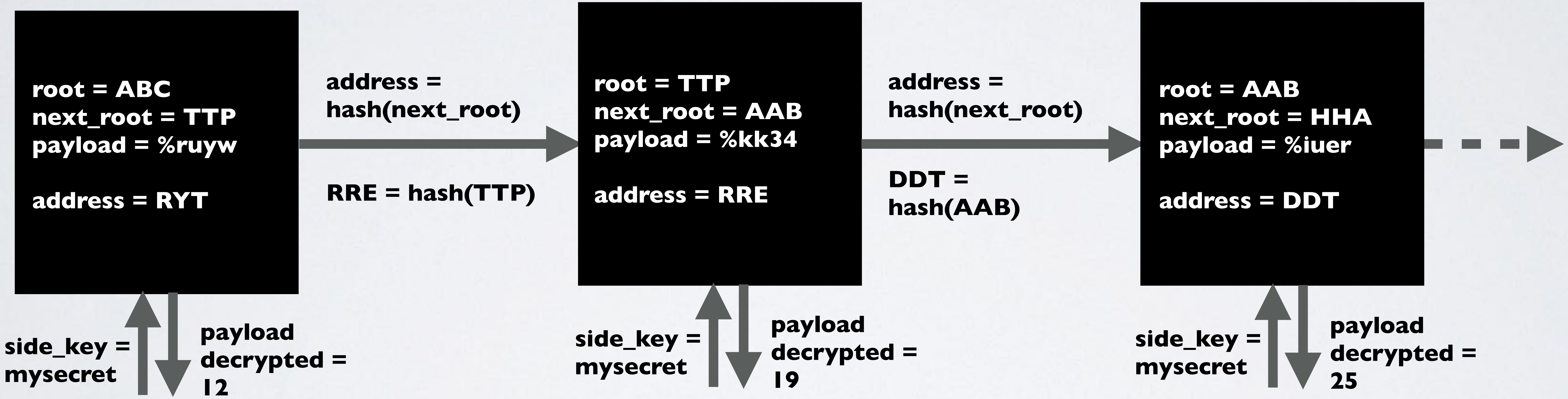
address = DDT

FIELD: MODE

In all modes: To view the payload you always need the root.

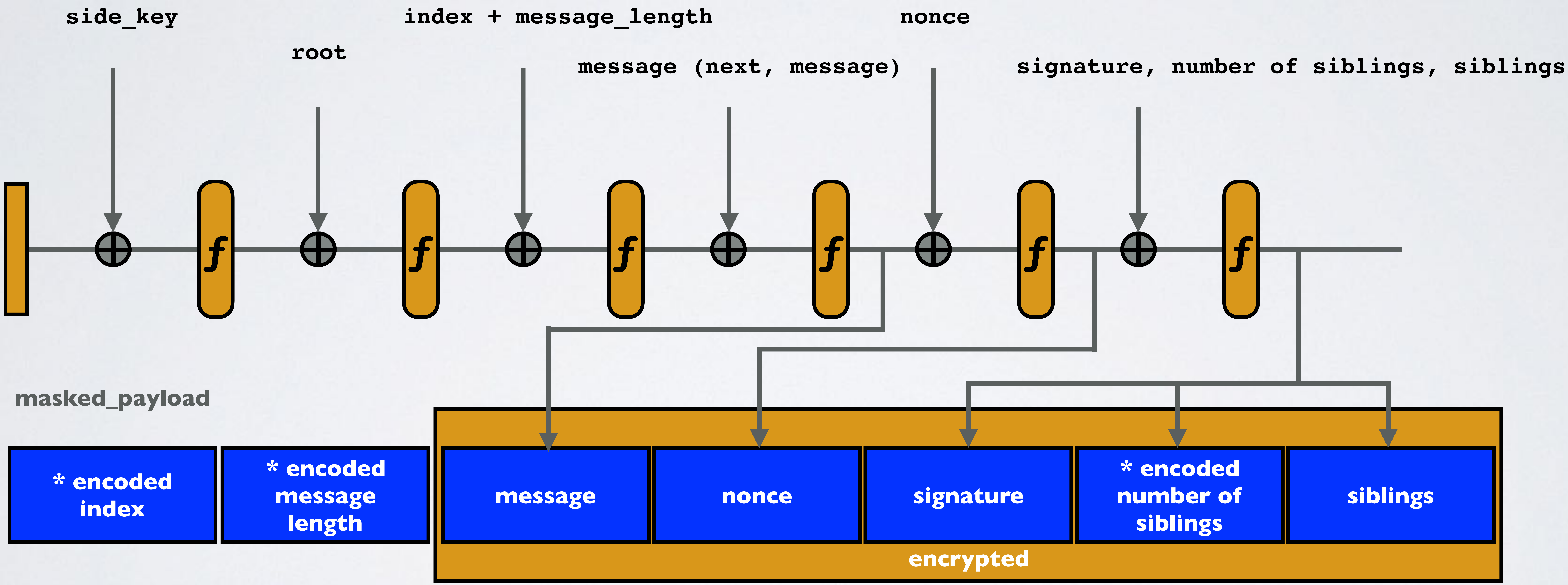
Restricted mode

If you stumble across address RRE, the payload can NOT be viewed because the root and the side_key are unknown.



MASKED_PAYLOAD

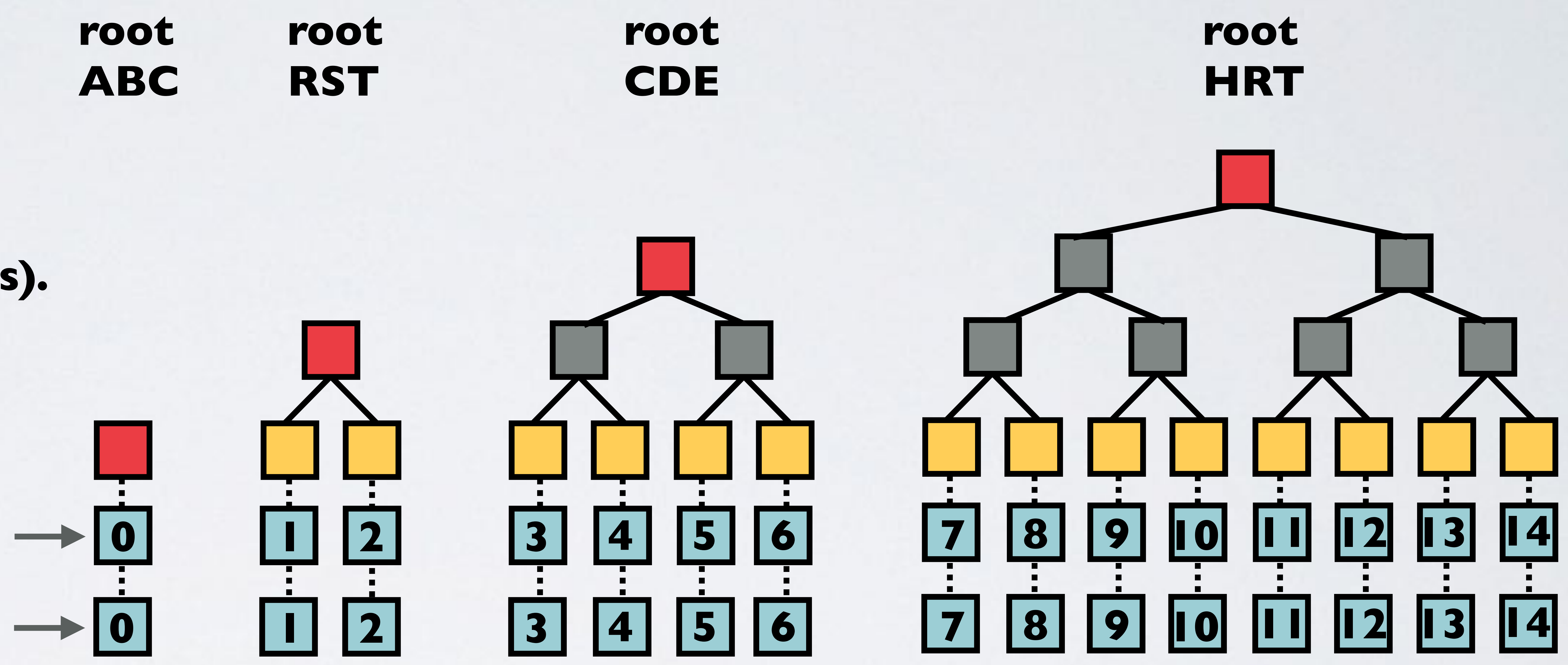
- This drawing will be explained in IOTA tutorial 20.



FIELD: ROOT, NEXT_ROOT, START, COUNT, NEXT_COUNT

#B

Create the Merkle tree using the key(s).



address →

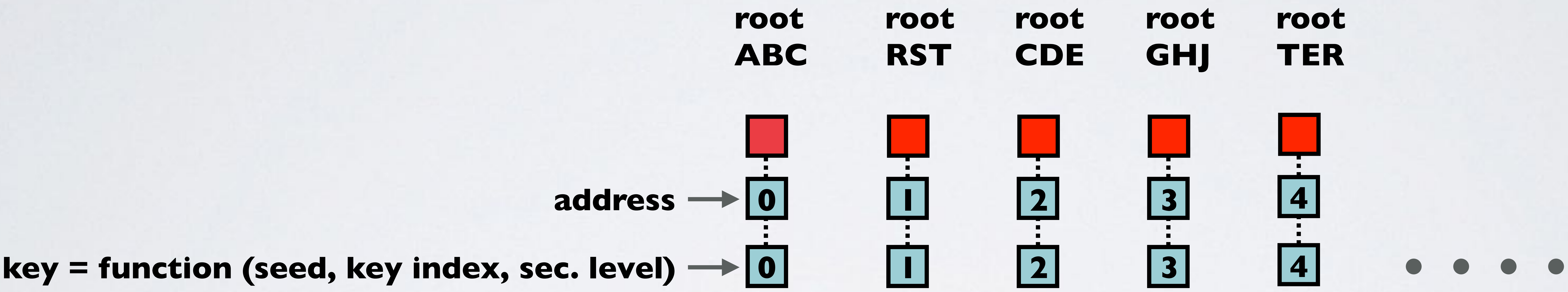
key = function (seed, key index, sec. level) →

count →	1	2	4	8
next_count →	2	4	8	
start →	0	1	3	7
next_root →	RST	CDE	HRT	

seed = 81 trytes (A-Z9)
index = integer [0-9007199254740991]
security level = 1, 2 or 3
public channel mode: address = root
private and restricted channel mode: address = hash(root)

FIELD: ROOT, NEXT_ROOT, START, COUNT, NEXT_COUNT

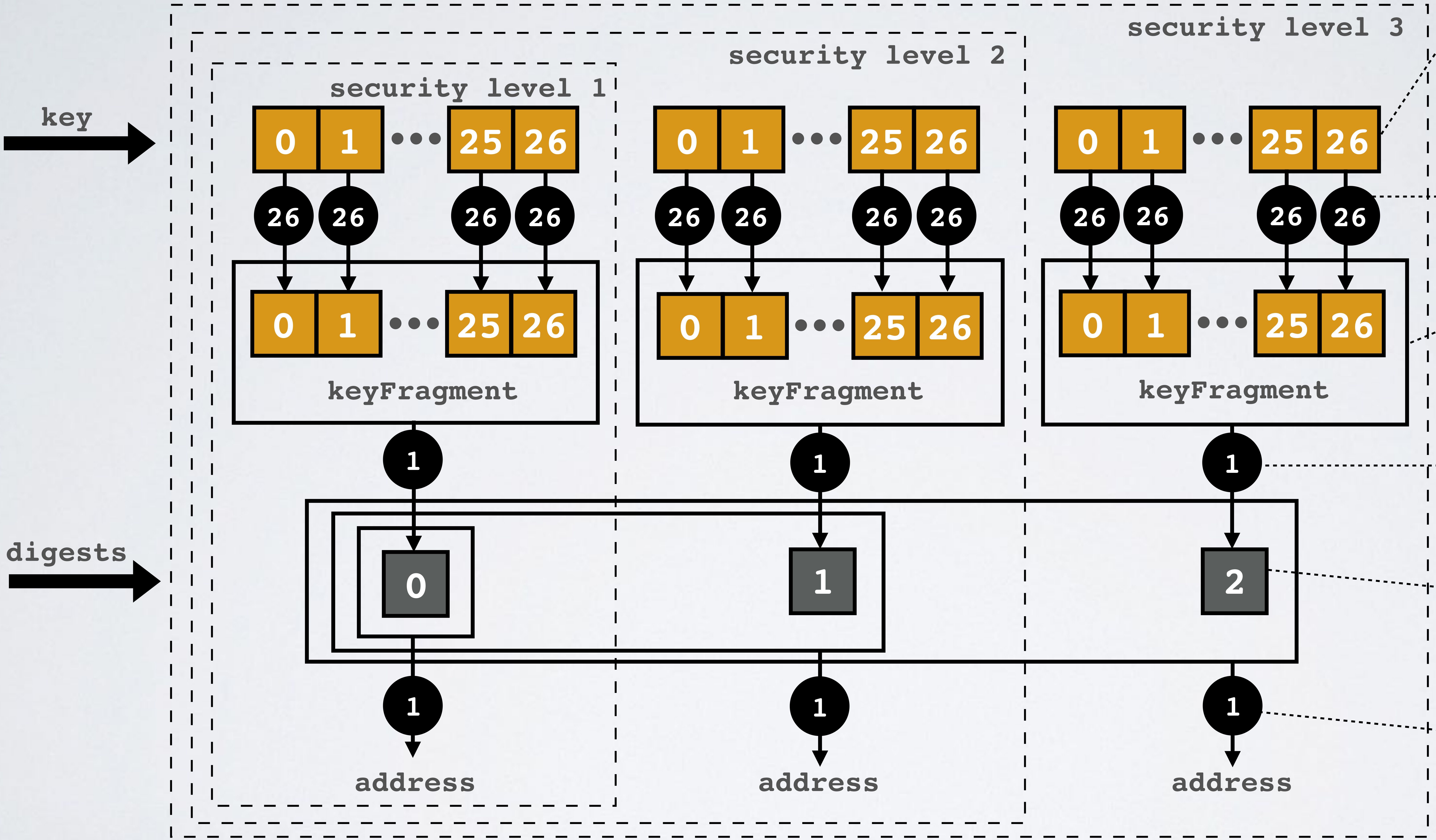
Create the Merkle tree using the key(s).
In the MAM demo each Merkle tree consists only of one node.



count →	**1**	**1**	**1**	**1**	**1**
next_count →	**1**	**1**	**1**	**1**	**1**
start →	**0**	**1**	**2**	**3**	**4**
next_root →	**RST**	**CDE**	**GHJ**	**TER**	**XTY**

seed = 81 trytes (A-Z9)
index = integer [0-9007199254740991]
security level = 1, 2 or 3
public channel mode: address = root
private and restricted channel mode: address = hash(root)

REMINDER OF WHAT A KEY LOOKS LIKE



segment

each segment consists of 81 trytes

hash each segment 26x

27 segments forms a keyFragment

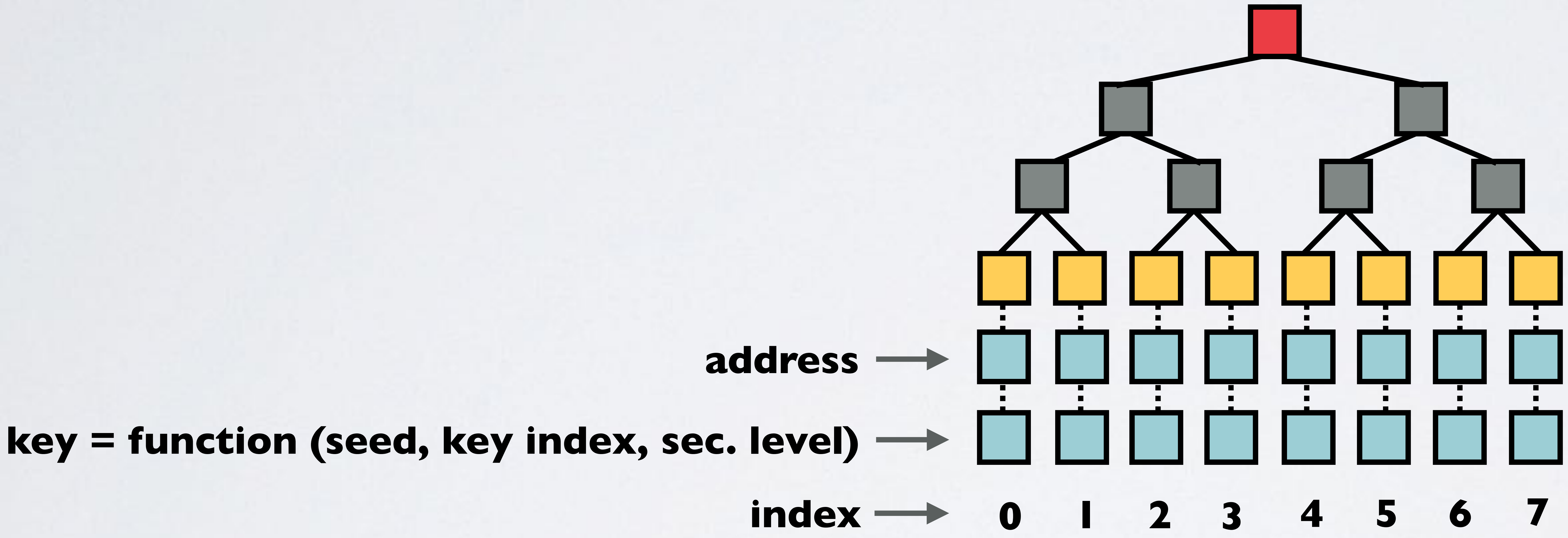
hash each keyFragment 1x

each digests consists of 81 trytes

hash n digests 1x

FIELD: INDEX

- Within the Merkle tree, the index number is used to process each leaf.



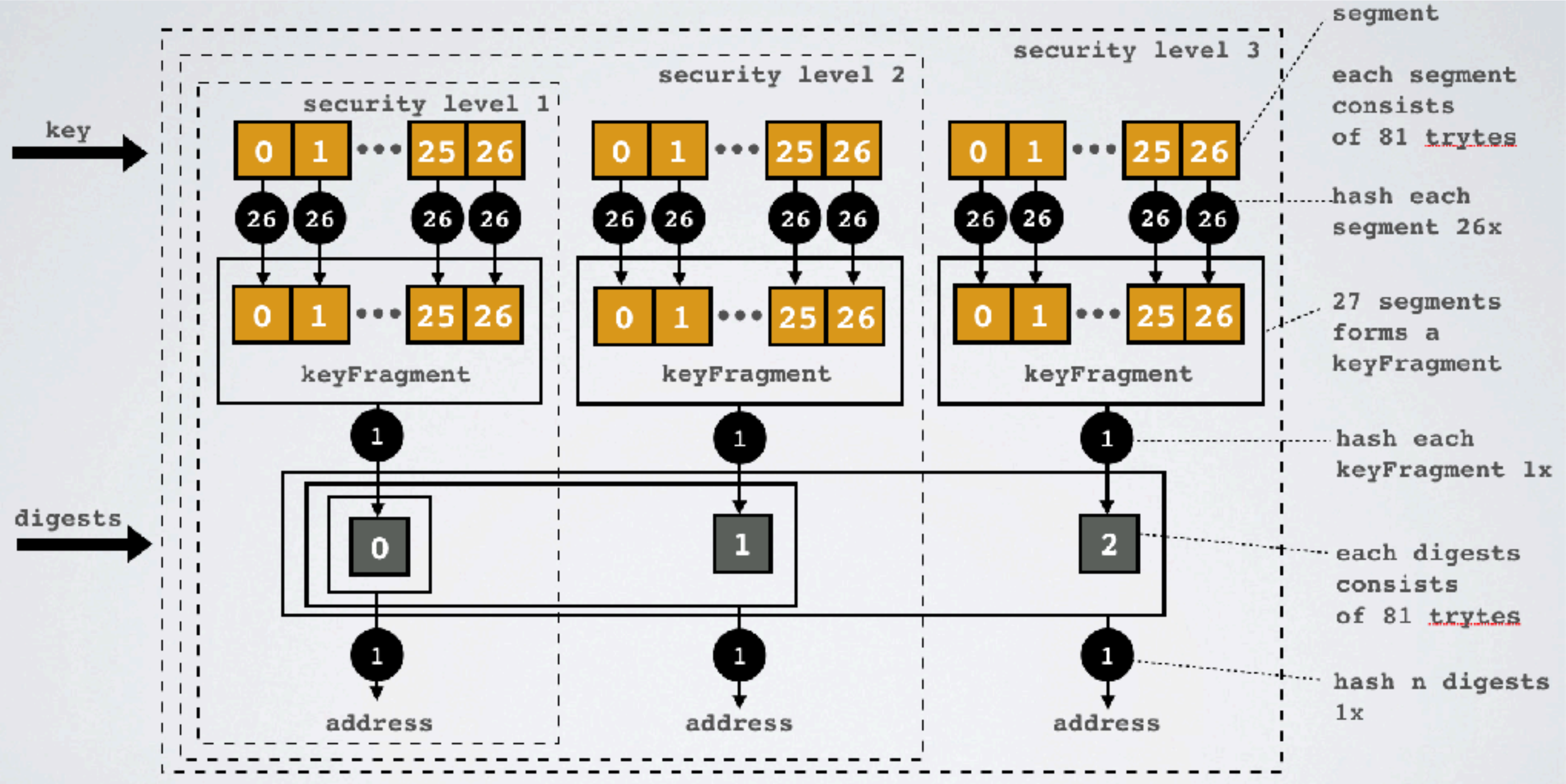
address →

key = function (seed, key index, sec. level) →

index → 0 1 2 3 4 5 6 7

FIELD: ADDRESS

- The address calculation:
  https://github.com/iotaledger/mam.client.js/blob/master/src/index.js
  const create = (state, message)
  If channel mode = public: **address = next_root**
  If channel mode = private or restricted: **address = hash(next_root)**



The address is NOT calculated the same way as calculating the address for a normal transaction.

MAM PUBLISHING AND RECEIVING EXPLAINED

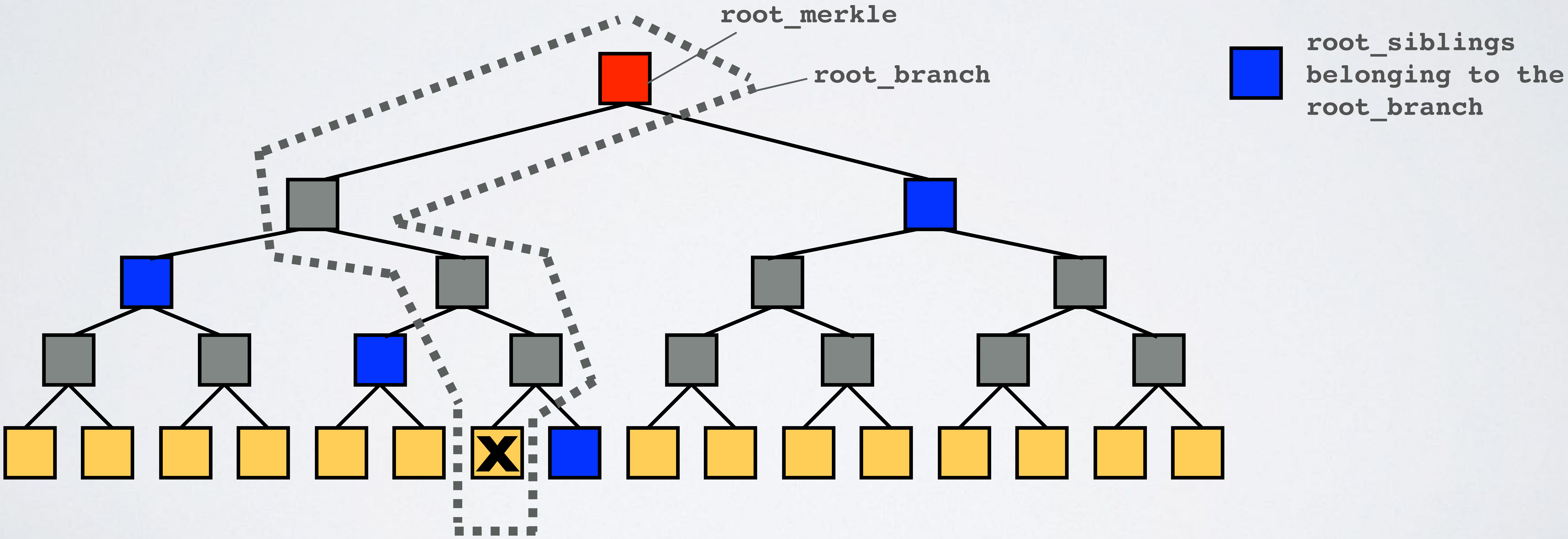
- The Masked Authenticated Messaging demo:
  https://www.mobilefish.com/services/cryptocurrency/mam.html

- The Masked Authenticated Messaging publishing data explained:
  https://www.mobilefish.com/download/iota/mam_publish_code.txt

- The Masked Authenticated Messaging receiving data explained:
  https://www.mobilefish.com/download/iota/mam_receive_code.txt

ROOT_MERKLE, ROOT_BRANCH, ROOT_SIBLINGS

- Setup Merke tree:
  https://github.com/iotaledger/mam.client.js/blob/master/lib/mam.web.js
  `function createMessage(SEED, MESSAGE, SIDE_KEY, CHANNEL)`



MAM EXAMPLES

#C

- MAM example 1: **security level = 2, channel mode = public**
  https://www.mobilefish.com/download/iota/mam_object1.txt
  https://www.mobilefish.com/download/iota/mam_tx_object1.txt

- MAM example 2: **security level = 2, channel mode = private**
  https://www.mobilefish.com/download/iota/mam_object2.txt
  https://www.mobilefish.com/download/iota/mam_tx_object2.txt

- MAM example 3: **security level = 2, channel mode = restricted, key = mysecret**
  https://www.mobilefish.com/download/iota/mam_object3.txt
  https://www.mobilefish.com/download/iota/mam_tx_object3.txt

MAM EXAMPLES

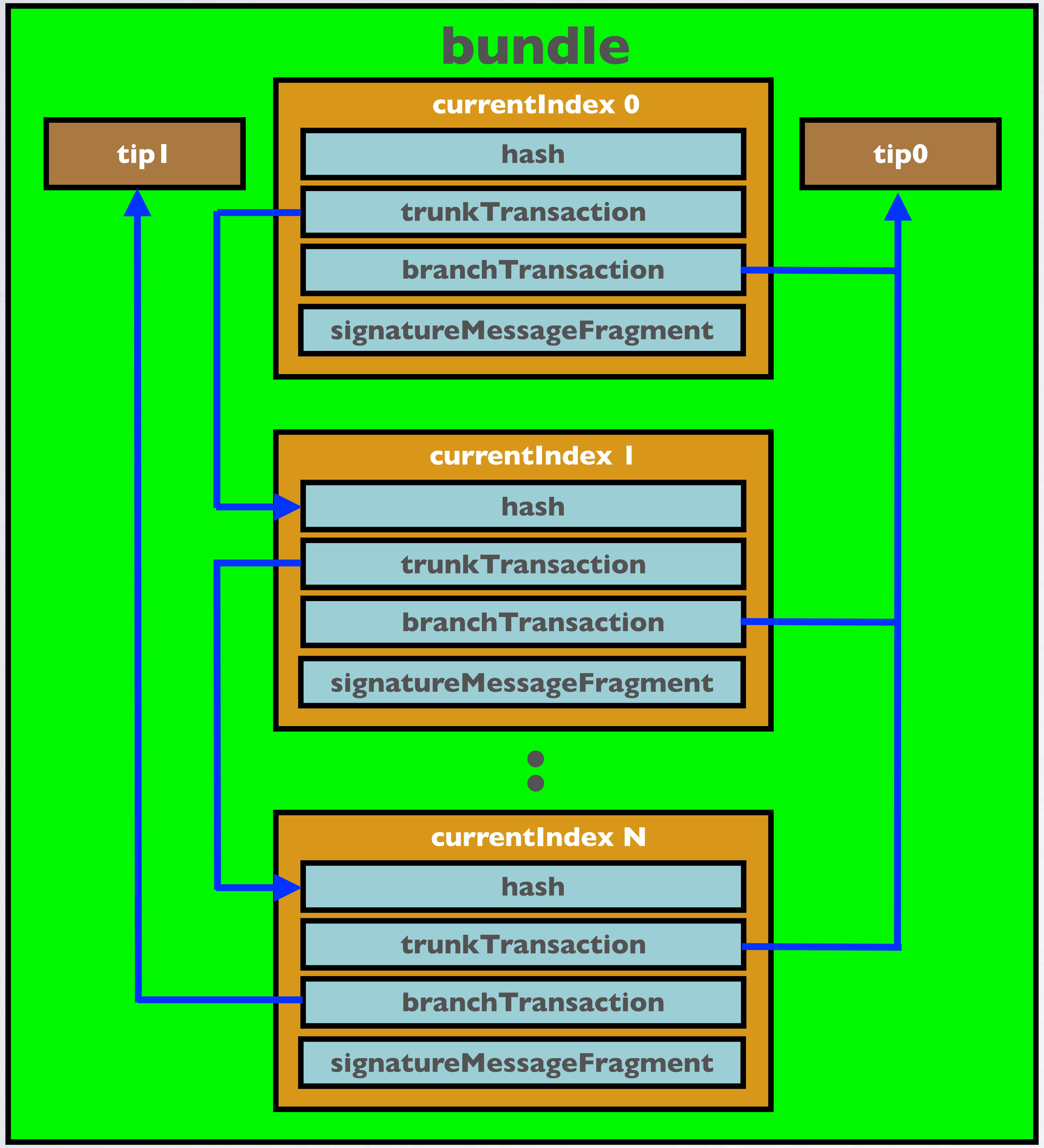
#D

- MAM example 4: **security level = 1, channel mode = public**
  https://www.mobilefish.com/download/iota/mam_object4.txt
  https://www.mobilefish.com/download/iota/mam_tx_object4.txt

- MAM example 5: **security level = 3, channel mode = public**
  https://www.mobilefish.com/download/iota/mam_object5.txt
  https://www.mobilefish.com/download/iota/mam_tx_object5.txt

TRANSACTIONS IN BUNDLE

#E



The MAM library creates the masked payload and calls the iota.api.sendTransfer function.

The sendTransfer function in turn calls the iota.api.prepareTransfers function
which divides the masked payload in smaller parts.

Each part size will be 2187 trytes, representing a signatureMessageFragment.

The last part will be padded with nines to make its size 2187 trytes.

A transaction object is created for each part.
The transaction objects together forms a transaction bundle.