```
BIP: 150
Layer: Peer Services
Title: Peer Authentication
Author: Jonas Schnelli <dev@jonasschnelli.ch>
Comments-Summary: Discouraged for implementation (one person)
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0150
Status: Draft
Type: Standards Track
Created: 2016-03-23
License: PD
```

Abstract

This BIP describes a way for peers to authenticate to other peers to guarantee node ownership and/or allow peers to access additional or limited node services, without the possibility of fingerprinting.

Motivation

We assume peer operators want to limit the access of different node services or increase datastream priorities to a selective subset of peers. Also we assume that peers want to connect to specific peers to broadcast or filter transactions (or similar actions that reveal sensitive information) and therefore operators want to authenticate the remote peer and ensure that they have not connected to a MITM (man-in-the-middle) attacker.

Benefits of peer authentication:

- Peers can detect MITM attacks when connecting to known peers
- Peers can allow resource hungry transaction filtering only to specific peers
- Peers can allow access to sensitive information that can lead to node fingerprinting (fee estimation)
- Peers can allow custom message types (private extensions) to authenticated peers

A simple authentication scheme based on elliptic cryptography will allow peers to identify each other and selectively allow access to restricted services or reject the connection if the peer identity cannot be verified.

Specification

The authentication scheme proposed in this BIP uses ECDSA, **secrets will never be transmitted**.

Authentication initialization must only happen if encrypted channels have been established (according to BIP-151 [1]).

The **encryption-session-ID** is available once channels are encrypted (according to BIP-151 [1]).

The identity-public-keys used for the authentication must be pre-shared over a different channel (mail/PGP, physical paper exchange, etc.). This BIP does not cover a "trust on first use" (TOFU) concept.

The authentication state must be kept until the encryption/connection terminates.

Only one authentication process is allowed per connection. Re-authentication require re-establishing the connection.

Known-peers and authorized-peers database

Each peer that supports p2p authentication must provide two user-editable "databases".

1. **known-peers** contains known identity-public-keys together with a network identifier (IP & port), similar to the "known-host" file supported by openssh.
2. **authorized-peers** contains authorized identity-public-keys

Local identity key management

Each peer can configure multiple identity-keys (ECC, 32 bytes). Peers should make sure that each network interface (IPv4, IPv6, tor) has its own identity-key (otherwise it would be possible to link a tor address to a IPvX address). The identity-public-key(s) can be shared over a different channel with other node-operators (or non-validating clients) to grant authorized access.

Authentication procedure

Authentication based on this BIP will require both sides to authenticate. Signatures/public-keys will only be revealed if the remote peer can prove that they already know the remote identity-public-key.

1. -> Requesting peer sends `AUTHCHALLENGE` (hash)
2. <- Responding peer sends `AUTHREPLY` (signature)
3. -> Requesting peer sends `AUTHPROPOSE` (hash)
4. <- Responding peer sends `AUTHCHALLENGE` (hash)
5. -> Requesting peer sends `AUTHREPLY` (signature)

For privacy reasons, dropping the connection or aborting during the authentication process must not be allowed.

`AUTHCHALLENGE` message

A peer can send an authentication challenge to see if the responding peer can produce a valid signature with the expected responding peer's identity-public-key by sending an `AUTHCHALLENGE`-message to the remote peer.

The responding peer needs to check if the hash matches the hash calculated with his own local identity-public-key. Fingerprinting the requesting peer is not possible.

Field Size	Description	Data type	Comments
32bytes	challenge-hash	hash	hash(encryption-session-ID

`challenge_type` is a single character. `i` if the `AUTHCHALLENGE`-message is the first, requesting challenge or `r` if it's the second, remote peers challenge message.

`AUTHREPLY` message

A peer must reply an `AUTHCHALLENGE`-message with an `AUTHREPLY`-message.

Field Size	Description	Data type	Comments
64bytes	signature	normalized comp.-signature	A signature of the encryption-session-ID done with t

If the challenge-hash from the `AUTHCHALLENGE`-message did not match the local authentication public-key, the signature must contain 64 bytes of zeros.

The requesting peer can check the responding peer's identity by checking the validity of the sent signature against with the pre-shared remote peers identity-public-key.

If the signature was invalid, the requesting peer must still proceed with the authentication by sending an `AUTHPROPOSE`-message with 32 random bytes.

`AUTHPROPOSE` message

A peer can propose authentication of the channel by sending an `AUTHPROPOSE`-message to the remote peer.

If the signature sent in `AUTHREPLY` was invalid, the peer must still send an `AUTHPROPOSE`-message containing 32 random bytes.

The `AUTHPROPOSE` message must be answered with an `AUTHCHALLENGE`-message - even if the proposed requesting-peers identity-public-key has not been found in the authorized-peers database. In case of no match, the responding `AUTHCHALLENGE`-message must contains 32 bytes of zeros.

Field Size	Description	Data type	Comments
32bytes	auth-propose-hash	hash	hash(encryption-session-ID

Post-Authentication Re-Keying

After the second `AUTHREPLY` message (requesting peer's signature -> responding peer), both clients must re-key the symmetric encryption according to BIP151 while using **a slightly different re-key key derivation hash**.

Both peers re-key with `hash(encryption-session-ID || old_symmetric_cipher_key || requesting-peer-identity-public-key || responding-peer-identity-public-key)`

Identity-Addresses

The peers should display/log the identity-public-key as an identity-address to the users, which is a base58-check encoded ripemd160(sha256) hash. The purpose of this is for better visual comparison (logs, accept-dialogs). The base58check identity byte is `0x0F` followed by an identity-address version number (=`0xFF01`).

An identity address would look like `TfG4ScDgysrSpodWD4Re5UtXmcLbY5CiUHA` and can be interpreted as a remote peer's fingerprint.

Compatibility

This proposal is backward compatible. Non-supporting peers will ignore the new `AUTH*` messages.

Example of an auth interaction

Before authentication (once during peer setup or upgrade)

1. Requesting peer and responding peer create each an identity-keypair (standard ECC priv/pubkey)
2. Requesting and responding peer share the identity-public-key over a different channel (mail/PGP, physical paper exchange, etc.)
3. Responding peer stores requesting peers identity-public-key in its authorized-peers database (A)
4. Requesting peer stores responding peers identity-public-key in its known-peers database together with its IP and port (B)

Encryption

1. Encrypted channels must be established (according to BIP-151 [1])

Authentication

1. Requesting peer sends an `AUTHCHALLENGE` message

```
 AUTHCHALLENGE:
[32 bytes, hash(encryption-session-ID || "i" || )]
```

1. Responding peer does create the same hash `(encryption-session-ID || "i" || )` with its local identity-public-key

2. If the hash does not match, response with an `AUTHREPLY` message containing 64bytes of zeros.
3. In case of a match, response with an `AUTHREPLY` message

```
 AUTHREPLY:
[64 bytes normalized compact ECDSA signature (H)] (sig of the encryption-session-ID done wit
```

1. Requesting peer does verify the signature with the `remote-peers-identity-public-key`
2. If the signature is invalid, requesting peer answers with an `AUTHREPLY` message containing 32 random bytes
3. In case of a valid signature, requesting peer sends an `AUTHPROPOSE` message

```
 AUTHPROPOSE:
[32 bytes, hash(encryption-session-ID || "p" || )]
```

1. Responding peer iterates over authorized-peers database (A), hashes the identical data and looks for a match.
2. If the hash does not match, responding peer answer with an `AUTHCHALLENGE` message containing 32 bytes of zeros.
3. In case of a match, responding peer sends an `AUTHCHALLENGE` message with the hashed client public-key

```
 AUTHCHALLENGE:
[32 bytes, hash(encryption-session-ID || "r" || )]
```

1. Requesting peer sends an `AUTHREPLY` message containing 64 bytes of zeros if server failed to authenticate
2. Otherwise, response with signature in the `AUTHREPLY` message

```
 AUTHREPLY:
[64 bytes normalized compact ECDSA signature (H)] (sig of the encryption-session-ID done wit
```

1. Responding peer must verify the signature and can grant access to restricted services.
2. Both peers re-key the encryption after BIP151 including the requesting-peer-identity-public-key and responding-peer-identity-public-key

Disadvantages

The protocol may be slow if a peer has a large authorized-peers database due to the requirement of iterating and hashing over all available authorized peer identity-public-keys.

Reference implementation

References

- [1] BIP 151: Peer-to-Peer Communication Encryption

Acknowledgements

- Gregory Maxwell and Pieter Wuille for most of the ideas in this BIP.
- Bryan Bishop for editing.

Copyright