HOPR - a Decentralized and Metadata-Private Messaging Protocol with Incentives

Dr. Sebastian Bürgel, Robert Kiel

November 2019, V1

0.1 Sphinx Packet Format

A sphinx packet consists of two parts:

1. Header:
    - Key derivation
    - Routing information
    - Integrity protection
2. Body:
    - Onion-Encrypted payload

Notation Let k be a security parameter. An adversary will have to do about 2^k work to break the security of Sphinx with non negligible probability. We suggest using $k = 128$. Let r be the maximum number of nodes that a Sphinx mix message will traverse before being delivered to its destination. G is a prime order cyclic group satisfying the Decisional Diffie-Hellman Assumption. The element g is a generator of G and q is the (prime) order of G, with $q \approx 2^k$. G^* is the set of non-identity elements of G. h_b is a hash function which we model by random oracles such that: $h : G^* \times G^* \to \mathbb{Z}_q^*$ where $\mathbb{Z}_q^*$ is the field of non-identity elements of $\mathbb{Z}_q$ (field of integers). Each node $n \in \mathbb{N}$ has a private key $x_n \in \mathbb{Z}_q^*$ and a public key $y_n = g^{x_n} \in G^*$ where $\mathbb{N} \subset \{0,1\}^k$is a set of mix nodes identifiers.

Key derivation The sender (A) picks a random $x \in \mathbb{Z}_q^*$ that is used to derive new keys for every packet.
(A) randomly picks a path consisting of intermediate nodes (B), (C),(D) [see section path-finding] and the final destination of the packet (E)
(A) performs an offline Diffie-Hellman key exchange with each of these nodes

and derives shared keys with each of these nodes.
(A) computes a sequence of r tuples (in our case r=4)

$$(a_0, s_0, b_0), ..., (a_{r-1}, s_{r-1}, b_{r-1})$$

as follows:

- $a_0 = g^x, s_0 = y_B^x, b_0 = h(a_0, s_0)$
- $a_1 = g^{xb_0}, s_1 = y_C^{xb_0}, b_1 = h(a_1, s_1)$
- $a_2 = g^{xb_0b_1}, s_2 = y_D^{xb_0b_1}, b_2 = h(a_2, s_2)$

Where y_B, y_C, y_D, y_E are the public keys of the nodes B, C, D which we assume are available to A . The a_i are the group elements which, when combined with the nodes' public keys, allows computing a shared key for each via Diffie-Hellman key exchange, and so the first node in the user-chosen route can forward the packet to the next, and only that mix-node can decrypt it. The s_i are the Diffie Hellman shared secrets, and the b_i are the blinding factors.

Routing information Each node on the path needs to know the next downstream node. Therefore, the sender (A) generates routing information β_i for (B), (C) and (D) as well as message END to tell (E) that it is the final receiver of the message.
As (A) has a shared secret with each of the nodes along the path, it is able to derive blindings for each of them which is symbolised as different hatchings.
Once (B) receives the packet, it derives the shared key s_0(for simplicity we call it s_B as it is the shared key with B) by computing

$$s_0 = (a_0)^b = (g^x)^b = (g^b)^x = y_B^x$$

and removes its blindings. This allows (B) to unblind the routing info that tells (B) the public key of the next downstream node (C). The unblinding works as follow:

1. B computes the keyed-hash of the encrypted routing information β_0 as $HMAC(s_0, \beta_0)$ and compares with the integrity tag γ_0 attached in the packet header. If the integrity check fails because the header has been tampered with, the packet is dropped. Otherwise, the mix-node proceeds to step 2.

2. B is now ready to decrypt the attached β_0. In order to extract the routing instructions, the mix-node B first appends a zero-byte padding at the end of β_0 and decrypts the padded block of routing information B by XORing it with $(h(s_0))$. Where $\varrho : \{0,1\}^k \rightarrow \{0,1\}^{(2r+3)k}$ is a pseudo-random generator (PRG) and $h_\varrho : G^* \rightarrow \{0,1\}^k$ is a hash function used to key ϱ .

3. (B) parses the routing instructions from (A) in order to obtain the address of the next mix-node (C), as well the new integrity tag γ_1 and β_1, which should be forwarded to the next hop.

4. (B) blinds the key share $a_0 = g^x$ by setting it to $a1 = g^{xb_0}$.

(B) also removes one layer of encryption from the payload. The payload δ_0 in the Sphinx packet is computed using a wide-block cipher to ensure that, if an adversary modifies the payload at any point, the message content is irrecoverably lost.
Once (C) receives the packet, it derives the shared key s_c and removes the blinding, extracts the public key of (D) and deletes the routing information from the packet. Afterwards, it fills the empty space with its own blinding which is different from the one of (B).
Same happens at (D): key derivation, unblinding, deleting, shifting, decryption and blinding.
Last but not least, the packet arrives at (E), the final destination of the packet. Like the other nodes, (E) first derives its shared key s_E and removes the blinding. In contrast to the previous nodes, namely (B), then (C), then (D), it finds a message that symbolizes the end of path and tells (E) that it's the recipient.

Integrity In addition to the routing information that is necessary to determine to which the transformed packet should be sent, each node along the path receives an authentication tag δ_i in the form of a message authentication code (MAC).
By using the derived shared secret, e.g. s_b, each node is able to recompute the authentication tag and check the integrity of the received packet as follows:

$$\delta_0 = HMAC(s_0, \beta_0)$$

The integrity tag encoded in the header allows checks only on whether the packet header was not also modified. The authentication tags are computed by the sender of the packet and cover also the blindings that nodes add to the packet after transforming it for the next downstream node. This is possible as the sender knows all shared secrets and the blindings are generated pseudorandomly by using the shared secrets.