MAM1.1 Overview

Vlad Semenov

IOTA Foundation

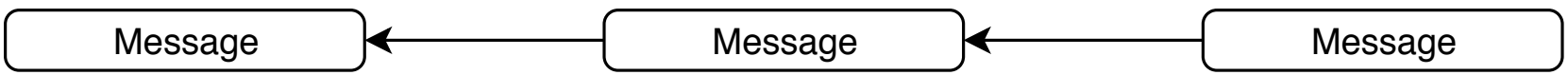
January 2020

MAM1.1

- MAM is a Framework for cryptographic protocols — Applications.
- MAM is **not** a channel, a chat, or a messenger.
- MAM Application is Message oriented.
- Message is described in Protobuf3 (PB3) syntax.
- PB3 messages are Wrapped/Unwrapped with Spongos.
- Messages can refer to other Messages.
- PB3 (and hence MAM) supports AE via Spongos, MSS signatures, NTRU key encapsulation.

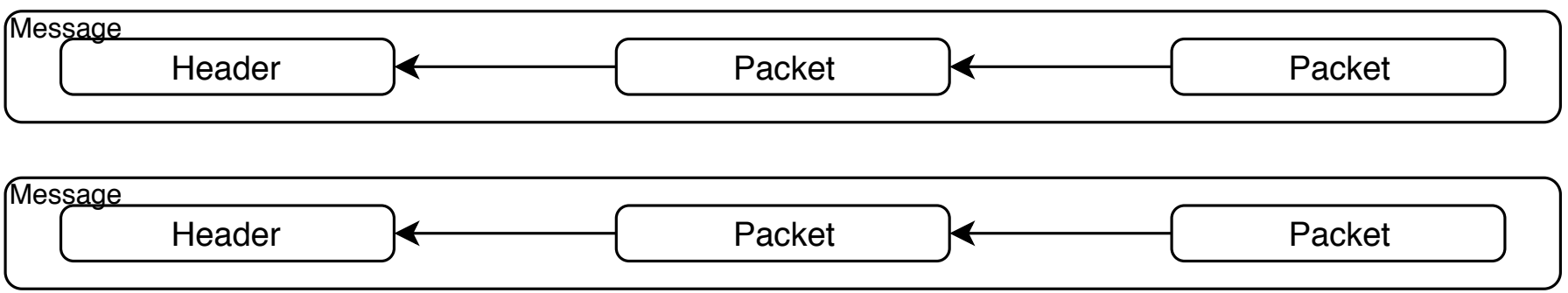
MAM0

- One fixed `SIDEKEY` preshared among all Subscribers.
- Explicit public key change (`NEXTROOT`) in each message.
- Fixed message format, all messages are signed.
- Linear(?) random access to a message within a channel.



MAM1

- Spongos, MSS, NTRU, PB3.
- Message has fixed format.
- Each Message contains an explicit Keyload.
- Packets have linear topology.
- Channel/Endpoint is not flexible enough.

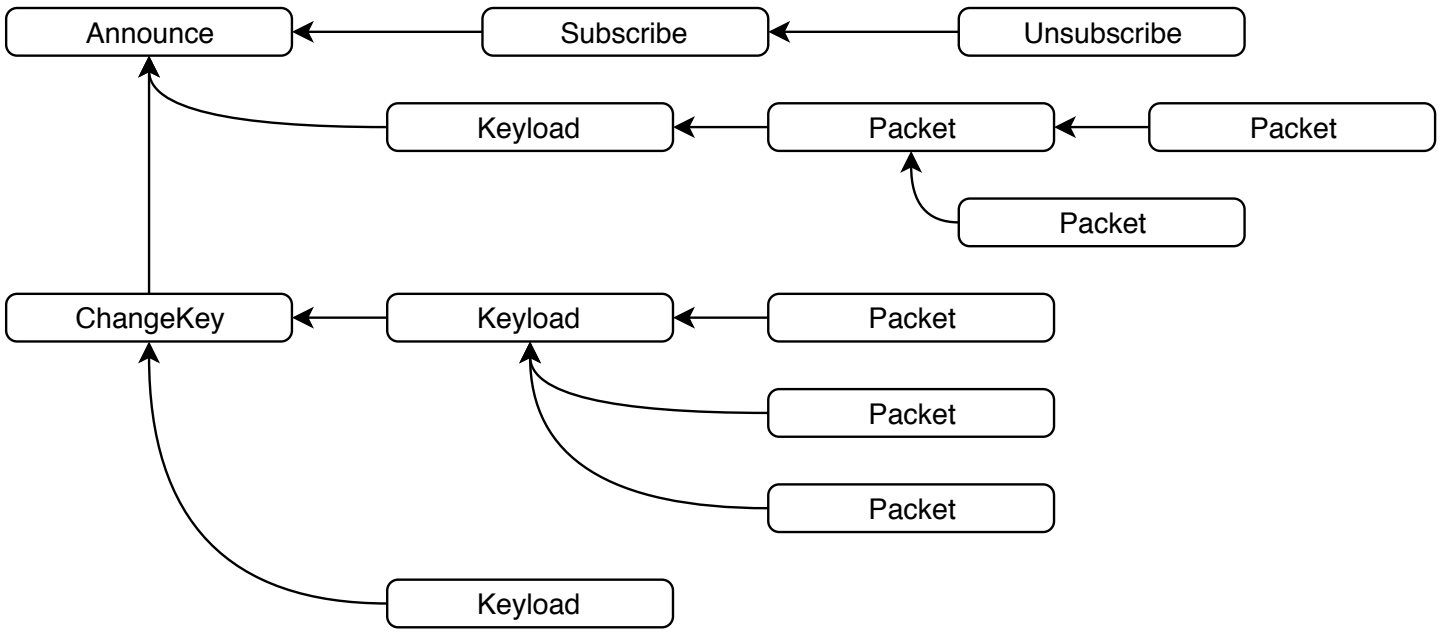


MAM1.1 Changes

- Added Spongos `join` operation.
- Added PB3 `link` type and `join` command.
- Changed NTRU to use `spongos` instance as input.
- Deprecated MAM1 Message format (including Channel/Endpoint).
- Added `Application` layer.
- Added `Channel` Application.

MAM1.1 Channel Application

- One Author, many Subscribers.
- Explicit messages: `Announce`, `ChangeKey`, `Subscribe`, `Unsubscribe`, `Keyload`, `SignedPacket`, `TaggedPacket`.
- Tree-like topology.
- Various use-cases.



Protobuf3 example

```
message Packet {
  oneof {
    join link external_keyload = 0;
    { absorb tryte nonce[27];
      repeated {
        fork;
        absorb tryte pk[3072];
        ntrukem(key) ekey;
      };
    } explicit_keyload = 1;
  };
  absorb external tryte key[81];
  commit;
  mask trytes payload;
  commit;
  squeeze external tryte hash[78];
  mssig(hash) sig;
}
```

Protobuf3 grammar

```
Message := ‘message‘ MIdent ‘{‘ [Statement] ‘}‘
Statement := CmdModifier* { Command | MIdent FIdent } ‘;‘
Command := SpongosCmd | PublicKeyCmd
SpongosCmd := ‘absorb‘ Arg | ‘squeeze‘ Arg
  | ‘mask‘ Arg | ‘commit‘ | ‘fork‘ | ‘join‘ Arg
PublicKeyCmd := ‘mssig(‘ FIdent ‘)‘ Arg
  | ‘ntrukem(‘ FIdent ‘)‘ Arg
CmdModifier := ‘oneof‘ | ‘repeated‘
ArgModifier := ‘external‘
Arg := ArgModifier Type FIdent;
Type := ‘tryte‘ | ‘size_t‘ | ‘trytes‘ | ‘link‘
  | ‘tryte[‘ Size ‘]‘
MIdent := identifier
FIdent := identifier
Size := constant | FIdent
```

Application layer

- A MAM Application is a cryptographic protocol.
- Application party:
  - keeps secrets,
  - trusts public keys,
  - processes messages,
  - implements application-specific logic.
- `Message` format is fixed for all Applications.
- `Content` is defined by `Header.type`.

```
message Message {
  Header header;
  Content content;
  commit;
}
```

```
message Header {
  absorb tryte version;
  absorb external tryte
      appinst[81];
  absorb external tryte
      msgid[27];
  absorb trytes type;
}
```

Future work

- Formal Protobuf3 spec, nicer API.
- Rust implementation:
  - fix current safety issues,
  - code review,
  - implement faster and more compact trit encodings;
  - implement MT traversal for MSS.
- New crypto: multi/group/ring signatures, NTRU sig, SIDH, SIKE.
- New Applications: `Multichat`, `DID`, `DDS`?

Overview

Author:

- has MSS keys, optional NTRU key and PSK keys;
- announces channel, changes MSS keys;
- publishes signed packets, can publish tagged packets;
- maintains a set of trusted Subscribers;
- publishes keyloads.

Subscriber:

- has optional NTRU key and PSK keys;
- trusts Author;
- can request subscription;
- can publish tagged packets;
- fetches and verifies signed/tagged packets.

Announce Message

- Published by Author, announces creation of the application instance.
- Contains Author's MSS public key and optionally NTRU public key.
- Subscribers must *t*rust Author's MSS public key.
- Signed with the current MSS private key.

```
message Announce {
  absorb tryte msspk[81];
  absorb oneof {
    null empty = 0;
    tryte ntrupk[3072] = 1;
  }
  commit;
  squeeze external tryte hash[78];
  mssig(hash) sig;
}
```

ChangeKey Message

- Contains Author's another MSS public key.
- Linked to an `Announce` or `ChangeKey` message.
- Signed with the current and linked MSS private keys.

```
message ChangeKey {
  join link msgid;
  absorb tryte msspk[81];
  commit;
  squeeze external tryte hash[78];
  mssig(hash) sig_with_msspk;
  mssig(hash) sig_with_linked_msspk;
}
```

Keyload Message

- Contains session key material for a set of Subscribers.
- Published by either Author or Subscriber.
- Can be linked to any message.
- Subscribers are identified by either PSK id or NTRU public key id.
- If linked to another `Keyload` message, then key identities are masked.

```
message Keyload {
    join link msgid;
    absorb tryte nonce[27];
    skip repeated {
        fork;
        mask tryte id[27];
        absorb external tryte psk
            [81];
        commit;
        mask(key) tryte ekey[81];
    }
    skip repeated {
        fork;
        mask tryte id[27];
        ntrukem(key) tryte ekey
            [3072];
    }
    absorb external tryte key[81];
    commit;
}
```

SignedPacket Message

- Published by Author.
- Linked to any message.
- Contains both public and masked payloads.
- Signed with linked MSS private key.

```
message SignedPacket {
    join link msgid;
    absorb trytes public_payload;
    mask trytes masked_payload;
    commit;
    squeeze external tryte hash[78];
    mssig(hash) sig;
}
```

TaggedPacket Message

- Published by either Author or Subscriber.
- Linked to any message.
- Contains both public and masked payloads.
- Authenticated with secret session key.

```
message TaggedPacket {
    join link msgid;
    absorb trytes public_payload;
    mask trytes masked_payload;
    commit;
    squeeze tryte mac[81];
}
```

Subscribe Message

- Published by Subscriber as request to subscribe.
- Contains Subscriber's masked NTRU public key.
- Linked to the `Announce` message.
- Author must *t*rust Subscriber's NTRU pk before sharing keyload for it.

```
message Subscribe {
    join link msgid;
    ntrukem(key) tryte unsub_key[3072];
    commit;
    mask tryte ntrupk[3072];
    commit;
    squeeze tryte mac[27];
}
```

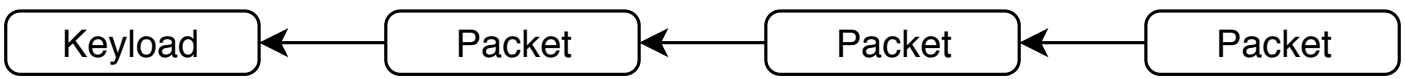
Unsubscribe Message

- Published by Subscriber as request to unsubscribe.
- Linked to `Subscribe` message.
- Author revokes Subscriber's NTRU pk after verifying MAC.

```
message Unsubscribe {
    join link msgid[27];
    commit;
    squeeze tryte mac[27];
}
```

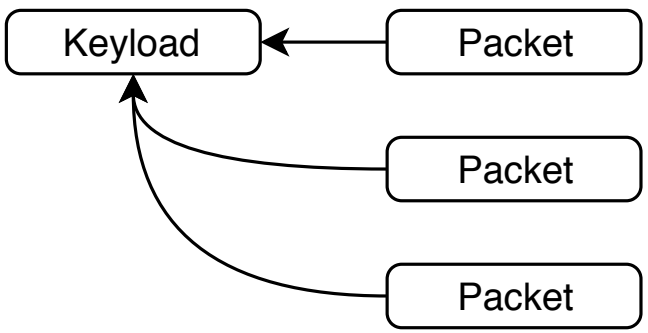
List-like topology

- The next `Packet` message is linked to the previous `Packet`.
- Subscriber must only maintain two `spongos` instances — current and linked — in order to wrap and unwrap packets.
- Suitable for restricted Authors wishing to conserve MSS keys and only sign few packets. All previous tagged packets also become implicitly authenticated after publishing a signed packet.
- Linear-time random access to a packet.



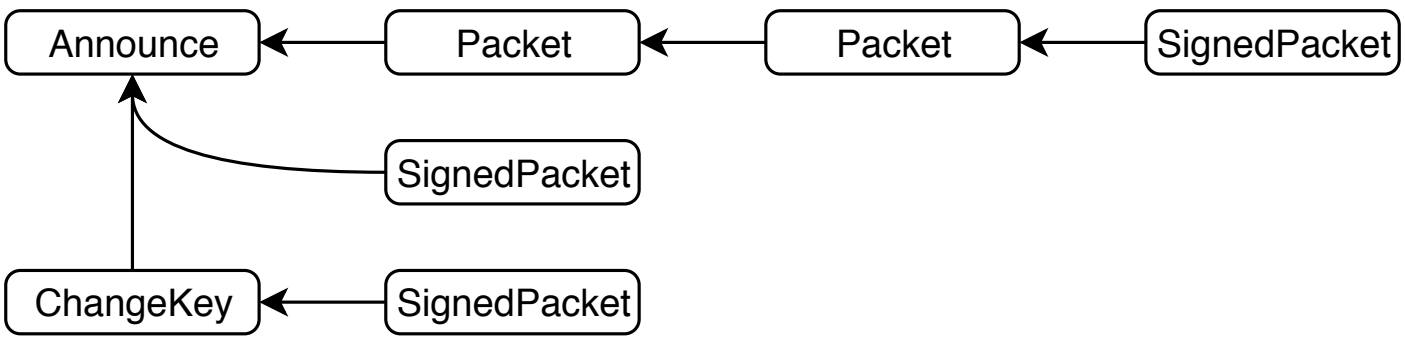
Set-like topology

- The next `Packet` message is linked to the `Keyload` message.
- Subscriber must only maintain two `spongos` instances — current and associated to keyload.
- Constant-time random access to a packet.



Public channel topology

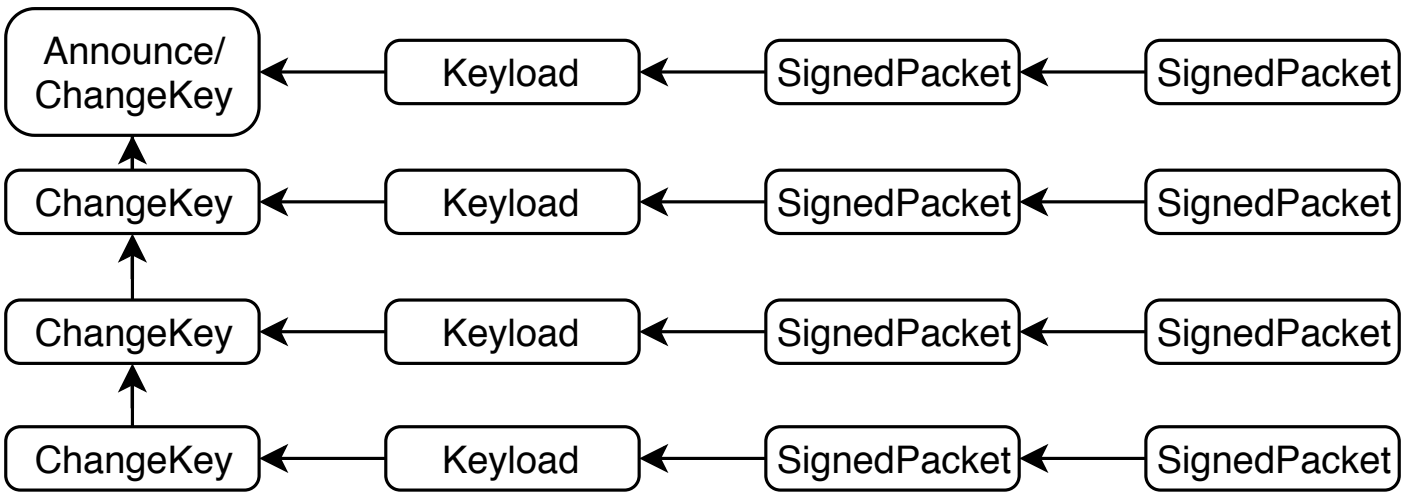
- No `Keyload` messages, all messages are public.
- Subscriber does not need to be subscribed.
- Both tagged and signed packets are published by the Author [1].
- Tagged messages are not authenticated, thus Author must end each chain of tagged messages with a signed one in order to authenticate the whole chain.



[1] Subscribers simply do not have key material to authenticate messages.

Short MT topology

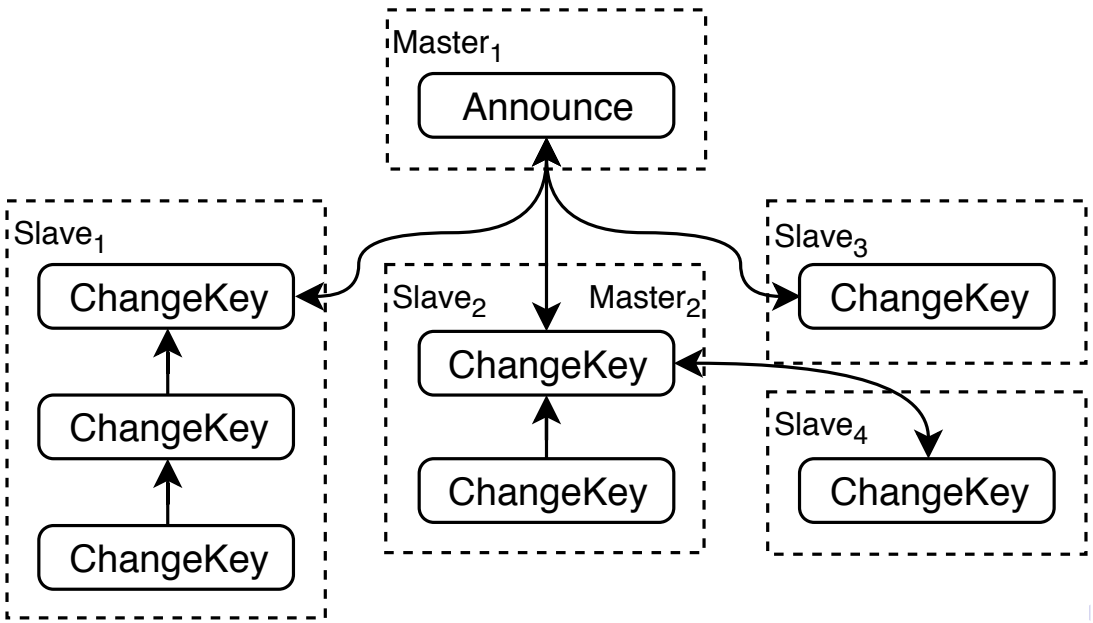
- Trades off heavy resource requirements imposed by Merkle tree implementations with more frequent MSS key changes.
- A restricted Author maintains only one Merkle tree of small height [2].



[2] To be productive Merkle tree must contain at least three leaves as two leaves are used in `ChangeKey` messages: one key is used to prove possesion of private key when public key is published, and another one is used to sign a new public key.

Distributed Author topology

- Author wishes to employ several devices to publish signed packets.
- One device is selected to be Master, it generates several new MSS private keys and publishes them in `ChangeKey` messages.
- Master then distributes MSS private keys among Slave devices via a secure channel.
- Slave devices can now publish signed packets, as well as establish their own hierarchy.



Keyload-based access control

- Keyload restricts access to following Packets for a group of Subscribers.
- Alice and Bob — in group AB, Cindy and Dan — in group CD.
- PSK-based Keyload is shorter, more convenient for more frequent session key changes.