Alexander Bajic and Georg T. Becker*

dPHI: An improved high-speed network-layer anonymity protocol

Abstract: The Internet infrastructure has not been built with security or privacy in mind. As a result, an adversary who has control over a single Autonomous System can set-up mass surveillance systems to gather meta data by passively collecting the headers of the messages they route. To solve this problem, lightweight anonymous routing protocols such as LAP, DOVETAIL and most recently PHI have been proposed which are efficient enough to be deployed in a large scale infrastructure such as the Internet. In this paper we take a closer look at PHI and introduce several de-anonymization attacks malicious nodes can perform to reduce the sender and receiver anonymity. As a direct consequence of this analysis we propose a new protocol called *dependable PHI* (dPHI). The security analysis of dPHI includes a detailed quantitative anonymity analysis that compares dPHI with PHI, LAP and HORNET. Together with the performance analysis, this allows for a good comparison of trade-offs for these anonymity protocols.

Keywords: Anonymous routing, network security, mass-surveillance, Internet infrastructure

DOI 10.2478/popets-2020-0054
Received 2019-11-30; revised 2020-03-15; accepted 2020-03-16.

1 Introduction

Most communication networks, and in particular the Internet, do not hide who is communicating with whom at what time. However, from a privacy perspective, such meta data are very critical as a lot information can be inferred from them. This is corroborated by the fact that most mass-surveillance programs aim for such data. An entity controlling an AS can store meta data — who communicated with whom and at what time and which service — of all communication going through that AS. Furthermore, a malicious AS can manipulate the Border Gateway Protocol (BGP) to increase the traffic it can eavesdrop [2, 30] on. Technical changes to the Internet infrastructure and the used protocols are therefore advocated by privacy-conscious actors [28]. The most famous initiative to counter surveillance is the Tor project [27] in which traffic is encrypted and relayed through hops to obfuscate the actual communication partners using onion routing. While Tor is still subject to traffic analysis and other attacks (e.g. [7, 16, 23]), it offers a good degree of privacy for the average user. However, the introduced overhead does not make it a viable solution to be used by every client and application. Tor's speed is often insufficient [11] and its scalability is limited [1, 21]. Anonymous communication protocols with provable anonymity guarantees such as Mixnet-based systems exhibit an even worse performance than Tor [10]. For comprehensive surveys of such protocols see the work of Shirazi *et al.* [26] or Ren and Wu [24].

To tackle this problem, lightweight anonymity protocols have been proposed which focus on optimizing performance and deployment costs with the goal to allow large-scale utilization at the network layer. The Lightweight Anonymity and Privacy (LAP) protocol [14] is the first proposed protocol in this category. It is deployed at the network layer on top of the Internet Protocol and offers sender anonymity. Dovetail [25] builds upon LAP and adds some receiver anonymity but requires user-controlled pathlet routing. HORNET [8] offers the highest degree of anonymity, yet requires client-based routing. The Path-hidden lightweight anonymity protocol (PHI) [9] builds upon the LAP and Dovetail protocol. Like LAP, PHI works with any inter-domain routing protocol such as BGP. A more detailed description of these protocols can be found in Section 2.

From a privacy and security perspective one would hope that a future Internet infrastructure has solid privacy protection built in. Performance requirements and costs are limiting factors for the wide deployment of anonymity preserving protocols. It is, therefore, important to investigate how anonymity can be added with as little cost as possible. Ultimately, whether or not anonymity protocols will be deployed in a future Internet is a political question. However, anonymity proto-

Alexander Bajic: Digital Society Institute, ESMT Berlin, E-mail: alexander.bajic@esmt.org
***Corresponding Author: Georg T. Becker:** Digital Society Institute, ESMT Berlin, E-mail: georg.becker@rub.de

cols are not only interesting from a privacy perspective but also for network security. In large corporate networks they constitute a promising approach to decrease information leakage in case of compromised infrastructure. This, in turn, can hamper lateral movement, thus impeding attacks such as Advanced Persistent Threats (APT). With the rise of software-defined networking (SDN), implementing these protocols in IP networks has become very realistic and cost-efficient.

1.1 Contribution

The contribution of this paper is twofold. First, we present novel de-anonymization attacks against PHI that significantly decrease receiver and sender anonymity. Second, we propose an improved lightweight anonymity protocol named dependable PHI (dPHI) that withstands these attacks. In summary, we provide the following contributions:

- Several de-anonymization attacks against PHI are presented that reduce the sender and receiver anonymity set size by observing and manipulating headers during session establishment.
- A new protocol named dependable PHI (dPHI) is proposed that withstands these attacks even in an extended threat model that is more realistic.
- Besides a careful security analysis of dPHI, we perform a quantitative anonymity analysis comparing PHI, dPHI, LAP and HORNET. The Matlab code of the analysis will be publicly available to make our research reproducible and extendable.
- A performance analysis based on an implementation in C shows that dPHI has similar throughput, goodput and latency as PHI while requiring less overall public key operations in the network.

2 Related work

In the following, we will introduce the most important lightweight anonymous routing protocols relevant for PHI and dPHI. LAP and HORNET will also be considered in the quantitative analysis in Section 8, and in the performance analysis in Section 10.

LAP is the first network layer lightweight anonymity protocol and was presented in 2012 [14]. The goal of LAP is to provide sender anonymity while not impeding the routing of the network it is deployed in. The session establishment works as follows: A session request is sent from source to destination. In the session request message, a header field is used by each routing node to store information on how it routed the message. That is, each node stores the ingress port (from where the message has been received) and the egress port (to where the message is forwarded), encrypted with its individual secret key. This way, routing information can only be retrieved by the routing node that wrote the information to prevent other nodes from learning from where the message originated. After the session request messages reaches the destination, all further messages are exchanged based on the encrypted routing information in the header. Each routing node receiving a message decrypts the routing information and sends the message to the corresponding ingres or egres port.

This way an attacker eavesdropping a message does not learn the source address. However, an attacker can determine the distance to the source by counting the routing elements in the header. To obfuscate this information, LAP has the option to employ a variable sized routing segment (VSS) with a parameter we denote as VSS (in LAP [14] denoted as M). In this case, for each routing element a random number (that does not exceed the VSS threshold) of dummy entries is added. This way the path length is obfuscated to a certain degree.

Dovetail was proposed two years after LAP by Sankey and Wright [25] and builds upon the basic idea of storing the routing information encrypted in the header. However, it adds some form of receiver anonymity by introducing a helper node to which session establishment requests are sent. The real destination of a message is encrypted with the helper nodes public key and, upon arrival at the helper node, decrypted and inserted into the message's header. The message is then forwarded to the destination via a tail node that lies on both the path from the source to the helper node and from the helper node to the destination. One main difference to LAP (and PHI) is that Dovetail assumes pathlet routing, which is a client-controlled scheme. Hence, it interferes with any routing policy employed by the underlying infrastructure. Furthermore, the number of nodes on a path is not hidden in Dovetail so that an attacker can learn the distance to source or destination. Dovetail notes that one should therefore not use a shortest path routing, thus obfuscating the real distance by sometimes using longer paths than necessary.

HORNET [8] employs a client-based routing scheme in which the client chooses a path to the destination. For each routing node on the path, the client encrypts information on how to forward messages, using the respective

routing node's public key, and stores it in the session establishment message. Each routing node receiving the session establishment message decrypts its corresponding routing information and re-encrypts it with a secret symmetric key and stores it in the header. After session establishment, messages are exchanged as done in LAP and Dovetail using these encrypted routing information fields. Each routing node only learns its predecessor and successor on the path, offering the highest anonymity of the discussed protocols. However, in the Internet, client-based routing can undermine routing policies employed by intermediate nodes to e.g. balance load. This makes adopting such routing policies unlikely. Besides anonymity, HORNET also introduces default payload encryption from hop to hop.

For completeness' sake, we would like to note that there have been several proposals to increase security in BGP inter-domain routing [6, 18, 19, 29, 32], including protocols that increase privacy [3, 12, 13]. However, privacy in these protocols does not refer to sender or receiver anonymity but rather focus on the privacy of the ASes and their business models. See Mitseva *et al.* [22] for a comprehensive survey of such proposals.

3 System and threat model

The PHI and dPHI network models are similar to the Internet architecture. We assume two clients want to communicate over a network of routing nodes. In the Internet analogy these routing nodes represent the interconnected Autonomous Systems (AS) that form the Internet. Each node is connected with one or more other nodes over dedicated interfaces. Furthermore, each node can be connected to a multitude of clients. In our nomenclature, clients are denoted with lower case letters, while the routing nodes are denoted with capital letters, typically an A followed by an index.

Each communication is initiated by a client s, called source, that wants to communicate with a client d, called destination. The routing nodes that source and destination are attached to are referred to as A_s and A_d respectively. According to the used protocol, a path is established between s and d via routing nodes. The path between two nodes A_i and A_j is denoted with $P_{A_i-A_j}$ and comprises all nodes in the route path. An asterisk * indicates that the corresponding node is included in the path, i.e. $P_{A_i-A_j*}$ includes A_j but not A_i. In the following we denote $|P_{A_i-A_j}|$ as the number of nodes on the path $P_{A_i-A_j}$, i.e the distance between A_i and A_j, not counting A_i or A_j. Using the Taxonomy of Kelly *et al.* [17], this translates to a *wired* network, with a *free* path topology and a *unicast* routing scheme.

3.1 Original threat model

In this work, we consider two different threat models. The first is the same as defined in PHI. We assume a single attacker controls exactly one routing node A_i and can ***i)*** read and store all packets passing through this node ***ii)*** modify or stop all packets passing this node and ***iii)*** send new packets originating from this node. Furthermore, it is assumed that the attacker knows the network topology and the routing policies of each individual node. Hence, for each packet the attacker can predict how a node would forward it. In principle, there are two attack goals:

- **Sender anonymity**: The attacker tries to de-anonymize the source of a communication request by minimizing the set of clients (the sender anonymity set size) that could have sent a message.
- **Receiver anonymity**: The attacker tries to de-anonymize the destination of a communication request. That is, it tries to minimize the set of clients (the receiver anonymity set size) that could be the recipient of a specific message.

According to the nomenclature of Kelly *et al.* [17] this translates to an adversary with *local* reachability, *dynamic* adaptability, and *active+passive/internal* attackability. The information the attacker can learn and the degree to which the attacker can de-anonymize the receiver or sender is defined in Section 7.7. The destination d does not need to be trusted by s when it comes to sender anonymity. That is, if the adversary is located at the exit node (and hence knows the destination), the security properties of dPHI and PHI hold even if d is malicious. Recently, Wails *et al.* [31] presented attacks against various anonymity protocols in case the source or destination moves within the network and the attacker located within the path between s and d is able to link sessions. While this is an interesting type of attack, it is not part of the threat model of dPHI.

3.2 Extended threat model

The original threat model from PHI can be seen as a realistic scenario in which a nation state actor has con-

trol over ASes which reside in its country. Due to their geographical proximity, these adjacent ASes could be modeled as a single large AS to be in-line with the threat model. Yet, getting control over an AS located geographically far away would be more difficult for such an actor. The assumption that an attacker controls a single AS is therefore realistic in many cases. However, we argue that in the Internet it is very easy to gain access to clients located at various different locations by setting up servers in different countries, using proxy servers, or renting botnets. Hence, it is reasonable to assume that an attacker does not only control a routing node but also a number of clients connected to different routing nodes. Therefore, the extended threat model assumes the same capabilities as before but add a fourth ability: ***iv)*** the attacker has full access to clients connected to different routing nodes. These clients can be used to receive and transmit messages.

4 The PHI protocol

PHI is a stateless routing protocol in which no information about sessions is stored within the routing nodes. Instead all necessary routing information for each AS is stored in the message header with the goal to anonymitize the communication partners to malicious ASes or outsiders. In the following a short introduction to the PHI protocol is given.For a detailed protocol description we refer the reader to the original PHI publication [9].

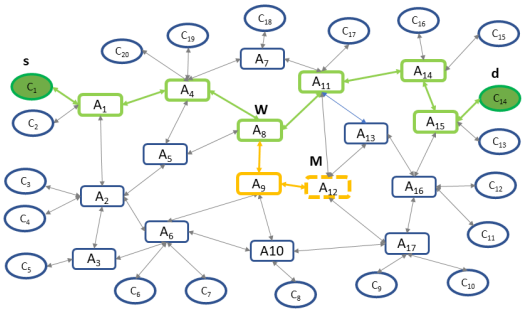


Fig. 1. An example routing procedure between source $s = c_1$, destination $d = c_{14}$ and helper node $M = A_{12}$ resulting in midway node $W = A_8$. The final path between s and d is depicted in green while yellow nodes are only used during setup.

Figure 1 gives an example of the routing process in PHI, where a source $s = c_1$ connected to AS A_1 wants to anonymously communicate with a destination $d = c_{14}$ connected to A_{15}. First, s chooses a helper node M and uses its public key to encrypt the destination address d. In this example $M = A_{12}$ was chosen. Afterwards, s sends a midway request to M that includes the encrypted destination d. Each node on path P_{s-M} stores its routing information, consisting of ingress and egress port to forward a given message, encrypted in the header's routing segment before forwarding it to the next node on the way to M. For this purpose, every node A_i has its own set of secret keys (k_i^{pos},k_i^{enc},k_i^{mac}), so that the encrypted routing information can only be decrypted and authenticated by the node that inserted it. The routing information is stored in the routing segment V at pseudo-random positions pos based on the node-specific secret key k_i^{pos}, session ID sid (which is derived from the session's fresh public key pub_s), and a Pseudo-Random-Function $PRF()$:

$$pos = \mathrm{PRF}(k_i^{pos}, sid) \tag{1}$$

V is initialized by the sender s with random values that are indistinguishable from real routing information. This way, a node receiving a message does not learn anything about a message's previous path. Since a node does not know if a routing element already contains routing information, it might happen that two nodes write to the same routing element. In this case important information is lost so that the session establishment fails. Hence, session establishment in PHI is not dependable and may require several attempts to initiate a session. The authors of PHI proposed to send out four session requests in parallel to achieve a success rate of 90% with the default parameters in the Internet environment. The routing information R_i for node A_i consists of egress and ingress ports and is encrypted with:

$$c_i = \mathrm{ENC}_{k_i^{enc}}(R_i||pos_{prev}||flags) \tag{2}$$

where pos_{prev} is the position of the routing segment which was inserted by the preceding routing node and transmitted in a specific header field. $flags$ contain additional protocol-specific information. A MAC is used to verify the integrity of the current and previous routing element with:

$$m_i = \mathrm{MAC}_{k_i^{mac}}(c_i||m_{i-1}) \tag{3}$$

where m_{i-1} is the MAC in the routing segment at position pos_{prev}. If M would directly forward incoming midway requests to d, the resulting path would be unnecessary long or violate routing policies such as valley-freeness [9]. For this reason, a backtracking phase was introduced in PHI. Instead of forwarding the message to d, it is sent back by helper node M to the previous

node with d in the destination field. Each AS that receives such a midway request message decides based on the routing policy and the stored ingress information if it should become the "midway node" W. In our example, we used a simple routing policy that checks if there exists a shortest path from the previous node to the destination that does not include the current node. If such a path exists, the message is sent back. Else, the current node becomes the midway node W. In Figure 1 this leads to selecting A_8 as the midway node W. We would like to stress that other routing policies such as valley-freeness can be used as well.

After midway node W has been found, the handshake message is forwarded to destination d. Each node on the path between W and d stores its routing information encrypted in the routing segment for later retrieval by the same node. Once d receives a handshake message it computes a session key k_{s-d} between s and d using ECDH key agreement with the session dependent public key pub_s enclosed in the payload of the handshake message. The session ID is generated via a cryptographic hash with $sid = Hash(pub_s)$ to securely link it to the session depended public key pub_s and prevent man-in-the-middle attacks. Each AS on the path back to s uses its (encrypted) routing information stored in the routing segment to determine how to forward the message. When the message reaches s, the handshake is complete and s and d have established a shared session key and a PHI header to securely communicate with each other. Note that a session establishment will not succeed if an aforementioned collision occurs, which is why several session establishment requests are sent in parallel.

5 Attacks on PHI

In the following, several novel sender and receiver deanonymization attacks against PHI are introduced. The quantitative impact on sender and receiver anonymity will be discussed in Section 8.

5.1 Passive distance leakage attack

An attacker controlling a node $A_i \in P_{s-W}$ between the source and the midway node can store the routing segment V during the path request message. The attacker then observes which elements have changed when the handshake reply message comes back. Changed elements belong to nodes in $P_{A_i-M} \cup P_{W-d}$. Let the number of changed elements be a then $a = |P_{A_i-M}| + |P_{W-d}|$. Knowing M, the attacker can predict $|P_{A_i-M}|$ and therefore also learn $|P_{W-d}|$. Note that the attacker does not know which node of the (known) path P_{A_i-M} is the midway node W. Consequently, he does not know $|P_{A_i-W}|$ and cannot directly compute the distance to d. Still, learning $|P_{W-d}|$ in conjunction with P_{A_i-M} can greatly reduce the anonymity set size.

5.2 Active attack on PHI to determine W and the distance to d

The attacker on a node $A_i \in P_{s-W}$ between source and midway node can perform an active attack during the transmission phase to determine which elements belong to P_{W-M*} and which to path P_{A_i-d}. For this attack, we assume that the attacker can determine whether or not a data transmission reaches the destination d. Depending on the used application layer protocol, this could be inferred from re-transmissions from s or responses from d, for example. The attack is fairly simple: The attacker modifies single routing elements and observes if the message still reaches d. If not, the corresponding routing element belongs to P_{A_i-d} since only these elements impact successful routing from A_i to d. This way the attacker can learn $|P_{A_i-d}|$.

The attack's efficiency can be increased by combining it with the passive attack from before that reveals which elements in the routing segment belong to $P_{A_i-M} \cup P_{W-d}$. This way, the attacker only needs to test these. Furthermore, the routing elements belonging to P_{W-M} are the only ones that changed during the passive attack but whose manipulation did not result in a transmission failure. Hence, the attacker also learns the distance $|P_{W-M}|$ between midway and helper node. Since the path between the attacker node and the helper node is known, the attacker can use this information to precisely determine the identity of W. This, in turn, reveals information about the distance to destination d in a topology-based attack.

5.3 Distance leakage to the source

The same active attack as in 5.2 can also be applied in a reverse order to determine the distance to s. The idea is that the attacker modifies routing elements of messages going from the attacker controlled node A_i to s and observes if the messages still arrive. The challenging part is that the source only expects a single valid

path reply. Modifications of V during the transmission phase can easily be detected by s. Hence, once a path reply reaches the source, it will not accept any further path replies. This problem can be circumvented with an attack strategy as follows.

Starting with $i = 1$, the attacker modifies all but i routing elements in V. The attacker does this in all $\binom{l}{i}$ possible ways to send $\binom{l}{i}$ modified path reply messages. If the distance to the source is i, one of these path reply messages is valid and s sends back an answer. If not, the attacker knows that the distance is greater than i and increases i by one and repeats the process. This way source s receives only a single valid setup message and will not get suspicious. Note that the position $prev$ of the previous routing element is known so that the attacker does not consider this position in his attack. Figure 2 illustrates this attack.

While the attack is efficient for small distances, the number of messages the attacker has to send increases exponentially with the distance to s and routing segment size m. However, in PHI the default segment size is $m = 12$, in which only 2048 messages need to be sent to test all distances from 1 to 12. Yet, for large sizes of $m = 48$, the attack quickly becomes impossible.

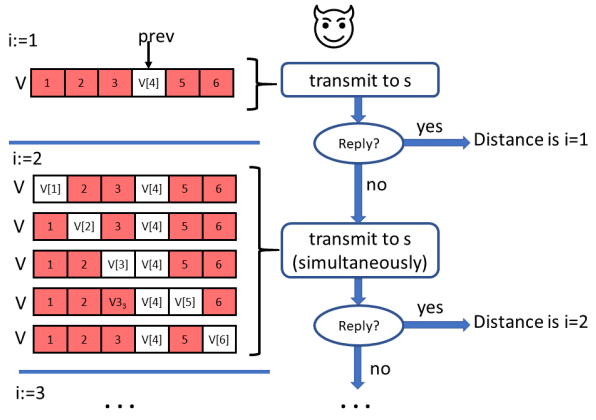


Fig. 2. Illustration of the attack to reveal the distance from A_i to s. In this example the routing segment V consists of six routing elements with $V[4]$ known to belong to the previous node. The routing segments the attacker modifies are indicated in red.

5.4 Attacks on implementations without freshness

In PHI, the used encryption is not specified and in particular it is not explicitly stated that freshness is needed for encryption. This omission could lead to implementations without freshness. Indeed, the implementation section in PHI[9] does not include any freshness and the performance was measured without additional seeds in the header. In the following we discuss what happens if freshness is omitted during the implementation of PHI. Routing elements in PHI are encrypted without authentication using:

$$c_i = \mathrm{enc}_{k_i^{enc}}(R_i || pos_{prev} || flags) \tag{4}$$

with k_i^{enc} being the encryption key of the respective node, R_i the routing information, pos_{prev} the position of the routing element in V corresponding to the preceding node $i-1$, and $flags$ containing protocol-specific information. Hence, the entropy of the plaintext is actually very small. If two messages of different sessions are routed in the same way, the corresponding ciphertexts will be identical if pos_{prev} is the same. Since $pos_{prev} \in \{1, .., 12\}$ with PHI's default parameters, chances for this to occur are very high. By comparing the ciphertexts c_i from different sessions, an attacker can detect routing elements that are equal with a high probability.

If an attacker starts many different sessions with publicly known destinations and different source addresses (e.g. by using proxies), he can craft a lookup table with the ciphertext and corresponding routing information for all ASes with moderate effort. Using such a lookup table, an attacker would in effect be able to decrypt the routing information without actually knowing k_i^{enc}. The only "freshness" stems from the fact that pos_{prev} is based on sid which is linked to a session's public key. Since the number of possible values for pos_{prev} is very small, so is the entropy. Furthermore, a malicious node can use the same sid and public key to send messages mimicking the "forward to A_d phase" of the protocol. The attacker cannot finish the handshake due to the missing private key but can still observe the returned message header, thus detecting overlapping paths with only a few messages. In particular, if the attacker wants to find out if a session belongs to a specific destination, it can use a single message to d with the old session ID sid. If the suspicion is correct, the routing segment returned by the message from d will be identical.

5.5 Correlation of failure probability and distance to destination

In PHI the session establishment is unreliable and the failure probability depends on the distance between

source s and helper node M as well as the distance between midway node W and destination d (see Section 10.2 for details regarding the collision probability). If the attacker is able to link different session establishments for the same destination d but different helper nodes M, he can use these probabilities to make predictions about d. The more data the attacker collects, the more accurate this prediction becomes.

How difficult it is to link session request depends on the used application layer protocol and application. Note that payload encryptions makes it hard to link session when the attacker is not the entry node and therefore does not know s. However, for the entry node it is not unlikely that one can link multiple session requests to the same (unknown) destination by observing the traffic flow and timing at a higher protocol level. Note that the attacker can drop some successful session establishment messages to force s to send out even more session establishment requests to get more data in shorter time.

5.6 Attacks using extended threat model to de-anonymize *s*

The previously described attacks are in line with PHI's threat model (Section 3.1). In Section 3.2 we introduced an *extended threat model* in which the adversary does not only control an AS A_i along the routing path but also n clients $c_1, .., c_n$ that are connected to different ASes. This is a realistic assumption in an open network such as the Internet, where an attacker could utilize proxies or resort to botnets. The attack discussed here assumes that the attacker has control over such clients.

The idea of the attack is to send modified backtracking messages from the attacker controlled AS A_i with different attacker controlled clients $c_1, .., c_n$ as the destination and the original routing segment V^1. Each client c_j receiving such a message records the routing segment V_j^1 and sends it to the attacker. The attacker then compares the routing segment V_j^1 with V^1 and counts the number of changed routing elements. The attacker knows that for a client $c_j \neq s$, one node in P_{s-A_i} is chosen as the midway node W_j. The number of changed routing elements observed in V_j^1 is equivalent to the distance $|P_{W_j-c_j}|$ between this midway node W_j and the attacker controlled client c_j. In a second step, the attacker can also determine $|P_{A_i-W_j}|$ by sending a modified backtracking message to client c_j, again, but this time with modifying some routing elements in V^1 to determine which modifications lead to message drops during backtracking. This is the same approach as done for the attacks described in Section 5.2 and 5.3. The degree of de-anonymization in this attack is very high as the attacker can use many different clients, resulting in many different midway nodes W_j, to narrow down the possible positions of s. Note that the attack becomes easier if the attacker also exploits the missing freshness in PHI's encryption as identical ciphertexts show common paths between different nodes.

6 The dPHI protocol

Several modifications to the PHI protocol are proposed to prevent the discovered attacks. The modifications are based on five main ideas: *i)* splitting the routing segment into two parts to prevent passive distance leakage, *ii)* circular insertion of routing elements instead of picking pseudo-random positions to avoid collision and enable integrity checks, *iii)* extending the backtracking phase back to source s and adding integrity checks of the routing segment, *iv)* addition of a midway nonce to prevent attackers from modifying the destination, and finally *v)* the consistent use of authenticated encryption with fresh IVs to ensure freshness and high entropy ciphertexts. In this Section we discuss these changes in more detail while a formal protocol description with pseudo code can be found in the Appendix A.

6.1 Key management

In dPHI every node A_i has a secret symmetric key k_i that is only known to this one node and used to encrypt and decrypt its own information in the routing header. Furthermore, every node A_i that can serve as a helper node has a ECDH key pair $(pub_i, priv_i)$. The public key pub_i of the chosen helper node needs to be known to source s in advance. In addition to these static keys, the source s generates a session key k_{s-M} with helper node M and a session key k_{s-d} with destination d using ECDH during the dPHI session establishment. The established session key k_{s-d} can be used by both source and destination to exchange encrypted messages that are protected against man-in-the-middle attacks. Note that no payload encryption is done by nodes in dPHI and hence needs to be taken care of by the clients.

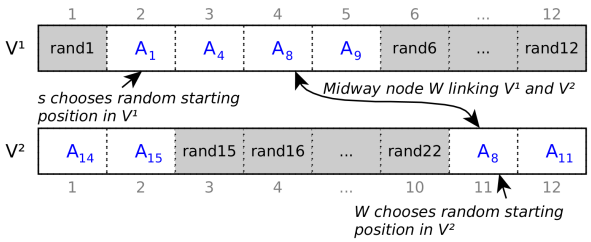


Fig. 3. Illustration of routing segments V^1 and V^2 for the example from Figure 1 with $A_s = A_1$, $M = A_9$, $W = A_8$ and $A_d = A_{15}$. Randomly initialized values are depicted in gray.

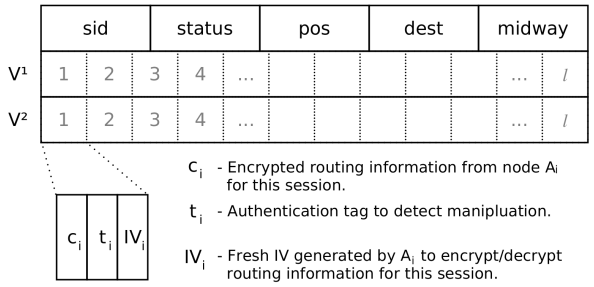


Fig. 4. Illustration of the dPHI header showing V^1 and V^2 that form the routing segment. Note that field sizes in this representation do not relate to real sizes of header elements.

6.2 Authenticated encryption

All symmetric encryption in dPHI is performed using an authenticated encryption algorithm such as AES-GCM [20]. An authenticated encryption algorithms gets three inputs, the key k, the plaintext p and additional authentication data a which is authenticated but not encrypted. The output is a triplet consisting of a freshly generated IV IV, authentication tag t and the corresponding cyphertext c with $(c, t, IV) = \mathrm{enc}_k(p, a)$. In dPHI, the session ID sid is always part of the additional authentication data a to ensure that the ciphertext is securely linked to the current session.

6.3 Modification to the routing segment

Two major changes are applied to the routing segment V. The first is to split it in two, with V^1 storing routing information for nodes on path P_{s-M} and V^2 storing the routing information for nodes on P_{W-d}. Furthermore, routing elements are inserted into V^1 and V^2 in a circular manner, starting at random positions that are chosen during their initialization. Both V^1 and V^2 default to length $l = 12$ but can be adapted individually to fit other use cases. Figure 3 depicts the shape of V^1 and V^2 for the example from Figure 1. Figure 4 shows the complete dPHI header. V^1 is initialized by source s and V^2 by midway node W using keyed cryptographic pseudo random number generators (CPRNGs) based on fresh and secret seeds. While s initializes V^1 immediately, V^2 is initialized by W after the backtracking phase, before forwarding the handshake to d. The seed used to initialize V^2 is stored by W encrypted in the midway field in the dPHI header. As shown in Figure 3, midway node W stores its routing element in both V^1 and V^2 (encrypted with different IVs).

Another modification in dPHI is the routing information R that is encrypted and stored in a routing element. In PHI, it only consisted of ingress and egress ports. In dPHI, it also contains two pointers $posV1$ and $posV2$ indicating the positions where the current routing element is stored in V^1 and V^2. If it is only stored in one of the segments, the other pointer is null. These pointers are used to detect if the position of the routing elements have been altered and enable the midway node to find its field in the other routing segment. A *type* flag indicating that the node is the midway node is stored in R as well. Equation 5 shows how a routing element consisting of (c_i, t_i, IV_i) is computed:

$$\begin{aligned} (c_i, t_i, IV_i) &= enc_{k_i}(R; sid||c_{pos-1}) \\ R &= (ingress||egress||type||posV1||posV2) \end{aligned} \tag{5}$$

See Algorithms 1 and 2 in the Appendix for details.

6.4 Backtracking to s and verification of V^1 and V^2

In PHI, backtracking is done from helper node M to midway node W from where the message is forwarded to destination d. To enable the source to verify the integrity of V^1, backtracking in dPHI is changed as to reach all the way back to s. The node that becomes the new midway node W, encrypts the destination address d with its secret key and stores it in the destination address field in the header. Then, the message is sent back to s. Consequently, nodes between W and s do not learn about d, while midway node W can retrieve the destination using its secret key in the next phase of the protocol when s sends the handshake to d. This modified backtracking phase allows source s to check if V^1 has been tampered with by comparing it to the initial version of V^1 that s stored while preparing the request. In dPHI it is assumed that s knows the maximum distance to the chosen helper node M. The idea of this integrity check is

that with knowledge about the distance, s can estimate which routing elements in V^1 should have changed and which not. If modifications outside the expected range occur, s drops the message and may initiate a new attempt. See Algorithms 3-5 in the Appendix for details.

Midway node W performs the same integrity verification of V^2 as s does of V^1. W initializes V^2 when it forwards the message to d for the first time, based on a fresh seed and a keyed CPRNG (See Algorithm 6 in A.4). To be able to retrieve the seed when the message gets back, W encrypts the seed together with a close upper bound $dist_d$ of the distance to d with its own secret key and stores it in a dedicated midway field in the header. In addition to the session id sid, the ciphertext c_{V1} of W's routing element in V^1 is used as additional authentication data for this encryption.

$$midway_{(to\_d)} \leftarrow \mathtt{enc}_{k_i}(seed||dist_d, sid||c_{V1}) \quad (6)$$

When the midway node receives the handshake reply from d, it decrypts the seed and distance value and verifies that only the expected routing elements in V^2 have changed. The additional authentication data securely links V^1 and V^2 so that an attacker cannot simply replace V^1 or V^2 as a whole without the midway node noticing. After verifying V^2, the midway node W replaces the midway field with a MAC of the entire routing segment comprising V^1 and V^2 to ensure this secure linkage for the remainder of the protocol.

6.5 Addition of the midway nonce

The last major change in dPHI is the addition of a midway nonce to prevent attacks in the extended threat model. During initialization, s generates a nonce n_{mid} (see Appendix A.2-A.5 for details). This nonce is encrypted with the shared session key k_{s-M} that s derived with help of M's publicly available key pub_M. To enable M to derive that exact same key, s includes pub_s of its freshly generated key pair in the payload when sending its request to M. The helper node M decrypts this nonce and puts it, unencrypted, in the midway field of the backtracking message that is sent back. Midway node W, in turn, computes a hash value of the midway nonce together with the destination $dest$ and routing segment V^1 and overwrites the midway field $midway_{(to\_s)}$:

$$midway_{(to\_s)} \leftarrow \mathtt{Hash}(dest||n_{mid}||V^1) \quad (7)$$

When source s receives the message at the end of the backtracking phase, it verifies this hash value and that the correct destination $dest$ has been used. If this is not the case, the message is dropped.

7 Security analysis

In Section 5 several novel attacks on PHI have been introduced. In this section we elaborate on how the modifications introduced with dPHI prevent these attacks.

7.1 Passive distance leakage attack

The passive distance leakage in PHI is the main reason for splitting the routing segment into two parts, V^1 and V^2. The number of changed elements in V^1 that a node in P_{s-M} can observe is equivalent to the distance to the known helper node M. Hence, the attacker does not learn anything new by counting the changed elements in V^1. When a node in P_{s-W} receives V^2 for the first time, all values are new as they are initiated by midway node W. Consequently, a node in P_{s-W} cannot make any observation about d. The routing element corresponding to W changes twice in V^1. However, no single node receives both versions so that W cannot be identified based on any such observation. All nodes in P_{W*-M} and P_{W*-d} know d so that distance leakage attacks to the destination are only interesting for P_{s-W}.

7.2 Active attack to determine W and the distance to d

In dPHI, an attacker controlling $A_i \in P_{s-W}$ would need to (repeatedly) modify routing elements in V^1 to identify which of them belong to P_{W-M} or W. To prevent this, backtracking is extended beyond W so that source s receives V^1 after backtracking. Destination d receives V^1 with the handshake message. Hence, any modification of V^1 after the handshake message is detected by s or d as they always check the integrity of received headers. To detect manipulations during the phases of backtracking to W and backtracking to s, the source uses the new midway reply message in the dPHI protocol:

$$H.midway_{(to\_s)} = n_{rep} = \mathtt{Hash}(d||n_{mid}||V^1) \quad (8)$$

The midway reply message is generated by midway node W, the last node to modify V^1. Destination d and n_{mid} are known to all nodes on P_{W*-M*} but not to nodes on P_{s-W}. Hence, an attacker on $A_i \in P_{s-W}$ cannot

compute a valid midway reply for a modified V^1. This way source s can detect any modifications to V^1 after processing by midway node W.

To prevent attacks during the handshake message to d, source s sends a MAC of V^1 with the shared session key to destination d. The destination can therefore authenticate V^1 based on the shared session key and discard any message with a modified V^1.

7.3 Distance leakage to the source

To prevent this attack, the source s stores the random values that were used to initialize V^1, the starting position *pos* pointing to the first routing element, and the maximum distance $dist_M$ to M. On receiving the midway reply, s uses this information to verify that only routing elements belonging to the path P_{s-M} have been modified, i.e., elements in V^1 between *pos* and $pos + dist_M$. If other elements have been modified, the message is dropped by s. This results in the same effect (session establishment failure) as if an element belonging to P_{s-A_i} had been modified by the attacker. After the midway request, s stores V^1 and drops any message in which the routing segment V^1 is not identical. This ensures that the attack on V^1 can also not be executed in a later stage of the protocol.

However, an attacker on $A_i \in P_{W-d}$ could try to manipulate V^2 to learn at least the distance between A_i and (the unknown) midway node W. To prevent such attacks, the midway node W verifies V^2 the same way as s has verified V^1. Midway node W has stored the seed for initializing V^2 and the distance $dist_d$ between W and d encrypted in the midway field of the header. With this information the midway node verifies that at most $dist_d$ routing elements have been inserted in V^2, starting from position pos_2 that W retrieves from its own routing element. At the end of the session establishment, source s stores V^2 so that any modifications of V^2 during the data transmission phase are detected. Hence, in dPHI the attacker cannot learn the distance to s nor the distance to W.

7.4 Exploiting missing freshness

In dPHI we propose to use authenticated encryption with fresh IVs whenever information is (re-)encrypted. Hence, the attacker cannot learn anything by observing ciphertexts. The position *pos* within a routing segment in dPHI is independent from a secret key or node address since routing elements are inserted in a circular manner. Therefore, which routing elements in V^1 or V^2 are modified cannot be used to link sessions or nodes.

7.5 Correlation of failure probability and distance to destination

In dPHI the circular insertion of routing elements ensures that no collisions occur as long as the maximal sizes of V^1 and V^2 are not exceeded by the chosen path to M and d. The session establishment is therefore dependable and this type of attack is not possible.

7.6 Active attacks in the extended threat model

In dPHI the backtracking phase has been altered so that the midway node sends a midway reply back to the source s. The source s then verifies that the correct destination has been used to determine the midway node W. To this end, the midway node W computes a midway reply $H.midway$ by hashing the midway nonce n_{mid} with the destination d and routing segment V^1, i.e. $n_{rep} \leftarrow \texttt{Hash}(d||n_{mid}||H.V^1)$. This midway reply n_{rep} is sent back in the midway field $H.midway$ of header H. The source recomputes this midway reply and can thereby verify that the midway node W has received the correct destination d.

It is important to ensure that an attacker cannot misuse the midway reply $H.midway$ to learn the destination d. The high entropy midway nonce n_{mid} has been sent encrypted to the helper node M. Only M and nodes on P_{W*-M}, the same that also know destination d, learn the midway nonce which is necessary to compute $H.midway$. Nodes in P_{s-W} do not know the midway nonce and can therefore not compute $\texttt{Hash}(d||n_{mid}||V^1)$. In consequence, it is not possible for an attacker on these nodes to misuse $H.midway$ as an oracle to determine d.

7.7 Information leakage in dPHI

dPHI does not offer perfect anonymity as the attacker can learn some information about communicating peers. In this section we define which information the attacker is able to learn about sender and receiver depending on the node that the attacker has compromised.

Sender anonymity:
Attacker on entry node A_s:
1. The entry node knows the source address s, i.e., there is no sender anonymity.

Attacker on A_i with $A_i \in P_{A_s-W}$ or $A_i \in P_{W-M*}$:
1. For (known) destination M the path P_{A_s-M} goes through $A_{i-1} \cup A_i$, thus reducing the anonymity set of s to sources reachable through A_{i-1}.
2. The distance to s cannot be higher than the maximum segment length m minus the distance $|P_{A_i-M|}$.

Attacker on midway node W:
1. Learns the same as attacker on P_{A_s-W} or P_{W-M*}.
2. W can perform an active attack by violating the routing policy and deciding not to become the midway node for the known destination d. The attacker can then observe whether or not the node W will be selected for the path $P_{W'-d}$ (where W' is the new midway node with $W' \in P_{A_s-W}$). In how far this de-anonymizes s depends on the deployed routing policy and layout of the network.

Attacker on A_i with $A_i \in P_{W-d*}$ learns:
1. For the (known) destination d the path P_{W-d} goes through $A_{i-1} \cup A_i$, thus reducing the anonymity set of unknown W.
2. It furthermore knows that the distance to W cannot be greater than the maximum segment length m minus the distance from A_i to d.

Receiver anonymity:
Attacker on A_i with $A_i \in P_{A_s*-W}$ learns:
1. Destination d can be reduced to such destinations that one of the nodes on the predictable path to M (i.e., P_{A_i-M*}) can serve as midway node W without violating the employed routing policy.

Attacker on A_i with $A_i \in P_{W*-d*}$ or $A_i \in P_{W-M*}$:
1. These nodes learn destination d and hence there is no receiver anonymity.

The entry node learns the identity of source s and the midway node the destination d which results in zero anonymity if the entry node also becomes the midway node. This is a major drawback of PHI/dPHI since topology information is required to choose M such that this does not happen.

8 Quantitative anonymity analysis

In this section we compare the achieved sender and receiver anonymity for the lightweight anonymity protocols LAP, HORNET, PHI and dPHI.

8.1 Assumptions and experimental setup

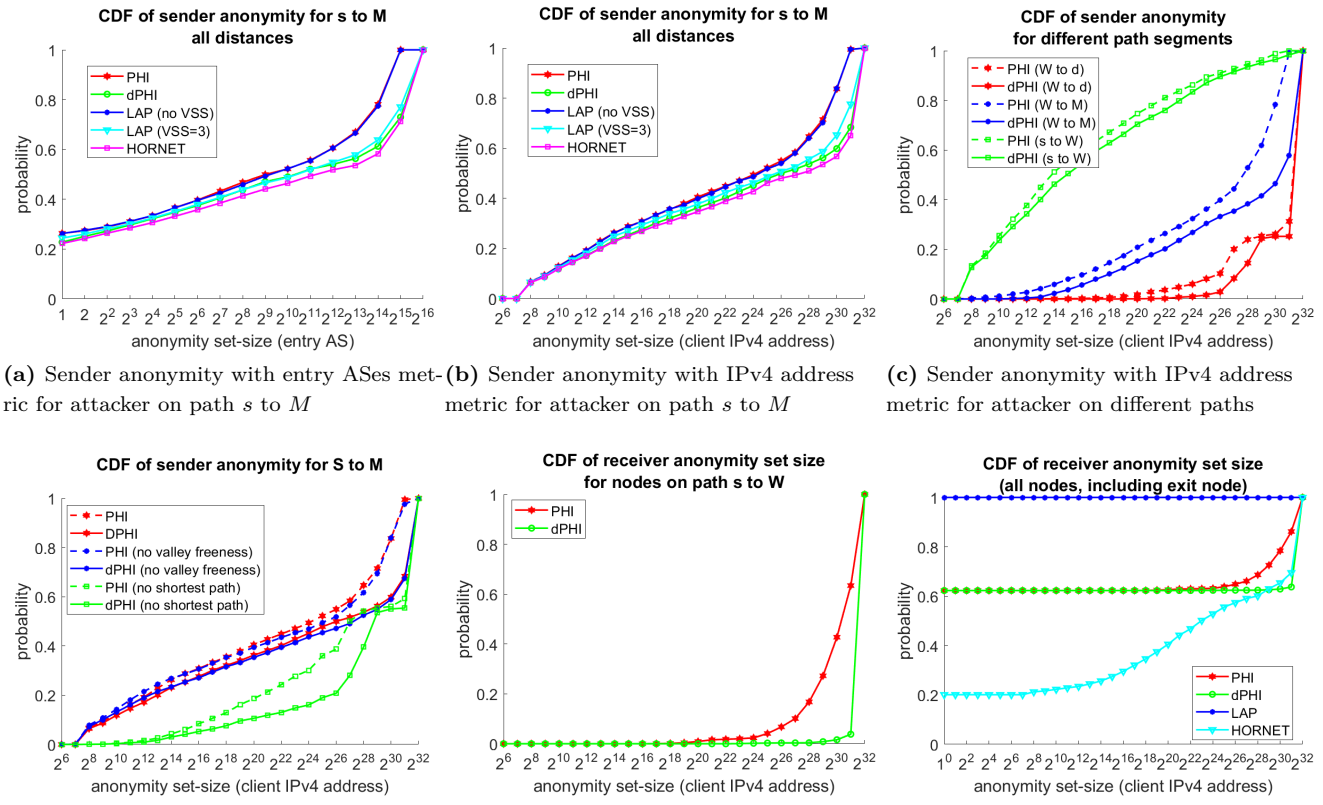
Computing the anonymity depends on several factors such as the network topology, the routing policies and the attacker's capabilities. In PHI [9] and HORNET [8], anonymity set size was computed based on a valley-free routing policy and the 2014 CAIDA dataset [15] representing the Internet infrastructure on the level of Autonomous Systems. In the valley-free routing model, the network is a directed graph in which each edge is either of the type provider-to-customer or peer-to-peer. A valley-free path is a path that starts with any number of customer-to-provider edges followed by zero or one peer-to-peer edge and any number of provider-to-customer edges. The PHI and HORNET papers [8, 9] assume all valley-free routes as equally likely, regardless of the length of the route. However, for a given source and destination pair it is reasonable to assume that only a predictable subset of paths will be chosen in practice based on the network topology, e.g. the shortest paths.

We, therefore, believe that a network model in which only a selected number of paths between two nodes is valid is more realistic. In our analysis, we use a shortest path valley-free routing policy in which all valid paths need to have the minimum number of hops. We computed the sender and receiver anonymity set size for PHI, dPHI, LAP with one element per hop (i.e., no VSS), LAP with thee elements per hop (VSS=3), and HORNET. For this, we chose 1000 random source and destination pairs and computed a PHI path for each by randomly choosing a helper node. Should no valley-free path exist or the midway node be identical with the entry node, we chose new random nodes. The computation was performed using Matlab and the 2014 CAIDA dataset [15]. The software as well as used data is publicly available on Github [4]. Our quantitative analysis uses the following assumptions and metrics:

Attacker capabilities: For PHI we assume the passive and active attacks from Sections 5.1-3 but not the attacks exploiting low entropy or those based on malicious clients (Section 5.4-6).

Routing assumption: If not specified otherwise, a valley-free shortest path routing is used in which each shortest path is equally likely to be chosen. Only in Figure 5d other routing schemes (non valley-free shortest path and valley-free without shortest path) are used.

Anonymity set size: We compute the anonymity set size based on two metrics: 1) The *number of ASes* that could be the entry node (sender anonymity) or the exit node (receiver anonymity). And 2) the size of the *IPv4 address space* associated with the ASes that could be



(a) Sender anonymity with entry ASes metric for attacker on path s to M

(b) Sender anonymity with IPv4 address metric for attacker on path s to M

(c) Sender anonymity with IPv4 address metric for attacker on different paths

(d) Sender anonymity with IPv4 address metric for attacker on path s to M with different routing policies

(e) Receiver anonymity with IPv4 address metric for attacker on path s to W

(f) Receiver anonymity with IPv4 address metric for attacker located anywhere on the path (including exit node)

Fig. 5. Quantitative sender and receiver anonymity set size analysis. Results are shown as cumulative distribution functions (CDF) in which the y-axis shows the probability that the anonymity set size is equal or smaller than the value depicted in the x-axis.

the entry node or exit node, respectively.

Location of attacker node: The location of the attacker on a PHI or dPHI path greatly influences the anonymity set size (See Section 7.7). The attacker location for dPHI is therefore specified in each Figure. For LAP and HORNET all nodes between the source and destination are used as attacker locations for the anonymity set size comparison as there is no midway node or helper node in these schemes.

8.2 Sender anonymity

Figure 5a shows the CDF of the sender anonymity set size for nodes on the path from source s to midway node M. The sender anonymity set sizes for LAP and HORNET for the path s to d are also depicted for comparison. In this Figure, anonymity set sizes based on entry ASes are shown while Figure 5b presents the same information on the basis of possible IPv4 source addresses. Please note that the anonymity set size on the x-axis has a logarithmic scale with base 2 so that a shift to the right by one is already an increase by a factor of two. As one can see, PHI only achieves the anonymity level of LAP without VSS while dPHI performs slightly better than LAP with $VSS = 3$.

The sender anonymity for paths W to d and W to M is shown in Figure 5c. These paths are especially interesting for PHI and dPHI as enclosed nodes learn the destination. The anonymity set size for W to d is considerably larger than W to M since the backtracking algorithm in PHI does not guarantee a shortest path routing. The anonymity set size in Figure 5c for dPHI, therefore, consists of all nodes that are reachable via the previous hop without compromising valley-freeness. In PHI, the attacker can learn the path length, hence, we excluded all nodes that cannot be reached with a number of hops that is less or equal to the path length. Note, however, that in 97.5% of the sessions a shortest path was chosen and in 2.5% of the cases, the resulting path

was only one hop longer. Only in one out of one thousand sessions, the resulting path was two hops longer. If an attacker would ignore the 2.5% chance of a longer route, the anonymity set size for W to d would look like that from W to M. The anonymity set size decreases the closer the node is to the source, which is why the anonymity set size for nodes on P_{s-W} is the lowest as compared to nodes on P_{W-M}.

We also tested the impact of different routing policies in Figure 5d. If no shortest path routing is assumed, i.e., all valley-free paths are assumed to be valid, the anonymity set size increases significantly. Using a shortest path routing policy without requiring valley-freeness also results in a larger anonymity set size than the valley-free shortest path routing policy. But this difference is considerably smaller than compared to not using a shortest path routing policy.

8.3 Receiver anonymity

The receiver anonymity set size is depicted in Figures 5e and 5f. LAP does not provide receiver anonymity and hence the anonymity set size is 1. PHI and dPHI only provide receiver anonymity for nodes on the path P_{s-W} from the source to the midway node. In our experiment this corresponds to 60% of the nodes being able to eavesdrop d during session establishment. While the receiver anonymity set size for both PHI and dPHI is already very high, dPHI outperforms PHI considerably. Note that a high receiver anonymity is especially important for the entry node A_s which knows the source so that the combined sender-receiver anonymity solely depends on the receiver anonymity.

8.4 Upper bound of sender-receiver anonymity

Computing the sender-receiver anonymity for PHI and dPHI is computationally extremely expensive as one has to compute all possible paths from all sources to all destinations for all helper nodes. Therefore, we only computed an upper bound of the sender-receiver anonymity set size. This was done by multiplying the sender anonymity set size for an attacker on node A_i (i.e., all possible sources for this session establishment) with the receiver anonymity set size for the same node (all possible destinations). The correct source and destination pair lies within this upper bound. However, the actual sender-receiver anonymity set size might

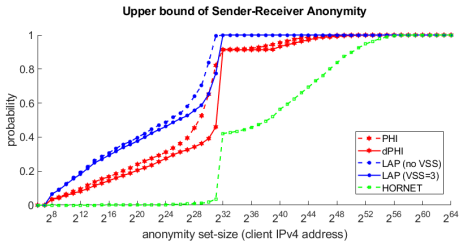


Fig. 6. Upper bound of sender-receiver anonymity set size with IPv4 address metric for an attacker located anywhere on a path.

be smaller as for some sender-receiver pairs in this anonymity set size there may exist no helper node such that a path traverses through the attacker node. Yet, this upper bound provides a rough estimation of the sender-receiver anonymity set size. Figure 6 shows the upper bound of the sender-receiver anonymity. dPHI considerably outperforms LAP in this metric. Nodes that have a receiver anonymity set size of 1 in dPHI (nodes on path P_{W-d} and P_{W-M}) exhibit a relatively high sender anonymity (see Figure 5c) while nodes with low sender anonymity (close to source s) have a relatively high receiver anonymity. This ensures that the minimum sender-receiver anonymity set size remains fairly high in dPHI. HORNET clearly performs the best in terms of sender-receiver anonymity as only the entry and exit nodes have a sender or receiver anonymity set size of one.

9 Limitations of dPHI

In this subsection we would like to explicitly point out some limitations of dPHI. One limitation of dPHI is that the threat model assumes the attacker to control only a single AS. If the attacker controls two ASes the attacker can learn considerably more by combining the individually learned information. If the nodes the attacker controls are close to each other, this is not as problematic as the case that the attacker controls two nodes that are further away. This is due to the fact that attacker nodes close to the source have a low source anonymity but large destination anonymity while attacker nodes close to the destination have no destination anonymity but large source anonymity. Hence, the anonymity can be greatly reduced if the attacker manages to gain control over two nodes on the path that are further apart. Active attacks on the routing layer could make this issue even worse in practice. For example, in BGP route poisoning is a known problem that was used in the past to

direct traffic over specific ASes for eavesdropping [2, 30]. Similarly, an attacker might drop session requests until a favorable dPHI path is established [5].

The biggest limitation and open problem of the dPHI protocol is the selection of the midway node. To be more precise, how to make sure that the entry node does not also become the midway node. If the entry node becomes the midway node there is no source or destination anonymity since the entry node (necessarily) knows the source and the midway node knows the destination. In a network scenario in which the source knows the employed routing policy of all nodes, it can verify that this does not happen. However, in adaptive routing policies such as BGP, one cannot accurately predict the route that will be chosen. Indeed, the fact that dPHI does not require client-based routing is the main benefit of dPHI compared to HORNET. Efficiently solving this problem without requiring client-based or client-controlled routing is important and very interesting future work. Note that if the entry node performs active attacks to shape the traffic as described above, the problem of preventing the entry node to become the midway node becomes even more severe.

10 Performance analysis

In the following, the performance of dPHI in terms of computation complexity, latency, header size and goodput is compared to PHI.

10.1 Latency

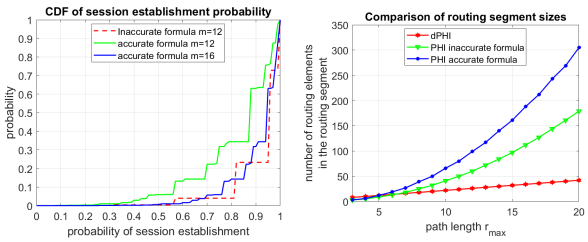
To compare the latency of processing PHI and dPHI messages during session establishment and transmission, the dPHI and PHI packet processing was implemented in C and is available on Github [4]. To make use of Intel's AES-NI instruction set, we utilized their *Intelligent Storage Acceleration Library Crypto Version* (ISA-L_crypto) to implement AES-GCM-256 and AES-CBC-256 for required AES operations. Any ECDH operations have been realized with the curve25519-donna library. For measuring the required clock cycles we used the intrinsic function *rdtsc()* and conducted one million independent measurements. After sorting these measurements, we discarded the top and bottom 37.5% and calculated the average of the remaining quarter, as is common practice when using *rdtsc()* and was also done by Chen *et al.* [8, 9]. Though, please note that in the presence of compiler and processor optimizations *rdtsc()* measurements on current processors for low numbers of clock cycles are quite inaccurate despite being the best available option. We performed our measurements on an Intel Core i7-7500U with 2.7GHz using a single thread. Table 1 shows the processing latency results for PHI and dPHI. Some dPHI operations require fresh random numbers that can either be precomputed or generated on the fly. We implemented both and supplied two values in the Table 1 for these operations.

	dPHI	PHI
Session establishment		
Midway Request ($A \neq M$)	430*/1151	110
Midway Request ($A = M$)	146000	144915
Backtracking ($A \neq W$)	117	105
Backtracking ($A = W$)	1600*/3169	222
Handshake to d ($A \neq W$)	430*/1151	110
Handshake to d ($A = W$)	1255*/3619	n/a
Handshake reply to s ($A \neq W$)	117	105
Handshake reply to s ($A = W$)	951*/2124	105
Transmission phase		
Transmission phase ($A \neq W$)	117	105
Transmission phase ($A = W$)	250	105

Table 1. Measured clock cycles of header processing for different nodes A in the protocol during the different phases of session establishment and the transmission phase. Entries marked with an asterisk are clock cycle measurements for an implementation based on precomputed random numbers (a node can precompute random numbers when the processor is idle).

For session establishment, the computation time of nodes besides the helper node is negligible, especially when considering that in both PHI and dPHI the helper node, source, and destination have to perform public key operations. As can be seen in Table 1, the public key operation of the helper node is up to two orders slower than all other cryptographic operations. In dPHI the backtracking phase is extended to s so that more nodes need to be traversed. While the additional cryptographic operations will not significantly impact the setup latency, propagation latency might be higher. However, a PHI session establishment only succeeds with 90% probability. When a session establishment fails, the process needs to be repeated, including the computationally expensive public key operations, thus nearly doubling the setup latency. In consequence, while the minimum setup latency in dPHI is slightly worse than PHI due to the additional hops, the average and especially worst case setup latency are better.

In HORNET, pubic key operations have to be performed by all nodes on the path so that the session establishment latency is considerably higher than in dPHI or PHI. Furthermore, the public key operations result in a considerable burden for the ASes if many sessions are created in parallel. An average path in the 2014 CAIDA dataset is about 4.2 hops, hence in average 4.2 sequential public key operations need to be performed per session in HORNET. In PHI, four parallel public key operations, resulting from four independent requests, suffice with a 90% probability, in 10% of the cases, eight or more are needed. In dPHI on the other hand, only one public key operation needs to be performed within the network. Therefore, dPHI reduces the overall computation load induced by the protocol considerably.



(a) CDF of the probability that a PHI session establishment is successful

(b) Comparison of routing segment sizes of dPHI and PHI

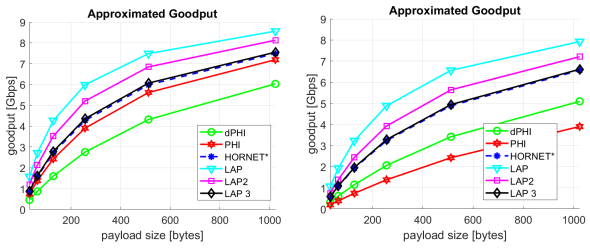
Fig. 7. a) CDF of the probability that a PHI session establishment is successful with $N = 4$ parallel session requests with the inaccurate formula from [9] and the accurate formula for different segment sizes m. b) Comparison of the routing segment size between PHI and dPHI depending on the maximum path length r_{max}. For PHI the routing segment size was computed such that the probability of a successful session establishment is at least 90% when sending out $N = 4$ request. For dPHI the routing segment size is $m = 2 \cdot l = 2 \cdot (r_{max} + 1)$.

10.2 Header size and PHI collision probability

The header size is greatly influenced by the routing segment size. The default parameters proposed in PHI [9] for the number of routing elements in V is $m = 12$ for path length smaller than 8 and $m = 48$ for larger paths. Furthermore, $N = 4$ session establishment request are sent out in parallel for small headers and $N = 5$ for large ones. The parameters were chosen so that a session establishment succeeds with 90% probability for the 2014 CAIDA dataset. However, the used formula to compute the success probability [9] is not accurate. It only computes the probability of a collision on the path between the source and destination but ignores the collision that can occur between the midway node and the helper node. In Figure 7a we compute the session establishment probability for 1000 random PHI paths with the inaccurate formula and an updated formula that includes the collision probability between midway and helper node. The success probability with the accurate formula is considerably smaller. In fact, $m = 16$ instead of $m = 12$ is needed to achieve the 90% probability. For a maximum path length r_{max}, the number of routing elements in a dPHI header is $(r_{max} + 1) \cdot 2$. We compared the size of the routing segment of PHI and dPHI for different values of r_{max} in Figure 7b under the assumption that for PHI a routing segment size m is chosen such that a session establishment success rate of 90% is achieved. For this computation, an estimation of the expected path length from W to M is needed. In our experiments of 1000 random PHI paths, the ratio of the path length r between s and d and the path between W to M was roughly 50%. This value is used in Figure 7b, as well. As one can see, the routing segment size in PHI grows exponentially while it only grows linearly in dPHI.

It is also possible in dPHI to use different sizes for V^1 and V^2 than the default value of $l = 12$ for each vector. This is due to the fact that the source chooses the helper node and can simply pick one with a distance equal or smaller to 7 without reducing too much anonymity. Most nodes will be within this distance and a larger size is reserved for V^2 to reach far away destinations. Furthermore, since the path is split into two routing segments it is possible to create routes longer than r by choosing a helper node such that the midway node is roughly half-way between s and d.

LAP uses a variably sized header that does not depend on the maximum path length but the current path length. To make comparisons between PHI, dPHI, LAP and HORNET we used different values for maximum path length r_{max} and average path length r_{avg} for LAP. We chose $r_{max} = 7$ with $r_{avg} = 5$ and $r_{max} = 11$ with $r_{avg} = 8$ for our analysis and assumed a value of $m = 16$ and $m = 66$ for PHI segment sizes for normal and large headers. The PHI values were chosen based on the more accurate collision formula. In dPHI the header size is made up of a fixed part of 50 bytes for sid, midway field, pointers and flags, as well as the routing segments V^1 and V^2 whose size depends on r_{max}. One routing element consists of 39 bytes so that the header size for dPHI is 674 bytes and 986 bytes for $r_{max} = 7$ and $r_{max} = 11$ respectively.



(a) r_{max}=7, $r_{avg} = 5$ **(b)** r_{max}=11, $r_{avg} = 8$

Fig. 8. Approximated goodput of the different protocols based on the ratio of header size and payload (assuming processing is not the bottleneck). LAP uses variable sized headers and we used the average number of hops in the computation.

10.3 Goodput and throughput

In [9] goodput measurements were performed using an SDN testbed. It was observed that PHI-related packet processing had no considerable impact on transmission rates of the used 10 Gbps link. Only in HORNET processing speed impacted the goodput since HORNET also performs payload encryption. The processing speed of packets during transmission is roughly the same for dPHI and PHI with a measured clock cycle count of 117 vs 105 (See Table 1). This means, from a computational perspective, that the number of dPHI headers to be processed, even with the small size of 674 bytes, could be one magnitude higher than the number that is actually needed to saturate the 10 Gbps link. Hence, only the header size impacts the goodput in dPHI, just as has been the case in PHI. We therefore approximated the goodput for different payload sizes by computing the ratio of header size divided by packet size. The result of this analysis can be seen in Figure 8. In the computation we assumed that the size of a PHI routing element is 24 bytes with 8 bytes resulting from encrypting the routing information with AES in counter mode and 16 bytes from the 128-Bit MAC that is also used in HORNET and dPHI. Note that we propose in dPHI to only use $r = 11$ or larger but included $r = 7$ for comparability.

The result of our analysis shows that the average goodput of LAP is the best, even with $VSS = 3$ (unlike what was stated in [9], which assumed the worst case header size, it seems). HORNET's goodput is also higher than that of both PHI and dPHI due to its smaller header size. Note that Figure 8 does not consider processing load as a potential bottleneck. But measurement data from [9] which include processing overhead yield similar results for large payloads (slightly above 7 Gbps for a 1024 byte payload). Comparing PHI with dPHI, we can see that, for a small $r = 7$, PHI has a higher goodput than dPHI. However, dPHI outperforms PHI for large paths of $r = 11$. Note that this difference will increase rapidly for larger path due to the exponential vs linear growths in header size.

11 Conclusion

In this paper, we showed that the PHI protocol has several limitations that can be exploited by attackers to significantly reduce the achieved anonymity. Based on PHI, a new protocol named dependable PHI (dPHI) was introduced that withstands these attacks. This holds true even if the threat model is extended to consider attackers who control clients in various ASes. dPHI also solves PHI's problem of collisions in the routing segment so that dPHI needs fewer session establishment requests and smaller header sizes. A quantitative anonymity and performance analysis shows that dPHI offers a good trade-off between performance and anonymity compared to LAP and HORNET. The sender-receiver anonymity set size of dPHI is considerably larger than that of LAP. This is mainly due to the fact that dPHI offers receiver anonymity for nodes on path s to W, while LAP does not offer receiver anonymity. HORNET provides the best anonymity of the discussed protocols.

In terms of setup latency and goodput, dPHI achieves similar or better results than PHI but lower goodput than LAP or HORNET. HORNET requires expensive public key operations on all routing nodes during session establishment. Furthermore, HORNET uses client-based routing that is not in line with today's Internet design. dPHI does not have such restrictions on the routing algorithm and can be used in conjunction with BGP. The only major issue in dPHI is that the client should choose a helper node so that the entry node does not become the midway node. How to ensure this without requiring the client to know the network topology is an open research problem.

12 Acknowledgements

This work was supported by Rheinmetall. We would also like to thank the anonymous reviewers and our shepherd Marios Isaakidis for their very helpful feedback.

References

[1] M. AlSabah and I. Goldberg, "Performance and security improvements for tor: A survey," *ACM Computing Surveys (CSUR)*, vol. 49, no. 2, p. 32, 2016.

[2] N. Anderson, "How china swallowed 15% of net traffic for 18 minutes," 11 2010, Arstechnica. [Online]. Available: https://arstechnica.com/information-technology/2010/11/how-china-swallowed-15-of-net-traffic-for-18-minutes/

[3] G. Asharov, D. Demmler, M. Schapira, T. Schneider, G. Segev, S. Shenker, and M. Zohner, "Privacy-preserving interdomain routing at internet scale," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 3, pp. 147–167, 2017.

[4] A. Bajic and G. T. Becker, "Github repository of used software and data," https://github.com/AlexB030/dPHI, [Online; uploaded 10-March-2020].

[5] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz, "Denial of service or denial of security?" in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 92–102.

[6] K. Butler, P. McDaniel, and W. Aiello, "Optimizing bgp security by exploiting path stability," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, pp. 298–310.

[7] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: Website fingerprinting attacks and defenses," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 605–616.

[8] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig, "Hornet: High-speed onion routing at the network layer," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1441–1454.

[9] C. Chen and A. Perrig, "Phi: Path-hidden lightweight anonymity protocol at network layer," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 1, pp. 100–117, 2017.

[10] D. Das, S. Meiser, E. Mohammadi, and A. Kate, "Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency - choose two," in *2018 IEEE Symposium on Security and Privacy (SP)*, vol. 00, May 2018, pp. 108–126.

[11] P. Dhungel, M. Steiner, I. Rimac, V. Hilt, and K. W. Ross, "Waiting for anonymity: Understanding delays in the tor overlay," in *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*. IEEE, 2010, pp. 1–4.

[12] D. Gupta, A. Segal, A. Panda, G. Segev, M. Schapira, J. Feigenbaum, J. Rexford, and S. Shenker, "A new approach to interdomain routing based on secure multi-party computation," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM, 2012, pp. 37–42.

[13] W. Henecka and M. Roughan, "Strip: Privacy-preserving vector-based routing," in *2013 21st IEEE International Conference on Network Protocols (ICNP)*. IEEE, 2013, pp. 1–10.

[14] H.-C. Hsiao, T. H.-J. Kim, A. Perrig, A. Yamada, S. C. Nelson, M. Gruteser, and W. Meng, "Lap: Lightweight anonymity and privacy," in *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012, pp. 506–520.

[15] http://www.caida.org/data/as-relationships, "The caida ucsd [as relationships] -[2014-09-01]," caida, Tech. Rep., 2014.

[16] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, "Users get routed: Traffic correlation on tor by realistic adversaries," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 337–348.

[17] D. Kelly, R. Raines, R. Baldwin, M. Grimaila, and B. Mullins, "Exploring extant and emerging issues in anonymous networks: A taxonomy and survey of protocols and metrics," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 2, pp. 579–606, 2012.

[18] S. Kent, C. Lynn, and K. Seo, "Secure border gateway protocol (s-bgp)," *IEEE Journal on Selected areas in Communications*, vol. 18, no. 4, pp. 582–592, 2000.

[19] M. Lepinski and K. Sriram, "Bgpsec protocol specification," RFC 8205, Tech. Rep., 2017.

[20] D. McGrew and J. Viega, "The galois/counter mode of operation (gcm)," *Submission to NIST Modes of Operation Process*, vol. 20, 2004.

[21] J. McLachlan, A. Tran, N. Hopper, and Y. Kim, "Scalable onion routing with torsk," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 590–599.

[22] A. Mitseva, A. Panchenko, and T. Engel, "The state of affairs in bgp security: A survey of attacks and defenses," *Computer Communications*, vol. 124, pp. 45–60, 2018.

[23] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of tor," in *Security and Privacy, 2005 IEEE Symposium on*. IEEE, 2005, pp. 183–195.

[24] J. Ren and J. Wu, "Survey on anonymous communications in computer networks," *Computer Communications*, vol. 33, no. 4, pp. 420–431, 2010.

[25] J. Sankey and M. Wright, "Dovetail: Stronger anonymity in next-generation internet routing," in *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2014, pp. 283–303.

[26] F. Shirazi, M. Simeonovski, M. R. Asghar, M. Backes, and C. Diaz, "A survey on routing in anonymous communication protocols," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, p. 51, 2018.

[27] P. Syverson, R. Dingledine, and N. Mathewson, "Tor: The secondgeneration onion router," in *Usenix Security*, 2004.

[28] M. van den Berg, P. de Graaf, P. Kwant, and T. Slewe, "Mass surveillance - part 2: Technology foresight, options for longer term security and privacy improvements," in *Study - Panel for the Future of Science and Technology (STOA)*. European Parliament, 2015.

[29] P. C. Van Oorschot, T. Wan, and E. Kranakis, "On interdomain routing security and pretty secure bgp (psbgp)," *ACM Transactions on Information and System Security (TISSEC)*, vol. 10, no. 3, p. 11, 2007.

[30] L. Vanbever, O. Li, J. Rexford, and P. Mittal, "Anonymity on quicksand: Using bgp to compromise tor," in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*. ACM, 2014, p. 14.

[31] R. Wails, Y. Sun, A. Johnson, M. Chiang, and P. Mittal, "Tempest: Temporal dynamics in anonymity systems," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 22–42, 2018.

[32] R. White, "Securing bgp through secure origin bgp (sobgp)," *Business Communications Review*, vol. 33, no. 5, pp. 47–47,

2003.

A Detailed dPHI protocol description

In the pseudo-code used to describe the protocol, capital letters denote structs with several fields, while minuscule letters denote single entries. The message header is denoted with H and the payload with P. A dot is used to address a specific field within a struct. The encryption function $(c, t, iv) = \texttt{enc}_k(p, a)$ denotes an authenticated encryption function (e.g. AES-GCM [20]) with plaintext p, key k and additional authentication data a. The output is ciphertext c, authentication tag t and a fresh initialization vector iv. Similarly, $p = \texttt{dec}_k(c, t, iv, a)$ is the authenticated decryption funcion which ensures the integrity of plaintext p and authentication data a. For public key operations $(priv, pub) = \texttt{ECDH}_{gen}()$ denotes the generation of a public-key private-key pair of an Elliptic-Curve Diffie Hellman algorithm. Generating a session key between two entities A and B is done using the ECDH function $k_{A-B} = \texttt{ECDH}(pub_A, priv_B) = \texttt{ECDH}(pub_B, priv_A) = k_{B-A}$, where pub_A and pub_B are the public keys of A and B respectively and $priv_A$ and $priv_B$ the private keys. The $\texttt{Assert}(S)$ function verifies that the Boolean expression S is true. If it is false, the protocol is aborted, i.e., the node drops the processing of the current message. The function $\texttt{sendMessage}(port, H, P)$ denotes the sending of the message of a node to the next node at port $port$ with header H and payload P.

A.1 Message header and routing segments

The bulk of the header is made up of two routing segments V^1 and V^2 (see Figure 4), each consisting of l routing elements that contain routing information for every node on the path. Each node A_i has a unique secret symmetric key k_i that is not shared with any other entity. It is used to encrypt the routing information R and store it in the routing segment so that it can be retrieved and authenticated later on. More formally, the encrypted information consists of a triplet (iv_i, c_i, t_i), where iv_i is a freshly generated initialization vector, c_i is the ciphertext and t_i the authentication tag of an authenticated encryption algorithm with:

$$(c_i, t_i, IV_i) = enc_{k_i}(R; sid||c_{pos-1}) \tag{9}$$

where R is the plaintext routing information that gets encrypted and $sid||c_{pos-1}$ is the additional authentication data. The routing information R consists of five fields in total.

$$R = (port1||port2||type||posV1||posV2) \tag{10}$$

The two fields $port1$ and $port2$ correspond to ingress and egress interfaces, and the $type$ field defines the role of the node in the session (e.g. midway node or entry node). The fields $posV1$ and $posV2$ point to the routing element in V^1 and V^2 where R is stored. Only the midway node W stores R in both routing segments so that for other nodes one of the pointers is **null**.

Algorithm 1 Phase 0: Setup

procedure SETUP(d)
- $(priv_s, pub_s) \leftarrow \texttt{ECDH}_{\texttt{gen}}()$
- $n_{mid} \leftarrow \texttt{random}()$
- $M \leftarrow \texttt{chooseHelperNode}()$
- $dist_M \leftarrow \texttt{maxDistanceTo}(s, M)$
- $pub_M \leftarrow \texttt{lookupPublicKey(M)}$
- $k_{s-M} \leftarrow \texttt{ECDH}(pub_M, priv_s)$
- $H.sid \leftarrow \texttt{Hash}(pub_s)$
- $(c, t, IV) \leftarrow enc_{k_{s-M}}(d||n_{mid}, H.sid)$
- $P \leftarrow (c, t, IV, pub_s)$
- $H.V^1 \leftarrow \texttt{random}()$
- $H.V^2 \leftarrow \texttt{zeros}()$
- $H.midway \leftarrow \texttt{zeros}()$
- $H.pos \leftarrow \texttt{random}(0, l-1)$
- $H.dest \leftarrow M$
- $H.status \leftarrow$ "$toHelperNode$"
- $H_s \leftarrow H$
- $\texttt{store}(H_s, priv_s, pub_s, k_{s-M}, dist_M, n_{mid})$
- $\texttt{sendMessage}(A_s, H, P)$

end procedure

A.2 Phase 0: Initialization

Algorithm 1 details the initialization performed by source s. For each path request, source s generates a fresh ECDH key pair, consisting of pub_s and $priv_s$. The source computes a session id sid by hashing the public key with a cryptographically secure hash function $sid = \texttt{Hash}(pub_s)$. This way a session is intrinsically linked to the freshly generated ECDH key pair. This idea was introduced in PHI to securely link the path setup and the transmission phase and is kept in dPHI.

The source then chooses a helper node M from a list of trusted ASes. It is assumed that the list also contains the maximum distance $dist_M$ (number of hops) between the source and the helper node M as well as the public key pub_M of M. The source uses its own private key $priv_s$ and M's public key pub_M to compute a session key k_{s-M} with $k_{s-M} = \texttt{ECDH}(priv_s, pub_M)$. The source uses this session key k_{s-M} to encrypt the destination address d as well as freshly generate a high-entropy random midway nonce n_{mid}. The routing elements in the routing segment V^1 are initiated by s with random values that are indistinguishable from real routing elements. This can, for example, be done using a cryptographically secure pseudo-random number generator with a fresh high-entropy seed value. Furthermore, s chooses a random start position $pos \in \{0, .., l-1\}$ and stores pos and V^1 locally for later reference. After initialization, the source s starts the midway request phase by sending a request message to its corresponding AS A_s. The payload of the midway request consists of the encrypted destination and the session's public key pub_s. The destination field $dest$ of the message header is set to $dest := M$, the status field to $status :=$ "$toHelperNode$" to indicate that the protocol is in the initial phase.

A.3 Phase 1: midway request

The processing of a midway request message is described in Algorithm 2. Each node on the path to M encrypts its routing information R and stores it in the routing segment V^1 using authenticated encryption. The session id sid as well as the ciphertext of the previous routing element are used as additional authentication data for the authenticated encryption algorithm. When the midway request reaches M, M decrypts the destination d and midway nonce n_{mid}. It sets the destination field $dest$ and the $midway$ field of the header to d and n_{mid} respectively. Then the message's status is set to "findMidway" and sent back to the previous node.

A.4 Phase 2: Find midwaynode W

Algorithms 3 and 4 explain Phase 2 of the dPHI protocol. The goal is to find a midway node W that is on the path P_{s-M} between s and M such that a path $P_{s-d} = P_{s-W*} \cap P_{W-d}$ meets the routing policy of the network (e.g. valley-freeness, shortest path etc.). Each node receiving a "findMidway" request decrypts its routing information in V^1 and uses the ingres information

Algorithm 2 Phase1: Midway Request

Require: $H.status ==$ "$toHelperNode$"

1: **procedure** MIDWAYREQUEST($ingres$,H,P)
2: **if** $self == H.dest$ **then**
3: $\texttt{Assert}(H.sid == \texttt{Hash}(P.pub_s))$
4: $k_{s-M} \leftarrow \texttt{ECDH}(priv_M, P.pub_s)$
5: $d||n_{mid} = \texttt{dec}_{k_{s-M}}(P.c, P.t, P.IV, sid)$
6: $H.dest \leftarrow d$
7: $H.status \leftarrow$ "$findMidway$"
8: $H.midway \leftarrow n_{mid}$
9: $H.pos \leftarrow H.pos - 1 \bmod l$
10: $\texttt{sendMessage}(ingres, H, P)$
11: **else** ▷ M not yet reached
12: $R.port2 \leftarrow \texttt{FindRouteTo}(H.dest)$
13: $R.port1 \leftarrow ingres$
14: $R.posV1 \leftarrow H.pos$
15: $R.posV2 \leftarrow$ **null**
16: $c_{prev} \leftarrow H.V^1[H.pos - 1 \bmod l][0]$
17: **if** $\texttt{isClient}(ingres)$ **then**
18: $R.type \leftarrow$ "$entryNode$"
19: **else**
20: $R.type \leftarrow$ "$V1$"
21: **end if**
22: $H.V^1[H.pos] \leftarrow \texttt{enc}_{k_i}(R, H.sid||c_{prev})$
23: $H.pos \leftarrow H.pos + 1 \bmod l$
24: $\texttt{sendMessage}(R.port2, H, P)$
25: **end if**
26: **end procedure**

and the destination d to decide whether or not to become the midway node. If not, the message is simply forwarded to the previous node, reachable through $R.port1$.

Once a node becomes the midway node, it prepares a midway reply n_{rep} using the midway nonce n_{mid} from the midway field. The idea behind the midway reply is to allow the source s to verify the integrity of destination d as well as V^1. To do this, it computes $n_{rep} = \texttt{Hash}(H.dest||n_{mid}||H.V^1)$ and sets the midway field in the header to n_{rep}. Furthermore, the midway node updates its routing information R with the updated $R.port2$ information to route messages to d. It also chooses a random start position in V^2 and stores it in $R.posV2$, sets $R.type =$ "$midway$", re-encrypts R, and updates its routing segment in V^1 accordingly. Note, that this updated V^1 is used by the hash function to determine n_{rep}. Then the midway node encrypts the destination field $H.dest$ in the header with its secret key so that nodes between s and W do not learn d. Lastly, the message is sent back to s via the nodes on path

Algorithm 3 Phase2: Find Midwaynode

Require: $H.status == \text{“}findMidway\text{”}$

procedure FINDMIDWAY($ingres$,H,P)
  $c_{prev} \leftarrow H.V^1[(H.pos - 1) \bmod l][0]$
  $R \leftarrow \texttt{dec}_{k_i}(H.V^1[H.pos], H.sid||c_{prev})$
  $\texttt{Assert}(R.type == \text{“}V1\text{”} \,\&\&\, R.posV1 == H.pos)$
  if $\texttt{checkIfNewMidway}(self, R.port1, H.dest)$ **then**
    $R.type \leftarrow \text{“}midway\text{”}$
    $n_{mid} \leftarrow H.midway$
    $R.posV2 \leftarrow \texttt{random}(0, l-1)$
    $R.port2 \leftarrow \texttt{FindRouteTo}(H.dest)$
    $H.V^1[H.pos] \leftarrow \texttt{enc}_{k_i}(R, H.sid||c_{prev})$
    $H.midway \leftarrow \texttt{Hash}(H.dest||n_{mid}||H.V^1)$
    $H.dest \leftarrow \texttt{enc}_{k_i}(H.dest, H.sid)$
    $H.status \leftarrow \text{“}midwayReply\text{”}$
  end if
  $H.pos \leftarrow H.pos - 1 \bmod l$
  $\texttt{sendMessage}(R.port1, H, P)$
end procedure

Algorithm 4 Phase2: Midway Reply

Require: $H.status == \text{“}midwayReply\text{”}$

procedure MIDWAYREPLY($ingres$,H,P)
  $c_{prev} \leftarrow H.V^1[(H.pos - 1) \bmod l][0]$
  $R \leftarrow \texttt{dec}_{k_i}(H.V^1[H.pos], H.sid||c_{prev})$
  $\texttt{Assert}(H.pos == R.posV1)$
  $H.pos \leftarrow H.pos - 1 \bmod l$
  $\texttt{sendMessage}(R.port1, H, P)$
  ▷ If $H.type == \text{“}entryNode\text{”}$ then $R.port1$ is a client address
end procedure

P_{s-W}. These nodes only relay the message but do not alter the routing segment.

A.5 Phase 3: handshake to d

When receiving a midway reply, source s first verifies the integrity of the received routing segment $H.V^1$. Only $dist_M$ routing elements should have been modified compared to the randomly initialized V_s^1 starting from position pos_s in consecutive order. If this is not the case the message is dropped. It then verifies that the midway field $H.midway$ is equal to $\texttt{Hash}(d||n_{mid}||H.V^1)$ to verify that the correct destination was used by midway node W and that V^1 has not been altered after processing by W. If these checks succeed, the source computes a session key k_{s-d} based on the private key of the ECDH key pair it generated for this session, as well as d's publicly available longterm key pub_d. It uses this session key to encrypt V^1 and sends it together with the public key pub_s to d in a handshake message. The process of this handshake initiation is covered in Algorithm 5.

Algorithm 5 Phase 3: Initiate handshake

Require: Client s receives message with $H.status == \text{“}midwayReply\text{”}$

procedure INITHANDSHAKE($ingress, H, P$)
  $H_s, priv_s, pub_s, dist_M, k_{s-M}, \leftarrow restore()$
  $\texttt{Assert}(H.sid == H_s.sid \,\&\, \&H.pos == H_s.pos)$
  /* Verify that only routing segments in V^1 have been modified between positions $H_s.pos$ and $(H_s.pos + dist_M \bmod l)$ */
  $pointer = H_s.pos + dist_M + 1 \bmod l$
  while $pointer \neq H_s.pos$ **do**
    if $H.V^1[pointer] \neq H_s.V^1[pointer]$ **then**
      $\texttt{Abort}()$
    end if
    $pointer = pointer + 1 \bmod l$
  end while
  $n_{rep} \leftarrow \texttt{Hash}(d||n_{mid}||H.V^1)$
  $\texttt{Assert}(n_{rep} == H.midway)$
  ▷ All checks valid, send message to d
  $k_{s-d} = ECDH(pub_d, priv_s)$
  $(c, t, IV) \leftarrow \texttt{enc}_{k_{s-d}}(H.V^1, sid)$
  $P \leftarrow c, t, IV, pub_s$
  $H.status \leftarrow \text{“}handshakeToW\text{”}$
  $\texttt{store}(V^1)$
  $\texttt{sendMessage}(ingress, H, P)$
end procedure

Algorithm 6 covers communication of nodes on the path P_{s-W} that forward the message to the midway node W according to the routing information stored in $H.V^1$. The midway node uses its secret key to decrypt the destination field $d = dec_{k_i}(H.dest)$ and sets $H.dest = d$. It then chooses a random seed $seed$ and uses a cryptographically secure pseudo-random number generator to initiate the second routing segment V^2 with $V^2 = CPRNG(seed)$. Furthermore, it determines the maximum expected number of hops $dist_d$ for the message to reach d. It uses authenticated encryption to encrypt $seed$ and $dist_d$ with W's routing element in V^1 as additional authentication data and stores it in the header's midway field. The midway node serves as the bridge between nodes writing their routing information in V^1 and V^2. It therefore re-encrypts its routing segment and stores it in V^2 at position $posV2$ (so that $V^1[posV1]$ and $V^2[posV2]$ contain the same plain-

Algorithm 6 Phase3: Handshake to W

Require: $H.status ==$ "*handshakeToW*"

procedure HANDSHAKETOW($ingress$,H,P)
  $c_{prev} \leftarrow H.V^1[(H.pos - 1 \bmod l][0]$
  $R \leftarrow \mathtt{dec}_{k_i}(H.V^1[H.pos], H.sid||c_{prev})$
  $\mathtt{Assert}(H.pos == R.posV1)$
  if $R.type ==$ "*midway*" **then**
    $H.dest \leftarrow \mathtt{dec}_{k_i}(H.dest, H.sid)$
    $seed \leftarrow \mathtt{random}()$
    $H.V^2 \leftarrow \mathtt{CPRNG}(seed)$
    $dist_d \leftarrow \mathtt{maxDistanceTo}(H.dest)$
    $c_{prevV2} \leftarrow H.V^2[(R.posV2 - 1 \bmod l][0]$
    $H.V^2[R.posV2] \leftarrow \mathtt{enc}_{k_i}(R, H.sid||c_{prevV2})$
    $c_{v1} \leftarrow H.V^1[R.posV1][0]$
    $H.midway \leftarrow \mathtt{enc}_{k_i}(seed||dist_d, sid||c_{v1})$
    $H.status \leftarrow$ "*handshakeToD*"
    $H.pos \leftarrow R.posV2 + 1 \bmod l$
  else
    $H.pos \leftarrow H.pos + 1 \bmod l$
  end if
  $\mathtt{sendMessage}(R.port2, H, P)$
end procedure

Algorithm 7 Phase3: Handshake to d

Require: $H.status ==$ "*handshakeToD*"

procedure HANDSHAKETOD($ingress$,H,P)
  if $\mathtt{isClient}(H.dest)$ **then** ▷ Message arrived, forward to client
    $R.type \leftarrow$ "*destNode*"
    $R.port2 \leftarrow H.dest$
  else
    $R.type \leftarrow$ "$V2$"
    $R.port2 \leftarrow \mathtt{findRouteTo}(H.dest)$
  end if
  $R.port1 \leftarrow ingress$
  $R.posV1 \leftarrow$ **null**
  $R.posV2 \leftarrow H.pos$
  $c_{prev} \leftarrow H.V^2[(H.pos - 1 \bmod l)[0]$
  $H.V^2[H.pos] \leftarrow \mathtt{enc}_{k_i}(R, H.sid||c_{prev})$
  $H.pos \leftarrow H.pos + 1 \bmod l$
  $\mathtt{sendMessage}(R.port2, H, P)$
end procedure

Algorithm 8 Phase 4: Prepare handshake reply

Require: Client d receives message with $H.status ==$ "*handshakeToD*"

procedure INITHANDSHAKEREPLY($ingress, H, P$)
  $\mathtt{Assert}(H.sid == \mathtt{Hash}(P.pub_s))$
  $k_{s-d} \leftarrow \mathtt{ECDH}(P.pub_s, priv_d)$
  $V_s^1 \leftarrow \mathtt{dec}_{k_{s-d}}(P.c, P.t, P.IV, H.sid)$
  $\mathtt{Assert}(H.V^1 == V_s^1)$
  $P \leftarrow \mathtt{enc}_{k_{s-d}}(H.V^1||H.V^2, sid)$
  $H.dest \leftarrow \mathtt{zeros}()$
  $H.status \leftarrow$ "*replyToW*"
  $\mathtt{store}(H)$
  $\mathtt{sendMessage}(ingress, H, P)$
end procedure

text but different ciphertexts). It then initiates the next phase of the protocol by setting $pos = posV2 + 1 \bmod l$ and sending the message via $port2$.

Should the midway node also be the exit node ($port2$ is a client address) the message is sent directly to the client. Otherwise the handshake message is forwarded to the next node specified in $port2$. Nodes between W and d retrieving the handshake message look up the destination and where to forward the package. Then, they store their routing information encrypted in V^2 the same way, as nodes between s and M have done with V^1 during the first phase of the protocol and forward the message. This process is described in Algorithm 7.

A.6 Phase 4: Handshake reply

Algorithm 8 describes the handshake reply preparation by destination d. At first, destination d verifies that sid is the hash of the received public key pub_s and computes the session key k_{s-d} using this public key and its own private key. The session key is then used to decrypt the routing segment stored in the payload and compare it with the received routing segment $H.V^1$. If they are identical, the destination d encrypts both routing segment V^1 and V^2 and uses this as the payload of the handshake reply message. It deletes the destination field from the header and sends back the message to s.

Algorithms 9 and 10 describe the processing of the handshake reply message by the routing nodes between d and s. The nodes between d and W look up the routing information in their routing segments and forward the message to W without modifying any routing segments. The midway node receiving a handshake reply message verifies that there were no unauthorized modifications in V^2. For this, the midway field is decrypted to retrieve the seed for V^2 and the distance $dist_d$ between W and d. It uses the seed to re-compute the initial value of V^2 and verifies that at most $dist_d$ routing elements have changed in V^2, starting at position pos. If this check

Algorithm 9 Phase4: Handshake Reply

Require: $H.status ==$ "$replyToW$"

procedure HANDREPLYTOW($ingress$,H,P)
  $c_{prev} \leftarrow H.V^2[(H.pos - 1 \bmod l][0]$
  $R \leftarrow \mathtt{dec}_{k_i}(H.V^2[H.pos], H.sid||c_{prev})$
  $\mathtt{Assert}(H.pos == R.posV2)$
  if $R.type ==$ "$midway$" **then**
    ▷ Verify V^2 and then switch to V^1
    $c_{v1} \leftarrow H.V^1[R.posV1][0]$
    $seed||dist_d \leftarrow \mathtt{dec}_{k_i}(H.midway, sid||c_{v1})$
    $V_s^2 \leftarrow \mathtt{CPRNG}(\mathtt{seed})$
    ▷ Only $dist_d$ elements in V^2 should have changed starting at $H.pos$
    $pointer = H.pos + dist_d + 1 \bmod l$
    while $pointer \neq H.pos$ **do**
      if $H.V^2[pointer] \neq V_s^2[pointer]$ **then**
        $\mathtt{Abort}()$
      end if
      $pointer = pointer + 1 \bmod l$
    end while
    $c_{v2} \leftarrow H.V^2[R.posV2][0]$
    $H.midway \leftarrow \mathtt{MAC}_{k_i}(c_{v1}||c_{v2}||sid)$
    $H.pos \leftarrow R.posV1 - 1 \bmod l$
    $H.status \leftarrow$ "$replyToS$"
  else ▷ Not the midway node
    $H.pos \leftarrow H.pos - 1 \bmod l$
  end if
  $\mathtt{sendMessage}(R.port1, H, P)$
end procedure

Algorithm 10 Phase4: handshake Reply to s

Require: $H.status ==$ "$replyToS$"

procedure HANDREPLYTOS($ingress$,H,P)
  $c_{prev} \leftarrow H.V^1[H.pos - 1 \bmod l][0]$
  $R \leftarrow \mathtt{dec}_{k_i}(H.V^1[H.pos], H.sid||c_{prev})$
  $\mathtt{Assert}(H.pos == R.posV1)$
  $\mathtt{sendMessage}(R.port1, H, P)$
end procedure

Algorithm 11 Phase 4: handshake finish

Require: Client c receives message with $H.status ==$ "$replyToS$"

procedure HANDSHAKEFINISH($ingress, H, P$)
  $H_s, priv_s, pub_s, k_{s-d}, n_{mid} \leftarrow restore()$
  $V_d^1||V_d^2 \leftarrow \mathtt{dec}_{k_{s-d}}(P, H.sid)$
  $\mathtt{Assert}(H_s.V^1 == H.V^1 \,\&\, \&H.V^1 == V_d^1)$
  $\mathtt{Assert}(H.V^2 == V_d^2)$
  $H.status \leftarrow$ "$transmissionPhaseToD1$"
  $\mathtt{store}(H)$
end procedure

succeeds, the pointer is switched to V^1 and the message forwarded to s.

The source s receiving the handshake reply verifies that the received routing segment V^1 matches the stored routing segment. It then decrypts the payload to verify that the routing segment received by d is the same as the one received by the source. If this is true, the handshake is complete and a secure path has been established that is linked to the session key k_{s-d}. This process is described in Algorithm 11.

Algorithm 12 Phase 5: Transmission phase

Require: $H.status ==$ "$transToD1$"

procedure TRANSMISSIONPHASED1($ingres$,H,P)
  $c_{prev} \leftarrow H.V^1[(H.pos - 1 \bmod l][0]$
  $R \leftarrow \mathtt{dec}_{k_i}(H.V^1[H.pos], H.sid||c_{prev})$
  $\mathtt{Assert}(H.pos == R.posV1)$
  $\mathtt{Assert}(R.type == ($"$midway$" OR "$V1$"$))$
  if $type ==$ "$midway$" **then**
    $c_{v1} \leftarrow H.V^1[R.posV1][0]$
    $c_{v2} \leftarrow H.V^2[R.posV2][0]$
    $m \leftarrow \mathtt{MAC}_{k_i}(c_{v1}||c_{v2}||H.sid)$
    $\mathtt{Assert}(m == H.midway)$
    $H.status \leftarrow$ "$transToD2$"
    $H.pos \leftarrow R.posV2 + 1 \bmod l$
  else
    $H.pos \leftarrow H.pos + 1 \bmod \mathrm{l}$
  end if
  $\mathtt{sendMessage}(R.port2, H, P)$
end procedure

Algorithm 13 Phase 5: Transmission phase

Require: $H.status ==$ "$transToD2$"

procedure TRANSMISSIONPHASED2($ingres$,H,P)
  $c_{prev} \leftarrow H.V^2[(h.pos - 1 \bmod l][0]$
  $R \leftarrow \mathtt{dec}_{k_i}(H.V^2[H.pos], H.sid||c_{prev})$
  $\mathtt{Assert}(R.posV2 == H.pos \,\&\, \&R.type ==$ "$V2$"$)$
  $H.pos \leftarrow H.pos + 1 \bmod \mathrm{l}$
  $\mathtt{sendMessage}(R.port2, H, P)$
end procedure

A.7 Phase 5: Transmission phase

Messages between s and d can now be exchanged without requiring a destination field *dest*. Each node on the

path $P_{s,d}$ simply forwards messages according to the stored routing information in its respective routing element in V^1 or V^2 and direction. The midway node W is responsible to switch between V^1 and V^2. Before this is done, the midway node verifies that the *midway* field contains the valid MAC of W's two routing elements (one in V^1, one in V^2) to securely link the two segments together. Source s and destination d have both stored routing segments V^1 and V^2. For every message they receive they first verify that the received routing segments match the stored routing segments before accepting it. The Algorithms describing the transmission phase for messages from s to d can be found in Algorithm 12 and 13. The reverse direction from d to s is analogous.