Privacy-Preserving Network Path Validation*

Binanda Sengupta[1], Yingjiu Li[2], Kai Bu[3], and Robert H. Deng[1]

[1] School of Information Systems, Singapore Management University
{binandas,robertdeng}@smu.edu.sg
[2] Computer and Information Science Department, University of Oregon
yingjiul@uoregon.edu
[3] College of Computer Science and Technology, Zhejiang University
kaibu@zju.edu.cn

Abstract. The end-users communicating over a network path currently have no control over the path. For a better quality of service, the source node often opts for a superior (or premium) network path in order to send packets to the destination node. However, the current Internet architecture provides no assurance that the packets indeed follow the designated path. Network path validation schemes address this issue and enable each node present on a network path to validate whether each packet has followed the specific path so far. In this work, we introduce two notions of privacy – *path privacy* and *index privacy* – in the context of network path validation. We show that, in case a network path validation scheme does not satisfy these two properties, the scheme is vulnerable to certain practical attacks (that affect the privacy, reliability, neutrality and quality of service offered by the underlying network). To the best of our knowledge, ours is the first work that addresses privacy issues related to network path validation. We design PrivNPV, a privacy-preserving network path validation protocol, that satisfies both path privacy and index privacy. We discuss several attacks related to network path validation and how PrivNPV defends against these attacks. Finally, we discuss the practicality of PrivNPV based on relevant parameters.

Keywords: Network path validation · Path privacy · Index privacy · Source authentication.

1 Introduction

Next-generation networks aim to provide more control over network paths to the end-users and service providers. More command over network paths not only enables the end-users to select paths themselves in order to get uninterrupted services, but also lets the service providers serve their users in a more reliable manner. This helps to build a robust communication network where packets traverse across the network in a fast and secure way.

Although higher control over network paths is desirable, the current Internet architecture does not support such control over paths. For example, an end-user (source) may decide a superior (e.g., high-speed) network path for communicating with a service provider (destination), and she wishes to pay higher for availing this path as per the service level agreement (SLA) with the corresponding Internet service providers (ISPs). However, the current Internet provides no guarantees that the packets would follow the same path as specified. Specifically, once a packet leaves the source node, it is beyond the control of the source. So, the (possibly malicious) intermediate nodes can forward the packets along a completely different (and inferior) path to reach the destination node. Upon receiving the packets, the destination node has no means to verify whether the packets have traversed through the specified superior path. On the other hand, in order to run their business smoothly, honest intermediate nodes present on the network path would try to maintain a better quality of service and detect a deviation from the correct execution of the protocol. To be precise, every honest on-path node has a stake in enforcing the specified path to be followed by the packets and discarding corrupted packets in order not to waste downstream resources further.

In order to verify whether packets have followed the network path specified by the source node, *network path validation* (or *path validation*) schemes come into play [28,19]. In a path validation

* This is the authors' version of the paper published in ACM Transactions on Internet Technology (the final published version is available at ACM Digital Library).

protocol, the source node enforces the network path to be followed by all the nodes present on the path in order to forward packets. Moreover, every on-path node can check whether the packets have traversed through the specified path so far. This is typically achieved by enabling on-path nodes to embed proofs in a packet such that downstream nodes can verify these proofs to validate the path. On the other hand, a malicious on-path node can inject packets of its choice (with the spoofed source) into the path. In a network protocol with *source authentication*, every node present on a network path can validate whether a packet propagated along that path originates from the designated source node. We note that path validation schemes require modifications in the existing Internet routing logic for packets. However, they are essential for the next-generation Internet architectures like SCION [47,31], NEBULA [2] and others [12].

In a path validation scheme, the source node and the destination node do not trust all the intermediate nodes present on the path (otherwise, path validation would not be required at all), and thus they do not want to leak additional information available from the network path (e.g., their personal preferences while selecting the path) to these nodes. Existing path validation schemes do not hide the network path from the intermediate nodes. As we will see shortly, revealing the path to the (possibly malicious) intermediate nodes makes these schemes vulnerable to various attacks. In this work, we introduce two privacy notions relevant to path validation: *path privacy* and *index privacy*. The notion of path privacy states that an intermediate node cannot identify other nodes present on the path. We note that a node can always identify its neighbor (predecessor and successor) nodes as a packet comes from one of them and the node has to forward the packet to the other. So, the notion of path privacy described in this work does not include the privacy of neighbor nodes. However, our notion of path privacy preserves the privacy of all other on-path nodes. On the other hand, the notion of index privacy states that an intermediate node cannot learn its own *index/node-index* (or exact position) on the path.

Why are path privacy and index privacy important? We note that path privacy does not guarantee index privacy in general. To be precise, we can find some cases where, in spite of achieving path privacy, a path validation scheme leaks the index of an intermediate node to the node itself. For example, we consider an n-hop network path where neighbor information of all n nodes are encrypted and stored *sequentially* in the path variable (say, PATH). The source node allows the i-th on-path node N_i to identify N_{i-1} and N_{i+1} by decrypting *only* the i-th ciphertext of the sequence embedded in PATH. It is not hard to see that path privacy is protected in such a scheme. However, the node N_i can easily identify its own index i from the sequence — which prevents the scheme from satisfying index privacy. We now discuss some practical situations where both path privacy and index privacy are crucial.

- **Preserving source anonymity**: In mobile crowd sensing [14], mobile nodes (e.g., smartphones) collect various types of data with the help of embedded sensors and send the data to a server for analysis (e.g., measuring air pollution level, sensing traffic congestion in an area). Similarly, in the eMbedded-Gateway-Cloud (MGC) model [24,25], a smartphone acts as a gateway connecting embedded IoT devices to a cloud sever and sends the data collected from IoT devices to the cloud server. For example, the smartphone collects data from low-power wearable fitness trackers (or patient monitoring devices) and sends the collected real-time data to a cloud server that provides utility services. In these situations, path validation without path/index privacy may reveal sensitive information (e.g., physical location) of the source node to possibly malicious on-path nodes.
- **Protecting neutrality**: In a path validation protocol without path privacy, knowledge of the revealed path can be combined with external information — which helps a malicious node to mount certain attacks. Let us consider the following example. Suppose the destination node D is a service provider, and there are two consumer nodes S_1 and S_2 that send service-requests to D. Let there be a malicious node (say, N) which is present on both paths $S_1 \sim D$ and $S_2 \sim D$. Without path privacy, N has the complete knowledge of both paths, and it can thus identify the service provider D. Now, N sharing strong business relations with S_1 (or S_2) can intentionally drop the requests sent by S_2 (or S_1) in order to favor S_1 (or S_2) with undue advantages (e.g., better service quality). Similarly, an ISP can favor a particular destination (e.g., a website for online shopping) by dropping packets destined to other nodes that provide

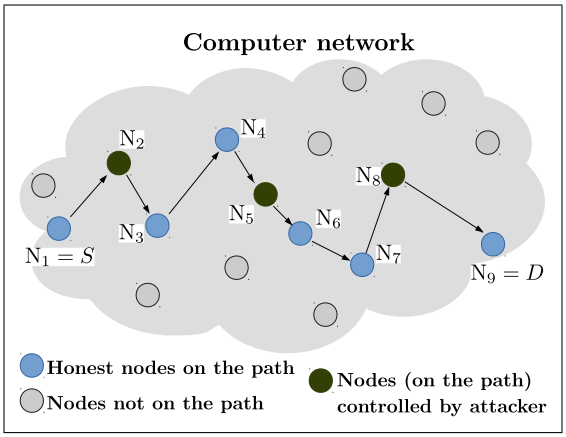

Fig. 1: A network path between the source node S and the destination node D.

similar services — which makes many end-users leave slow websites and switch to the fast one [44,23,20].

– **Preventing attacks that exploit index-information**: As we have discussed above, path privacy does not guarantee index privacy. In a path validation scheme without index privacy, an intermediate node can derive non-trivial information from the knowledge of its own index on the path. We consider the following example. Given the path-length n (including the source and destination nodes), if the $(n-1)$-th on-path node (say, N) knows its own index, then N easily derives an additional piece of information that *it is the pre-destination node* (i.e., its next-hop on the path is the destination node) — which does not protect anonymity of the destination node D. Moreover, if N happens to be the pre-destination node for two or more network paths ending at D (but originating from different source nodes), then N can selectively forward the packets sent by a particular source node only (similar to the example described above). Similar situation arises for the second on-path node which, given its node-index, can easily identify its predecessor node as the source.
– **Preventing identification of critical nodes**: In a network, there exist critical nodes that have large degrees (i.e., they are connected to many other nodes in the network and are likely to be part of many network paths), and corruption/disruption of such nodes causes widespread damages [36]. In the absence of path privacy, a network attacker compromising a single node on a network path can identify all the on-path nodes. This helps the attacker to identify one or more critical nodes in case it compromises multiple nodes each from a different network path. Then, the attacker can attempt to corrupt such a critical node in order to maximize its capability of designing efficient attack strategies that may potentially affect many other network paths.
– **Defending against an attacker having control over a small fraction of nodes**: In a network, multiple nodes can be compromised by a network attacker (or adversary), and they can collude with each other to mount certain attacks [1,17]. Given that the attacker can control a small fraction of on-path nodes in a path validation scheme with path privacy, the whole path may be revealed to the attacker if *the protocol does not satisfy index privacy*. For example, an attacker, having control over only three nodes (N_2, N_5 and N_8) as shown in Figure 1, can identify all nodes present on the network path $S \sim D$ (we note that each on-path node knows its neighbors which the node receives a packet from or sends a packet to).

Our contribution: We summarize our major contributions to privacy-preserving path validation as follows.

– We introduce, for the first time, two notions of privacy in the context of path validation: *path privacy* and *index privacy*. Path privacy ensures that an intermediate node cannot identify other on-path nodes (except its neighbors). This notion also includes the anonymity of the

Table 1: Notations used

Notation	Description	Notation	Description
$S \sim D$	network path $S = N_1 - N_2 - \cdots - N_n = D$ of length n with source S, destination D, N_i = identity of i-th on-path node	$\mathcal{E}$	symmetric-key encryption scheme
		MAC	message authentication code
		Π	pseudo-random permutation
$s_1 \| \| s_2$	concatenation of strings s_1 and s_2	$\mathcal{K}$	key space of $\mathcal{E}$, MAC and Π
H, H', H_1, H_2	hash functions	$\mathcal{S}$	space of session-identifiers
T	timestamp	$\mathcal{M}$	output space of H_2
$\mathbb{G} = \langle g \rangle$	g is a generator of the group $\mathbb{G}$	$\mathbb{Z}_q, \mathbb{F}_q$	finite field of prime order q
MPK	public parameters of KGC	MSK	master secret key of KGC
PK_i	public key of N_i	SK_i	secret key of N_i
sk_i	anonymous key shared between S and N_i	sk	non-anonymous key shared between S and D
$\mathtt{id}_s$	session-identifier	$\mathcal{P}$	random pseudonym of source (per session)
$\mathtt{path}_{\mathcal{E}}$	encrypted path	$\mathtt{r}_D$	random element associated with D (per session)
r_i	encrypted next-node info for N_i	r'_j	r_i such that $j = \Pi_{sk}(i)$
CAF	chained authentication field	A	array of verification fields
σ	bit-string containing $\mathcal{P}, \mathtt{id}_s, T, \mathtt{r}_D$	$\mathcal{A}$	adversary
σ_1	bit-string containing $\mathcal{P}, \mathtt{id}_s, T, \mathtt{r}_D, \mathtt{path}_{\mathcal{E}}$	σ_2	short digest of σ_1
$\mathtt{payload}$	payload in a packet	d_p	short digest of $(\mathtt{payload}, \sigma)$

source and destination nodes. On the other hand, index privacy guarantees that an intermediate node cannot learn its node-index on the path. These two notions are crucial for path validation in order to prevent certain attacks as discussed above.
- We construct PrivNPV, a *privacy-preserving* network path validation protocol satisfying both path privacy and index privacy. Moreover, the destination node in PrivNPV can check whether the packets originate from the designated source node.
- Once the network path and keys are set up for a PrivNPV session, an on-path node has to perform only (lightweight) symmetric-key cryptographic operations in order to validate and process subsequent payload-packets.
- We analyze the security of PrivNPV based on various attacks. In addition to the attacks relevant to path validation protocols (in general), we consider other possible attacks specific to a privacy-preserving path validation protocol.
- Finally, we discuss the practicality of PrivNPV based on relevant parameters.

Organization: The rest of the paper is organized as follows. Section 2 describes the problem and background related to this work. In Section 3, we discuss the challenges in constructing a privacy-preserving path validation protocol and the techniques we employ to address these challenges. We provide the detailed construction of PrivNPV, our privacy-preserving network path validation protocol, and discuss its properties in Section 4. In Section 5, we analyze the security of PrivNPV. We describe the practicality of PrivNPV in Section 6 and conclude the paper in Section 7.

2 Problem Definition and Background

2.1 Definition

A *network path* (or *path*) of length n between a *source* node S and a *destination* node D is an ordered collection of nodes $N_1 = S, N_2, N_3, \ldots, N_{n-1}, N_n = D$ such that packets sent by S traverse the *intermediate* nodes $N_2, N_3, \ldots, N_{n-1}$ in the same order to reach D. The i-th on-path node is identified by its node-identifier N_i. We denote such a network path either by $N_1 - N_2 - \cdots - N_n$ or simply by $S \sim D$ (omitting the intermediate nodes). For $2 \leq i \leq n-1$, the intermediate node N_i has a *predecessor node* N_{i-1} (the node which N_i receives a packet from) and a *successor node* N_{i+1} (the node which N_i sends a packet to) along the path. The source (or destination) node has only a successor (or predecessor) node along the path. The source node S decides the network path and lets D know the same. The notations we use in this paper are enlisted in Table 1. We now describe some notions related to a privacy-preserving path validation protocol as follows.

Definition 1 (Path validation [28,19,4]). *A network protocol is called a network path validation (or path validation) protocol if it satisfies both path enforcement and path verification defined as follows.*

1. *A network protocol satisfies path enforcement if a source node S decides a network path $S \sim D$ to communicate with a destination node D and every intermediate node on $S \sim D$ is directed to follow that specific path in order to send packets from S to D.*
2. *A network protocol satisfies path verification if every node present on a network path $S \sim D$ receives authenticated proofs (along with packets) from its upstream nodes, such that it can verify whether the packets have so far traversed the path specified by S.*

Path validation schemes assume that the source node knows the exact path a packet should traverse in order to reach the designated destination node. Path validation can be used in both inter- and intra-AS (autonomous system) scenarios. For either type of application scenario, we need the corresponding routing protocol to find out the network topology and regulated forwarding paths. For example, Kim et al. [19] assume that the source node has access to the information of each node (along with the information of the AS it belongs to) present on the path. The source node also knows which intermediate nodes along the path may opt for validating the path. They argue that the routing information can be obtained from the Border Gateway Protocol (BGP) [34], or the source node can be provided with this information by the respective ISPs. Similarly, for intra-AS path validation, routing information can be obtained from mainstream Interior Gateway Protocols (IGPs) like Open Shortest Path First (OSPF) protocol [27].

Definition 2 (Source authentication [19]). *A path validation protocol satisfies source authentication if a source node S authenticates each packet sent along a network path $S \sim D$, such that every node on the path can validate whether the packet originates from the designated source node S.*

Definition 3 (Path privacy). *A path validation protocol satisfies path privacy if any intermediate node present on a network path $S \sim D$ cannot identify other nodes (except its predecessor and successor nodes) on that path. In general, a set of intermediate nodes colluding with each other cannot identify any other nodes unless those nodes are predecessor or successor to at least one of the colluding nodes.*

Definition 4 (Index privacy). *A path validation protocol satisfies index privacy if any intermediate node present on a network path $S \sim D$ cannot learn its node-index on that path.*

Definition 5 (Privacy-preserving path validation). *A path validation protocol is privacy-preserving if it satisfies both path privacy and index privacy.*

An intermediate node has to identify its neighbor nodes in order to forward (or receive) packets correctly. Thus, the notion of path privacy defined above does not include the privacy of neighbor nodes. The notions of path privacy and index privacy preserve the privacy of all other nodes (including the anonymity of the source node S and the destination node D).

2.2 Related Work

Researchers have proposed various solutions to secure and verify network paths. We hereby discuss key solutions and refer interested readers to a recent survey on path validation [4] for more details. *Secure routing* protocols [18,15] are designed to find the best path (e.g., the shortest path) between a source node and a destination node such that the path finding process is secure against certain attacks. However, these protocols do not ensure that the path thus selected is actually followed by the packets sent later. In *source routing*, the source node embeds the path in packet headers such that the intermediate nodes know the exact path to be followed [40,43]. However, the intermediate nodes are assumed to be honest and follow the path correctly — this assumption does not suffice in practice where nodes present on the path can be malicious. *Traceroute* enables the intermediate nodes either to mark a passing packet with their respective identifiers (*packet marking*) or to store a packet-digest locally (*packet logging*). For packet marking, an intermediate node marks packets either *probabilistically* [35,38] or *deterministically* [7,42] — these marks are later checked by the destination node. In case of packet logging, the destination node asks for digests from the intermediate nodes in order to retrieve the path followed [37]. However, these marks and digests are not designed to be cryptographically secure — which makes them vulnerable to forgery. *Path*

enforcement enables the source node to embed the path directives in packet headers such that every on-path node can forward those packets along the specified path [3,33,9]. Unlike source routing, path directives are secure against malicious tampering. *Path verification* protocols [30,45] are similar to traceroute protocols, except that the packet-marks (or packet-digests) in a path verification protocol are cryptographically secure in that a malicious node cannot forge them.

Path validation protocols achieve both path enforcement and path verification. There exist a few path validation schemes in the literature. All of them ensure path enforcement by including path directives in the packets. ICING [28] embeds, in a packet, a verification field for each intermediate node. The source node initially populates these verification fields with authenticators. As the packet passes through each intermediate node, the node verifies the proofs (that were computed by its upstream nodes) present in its verification field. It also inserts proofs into each of the verification fields corresponding to its downstream nodes. Thus, every on-path node can verify whether a packet has traversed the path specified by the source node. In the origin and path trace (OPT) protocol [19,46], each intermediate node lets the source and destination nodes know a secret key generated for a session. Based on these keys, the source node later computes message authentication codes (MACs)[4] and embeds them in the corresponding verification fields present in a packet-header. Each on-path node can check, using its verification field and a proof sent by its predecessor node, if the packet has followed the designated path so far. These proofs form a chain of MACs, and the destination node validates the path by verifying the proof sent by its predecessor node. The design of the orthogonal sequence verification (OSV) [5,6] protocol is similar to that of OPT, except that OSV uses orthogonal sequences to make the generation of the verification fields and proofs faster.

Unlike the path validation schemes described above, alibi routing [22] addresses avoidance routing, where it is validated if the packets have avoided traversing through certain forbidden nodes (or a geographic region). The idea is to select a trusted node located far from a forbidden node and enforce the packets traverse through this trusted node. If a packet passes through both of the nodes, it encounters much higher latency compared to when it traverses through the trusted node only — this difference in latency can be detected by the destination node.

Changes required in the current Internet architecture to incorporate path validation: Path validation requires modifications in the existing Internet routing logic for packets, that are necessary to achieve stronger security guarantees, i.e., path enforcement and path verification. Path validation schemes demand more computational logic on routers which is required for parsing packet headers and performing cryptographic operations. However, this computational logic can be efficiently implemented in both software [28,19] and hardware [28]. On the other hand, the routing protocol also needs to be updated — which can be done through firmware/software upgrade on routers. In OPT [19], an autonomous system (AS) can announce its path validation functionality within BGP update messages — which enables end-hosts to get the information required for deciding a path.

On the possibility of extending similar protocols to achieve path/index privacy: We now discuss whether the existing path validation schemes (or apparently similar protocols) can be extended incrementally in order to design a path validation protocol with path/index privacy.

Extending ICING: In ICING [28], apart from the path variable (say, PATH) containing the whole network path, a verification field is embedded in the packet for each on-path node. An intermediate node (say, N_i) identifies each of its downstream nodes (i.e., $N_{i+1}, N_{i+2}, \ldots$) from PATH and inserts proofs $\pi_{i,i+1}, \pi_{i,i+2}, \ldots$ into the verification fields $V_{i+1}, V_{i+2}, \ldots$, respectively. We note that the proofs $\pi_{i,i+1}, \pi_{i,i+2}, \ldots$ are computed using the public keys of the corresponding downstream nodes $N_{i+1}, N_{i+2}, \ldots$ — which requires N_i to identify these nodes. Similarly, the node N_i has to identify its upstream nodes $N_1, N_2, \ldots, N_{i-1}$ in order to verify the proofs $\pi_{1,i}, \pi_{2,i}, \ldots, \pi_{i-1,i}$ present in the verification field V_i. As each node has to know every other node present on the path, path/index privacy cannot be achieved without making non-trivial changes in the design of ICING.

[4] Given a message and a secret key, a MAC scheme outputs a "digest" for the message. A MAC scheme is secure if it is computationally hard to produce the digest for a message m, given that the digest for m is not available already.

Extending OPT/OSV: OPT [19,46] involves a key setup phase where a special packet P circulates along the path. In order to enable all nodes to identify their neighbors, P embeds the whole path (in a variable PATH) in clear — thus neither path privacy nor index privacy is achieved in OPT. To achieve path privacy, the whole path must not be given in clear, but there should be some mechanism such that each node knows its successor node to forward packets. Let us consider the scenario if PATH were not embedded in P. Even then, an intermediate node could learn its index as follows. Each node in OPT *appends* its (encrypted and authenticated) secret key to P. So, a node can simply count the number of such secret keys already appended to P in order to get its own index. Lastly, in the OPT protocol, origin-path-verification (OPV) fields are included in the header of a packet in order to enable validation. These fields are ordered according to the node-indices, so that a node can identify and validate the corresponding OPV value — this reveals respective indices of the nodes. In order to extend OPT to achieve path/index privacy, these issues must be addressed — which requires substantial changes in the OPT design. We note that similar issues arise in the orthogonal sequence verification (OSV) [5,6] protocol which borrows similar design from OPT.

Using onion routing/Tor: In onion routing [41] and Tor [13], the source node encrypts packets in a specific order (using several layers) such that each intermediate node can decrypt only one layer and pass this partially decrypted packet to its next hop (information of the next node is obtained from this partially decrypted packet). Finally, when the packet reaches the destination node, the destination node decrypts the last layer of encryption to retrieve the original payload. These techniques appear to be probable solutions for privacy-preserving network path validation. However, onion routing/Tor provides neither path verification nor source authentication. In order to achieve path verification, each intermediate node must obtain authenticated proofs [28,19] from the upstream nodes, such that it can verify all the nodes on the path (up to that node) using those proofs. We note that encryptions do not provide such authenticated proofs. It is not straightforward to design such a privacy-preserving path validation protocol without revealing the mapping between a proof and the node that generates it (and still enabling a downstream node to verify whether the proof has been generated by that node). The key setup phase for transmitting a packet through the Tor network is expensive due to $O(n^2)$ rounds of communication with n Tor-routers (in order to set up a key with each Tor-router N_i, it requires communication between pairs $N_1 - N_2, N_2 - N_3, \ldots, N_{i-1} - N_i$). As the number of routers in a typical Tor-circuit is quite small (e.g., 3), this cost is not too high. However, this technique is not suitable for privacy-preserving path validation where the number of on-path nodes can be large (say, 40). We note that Catalano et al. [8] later reduced the number of communication rounds required for setup to $O(n)$. Along with other techniques, we exploit the anonymous key-agreement technique of [8] for our privacy-preserving path validation scheme.

2.3 An Anonymous Key-Agreement Protocol

Catalano et al. [8] proposed a *certificateless anonymous key-agreement* protocol between nodes present in a network. In this protocol, a node is associated with an *identity* (say, ID), and a trusted party, called the *Key Generation Center* (KGC), issues a partial secret key p_{ID} associated with ID to the node. In a *one-way* anonymous key-agreement protocol, a node is allowed to authenticate itself, without revealing its identity, to another node. However, the former node is able to identify the latter node that it authenticates itself to.

Let $\mathbb{G} = \langle g \rangle$ be a multiplicative group of prime order q, where g is a generator of the group, and $H : \{0,1\}^* \to \mathbb{Z}_q$ and $H' : \{0,1\}^* \to \mathcal{K}$ be two hash functions, where $\mathcal{K}$ is the output space of H'. The KGC selects a random element x from $\mathbb{Z}_q$ and sets $y = g^x \in \mathbb{G}$; it outputs the master secret key $MSK = x$ and the public parameters $MPK = (q, \mathbb{G}, g, y, H, H')$.

For a node with identity ID, the KGC verifies ID, selects a random elements $a_{ID} \in \mathbb{Z}_q$ and sets $b_{ID} = g^{a_{ID}}$. The KGC uses the master secret key $MSK = x$ to compute $c_{ID} = a_{ID} + H(ID||b_{ID})x$ and sends the partial secret key $p_{ID} = (b_{ID}, c_{ID})$ to the node. Once the node gets p_{ID} from the KGC, it selects a random element $x_{ID} \in \mathbb{Z}_q$ and sets $y_{ID} = g^{x_{ID}}$. The node outputs the public key $PK_{ID} = (b_{ID}, y_{ID})$ and the secret key $SK_{ID} = (c_{ID}, x_{ID})$.

The identities and the corresponding public keys of all nodes are maintained in a *public list* so that any node in the network can search for the credentials of another node in order to establish

a shared key anonymously. Suppose a node N_1 wants to establish a shared key with another node N_2, such that N_2 cannot identify N_1. N_1 selects a random element $w \in \mathbb{Z}_q$ and sets $\mathcal{P} = g^w$ as its *pseudonym*. It gets N_2's public key $PK_2 = (b_2, y_2)$ from the public list of identities. Given PK_2, the node N_1 computes the shared key $sk_2 \leftarrow H'(z_{2,1}||z_{2,2})$, where $z_{2,1} = (b_2 y^{H(N_2||b_2)})^w$ and $z_{2,2} = y_2^w$. On the other hand, given $\mathcal{P}$ and the secret key $SK_2 = (c_2, x_2)$, the node N_2 computes the shared key $sk_2 \leftarrow H'(z_{2,1}||z_{2,2})$, where $z_{2,1} = \mathcal{P}^{c_2}$ and $z_{2,2} = \mathcal{P}^{x_2}$. We note that the shared keys computed by N_1 and N_2 are same as both of them compute the same values $z_{2,1} = g^{wc_2}$ and $z_{2,2} = g^{wx_2}$.

3 Challenges for Constructing a Network Path Validation Protocol with Path/Index Privacy

In this section, we first discuss the challenges for constructing a path validation protocol with path/index privacy. Then, we describe the techniques we use to address these challenges.

Challenges: We mention the design challenges as follows.

- The source node in existing path validation schemes embeds the whole network path in a packet such that each on-path node can know the path and check, using proofs computed by its upstream nodes, whether the packet has actually traversed that path. The main challenge in designing a privacy-preserving path validation protocol is to hide the path from the intermediate nodes with an assurance that these nodes still can validate the path (without knowing it) and forward packets correctly.
- One possible way to achieve path privacy is to encrypt individual nodes present on the path so that an intermediate node can decrypt only the ciphertext intended for it (to obtain validation and forwarding information). This requires the source node and an intermediate node to compute a unique shared session key such that the intermediate node can decrypt only the ciphertext which was generated by the source node using the same key. However, the source node has to then establish a separate secure channel for each of the intermediate nodes in order to set up the corresponding session key — which demands much communication overhead (especially, for a path with a large number of nodes).
- Even if the source node establishes a dedicated secure channel with an intermediate node in order to set up a session key, source anonymity is still not preserved since setting up such a channel requires the credentials (e.g., public key) of the source node to be known to the intermediate node.
- We recall that path privacy does not guarantee index privacy, and a path validation protocol achieving path privacy (but not index privacy) is still vulnerable to certain attacks as discussed in Section 1. In order to achieve index privacy, the protocol must be designed in such a way that intermediate nodes cannot learn their respective node-indices. To be precise, the order of the ciphertexts must not reveal node-indices on the path.

Our approach to address the challenges: We adopt the following techniques to address the preceding challenges. We provide the detailed construction of our privacy-preserving path validation protocol satisfying both path privacy and index privacy in Section 4.

- In the setup phase of our path validation protocol, the session keys (shared between the source node S and the intermediate nodes) are generated in such a way that S does not need to form a dedicated secure channel for each intermediate node in order to communicate the corresponding shared session key. Moreover, in order to *not* reveal the identity of S to the intermediate nodes, we use a one-way anonymous key-agreement protocol [8], where S picks a random *pseudonym* for a session and computes the session keys based on a secret associated with this pseudonym and the respective public keys of the intermediate nodes. S embeds this pseudonym in a setup-packet and sends it along the network path. An intermediate node can compute the same session key using its secret key and the pseudonym embedded in the setup-packet. In addition, the source node and the destination node agree upon another (non-anonymous) session key that enables the destination node to authenticate the source node.

- The source node encrypts each node present on the path. The successor-node information for an intermediate node N is encrypted in such a way that only N can decrypt its corresponding ciphertext using its session key (on the other hand, N cannot decrypt the ciphertexts intended for other nodes).
- Message authentication codes (MACs) are employed in a chained fashion [19], such that a single incorrect MAC makes all the subsequent MACs in the chain invalid. To be specific, the source node embeds in a packet a verification field for each on-path node. The MAC present in the verification field corresponding to a particular on-path node takes as input another MAC that is computed using the secret key of the predecessor node of that on-path node — which forms a chain of MACs in the same order as that of the nodes present on the network path. So, if the MAC computation is corrupted somewhere along the path (e.g., bypassing some honest on-path nodes, or changing the order of MAC computations), the next honest on-path node can detect the same as the MAC verification at its end will fail. Thus, an on-path node can validate all MACs (computed by its upstream nodes) by verifying only the MAC sent by its predecessor node.
- For each intermediate node N, the source node S embeds a verification field in the packet such that N can check if all the upstream nodes have followed the protocol correctly. However, in order to preserve index privacy, N must not learn its index from the list of verification fields. To address this concern, we use a pseudo-random permutation[5] (PRP) to shuffle the verification fields. This shuffling is done by S using its secret key (that is not shared with any of the intermediate nodes). Moreover, in order to let N correctly identify the verification field intended for it, S encrypts N's permuted index using the session key shared with N. This can be done in a similar way as described above for encrypting the successor-node information. In fact, both of them are encrypted together in our path validation protocol, and an intermediate node uses its session key to decrypt the corresponding ciphertext and obtains both information.
- Pseudo-random permutations (PRPs) are typically defined for large domains (e.g., 128-bit AES). On the other hand, the number of nodes present on a network path is comparatively much smaller (15–20, on an average). Thus, we need PRPs with small domains for our path validation protocol. There exist a few small-domain PRP constructions in the relevant literature. Such a PRP (e.g., FastPRP [39] — which is efficient and can be applied to arbitrarily small domains) is suitable for our path validation protocol.

4 PrivNPV: A Privacy-Preserving Network Path Validation Protocol

In this section, we describe PrivNPV, the first privacy-preserving network path validation protocol. We assume that a source node S sets up a session with a destination node D, where S decides a path of length n and lets D know the specified path. Figure 2 illustrates the steps performed by the on-path nodes during the setup and payload-forwarding phases.

Long-Term Keys for Node-Identifiers: In order to achieve source anonymity in PrivNPV, we use the *anonymous* key-agreement protocol [8] described in Section 2.3. We recall that the master secret key and the public parameters of the KGC are $MSK = x$ and $MPK = (q, \mathbb{G}, g, y, H, H')$, respectively, where $\mathbb{G}$ is a multiplicative group of prime order q, g is a generator of $\mathbb{G}$, and H, H' are two hash functions. For ease of representation, as in [8,28], we describe our protocol considering $\mathbb{G}$ as a multiplicative group. In practice, operations in $\mathbb{G}$ can be performed much more efficiently for an (additive) elliptic curve group [21] than for a multiplicative group of the same order. Thus, for efficiency reasons, $\mathbb{G}$ is typically realized as an elliptic curve group (over the finite field $\mathbb{F}_q$) [8,28]. We note that multiplication and exponentiation operations in a multiplicative group are equivalent to addition and scalar multiplication operations, respectively, in an additive group.

In order to enable S and D to establish a (*non-anonymous*) shared key also, we use Diffie-Hellman key exchange protocol [11] as follows. In addition to the partial secret key p_{ID} (see Section 2.3), the KGC issues another secret key u_{ID} to a node with identity ID. The node outputs the public key $PK_{ID} = (b_{ID}, y_{ID}, v_{ID} = g^{u_{ID}})$ and the secret key $SK_{ID} = (c_{ID}, x_{ID}, u_{ID})$. Suppose N_1 and N_2 want to establish a non-anonymous shared key with each other. N_1 and N_2

[5] A pseudo-random permutation over a domain $\mathfrak{D}$ is computationally indistinguishable from a random permutation over $\mathfrak{D}$.

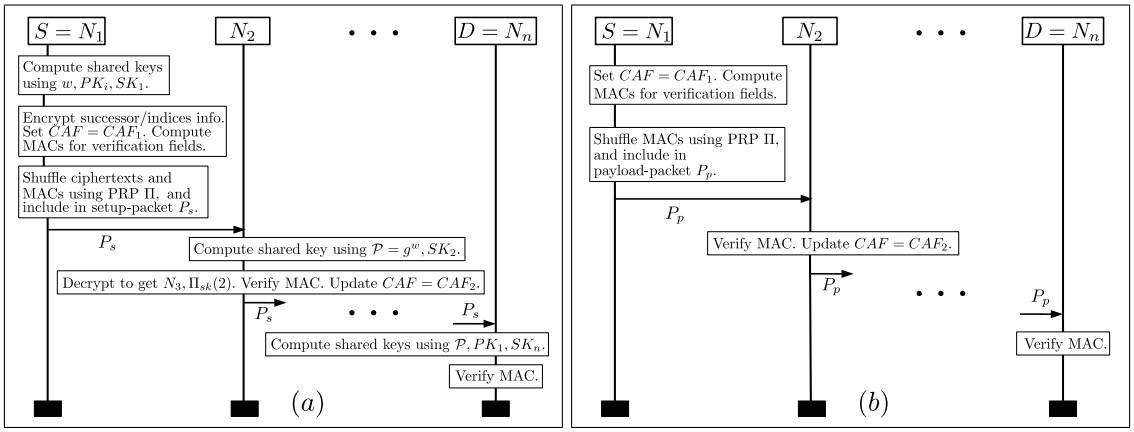

Fig. 2: An overview of the steps performed by the on-path nodes in PrivNPV during: (a) the setup phase and (b) a payload-forwarding phase.

compute $H'(v_2^{u_1})$ and $H'(v_1^{u_2})$ as their secret keys, respectively. Since $v_2^{u_1} = v_1^{u_2} = g^{u_1 u_2}$, they compute the same secret key.

The secret key-public key pair $((c_{ID}, x_{ID}), (b_{ID}, y_{ID}))$ are used for anonymous key agreement, and the secret key-public key pair (u_{ID}, v_{ID}) is used for non-anonymous key agreement. In PrivNPV, the shared session keys obtained from these long-term keys are used to validate packets.

4.1 Setup Phase for Path and Keys

Suppose a source node selects a network path $N_1 - N_2 - \cdots - N_n$ to communicate with a destination node, where N_i is the publicly known *identity* associated with the i-th on-path node. We have $S = N_1$ as the source node, $D = N_n$ as the destination node and $N_2, N_3, \ldots, N_{n-1}$ as the intermediate nodes. Let $\mathcal{E} = (\mathsf{KeyGen}_e, \mathsf{Enc}, \mathsf{Dec})$ be a secure symmetric-key encryption scheme and $\mathsf{MAC} = (\mathsf{KeyGen}_m, \mathsf{MACS}, \mathsf{MACV})$ be a secure message authentication code. Let Π be a secure pseudo-random permutation (PRP) over the set of node-indices $[1, n]$. The schemes $\mathcal{E}, \mathsf{MAC}$ and Π share the same key space $\mathcal{K}$ which is equal to the output space of H'. Let $\mathcal{S}$ be the space of session-identifiers. Let $H_1 : \{0,1\}^* \to \mathcal{S}$ and $H_2 : \{0,1\}^* \to \mathcal{M}$ be collision-resistant hash functions,[6] where $\mathcal{M}$ is the output space of H_2. We refer to Section 6.1 for possible realization of these cryptographic primitives.

Processing at the source node: Let T be the current timestamp. We assume that on-path nodes in a session are loosely time synchronized (e.g., using the network time protocol (NTP) [26]). The source node S selects a random element $w \in \mathbb{Z}_q$ and sets $\mathcal{P} = g^w$ as its *pseudonym*. For the current session, S selects a random element $\mathtt{r}_D \in \{0,1\}^{128}$ for the destination node D and computes a session-identifier $\mathtt{id}_s = H_1(\mathcal{P}||T||\mathtt{r}_D)$. The random element $\mathtt{r}_D$ serves the purpose of generating different session-identifiers for different destination nodes. S performs the following steps.

– For each $i \in [2, n]$, S establishes a shared session key with the i-th on-path node N_i. It searches N_i in the public list of identities in order to obtain N_i's public key $PK_i = (b_i, y_i, v_i)$. Given $\mathtt{id}_s$, the source node S computes the following session keys

$$\begin{aligned} &sk_i \leftarrow H'(z_{i,1}||z_{i,2}||\mathtt{id}_s) \qquad \text{for each } i \in [2, n], \\ &sk_1 \leftarrow sk_n,\ sk \leftarrow H'(v_n^{u_1}||\mathtt{id}_s), \end{aligned} \tag{1}$$

where $z_{i,1} = (b_i y^{H(N_i||b_i)})^w$ and $z_{i,2} = y_i^w$ for $i \in [2, n]$. The session key shared between S and N_i for the particular session $\mathtt{id}_s$ is sk_i. We note that S and D share two session keys sk_n (using anonymous agreement) and sk (using non-anonymous agreement).

[6] For a collision-resistant hash function, it is computationally hard to find two inputs whose hash values are equal.

σ_1	$CAF = CAF_1$	r'_1	r'_2	$\cdots$	r'_{n-1}	r'_n
		$A[1]$	$A[2]$	$\cdots$	$A[n-1]$	$A[n]$

Fig. 3: Initial structure of the setup-packet P_s.

- S encrypts the network path as $\mathtt{path}_{\mathcal{E}} \leftarrow \mathsf{Enc}_{sk_1}(n||N_1||N_2||\cdots||N_{n-1}||N_n)$. S takes a bit-string $\sigma_1 = \mathcal{P}||\mathtt{id}_s||T||\mathfrak{r}_D||\mathtt{path}_{\mathcal{E}}$ and computes a short digest $\sigma_2 = H_2(\sigma_1)$. S computes the PRP Π over the set $[1, n]$ (using the secret key sk) in order to obtain the permuted indices $\Pi_{sk}(1), \Pi_{sk}(2), \ldots, \Pi_{sk}(n)$. Then, for each on-path node, S encrypts that node's successor information and its permuted index as

$$\begin{aligned} r_i &\leftarrow \mathsf{Enc}_{sk_i}(\sigma_2||N_{i+1}||\Pi_{sk}(i)) \qquad \text{for each } i \in [1, n-1], \\ r_n &\leftarrow \mathsf{Enc}_{sk_n}(\sigma_2||N_1||\Pi_{sk}(n)). \end{aligned} \tag{2}$$

- S shuffles the sequence $R_1 = \{r_1, r_2, \ldots, r_n\}$ of ciphertexts using Π (and the secret key sk) in order to form another sequence $R_2 = \{r'_1, r'_2, \ldots, r'_n\}$ such that $r'_{\Pi_{sk}(i)} = r_i$ for each $1 \le i \le n$. We note that $r'_{\Pi_{sk}(i)}$ is the ciphertext intended for the i-th on-path node.
- The source node S processes the setup-packet P_s as follows. S includes the bit-string σ_1, the sequence R_2, an array A of verification fields and a chained authentication field CAF in P_s (see Figure 3). S computes the initial CAF value as $CAF_1 = \mathsf{MACS}_{sk_1}(\sigma_2)$. Then, it sets $N_0 = N_n$ and includes $CAF = CAF_1$ and $A[\Pi_{sk}(1)] = \mathsf{MACS}_{sk_1}(\sigma_2||N_0)$ in the packet P_s. For each index $i \in [2, n-1]$, S computes a CAF value $CAF_i = \mathsf{MACS}_{sk_i}(CAF_{i-1})$ and sets $A[\Pi_{sk}(i)] = \mathsf{MACS}_{sk_i}(\sigma_2||N_{i-1}||CAF_{i-1})$ in P_s. S also sets $A[\Pi_{sk}(n)] = \mathsf{MACS}_{sk}(\sigma_2||N_{n-1}||CAF_{n-1})$ in P_s. We note that, for any $i \in [2, n-1]$, the value CAF_i is computed using MACS that takes CAF_{i-1} and sk_i as input — which forms a chain of MACs which ensures that one invalid MAC computation in this chain makes all the subsequent MACs invalid.
- Finally, S sends the setup-packet P_s to its successor node N_2 on the path.

Processing at an intermediate node: The intermediate node N_i $(2 \le i \le n-1)$ processes the incoming setup-packet P_s as follows.

- N_i parses the bit-string σ_1 as $\mathcal{P}||\mathtt{id}_s||T||\mathfrak{r}_D||\mathtt{path}_{\mathcal{E}}$ and checks whether $\mathtt{id}_s \stackrel{?}{=} H_1(\mathcal{P}||T||\mathfrak{r}_D)$. It also computes $\sigma_2 = H_2(\sigma_1)$.
- Given $\mathcal{P}$ and the secret key $SK_i = (c_i, x_i, u_i)$, the node N_i computes the session key

$$sk_i \leftarrow H'(z_{i,1}||z_{i,2}||\mathtt{id}_s), \tag{3}$$

  where $z_{i,1} = \mathcal{P}^{c_i}$ and $z_{i,2} = \mathcal{P}^{x_i}$. We note that, as both S and N_i compute the same values $z_{i,1} = g^{wc_i}$ and $z_{i,2} = g^{wx_i}$, the session key sk_i computed in Eqn. 3 is same as that in Eqn. 1.
- N_i uses the session key sk_i to decrypt the elements of $R_2 = \{r'_1, r'_2, \ldots, r'_n\}$ one by one and checks whether the plaintext thus obtained begins with $\sigma_2 = H_2(\sigma_1)$. Among these ciphertexts, only one r'_j is decrypted correctly (for some $j \in [1, n]$). We note that all other ciphertexts present in R_2 were originally encrypted (by S) using keys *different* from sk_i. Thus, decrypting these other ciphertexts using sk_i produces random plaintexts that, with high probability, do not begin with σ_2.
- N_i obtains $(\Pi_{sk}(i), N_{i+1})$ after decrypting r'_j and checks if $j \stackrel{?}{=} \Pi_{sk}(i)$. We note that the ciphertext $r'_{\Pi_{sk}(i)} \in R_2$ is same as $r_i \in R_1$.
- Let N'_{i-1} be the node from which N_i has received the setup-packet P_s. N_i computes $\mathtt{temp}_i = \mathsf{MACS}_{sk_i}(\sigma_2||N'_{i-1}||CAF)$ using the CAF value (which is $CAF = CAF_{i-1}$ currently) from P_s. If $\mathtt{temp}_i = A[\Pi_{sk}(i)]$, then N_i is convinced that all previous CAF values have been computed correctly — *which enables path verification by* N_i. If the path is verified to be correct, N_i computes $CAF_i = \mathsf{MACS}_{sk_i}(CAF)$ and sets $CAF = CAF_i$ in the setup-packet P_s.
  We note that if N_i and N_{i-1} have followed the protocol correctly, then $N_{i-1} = N'_{i-1}$. Otherwise, the order of the node-sequence has not been followed properly, and N_i computes MAC on an incorrect input $N'_{i-1} \neq N_{i-1}$ (which is detected by the next honest node present on the path).

d_p / σ	$CAF = CAF_1$	$A[1]$	$A[2]$	$\cdots$	$A[n-1]$	$A[n]$	$\texttt{payload}$

Fig. 4: Initial structure of a payload-packet P_p.

– If any preceding verification fails, the node N_i drops the setup-packet P_s. Otherwise, N_i stores $(\texttt{id}_s, sk_i, \Pi_{sk}(i), N_{i-1}, N_{i+1})$ and sends the updated P_s to the next on-path node N_{i+1}.

Processing at the destination node: The destination node D processes the packet P_s as follows.

– D parses the bit-string σ_1 as $\mathcal{P}||\texttt{id}_s||T||\mathfrak{r}_D||\texttt{path}_{\mathcal{E}}$ and checks whether $\texttt{id}_s \stackrel{?}{=} H_1(\mathcal{P}||T||\mathfrak{r}_D)$. It computes the hash value $\sigma_2 = H_2(\sigma_1)$.
– Given $\mathcal{P}$ and the secret key $SK_n = (c_n, x_n, u_n)$, D computes the session keys

$$sk_n \leftarrow H'(z_{n,1}||z_{n,2}||\texttt{id}_s), \quad sk_1 \leftarrow sk_n, \tag{4}$$

where $z_{n,1} = \mathcal{P}^{c_n}$ and $z_{n,2} = \mathcal{P}^{x_n}$. As both S and D essentially compute the same values $z_{n,1} = g^{wc_n}$ and $z_{n,2} = g^{wx_n}$, the session key sk_n computed in Eqn. 4 is same as that in Eqn. 1.
– D uses the session key sk_1 to decrypt $\texttt{path}_{\mathcal{E}}$ to obtain the *network path* (and n). Given the public key $PK_1 = (b_1, y_1, v_1)$ of the source node $N_1 = S$ and its own secret key $SK_n = (c_n, x_n, u_n)$, the destination node D computes the session key

$$sk \leftarrow H'(v_1^{u_n}||\texttt{id}_s). \tag{5}$$

As $v_n^{u_1} = v_1^{u_n} = g^{u_1 u_n}$, the session key sk computed in Eqn. 5 is same as that in Eqn. 1.
– Let N'_{n-1} be the node from which D has received P_s. D checks if $N_{n-1} \stackrel{?}{=} N'_{n-1}$.
– D uses sk in order to compute $\Pi_{sk}(n)$ and $\texttt{temp} = \mathsf{MACS}_{sk}(\sigma_2||N_{n-1}||CAF)$, where the current CAF value present in P_s is $CAF = CAF_{n-1}$. If $\texttt{temp} = A[\Pi_{sk}(n)]$, then D is convinced that all previous CAF values have been computed correctly — *which enables path verification by D*.
– If any preceding verification fails, D drops P_s. Otherwise, D stores $(\texttt{id}_s, sk, \Pi_{sk}(n), N_{n-1})$, encrypts the string $\mathcal{P}||\texttt{id}_s||T||\mathfrak{r}_D$ using sk and sends it to S as a confirmation. The path $D \sim S$ need not be the same as $S \sim D$ (in reverse order), and it does not require any path validation.

4.2 Payload-Forwarding Phase

After the path and session keys are set up, S transmits payload-packets along the same path. The payload embedded in a payload-packet contains the actual data that the source node S wants to send to the destination node D through the decided network path. The payload can be in clear or in an encrypted format (if S and D decide to hide the content of the payload). In the latter case, S encrypts the payload using a key shared with D before putting it in the payload-packet.

Processing at the source node: Given $\texttt{id}_s$, T and $\mathfrak{r}_D$, the source node S proceeds as follows.

– S processes a payload-packet P_p as follows. It takes the bit-string $\sigma = \mathcal{P}||\texttt{id}_s||T||\mathfrak{r}_D$ and includes a short digest $d_p = H_2(\texttt{payload}||\sigma)$ along with $\texttt{payload}$ in P_p.
S includes an array A of verification fields and a chained authentication field CAF in P_p (see Figure 4). S computes the initial CAF value as $CAF_1 = \mathsf{MACS}_{sk_1}(d_p)$. S sets $N_0 = N_n$, and it includes $CAF = CAF_1$ and $A[\Pi_{sk}(1)] = \mathsf{MACS}_{sk_1}(d_p||N_0)$ in the packet P_p. For each index $i \in [2, n-1]$, S computes a CAF value $CAF_i = \mathsf{MACS}_{sk_i}(CAF_{i-1})$ and sets $A[\Pi_{sk}(i)] = \mathsf{MACS}_{sk_i}(d_p||N_{i-1}||CAF_{i-1})$ in P_p. S also sets $A[\Pi_{sk}(n)] = \mathsf{MACS}_{sk}(d_p||N_{n-1}||CAF_{n-1})$ in P_p. We note that, for any $i \in [2, n-1]$, the value CAF_i is computed using MACS that takes CAF_{i-1} and sk_i as input — which forms a chain of MACs which ensures that one invalid MAC computation in this chain makes all the subsequent MACs invalid.
– Finally, S sends the payload-packet P_p to its successor node N_2 on the path.

Table 2: Properties of path validation schemes

Schemes	Path validation	Path privacy	Index privacy	Source/destination anonymity	Source authentication
ICING [28]	✓	✗	✗	✗	✓
OPT [19,46]	✓	✗	✗	✗	✓
OSV [5,6]	✓	✗	✗	✗	✓
PrivNPV	✓	✓	✓	✓	✓[†]

[†] Source authentication is done by the destination node only.

Processing at an intermediate node: The intermediate node N_i ($2 \leq i \leq n-1$) processes the incoming payload-packet P_p as follows.

- N_i parses σ as $\mathcal{P}||\mathtt{id}_s||T||\mathfrak{r}_D$ and checks whether $d_p \stackrel{?}{=} H_2(\mathtt{payload}||\sigma)$.
- N_i stores the tuple $(sk_i, \Pi_{sk}(i), N_{i-1}, N_{i+1})$ corresponding to $\mathtt{id}_s$ (these values were computed/obtained during the setup phase). Let N'_{i-1} be the node from which N_i has received the packet P_p. The intermediate node N_i checks whether $N_{i-1} \stackrel{?}{=} N'_{i-1}$. N_i takes the CAF value (which is $CAF = CAF_{i-1}$ currently) from the incoming packet P_p and computes $\mathtt{temp}_i = \mathsf{MACS}_{sk_i}(d_p||N_{i-1}||CAF)$. If $\mathtt{temp}_i = A[\Pi_{sk}(i)]$, then N_i is convinced that all previous CAF values have been computed correctly — *which enables path verification by* N_i. In that case, N_i computes $CAF_i = \mathsf{MACS}_{sk_i}(CAF)$ and sets $CAF = CAF_i$ in the payload-packet P_p.
- If any preceding verification fails, N_i drops P_p. Otherwise, N_i sends the updated P_p to N_{i+1}.

Processing at the destination node: The destination node D processes the packet P_p as follows.

- D parses σ as $\mathcal{P}||\mathtt{id}_s||T||\mathfrak{r}_D$ and checks if $d_p \stackrel{?}{=} H_2(\mathtt{payload}||\sigma)$.
- We note that D stores $(sk, \Pi_{sk}(n), N_{n-1})$ corresponding to $\mathtt{id}_s$ (these values were computed during the setup phase). Let N'_{n-1} be the node from which D has received the packet P_p. D checks whether $N_{n-1} \stackrel{?}{=} N'_{n-1}$ and computes $\mathtt{temp} = \mathsf{MACS}_{sk}(d_p||N_{n-1}||CAF)$ using the CAF value (which is $CAF = CAF_{n-1}$ currently) from P_p. If $\mathtt{temp} = A[\Pi_{sk}(n)]$, then D is convinced that all CAF values have been computed correctly — *which enables path verification by* D.
- If any preceding verification fails, D drops the payload-packet P_p.

4.3 Properties of PrivNPV

We discuss the properties of PrivNPV as follows. Based on some of these properties, a comparison among path validation schemes is given in Table 2.

- **Path enforcement**: During the setup phase, the source node enforces the path by embedding, for each intermediate node N_i, the successor node N_{i+1} in the ciphertext r_i intended for N_i (this encryption is done using the session key shared with N_i).
- **Path verification**: Every i-th on-path node ($i \in [2, n]$) can check, using sk_i (or sk), CAF_{i-1} and $A[\Pi_{sk}(i)]$, if a packet has traversed along the previous nodes mentioned in the path. This is ensured by the chain of MACs computed according to the node-sequence. A single malicious (incorrect or out-of-order) MAC computation makes the subsequent MACs invalid. Such a mismatch in the MAC can be easily detected by the next honest intermediate node and D. Path verification is enabled in both setup and payload-forwarding phases.
- **Path privacy**: PrivNPV achieves path privacy in that each intermediate node can identify its predecessor and successor nodes only (instead of the whole path). This is ensured by encrypting the neighbor information using that node's session key. On the other hand, an intermediate node cannot decrypt the ciphertexts intended for other nodes.
  We note that the malicious intermediate nodes controlled by an attacker can combine their knowledge (information about their respective predecessor and successor nodes) to identify

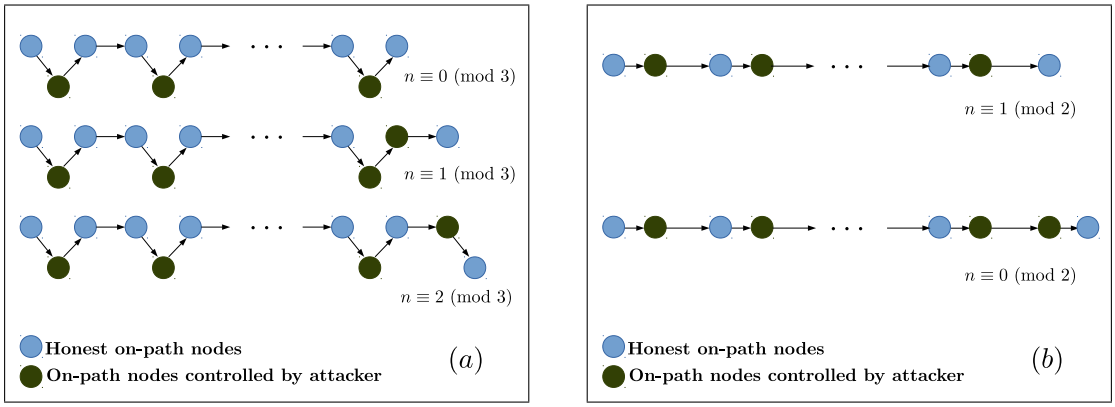

Fig. 5: The minimum number of intermediate nodes to compromise in order to reveal the whole path in PrivNPV: (a) with path privacy (but without index privacy) and (b) with both path privacy and index privacy.

parts of the network path. Suppose the attacker aims to identify the whole path by corrupting the minimum number of intermediate nodes such that: 1) every honest on-path node (i.e., the source node, or the destination node, or an honest intermediate node) has at least one malicious node as its neighbor, and 2) malicious nodes are located as far as possible from one another. Figure 5(a) illustrates the situation for PrivNPV if it had achieved only *path privacy (without index privacy)*. In order to satisfy both conditions, the number of honest nodes in between two successive malicious nodes can be at most two, and the attacker must corrupt the nodes N_2 and N_{n-1}. Thus, the attacker has to corrupt at least $\lceil \frac{n}{3} \rceil$ intermediate nodes to learn the whole network path.

- **Index privacy**: Due to the use of the PRP Π, the index of any intermediate node present on the path is not revealed from the sequence R_2 of shuffled ciphertexts.
  Figure 5(b) illustrates a situation for PrivNPV *with both path privacy and index privacy*, where an attacker attempts to identify the whole path by corrupting the minimum number of intermediate nodes. In order to satisfy both conditions mentioned above, the number of honest nodes in between two successive malicious nodes can be at most one, and the attacker must corrupt the nodes N_2 and N_{n-1}. Thus, the attacker has to corrupt at least $\lceil \frac{n}{2} \rceil$ intermediate nodes to learn the whole network path. We observe that index privacy increases this bound from $\lceil \frac{n}{3} \rceil$ to $\lceil \frac{n}{2} \rceil$.
- **Source and destination anonymity**: Due to path privacy and index privacy, source anonymity and destination anonymity are preserved in PrivNPV. For each session, $S = N_1$ chooses a random pseudonym $\mathcal{P}$ that is sufficient for an intermediate node to compute the shared session key. An intermediate node can only identify the source node by $\mathcal{P}$ and the destination node $D = N_n$ by a random element $\mathfrak{r}_D$. However, the identity of S (or D) is not revealed to any intermediate node; even N_2 (or N_{n-1}) cannot identify if its predecessor (or successor) node is the source (or destination) node. We note that the destination node D can identify S and authenticate if a packet originates from S.

  In addition to source and destination anonymity, PrivNPV achieves unlinkability across multiple sessions in the sense that an intermediate node present on two or more network paths for these sessions cannot link if the packets from these paths are generated by the same source node or destined to the same destination node. This follows from the fact that the source node selects a random pseudonym $\mathcal{P}$ and a random element $\mathfrak{r}_D$ for each session.
- **Source authentication**: The destination node D derives the shared key sk using the public key of S. During the setup/payload-forwarding phase, D validates the MAC value computed using sk and stored at $A[\Pi_{sk}(n)]$. Therefore, source authentication is done implicitly by D. In case a malicious node attempts to spoof a source, this MAC authentication fails.
- **Symmetric-key cryptography**: The payload-forwarding phase in PrivNPV involves only (fast) symmetric-key primitives — that results in fast computation at each node. The setup

phase requires public-key primitives (for computing shared session keys). However, we note that public-key operations are necessary for two parties generating a shared key without establishing a dedicated channel between them.

5 Security of PrivNPV

5.1 Security Assumptions

A computationally-bounded network attacker is considered to be an adversary $\mathcal{A}$ in privacy-preserving path validation. We assume that $\mathcal{A}$ is Byzantine (i.e., it can deviate from the protocol in an arbitrary and malicious fashion) and can corrupt some intermediate nodes (these nodes can *collude* in order to mount certain attacks collectively). The adversarial model and attacks are similar to those described in the existing path validation schemes [28,19,4]. We assume that $\mathcal{A}$ has no control over the end-hosts (i.e., S and D are considered to be honest). Otherwise, as both of them have the complete knowledge of the path, $\mathcal{A}$ is always able to know the path — which is not possible to prevent by any means. However, this assumption is rational as the end-hosts may not want to reveal the path to intermediate nodes whom they do not trust. Moreover, $\mathcal{A}$ may attempt to identify the honest on-path nodes or to learn the index of a node it controls. We require the following assumptions for PrivNPV to be secure.

- The hash functions H and H' used for key agreement are assumed to be random functions [8]. The encryption scheme $\mathcal{E}$, the MAC scheme MAC and the PRP Π used in PrivNPV are secure. The hash functions H_1 and H_2 are collision-resistant.
- Each node is identified by its identity/node-identifier (e.g., N_i), and the Key Generation Center (KGC) issues a partial secret key to the node based on its identity. The public key of each node is included in a public list available to all nodes in the network. In PrivNPV, we consider $\mathbb{G}$ to be an elliptic-curve group [21] over the finite field $\mathbb{F}_q$, where q is a 160-bit prime. Thus, the size of each secret key in $\mathbb{Z}_q$ (and the corresponding public key in $\mathbb{G}$) is 160 bits.

5.2 Security Analysis: Possible Attacks and Defenses

We discuss the possible attacks an adversary $\mathcal{A}$ can mount on a privacy-preserving path validation scheme and how PrivNPV defends against them.

- **Path-revealing attack**: A malicious intermediate node tries to learn the network path partially or fully. We note that it can always identify its predecessor and successor nodes.
  Defense: In PrivNPV, as the path is encrypted, an intermediate node cannot identify the on-path nodes except its neighbors. A malicious intermediate node, without having the secret session key of an honest node, cannot decrypt the ciphertext intended for that honest node. We note that an attacker can compromise some of the intermediate nodes and learn parts of the network path from the neighbor information of those nodes. However, as we have discussed in Section 4.3, the attacker has to compromise at least $\lceil \frac{n}{2} \rceil$ intermediate nodes in PrivNPV in order to learn the whole network path of length n.
- **Index-revealing attack**: A malicious intermediate node can attempt to learn its index on the network path to mount certain attacks similar to that described in Section 1.
  Defense: The source node in PrivNPV shuffles the ciphertexts using the pseudo-random permutation Π. As an intermediate node does not know the secret key sk for Π, it cannot learn its node-index on the network path. Moreover, as PrivNPV satisfies both path privacy and index privacy, a malicious intermediate node (even N_2 or N_{n-1}) cannot identify S and D only from the random pseudonym $\mathcal{P}$ and the random element $\mathfrak{r}_D$, respectively.
- **Counterfeit attack**: In a counterfeit attack, $\mathcal{A}$ aims to propagate an incorrect packet so that the packet passes verification. This type of attacks includes *packet-alteration* attacks (modifying a packet) and *packet-injection* attacks (injecting a packet of $\mathcal{A}$'s choice). Packet-injection attacks also include *replay* attacks where $\mathcal{A}$ injects older packets into the path.
  Defense: In a packet-alteration attack, a malicious intermediate node tries to alter a packet without being noticed by the next honest on-path node. In PrivNPV, these attacks are prevented by using short digests (σ_2 in the setup phase and d_p in the payload-forwarding phase).

The digest $\sigma_2 = H_2(\sigma_1)$ is used as an input to the encryption operations (see Eqn. 2), and $d_p = H_2(\texttt{payload}||\sigma)$ is included in the packet itself (see Figure 4). These digests are computed using a collision-resistant hash function — which ensures that tampering with the input would produce a different digest. Moreover, these digests are fed as input to MAC computations. Thus, an incorrect value of a digest would produce a different MAC value (despite having the corresponding secret key) — an honest node can easily detect this anomaly (as the MACs are chained) and drop the packet. Similarly, as all MAC, encryption and PRP operations need secret keys, it is hard for a malicious node to tamper with CAF, the verification fields of A or the permuted ciphertexts (embedded in a packet) without being detected by an honest node.

In a packet-injection attack, a malicious node S' in the network tries to impersonate another node S as the source node and inject a packet (of its choice) along a network path. In PrivNPV, the destination node D authenticates the source node S using the shared session key derived from the public key of S. Therefore, the only possibility of successfully mounting such an attack is to guess/compute the secret key of S — which is hard for the malicious node S'. Replay attacks are prevented in PrivNPV by embedding the session-identifier and timestamp in the packet (in an authenticated fashion) which can be validated by every node on the path.

– **Denial-of-service (DoS) attack**: In a DoS (or distributed DoS) attack, the adversary tries to make the on-path nodes perform memory-intensive and/or computation-intensive work. Replay attack (injecting older packets into the path) is one such example which not only increases traffic in the network but also enhances redundant computation (e.g., verification) for an honest node.

Defense: In PrivNPV, each intermediate node N_i stores only $(\texttt{id}_s, sk_i, \Pi_{sk}(i), N_{i-1}, N_{i+1})$ corresponding to a session — this small amount of storage rules out memory exhaustion of an intermediate node. In terms of computational overhead, the intermediate nodes in PrivNPV need to perform only a small number of symmetric-key cryptographic operations in a payload-forwarding phase (see Table 3). For each payload-packet P_p, these symmetric-key operations can be performed efficiently. However, in the setup phase, an intermediate node has to perform some computation-intensive work (e.g., it has to perform expensive exponentiation operations and $\frac{n}{2}$ decryption operations on an average) — which may help the adversary to effectively mount DDoS attacks on one or more intermediate nodes. In case such an attack occurs in the setup phase and the source node S does not get a confirmation (from the destination node D) of receiving the valid setup-packet P_s after a predefined timeout period, S decides a new network path to communicate with D.

As we have discussed earlier, replay attacks are hard to mount in PrivNPV.

– **Coward attack**: The adversary performs a coward attack when the attack is less likely to be detected. For example, it can mount any attack mentioned above when the path validation protocol is not being executed (e.g., when the key setup is not done, or when packets are being sent without authentication in order to achieve fast propagation).

Defense: Due to the additional computational burden needed for verification, the path-validation procedure may be invoked only when there are anomalies regarding packet loss (or delay), or when the source node and the destination node set up a new network path to be followed for communication [4]. After these issues are resolved (or the setup is done), every packet transmitted onwards is not checked for validation — which the adversary can utilize to mount some of the attacks mentioned above. In this scenario, it is hard to detect such attacks. Among the existing path-validation schemes, only OPT [19] addresses this issue by executing the path validation probabilistically (i.e., for random packets). Thus, the adversary fails to predict when a validation would be run — which makes it execute the protocol correctly all the time. We can also use this method in PrivNPV in order to prevent such coward attacks. On the other hand, path validation may be applied only to specific packets that are required to follow a specific path chosen by the source node. However, in order to ensure validation of all the packets, authentication (and verification) must be enabled for each of them. In that case, PrivNPV is still practical as only a few (efficient) symmetric-key cryptographic operations needed to be performed in the payload-forwarding phase (see Table 3 for the number of cryptographic operations performed by the nodes).

– **Out-of-order traversal**: One or more malicious nodes (controlled by $\mathcal{A}$) can send the packet through all (or some) of the specified nodes but not in the order decided by the source. It includes the case where an honest node on the path is bypassed by the malicious nodes.
Defense: In PrivNPV, the CAF values must be computed in the same order as that of the nodes present on the network path. These values form a chain of MACs, where each MAC in the chain is computed from its previous MAC using the session key of the corresponding on-path node. For an out-of-order traversal that involves at least one honest node N, the CAF value of N cannot be computed correctly without its session key — which makes all subsequent CAF values incorrect. This is detected by the next honest on-path node.

Finally, we mention some issues that path validation does not address well in general [28,19]. They remain unresolved in PrivNPV as well.

– An out-of-order traversal, that involves only some nodes controlled by $\mathcal{A}$, may not be detected in a path validation scheme (e.g., $\mathcal{A}$, having control over the nodes $N_i, N_{i+1}, N_{i+2}, N_{i+3}$, can always make a packet traverse through $N_i - N_{i+2} - N_{i+1} - N_{i+3}$ or $N_i - N_{i+3}$, and it can still generate correct CAF values using the session keys of these nodes).
– In a path-detour attack, a malicious intermediate node (say, N_i) sends a packet through an unspecified path $N_i - N'_1 - N'_2 - \cdots - N'_l - N_{i+1}$ (where the detour nodes are $N'_1, N'_2, \ldots, N'_l$). If N_i and N_{i+1} collude with each other, they can always produce correct CAF values — which makes such an attack hard to detect.
– Path validation does not ensure the delivery of a packet to the destination node D. A node can drop a packet maliciously. On the other hand, if a packet fails verification at an honest node, the node drops it to avoid wasting downstream resources.
– D cannot identify the exact node where a packet (if any) has been dropped or corrupted.
– Some of the intermediate nodes in a path validation scheme may fail to respond permanently or temporarily due to various reasons (e.g., accidentally or under the influence of certain attacks). This may result in packet losses at the destination node D. This issue can be addressed by borrowing ideas from the reliable data-transmission mechanisms of the Transmission Control Protocol (TCP) [32]. The packet losses can be detected by associating sequence numbers with the packets sent along $S \sim D$ and by enabling S to receive an acknowledgment from D for each of these packets. For a temporary disruption, the source node detects the same (e.g., based on duplicate acknowledgments or a timeout parameter) and retransmits the lost packets along the same path. On the other hand, in case some of the intermediate nodes fail permanently, recovery techniques like finding alternative path dynamically can be applied. However, as the source node in a path validation scheme initially fixes the network path and packets are allowed to traverse that particular path only, these recovery techniques are not suitable for path validation. This requires the source node to decide another path (and set up session keys with the nodes on this path) in order to further communicate with the destination node — which increases the complexity of the path validation protocol significantly.

6 Practicality of PrivNPV

6.1 Realization of Cryptographic Primitives

The AES-128 block cipher [29] can be used for symmetric-key primitives in PrivNPV as follows. The encryption scheme $\mathcal{E}$ can be instantiated using AES-128 in cipher-block-chaining (CBC) mode; the MAC scheme MAC can be realized as CMAC-AES; the PRP Π can be constructed using FastPRP [39] (FastPRP uses AES-128 to generate required pseudo-random bits). For the initial key agreement, $\mathbb{G}$ is taken to be an elliptic curve group over $\mathbb{F}_q$, where q is a 160-bit prime. Thus, for the node N_i, each component of $PK_i = (b_i, y_i, v_i)$ and $SK_i = (c_i, x_i, u_i)$ is 160-bit long. The output space for the hash function H is $\{0,1\}^{160}$. The output space for each of the hash functions H', H_1, H_2 is $\{0,1\}^{128}$, that is, $\mathcal{S} = \mathcal{K} = \mathcal{M} = \{0,1\}^{128}$. We can use SHA-256 for computing the hashes and truncate the 256-bit outputs to 128-bit values. We take the identity of a node (e.g., N_i) to be an element of $\{0,1\}^{128}$. Based on these possible instantiations, each of the following elements is 16-byte long: $\mathtt{id}_s, \sigma_2, d_p, CAF, sk, sk_2, sk_3, \ldots, sk_{n-1}, sk_n,\ A[1], A[2], \ldots, A[n], N_1, N_2, \ldots, N_n$.

Table 3: Number of cryptographic operations performed by the source node S, an intermediate node N and the destination node D during different phases of PrivNPV/OPT/ICING, respectively

Operations	Setup phase			Payload-forwarding phase		
	S	N	D	S	N	D
Exponentiation	$3n-1/0/\sim$	$2/0/\sim$	$3/0/\sim$	$0/0/n$	$0/0/n$	$0/0/n$
Hash	$2n+1/1/\sim$	$3/0/\sim$	$4/1/\sim$	$1/2/n+1$	$1/0/n+1$	$1/0/n+1$
PRP (Π)	$1/0/\sim$	$0/0/\sim$	$1/0/\sim$	$0/0/0$	$0/0/0$	$0/0/0$
Encryption/ decryption	$n+2/0/\sim$	$n^{\dagger}/1/\sim$	$2/n-2/\sim$	$0/0/0$	$0/0/0$	$0/0/0$
MAC	$2n-1/0/\sim$	$2/0/\sim$	$1/0/\sim$	$2n-1/2n-2/\ddagger$	$2/2/\ddagger$	$1/n+1/\ddagger$
PRF	$0/2/\sim$	$0/1/\sim$	$0/3/\sim$	$0/0/4n$	$0/0/n+3$	$0/0/n+3$
Authenticated encryption/decryption	$0/2/\sim$	$0/0/\sim$	$0/2/\sim$	$0/0/0$	$0/0/0$	$0/0/0$
Signature generation/verification	$0/0/\sim$	$0/1/\sim$	$0/n-2/\sim$	$0/0/0$	$0/0/0$	$0/0/0$

$\sim$ ICING does not consider a separate setup phase for a session. Each packet in ICING carries a payload along with authentication information.
† In the worst case, an intermediate node in PrivNPV tries to decrypt $n-1$ ciphertexts before it finds the ciphertext intended for it (an intermediate node finds the corresponding ciphertext after $\frac{n}{2}$ decryption operations, on an average).
‡ MAC computations in ICING use pseudorandom functions (PRFs) and hash functions; we have thus added the respective counts to that of the corresponding operations.

As Π operates over $[1,n]$, we have $\Pi_{sk}(1)$, $\Pi_{sk}(2),\ldots,\Pi_{sk}(n) \in \{0,1\}^{\lceil \log_2 n \rceil}$. A timestamp T is typically represented using 4 bytes. Each of the ciphertexts $r_1, r_2, \ldots, r_n$ is 64-byte long (after padding the plaintexts appropriately).

6.2 Storage Overhead per Packet

The storage overhead for the setup-packet P_s per on-path node is roughly attributed to an element of A (the list of verification fields) and an element of R_2 (the sequence of shuffled ciphertexts). Thus, the storage overhead for P_s (per on-path node) is around 80 bytes. On the other hand, the storage overhead for a payload-packet P_p (per on-path node) is due to an element of A — which accounts for 16 bytes.

6.3 Computation per Node

Table 3 shows cryptographic operations (to be performed by the source node S, an intermediate node N and the destination node D) during the setup and payload-forwarding phases of PrivNPV/OPT/ICING. From Table 3, we make the following observations regarding PrivNPV.

- Most of the computations are done in the setup phase.
- Only a few symmetric-key operations need to be performed in the payload-forwarding phase.
- During the setup phase, S has to perform some of the cryptographic operations for each on-path node (i.e., the number of times each of these operations to be performed by S grows linearly with the path-length n).
- The number of times any cryptographic operation to be performed by D is independent of n. This holds for an intermediate node also, except that it has to perform $\frac{n}{2}$ decryption operations (on an average) in the setup phase.

Estimation of time required per node: We estimate the time required by the source node S, an intermediate node N and the destination node D in different phases of a PrivNPV session as follows. The (additive) elliptic curve group $\mathbb{G}$ is defined over the finite field $\mathbb{F}_q$ for a 160-bit

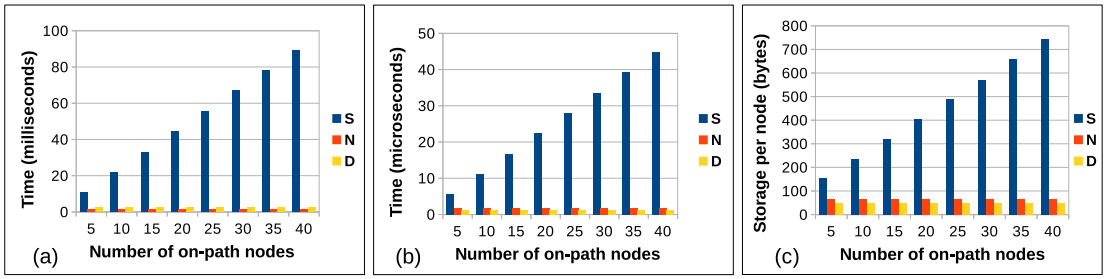

Fig. 6: Time required for performing cryptographic operations in: (a) the setup phase and (b) the payload-forwarding phase. (c) Storage required at the source S, an intermediate node N and the destination D.

prime q. According to the benchmarks given in [16], an exponentiation operation (or, equivalently, a scalar multiplication operation in the additive group) takes around 0.75 milliseconds when evaluated on a 1.83 GHz Intel Core 2 Duo processor. FastPRP [39] requires around $n \log n$ pseudo-random bits that are generated by encrypting non-negative integers using AES-128 with the secret key sk. For example, for $n = 20$, Π requires around 100 bits which can be obtained by invoking AES-128 once (i.e., AES-128$_{sk}(0)$); for $n = 40$, it requires around 240 bits which can be obtained by invoking AES-128 twice (i.e., AES-128$_{sk}(0)$ and AES-128$_{sk}(1)$). For symmetric-key cryptographic primitives involved in PrivNPV, we consider the widely used cryptographic benchmarks [10] evaluated on a 1.83 GHz Intel Core 2 Duo processor: each hashing takes 0.55 microseconds (using SHA-256), each encryption/decryption takes 0.56 microseconds (using 128-bit key AES-CBC) and each MAC operation takes 0.56 microseconds (using CMAC-AES) — assuming 64-byte inputs for these primitives. Figure 6(a) and Figure 6(b) show the time required for performing cryptographic operations in the setup and payload-forwarding phases, respectively, for varying path-length n.

In practice, the path-length n is small (e.g., 15–20, on an average). Thus, the per-session computational cost for each node is low (e.g., for $n = 20$, the nodes S, N, D take around 44.31, 1.51, 2.25 *milliseconds*, respectively, in the setup phase and 22.95, 1.67, 1.11 *microseconds*, respectively, in the payload-forwarding phase). The computational cost in the setup phase (*one-time per session*) is predominated by expensive public-key operations. However, we note that public-key operations are necessary for a pair of nodes computing a shared key without establishing a dedicated channel between them. Once the network path and session keys are set up for a session, S typically transmits many payload-packets. As the computational cost in each payload-forwarding phase is small (in the order of microseconds), these payload-packets are processed fast at each on-path node.

6.4 Storage per Node

Along with the specified network path, the source node S in a PrivNPV session (identified by $\mathtt{id}_s$) stores the following: $\mathcal{P}, \mathtt{id}_s$, T, $\mathtt{t}_D$, the keys $(sk, sk_2, sk_3, \ldots, sk_n)$ and the permuted indices $(\Pi_{sk}(1), \Pi_{sk}(2), \ldots, \Pi_{sk}(n))$ — which accounts for total $576 + n(128 + \lceil \log_2 n \rceil)$ bits. For example, this storage is around 405 bytes for $n = 20$. The destination node D stores a tuple $(\mathtt{id}_s, sk, \Pi_{sk}(n), N_{n-1})$. For example, D stores around 49 bytes for $n = 20$. Each intermediate node N_i $(2 \leq i \leq n-1)$ stores a tuple $(\mathtt{id}_s, sk_i, \Pi_{sk}(i), N_{i-1}, N_{i+1})$. Thus, for a path of length 20, each intermediate node stores around 65 bytes. Figure 6(c) shows the storage required at different on-path nodes for varying n.

6.5 Comparison among Path Validation Schemes

In this section, we compare the PrivNPV protocol with ICING [28] and OPT [19] based on certain parameters. Unlike ICING and OPT, the PrivNPV protocol offers privacy of the path being validated. However, such privacy comes at a cost. For some of these parameters, PrivNPV

Table 4: Storage and communication overhead required in path validation schemes

		S	N	D
Storage (in terms of number of items stored)	ICING (per session)	$n+1$	2	$2n+1$
	OPT (per session)	$n+2$	0	$n+2$
	PrivNPV (per session)	$2n+4$	5	4
	ICING (long term)	0	$\leq 400,000$	0
	OPT (long term)	1	1	2
	PrivNPV (long term)	3	3	3
Number of packets communicated during key setup in a session	ICING	$4n+4$		
	OPT	2		
	PrivNPV	2		

has an extra overhead compared to ICING and OPT. PrivNPV enjoys similar efficiency for the rest of the parameters.

We recall that ICING is designed based on aggregate MACs and self-certifying names. Each node in the network locally computes a unique public/private key pair. It then broadcasts the public key (self-certifying name) to other nodes. Given the public key of a node, another node computes a shared key using its private key and stores the shared key at its end. In addition to the specified path, each ICING packet contains a verification field for each on-path node. An intermediate node identifies each of its downstream nodes from the path and inserts authenticated proofs (MACs) into the respective verification fields. We note that the verification field for a particular node contains a single MAC that is an aggregation of the MACs computed by all of its upstream nodes. Each node verifies the MAC in its verification field, inserts MACs in the verification fields for its downstream nodes, and forwards the packet to its successor node. MAC computations in ICING are done using pseudorandom functions (PRFs) and hash functions.

In the key setup phase of OPT, the source node sends a packet along the path. Each intermediate node computes a secret key (by applying a pseudorandom function on the session-identifier) and puts it in the packet in an encrypted and authenticated format. The destination node decrypts these secret keys, checks their authenticity, performs authenticated encryption on them and sends them to the source node. The source node performs authenticated decryption and sets these secret keys as session keys shared with the corresponding intermediate nodes. Later, while sending a packet, the source node embeds in the packet a chain of MACs computed using the session keys such that the intermediate nodes can validate the path using these MACs.

We compare PrivNPV with ICING and OPT as follows. Storage overheads per payload-packet (per on-path node) are 42, 16 and 16 bytes for ICING, OPT and PrivNPV, respectively. In order to process packets, each node needs some (long-term/per-session) storage. Table 4 shows the comparison in terms of storage required for different nodes (the figures for ICING/OPT are taken from [19]). In PrivNPV, each node N_i has to store its long-term secret key $SK_i = (c_i, x_i, u_i)$. Per-session storage for the source node S in PrivNPV is higher as it stores all the permuted node-indices in order to populate the verification fields later. The destination node D has to store only a tuple of four elements for a PrivNPV session. An intermediate node N in PrivNPV needs a little extra amount of storage (compared to ICING/OPT) for its session key, permuted index and neighbor information.

The computational overhead in PrivNPV (compared to ICING and OPT) is due to exponentiation, encryption/decryption operations and PRP computations (see Table 3). This overhead is attributed to path/index privacy offered by PrivNPV. For a source node S transmitting a large number of payload-packets in a session, the amortized cost for these operations is reduced significantly (e.g., for $O(n)$ packets transmitted in a session, the cost per packet is constant). Therefore, this overhead is practical and justified.

In a PrivNPV session, S lets each node N_i know its permuted index and successor node by embedding the ciphertext r_i in P_s — this requires sending one packet to D along $S \sim D$. In case P_s passes the verification at D, D encrypts the string $\mathcal{P}||\mathtt{id}_s||T||\mathfrak{t}_D$ using sk and sends it to S as a confirmation. We note that, in addition to per-session communication, PrivNPV nodes joining

the network need to communicate with the KGC initially in order to obtain their respective secret key-public key pairs and to include their identities (along with the public keys) in a public list.

6.6 Hiding Path-Length

An intermediate node in PrivNPV can infer the path-length n from the size of A (or R_2). One possible way to hide the actual path-length is to pad the path by adding dummy fields (for dummy intermediate nodes) in A and R_2. As an intermediate node on the actual path derives its next-node from its respective ciphertext, these dummy nodes are never traversed. Similarly, S puts random elements into the dummy fields of A and R_2 (which are never checked/decrypted by any node). This hides the path-length to some extent (e.g., its upper bound is still revealed). If all paths are padded to be of length $n = n_{\mathsf{max}}$ (say) in order to minimize leakage further, then n_{max} should be large enough to accommodate the paths consisting of a large number (say, 40) of intermediate nodes. However, for such a large n_{max}, this *length-hiding* routing becomes inefficient (compared to that without padding) in case the path consists of a few (say, less than 5) intermediate nodes.

7 Conclusion and Future Work

Network path validation enables a source node to enforce packets to traverse along a specified network path, such that every on-path node can check if the packets have followed that path so far. In this work, we have addressed certain privacy concerns that may arise in network path validation. We have introduced two privacy notions: path privacy and index privacy. These notions are crucial to preserve privacy of on-path nodes in a path validation scheme and to defend against certain attacks mounted by a network attacker. Path privacy and index privacy also provide source anonymity and destination anonymity in the presence of malicious intermediate nodes controlled by the attacker. We have constructed PrivNPV, the first privacy-preserving network path validation protocol, that exploits mostly lightweight cryptographic operations in order to achieve both path privacy and index privacy. PrivNPV also enables the destination node to verify if the packets are indeed generated by the source node. We have analyzed the security of PrivNPV where we have considered, in addition to attacks related to path validation schemes, other possible attacks specific to privacy-preserving path validation. Finally, we have discussed the practicality of PrivNPV and compared PrivNPV with existing path validation schemes based on storage, communication and computational overhead required. We mention some future research directions related to privacy-preserving network path validation.

- Network path validation schemes are designed for next-generation Internet architecture that is more secure and robust than the current Internet, and the ability of each on-path node to validate the network path comes at a cost of reduced efficiency attributed to the cryptographic primitives used in these schemes. Thus, the latency incurred by packet processing in PrivNPV does not reach the line rate of the current Internet — which demands further investigation whether we can design a privacy-preserving network path validation scheme with lower latency that copes up with the line rate of the current Internet.
- As we have discussed in Section 5.2, a malicious intermediate node in a path validation scheme can drop or corrupt a packet, and the destination node cannot identify the exact node where a packet (if any) has been dropped or corrupted. If an intermediate node in a path validation scheme corrupts a packet, the next honest on-path node can inform the source/destination node about the probable location of the corruption (e.g., by sending another packet to the source/destination node). On the other hand, for a malicious packet drop, it is hard for the source/destination node to get notified about the probable location of the packet drop since the next honest on-path node never receives the packet. Unfortunately, for a privacy-preserving path validation scheme with path privacy (e.g., PrivNPV), the aforementioned technique does not work for packet corruption due to source/destination anonymity. It appears to be non-trivial to come up with techniques in order to address the issue for a privacy-preserving path validation scheme like PrivNPV.

– PrivNPV employs a trusted Key Generation Center (KGC) for one-way anonymous key agreement, such that an intermediate node can compute a session key shared with the source node without knowing the actual identity of the source. This shared key helps the intermediate node to validate if a packet has traversed through the specified path so far. On the other hand, PrivNPV (like many other protocols relying on trusted third parties) is vulnerable to certain attacks if the KGC is compromised by an attacker. In that case, the attacker can attempt to learn the session keys and the network paths followed by packets, or to tamper with the public parameters published by the KGC. However, without such a trusted third party, it seems to be quite challenging to generate shared session keys while preserving source anonymity.

Acknowledgments

The work of Yingjiu Li was supported, in part, by Lee Kong Chian Fellowship while at Singapore Management University. The work of Kai Bu was supported by The Natural Science Foundation of Zhejiang Province under Grant No. LY19F020050.

References

1. Majeed Alajeely, Robin Doss, Asma'a Ahmad, and Vicky H. Mak-Hau. Defense against packet collusion attacks in opportunistic networks. *Computers & Security*, 65:269–282, 2017.
2. Tom Anderson, Ken Birman, Robert M. Broberg, Matthew Caesar, Douglas Comer, Chase Cotton, Michael J. Freedman, Andreas Haeberlen, Zachary G. Ives, Arvind Krishnamurthy, William Lehr, Boon Thau Loo, David Mazières, Antonio Nicolosi, Jonathan M. Smith, Ion Stoica, Robbert van Renesse, Michael Walfish, Hakim Weatherspoon, and Christopher S. Yoo. A brief overview of the NEBULA future internet architecture. *Computer Communication Review*, 44(3):81–86, 2014.
3. Ioannis C. Avramopoulos, Hisashi Kobayashi, Randy Wang, and Arvind Krishnamurthy. Highly secure and efficient routing. In *IEEE Conference on Computer Communications, INFOCOM*, pages 197–208, 2004.
4. Kai Bu, Yutian Yang, Avery Laird, Jiaqing Luo, Yingjiu Li, and Kui Ren. What's (not) validating network paths: A survey. *CoRR*, abs/1804.03385, 2018.
5. Hao Cai and Tilman Wolf. Source authentication and path validation with orthogonal network capabilities. In *IEEE Conference on Computer Communications (INFOCOM) Workshops*, pages 111–112, 2015.
6. Hao Cai and Tilman Wolf. Source authentication and path validation in networks using orthogonal sequences. In *International Conference on Computer Communication and Networks, ICCCN*, pages 1–10, 2016.
7. André O. Castelucio, Antônio Tadeu A. Gomes, Artur Ziviani, and Ronaldo M. Salles. Intra-domain IP traceback using OSPF. *Computer Communications*, 35(5):554–564, 2012.
8. Dario Catalano, Dario Fiore, and Rosario Gennaro. Certificateless onion routing. In *ACM SIGSAC Conference on Computer and Communications Security, CCS*, pages 151–160, 2009.
9. Chen Chen, Daniele Enrico Asoni, David Barrera, George Danezis, and Adrian Perrig. HORNET: High-speed onion routing at the network layer. In *ACM SIGSAC Conference on Computer and Communications Security, CCS*, pages 1441–1454, 2015.
10. Wei Dai. Crypto++ 5.6.0 benchmarks, 2009. `https://www.cryptopp.com/benchmarks.html`.
11. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
12. Wenxiu Ding, Zheng Yan, and Robert H. Deng. A survey on future internet security architectures. *IEEE Access*, 4:4374–4393, 2016.
13. Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, pages 303–320, 2004.
14. Bin Guo, Zhu Wang, Zhiwen Yu, Yu Wang, Neil Y. Yen, Runhe Huang, and Xingshe Zhou. Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm. *ACM Computing Surveys*, 48(1):7:1–7:31, 2015.
15. Yih-Chun Hu, Adrian Perrig, and Marvin A. Sirbu. SPV: Secure path vector routing for securing BGP. In *ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 179–192, 2004.

16. Yixin Jiang, Haojin Zhu, Minghui Shi, Xuemin (Sherman) Shen, and Chuang Lin. An efficient dynamic-identity based signature scheme for secure network coding. *Computer Networks*, 54(1):28–40, 2010.
17. Bounpadith Kannhavong, Hidehisa Nakayama, Nei Kato, Yoshiaki Nemoto, and Abbas Jamalipour. A collusion attack against OLSR-based mobile ad hoc networks. In *IEEE Global Communications Conference, GLOBECOM*, 2006.
18. Stephen T. Kent, Charles Lynn, and Karen Seo. Secure border gateway protocol (S-BGP). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592, 2000.
19. Tiffany Hyun-Jin Kim, Cristina Basescu, Limin Jia, Soo Bum Lee, Yih-Chun Hu, and Adrian Perrig. Lightweight source authentication and path validation. In *ACM SIGCOMM Conference*, pages 271–282, 2014.
20. David Kirkpatrick. Google: 53% of mobile users abandon sites that take over 3 seconds to load, 2016. `https://www.marketingdive.com/news/google-53-of-mobile-users-abandon-sites-that-take-over-3-seconds-to-load/426070/`.
21. Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.
22. Dave Levin, Youndo Lee, Luke Valenta, Zhihao Li, Victoria Lai, Cristian Lumezanu, Neil Spring, and Bobby Bhattacharjee. Alibi routing. In *ACM SIGCOMM Conference*, pages 611–624, 2015.
23. Steve Lohr. For impatient web users, an eye blink is just too long to wait, 2012. `https://www.nytimes.com/2012/03/01/technology/impatient-web-users-flee-slow-loading-sites.html`.
24. William McGrath, Mozziyar Etemadi, Shuvo Roy, and Bjoern Hartmann. fabryq: Using phones as gateways to prototype internet of things applications using web scripting. In *ACM Symposium on Engineering Interactive Computing Systems, EICS*, pages 164–173, 2015.
25. Microsoft. Microsoft Azure IoT reference architecture (version 2.1), September 2018. `https://docs.microsoft.com/en-us/azure/iot-fundamentals/iot-introduction`.
26. David L. Mills. Network time protocol (version 3) specification, implementation and analysis, 1992. `https://tools.ietf.org/html/rfc1305`.
27. John Moy. Ospf version 2, 1998. `https://tools.ietf.org/html/rfc2328`.
28. Jad Naous, Michael Walfish, Antonio Nicolosi, David Mazières, Michael Miller, and Arun Seehra. Verifying and enforcing network paths with ICING. In *Conference on Emerging Networking Experiments and Technologies, CoNEXT*, pages 30:1–30:12, 2011.
29. NIST. Advanced Encryption Standard (AES), 2001. `https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf`.
30. Venkata N. Padmanabhan and Daniel R. Simon. Secure traceroute to detect faulty or malicious routing. *Computer Communication Review*, 33(1):77–82, 2003.
31. Adrian Perrig, Pawel Szalachowski, Raphael M. Reischuk, and Laurent Chuat. *SCION: A Secure Internet Architecture*. Information Security and Cryptography. Springer, 2017.
32. Jon Postel. Transmission control protocol, 1981. `https://tools.ietf.org/html/rfc793`.
33. Barath Raghavan and Alex C. Snoeren. A system for authenticated policy-compliant routing. In *ACM SIGCOMM Conference*, pages 167–178, 2004.
34. Yakov Rekhter, Tony Li, and Susan Hares. A border gateway protocol 4 (bgp-4), 2006. `https://tools.ietf.org/html/rfc4271`.
35. Stefan Savage, David Wetherall, Anna R. Karlin, and Thomas E. Anderson. Practical network support for IP traceback. In *ACM SIGCOMM Conference*, pages 295–306, 2000.
36. Yilin Shen, Thang N. Dinh, and My T. Thai. Adaptive algorithms for detecting critical links and nodes in dynamic networks. In *IEEE Military Communications Conference, MILCOM*, pages 1–6, 2012.
37. Alex C. Snoeren. Hash-based IP traceback. In *ACM SIGCOMM Conference*, pages 3–14, 2001.
38. Dawn Xiaodong Song and Adrian Perrig. Advanced and authenticated marking schemes for IP traceback. In *IEEE Conference on Computer Communications, INFOCOM*, pages 878–886, 2001.
39. Emil Stefanov and Elaine Shi. FastPRP: Fast pseudo-random permutations for small domains. *IACR Cryptology ePrint Archive*, 2012:254, 2012.
40. Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. In *ACM SIGCOMM Conference*, pages 73–86, 2002.
41. Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous connections and onion routing. In *IEEE Symposium on Security and Privacy, S&P*, pages 44–54, 1997.
42. Praveen Tammana, Rachit Agarwal, and Myungjin Lee. CherryPick: Tracing packet trajectory in software-defined datacenter networks. In *ACM SIGCOMM Symposium on Software Defined Networking Research, SOSR*, pages 23:1–23:7, 2015.
43. Michael Walfish, Jeremy Stribling, Maxwell N. Krohn, Hari Balakrishnan, Robert Tappan Morris, and Scott Shenker. Middleboxes no longer considered harmful. In *Symposium on Operating System Design and Implementation, OSDI*, pages 215–230, 2004.

44. Matthew Wall. How long will you wait for a shopping website to load?, 2016. https://www.bbc.com/news/business-37100091.
45. Edmund L. Wong, Praveen Balasubramanian, Lorenzo Alvisi, Mohamed G. Gouda, and Vitaly Shmatikov. Truth in advertising: Lightweight verification of route integrity. In *ACM Symposium on Principles of Distributed Computing, PODC*, pages 147–156, 2007.
46. Fuyuan Zhang, Limin Jia, Cristina Basescu, Tiffany Hyun-Jin Kim, Yih-Chun Hu, and Adrian Perrig. Mechanized network origin and path authenticity proofs. In *ACM SIGSAC Conference on Computer and Communications Security, CCS*, pages 346–357, 2014.
47. Xin Zhang, Hsu-Chun Hsiao, Geoffrey Hasker, Haowen Chan, Adrian Perrig, and David G. Andersen. SCION: Scalability, control, and isolation on next-generation networks. In *IEEE Symposium on Security and Privacy, S&P*, pages 212–227, 2011.