Privacy-Preserving Decentralized Micropayments

Onion Routing in Lightning

Olaoluwa Osuntokun
roasbeef
laolu@lightning.network

Lightning Labs

Scaling Bitcoin Milan, 2016

Overview - Onion Routing

- Distributed set of Onion Routers (*OR*) [1]
- Users create *circuits* with sub-set of nodes
- Difficult for *OR*'s to gain more info than predecessor+successor in path
- Low Latency – usable within greater Internet
- Notable success: Tor: 2nd Generation Onion Router [2]

Domain Application - Lightning

- Goals: privacy + censorship resistance
- Each node in network doubles as an *OR*
- Source routing: sender *fully* specifies route payments take:
  - Path length
  - Absolute time-locks (*CLTV*)
  - Fees at each hop
  - Inter-chain links

Sphinx

- Provably secure Mix Format [3]
  - Header: routing instructions
  - Body: end-to-end message
- Sphinx header+body is *re-obsfucated* at each hop
  - Intermediate *OR*'s unable to *distinguish* one packet from another (**IND-CPA**)
- Entire packet remains *fixed-sized* through processing
  - Intermediate *OR*'s gain no *positional* informaiton

Sphinx - Key Agreement

- Sender needs to derive unique *shared-secret* for each hop
    - Used to encrypt+authenticate packet fields
- To achieve unlink-ability, group-element for DH need to change *at each hop*
- Past solutions: include N group-elements within packet, one for each hop
- Sphinx's solution: repeatedly *blind* (randomize) a *single* group element

This One Little Trick Drives Adversaries Insane!

Source derives unique session key: x
Source obtains list of OR pubkeys: $\{N_1, N_2, N_3, ...N_i\}$
Source computes the per-hop group element:

$$a_0 = g^x \qquad s_0 = N_1^x \qquad b_0 = h(a_0, s_0)$$

$$a_1 = g^{x^{b_0}} \qquad s_1 = N_2^{x^{b_0}} \qquad b_1 = h(a_1, s_1)$$

$$a_2 = g^{x^{b_0^{b_1}}} \qquad s_2 = N_2^{x^{b_0^{b_1}}} \qquad b_2 = h(a_2, s_2)$$

$$\ldots$$

Each hop re-blinds (b_i) the group-element for their successor based on the random group-element (a_i) and shared secret (s_i)!

Packet Processing

ProcessSphinxPacket (packet)

$a, header, MAC, body \leftarrow packet$

$s \leftarrow a^x \quad \star$

$\hat{MAC} = MAC(s, header || body)$

if $\hat{MAC} != MAC$

$\quad REJECT$

endif

$\hat{a} \leftarrow a^{h(s,a)} \quad \star$

$nextHop, \hat{packet} = parse(s, header, body)$

return $nextHop, \hat{packet}$

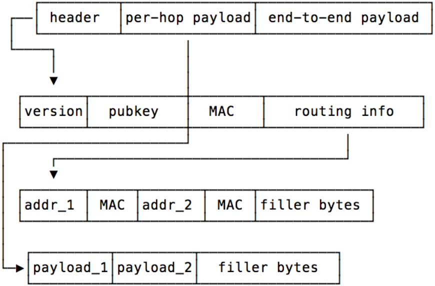
$\star$ indicates an asymmetric cryptographic operation

parse shifts the bytes, and decrypts a layer from header+body

Sphinx Modifications

Onion Routing spec draft, led by Christian Decker [4]

- Addition of version-byte to header
- Packet now contains a *per-hop* payload
- *Entire* packet protected under MAC
- Switch from AES-SHA256 to ChaCha20-Poly1305



Performance Considerations

Two asymmetric crypto operations now in *critical-path* for forwarding:

1. DH operation to derive shared-secret
2. Exponentiation/Scalar-Multiplication to re-blind group-element

OR's need to maintain *per-session* circuit state

- Circuit: payment hash, incoming link, outgoing link
- Needs to be persisted to disk to survive restarts

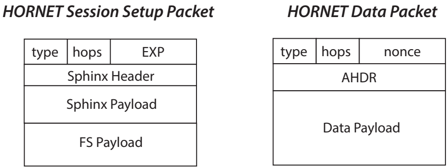
Overview

- Progression of Onion Routing to achieve efficient *internet-scale* data forwarding [5]
- Eliminates asymmetric crypto operations during data-forwarding
- Creates a *bi-directional* ephemeral circuit during set up

HORNET's Optimizations

Constructs an Onion Circuit with a double-pass:

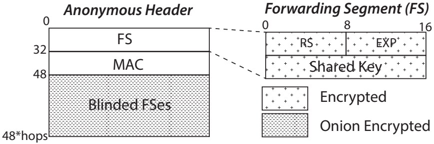
- Sphinx used for session initialization
- Intermediates *OR*'s populate a *Forwarding Segment*
- Allows for forward secrecy within set-up phase



HORNET's Optimizations

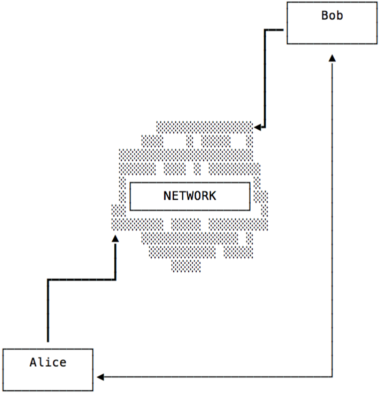
Circuit state pushed to the *endpoints*:

- HORNET packets carry information *complete* necessary for forwarding
- Node state reduced to *SV* symmetric key ($O(1)$ storage)
- Solely *symmetric* cryptography used for data forwarding



In-Network Payment Negotiation

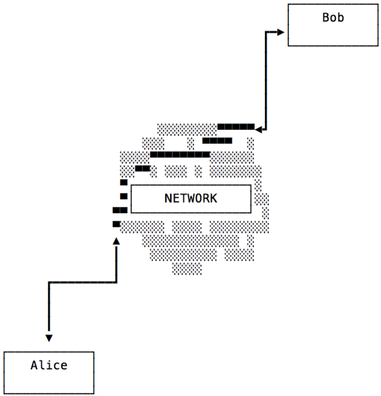
Currently, payment negotiation assumed to be out-of-band:



In-Network Payment Negotiation

With HORNET, payment negotiation can be done *purely* over the network

- Reduces payment info to simply: $< nodePubKey >$
- Ideal for the *streaming* payment setting!
- Additional payment hashes exchanged over HORNET circuit
- Streamlines possible network layer payment *fragmentation*



Shared Secret Log

- Without replay protection, packets can be *re-injected* into the network, possibly leaking route information.
- Solution: remember *all* past shared secrets, rejecting "double-spends".
- Problem: log of shared secrets grows *unbounded*

If we periodically rotate keys, can garbage collect prior log entries!

Active Key Rotation

- Use the network communication layer (irc, broadcast, etc) to advertise new keys
- Key advertisements authenticated via current *identity* key
- Advertise staggered overlapping windows to allow loosely synchronized rotation

Active rotation incurs additional bandwidth overhead, can we eliminate this?

Passive Key Rotation - First Attempt

- Using BIP 32 Public Derivation, the *edges* can rotate passively
- Initially communicate:
  - Master Public Key (MPK)
  - Anchor: $< blockhash >$
- Edges then *passively* rotate keys (eg, every 144 blocks from anchor)

However, compromise of child priv key, and MPK defeats forward secrecy!

Passive Key Rotation - Second Pass

Using pairing cryptography, we can achieve non-interactive passive key rotation![6]

We modify the Boneh-Franklin Identity Based Encryption (BF-IBE)[7] scheme to our domain:

- Three cyclic groups: $\mathbb{G}$, $\hat{\mathbb{G}}$, $\mathbb{G}_T$ for prime order q.
- A *bilinear pairing*: $e : \mathbb{G} \times \hat{\mathbb{G}} \rightarrow \mathbb{G}_T$: $e(g^a, \hat{g}^b) = e(g, \hat{g})^{ab}$
- Each node generates a master secret: s, and advertises $y = g^s$

Passive Key Rotation - Second Pass

Given H a *full-domain* hash function: $H : \{0, 1\}^* \rightarrow \hat{\mathbb{G}}\star$

Rotation:

- Assuming a BF-IBE setting: $ID = H(blockHash)$
- Each nodes is it's own Private Key Generator (PKG)
- PKG extraction: $n = ID^s = H(blockHash)^s$

Key agreement:

- Given Sphinx pseudonym: $R = g^r$ (recall our "little trick")
- Source derives secret: $e(y, ID)^r$, node derives secret: $e(R, n)$
- $e(y, ID)^r = e(g^s, H_{id})^r = e(R, n) = e(g^r, H_{id}{}^s) = e(g, H_{id})^{rs}$

Limitations

- In practice, security relies on high-degree of path diversity
- Additional correlation possible via payment values, link capacities, etc.
- Active network analysis via timing attacks, packet sizes, etc.

Future Directions

- Integrate HORNET
  - Sphinx (a prerequisite) currently implemented in *lnd* and *lightningd*
- Per-hop payload structure: inter-chain, timeouts, amounts, etc.
- Investigation into alternative higher-latency systems: mix-net, DC-net, etc.
- Non-source-routed privacy schemes

[Goldschlag et al., 1997] D. M. Goldschlag, M. Reed, and P. Syverson
Hiding Routing Information, 1997.

[The Tor Project, 2016]
https://www.torproject.org, 2016.

[Sphinx: A Compact and Provably Secure Mix Format, 2009] G. Danezis and I. Goldberg
IEEE Symposium on Security and Privacy, 2009.

https://github.com/cdecker/lightning-rfc/blob/master/bolts/onion-protocol.md

[HORNET: High-speed Onion Routing at the Network Layer, 2015] Chen Chen et al.
ACM CCS, 2015.

[Pairing-Based Onion Routing, 2007] A. Kate, G. M. Zaverucha, and I. Goldberg
PETS '07, 2007.

[Identity-Based Encryption from the Weil Pairing, 2001] D. Boneh and M. Franklin
Advances in Cryptology - CRYPTO'01, 2001.