VALNET: Privacy-Preserving Multi-Path Validation[☆]

Binanda Sengupta

*Institute for Infocomm Research, A*STAR, Singapore*

Abstract

Network path validation (or simply, path validation) provides network end-hosts with the ability to enforce the network paths they want their packets to traverse. Path validation also enables each on-path node to validate whether a packet has followed the same path specified by the respective source node. Traditional networking uses single-path routing. Multi-path routing provides on-path nodes (including the source node) with the flexibility to choose a path, among a set of options available to them, in order to forward a packet to the same destination node. Multi-path validation combines these two notions: a source node, instead of choosing a single path, can choose a set of paths to send its packets to a destination node; and each of the other on-path nodes can verify whether a packet has indeed traversed one of the legitimate paths (the paths designated by the source) and can further select any of these legitimate paths downstream to forward packets to the destination node. The source node typically embeds the set of paths in a packet-header to enable path validation at each on-path node. However, this results in several privacy issues and makes the underlying network prone to certain attacks. In this work, we introduce privacy-preserving multi-path validation with two privacy notions: path privacy and index privacy. We design `VALNET`, a network that achieves privacy-preserving multi-path validation. We analyze the security as well as the performance of `VALNET`. `VALNET` finds numerous use cases (e.g., in secure and private military communications between base stations and on-field devices such as sensor nodes and drones), where data communications need to take place over multiple paths (enforced by the end-hosts) for enhanced reliability, and privacy of any of these paths must not be compromised even if some of the on-path nodes are compromised. To the best of our knowledge, `VALNET` is the first work that addresses privacy concerns in the context of multi-path validation.

Keywords: Multi-path validation, multi-path routing, path privacy, index privacy, chameleon hash functions.

[☆]This is the accepted version of the article published in Computer Networks (Elsevier) (the final published version of the article is available at ScienceDirect: https://doi.org/10.1016/j.comnet.2021.108695).

Email address: `binandas@i2r.a-star.edu.sg` (Binanda Sengupta)

1. Introduction

The current Internet architecture provides best-effort services to end-hosts (where the end-hosts can only hope that their packets will be correctly directed towards the intended destinations) and is prone to several attacks and security breaches [1, 2]. Over the past few years, a large volume of research work on future Internet architectures [3, 4, 5, 6, 7, 8] have come up in order to enhance its security and reliability. Selecting appropriate network paths[1] for communication carries utmost importance. However, the intermediate nodes on a network path in the current Internet is invisible to the end-hosts and solely determined by the Internet control plane. Thus, the end-hosts have no information about the actual path their packets are following, and they have no control over the packets once they leave the source node.

Path-aware networking [9, 3] exposes various properties of available paths to end-hosts and lets them select (and enforce) paths, based on these properties, in order to route their traffic. For example, an end-host may want its packets to go through a service function chain [10] (a chain of network services such as firewall, intrusion detection, and so on), or it may opt for a premium high-speed path for its packets. Future Internet architectures such as SCION [3], which are based on path-aware networking, have drawn significant attention from academia as well as industries. For example, in the past few years, SCION has developed several testbeds, implemented SCION routers, collaborated with leading Internet service providers (ISPs) such as Swisscom [11], and worked with renowned institutions such as the Swiss National Bank [12]. The Internet Research Task Force (IRTF) and Internet Engineering Task Force (IETF) have also formed a separate research group [13, 14] to advance research in this particular domain. Path-aware networking addresses certain security aspects such as source/packet authentication (in order to prevent denial-of-service attacks, malicious packet-injection attacks) [15], network path authorization (where network operators enforce their own policies to rule out certain paths which are not compliant) [16, 17], network path validation (where the end-hosts specify a path for packet transmission and every on-path node can check whether that path has been followed so far) [15, 18, 19].

A *network path validation* (or *path validation*) protocol [20] enables a source node to enforce a network path in order to send packets to a destination node (*path enforcement*), such that each on-path node can verify if the packets have indeed traversed, so far, the same path designated by the source node (*path verification*). Several path validation schemes [21, 15, 22, 23, 24, 16, 25] have been proposed which employ authenticated proofs to enable path verification (where every on-path node produces proofs that each of its downstream nodes, if any, can verify later).

[1] A network path from a source node to a destination node consists of the following on-path nodes: two end-hosts (i.e., the source and destination nodes) and an ordered collection of intermediate nodes between them.

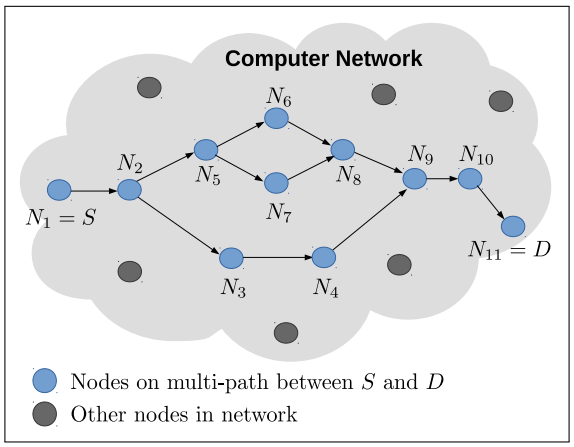

Figure 1: In multi-path routing, a source node S and a destination node D share a multi-path between them. Given a choice, the nodes on the multi-path can forward packets towards D through any of these paths. For example, the node N_2 has the flexibility to forward a packet, sent by S, through any one of the nodes N_3 and N_5.

Path validation schemes typically reveal the whole network path, enforced by the source node, to the (possibly untrusted) intermediate nodes. A recent work [26] introduces two notions of privacy applicable to path validation – path privacy (an intermediate node cannot know the path except its neighbors) and index privacy (an intermediate node cannot know its position or node-index on the path). It also discusses several practical scenarios where both these privacy notions are critical. For example, path validation without path privacy can disclose the identity of the end-hosts (e.g., smartphones, military base stations, sensor nodes, drones) to the intermediate nodes which can relate the end-hosts with certain sensitive information (e.g., medical data, physical location). In another attack scenario, a malicious intermediate node may be present on two different paths destined to two competing service providers (e.g., two shopping websites). In case the malicious node can identify the destination nodes from the respective network paths and it shares a strong business relation with one of them, it can favor its preferred service provider by selectively dropping the packets destined to the other. Identifying critical nodes[2] can also be advantageous for an attacker to maximize the disruption it can cause with a limited amount of resources. Moreover, identifying the node-index of a compromised node may also help an attacker mount some of the aforementioned attacks. For example, if a node knows that its own position, along a path with n (say) nodes, is 2 or $(n-1)$, then it can easily identify the source node or the destination node, respectively. Hence, it is imperative to address privacy issues in network path validation.

[2]A critical node [27] in a network has many neighbors in the network and is thus possibly present on many network paths.

Multi-path routing [28] enables a pair of source/destination nodes to share multiple paths between them; and it also allows other intermediate nodes to forward packets to the destination node through any of these paths (as shown in Figure 1), possibly based on various factors such as availability, communication cost, and so on. In a recent work, Ma et al. [19] combine the notions of path validation and multi-path routing to introduce *multi-path validation* and provide their construction, called Atlas, where a source can select multiple paths for communicating with a destination and each on-path node can verify if one of these paths has been followed by a packet.

In this work, we focus on privacy-preserving multi-path validation which satisfies both the notions of path privacy as well as index privacy (in the context of *multi-path validation*). To achieve multi-path validation (without privacy), Atlas [19] employs a chain of authentication fields where each such field corresponding to an on-path node depends on that of its predecessor node (i.e., the node it receives a packet from). However, a node having multiple predecessors (we call such a node a converging node; see Section 2.1) may have to keep track of many such fields with different values (e.g., the authentication fields computed and forwarded by the nodes N_6 and N_7 in Figure 1 may differ, and the node N_8 has to keep track of them). Consequently, all subsequent nodes (e.g., N_9, N_{10} and N_{11}) need to keep track of multiple authentication fields. Atlas addresses this issue using a novel technique called hierarchical validation, where paths are split into single-segments and multi-segments, and these multi-segments are further split into single-segments and multi-segments recursively. In a higher level of the hierarchy, two nodes still remain neighbors, whereas all the segments in between them are moved to the next lower level of the hierarchy. This enables the latter of those two nodes to have a single authentication field irrespective of different intermediate segments present in the lower level. The source node in Atlas initially sends a list of tagged paths to all on-path nodes. During packet transmission, other on-path nodes can compare the tags carried in packets with the local tags to identify the path they need to verify. In order to achieve this, hierarchical validation requires each packet-header to contain the hierarchy information along with the tags for path-segments. Therefore, it is challenging to design a privacy-preserving multi-path validation protocol using hierarchical validation, while maintaining a single authentication field for each converging node. In this work, instead of using hierarchical validation, we exploit, for the first time to the best of our knowledge, chameleon hash functions [29] in multi-path validation. Chameleon hash functions are cryptographic primitives that allow an entity, having a trapdoor (i.e., a secret key), to find arbitrary collisions for a given hash value (i.e., to modify a message arbitrarily without changing its hash value). In our construction, we use such a chameleon hash function [29] in a novel way so that a single authentication field is sufficient for a converging node, irrespective of the predecessor node it receives a packet from. This also resolves the issue of each subsequent on-path node keeping track of multiple authentication fields.

Our contribution: We summarize our contributions in this work as follows.

- We introduce the notion of privacy-preserving multi-path validation which supports network path validation, in the presence of multi-paths between end-hosts, while maintaining both path privacy and index privacy.
- We construct `VALNET`, a network that achieves privacy-preserving multi-path validation. The destination node in a `VALNET` session can also validate the authenticity of the designated source node. `VALNET` involves a novel use of chameleon hash functions along with other cryptographic techniques.
- We analyze the security of `VALNET` in light of several possible attacks relevant to privacy-preserving multi-path validation.
- We analyze the performance of `VALNET` based on the computational and storage overhead incurred at each on-path node. We also discuss per-packet communication overhead required in `VALNET`.

Organization of the paper: We organize the rest of the paper as follows. Section 2 discusses the preliminaries and background that include the relevant definitions and the related work. In Section 3, we describe the construction of `VALNET`, a network with privacy-preserving multi-path validation. Section 4 analyzes the security of `VALNET`. We discuss the performance of `VALNET` in Section 5 and conclude the paper in Section 6.

2. Preliminaries and Background

In this section, we describe certain notions related to multi-path validation and its privacy. We also discuss the cryptographic primitives that we use later in this work. We then report the related work that exist in the literature and give a brief overview how the current Internet needs to be changed to incorporate multi-path validation.

2.1. Multi-Path Validation and Privacy Properties

Multi-path validation extends the notion of single-path validation[3] where a source node can decide multiple possible paths for its packets to traverse towards a destination node. Given a *source node* S and a *destination node* D, a *multi-path* from S to D is a collection of multiple (say, n, including S and D) nodes, where there are one or more paths from S to D, and each of these paths consists of S and D and also some of the other $(n-2)$ *intermediate nodes*. Thus, the source node can send a packet to the destination node along any of these paths (as opposed to a single path where each packet traverses through a single chain of nodes starting from S and ending at D). We denote such a multi-path from S to D having n nodes by $S \overset{n}{\sim} D$. A multi-path can also be visualized as a directed acyclic graph $\mathcal{G}$ with a single source S and a single sink D.

[3]In order to distinguish from multi-path validation, we use the term *single-path validation* for the path validation schemes designed for a single path between a source and a destination.

The nodes present on a multi-path (i.e., the *on-path nodes*) are ordered in the sense that the source node can choose any arbitrary ordering and assign each of the nodes a unique number $i \in [1, n]$ accordingly.[4] It also labels the identity of the i-th node with a node-identifier N_i. The only restriction on the ordering chosen by the source node is that the relation $1 \leq j < i \leq n$ must hold whenever (N_j, N_i) is an edge in $\mathcal{G}$ (which implies that $N_1 = S$ and $N_n = D$). For example, the ordering shown in Figure 1 is a valid one. Similarly, if we interchange the node-identifiers N_4 and N_5 (or, N_6 and N_7) in Figure 1, then also the respective orderings are valid. Throughout the rest of the paper, we consider a multi-path $S \overset{n}{\rightsquigarrow} D$ with total n nodes, and we refer to the ordering initially chosen (and fixed) by S when we mention the order or indices of on-path nodes. For any edge (N_j, N_i) in $\mathcal{G}$, we say that the node N_j is a *predecessor node* of N_i and N_i is a *successor node* of N_j. Thus, the source (or destination) node has no predecessor (or successor) nodes in $\mathcal{G}$. We say that N_j is a *diverging node* if $\mathcal{G}$ has two or more edges of the form (N_j, N), and we call these edges (i.e., the outgoing edges for a diverging node) *diverging edges* (e.g., the node N_2 in Figure 1 is a diverging node, and (N_2, N_3) and (N_2, N_5) are the corresponding diverging edges). Similarly, N_j is a *converging node* if $\mathcal{G}$ has two or more edges of the form (N, N_j), and we call these edges (i.e., the incoming edges for a converging node) *converging edges* (e.g., the node N_9 in Figure 1 is a converging node, and (N_4, N_9) and (N_8, N_9) are the corresponding converging edges). Thus, a converging node may receive a packet from any of its predecessors (via a converging edge), and a diverging node may forward a packet to any of its successors (via a diverging edge).

A *single-path validation* protocol [20, 21, 15] satisfies both *path enforcement* (where the source node S decides a single path to the destination node D and directs the on-path nodes to follow that path for sending packets to D) and *path verification* (where each on-path node receives authenticated proofs, embedded in packets, from its upstream nodes, and it can verify these proofs in order to check if these packets have followed, so far, the same path enforced by S). A *multi-path validation* protocol [19] is similar to a single-path validation protocol, except that the source node S enforces a multi-path for sending its packets to D and each on-path node can verify whether the packets have so far traveled along one of the paths present in that multi-path. We note that, as a multi-path may consist of only a single path from S to D, the notion of multi-path validation generalizes the notion of single-path validation.

We define three properties – source authentication [15, 26], path privacy [26] and index privacy [26] – in the context of multi-path validation. We note that they are similar to those defined for a single-path validation, except that we consider multi-paths in this work. We say that a multi-path validation protocol satisfies *source authentication*, if S authenticates each packet it sends to D, so that every on-path node traversed by the packet can verify whether the packet has originated from S. We say that a multi-path validation protocol satisfies *path privacy*, if an intermediate node present on the multi-path $S \overset{n}{\rightsquigarrow} D$

[4] We denote the set $\{a, a+1, \ldots, b\}$ also by $[a, b]$, where a and b are two integers with $a \leq b$.

cannot identify the other nodes present on $S \overset{n}{\sim} D$ (except its predecessor and successor nodes since this information is required for packet forwarding). To be more generic, for multi-path validation with path privacy, a set of colluding intermediate nodes cannot identify any honest nodes on $S \overset{n}{\sim} D$ unless they are predecessors or successors to at least one of the colluding nodes. We say that a multi-path validation protocol satisfies *index privacy*, if an intermediate node present on the multi-path $S \overset{n}{\sim} D$ cannot learn its own node-index $i \in [2, n-1]$ (assigned according to the ordering initially chosen by S). Finally, we say that a *privacy-preserving* multi-path validation protocol is the one which satisfies both the notions of path privacy and index privacy.

2.2. Overview of Cryptographic Primitives Used

In this work, we use several cryptographic primitives as follows. In order to achieve message privacy, we use a secure symmetric-key encryption scheme [30, 31] $\mathcal{E} = (\mathsf{KeyGen}_e, \mathsf{Enc}, \mathsf{Dec})$, where the algorithm KeyGen_e generates a symmetric secret key sk, the algorithm Enc encrypts a message (plaintext) using sk to form a ciphertext, and the algorithm Dec decrypts a ciphertext using sk to obtain a message. In order to achieve message authenticity, we use a secure message authentication code (MAC) scheme [32] $\mathsf{MAC} = (\mathsf{KeyGen}_m, \mathsf{MACS}, \mathsf{MACV})$, where the algorithm KeyGen_m generates a symmetric secret key sk, the algorithm MACS takes a message and sk and outputs a message authentication code MAC (usually a short digest), and the algorithm MACV uses sk to check if a given MAC is valid for a given message.[5] In order to shuffle a set of n elements with each element assigned to a unique number in $[1, n]$, we use a secure pseudorandom permutation (PRP) scheme [33]. The scheme takes as input a symmetric secret key sk and the domain $[1, n]$, and it outputs a permutation (or shuffling) of $[1, n]$.

From the security point of view, without knowing the corresponding secret keys, it is hard to produce a valid ciphertext c for a plaintext m (or to produce a valid plaintext m from a ciphertext c) and to generate a MAC for a message m, unless they are available already; and it is also hard to distinguish a PRP from a random permutation (which is a permutation chosen uniformly at random from the family of all permutations over the same domain $[1, n]$).

We use two collision-resistant hash functions H_1 and H_2. For each of these hash functions, it is hard to find two inputs whose hash values are equal. We also use a secure chameleon hash function [29] $\mathsf{CH} = (\mathsf{KeyGen}_c, \mathsf{Eval}, \mathsf{FindColl})$, where the algorithm KeyGen_c generates a secret key-public key pair, the algorithm Eval uses the public key to generate a hash value corresponding to a message, and the algorithm $\mathsf{FindColl}$ uses the secret key to find a collision for a given hash value. We note that, *without* the knowledge of the secret key, CH acts as a conventional collision-resistant hash function. We describe a chameleon hash

[5] Given a message, a MAC and a secret key, the algorithm MACV runs MACS on the message to generate another MAC, and it outputs 1 or 0 depending on whether the MAC generated is equal to the given MAC or not.

function, that we use in our protocol, in Section 3.3. We also use a certificateless one-way anonymous key-agreement protocol [34], such that the source node can authenticate itself to another on-path node without revealing its own identity (the description is given in Section 3.2).

2.3. Related Work

In this section, we discuss some existing single- and multi-path validation schemes. We note that there are several schemes that address either path enforcement [35, 36, 37, 38, 39] or path verification [40, 41, 42] separately. On the contrary, path validation schemes achieve both simultaneously. In order to achieve path enforcement, many schemes embed the path in the packet-header, so that each on-path node can identify the successor node(s) it has to forward the packet to. For path verification, these schemes also embed a verification field, for each of the on-path nodes, in the packet-header. Moreover, most of the path validation schemes also allow source authentication, where an on-path node or the destination node can validate the source. For a detailed description of these schemes and many other related schemes, we refer to a recent and comprehensive survey by Bu et al. [20].

Postmodern forwarding and routing infrastructure (PFRI) [43] includes four additional fields in packet headers – motivation, knobs, accountability and dials – in order to validate the path. While the first two fields are required to enforce the path, the other two fields help to verify the path. ICING [21] achieves path verification by assigning a verification field to each on-path node N. The source node initially computes an authenticator for each N and embeds it in the corresponding verification field. Later on, each of the upstream nodes of N adds a proof to that field. Upon receiving the packet, the node N checks if all the proofs present in its verification field are correct. Then, N computes and adds its proof to the verification field for each of its downstream nodes. The origin and path trace (OPT) protocol [15, 18] employs chained MACs for on-path nodes, and the corresponding verification fields (which are also MACs) are computed using these MACs, such that any incorrect value in that chain makes the verification fields for all subsequent nodes invalid. While ICING imposes $O(n)$ proof generation/verification on each on-path node, OPT reduces this number to $O(1)$. On the other hand, for each on-path node in OPT, the computation of the corresponding proof (stored in the respective verification field) is delegated to the source node. The orthogonal sequence verification protocol (OSV) [22, 44, 45] shares a design similar to that of OPT, except that the verification fields in OSV are computed using orthogonal sequences (instead of MACs as in OPT). OPT is more generalized in that it enforces per-packet validation. Wu et al. [46] propose an efficient path validation mechanism which verifies packets probabilistically.

In avoidance routing such as alibi routing [23], packets are required to avoid specific nodes in a network. To achieve such a guarantee, path validation is enabled, and the packets need to traverse through certain (legitimate) nodes that are located far from the forbidden nodes. If the packets do not follow the legitimate path, path validation fails. On the other hand, if they travel through

both sets of nodes, then the destination node can easily detect this misbehavior by observing the higher latency.

Due to the recent surge in research on path-aware networking [14, 13, 9], several path validation schemes have been proposed in the past few years. He et al. [24] provide the construction of a non-commutative homomorphic asymmetric-key encryption scheme that offers constant-size proofs. They exploit this encryption scheme to design a path validation scheme, called Atomos, that achieves constant-size path validation proofs. Legner et al. [16] propose EPIC, a family of data-plane protocols with several security properties such as path validation and path authorization [17]. Unlike path validation that provides the end-hosts more control over the paths, path authorization enables autonomous systems (ASes) to authorize paths (e.g., in case they want to rule out certain uneconomical paths) such that the end-hosts must choose one of these authorized paths only to communicate their packets. EPIC also reduces the per-packet communication overhead by using short authentication fields for intermediate nodes with a longer authentication field for the destination node. Yang et al. [25] propose a protocol for source and path validation that enforces the intended path policies. They employ a new authentication structure that reduces the packet-processing latency significantly and optimizes the communication cost as well. In another work, Sengupta et al. [26] introduce two privacy notions – path privacy and index privacy – in the context of single-path validation and show that these properties are required to prevent certain attacks. They also construct a privacy-preserving single-path validation protocol, called PrivNPV, which satisfies these privacy notions.

All of the aforementioned schemes consider single-path validation. However, in case of multi-path routing [28], single-path validation schemes are not sufficient. Ma et al. [19] recently introduce a multi-path validation protocol, called Atlas, where the end-hosts can select multiple paths to communicate with each other. Atlas uses several techniques such as OPT-like chained MACs, hierarchical validation (where paths are split into single-segments and multi-segments, and these multi-segments are further split into single-segments and multi-segments recursively) and tagged pruning (where the segments present in paths are tagged for identification during verification and redundant path credentials are later removed in order to save the bandwidth required downstream). For multi-path routing, it may happen that a particular on-path node N is a converging node. As N is present on different paths from S to D, its verification fields corresponding to these paths vary as well (since the chained MAC for N corresponding to one of these path differs from that of others as their upstream nodes differ). So, each packet has to contain multiple verification fields for N corresponding to different paths. Similarly, multiple verification fields for each of the subsequent nodes also need to be stored in each packet. Atlas [19] addresses this problem using hierarchical validation, where multiple path-segments between two nodes N_i and N_j are moved to a lower level in the hierarchy. This makes N_i and N_j neighbors in the higher level of the hierarchy (despite having different path-segments in between). So, the verification field of N_j now depends only on the MAC provided by N_i, irrespective of the inter-

mediate path-segment followed by a packet. We note that, in this work, we use a different and novel technique in order to get rid of multiple verification fields for a converging on-path node (and its subsequent nodes). We employ a trap-door hash function, called chameleon hash (as discussed in Section 2.2), which allows a converging node with a secret key to find an arbitrary collision (i.e., the same value for its verification field) for different MAC values computed by its predecessors. In addition, we use several other techniques to construct the first privacy-preserving multi-path validation protocol as described in Section 3.

The current Internet requires changes to accommodate path validation. Path validation is designed for the future Internet and requires appropriate changes in the current Internet architecture as well as the processing and routing logics for packets. We refer to [24, 20] for a detailed discussion on the same and briefly mention some of the points as follows.

The end-hosts in path validation need to select paths for packet transmission, compute keys shared with other on-path nodes, and embed authenticated proofs in packets required for path verification. ISPs can provide the possible options of permitted paths to the end-hosts (possibly from the set of paths authorized by the respective ISPs [16]), and the end-hosts can choose one or more paths among them for communication. Different ISPs can collaborate to get path information by sharing their internal topology with others. They can provide the end-hosts with path validation as a service. Shared key generation can be done following a key-exchange protocol or using an on-the-fly key derivation from the packets themselves [15]. Lastly, the end-hosts should be able to compute cryptographic primitives required for path verification. Moreover, the routers should be able to compute and verify authenticated proofs embedded in packets. This requires the routers to set up shared keys. Existing routing logic should also be updated in each of the routers — which requires updates to router software [15].

3. `VALNET`: A Network with Privacy-Preserving Multi-Path Validation

We construct `VALNET`, a network that enables privacy-preserving multi-path validation. In this section, we first provide a brief overview of the `VALNET` construction followed by the description of certain cryptographic protocols and primitives that we use in `VALNET`. Finally, we describe in details how packets are communicated between end-hosts in a `VALNET` session.

3.1. Overview of `VALNET`

In order to enable path verification by each node present on a multi-path from a source node S to a destination node D in `VALNET`, S includes in each packet a verification field. When a packet reaches a particular on-path node, the node is able to validate the path using the verification field (which is specific to that node and computed by the source node) and an authentication field (which is initially computed by the source node and thereafter updated by each upstream node) embedded in the packet. S computes the verification fields using the session keys it shares with the corresponding on-path nodes. We provide an

overview of the techniques we use in VALNET as follows. We note that some of these techniques are used in [26] but in the context of single-path validation.

We assume that each node in a network has an identity, and a key generation center (KGC) helps each node to derive a pair of (long-term) secret key-public key. We also assume that there is a public list [34] that maintains these identities and public keys of all network nodes. The secret key-public key pairs are later used for computing the session keys for a particular VALNET session. We exploit a *one-way anonymous key-agreement protocol* [34] in order to enable the source node S and other on-path nodes to establish corresponding shared session keys while maintaining the anonymity of S (see Section 3.2). These session keys are computed in the setup phase of a VALNET session (see Section 3.4). Specifically, for each session, S picks a random secret w (and its corresponding pseudonym $\mathcal{P}$), and computes the session keys using w and the corresponding public keys of the on-path nodes. An on-path node later derives the same session key from $\mathcal{P}$ and its own secret key. We note that the destination node must be able to identify and authenticate the source node. Therefore, they agree upon another (*non-anonymous*) session key for this purpose (see Section 3.2). During the setup phase as well as payload-forwarding phase, the destination node validates a verification field computed using this key — which implicitly enables it to perform *source authentication*. If a malicious node in the network, without the knowledge of this key, tries to impersonate the source node, this authentication fails at the destination node.

A multi-path from S to D can be seen as a collection of single paths, where some of these paths possibly share path-segments among them. In order to achieve *path verification* for an individual path, the source node computes a chain of message authentication codes (MACs) [15, 19], where each MAC in the chain corresponds to an on-path node. Based on this MAC value (that we call the *chained authentication field*), an on-path node computes another MAC value using its own session key and checks if the latter MAC value matches with its corresponding verification field. We note that a single incorrect chained authentication field value causes all subsequent verification fields to be invalid — which makes path verification fail at the next honest on-path node. Instead of storing a chained authentication field for each on-path node, a packet stores a single such field, and each node updates this value in the packet and sends the updated packet to its successor node(s) if the packet passes path verification. For multi-path validation, we will have multiple such chains of MACs. For a converging node N, the respective chained authentication fields sent by its predecessors may vary for different paths due to the difference in its upstream nodes. This requires storing *multiple verification fields* for a single converging node N (based on different chained authentication fields it receives) and, consequently, multiple verification fields for all successors of N as well (based on different chained authentication fields they receive from N) in a packet — which incurs a considerable overhead in terms of packet size. In order to address this issue, we employ a *chameleon hash function* [29], such that different chained authentication fields received by such a converging node N are mapped to the same value (say, temp). Therefore, both the verification field of N and the

chained authentication field updated by N are computed from `temp`; thus their values do not vary depending on which of the predecessor nodes N receives a packet from. We note that the chameleon hash function uses randomness values for a converging node. The source node S encrypts these randomness values for all converging nodes using their respective session keys, such that a converging node can decrypt its corresponding randomness values only. S then embeds all of these ciphertexts in the packet after shuffling them using a *pseudorandom permutation*.

In order to achieve *path enforcement* with *path privacy*, the source node *encrypts* the successor-node information for each of the intermediate nodes (this encryption is different from the encryption used for the randomness values corresponding to the converging nodes), such that only that node can decrypt its corresponding ciphertext using its session key in order to identify its successor node(s) the packet needs to be forwarded to. For a converging on-path node, the corresponding ciphertext also encrypts the permuted indices of the randomness values for the chameleon hash function as discussed above. We note that no intermediate node can decrypt the ciphertexts meant for other on-path nodes — which preserves path privacy.

In order to achieve *index privacy*, neither the sequence of verification fields nor the sequence of ciphertexts embedded in a packet should leak the index of an intermediate node. In order to hide the respective indices of on-path nodes, we employ another *pseudorandom permutation* to shuffle the verification fields as well as the ciphertexts. The source node S uses its own secret key for this shuffling. Therefore, an intermediate node cannot learn its own node-index on the multi-path, even though it can successfully decrypt its ciphertext and identify its verification field.

3.2. Key Agreement in `VALNET`

In `VALNET`, we use a certificateless one-way anonymous key-agreement protocol [34], which enables the source node S to authenticate itself to an intermediate node N without revealing its own identity. However, S knows the identity of the node N. On the other hand, we also use non-anonymous key agreement to let the destination node D identify the source node S and authenticate it. `VALNET` assumes the existence of a key generation center (KGC). We describe the key agreement in `VALNET` as follows.

`VALNET` uses a multiplicative group $\mathbb{G} = \langle g \rangle$ of order q, where q is a prime and g is a generator of $\mathbb{G}$. It also uses two hash functions $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$ and $H' : \{0,1\}^* \rightarrow \{0,1\}^{128}$. Initially, the KGC chooses a random element $x \in \mathbb{Z}_q$ as the master secret key MSK, and it sets the tuple $(q, \mathbb{G}, g, H, H', y = g^x)$ as the public parameter MPK.

For the node associated with an identity ID, the KGC first validates the respective identity. For *one-way anonymous* key agreement, the KGC chooses a random element $a_{ID} \in \mathbb{Z}_q$ and sets $b_{ID} = g^{a_{ID}}$. The KGC computes $d_{ID} = H(ID||b_{ID})x + a_{ID}$ using the master secret key $MSK = x$ and sends the pair (d_{ID}, b_{ID}) to the node. The node then chooses a random element $x_{ID} \in \mathbb{Z}_q$ and sets $y_{ID} = g^{x_{ID}}$. On the other hand, for *non-anonymous* key agreement [47],

the KGC issues a secret key $u_{ID} \in \mathbb{Z}_q$ to the node, and the node computes the corresponding public key $v_{ID} = g^{u_{ID}}$. Finally, the node outputs its secret key $SK_{ID} = (d_{ID}, x_{ID}, u_{ID})$ and its public key $PK_{ID} = (b_{ID}, y_{ID}, v_{ID})$. We note that, among these long-term keys, the anonymous key agreement uses the secret key-public key pair $((d_{ID}, x_{ID}), (b_{ID}, y_{ID}))$, and the non-anonymous key agreement uses the secret key-public key pair (u_{ID}, v_{ID}).

We assume that there is a public list that contains the identities and corresponding public keys of all nodes [34]. The source node searches this list for the credentials of other on-path nodes in order to establish corresponding shared keys. As shown in Eqn. 1,Eqn. 3,Eqn. 4,Eqn. 5 (Section 3.4), the keys established between the respective pairs of nodes during the setup phase of a `VALNET` session are derived from the long-term keys mentioned above.

3.3. Cryptographic Primitives Used in `VALNET`

`VALNET` uses the following cryptographic primitives: a secure (symmetric-key) encryption scheme $\mathcal{E} = (\mathsf{KeyGen}_e, \mathsf{Enc}, \mathsf{Dec})$, a secure message authentication code (MAC) scheme $\mathsf{MAC} = (\mathsf{KeyGen}_m, \mathsf{MACS}, \mathsf{MACV})$, two secure pseudo-random functions (PRPs) Π and Π', and a secure discrete-log-based chameleon hash function $\mathsf{CH} = (\mathsf{KeyGen}_c, \mathsf{Eval}, \mathsf{FindColl})$ [29, 48]. It also uses two collision-resistant hash functions $H_1 : \{0,1\}^* \rightarrow \mathcal{S}$ and $H_2 : \{0,1\}^* \rightarrow \mathcal{M}$, where $\mathcal{S}$ is the space of all session-identifiers and $\mathcal{M}$ is the output space of H_2.

We assume that the chameleon hash function CH is defined on the same group $\mathbb{G} = \langle g \rangle$ used during key agreement (see Section 3.2), where g is a generator of $\mathbb{G}$ and $\mathbb{G}$ has a prime order q. The algorithm $\mathsf{CH.KeyGen}_c$ generates a secret key-public key pair (sk_{CH}, pk_{CH}) such that sk_{CH} is chosen from $\mathbb{Z}_q$ uniformly at random, $pk_{CH} = (q, g, h_{CH})$ and $h_{CH} = g^{sk_{CH}}$. The algorithm $\mathsf{CH.Eval}$ takes the public key pk_{CH}, a message $m_{CH} \in \mathbb{Z}_q$ and a randomness value $r_{CH} \in \mathbb{Z}_q$ as input and outputs $\mathsf{CH.Eval}(pk_{CH}, m_{CH}, r_{CH}) = g^{m_{CH}} h_{CH}^{r_{CH}} \in \mathbb{G}$. So, anyone with a message-randomness value pair (m_{CH}, r_{CH}) can evaluate the hash with the help of the public information g and h_{CH} using two exponentiations. We note that the knowledge of the secret key sk_{CH} helps evaluating the hash function using only one exponentiation: $g^{m_{CH}} h_{CH}^{r_{CH}} = g^{m_{CH}} (g^{sk_{CH}})^{r_{CH}} = g^{m_{CH} + sk_{CH} r_{CH}}$. To simplify, we use $\mathsf{CH.Eval}(m_{CH}, r_{CH})$ in order to denote the hash evaluation (a node evaluates it using the secret key or public key depending on whether it knows the corresponding secret key or not). On the other hand, given the secret key sk_{CH}, the public key pk_{CH}, a message-randomness value pair (m_{CH}, r_{CH}) and an arbitrary message $m'_{CH} \neq m_{CH}$, the collision-finding algorithm $\mathsf{CH.FindColl}$ finds another randomness value $r'_{CH} = sk_{CH}^{-1}(m_{CH} - m'_{CH}) + r_{CH} \pmod q$, such that $\mathsf{CH.Eval}(m_{CH}, r_{CH}) = \mathsf{CH.Eval}(m'_{CH}, r'_{CH})$. However, without the knowledge of sk_{CH}, the secure chameleon hash function CH acts as a conventional collision-resistant hash function.

3.4. Setup Phase in a `VALNET` *Session*

As discussed in Section 2.1, in multi-path routing, a source node S may select a multi-path with n (say) nodes to communicate with a destination node

D. Such a multi-path $S \overset{n}{\sim} D$ can be visualized as a directed acyclic graph $\mathcal{G}$ with total n nodes, where S is the single source and D is the single sink. We denote all nodes present on the multi-path (i.e., all nodes in $\mathcal{G}$) by $N_1, N_2, \cdots, N_n$, where N_i is the node-identifier containing the identity of the i-th on-path node, $N_1 = S$, $N_n = D$, and $N_2, N_3, \ldots, N_{n-1}$ are the intermediate nodes (the ordering of these nodes is chosen by the source node as stated in Section 2.1). We assume that there is a public list that contains the identities as well as the public keys of all nodes in the network (see Section 3.2). In the setup phase of a `VALNET` session, the long-term secret key-public key pairs of the on-path nodes are used to derive respective session keys. The nodes in `VALNET` are assumed to be loosely time synchronized [49].

Let n_c be the number of converging nodes in $\mathcal{G}$. For the i-th node, let $\mathcal{N}_i^+$ denote the set of those nodes N such that (N_i, N) is an edge in $\mathcal{G}$. Similarly, $\mathcal{N}_i^-$ denotes the set of those nodes N such that (N, N_i) is an edge in $\mathcal{G}$. We denote the sizes of these sets (i.e., $|\mathcal{N}_i^+|$ and $|\mathcal{N}_i^-|$) by δ_i^+ and δ_i^-, respectively. Therefore, we have $\delta_i^+ > 1$ only for a diverging node N_i and $\delta_i^- > 1$ only for a converging node N_i. For the source node $N_1 = S$, we have $\delta_1^- = 0$; and for the destination node $N_n = D$, we have $\delta_n^+ = 0$. Let Δ be the total number of converging edges in $\mathcal{G}$ (i.e., δ^- values for all n_c converging nodes sum up to the value Δ). Let $[\mathcal{N}_i^+]$ and $[\mathcal{N}_i^-]$ denote the ordered collection of nodes present in $\mathcal{N}_i^+$ and $\mathcal{N}_i^-$, respectively (e.g., sorted according to the increasing indices of the respective nodes). Each of these collections is sorted according to the respective index-ordering done earlier by the source, where the first element is one with the lowest index. For each converging node N_i, let $[\mathcal{R}_i^-]$ denote the ordered collection of randomness values that correspond to the nodes of $[\mathcal{N}_i^-]$ (for all other nodes in $\mathcal{G}$, this collection is `null`). Each of these randomness values is chosen from $\mathbb{Z}_q$. Let $[\mathcal{N}_i^+]_j$ (or $[\mathcal{N}_i^-]_j$) denote the j-th node present in $[\mathcal{N}_i^+]$ (or $[\mathcal{N}_i^-]$), where $j \in \delta_i^+$ (or $j \in \delta_i^-$). For example, if the set $\mathcal{N}_{12}^-$ for a node N_{12} is $\{N_9, N_5, N_7\}$, then $[\mathcal{N}_{12}^-]$ is the ordered collection $\{N_5, N_7, N_9\}$ and $[\mathcal{N}_{12}^-]_2 = N_7$. We note that each of the collections $\mathcal{N}_1^-$, $[\mathcal{N}_1^-]$, $\mathcal{N}_n^+$ and $[\mathcal{N}_n^+]$ is `null`.

3.4.1. Setup-Packet Processing at Source Node

The source node S chooses a random element w from the set $\mathbb{Z}_q$ and sets a *pseudonym* $\mathcal{P} = g^w$. S then selects another random element $\mathfrak{r}_D \in \{0,1\}^{128}$ corresponding to D and generates a session-identifier $\mathtt{id}_s = H_1(\mathcal{P}||T||\mathfrak{r}_D)$, where T is the current timestamp and $\mathfrak{r}_D$ serves as a randomness value to derive a unique session-identifier for the destination node in the current session. The processing of the setup packet for a `VALNET` session is done at S as follows.

- For each $i \in [2, n]$, the source node S computes a session key that is to be shared with the i-th intermediate node N_i. S searches for the node-identifier N_i in the public list of identities and retrieves the corresponding public key $PK_i = (b_i, y_i, v_i)$. S computes the individual session keys as

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

where $l_{i,1} = (b_i y^{H(N_i||b_i)})^w$ and $l_{i,2} = y_i^w$ for each $i \in [2, n]$. S shares the session key sk_i with the node N_i for the current session (as we will see in Section 3.4.2 and Section 3.4.3, N_i later produces the same session key using the pseudonym $\mathcal{P}$). We note that the end-hosts S and D share two session keys sk and sk_n derived from the non-anonymous and anonymous key agreements, respectively.

- The source node S encrypts its own identity and the total number of on-path nodes as $\mathtt{info}_{\mathcal{E}} \leftarrow \mathsf{Enc}_{sk_1}(N_1||n)$. S forms $\sigma_1 = \mathcal{P}||\mathtt{id}_s||T||\mathtt{t}_D||\mathtt{info}_{\mathcal{E}}$ and generates a short digest $\sigma_2 = H_2(\sigma_1)$ on σ_1. S then applies the PRP Π (using sk) over the set $[1, n]$ in order to get the permuted indices $\Pi_{sk}(1), \Pi_{sk}(2), \ldots, \Pi_{sk}(n)$. As we will see shortly, these permuted indices are later used to shuffle the encrypted successor information and also the verification fields corresponding to the on-path nodes.

- The source node S computes chained authentication fields (CAFs), an array V containing n verification fields, and an array R containing Δ randomness values as follows. S computes the initial CAF value for the node $N_1 = S$ as $CAF_1 = \mathsf{MACS}_{sk_1}(\sigma_2)$. Then, it sets $CAF = CAF_1$ and $V[\Pi_{sk}(1)] = \mathsf{MACS}_{sk_1}(\sigma_2||CAF_1)$.

  S uses R to store Δ (encrypted) randomness values corresponding to the converging nodes in a permuted way. Initially, R is empty, and it gets populated once S processes the converging nodes in order. For each index $i \in [2, n-1]$, the source node S computes the successive CAF values, verification fields and randomness values as follows.

  - If N_i is a converging node, S derives its CAF value from the δ_i^- CAF values corresponding to the nodes in $[\mathcal{N}_i^-]$. Let $N_{j_{i,1}}, N_{j_{i,2}}, \ldots, N_{j_{i,\delta_i^-}}$ be the ordered nodes (in the increasing order of the node-indices) in $[\mathcal{N}_i^-]$ and $CAF_{j_{i,1}}, CAF_{j_{i,2}}, \ldots, CAF_{j_{i,\delta_i^-}}$ be the corresponding CAF values, where $j_{i,1}, j_{i,2}, \ldots, j_{i,\delta_i^-} \in [1, n]$. Then, the corresponding randomness values $r_{j_{i,1}}, r_{j_{i,2}}, \ldots, r_{j_{i,\delta_i^-}}$ in $[\mathcal{R}_i^-]$ are derived as follows. S chooses the first value $r_{j_{i,1}}$ from the set $\mathbb{Z}_q$ uniformly at random. S uses the session key sk_i and the collision-finding algorithm $\mathsf{CH.FindColl}$ of the chameleon hash function CH (see Section 3.3) to obtain other randomness values $r_{j_{i,2}}, r_{j_{i,3}}, \ldots, r_{j_{i,\delta_i^-}}$, such that

    $$\begin{aligned}\mathsf{CH.Eval}(H(CAF_{j_{i,1}}), r_{j_{i,1}}) = \mathsf{CH.Eval}(H(CAF_{j_{i,2}}), r_{j_{i,2}}) = \cdots = \\ \mathsf{CH.Eval}(H(CAF_{j_{i,\delta_i^-}}), r_{j_{i,\delta_i^-}}) = \\ \mathtt{temp}_i \text{ (say)}.\end{aligned}$$

    For $\mathsf{CH.FindColl}$, we consider the integer representation of $sk_i \in \{0,1\}^{128}$ as an element of $\mathbb{Z}_q$. S encrypts the randomness values

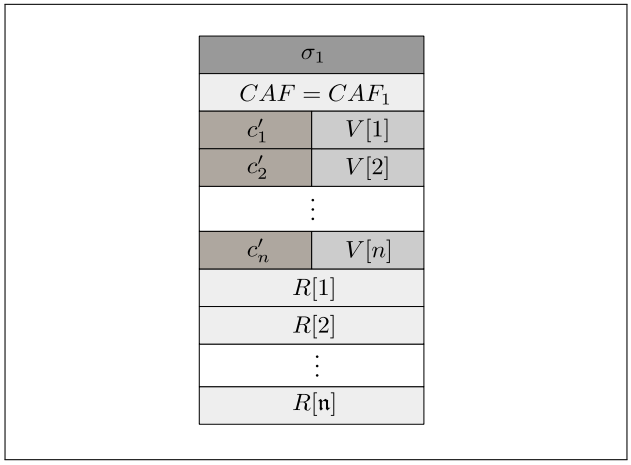

Figure 2: The initial content of the setup-packet P_s transmitted by the source node.

as $\mathsf{Enc}_{sk_i}(r_{j_{i,1}}), \mathsf{Enc}_{sk_i}(r_{j_{i,2}}), \ldots, \mathsf{Enc}_{sk_i}(r_{j_{i,\delta_i^-}})$ and appends these encrypted values respectively to the array R. S computes $CAF_i = \mathsf{MACS}_{sk_i}(\mathtt{temp}_i)$ and sets $V[\Pi_{sk}(i)] = \mathsf{MACS}_{sk_i}(\sigma_2||\mathtt{temp}_i)$.

– If N_i is not a converging node (i.e., N_i has a single predecessor node, say N_j, in $\mathcal{G}$), S sets $\mathtt{temp}_i = CAF_j$. It then computes $CAF_i = \mathsf{MACS}_{sk_i}(\mathtt{temp}_i)$ and sets $V[\Pi_{sk}(i)] = \mathsf{MACS}_{sk_i}(\sigma_2||\mathtt{temp}_i)$.

Finally, for the destination node $N_i = D$ (where $i = n$), the source node S proceeds similarly and derives the corresponding $\mathtt{temp}_n$ value as described above (depending on whether D is a converging node or not). It then sets $V[\Pi_{sk}(n)] = \mathsf{MACS}_{sk}(\sigma_2||\mathtt{temp}_n)$. We note that, unlike other intermediate nodes, the MAC value in the verification field for D is computed using the secret key sk which is derived from the non-anonymous agreement between S and D, such that D can authenticate the real identity of the source node (instead of its pseudonym).

- After processing every node, the array R stores all Δ (encrypted) randomness values corresponding to the converging nodes. S applies the PRP Π' (using sk) to shuffles the values in R. Let $\bar{j}_{i,1}, \bar{j}_{i,2}, \ldots, \bar{j}_{i,\delta_i^-} \in [1, \Delta]$ be the permuted indices of R that store the respective (encrypted) randomness values $\{r_{j_{i,1}}, r_{j_{i,2}}, \ldots, r_{j_{i,\delta_i^-}}\}$ corresponding to the converging node N_i.

- For each node N_i (for $i \in [1, n]$), S encrypts the predecessor and successor information as well as the permuted indices of its verification field and randomness values as

$$c_i \leftarrow \mathsf{Enc}_{sk_i}(\sigma_2||[\mathcal{N}_i^-]||\Pi_{sk}(i)||\bar{j}_{i,1}||\bar{j}_{i,2}||\cdots||\bar{j}_{i,\delta_i^-}||[\mathcal{N}_i^+]), \tag{2}$$

where the string $\bar{j}_{i,1}||\bar{j}_{i,2}||\cdots||\bar{j}_{i,\delta_i^-}$ is `null` if N_i is not a converging node.

- S applies the PRP Π (using sk) to shuffle the sequence of ciphertexts $C_1 = \{c_1, c_2, \ldots, c_n\}$ and forms another sequence $C_2 = \{c'_1, c'_2, \ldots, c'_n\}$, such that $c'_{\Pi_{sk}(i)} = c_i$ is the ciphertext intended for N_i for each $i \in [1, n]$.
- S constructs the setup-packet P_s as follows. It includes in the packet the bit-string σ_1, a chained authentication field CAF (initialized to $CAF = CAF_1$), the sequence of shuffled ciphertexts C_2, the array V containing shuffled verification fields, and the array R containing shuffled randomness values (see Figure 2). For any index $i \in [2, n]$, CAF_i is computed using the algorithm MACS which takes CAF_j (for some predecessor node N_j) and sk_i as input. The successive CAF values form a chain of MACs which guarantees that injecting one invalid CAF in this chain makes all the CAF values computed thereafter (and the corresponding verification fields as well) invalid.
- The source node S forwards the setup-packet P_s to each of its successor nodes in $[\mathcal{N}_1^+]$.

3.4.2. Setup-Packet Processing at Intermediate Node

For each $i \in [2, n-1]$, the intermediate node N_i processes the setup-packet P_s received from its predecessor node. We note that, if N_i is a converging node, it may receive multiple setup-packets from its predecessor nodes that may differ in the CAF values. In that case, the node N_i chooses any one of these packets and proceeds as follows.

- N_i parses σ_1 embedded in the setup-packet as $\mathcal{P}||\mathtt{id}_s||T||\mathfrak{r}_D||\mathtt{info}_{\mathcal{E}}$ and verifies if $\mathtt{id}_s \stackrel{?}{=} H_1(\mathcal{P}||T||\mathfrak{r}_D)$. If the previous equality holds, it computes $\sigma_2 = H_2(\sigma_1)$.
- N_i uses the pseudonym $\mathcal{P}$ and its own secret key $SK_i = (d_i, x_i, u_i)$ to compute the session key

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

  where $l_{i,1} = \mathcal{P}^{d_i}$ and $l_{i,2} = \mathcal{P}^{x_i}$. This session key sk_i is same as that computed using Eqn. 1, since both of them are derived from the same values of $l_{i,1} = g^{wd_i}$ and $l_{i,2} = g^{wx_i}$.
- N_i decrypts the elements of the sequence of (shuffled) ciphertexts $C_2 = \{c'_1, c'_2, \ldots, c'_n\}$ one by one using sk_i and checks if the resulting plaintext begins with $\sigma_2 = H_2(\sigma_1)$. Only one of these ciphertexts (say, c'_j for some $j \in [1, n]$) can be decrypted correctly using sk_i. As S initially encrypted all other ciphertexts in C_2 using keys that are different from sk_i, the decryption of these ciphertexts using sk_i results in random plaintexts which do not begin with σ_2.
- N_i obtains the values $\Pi_{sk}(i), [\mathcal{N}_i^-], [\mathcal{N}_i^+], \bar{j}_{i,1}, \bar{j}_{i,2}, \ldots, \bar{j}_{i,\delta_i^-}$ by decrypting c'_j and checks whether $j \stackrel{?}{=} \Pi_{sk}(i)$. We note that the shuffled ciphertext $c'_{\Pi_{sk}(i)} \in C_2$ is same as $c_i \in C_1$.

- Suppose N_i has received the setup-packet P_s with $CAF = CAF_j$ from the node N_j for some $j \in [1, n]$. N_i first checks whether N_j is present in $[\mathcal{N}_i^-]$ and processes the packet as follows, in case $N_j \in [\mathcal{N}_i^-]$.
  - If N_i is a converging node, then $|[\mathcal{N}_i^-]| > 1$. Suppose N_j is the k-th element of $[\mathcal{N}_i^-]$ for some $k \in [1, \delta_i^-]$. N_i decrypts $R[\bar{j}_{i,k}]$ using sk_i and computes $\mathtt{temp}_i = \mathsf{CH.Eval}(H(CAF_j), \mathsf{Dec}_{sk_i}(R[\bar{j}_{i,k}]))$.
  - If N_i is not a converging node, N_i sets $\mathtt{temp}_i = CAF_j$.

  Then, N_i computes $\mathsf{MACS}_{sk_i}(\sigma_2||\mathtt{temp}_i)$ and checks whether it is equal to $V[\Pi_{sk}(i)]$. If they are equal, then N_i is convinced that each of the previous CAF values in the chain has been computed correctly by the respective node. In that case, N_i updates the CAF value in P_s as $CAF = CAF_i = \mathsf{MACS}_{sk_i}(\mathtt{temp}_i)$.
- If any of the aforementioned verifications fails, N_i drops P_s (if N_i is a converging node and has some other setup-packets, received from its other predecessors, to explore, it then processes one of them in a similar way). If at least one such setup-packet P_s passes all verification, N_i stores the tuple $(\mathtt{id}_s, sk_i, \Pi_{sk}(i), [\mathcal{N}_i^-], [\mathcal{N}_i^+], \bar{j}_{i,1}, \bar{j}_{i,2}, \ldots, \bar{j}_{i,\delta_i^-})$ and sends the updated P_s to each of its successor nodes in $[\mathcal{N}_i^+]$.

3.4.3. Setup-Packet Processing at Destination Node

The destination node D processes the setup-packet P_s received from one of its predecessors as follows.

- D parses σ_1 embedded in the setup-packet as $\mathcal{P}||\mathtt{id}_s||T||\mathtt{r}_D||\mathtt{info}_{\mathcal{E}}$ and verifies if $\mathtt{id}_s \stackrel{?}{=} H_1(\mathcal{P}||T||\mathtt{r}_D)$. If the previous equality holds, it computes $\sigma_2 = H_2(\sigma_1)$.
- D uses the pseudonym $\mathcal{P}$ and its own secret key $SK_n = (d_n, x_n, u_n)$ to compute the session key

$$sk_n \leftarrow H'(l_{n,1}||l_{n,2}||\mathtt{id}_s), \tag{4}$$

  where $l_{n,1} = \mathcal{P}^{d_n}$ and $l_{n,2} = \mathcal{P}^{x_n}$. This session key sk_n is same as that computed using Eqn. 1, since both of them are derived from the same values of $l_{n,1} = g^{wd_n}$ and $l_{n,2} = g^{wx_n}$.
- D decrypts $\mathtt{info}_{\mathcal{E}}$ using the session key $sk_1 = sk_n$ and obtains the total number of nodes n and the identity of the source node $N_1 = S$. D uses the public key $PK_1 = (b_1, y_1, v_1)$ of S and its own secret key $SK_n = (d_n, x_n, u_n)$ to compute the session key

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

  Since $v_n^{u_1} = v_1^{u_n} = g^{u_1 u_n}$, this session key sk is same as that computed in Eqn. 1.

- $D = N_n$ uses sk to compute $j = \Pi_{sk}(n)$. By decrypting c'_j, D obtains the values $\Pi_{sk}(n), [\mathcal{N}_n^-], [\mathcal{N}_n^+], \bar{j}_{n,1}, \bar{j}_{n,2}, \ldots, \bar{j}_{n,\delta_n^-}$ and checks if j is equal to the value $\Pi_{sk}(n)$ embedded in c'_j.
- Suppose D has received the setup-packet P_s with $CAF = CAF_j$ from the node N_j for some $j \in [1, n-1]$. D first checks whether N_j is present in $[\mathcal{N}_n^-]$ and processes the packet as follows, in case $N_j \in [\mathcal{N}_n^-]$.
  - If D is a converging node, then $|[\mathcal{N}_n^-]| > 1$. Suppose N_j is the k-th element of $[\mathcal{N}_n^-]$ for some $k \in [1, \delta_n^-]$. D decrypts $R[\bar{j}_{n,k}]$ using sk_n and computes $\mathtt{temp}_n = \mathsf{CH.Eval}(H(CAF_j), \mathsf{Dec}_{sk_n}(R[\bar{j}_{n,k}]))$.
  - If D is not a converging node, D sets $\mathtt{temp}_n = CAF_j$.

  Then, D computes $\mathsf{MACS}_{sk}(\sigma_2||\mathtt{temp}_n)$ and checks whether it is equal to $V[\Pi_{sk}(n)]$. If they are equal, then D is convinced that each of the previous CAF values in the chain has been computed correctly by the respective on-path node.
- If any of the aforementioned verifications fails, D drops P_s (if D is a converging node and has some other setup-packets, received from its other predecessors, to explore, it then processes one of them in a similar way). If at least one such setup-packet P_s passes all verification, D stores at its end the tuple $(\mathtt{id}_s, sk, sk_n, \Pi_{sk}(n), [\mathcal{N}_n^-], \bar{j}_{n,1}, \bar{j}_{n,2}, \ldots, \bar{j}_{n,\delta_n^-})$, encrypts the string $\mathcal{P}||\mathtt{id}_s||T||\mathfrak{t}_D$ using the session key sk and sends this encrypted string to S as a confirmation for a successful setup phase.

3.5. Payload-Forwarding Phase in a `VALNET` *Session*

The source node S sends payload-packets along some paths present in the multi-path set up in the initial setup phase. Let `payload` denote the payload (actual data) embedded in such a packet that S wants to send to D. We note that, unlike the setup-phase where a diverging node N_i needs to send the processed setup-packet P_s to each of its successor nodes in $[\mathcal{N}_i^+]$ (so that the nodes on each path specified by S can configure the shared session keys and obtain other relevant information from P_s), N_i may send a payload-packet to any node in $[\mathcal{N}_i^+]$ of its choice (often based on several factors like availability, latency and so on) during a packet-forwarding phase.

3.5.1. Payload-Packet Processing at Source Node

In the payload-forwarding phase, the source node S, given $\mathtt{id}_s$, T, $\mathfrak{t}_D$ and `payload`, processes a payload-packet P_p as follows.

- S forms a bit-string $\sigma'_1 = \mathcal{P}||\mathtt{id}_s||T||\mathfrak{t}_D$ and generates a short digest $\sigma'_2 = H_2(\mathtt{payload}||\sigma'_1)$.
- Similar to the setup phase, S computes chained authentication fields (i.e., CAF values), an array V containing n verification fields and an array R containing Δ randomness values as follows. S computes the initial

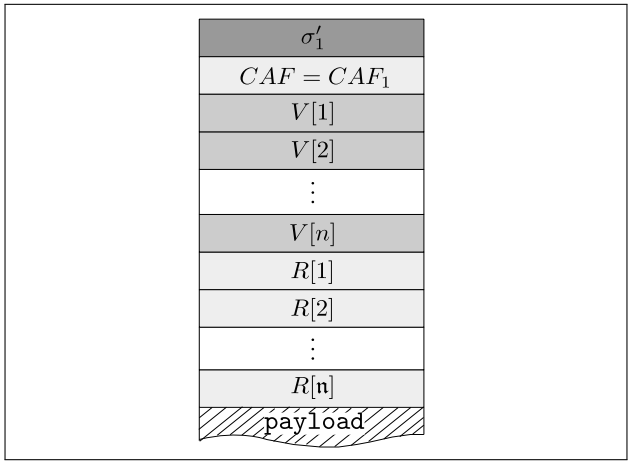

Figure 3: The initial content of a payload-packet P_p transmitted by the source node.

CAF value for the node $S = N_1$ as $CAF_1 = \mathsf{MACS}_{sk_1}(\sigma'_2)$. Then, it sets $CAF = CAF_1$ and $V[\Pi_{sk}(1)] = \mathsf{MACS}_{sk_1}(\sigma'_2||CAF_1)$.

Initially, the array R is empty. S populates R as it processes the converging nodes in order. For each $i \in [2, n-1]$, the source node S computes the corresponding CAF value, verification field and randomness values (if any) as follows.

– If N_i is a converging node, S derives its CAF value from the δ_i^- CAF values corresponding to the nodes in $[\mathcal{N}_i^-]$. Let $N_{j_{i,1}}, N_{j_{i,2}}, \ldots, N_{j_{i,\delta_i^-}}$ be the ordered nodes (in the increasing order of the node-indices) in $[\mathcal{N}_i^-]$ and $CAF_{j_{i,1}}, CAF_{j_{i,2}}, \ldots, CAF_{j_{i,\delta_i^-}}$ be the corresponding CAF values, where $j_{i,1}, j_{i,2}, \ldots, j_{i,\delta_i^-} \in [1, n]$. Then, the corresponding randomness values $r_{j_{i,1}}, r_{j_{i,2}}, \ldots, r_{j_{i,\delta_i^-}}$ in $[\mathcal{R}_i^-]$ are derived as follows. S chooses the first value $r_{j_{i,1}}$ from the set $\mathbb{Z}_q$ uniformly at random. S uses the session key sk_i and the collision-finding algorithm $\mathsf{CH.FindColl}$ of the chameleon hash function CH (see Section 3.3) to obtain other randomness values $r_{j_{i,2}}, r_{j_{i,3}}, \ldots, r_{j_{i,\delta_i^-}}$, such that

$$\mathsf{CH.Eval}(H(CAF_{j_{i,1}}), r_{j_{i,1}}) = \mathsf{CH.Eval}(H(CAF_{j_{i,2}}), r_{j_{i,2}}) = \cdots = \mathsf{CH.Eval}(H(CAF_{j_{i,\delta_i^-}}), r_{j_{i,\delta_i^-}}) = \mathtt{temp}_i \text{ (say).}$$

S encrypts the randomness values as $\mathsf{Enc}_{sk_i}(r_{j_{i,1}}), \mathsf{Enc}_{sk_i}(r_{j_{i,2}}), \ldots, \mathsf{Enc}_{sk_i}(r_{j_{i,\delta_i^-}})$ and appends these encrypted values respectively to the array R. S computes $CAF_i = \mathsf{MACS}_{sk_i}(\mathtt{temp}_i)$ and sets $V[\Pi_{sk}(i)] = \mathsf{MACS}_{sk_i}(\sigma'_2||\mathtt{temp}_i)$.

– If N_i is not a converging node (i.e., N_i has a single predecessor node, say N_j, in $\mathcal{G}$), S sets $\texttt{temp}_i = CAF_j$. It then computes $CAF_i = \mathsf{MACS}_{sk_i}(\texttt{temp}_i)$ and sets $V[\Pi_{sk}(i)] = \mathsf{MACS}_{sk_i}(\sigma_2'||\texttt{temp}_i)$.

Finally, for the destination node $N_i = D$ (where $i = n$), the source node S proceeds similarly and derives the corresponding $\texttt{temp}_n$ value as described above (depending on whether D is a converging node or not). It then sets $V[\Pi_{sk}(n)] = \mathsf{MACS}_{sk}(\sigma_2'||\texttt{temp}_n)$.

- After processing every node, the array R stores all Δ (encrypted) randomness values corresponding to the converging nodes. The source node S shuffles the values in R according to the same permuted indices computed in the setup phase.

- S constructs the payload-packet P_p as follows. It includes in the packet the bit-string σ_1', a chained authentication field CAF (initialized to $CAF = CAF_1$), the array V containing shuffled verification fields, and the array R containing shuffled randomness values (see Figure 3). For any index $i \in [2, n]$, the value CAF_i is computed using the algorithm MACS which takes CAF_j (for some predecessor node N_j) and sk_i as input. The successive CAF values form a chain of MACs which guarantees that injecting one invalid CAF in this chain makes all the CAF values computed thereafter (and the corresponding verification fields as well) invalid.

- The source node S forwards the payload-packet P_p to any of its successor nodes present in $[\mathcal{N}_1^+]$.

3.5.2. Payload-Packet Processing at Intermediate Node

For each $i \in [2, n-1]$, the intermediate node N_i processes the payload-packet P_p, received from its predecessor node, as follows.

- N_i parses σ_1' embedded in the payload-packet as $\mathcal{P}||\texttt{id}_s||T||\mathfrak{r}_D$ and computes $\sigma_2' = H_2(\texttt{payload}||\sigma_1')$.

- Suppose N_i has received the payload-packet P_p with $CAF = CAF_j$ from the node N_j for some $j \in [1, n]$. N_i first checks whether N_j is present in $[\mathcal{N}_i^-]$ and processes the packet as follows, in case $N_j \in [\mathcal{N}_i^-]$.

  – If N_i is a converging node, then $|[\mathcal{N}_i^-]| > 1$. Suppose N_j is the k-th element of $[\mathcal{N}_i^-]$ for some $k \in [1, \delta_i^-]$. N_i decrypts $R[\bar{j}_{i,k}]$ using sk_i and computes $\texttt{temp}_i = \mathsf{CH.Eval}(H(CAF_j), \mathsf{Dec}_{sk_i}(R[\bar{j}_{i,k}]))$.

  – If N_i is not a converging node, N_i sets $\texttt{temp}_i = CAF_j$.

  Then, N_i computes $\mathsf{MACS}_{sk_i}(\sigma_2'||\texttt{temp}_i)$ and checks whether it is equal to $V[\Pi_{sk}(i)]$. If they are equal, then N_i is convinced that each of the previous CAF values in the chain has been computed correctly by the respective node. In that case, N_i updates the CAF value in the payload-packet P_p as $CAF = CAF_i = \mathsf{MACS}_{sk_i}(\texttt{temp}_i)$.

- If any of the aforementioned verifications fails, N_i drops the payload-packet P_p. Otherwise, N_i sends the updated packet to any of its successor nodes in $[\mathcal{N}_i^+]$.

3.5.3. Payload-Packet Processing at Destination Node

The destination node D processes the payload-packet P_p received from its predecessor node as follows.

- D parses σ_1' embedded in the payload-packet as $\mathcal{P}||\mathtt{id}_s||T||\mathfrak{t}_D$ and computes $\sigma_2' \stackrel{?}{=} H_2(\mathtt{payload}||\sigma_1')$.
- Suppose D has received the payload-packet P_p with $CAF = CAF_j$ from the node N_j for some $j \in [1, n-1]$. D first checks whether N_j is present in $[\mathcal{N}_n^-]$ and processes the packet as follows, in case $N_j \in [\mathcal{N}_n^-]$.
  - If D is a converging node, then $|[\mathcal{N}_n^-]| > 1$. Suppose N_j is the k-th element of $[\mathcal{N}_n^-]$ for some $k \in [1, \delta_n^-]$. D decrypts $R[\bar{j}_{n,k}]$ using sk_n and computes $\mathtt{temp}_n = \mathsf{CH.Eval}(H(CAF_j), \mathsf{Dec}_{sk_n}(R[\bar{j}_{n,k}]))$.
  - If D is not a converging node, D sets $\mathtt{temp}_n = CAF_j$.

  Then, D computes $\mathsf{MACS}_{sk}(\sigma_2'||\mathtt{temp}_n)$ and checks whether it is equal to $V[\Pi_{sk}(n)]$. If they are equal, then D is convinced that each of the previous CAF values in the chain has been computed correctly by the respective on-path node.
- If any of the aforementioned verifications fails, D drops P_p.

4. Security Analysis

In this section, we analyze the security of VALNET in light of certain attacks possible in path validation. We stress that both the source node S and the destination node D are assumed to be honest (all the paths, that both S and D know, can be leaked otherwise). The intermediate nodes are the only nodes that may be compromised and may try to learn some of the paths (beyond the knowledge of their predecessor/successor nodes), their respective indices on some of the paths, or the identities of the end-hosts. Moreover, the compromised intermediate nodes may collude with each other [50] to mount some of these attacks. We note that multi-path validation is generic and includes the case of single-path validation as well. Therefore, all the attack vectors pertaining to single-path validation [21, 15, 26] are also relevant for multi-path validation. We also note that path validation does not ensure that a packet is delivered to the destination node (an honest node can drop an invalid packet to avoid wasting downstream resources, whereas a malicious node can drop a packet intentionally). Moreover, the destination node in a path validation protocol is unable to identify the exact on-path node which has dropped or corrupted a packet. In the rest of this section, we discuss how VALNET addresses certain attacks related to path validation.

- In a *path-deviation attack*, a malicious node can send a packet through a path not specified (in $\mathcal{G}$) by the source node. However, the packet is later sent to one of its successors in $\mathcal{G}$, and the packet then traverses the rest of the downstream nodes correctly (path detour). On the other hand, two or more malicious nodes can collude with each other to send the packet through some of the specified nodes on a certain path in $\mathcal{G}$ but not in the order specified in $\mathcal{G}$ (out-of-order traversal). Out-of-order traversals also include the case where the malicious intermediate nodes skip an honest on-path node.

  - As discussed above, in a path-detour attack, a malicious intermediate node (say, N_j) can send a packet through a path $N_j - N'_1 - N'_2 - \cdots - N'_k - N_i$ not specified in $\mathcal{G}$, where N_i is one of the successors of N_j in $\mathcal{G}$ and $N'_1, N'_2, \ldots, N'_k$ are the detour nodes. If N_j and N_i are compromised by an attacker, they can always collude and produce correct CAF values. Apart from forwarding the packet along the legitimate path (to the specified destination D), a malicious intermediate node can also route the packet through arbitrary nodes which may execute arbitrary commands on the packet. These attacks are hard to detect/defend — which is true for the existing path validation schemes [21, 15, 26] also. On the other hand, if an honest node N is bypassed in an out-of-order traversal, then the colluding nodes cannot compute the CAF value corresponding to N without knowing its session key. This makes all subsequent CAF values in the chain incorrect, which is detected by the next honest node on that path while validating its verification field. We note that an out-of-order traversal, involving only some of the colluding malicious nodes, cannot be detected (e.g., if the nodes on a path $N_{i_1} - N_{i_2} - N_{i_3} - N_{i_4}$ specified in $\mathcal{G}$ collude with each other, they can always make a packet traverse through $N_{i_1} - N_{i_3} - N_{i_2} - N_{i_4}$ or $N_{i_1} - N_{i_4}$ in an out-of-order fashion, while generating the correct CAF values with the help of the appropriate session keys).

- In a *denial-of-service (DoS) attack*, the on-path nodes need to perform work that is intensive in terms of storage and/or computation — which makes some of these nodes overloaded (and possibly unavailable) and thus decreases the performance of the network significantly.

  - As described later in Section 5, most of the intermediate nodes in a `VALNET` session need to maintain only a small amount of storage at its end and perform (efficient) symmetric-key operations during the payload-forwarding phase. This reduces the possibility of DoS attacks in `VALNET`. Moreover, as `VALNET` provides a solution for multi-path validation, an honest intermediate node has the flexibility to choose a different path in $\mathcal{G}$ if some of its successor nodes are known to be compromised or unavailable.

We note that payload-packets can be forwarded along all successors of a (possibly malicious) diverging node when forwarding along any one of them is sufficient. As all of these packets are indeed valid packets, each of the subsequent on-path nodes validates and forwards these packets downstream, until the redundant packets reach some converging on-path nodes and are discarded thereafter. In case there are consecutive (and malicious) diverging nodes, this type of attack may saturate the network path itself. The positions of the diverging and converging nodes on the multi-path play a crucial role to prevent such attacks.

- A malicious intermediate node in `VALNET` may attempt to mount a *counterfeit attack* by altering a valid packet in a malicious way or by injecting malicious packets of its choice along the path. The packets injected by the node can be either new (packets created by the malicious node itself) or older (valid but older packets – replay attacks).

  – In `VALNET`, packet alteration is prevented using two short digests σ_2 and σ_2'. Each of these digests is computed using the collision-resistant hash function H_2 which guarantees that tampering with its input produces a completely different digest. Since these digests are in turn used to compute the verification fields (MACs) corresponding to the on-path nodes in the setup and payload-forwarding phases of a `VALNET` session, an incorrect digest would produce a different MAC value that would not match with the verification field stored in the packet — which enables an honest node to easily detect this anomaly and drop the packet. Similarly, tampering with the CAF value in a packet makes all the subsequent verification fields invalid, as these fields are computed using MACs which take both the digest and the (respective) previous CAF values as input. We note that, if a predecessor node of a converging node N_i computes an incorrect CAF value and passes it to N_i, then $\texttt{temp}_i$ will also be incorrect (as the chameleon hash function, when later applied on the corresponding randomness value stored in R and the incorrect CAF value, will produce another value other than $\texttt{temp}_i$). So, the value $\mathsf{MACS}_{sk_i}(\sigma_2'||\texttt{temp}_i)$ will not match the value $V[\Pi_{sk}(i)]$. On the other hand, if the converging node N_i computes an incorrect CAF value, then also the packet verification at the next honest node will fail as discussed above.

  While injecting a new packet in `VALNET`, the only possibility for a malicious intermediate node to successfully impersonate the source S is to guess/compute all session keys or the secret key of S — either of which is hard to accomplish. Moreover, every packet includes a unique session-identifier and the current timestamp which are used to compute the digest σ_2 or σ_2' (these digests are in turn used to compute the verification fields). As this information can be validated by each

node on the multi-path, VALNET prevents replay attacks as well.

- An intermediate node can attempt to perform a *coward attack* (e.g., any of the attacks mentioned above) when the chances of detection of its malicious activities are known to be low. Path validation involves certain computational overhead for the on-path nodes. So, path validation may be enabled the source node under certain circumstances only (e.g., in case there is a packet loss, or malicious activities by certain nodes are detected, or the source node wants to validate the setup packet in a VALNET session) [20]. If a malicious node knows that path validation is disabled for the current packet, it may mount some (or all) of the attacks mentioned above without having the corruption detected.

  – One obvious solution is to enforce path validation for every packet, such that a malicious node cannot mount such coward attacks. If this strict guarantee is not needed for every packet, the source node can enable probabilistic path validation (i.e., random packets transmitted by the source node S are validated) [15, 20]. For example, the on-path nodes only update the CAF value in a packet (without checking their corresponding verification fields), and the destination node probabilistically verifies its verification field to validate the path followed. As it makes a malicious intermediate node unable to predict which packets would be checked by the destination node for path validation, the intermediate node would always execute the protocol correctly in order to avoid possible detection.

- In a *path-revealing attack*, a malicious intermediate node present in $\mathcal{G}$ attempts to learn information about the multi-path $S \overset{n}{\sim} D$. Each on-path node has certain partial information (e.g., the identities of its neighbors) about the some of the paths from S to D. So, in a path-revealing attack, an intermediate node may try to gain knowledge of these (full or partial) paths beyond the information which it requires for packet forwarding.

  – As the path information in a VALNET session is encrypted using the corresponding secret session keys of the on-path nodes, each intermediate node can decrypt only one ciphertext meant for it, and it can know the identities of its neighbors only. On the other hand, as the encryption scheme $\mathcal{E}$ used in VALNET is assumed to be secure and a malicious intermediate node does not have the session keys of the other honest nodes, it cannot decrypt any of the ciphertexts corresponding to these honest nodes.

- In an *index-revealing attack*, a malicious intermediate node tries to learn its own node-index on the multi-path $S \overset{n}{\sim} D$ (according to the ordering of nodes initially assigned by the source node, as discussed in Section 2.1 and Section 3.4). It can further exploit this information to identify other

on-path nodes (e.g., a node with index 2 or $n-1$ can easily identify the source node or the destination node, respectively).

  - An intermediate node in a `VALNET` session does not know the session key sk that is shared by S and D and used to compute the PRP Π. The PRP shuffles both the verification fields and the ciphertexts (containing successor information) for all the on-path nodes. Assuming that Π is a secure PRP, a malicious intermediate node cannot exploit these shuffled fields to learn its own node-index on $S \overset{n}{\sim} D$.

- A set of malicious intermediate nodes may collude with each other to mount a *key-extraction attack*, where they try to extract the session key of an honest node. The chameleon hash function CH has the following property: given two message-randomness value pairs (m_{CH}, r_{CH}) and (m'_{CH}, r'_{CH}) such that $\mathsf{CH.Eval}(m_{CH}, r_{CH}) = \mathsf{CH.Eval}(m'_{CH}, r'_{CH})$, it is easy to extract the session key $sk_{CH} = (m_{CH} - m'_{CH})(r'_{CH} - r_{CH})^{-1} \pmod q$. Suppose the node N_i (for some $i \in [2, n]$) is a converging node with two predecessors (say, $N_{j_{i,1}}$ and $N_{j_{i,2}}$) in $\mathcal{G}$, where $j_{i,1}, j_{i,2} \in [1, n-1]$. Let $CAF_{j_{i,1}}$ and $CAF_{j_{i,2}}$ be two CAF values received from the nodes $N_{j_{i,1}}$ and $N_{j_{i,2}}$, and $r_{j_{i,1}}$ and $r_{j_{i,2}}$ be the corresponding randomness values such that $\mathsf{CH.Eval}(H(CAF_{j_{i,1}}), r_{j_{i,1}}) = \mathsf{CH.Eval}(H(CAF_{j_{i,2}}), r_{j_{i,2}})$. As $N_{j_{i,1}}$ and $N_{j_{i,2}}$ know the values $CAF_{j_{i,1}}$ and $CAF_{j_{i,2}}$ values, they may collude with each other and attempt to extract the session key sk_i of the node N_i.

  - In `VALNET`, shared session keys are established using secure (anonymous and non-anonymous) key-agreement protocols discussed in Section 3.2. So, these keys can be computed only by the individual nodes involved in the respective key agreement. On the other hand, in order to extract the session key of a converging node N_i, its predecessors $N_{j_{i,1}}$ and $N_{j_{i,2}}$ must know the corresponding randomness values $r_{j_{i,1}}$ and $r_{j_{i,2}}$. However, these randomness values in `VALNET` are also encrypted using sk_i, such that only the node N_i can identify and decrypt them. In other words, in order to extract the session key sk_i, the nodes $N_{j_{i,1}}$ and $N_{j_{i,2}}$ would need the key itself.

5. Performance Analysis of `VALNET`

In order to analyze the performance of `VALNET`, we rely on the following assumptions and instantiations of the cryptographic primitives.

- The symmetric-key encryption scheme $\mathcal{E}$, the message authentication code (MAC) scheme MAC and the pseudorandom permutations (PRPs) Π and Π' can be instantiated with the help of AES-128 block cipher [31]: AES-CBC, CMAC-AES and FastPRP [33], respectively (each of them uses AES-128 as the underlying block cipher).

- We use a group $\mathbb{G}$ of order q for the one-way anonymous key-agreement protocol as well as the chameleon hash function CH. We take q to be a 160-bit large prime and $\mathbb{G}$ to be an elliptic curve group over the finite field $\mathbb{F}_q$. We note that, in the description of `VALNET`, we used multiplicative notion for the representation of the group $\mathbb{G}$ (operations allowed over the group elements are multiplications and exponentiations). For an efficient instantiation, $\mathbb{G}$ is taken to be an (additive) elliptic curve group defined over $\mathbb{F}_q$, where the corresponding operations over the group elements are additions and scalar multiplications.

- For the hash function H, we take its output space to be $\{0,1\}^{160}$. For each of the other hash functions H', H_1 and H_2, we take the output space to be $\{0,1\}^{128}$. We compute hashes using SHA-256 and then truncate the outputs up to the required number of bits.

- We represent a destination-specific random element $\mathtt{r}_D$, a timestamp and the identity of a node (e.g., N_i) using a 128-bit string, a 32-bit string and a 128-bit string, respectively.

Based on these instantiations, we make the following observations.

- For an on-path node N_i, all the individual components of the long-term public key $PK_i = (b_i, y_i, v_i)$ and that of the long-term secret key $SK_i = (d_i, x_i, u_i)$ are 20-byte long. The pseudonym $\mathcal{P}$ for a `VALNET` session is also 20-byte long.

- Each of the following elements is 16-byte long: the session-identifier $\mathtt{id}_s$, the random element $\mathtt{r}_D$, the session keys $sk, sk_2, sk_3, \ldots, sk_{n-1}$ and sk_n, the short digests σ_2 and σ_2', the CAF value CAF, the verification fields $V[1], V[2], \ldots, V[n]$, and the node-identifiers $N_1, N_2, \ldots, N_n$.

- Each of the randomness values $R[1], R[2], \ldots, R[\Delta]$ requires 32 bytes of storage (where each randomness value is actually 20-byte long, and it is then padded and encrypted using 128-bit AES-CBC).

- The timestamp T included in a packet typically takes 4 bytes.

- As the PRPs Π and Π' operate over the domains $[1, n]$ and $[1, \Delta]$ respectively, we have the following: $\Pi_{sk}(1), \Pi_{sk}(2), \ldots, \Pi_{sk}(n) \in \{0,1\}^{\lceil \log_2 n \rceil}$ and $\Pi'_{sk}(1), \Pi'_{sk}(2), \ldots, \Pi'_{sk}(\Delta) \in \{0,1\}^{\lceil \log_2 \Delta \rceil}$.

5.1. Computational Cost

The computational costs incurred at different nodes in `VALNET` are different. Table 1 shows different cryptographic operations that `VALNET` nodes (i.e., the source node S, the destination node D and an intermediate node N_i for $i \in [2, n-1]$) need to perform during the setup and payload-forwarding phases. We estimate the time required by these nodes for processing a packet as follows.

For an elliptic-curve group $\mathbb{G}$ of order q (for a 160-bit prime q), each scalar multiplication in $\mathbb{G}$ takes around 0.75 milliseconds when measured on a 1.83 GHz

Table 1: Number of cryptographic operations performed by different VALNET nodes

Cryptographic Operations	Setup phase			Payload-forwarding phase		
	S	D	N_i	S	D	N_i
Scalar multiplication for key-setup	$3n-1$	3	2	0	0	0
Scalar multiplication for CH	n_c	$1^\dagger$	$1^\dagger$	n_c	$1^\dagger$	$1^\dagger$
Hash evaluation (excluding CH)	$2n+1$	4	3	1	1	1
Hash evaluation for CH	Δ	$1^\dagger$	$1^\dagger$	Δ	$1^\dagger$	$1^\dagger$
PRP (Π or Π')	2	1	0	0	0	0
Encryption/decryption (excluding R)	$n+1$	2	$n^\ddagger$	0	0	0
Encryption/decryption for R	Δ	$1^\dagger$	$1^\dagger$	Δ	$1^\dagger$	$1^\dagger$
MAC evaluation	$2n-1$	1	2	$2n-1$	1	2

Parameters of the graph $\mathcal{G}$: n is the total number of nodes; n_c is the total number of converging nodes; Δ is the total number of converging edges.
† These operations are required if and only if the corresponding node is a converging node.
‡ On an average, an intermediate node in VALNET attempts to decrypt $\frac{n}{2}$ ciphertexts before it successfully decrypts its corresponding ciphertext.

Intel Core 2 Duo processor [51]. The instantiations of the PRPs Π and Π' (for C_2 and R) using FastPRP [33] need around $n\lceil\log_2 n\rceil$ and $\Delta\lceil\log_2 \Delta\rceil$ pseudorandom bits, respectively. These bits are generated by encrypting non-negative integers using the session key sk and AES-128. For example, given $n = 16$ and $\Delta = 8$, Π and Π' require around 64 bits and 24 bits, respectively, and a single invocation of AES-128 would suffice to generate these bits for each of them (e.g., AES-$128_{sk}(0)$ for Π and AES-$128_{sk}(1)$ for Π'). We use a cryptographic benchmark [52] in order to estimate the time required by different cryptographic primitives involved in VALNET, where the results are reported for a 1.83 GHz Intel Core 2 Duo processor. According to this benchmark, each hash evaluation using SHA-256 takes 0.48 microseconds (assuming at most 56-byte inputs of the form $l_{i,1}||l_{i,2}||\mathtt{id}_s$ while computing session keys) and each MAC evaluation using CMAC-AES takes 0.32 microseconds (assuming at most 36-byte inputs of the form $\sigma_2||\mathtt{temp}_i$ or $\sigma_2'||\mathtt{temp}_i$ while computing verification fields). On the other hand, the running time of an encryption/decryption using 128-bit AES-CBC depends on the size of its input of the form $\sigma_2||[\mathcal{N}_i^-]||\Pi_{sk}(i)||\bar{j}_{i,1}||\bar{j}_{i,2}||\cdots||\bar{j}_{i,\delta_i^-}||[\mathcal{N}_i^+]$ (for C_2) or $r \in \mathbb{Z}_q$ (for R). For each randomness value stored in R, the input size is fixed (i.e., 20 bytes); thus, each encryption/decryption using 128-bit AES-CBC takes 0.17 microseconds. For the values stored in C_2, we assume that the intermediate node N_i has both the maximum in-degree and the maximum out-degree among the nodes in $\mathcal{G}$ (in order to estimate the maximum size of a plaintext input fed to an encryption). The time required for the encryption/decryption of a value stored in C_2 depends on various parameters, such as n, Δ, $|\mathcal{N}_i^-|$ and $|\mathcal{N}_i^+|$. For example, such an encryption/decryption takes: 0.71 microseconds (for $n = 8$, $\Delta = 2$, $|\mathcal{N}_i^-| = |\mathcal{N}_i^+| = 2$), 1.28 microseconds (for $n = 16$, $\Delta = 4$,

Table 2: Time (in milliseconds) required by different VALNET nodes for performing the cryptographic operations stated in Table 1 (estimated for different combinations of parameters)

Combinations of Parameters	Setup phase			Payload-forwarding phase		
	S	D	N_i	S	D	N_i
$n = 8$, $\Delta = 4$, $n_c = 2$, $\lvert\mathcal{N}_i^-\rvert = \lvert\mathcal{N}_i^+\rvert = 2$	18.78	3.00	2.26	1.51	0.75	0.75
$n = 16$, $\Delta = 8$, $n_c = 3$, $\lvert\mathcal{N}_i^-\rvert = \lvert\mathcal{N}_i^+\rvert = 4$	37.55	3.00	2.26	2.27	0.75	0.75
$n = 25$, $\Delta = 8$, $n_c = 2$, $\lvert\mathcal{N}_i^-\rvert = \lvert\mathcal{N}_i^+\rvert = 5$	57.09	3.00	2.27	1.52	0.75	0.75
$n = 32$, $\Delta = 12$, $n_c = 5$, $\lvert\mathcal{N}_i^-\rvert = \lvert\mathcal{N}_i^+\rvert = 3$	75.09	3.00	2.27	3.78	0.75	0.75
$n = 40$, $\Delta = 16$, $n_c = 8$, $\lvert\mathcal{N}_i^-\rvert = \lvert\mathcal{N}_i^+\rvert = 2$	95.35	3.00	2.27	6.04	0.75	0.75

– We assume that N_i (for some $i \in [2, n-1]$) is a converging node, and it has both the maximum in-degree and the maximum out-degree among all the nodes present in $\mathcal{G}$, so that time required by N_i gives an estimate on the (maximum) time that would be required by any intermediate node in $\mathcal{G}$. For the same reason, we assume that D is converging node and it has the maximum in-degree among all nodes.
– We note that scalar multiplications are the most expensive operations in VALNET. Without these operations, each of the figures in the table would have been a few microseconds only.
– In the payload-forwarding phase, the source node S performs one scalar multiplication for each converging node. Thus, the time required by S in this phase varies mostly with the number n_c of converging nodes present in $\mathcal{G}$. On the other hand, a converging (intermediate or destination) node needs to perform one scalar multiplication in this phase.
– The running time of a non-converging intermediate (or destination) node is a few microseconds, as it does not involve any scalar multiplications.

$|\mathcal{N}_i^-| = |\mathcal{N}_i^+| = 4$), 1.57 microseconds (for $n = 25$, $\Delta = 8$, $|\mathcal{N}_i^-| = |\mathcal{N}_i^+| = 5$), 1.01 microseconds (for $n = 32$, $\Delta = 12$, $|\mathcal{N}_i^-| = |\mathcal{N}_i^+| = 3$), 0.72 microseconds (for $n = 40$, $\Delta = 16$, $|\mathcal{N}_i^-| = |\mathcal{N}_i^+| = 2$). We note that, for the same encryption scheme (e.g., 128-bit AES-CBC), the size of a plaintext/ciphertext (and, thus, also the time required) for an encryption/decryption mainly depends on the maximum in-degree and maximum out-degree of a node, as the other inputs of an encryption contribute much less (compared to the node-identifiers of the predecessor and successor nodes) to the size of the plaintext.

Based on these estimates on the running time of the cryptographic primitives used in VALNET, we estimate the time required by each on-path node according to the figures shown in Table 1. Table 2 shows the total time required by different VALNET nodes (i.e., source node S, destination node D and intermediate node N_i) based on different combinations of the parameters n, Δ, n_c, $|\mathcal{N}_i^-|$, $|\mathcal{N}_i^+|$.

5.2. Communication Overhead per Packet

The size of each of the ciphertexts $c_1, c_2, \ldots, c_n$ depends on the total number n of nodes in $\mathcal{G}$, the total number (Δ) of converging edges in $\mathcal{G}$ and also the maximum in-degree of any on-path node N_i (for some $i \in [1, n]$) and the maximum out-degree of any on-path node N_j (for some $j \in [1, n]$). For example, given $n = 16$, $\Delta = 8$, $|\mathcal{N}_i^-| = 4$ (where N_i has the maximum in-degree among the nodes in $\mathcal{G}$) and $|\mathcal{N}_j^+| = 4$ (where N_j has the maximum out-degree among the nodes in $\mathcal{G}$), the size of a ciphertext is around 146 bytes. The communication

overhead per on-path node for the setup-packet P_s is due to a verification field and a ciphertext (which is around 162 bytes) and that for a payload-packet P_p is due to a verification field (which is around 16 bytes). Apart from this, each packet contains a short digest σ_2 (or σ_2'), a CAF value CAF and an array R containing randomness values — which incurs around 192 bytes communication overhead in total.

5.3. Storage per Node

In the setup phase of a `VALNET` session, each on-path node computes/obtains and stores certain elements that help them process payload-packets later. The source node S stores a pseudonym $\mathcal{P}$, a session-identifier $\mathtt{id}_s$, a random element $\mathfrak{r}_D$, n session keys (i.e., $sk, sk_2, sk_3, \ldots, sk_n$), n permuted indices for verification fields (i.e., $\Pi_{sk}(1), \Pi_{sk}(2), \ldots, \Pi_{sk}(n)$) and Δ permuted indices for randomness values in R (i.e., $\Pi'_{sk}(1)$, $\Pi'_{sk}(2), \ldots, \Pi'_{sk}(\Delta)$, where δ_i^- permuted indices are assigned to each converging node N_i). So, S stores total $416 + n(128 + \lceil \log_2 n \rceil) + \Delta \lceil \log_2 \Delta \rceil$ bits. Given $n = 16$ and $\Delta = 8$, this storage accounts for around 319 bytes. On the other hand, the destination node D stores the elements $\mathtt{id}_s, sk, sk_n, \Pi_{sk}(n), [\mathcal{N}_n^-], [\mathcal{N}_n^+], \bar{j}_{n,1}, \bar{j}_{n,2}, \ldots, \bar{j}_{n,\delta_n^-}$. Given $n = 16$, $\Delta = 8$ and $|\mathcal{N}_n^-| = 4$, this storage is around 114 bytes. The storage required at each intermediate node N_i (for some $i \in [2, n-1]$) is due to $\mathtt{id}_s, sk_i, \Pi_{sk}(i), [\mathcal{N}_i^-], [\mathcal{N}_i^+], \bar{j}_{i,1}, \bar{j}_{i,2}, \ldots, \bar{j}_{i,\delta_i^-}$. Given $n = 16$, $\Delta = 8$, $|\mathcal{N}_i^-| = 4$ and $|\mathcal{N}_i^+| = 4$, the node N_i stores around 162 bytes. We note that the storage overhead is much less for an intermediate node (or the destination node) in case it is neither a converging nor a diverging node.

6. Conclusion

Path-aware networking has recently attracted attention from various research communities. It exposes to the end-hosts various properties of available network paths, and it also lets them select certain paths to route their packets through the network. Path validation enables the end-hosts to enforce a path, thus selected, as well as allows on-path nodes to validate that path. Multi-path routing enhances reliability of a network by enabling packet-forwarding through alternative paths in case certain paths become unavailable or compromised. Multi-path validation combines these two notions and offers guarantees of path validation while enabling the end-hosts to specify a multi-path instead of a single path for packet transmission.

In this work, we have introduced the notion of privacy-preserving multi-path validation that satisfies the following properties: 1) a source node can choose one or more paths (i.e., a multi-path) to send its packets to a destination node; 2) an intermediate node present on the multi-path can select any of these specified paths in order to forward packets to the destination node; 3) any on-path node can validate whether the packets have traversed one of the paths included in the designated multi-path; 4) no intermediate node can learn the multi-path

(except the part consisting of its immediate predecessor and successor nodes) or learn its own node-index along the multi-path.

Following the aforementioned properties, we have designed VALNET, a network with privacy-preserving multi-path validation. Apart from using several cryptographic techniques to attain path validation and privacy guarantees, we have applied chameleon hash functions, in a novel way, in the context of multi-path validation to eliminate the need for multiple verification fields for each converging node (and each of its subsequent nodes). To the best of our knowledge, ours is the first work that addresses privacy-preserving multi-path validation. We have also discussed the security of VALNET by considering various attacks that malicious intermediate nodes can attempt to mount, either individually or by colluding with each other, on VALNET. Finally, we have analyzed the performance of VALNET based on the computational, storage and communication overheads involved.

VALNET satisfies two seemingly contradictory notions of path privacy and multi-path validation — which enables it to enjoy the benefits of the latter one while still protecting the privacy of the network nodes present on a multi-path. We note that VALNET has a trade-off between privacy and efficiency. On the one hand, the use of cryptographic primitives are required to achieve path/index privacy guarantees in VALNET; these primitives, on the other hand, induce an additional computational cost for processing packets at different on-path nodes — which does not appear to meet the line rate of the current Internet. Further research in privacy-preserving multi-path validation is required to close this gap. Some of the research directions in this regard may be to come up with different (and more efficient) instantiations of the same cryptographic primitives used in this work, or to exploit another set of primitives that leads to a more efficient construction.

Acknowledgments

This research is supported by the Agency for Science, Technology and Research (A*STAR) under its IAF-PP (A20F8a0044). The author thanks Anantharaman Lakshminarayanan, Sriram Ramachandran and the anonymous reviewers for their insightful comments and suggestions regarding this work.

References

[1] S. Kottler, February 28th DDoS incident report, https://github.blog/2018-03-01-ddos-incident-report/ (Accessed: November 2021).

[2] N. Woolf, DDoS attack that disrupted Internet was largest of its kind in history, experts say, https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet (Accessed: November 2021).

[3] X. Zhang, H. Hsiao, G. Hasker, H. Chan, A. Perrig, D. G. Andersen, SCION: Scalability, control, and isolation on next-generation networks, in: IEEE Symposium on Security and Privacy, S&P, 2011, pp. 212–227.

[4] D. Barrera, L. Chuat, A. Perrig, R. M. Reischuk, P. Szalachowski, The SCION Internet architecture, Communications of the ACM 60 (6) (2017) 56–65.

[5] A. Perrig, P. Szalachowski, R. M. Reischuk, L. Chuat, SCION: A Secure Internet Architecture, Information Security and Cryptography, Springer, 2017.

[6] T. Anderson, K. Birman, R. M. Broberg, M. Caesar, D. Comer, C. Cotton, M. J. Freedman, A. Haeberlen, Z. G. Ives, A. Krishnamurthy, W. Lehr, B. T. Loo, D. Mazières, A. Nicolosi, J. M. Smith, I. Stoica, R. van Renesse, M. Walfish, H. Weatherspoon, C. S. Yoo, The NEBULA future Internet architecture, in: The Future Internet - Future Internet Assembly 2013: Validated Results and New Horizons, 2013, pp. 16–26.

[7] T. Anderson, K. Birman, R. M. Broberg, M. Caesar, D. Comer, C. Cotton, M. J. Freedman, A. Haeberlen, Z. G. Ives, A. Krishnamurthy, W. Lehr, B. T. Loo, D. Mazières, A. Nicolosi, J. M. Smith, I. Stoica, R. van Renesse, M. Walfish, H. Weatherspoon, C. S. Yoo, A brief overview of the NEBULA future Internet architecture, Computer Communication Review 44 (3) (2014) 81–86.

[8] W. Ding, Z. Yan, R. H. Deng, A survey on future Internet security architectures, IEEE Access 4 (2016) 4374–4393.

[9] B. Trammell, J. Smith, A. Perrig, Adding path awareness to the Internet architecture, IEEE Internet Computing 22 (2) (2018) 96–102.

[10] IETF, Service function chaining (SFC) architecture, `https://datatracker.ietf.org/doc/html/rfc7665` (Accessed: November 2021).

[11] Swisscom, The secure, high-speed Internet under your control, `https://www.swisscom.ch/en/business/enterprise/offer/wireline/scion.html` (Accessed: November 2021).

[12] Swiss National Bank, SNB and SIX launch the communication network Secure Swiss Finance Network, `https://www.snb.ch/en/mmr/reference/pre_20210715/source/pre_20210715.en.pdf` (Accessed: November 2021).

[13] IRTF, Path aware networking research group PANRG, `https://irtf.org/panrg` (Accessed: November 2021).

[14] IETF, Path aware networking rg (panrg), `https://datatracker.ietf.org/rg/panrg/about/` (Accessed: November 2021).

[15] T. H. Kim, C. Basescu, L. Jia, S. B. Lee, Y. Hu, A. Perrig, Lightweight source authentication and path validation, in: ACM SIGCOMM Conference, 2014, pp. 271–282.

[16] M. Legner, T. Klenze, M. Wyss, C. Sprenger, A. Perrig, EPIC: Every packet is checked in the data plane of a path-aware Internet, in: USENIX Security Symposium, 2020, pp. 541–558.

[17] T. Klenze, C. Sprenger, D. A. Basin, Formal verification of secure forwarding protocols, in: IEEE Computer Security Foundations Symposium, CSF, 2021, pp. 159–174.

[18] F. Zhang, L. Jia, C. Basescu, T. H. Kim, Y. Hu, A. Perrig, Mechanized network origin and path authenticity proofs, in: ACM SIGSAC Conference on Computer and Communications Security, CCS, 2014, pp. 346–357.

[19] L. Ma, K. Bu, N. Wu, T. Luo, K. Ren, Atlas: A first step toward multipath validation, Computer Networks 173 (2020) 107224.

[20] K. Bu, A. Laird, Y. Yang, L. Cheng, J. Luo, Y. Li, K. Ren, Unveiling the mystery of Internet packet forwarding: A survey of network path validation, ACM Computing Surveys 53 (5) (2020) 104:1–104:34.

[21] J. Naous, M. Walfish, A. Nicolosi, D. Mazières, M. Miller, A. Seehra, Verifying and enforcing network paths with ICING, in: Conference on Emerging Networking Experiments and Technologies, CoNEXT, 2011, pp. 30:1–30:12.

[22] H. Cai, T. Wolf, Source authentication and path validation with orthogonal network capabilities, in: IEEE Conference on Computer Communications (INFOCOM) Workshops, 2015, pp. 111–112.

[23] D. Levin, Y. Lee, L. Valenta, Z. Li, V. Lai, C. Lumezanu, N. Spring, B. Bhattacharjee, Alibi routing, in: ACM SIGCOMM Conference, 2015, pp. 611–624.

[24] A. He, K. Bu, Y. Li, E. Chida, Q. Gu, K. Ren, Atomos: Constant-size path validation proof, IEEE Transactions on Information Forensics and Security 15 (2020) 3832–3847.

[25] F. Yang, K. Xu, Q. Li, R. Lu, B. Wu, T. Zhang, Y. Zhao, M. Shen, I know if the journey changes: Flexible source and path validation, in: IEEE/ACM International Symposium on Quality of Service, IWQoS, 2020, pp. 1–6.

[26] B. Sengupta, Y. Li, K. Bu, R. H. Deng, Privacy-preserving network path validation, ACM Transactions on Internet Technology 20 (1) (2020) 5:1–5:27.

[27] Y. Shen, T. N. Dinh, M. T. Thai, Adaptive algorithms for detecting critical links and nodes in dynamic networks, in: IEEE Military Communications Conference, MILCOM, 2012, pp. 1–6.

[28] I. Cidon, R. Rom, Y. Shavitt, Analysis of multi-path routing, IEEE/ACM Transactions on Networking 7 (6) (1999) 885–896.

[29] H. Krawczyk, T. Rabin, Chameleon signatures, in: Network and Distributed System Security Symposium, NDSS, The Internet Society, 2000.

[30] NIST, Recommendation for block cipher modes of operation: Methods and techniques, https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf (Accessed: November 2021).

[31] NIST, Advanced Encryption Standard (AES), https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf (Accessed: November 2021).

[32] NIST, Recommendation for block cipher modes of operation: The CMAC mode for authentication, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38b.pdf (Accessed: November 2021).

[33] E. Stefanov, E. Shi, FastPRP: Fast pseudo-random permutations for small domains, IACR Cryptology ePrint Archive 2012 (2012) 254.

[34] D. Catalano, D. Fiore, R. Gennaro, Certificateless onion routing, in: ACM SIGSAC Conference on Computer and Communications Security, CCS, 2009, pp. 151–160.

[35] I. C. Avramopoulos, H. Kobayashi, R. Wang, A. Krishnamurthy, Highly secure and efficient routing, in: IEEE Conference on Computer Communications, INFOCOM, 2004, pp. 197–208.

[36] B. Raghavan, A. C. Snoeren, A system for authenticated policy-compliant routing, in: ACM SIGCOMM Conference, 2004, pp. 167–178.

[37] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, A. Perrig, HORNET: High-speed onion routing at the network layer, in: ACM SIGSAC Conference on Computer and Communications Security, CCS, 2015, pp. 1441–1454.

[38] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, S. Shenker, Ethane: Taking control of the enterprise, in: ACM SIGCOMM Conference, 2007, pp. 1–12.

[39] S. Peter, U. Javed, Q. Zhang, D. Woos, T. E. Anderson, A. Krishnamurthy, One tunnel is (often) enough, in: ACM SIGCOMM Conference, 2014, pp. 99–110.

[40] V. N. Padmanabhan, D. R. Simon, Secure traceroute to detect faulty or malicious routing, Computer Communication Review 33 (1) (2003) 77–82.

[41] E. L. Wong, P. Balasubramanian, L. Alvisi, M. G. Gouda, V. Shmatikov, Truth in advertising: Lightweight verification of route integrity, in: ACM Symposium on Principles of Distributed Computing, PODC, 2007, pp. 147–156.

[42] A. Chen, A. Haeberlen, W. Zhou, B. T. Loo, One primitive to diagnose them all: Architectural support for Internet diagnostics, in: European Conference on Computer Systems, EuroSys, 2017, pp. 374–388.

[43] K. L. Calvert, J. Griffioen, L. B. Poutievski, Separating routing and forwarding: A clean-slate network layer design, in: International Conference on Broadband Communications, Networks and Systems, BROADNETS, 2007, pp. 261–270.

[44] H. Cai, T. Wolf, Source authentication and path validation in networks using orthogonal sequences, in: International Conference on Computer Communication and Networks, ICCCN, 2016, pp. 1–10.

[45] H. Cai, T. Wolf, Implementation of network source authentication and path validation using orthogonal sequences, in: International Conference on Computer Communication and Networks, ICCCN, 2020, pp. 1–7.

[46] B. Wu, K. Xu, Q. Li, Z. Liu, Y. Hu, M. J. Reed, M. Shen, F. Yang, Enabling efficient source and path verification via probabilistic packet marking, in: IEEE/ACM International Symposium on Quality of Service, IWQoS, 2018, pp. 1–10.

[47] W. Diffie, M. E. Hellman, New directions in cryptography, IEEE Transactions on Information Theory 22 (6) (1976) 644–654.

[48] T. P. Pedersen, Non-interactive and information-theoretic secure verifiable secret sharing, in: Advances in Cryptology - CRYPTO, 1991, pp. 129–140.

[49] IETF, Network time protocol version 4: Protocol and algorithms specification, `https://datatracker.ietf.org/doc/html/rfc5905` (Accessed: November 2021).

[50] J. Eriksson, M. Faloutsos, S. V. Krishnamurthy, Routing amid colluding attackers, in: IEEE International Conference on Network Protocols, ICNP, 2007, pp. 184–193.

[51] Y. Jiang, H. Zhu, M. Shi, X. S. Shen, C. Lin, An efficient dynamic-identity based signature scheme for secure network coding, Computer Networks 54 (1) (2010) 28–40.

[52] W. Dai, Crypto++ 5.6.0 benchmarks, `https://www.cryptopp.com/benchmarks.html` (Accessed: November 2021).