Privacy-Preserving and Lightweight Verification of Deep Packet Inspection in Clouds

Xiaoli Zhang, Wei Geng, Yiqiao Song, Hongbing Cheng, *Member, IEEE*, Ke Xu, *Senior Member, IEEE, Member, ACM*, and Qi Li, *Senior Member, IEEE*

***Abstract*— In the trend of network middleboxes as a service, enterprise customers adopt in-the-cloud deep packet inspection (DPI) services to protect networks. As network misconfigurations and hardware failures notoriously exist, recent efforts envision to ensure the execution integrity of DPI services in untrusted clouds. However, they either require enterprise customers to know proprietary DPI rulesets of cloud providers or introduce forbidden overhead in the network context. In the paper, we propose a privacy-preserving and lightweight verification scheme that efficiently checks whether in-the-cloud DPI services run correctly without leaking private DPI rulesets. Particularly, our design introduces one trusted third party to perform privacy-preserving and trustworthy ruleset evaluation and DPI execution verification. Meanwhile, it devises a novel DPI ruleset authentication method that enables tamper-proof DPI operations and facilitates fast proof generation. The proofs can be verified without requiring the verifier to always maintain all rulesets. To further reduce the verification costs while resisting cloud cheating behaviors like bias treatments of packets, it employs a commitment-based delayed sampling mechanism which requires the DPI services to first demonstrate that all packets have been processed before receiving sampling decisions. Moreover, extensive experiments are conducted based on Click modules. The results show that the proposed scheme is practical and only incurs the real-time overhead of 10-20 microseconds.**

***Index Terms*— Middlebox verification, cryptographic protocols, network security, computer network reliability.**

Manuscript received 16 November 2022; revised 12 April 2023; accepted 19 May 2023; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Liu. This work was supported in part by NSFC under Grant 62072407, Grant 62132011, and Grant 61825204; in part by the Zhejiang Provincial Natural Science Foundation of China under Grant LQ23F020019; in part by the "Leading Goose Project Plan" of Zhejiang Province under Grant 2022C01086 and Grant 2022C03139; in part by the National Key Research and Development Program of China under Grant 2022YFB2701400; and in part by the Beijing Outstanding Young Scientist Program under Grant BJJWZYJH012019100003011. *(Corresponding authors: Qi Li; Hongbing Cheng.)*

Xiaoli Zhang, Yiqiao Song, and Hongbing Cheng are with the Department of Computer Science, Zhejiang University of Technology, Hangzhou 310023, China (e-mail: xiaoli.z@outlook.com; 201806062522@zjut.edu.cn; chenghb@zjut.edu.cn).

Wei Geng is with the Department of Computer Science and Technology, and the Institute of Network Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: gengw21@mails.tsinghua.edu.cn).

Ke Xu is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: xuke@tsinghua.edu.cn).

Qi Li is with the Institute of Network Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: qli01@tsinghua.edu.cn).

Digital Object Identifier 10.1109/TNET.2023.3282100

I. INTRODUCTION

MODERN enterprise networks heavily rely on middleboxes like network intrusion detection systems [1], [2] and web application firewalls [3], [4] to defeat ubiquitous network security threats [5], [6]. Those middleboxes perform deep packet inspection (DPI) tasks, namely, checking packets' payloads against malicious signatures. As indicated by a recent report [7], DPI devices have led to a market valued at about $19B in 2023 and would take an increasingly important role in the future.

With the prevalence of cloud computing and network function virtualization (NFV) [8], enterprises usually rent in-the-cloud DPI services rather than deploying in-house middleboxes [5], [9], [10], so as to offload local maintenance burdens and hardware capital costs. In practice, an enterprise customer pays for DPI services in terms of service level agreements (SLAs) and interested DPI rulesets. In turn, the cloud provider initializes those services with the specified rulesets and guarantees SLAs by allocating resources dynamically according to real-time traffic changes. Usually, DPI rulesets are important intellectual properties of cloud providers, because they include valuable signatures of the latest attacks, malware infections, policy violations, and other exposures.

Despite the benefits of in-the-cloud middlebox services, enterprise customers cannot ensure whether those DPI services run as expected due to the following reasons. First, the customers lose direct control over those computation-intensive DPI services, rendering the cloud provider chances to cheat via setting up simple network rulesets for economic profits [11], [12]. Second, unexpected hardware failures, software bugs, and network misconfigurations inevitably occur in the cloud environment [13]. All of these problems would lead to SLA violations and further incur tremendous losses to enterprises, especially for those deploying middlebox services on critical paths of QoS-sensitive or security-sensitive applications [14]. Therefore, it is highly desired for enterprise customers to ensure that network traffic is correctly processed by remote middlebox services.

In the paper, we aim to devise a verification scheme to check the execution integrity of remote DPI services with the following design objectives. 1) The proposed scheme should be able to evaluate the quality of the rulesets and

confirm that the rulesets installed in the middlebox services are indeed enterprise-bought ones. 2) It should have the ability to detect the packet processing correctness of DPI services. That is, it should guarantee that the whole packet payloads are inspected by DPI services with the intended DPI ruleset. 3) It should also preserve the privacy of the rulesets against enterprises, because the rulesets are critical intellectual property of cloud providers. 4) The verification process should be efficient in terms of real-time computation overhead and communication overhead. 5) The proposed scheme should be agilely developed in a minimally intrusive manner.

Unfortunately, existing research efforts cannot achieve the above design goals simultaneously as shown in Table I. Specifically, some cryptographic primitives like zero knowledge proof protocols are designed to verify the integrity of remote generic computations while preserving the privacy of some input information [15]. However, their proof generation is typically thousands of times slower than native execution. Blind signatures enable a user to obtain a signature from a signer without revealing the message to be signed [20]. Anonymous credential schemes allow an individual to prove possession of certain attributes (such as age or nationality) without leaking their identity [21]. While providing privacy-preserving properties, those techniques cannot be applied to check the computation integrity of remote DPI services. Authenticated data structure constructions guarantee the operation integrity of outsourced databases [16], [17]. Unfortunately, they not only reveal data knowledge in verification, but also introduce large query latency (about two orders of magnitude higher than the original queries).

Besides cryptographic designs, some prior arts leverage TEE (Trusted Execution Environment), e.g., Intel SGX [22], to attest to the computation integrity [18], [19]. Their extra assumption of trusted hardware goes against our agile deployment goal, i.e., *Goal 5)*. Other research advancements involve duplicated execution for verification [11], [12]. They usually replay packets at local middleboxes and generate proof references to check the execution correctness of middleboxes in the cloud. Inevitably, they require that enterprise customers have knowledge of DPI rulesets, thus violating the *Goal 3)*. Additionally, these efforts cannot estimate the value of remote rulesets while preserving their privacy.

The analysis above motivates us to build a full-fledged verification scheme that satisfies all design goals. Unfortunately, it is a non-trivial task because of the following three challenges. Firstly, it is challenging to implement ruleset evaluation and DPI execution verification without the knowledge of the rulesets. Second, constructing execution proofs for DPI services is difficult, since the operations of DPI services are sophisticated that not only examine packet headers but also inspect packet payload. Traditional middlebox verification methods that generate proofs based on packet headers or forwarding ports like [12] cannot be adopted. The third challenge is how to perform the verification in an efficient and attack-resistant manner. Sampling-based verification approaches can achieve the goal of high efficiency (*Goal 4)*), yet fail to defeat untrusted clouds with biasing behaviors (e.g., only processing sampled packets).

To overcome the first challenge, we introduce a trusted third party based on the trusted execution environment (TEE), e.g., Intel SGX [22], as the starting point. The SGX-based third party is responsible for evaluating the value of rulesets and checking the execution integrity of DPI services without leaking the privacy of DPI rulesets. Different from existing TEE-based solutions [18], [19], our scheme requires only one SGX-enabled device and thus can be agilely deployed in the current cloud infrastructure. To address the second challenge, we devise a lightweight authentication method for DPI rulesets. With authenticated DPI rulesets, any operations can be directly recorded in a tamper-proof manner, facilitating fast proof generation. The proofs can be further verified by checking whether these operations are consistent with packet payloads without requiring the verifier to maintain the entire ruleset. In addition, we propose a commitment-based delayed sampling approach that requires the cloud to bind succinct execution assurances, i.e., commitments, with packets at runtime. Proofs for sampled packets can then be efficiently verified afterward while resisting potential cloud cheating behaviors.

In summary, our contributions are three-fold:

- We propose a privacy-preserving verification framework to ensure the execution integrity of DPI services in clouds without revealing DPI rulesets. Especially, it leverages one trusted third party to perform privacy-preserving and trustworthy ruleset evaluation and DPI execution verification.
- Our scheme is lightweight and secure. It introduces a lightweight authentication method for DPI rulesets to achieve fast proof generation and verification. Meanwhile, it employs a commitment-based delayed sampling technique to reduce overall verification costs while resisting the cloud cheating behaviors.
- We theoretically analyze the security of the proposed scheme and implement a prototype system based on Click modules. The experimental results with real traffic show that our scheme introduces negligible real-time overhead of about 10-20 microseconds.

II. Related Work

Verifiable middlebox execution. With the advent of network functions as a service, recent proposals explored how to ensure the execution correctness of outsourced middleboxes. Some researchers envision porting remote middlebox services into trusted execution environments based on trusted hardware [18], [19], e.g., Intel SGX. The heavy dependence on trusted hardware would incur sophisticated development/deployment overhead and possible performance degradation, making the solution hard to be adopted by real businesses. Contrastly, we use only one TEE-enabled server which can be easily deployed in the existing cloud infrastructure. Other research arts [11], [12] verify the execution correctness of remote middlebox services by duplicating middlebox execution in local environments. Unfortunately, they would leak the privacy of DPI rulesets. Another work called vSFC [23] checks the correctness of the outsourced path traversal in the network layer, yet fails to ensure whether middlebox applications that deal with L7 protocols run faithfully.

TABLE I
COMPARISON OF REPRESENTATIVE SOLUTIONS FOR COMPUTATION VERIFICATION

Related work	Design goals				
	Estimating the value of rulesets	Detecting DPI packet processing correctness	Preserving privacy of rulesets	Efficient verification	Agile development
Zero knowledge proof protocols [15]	✓	✓	✓	✗	✓
Authenticated data structures [16], [17]	✗	✓	✗	✗	✓
TEE-guarded middleboxes [18], [19]	✗	✓	✓	✓	✗
Duplicated middlebox execution [11], [12]	✗	✓	✗	✓	✓
Our work	✓	✓	✓	✓	✓

Cryptographic verification methods. Researchers from the cryptographic community propose a series of primitives to achieve common verification, including zero knowledge proof (ZKP) protocols [15], blind signatures [20], anonymous credential schemes [21], and authenticated data structures [16], [17]. ZKP protocols [15] can verify generic computations while preserving the input privacy. However, they cannot be applied in the network context due to huge proof generation costs (several orders of magnitude higher than the original execution). Blind signatures [20] obtain signatures without leaking the messages to be signed, and anonymous credential schemes [21] achieve privacy-preserving identification of some identity statements. However, they cannot check the computation integrity of middlebox services. Another primitive of authenticated data structures can ensure the correct execution of pattern-matching queries over outsourced databases [16], [17]. While the scheme proposed by Papadopoulos [16] only supports single-pattern matching, Zhou et al. [17] focus on verifying the correctness of multi-pattern matching operations, which is more relevant to our work. However, both of them incur high computation and communication overhead and thus fail to be applied in the network context.

Network-wide invariant verification. A line of work verifies whether configurations of network elements are compliant with network-wide invariants like network reachability or isolation. VMN [24] models various network elements, including routers and middleboxes, as a forwarding model with a set of abstract packet classes. To facilitate fast and scalable network verification, VMN divides a network into several subnetworks and verifies invariants in each subnetwork. Besides the above static verification of network configurations, Buzz [25] is proposed to test the correctness of network policies with middleboxes via probing packets. However, these efforts work in trusted network environments. In contrast, we focus on bringing execution assurance of DPI services deployed in the untrusted cloud.

Privacy-preserving middleboxes. Some research advancements protect the privacy of enterprise traffic when using in-the-cloud middleboxes. BlindBox [26] develops encrypted token matching to enable privacy-preserving DPI services. Embark [9] supports a wide spectrum of middlebox services like load balancers and firewalls. PrivDPI [27] further reduces the setup delay in achieving privacy-Preserving encrypted traffic inspection. These studies are somewhat orthogonal to our work that aims to verify the computation integrity of outsourced middlebox services.

Security and privacy countermeasures in cloud computing. Cyber security threats have drawn widespread attention in the area of cloud computing [28] and mobile networks [29]. At the same time, related mitigation and defense techniques are proposed to confront a wide variety of security and privacy challenges. Almaiah et al. [30] devise a new hybrid encryption method using the elliptic curve cryptosystem and hill cipher algorithm to achieve secure data exchange over a mobile ad hoc network. Adil et al. [31] introduce a lightweight anonymous authentication technique to guarantee the authenticity of operational devices and resolve black-hole attacks in dynamic wireless sensor networks. In addition to the method of excluding malicious nodes in networks [31], Al Hwaitat et al. [32] improve the Particle Swarm Optimization (PSO) algorithm to enhance the detection of jamming attack sources over randomized mobile networks. The above studies are orthogonal to our focus.

III. PROBLEM STATEMENT

A. System Setting

In the paper, we study a network service scenario where enterprises adopt in-the-cloud DPI middleboxes to secure their network with reduced capital and operational costs. Usually, the cloud shows available DPI network services equipped with different ruleset qualities, peak traffic processing capabilities, etc. The enterprise specifies a deployment topology of remote DPI services with desired SLAs. In particular, the enterprise sets up a gateway (GW) in its own network, similar to [5], [9], [11]. The GW is responsible for directing its traffic to the cloud and performing related operations for subsequent DPI verification.

B. Threat Model

Untrusted cloud. The cloud is not fully trusted by enterprise customers [9], [11]. A wide range of abnormities may affect the correctness of DPI execution. First, potential network misconfigurations, unreliable software implementations, and unpredictable hardware failures may result in network errors [33]. Second, the cloud may deliberately make traffic bypass desired DPI functions due to economic incentives. For example, it may obtain profits with minimal resource consumption or conceal sudden network failures. More importantly, we study a strong attack model where the cloud may confound any detection or verification methods of the

enterprise. It may launch coward attacks and only process special packets that are used for detection.

Honest-but-curious enterprises. The GW in the enterprise customer is trusted. It faithfully performs verification-related operations. However, it may try to steal rulesets installed in the remote DPI services. For instance, network attackers may pretend to be normal customers to probe DPI rulesets and further launch stealthy network attacks. In addition, as DPI rulesets are critical intelligence properties of the cloud, a business competitor may be motivated to obtain the rulesets to further improve its own DPI rulesets.

Therefore, our security objective is to verify the execution integrity of in-the-cloud DPI services against untrusted clouds while preserving the privacy of DPI rulesets against curious enterprise customers. We assume that the cloud has no incentive to make any deviation after faithfully performing DPI functions. Other network attacks that lead to packet traversing deviations are not our focus and can be orthogonally addressed by existing path validation protocols [23].

C. Preliminaries

DPI services. We briefly look at how DPI services process packets. DPI devices like network intrusion detection systems (NIDSs) [1], [2] inspect packet payloads and apply multi-pattern string matching based on DPI rulesets to discover malicious patterns in network flows. As the most popular string-matching algorithm, Aho-Corasick (AC) algorithm [34] has been widely applied in contemporary DPI services, including Snort [1], Suricate [2], ModSecurity [3], and CloudFlare's Web Application Firewall [4]. It constructs a deterministic finite automaton (DFA) based on the pattern ruleset and outputs all matched strings when scanning the query string over the DFA.

In detail, the DFA is built in two phases. First, a trie tree is a prefix tree where each pattern string is denoted as a path from the root to a node and each node from the root shows the common prefix. All edges appended in the current stage are named as *forward transitions*. Meanwhile, two types of additional edges, called *failP* and *suffixP* pointers, are added to each node. Specifically, *failP* pointers are used to achieve an efficient transition from a longer suffix to a shorter one, when there is no eligible edge in the forward transition. The *suffixP* pointer of a node represents the longest suffixes of the node contained in the ruleset. That means, if the node v is traversed when searching the matched prefixes for the query q on the trie, we can efficiently derive all suffixes for the string along the path from the root to v by backtracing from v along the *suffixP* pointer until the root is visited.

Figure 1 shows an example of the AC automaton for the pattern set {"eds", "d", "ti", "tied", "tic"}. The AC automaton traverses the tree from the root and takes one character from the query string in order. For each input character, if the current node has an eligible edge, the automaton moves along the edge to the child node and outputs the matched patterns at present by iteratively backtracing along *suffixP* pointers. If there is no such edge, it visits the *failP* pointers to discover a node that

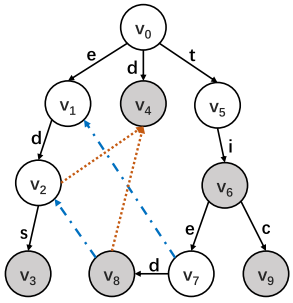


Fig. 1. An example of AC automaton for the ruleset {"eds", "d", "ti", "tied", "tic"}. All patterns end with a gray node. The black lines show the forward transitions, the blue lines represent *failP* pointers, the red lines denote *suffixP* pointers.

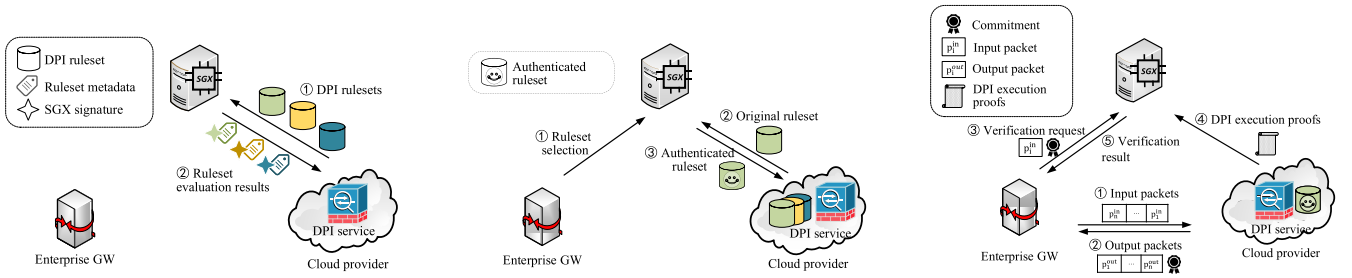
maintains the edge corresponding to the character for further traversal. Finally, all matched patterns are derived once the last character in the query is processed. Take a query string "stieds" as an example. The traversing nodes and corresponding output patterns are v_0 ( ) $\rightarrow$ v_5 ( ) $\rightarrow$ v_6 ("ti") $\rightarrow$ v_7 ( ) $\rightarrow$ v_8 ("tied", "d") $\rightarrow$ v_3 ("eds").

As for regular expression matching, DPI devices usually extract patterns from the regular expressions and perform the exact string as the first step. In our work, we focus on the exact multi-pattern matching process based on the AC automaton, which will also be useful for the verification of regular expression matching.

SGX. Intel Software Guard Extension (SGX) offers integrity and confidentiality protection for user-level code and data while merely trusting on-chip security engines [22]. Concretely, SGX is a set of secure instructions integrated by specific Intel CPUs such as Skylake or Icelake, which can build a trusted execution environment called enclave. Any access to enclave memory is forbidden. Meanwhile, one core technique for SGX is the remote attestation that attests whether the software in the enclave is faithfully initialized and runs.

For the sake of simplicity, we follow the formal SGX model. There is an attestation key pair $(\mathsf{sk}_{\mathsf{TEE}}, \mathsf{pk}_{\mathsf{TEE}})$ generated by an enclave instance and an existentially unforgeable signature scheme named Σ. An enclave can compute a digital signature $\sigma = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{TEE}}, \Phi, \mathsf{out})$ as the attestation proof. The attestation can be further verified via $\Sigma.\mathsf{Veri}(\mathsf{pk}_{\mathsf{TEE}}, \sigma, \Phi, \mathsf{out})$ to demonstrate that the output out is indeed produced by the software Φ launched in the enclave.

Cryptographic accumulator. A cryptographic accumulator combines a large set of values into one succinct value [35], [36]. In the literature, there are a number of cryptographic constructions, each with varied functionalities and security properties. Known cryptographic accumulators are constructed based on Merkle hash trees [37], RSA [38], and bilinear maps [39]. They are called dynamic cryptographic accumulators, since they support both additions and deletions of elements. Meanwhile, they can generate a witness or proof of membership to prove that an element belongs to the set. They basically provide two security properties: 1) correctness meaning that an honest witness can always prove membership for every element in the accumulator; 2) soundness denoting



(a) Phase I: privacy-preserving and trustworthy ruleset evaluation

(b) Phase II: lightweight DPI ruleset authentication

(c) Phase III: commitment-based DPI execution verification

Fig. 2. The workflow of the proposed scheme.

that no valid witness can be constructed to prove membership for every element not in the accumulator. They are broadly applied in many applications like membership revocation in ID escrow systems. In the paper, we customize cryptographic accumulators according to our specific design requirements (see Sec. V-B).

IV. Design Overview

In this section, we introduce a practical scheme that verifies the execution correctness of in-the-cloud DPI services while meeting all goals specified in Section I. One intuitive solution is that the GW first randomly selects a small subset of packets and then verifies the execution proofs returned from remote DPI services via local duplicated execution. However, to simultaneously achieve all goals, we face the following **key challenges**:

Challenge 1: There is a natural conflict between the requirements of enterprise customers and the cloud. As for the enterprise customers, they need to estimate the value of rulesets and detect whether the desired rulesets are used to process traffic. Moreover, middlebox execution proofs usually need to include the knowledge of DPI rulesets [11], [12]. However, these requirements inevitably breach the ruleset privacy from the perspective of the cloud.

Challenge 2: It is challenging to construct proofs for the execution of complex DPI services in a lightweight manner. Existing efforts either require the verifier to store the whole rulesets [11], [12], or introduce unacceptable overhead [17]. For example, the work most related to us [17] verifies the correctness of queries on authenticated AC automaton. It incurs huge processing latency (about two orders of magnitude higher than the latency of original queries) and a big proof size (about ten times larger than the size of one packet).

Challenge 3: The sampling-based checking method is necessary to reduce verification overhead yet would be disturbed by the cloud cheating behaviors. The untrusted cloud may treat packets preferentially, for instance, only perform sampled packets. While researchers proposed the delayed-exposure technique to defeat the sampling bias of merchant routers [40], it cannot successfully resist the malicious cloud with unlimited computation and storage power [12].

To overcome the above challenges, our scheme introduces three types of entities: the cloud that runs DPI services, enterprise customers that use remote DPI services and deploy one gateway for verification, and one SGX-based trusted third party that verifies DPI execution in a privacy-preserving manner. Assuming that we have established secure channels between the three entities, the **high-level workflow** of our scheme consists of three stages:

Firstly, we propose a *privacy-preserving verification framework* to guarantee the execution integrity of in-the-cloud DPI services without leaking ruleset privacy. In particular, we introduce a trusted third party, as shown in Fig. 2. It is developed based on the technique of Intel SGX and is responsible for ruleset evaluation and DPI verification in a privacy-preserving and trustworthy manner (see Fig. 2(a) and Fig. 2(c)). Thus, only evaluation and verification results are returned to the enterprise without leaking the privacy of DPI rulesets. Related details are described in Sec. V-A.

Secondly, we devise a *lightweight DPI ruleset authentication* algorithm, as depicted in Fig. 2(b). Before deployment of DPI services in the cloud, DPI rulesets are authenticated by the SGX-based third party for security and efficiency. The authenticated AC automaton accumulates all matched patterns for each node in a tamper-proof manner, enabling it to securely record any query operations. Therefore, DPI execution proofs can be efficiently constructed by directly collecting all traversing nodes in each query. Besides, the authentication method effectively avoids the verifier always maintaining the whole rulesets (see Sec. V-B).

Thirdly, to further reduce verification costs while defeating possible cheating behaviors of the cloud, we develop the method of *commitment-based DPI execution verification* as depicted in Fig. 2(c). Before the exposure of sampling decisions, the cloud has to generate a commitment to prove that all packets have been processed. Thus, our scheme not only achieves efficient sampling-based verification, but also prevents the cloud from performing bias treatments according to which packets are sampled. All design details are presented in Sec. V-C.

V. Design Details

In this section, we present the design details of our scheme, including privacy-preserving and trustworthy ruleset evaluation, lightweight DPI ruleset authentication, and

TABLE II
KEY NOTATIONS

Hash	cryptographic hash function
HMAC	hash-based message authentication code
U	pseudo-random function
AccGen	accumulator generation function
AccUpdate	accumulator update function
AccVerify	accumuator verification function
sk	secret key used by HMAC
$RuleSet$	a ruleset
$RuleID$	unique identifier of one ruleset
$RuleMeta$	ruleset metadata
pkt	packet
n, nid	node and its identifier in AC trie
$\{w_i\}$	the set of child nodes for one node in AC automaton
acc	cryptographic accumulator
σ	signature
p	pattern in a DPI ruleset
$\mathcal{T}$	AC automaton
$Proof^{pkt}$	succinct packet-level proof for commitment generation
$Proof^{f}$	succinct flow-level proof for commitment generation
ζ	commitment message
$dProof^{pkt}$	detailed packet-level proof in verification
π	all detailed proofs for one sampled flow in verification

commitment-based DPI execution verification. Table II lists the key notations used in this paper.

A. Privacy-Preserving and Trustworthy Ruleset Evaluation

We leverage an SGX-based third party for ruleset evaluation, so as to prevent the cloud from tampering with rulesets when running DPI services. Since rulesets are transmitted via a secure channel from the cloud to the enclave, the privacy of rulesets is protected in the network. Meanwhile, the rulesets' privacy would not be leaked on the SGX platform due to enclave memory encryption and isolation. We do not focus on the side-channel attacks that may compromise the data confidentiality of the enclave [41], because they can be orthogonally addressed by existing side-channel defense schemes like [42].

In detail, we perform a RuleEvaluation function in the enclave to evaluate the ruleset values. It takes rulesets as input (i.e., $RuleSet$), then generates a unique identifier $RuleID$ and evaluates the value of the ruleset denoted as $RuleMeta$ (i.e., the ruleset metadata). In particular, $RuleID$ is calculated using a cryptographic hash function (SHA256) based on the ruleset contents as follows:

$$RuleID = \mathsf{Hash}(RuleSet) \tag{1}$$

$RuleMeta$ is calculated by counting the number of malicious signatures. In addition, for the purpose of subsequent rule verification, we also generate a key sk for the ruleset. All the information (i.e., $RuleID$, $RuleMeta$, sk) is securely stored in the SGX platform via the SGX sealing technology [22]. Finally, the enclave signs the output $RuleID$ and $RuleMeta$ to prove the trustworthiness of the evaluation results without leaking ruleset privacy. The pseudocode of RuleEvaluation function is shown in Algorithm 1.

Algorithm 1 RuleEvaluation Function in the Enclave

Input: $RuleSet$
Output: $RuleID, RuleMeta, \sigma_{\mathsf{SGX-e}}$
$RuleID = \mathsf{Hash}(RuleSet)$;
$sk = \mathsf{RNG}(1^{\lambda})$;
$RuleMeta = \mathsf{CountPatterns}(RuleSet)$;
Store $< RuleID, RuleMeta, sk >$;
$\sigma_{\mathsf{SGX-e}} = \Sigma_{\mathsf{SGX}}.\mathsf{Sign}(\mathcal{M}_{\mathsf{enclave}}, RuleID||RuleMeta)$

Algorithm 2 RuleSetup Function for Ruleset Authentication in SGX Enclave

function RuleSetup($RuleSet, RuleID'$)**:**
  if $RuleID' \neq \mathsf{Hash}(RuleSet)$ **then return** err ;
  Retrieve $< RuleID, sk >$;
  Construct AC automaton $\mathcal{T}$;
  $r_{\mathcal{T}}.acc = \mathsf{AccGen}(\phi)$;
  Let Q be a queue;
  $Q.\mathsf{Enqueue}(r_{\mathcal{T}})$;
  while $Q \neq \emptyset$ **do**
    $n = Q.\mathsf{Dequeue}()$;
    if p_n *is a pattern* **then**
      $n.acc = \mathsf{AccUpdate}(w_{suffixP}.acc, p_n)$;
    else
      $n.acc = n.w_{suffixP}.acc$;
    $n.\sigma_n = \mathsf{HMAC}(sk, n.nid||\{n.w\}||n.acc)$;
    for *all child nodes of* n **do**
      $Q.\mathsf{Enqueue}(n.w)$;
  $\sigma_{\mathsf{SGX-s}} = \Sigma_{\mathsf{SGX}}.\mathsf{Sign}(\mathcal{M}_{\mathsf{RuleSetup}}, \mathcal{T})$;
  return $(\mathcal{T}, \sigma_{\mathsf{SGX-s}})$;

B. Accumulator-Based Lightweight Ruleset Authentication

The ruleset authentication is performed by the trusted third party for further proof construction and verification of DPI execution. The third party first constructs the initial AC automaton and then authenticates each node in the trie tree by recording corresponding matched patterns in a tamper-proof manner. Consequently, all nodes in a traversal path can be used to construct proofs to demonstrate whether the DPI function faithfully inspects packet payloads and outputs all matched patterns in the ruleset. The whole authentication process is shown in Algorithm 2.

To authenticate the AC automaton, each node stores a tuple:$(nid, \{w_i\}_{i\in\Sigma\cup\{failP, suffixP\}}, acc, \sigma_n)$. nid is the identifier of each node, w_i is the child node for the edge with a character i, Σ is the set of all possible characters for the edges in the AC automaton. w_{failP} and $w_{suffixP}$ denote the nodes

pointed via two special pointers $failP$ and $suffixP$, respectively. acc is the cryptographic accumulator of all suffixes of the word represented by the path from the root to the current node. This information is used to check the integrity of the matched patterns for a query string. σ_n is a signature over the information of the node, so as to prevent the cloud or other attackers from tampering with the AC automaton. Note that, acc of the root node (denoted as $r_{\mathcal{T}}$) is NULL, $\{w_i\}$ for the leaf node (i.e., the node without child edges) is NULL as well.

Before authenticating the AC automaton, our scheme first verifies whether the DPI ruleset is tampered with after the ruleset evaluation. It recalculates the hash value based on the ruleset and determines whether it is consistent with the corresponding identifier securely stored in the SGX machine. Then, it authenticates the AC automaton via a breadth-first search approach. For each node, e.g., n, it retrieves the node pointed by the $suffixP$ pointer and gets its corresponding accumulator, i.e., $n.w_{suffixP}.acc$. Then, it sets the accumulator of the node according to whether the string from the root to the current node (denoted as p_n) is a pattern in the ruleset, namely:

$$n.acc = \begin{cases} \mathsf{AccUpdate}(n.w_{suffixP}.acc, p_n), & \text{if } p_n \in RuleSet, \\ n.w_{suffixP}.acc, & \text{if } p_n \notin RuleSet. \end{cases} \tag{2}$$

The function AccUpadate is to update an accumulator by adding an element (which will be described in the next paragraph). Finally, it performs an HMAC function with sk as the secret key to authenticate the node content:

$$\sigma_n = \mathsf{HMAC}(sk, nid||\{w\}||acc). \tag{3}$$

After finishing the traversal of the AC trie, the trusted third party sends the authenticated AC automaton $\mathcal{T}$ along with SGX signatures to the cloud.

Cryptographic hash based accumulator. We use cryptographic accumulators to succinctly represent all matched suffixes for a word from the path to a certain node. Below we propose our accumulator construction with two key requirements and compare it with existing accumulator algorithms.

Our accumulator construction: In the paper, two properties for accumulators are desired: 1) efficiency; 2) collision resistance. The former says the generation and verification of an accumulator are efficient. The latter means that given an accumulator (e.g., acc_S) for a set of values denoted as S, no probabilistic polynomial-time (PPT) adversary can construct another set (e.g., S') with the same accumulator acc_S. Thus, to realize these, we construct a cryptographic hash-based accumulator scheme including three algorithms: AccGen, AccUpadate, and AccVerify. Assume a list of values denoted as $S = \{s_1, s_2, \cdots, s_n\}$, AccGen generates the corresponding accumulator as:

$$\mathsf{AccGen}(\{s_i\}_{i=1}^n) = \mathsf{Hash}(\mathsf{Hash}(\mathsf{Hash}(s_1||s_2)||\cdots)s_n). \tag{4}$$

AccUpdate updates the accumulator (e.g., acc) by adding a new element (e.g., s_i) as:

$$\mathsf{AccUpdate}(acc, s_i) = \mathsf{Hash}(acc||s_i). \tag{5}$$

AccVerify verifies whether the accumulator (e.g., acc) is consistent with a member list (e.g., $\{s_1, s_2, \cdots, s_n\}$).

$$\mathsf{AccVerify}(acc, \{s_i\}_{i=1}^n) = \begin{cases} 1, & \text{if } \mathsf{AccGen}(\{s_i\}_{i=1}^n) = acc, \\ 0, & \text{if } \mathsf{AccGen}(\{s_i\}_{i=1}^n) \neq acc. \end{cases} \tag{6}$$

Comparison with related methods: We notice that there are extensive cryptographic accumulator constructions in academia like RSA-based accumulators [38], bilinear-pairing-based accumulators [39], and (dynamic) Merkle-tree-based accumulators [37]. Although these accumulators provide rich functions like proving membership and supporting element addition and deletion as described in Sec. III-C, they either incur forbidden computation overhead or huge storage costs in the network context. For example, when using the bilinear-pairing-based algorithm without a trapdoor, one recent scheme [17] consumes 183ms when querying a sequence of 1000 letters in a ruleset containing 10^4 patterns, which is about two orders of magnitude higher than the packet processing latency of DPI services. The Merkle-tree-based solution has to store internal nodes for verification. When using the ruleset of Snort v3.0 [1], the maximum number of matched suffixes of one node in AC automaton is about 32, and thus the storage overhead for a single node would be $\log(32+1)\times 32\text{B} = 160\text{B}$.

Compared with the above accumulator approaches, our cryptographic hash-based accumulator construction in the paper incurs small computation overhead (due to merely conducting efficient hash functions) and constantly negligible storage overhead for each node (due to using a hash value as the accumulator). The property of collision resistance is held by the cryptographic hash algorithms. Note that our construction achieves high efficiency (demonstrated in experiments in Sec. VII-D) at the cost of a lack of rich functions like witness generation in the accumulators [37], [38], [39]. Fortunately, those functions are not required here.

C. Commitment-Based Efficient Verification of DPI Execution

This part describes how our scheme efficiently verifies the integrity of the DPI execution while resisting cloud cheating behaviors. Briefly, packets are first randomly sampled at the enterprise gateways, then directed to the in-the-cloud DPI services. The cloud outputs packets along with a commitment message to prove that all packets have been processed. Then, verification requests (i.e., sampled packets) are sent to the trusted third party, which performs the verification via the commitment and specific proofs from the cloud. Finally, only trustworthy verification results are sent to enterprises without leaking the privacy of DPI rulesets. Below we present the design details at enterprise gateways, clouds, and the third party, respectively. We end by introducing how to deal with middlebox service chains.

1) Packet Sampling at Enterprise Gateways: The enterprise GW redirects enterprise traffic to the cloud and randomly samples a small set of packets for verification. Facing the continuous streaming traffic, we divide packets into each batch according to the packet arriving timing and the batch size. Meanwhile, we conduct the verification at the granularity

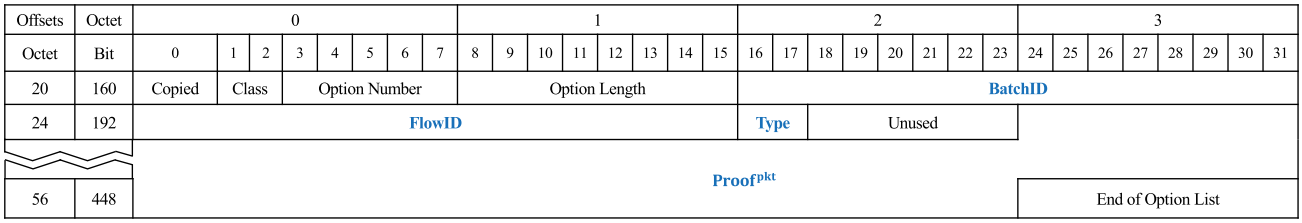


Fig. 3. The customized packet headers for verification. The first two octets show meta information of the option field, and the last one octet represents the end of entire field. The intermediate subfields marked blue are leveraged by our design.

of network flow, since DPI functions inspect the network payloads at the flow level.

In detail, GW inserts four contents into unused header fields of packets, namely, 16-bit $BatchID$ as the unique identifier of a batch, 16-bit $FlowID$ as the flow identifier in a batch, 2-bit $Type$ indicating the type of packets, and 256-bit $Proof^{pkt}$ representing the succinct proof for a packet e.g., pkt, generated from the previous middlebox for verification (which will be used in Sec. V-C.4). The detailed customized packet header for verification is shown in Fig. 3. The type of "00" represents normal data packet denoted as *Data*, "01" shows the special packets indicating the end of a batch called *BatchEND*, and "10" depicts packets triggering the verification denoted as *Request*. It is worth noting that we set the sizes of these fields so that they can accommodate practical scenarios. For example, in an OC-192 link (10 Gbps) with up to 10^7 packets per second, 16-bit $BatchID$ and 16-bit $FlowID$ can label every packet uniquely, even in the extreme case where small flows dominate.

Assuming a batch includes R network flows, GW performs the GWProcess function as shown in Algorithm 3. To determine the flow samples, GW maintains $BatchID_{\text{now}}$ denoting the current batch identifier, $FlowSet$ representing a mapping between the flow identifier $FlowID$ and corresponding five-tuple values (i.e., source IP address, destination IP address, source port, destination port, and the protocol number), a counter r recording the number of flows in the batch, the sampled flows $\{s_1, s_2, \cdots, s_m\}(s_i \in [1, R])$. When receiving a packet, GW assigns a $FlowID$ to the packet based on its five-tuple values and checks whether the flow information is in $FlowSet$ (lines 5-11 in Algorithm 3). Then, GW samples packets and caches the selected ones (lines 14-15 in Algorithm 3). Finally, GW embeds $BatchID$, $FlowID$, and $Type$ into the packet and forwards it to the cloud. Note that, at the end of the batch, GW sends a special packet, i.e., *BatchEND*, to the cloud.

Algorithm 3 GWProcess Function at GW

```
1  function GWProcess(pkt, {s_1, s_2, ..., s_m}):
2      FlowID = Find(FlowSet, pkt.(5-tuple));
3      if FlowID = NULL then
4          r++;
5          if r > R then
6              Send BatchEND;
7              BatchID_now++;
8              Clear FlowSet;
9              FlowID = r = 1;
10             Store < pkt.(5-tuple), FlowID > in
                 FlowSet;
11             Re-generate {s_1, s_2, ..., s_m};
12         FlowID = r;
13         Store < pkt.(5-tuple), FlowID > in FlowSet;
14     if FlowID in {s_1, s_2, ..., s_m} then
15         Store pkt;
16     Embed BatchID_now, FlowID and Type in pkt;
17     Forward pkt;
```

2) Proof Generation at in-the-Cloud DPI Devices: DPI services process packets and follow a two-stage proof generation. First, the DPI services efficiently generate a succinct commitment message (denoted as ζ) to indicate that a batch of packets are honestly inspected by DPI services at runtime. Second, they produce detailed proofs (denoted as π) for the delayed verification requests.

Succinct commitment construction. The commitment message is used to prevent the cloud from evading detection via merely processing sampled packets. To achieve the goal, we employ a widely applied structure, i.e., Merkle hash tree [43], as a case study. It produces a digest for a set of data by building a complete binary tree from the bottom up with a one-way hash function. Thus, by leveraging leaf-to-root nodes, it supports membership tests with logarithmic costs.

As shown in Algorithm 4, when a data packet arrives, the DPI device inspects its payloads and detects malicious patterns by traversing the AC automaton. In particular, in order to produce the commitment, it first generates proofs for flows and then constructs a Merkle tree with these proofs as leaf nodes. The proof for a flow demonstrates that the DPI service has honestly traversed the authenticated AC automaton along the payloads of all packets in the flow, which is constructed as:

$$Proof^f = Proof^f + Proof^{pkt} \cdot r_p, \tag{7}$$

r_p is a pseudo-random number generated by a pseudo-random function $U(e)$, e is the seed of this pseudo-random function and is bound to the packet content. $Proof^{pkt}$ records query operations for a packet pkt as:

$$Proof^{pkt} = \mathsf{Hash}(\{n_i\}||\{\{p_j\}_i\}). \tag{8}$$

$\{n_i\}$ is a sequence of nodes when traversing the authenticated AC automaton according to the packet payload. Note that it does not include the nodes pointed by the *suffixP* pointers

Algorithm 4 DPIProofGen Function at in-the-Cloud DPI Devices

```
1  function DPIProofGen(pkt):
2      if (pkt is a Data packet) then
3          Let N, P be arrays;
4          str = pkt.payload;
5          n = r_T;
6          for (i = 1 to Len(str)) & !(Stopping
             condition) do
7              while (n.w(str[i]) = null) & (n ≠ root_T)
                 do
8                  N.Insert(n);
9                  n = n.w(failP);
10             if (n.w(str[i]) = null) & (n = root_T)
                 then
11                 N.Insert(n);
12                 Continue;
13             n = n.w(str[i]);
14             N.Insert(n);
15             P.Insert(tracePatterns(n));
16         Proof(pkt) = Hash(P||N);
17         r_p = U(e);
18         Proof(pkt.FlowID)+ = Proof(pkt) · r_p;
19         Forward or Drop pkt;
20     if (pkt is a BatchEND packet) then
21         Construct Merkle hash tree MHT;
22         Send ζ = r_MHT to GW;
```

in the traversing path. $\{p_j\}_i$ denotes output patterns when arriving at the node n_i. The authenticated node information is later used to check the integrity of returned patterns.

When receiving the *BatchEND* packet, the DPI device generates a commitment ζ based on the Merkle hash tree (lines 20-22 in Algorithm 4). The leaf nodes of the hash tree are the hash values of the flow-level DPI execution proofs. By constructing a binary tree with a hash function from the bottom up, the value of the root node is the DPI execution commitment ζ which is further sent to the enterprise customer. Note that, the cloud could only store all leaf nodes and remove internal nodes in the Merkle tree after sending the commitment, so as to avoid extra storage consumption.

In practice, one remote DPI function would be launched as multiple instances for load balance. In this case, packets belonging to the same batch may be directed into different instances, making the commitment generation difficult. Regarding this, we can follow the idea proposed by Zhang et al. [12] and develop a two-tier commitment scheme. Here, each DPI instance generates a local commitment for the received flows as usual. Then a controller is set to generate the global commitment by building a Merkle tree with all local commitments as leaves. The key observation behind is that the verification is performed at the granularity of network flows, meanwhile, the packets belonging to one flow are forwarded to the same instance.

Detailed proof generation for delayed verification requests. When the cloud receives a delayed verification request (denoted as *reqFlow*), it produces detailed proofs as follows. Take the flow f as an example. The remote DPI service feeds all packets of the request flow (e.g., $\{pkt_1, pkt_2, \cdots pkt_k\}$) to the AC automaton to obtain the packet-related proof information. For each packet (e.g., pkt_i), the detailed proof denoted as $dProof^{pkt_i}$ consists of all nodes during the traversal of AC automaton while excluding those accessed via the *suffixP* pointers and the corresponding output patterns at each node. That is, $dProof^{pkt_i} = \{(n_1, \{p_x\}_1), (n_2, \{p_y\}_2), \cdots, (n_t, \{p_z\}_t)\}$. Via such a detailed proof, we can concatenate all nodes in order and corresponding matched patterns in order, so as to generate $Proof^{pkt}$ (see Eq.(8)) and further produce $Proof^f$ (see Eq.(7)) in verification.

After finishing the traversal on the AC automaton, the cloud returns the auxiliary information Aux to achieve the verification of the commitment. Aux includes the leaf node corresponding to the *reqFlow* in the hash tree and the sibling nodes on the path from the requested leaf node to the root node, denoted as $\{\Gamma_1, \Gamma_2, \cdots, \Gamma_t\}$. Finally, the detailed proof information π for the sampled flow f is $(dProof^{pkt_1}, dProof^{pkt_2}, \cdots, dProof^{pkt_k}, Aux)$. All the information is sent to the trusted third party via a secure channel. It is obvious that the above procedures are performed after all packets are processed and thus do not affect the real-time packet processing capability of DPI services.

3) DPI Verification at Trusted Third Party: The SGX-based third party verifies the execution integrity of in-the-cloud DPI services after the detailed proofs from the cloud.

The verification proceeds as follows: 1) The third party retrieves the key sk for the ruleset used by the DPI services and verifies the authenticity of each node in the detailed proof via the node signature (see Eq. (3)). 2) It verifies whether the traversal path on the AC automaton is consistent with the flow payloads. Since all nodes include a set of child pointers, we can easily check the correctness of the traversal path for the specific flow payloads. 3) It checks the integrity of the matched patterns for each packet. To achieve this, it accumulates all output patterns of each node in the detailed proof and compares the result with the accumulator of each node (see Equation (6)). 4) It verifies the proof information for the sampled flow. With all verified proofs for packets, we construct the flow-level proof as shown in Eq. (7), then compare its hash value with Γ_1 in the auxiliary proof Aux. 5) We check the correctness of the commitment ζ to ensure that the packets are indeed processed by the cloud at runtime. Specifically, the trusted third party uses the auxiliary information Aux in π to recompute the root node of the Merkle hash tree (e.g., ζ' shown in Eq. (9)) and compare it with the commitment ζ.

$$\zeta' = \mathsf{Hash}(\mathsf{Hash}(\Gamma_1||\Gamma_2)\cdots||\Gamma_t) \tag{9}$$

Finally, the trusted third party returns the verification results to the enterprise user along with the corresponding SGX signature. Since the entire verification process is only performed by the SGX-based third party, the privacy of the remote ruleset would not be leaked.

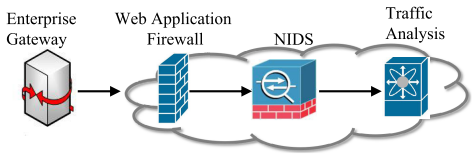


Fig. 4. An example of a middlebox service chain.

Note that, if an enterprise customer requests to check multiple network flows in the same batch, the proof information returned by the cloud can be further de-duplicated to reduce the communication overhead. For example, the traversed nodes along with the matched suffixes are stored only once , while other identical node traversals can be represented using indexes. Also, the auxiliary proofs for different requests may share the same nodes in the Merkle hash tree. The shared information would be transferred only once for efficiency.

4) Verification of Middlebox Service Chain : In practice, enterprise customers deploy a middlebox service chain to initiate an ordered sequence of network functions that must be applied to a certain class of packets. As illustrated in Fig. 4, packets from one GW are sequentially processed by a web application firewall, an intrusion detection system, and a traffic analysis system. Regarding this, we extend our design to ensure the integrity of chained middlebox services.

We need to check the execution sequence of multiple middleboxes for each batch of packets besides the execution integrity of individual middleboxes. To achieve this, we employ a technique of hash chaining. That is, proofs generated from one individual MB on the chain encode the proofs of the former MB via a cryptographic hash function as follows:

$$Proof_{t+1}^{pkt} = \mathsf{Hash}(\{n_i\}||\{\{p_j\}_i\}||Proof_t^{pkt}), \qquad (10)$$

where $Proof_{t+1}^{pkt}$ means the proof generated by MB_{t+1} for a packet pkt, $Proof_t^{pkt}$ is the proof of MB_t. To facilitate the above computation, $Proof_t^{pkt}$ is embedded in the header of the packet p as shown in Fig. 3. These packet-level proofs are used to update the corresponding flow-level proofs, e.g., $Proof_{t+1}^{f}$. Finally, the commitment, e.g., ζ_{t+1}, is generated by the middlebox service, e.g., MB_{t+1}, via all proof-level proofs (e.g., $Proof_{t+1}^{f}(\cdot)$) in a batch as described before.

When the trusted third party is going to verify whether a middlebox chain faithfully runs, it sequentially checks proofs for each individual middlebox. For one middlebox and one sampled flow, e.g., MB_{t+1} and f, it performs the verification as illustrated in Sec. V-C.3. The only difference is that, in step three, it constructs the proof references for each packet based on the matched patterns, the nodes in the traversal path on the AC automaton of MB_{t+1}, and the proof references generated by the previous middlebox, e.g., MB_t. Therefore, we can ensure that the packets are correctly processed by the expected MB chain.

Remark: In the paper, we focus on verifying the execution integrity of the AC algorithm for remote DPI services, since it is the most popular string-matching function and has been applied in many network applications like intrusion detection systems (Snort [1] and Suricata [2]) and Web application firewalls (Modsecurity [3] and CloudFlare's WAF [4]). We notice that several research efforts have been attempted to optimize the performance of string pattern matching for network applications [44], [45]. For example, DFC [44] is a memory-efficient and cache-friendly data structure to deliver high-performance string pattern matching. Hyperscan [45] devises a fast multi-pattern regex matcher with graph-based regular expression decomposition. How to deal with these efficient pattern-matching algorithms in DPI verification is an interesting topic in the future.

VI. Security Analysis

We theoretically analyze how the proposed scheme defeats both the untrusted cloud operators and the honest-but-curious enterprise customers.

Security against untrusted cloud providers. We analyze the *correctness* and then the *soundness* of the scheme. The correctness of the scheme follows the correctness of the authenticated AC automaton. If the remote DPI service faithfully inspects the packet payloads over the authenticated AC automaton, it can derive the desired output patterns and traversal paths on the trie for each packet. Thus, it can construct the expected packet-level proofs and further produce the correct flow-level proofs. Moreover, if the DPI services honestly process all network flows in a batch, it is able to build the intended Merkle tree and produce the valid auxiliary proofs for the delayed verification requests.

In turn, the correctness of the proof is guaranteed as follows: 1) HMAC-based signatures ensure the authenticity of nodes in AC automaton since the HMAC function is performed by the SGX-based third party and is collision-resistant; 2) verification of the traversal path on the trie according to the flow's payloads guarantees the expected inspection of DPI services; 3) the integrity of matched patterns is ensured via the collision-resistant accumulators of nodes in the detailed proof; 4) the consistency between the derived flow-level proof with the value of the leaf node of the Merkle tree and the correctness of the commitment guarantee that packet processing is completed at run-time.

The soundness of our scheme is said by detecting adversarial behaviors with high probability. It is determined by the sampling probability and the security of the cryptographic hash function. In the scheme, for a sampled flow, the cloud provider has to ensure that the list of nodes in the detailed proof is consistent with the payloads of the requested flow. Otherwise, the scheme directly rejects the incorrect proofs. All other verification processes (including checking signatures of nodes, accumulators, and commitment) rely on the security of the cryptographic hash function.

In terms of the integrity of processing all packets, we analyze the soundness error of our scheme incurred by the sampling-based verification strategy. We consider a strong adversary that is aware of our verification strategy and adaptively processes a portion of flows in a batch. Assume that the ratio of the number of network flows correctly processed by the in-the-cloud DPI device in a batch is γ. For any sampled flow, the probability of obtaining the correct proof is $(\gamma + (1-\gamma)q)$, where q is the probability of successfully

guessing the proof. Thus, for a batch including f network flows, the probability of the adversary evading the verification is:

$$Pr_{evade} = (\gamma + (1-\gamma)q)^{f \cdot p}, \tag{11}$$

where p is the sampling probability of the enterprise customer. Obliviously, the soundness error (i.e., Pr_{evade}) is tunable , which is negatively related to the sampling probability p and positively related to the honest ratio γ and the successfully guessing probability q. Take an example to understand the soundness of the proposed scheme where 80% of flows are faithfully processed , and q is 10%. Only sampling 35 flows in a batch with any size can guarantee that the soundness error is about 0.1%.

Security against honest-but-curious enterprise customers. Our scheme successfully preserves the privacy of in-the-cloud DPI rulesets, because only commitment messages (from the cloud) and boolean results (from the third party) are sent to the customers. Other ruleset-related information is protected in an end-to-end manner. Namely, the in-transmission data privacy is ensured via encrypted channels, meanwhile, the in-use and at-rest data privacy is guaranteed via the SGX-based third party.

VII. Implementation and Evaluation

A. Implementation and Experimental Setup

We implement a prototype system including an SGX-based third party, an enterprise gateway, and a DPI function. Both the gateway and the DPI function are written in C++ and built as Click instances [46] with customized elements based on the Crypto++ library. All enclave functions of the SGX-based third party are implemented in C++ with the Crypto++ library. The communication channels among the above three entities are built via TCP sockets.

We deploy the enterprise gateway and the DPI function in the AWS EC2 t2xlarge instances in Ubuntu Server 16.04 LTS (4 vcpus and 16 GB memory) in the same region. Meanwhile, all enclave functions of the third party are developed using Intel SGX SDK 2.6 and run on an Intel Core i5-8500 Skylake 3.0GHz CPU with 16GB RAM and SGX enabled. All C++ algorithms are compiled in the same optimization level (-O2).

To evaluate the effectiveness and efficiency of the prototype, we conduct the following comparison experiments:

- *Baseline* denotes the original AC automaton without augmenting our verification-related functions.
- *VeriDPI* depicts the relevant functions for the AC automaton with the proposed verification scheme.

Besides, we use two open-source rulesets: Snort v3.0 [1] (abbreviated as *Snort*) and Emerging Threats Open[1] (abbreviated as *ET-Open*). *Snort* represents a small ruleset with 3459 patterns, while *ET-Open* is a large ruleset with 26763 patterns. Both real traffic[2] and synthetic traces are used. The latter traffic is generated with various packet sizes of 400B, 800B, and 1200B. Especially, since the performance of the DPI function is related to the content of packet payloads, we construct malicious traffic by selecting different percentages of packets (25%, 50%, 100%) from the synthetic network streams and inserting random pattern strings from rulesets into them. Besides, to facilitate the construction of the Merkle tree, we set up a batch of traffic with 1024 network flows.

[1]Emerging Threats: Open Source Signatures, online at https://rules.emergingthreats.net/open/snort-2.9.0/rules/

[2]m57-Patents Scenario: A public trace file released by an IDS system Bro, online at http://digitalcorpora.org/corpora/scenarios/m57-patents-scenario

TABLE III
Performance of Different Ruleset Evaluation

Ruleset	*Snort*	*ET-Open*
# of patterns	3459	26763
Evaluation latency (s)	0.05	0.49

TABLE IV
Performance of Different Ruleset Authentication

Overhead	Ruleset	Baseline w/o SGX	Baseline w/ SGX	VeriDPI w/o SGX	VeriDPI w/ SGX
Comp.(s)	*Snort*	0.04	0.05	0.13	0.19
	ET-Open	0.19	0.53	0.96	1.72
Storage (MB)	*Snort*	-	-	2	2
	ET-Open	-	-	21	21

B. Performance of Ruleset Evaluation and Authentication

We estimate the performance of ruleset evaluation and authentication for both *Snort* and *ET-Open*. Since the computation overhead of the hash function is positively related to the input size, directly concatenating and hashing all the contents of the ruleset may introduce significant overhead. Therefore, we perform per-line compression of the ruleset contents before hashing rules, similar to [47].

As shown in Table III, the latency of ruleset evaluation increases with the size of the ruleset. This is because the hashing time grows linearly with the input size. While the evaluation latency for the *ET-Open* ruleset is 0.49 seconds, such pre-processing overhead is one-time and would not affect the real-time performance of DPI services.

As for the performance of ruleset authentication, we compare both the original AC trie construction algorithm (as the baseline) and the ruleset authentication algorithm (i.e., Algorithm 2). Both of them are performed outside and inside the enclave. We measure the authentication performance of various rulesets and compare it with the baseline. As shown in Table IV, the latency of the AC automaton setup increases with the size of the rulesets. Compared with the baseline, AC automaton authentication in the proposed scheme introduces extra computation overhead due to the generation of cryptographic accumulators and signatures, resulting in the additional latency of 0.09s for *Snort* and 0.77s for *ET-Open*. Besides, the latency of both the baseline and our scheme grows when running them in the enclaves. For a large ruleset *ET-Open*, the proposed scheme takes about 1.72s to authenticate the ruleset inside the enclave. Such overhead is acceptable since these

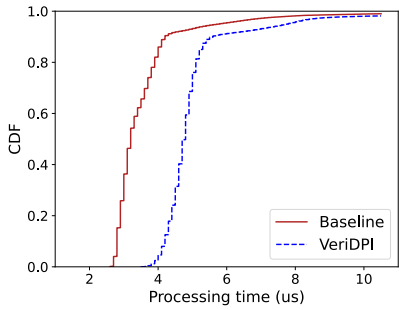


Fig. 5. Processing latency at the gateway without and with our scheme for real trace.

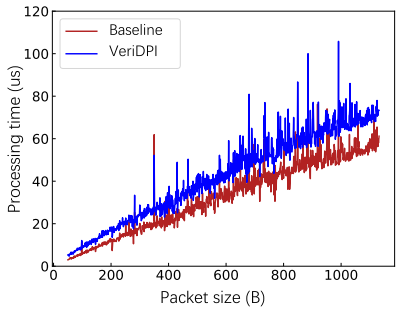


Fig. 6. Processing latency of DPI services for real traffic using the Snort ruleset.

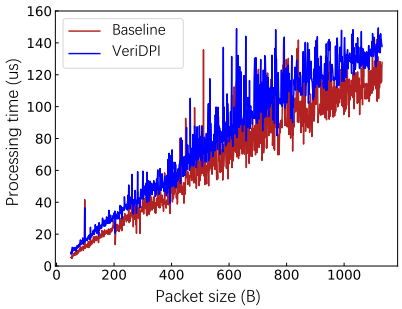


Fig. 7. Processing latency of DPI services for real traffic using the ET-Open ruleset.

operations incur one-time overhead and would not affect the real-time packet processing capability of DPI functions.

In addition, we evaluate the extra storage space introduced by the AC authentication scheme. From Table IV, we can see that the additional storage overhead increases with the size of the rulesets. As expected, the additional overhead is positively proportional to the number of nodes in the AC automaton. For the rulesets *Snort* and *ET-Open*, the extra overhead is 2MB and 21MB, respectively.

C. Gateway Performance

In our scheme, GW simply preprocesses packets. It embeds customized identifiers in packets and stores sampled ones. We evaluate both processing delay and storage overhead introduced at GW for a batch of packets. Figure 5 shows that over 95% of packets experience less than 9μs in GW. Meanwhile, our scheme increases the packet processing latency at the gateway by about 2μs compared with the latency for merely packet forwarding (i.e., baseline). Besides, it is marginal compared with the end-to-end transmission latency (see Fig. 13 and Fig. 14).

In addition, the storage overhead at the gateway is insignificant, as it only keeps a small portion of the traffic. When experimenting with the real traffic, the gateway randomly caches 35 network flows in order to verify the integrity of DPI execution with a 0.1% soundness error. For a batch of the traffic of 8.52MB, the storage overhead of the gateway is only 120KB, accounting for 1.5% of the total traffic size.

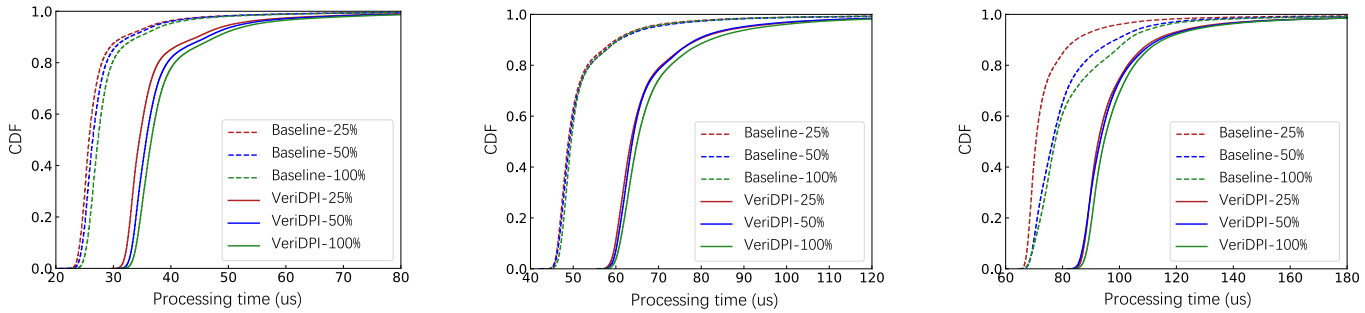
D. Performance of Verifiable in-the-Cloud DPI Services

We assess the impact of the proposed verification scheme on the performance of in-the-cloud DPI services in terms of processing latency, throughput, end-to-end latency, and storage costs. We set the query performance of the original AC automaton as the baseline.

Processing latency. We first experimentally evaluate the impact of our scheme on the packet processing latency of DPI functions with varied packet sizes and different malicious traffic ratios. Figure 8 and Figure 9 describe the processing latency of DPI functions using the rulesets *Snort* and *ET-Open*, respectively. We can see that the processing latency increases as the percentage of malicious traffic increases. The reason for the phenomenon is that when the packet payloads contain malicious pattern strings, the DPI service conducts more operations, such as retrieving matched pattern strings via *suffixP* pointers and generating proofs for them. Meanwhile, the processing latency gradually grows with the increasing size of packets. When using the *ET-Open* ruleset, the packet processing latency is much longer than that based on the *Snort* ruleset. This is because the *ET-Open* ruleset would match more packet payloads which lead to longer processing latency. In addition to the aforementioned trends, the proposed scheme introduces an additional processing delay of about $10 \sim 20\mu$s. The delay of the microsecond level is basically negligible compared with the packet transmission delay of the millisecond level in the whole link.

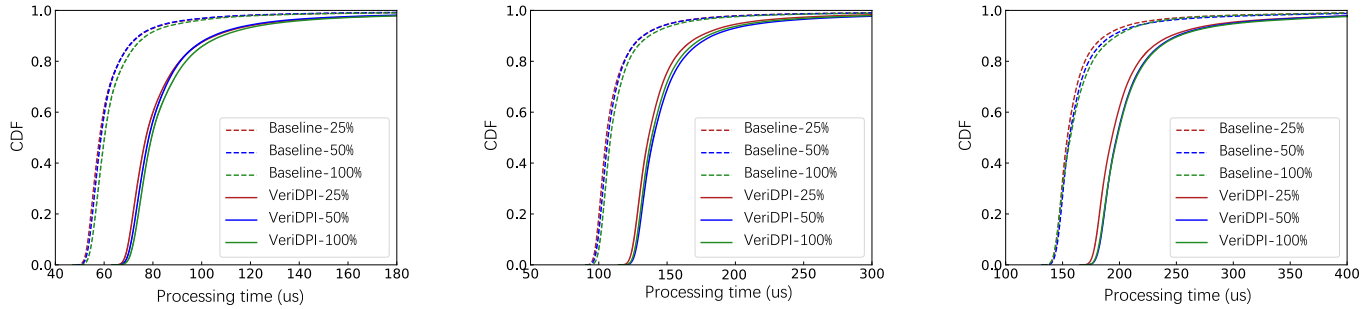
We also evaluate the performance of DPI functions on real traffic. Figure 6 and Figure 7 show the results with *Snort* and *ET-Open* installed, respectively. Similar to the synthetic traffic, the processing latency for real traffic increases as the packet size grows. Especially, the processing latency on real traffic is less than 10μs on average. The possible reason is that the real traffic generally does not match malicious pattern strings in the ruleset and thus introduces less computational overhead.

Throughput. To further estimate the impact of our scheme on DPI processing capacity, we measure the throughput of the DPI services without and with the proposed scheme on both synthetic and real traffic. Figure 10 depicts the throughput of the DPI functions with the *Snort* ruleset installed when processing different sizes of packets with different malicious traffic ratios. The results illustrate that the throughput of the DPI functions decreases slightly as the percentage of malicious traffic increases. At the same time, the throughput of the DPI services increases slightly with the growing size of packet payloads. Especially, Figure 10 (b) shows that our scheme



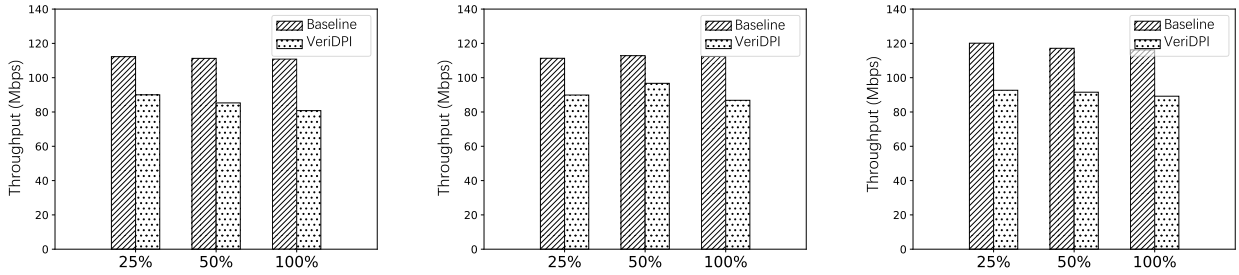
(a) Synthetic traffic of 400B (b) Synthetic traffic of 800B (c) Synthetic traffic of 1200B

Fig. 8. Processing time of DPI services without and with our scheme for varied synthetic traffic (using *Snort*).



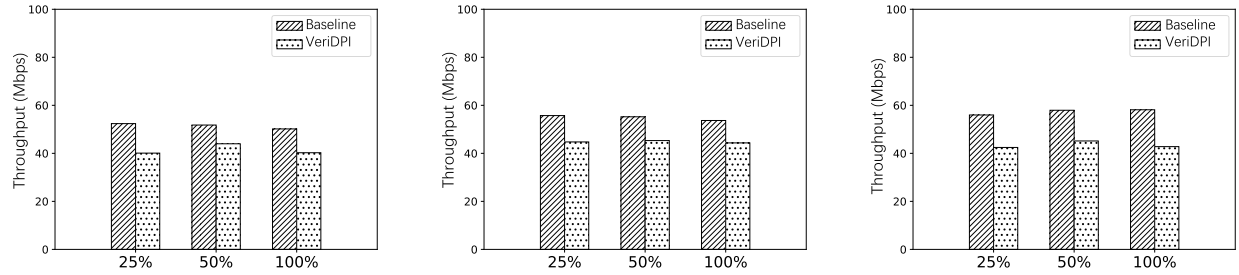
(a) Synthetic traffic of 400B (b) Synthetic traffic of 800B (c) Synthetic traffic of 1200B

Fig. 9. Processing time of DPI services without and with our scheme for varied synthetic traffic (using *ET-Open*).



(a) Synthetic traffic of 400B (b) Synthetic traffic of 800B (c) Synthetic traffic of 1200B

Fig. 10. Throughput of DPI services without and with our scheme for varied synthetic traffic (using *Snort*).



(a) Synthetic traffic of 400B (b) Synthetic traffic of 800B (c) Synthetic traffic of 1200B

Fig. 11. Throughput of DPI services without and with our scheme for varied synthetic traffic (using *ET-Open*).

has less impact on the throughput of the DPI services when processing traffic with 50% malicious flows. The possible reason is that, compared with the other two types of synthetic traffic, this synthetic traffic matches fewer strings or matches shorter strings, thus increasing the efficiency of the DPI services. Figure 11 depicts the throughput of DPI functions with the *ET-Open* ruleset installed under various synthetic traffic. The results are similar to those with the *Snort* ruleset. Comparing the *Snort* ruleset-based DPI function with the *ET-Open* ruleset-based DPI function, we find that the throughput

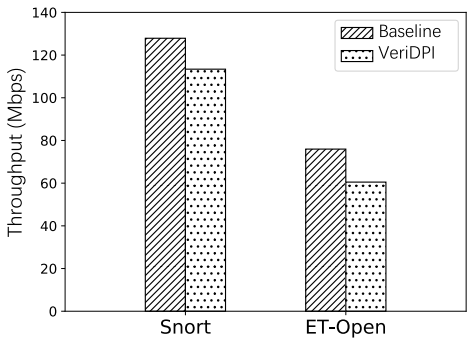


Fig. 12. Throughput of DPI services without and with our scheme for real traffic.

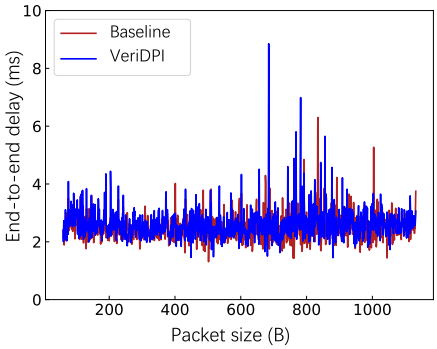


Fig. 13. End-to-end transmission delay of DPI services for real traffic with *Snort* installed.

of the latter is much smaller than that of the former. The reason is that inspecting packets based on the *ET-Open* ruleset may match more pattern strings due to the larger ruleset size, which would introduce more operations (such as string copy) and thus reduce the DPI throughput. On average, our scheme degrades the DPI throughout by about $15\% \sim 25\%$ when the *Snort* or *ET-Open* ruleset is installed.

In addition, the impact of our scheme on the throughput of DPI services for real traffic is experimentally evaluated , and the results are presented in Figure 12. The throughput degradation introduced by our scheme is about 11.3% and 20.3% with *Snort* and *ET-Open* installed, respectively. From the above experimental results, our scheme inevitably produces a negative impact on the throughput of DPI services. To achieve similar throughput performance to the baseline, enterprise customers can deploy more DPI instances to handle massive network flows. While such a solution introduces additional overhead (due to buying more instances), a recent study [5] demonstrated that even deploying twice as many DPI instances in the cloud would result in much less deployment and maintenance overhead than the original in-the-house deployment.

End-to-end delay. To demonstrate the overall performance of our scheme in the network environment, we measure the end-to-end delay, i.e., the time taken by a packet to traverse from the GW to the DPI function and finally back to the GW. All packets are sent by the GW at the rate of 10Mbps. Figure 13 reports the end-to-end transmission delay of real traffic in the network when the *Snort* ruleset is deployed in the

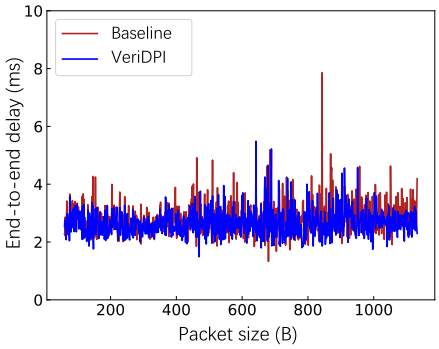


Fig. 14. End-to-end transmission delay of DPI services for real traffic with *ET-Open* installed.

DPI function. From the figure, it can be seen that our scheme does not significantly affect the end-to-end transmission delay. Meanwhile, the packet size also negligibly affects the end-to-end transmission delay. Figure 14 depicts the end-to-end transmission delay of DPI services for real traffic with the *ET-Open* ruleset. The experimental results are similar to those with *Snort*. Therefore, we can conclude that the proposed scheme is lightweight and the introduced processing latency of DPI functions would not significantly affect the packet transmission in the network.

Storage costs. To facilitate verification, in-the-cloud DPI services not only store authenticated AC automaton, but also cache the flow-level DPI execution proofs or the Merkle tree for an unverified batch of traffic. Since the former is evaluated in Sec. VII-B, we assess the latter storage costs here. More specifically, we study the space required for DPI verification of one batch. In theory, the maximum storage overhead is $\max(x \cdot |Proof^f|, 32 \cdot (2x-1))$, because the cached execution proofs would be removed once the tree is constructed. Note that x is the number of network flows in a batch, and $|Proof^f|$ is the size of one flow-level DPI execution proof. In the real setting with a batch including 1024 flows, we set $|Proof^f|$ as 36B equal to the accumulation of 32-byte packet-level proofs multiplying 4-byte randomness. The maximum storage overhead introduced by our scheme is about 64KB which is dominated by the storage associated with the Merkle tree. Supposing that there are 1000 batches concurrently verified, the storage overhead is near to only 64MB in the worst case.

E. Communication Overhead

Our scheme introduces both the in-band communication overhead and the out-of-band communication overhead, respectively. The in-band overhead is mainly introduced by the 290-bit customized headers for verification, including a 16-bit *BatchID* field, a 16-bit *FlowID* field, a 2-bit *Type* field, and a 256-bit $Proof^{pkt}$ field. The overhead is only relevant to the number of packets transmitted in the network and is negligible compared to the total size of the real traffic. For example, in a network link with a bandwidth of 1 Gbps, the number of packets transmitted per second is about 8.3×10^4 (each packet size is denoted as 1500B). The additional in-band communication overhead is about 2.42% of the bandwidth of the total link.

The out-of-band communication overhead introduced by the proposed scheme mainly consists of the end-of-batch identification packets *BatchEND*, the DPI execution commitment ζ, and the DPI execution proof for each stream π. In the experiment, we set the number of network flows in a batch as 1024 and the number of sampled flows as 35. When replaying the real traffic, the total size of a batch of traffic is 8.52MB , and the total size of the sampled flows' payloads is 120KB. The out-of-band communication overhead for verification is about 8.28MB, approximately equal to the traffic size of the batch. Note that the theoretical analysis in Section VI yields that the soundness error ϵ is only related to the number of sampled flows in a batch. Therefore, we can make each batch contains more network streams in practice, so as to reduce the proportion of the out-of-band communication overhead. For example, when a batch contains 10^4 network flows, the out-of-band communication overhead is about 1% of the total traffic size of the batch.

VIII. Conclusion

This paper presents a privacy-preserving and lightweight verification scheme to ensure the correct execution of DPI services in clouds. To protect the privacy of remote DPI rulesets against enterprise customers, we employ a trusted third party to perform privacy-preserving and trustworthy ruleset evaluation and DPI verification. Meanwhile, we devise a lightweight DPI ruleset authentication method to achieve fast proof generation and efficient verification without requiring the verifier to always maintain the entire rulesets. Moreover, we develop a commitment-based secure sampling scheme to further reduce verification overhead while defeating the cheating behaviors of cloud providers. In the end, we implement a prototype using Click modules. Extensive experimental results with real traces demonstrate the feasibility and efficiency of our scheme.

In the future, we will explore the emerging blockchain technology for building a decentralized trusted third party [48], [49], [50]. One key challenge is how to address privacy concerns raised by the transparency characteristic of the blockchain.

References

[1] (1998). *Snort: Open-Source Intrusion Detection and Prevention System*. [Online]. Available: https://www.snort.org/

[2] (2009). *Suricata: High Performance, Open Source Network Analysis and Threat Detection Software*. [Online]. Available: https://suricata.io/

[3] (2021). *Modsecurity: Open-Source Web Application Firewall*. [Online]. Available: https://github.com/SpiderLabs/ModSecurity

[4] C. Ueland. (2021). *Scaling Cloudflare's Massive WAF*. [Online]. Available: https://www.cloudflare.com/waf/

[5] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proc. ACM SIGCOMM*, 2012, pp. 13–24.

[6] C. Wang, X. Yuan, Y. Cui, and K. Ren, "Toward secure outsourced middlebox services: Practices, challenges, and beyond," *IEEE Netw.*, vol. 32, no. 1, pp. 166–171, Jan. 2018.

[7] Research and Markets. (2019). *World Deep Packet Inspection Market—Opportunities and Forecasts 2017–2023*. [Online]. Available: https://www.researchandmarkets.com/reports/3773051/world-deep-packet-inspection-market/

[8] European Telecommunications Standards Institute (ETSI). (2012). *Network Functions Virtualisation*. [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[9] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu, "Embark: Securely outsourcing middleboxes to the cloud," in *Proc. NSDI*, 2016, pp. 255–273.

[10] China Telecom. (2022). *Protect Your Network From DDoS Attacks With Proactive Real-Time Traffic Mitigation*. [Online]. Available: https://www.ctamericas.com/products-services/internet/anti-ddos/

[11] X. Yuan, H. Duan, and C. Wang, "Assuring string pattern matching in outsourced middleboxes," *IEEE/ACM Trans. Netw.*, vol. 26, no. 3, pp. 1362–1375, Jun. 2018.

[12] X. Zhang, H. Duan, C. Wang, Q. Li, and J. Wu, "Towards verifiable performance measurement over in-the-cloud middleboxes," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2019, pp. 1162–1170.

[13] C. Zeng, F. Liu, S. Chen, W. Jiang, and M. Li, "Demystifying the performance interference of co-located virtual network functions," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2018, pp. 765–773.

[14] N. Shalom. (2019). *Amazon Found Every 100 ms of Latency Cost Them 1% in Sales*. [Online]. Available: https://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/

[15] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," 2018. [Online]. Available: https://eprint.iacr.org/2018/046

[16] D. Papadopoulos, C. Papamanthou, R. Tamassia, and N. Triandopoulos, "Practical authenticated pattern matching with optimal proof size," *Proc. VLDB Endowment*, vol. 8, no. 7, pp. 750–761, 2015.

[17] Z. Zhou, T. Zhang, S. S. M. Chow, Y. Zhang, and K. Zhang, "Efficient authenticated multi-pattern matching," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, May 2016, pp. 593–604.

[18] H. Duan, C. Wang, X. Yuan, Y. Zhou, Q. Wang, and K. Ren, "LightBox: Full-stack protected stateful middlebox at lightning speed," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 2351–2367.

[19] R. Poddar, C. Lan, R. A. Popa, and S. Ratnasamy, "SafeBricks: Shielding network functions in the cloud," in *Proc. NSDI*, 2018, pp. 201–216.

[20] R. del Pino and S. Katsumata, "A new framework for more efficient round-optimal lattice-based (partially) blind signature via trapdoor sampling," in *Proc. CRYPTO*. Cham, Switzerland: Springer, 2022.

[21] J. Doerner, Y. Kondi, E. Lee, and L. Tyner, "Threshold BBS+ signatures for distributed anonymous credential issuance," in *Proc. IEEE SP*, 2023, pp. 773–789.

[22] Intel Corporation. (2014). *Intel Software Guard Extensions Programming Reference*. [Online]. Available: https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf

[23] X. Zhang, Q. Li, Z. Zhang, J. Wu, and J. Yang, "VSFC: Generic and agile verification of service function chains in the cloud," *IEEE/ACM Trans. Netw.*, vol. 29, no. 1, pp. 78–91, Feb. 2021.

[24] A. Panda, O. Lahav, K. J. Argyraki, M. Sagiv, and S. Shenker, "Verifying reachability in networks with mutable datapaths," in *Proc. NSDI*, 2017, pp. 699–718.

[25] S. K. Fayaz, T. Yu, Y. Tobioka, S. Chaki, and V. Sekar, "Buzz: Testing context-dependent policies in stateful networks," in *Proc. NSDI*, 2016, pp. 275–289.

[26] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "BlindBox: Deep packet inspection over encrypted traffic," in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015, pp. 213–226.

[27] J. Ning, G. S. Poh, J.-C. Loh, J. Chia, and E.-C. Chang, "PrivDPI: Privacy-preserving encrypted traffic inspection with reusable obfuscated rules," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1657–1670.

[28] R. A. Nafea and M. Amin Almaiah, "Cyber security threats in cloud: Literature review," in *Proc. Int. Conf. Inf. Technol. (ICIT)*, Jul. 2021, pp. 779–786.

[29] M. A. Almaiah, A. Al-Zahrani, O. Almomani, and A. K. Alhwaitat, "Classification of cyber security threats on mobile devices and applications," in *Artificial Intelligence and Blockchain for Future Cybersecurity Applications*. Berlin, Germany: Switzerland: Springer, 2021, pp. 107–123.

[30] M. A. Almaiah, Z. Dawahdeh, O. Almomani, A. Alsaaidah, A. Al-Khasawneh, and S. Khawatreh, "A new hybrid text encryption approach over mobile ad hoc network," *Int. J. Electr. Comput. Eng.*, vol. 10, no. 6, pp. 6461–6471, 2020.

[31] M. Adil et al., "MAC-AODV based mutual authentication scheme for constraint oriented networks," *IEEE Access*, vol. 8, pp. 44459–44469, 2020.

[32] A. K. A. Hwaitat et al., "Improved security particle swarm optimization (PSO) algorithm to detect radio jamming attacks in mobile networks," *Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 4, 2020.

[33] S. T. Setty, R. McPherson, A. J. Blumberg, and M. Walfish, "Making argument systems for outsourced computation practical (sometimes)," in *Proc. ISOC NDSS*, 2012, pp. 1–17.
[34] A. V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," *Commun. ACM*, vol. 18, no. 6, pp. 333–340, Jun. 1975.
[35] N. Fazio and A. Nicolosi, "Cryptographic accumulators: Definitions, constructions and applications," New York Univ., 2002. [Online]. Available: https://www.cs.nyu.edu/nicolosi/papers/accumulators.pdf
[36] F. Baldimtsi et al., "Accumulators with applications to anonymity-preserving revocation," in *Proc. IEEE Eur. Symp. Secur. Privacy*, Apr. 2017, pp. 301–315.
[37] T. Dryja, "Utreexo: A dynamic hash-based accumulator optimized for the bitcoin UTXO set," 2019. [Online]. Available: https://eprint.iacr.org/2019/611
[38] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *Proc. Annu. Int. Cryptol. Conf.*, 2002, pp. 61–76.
[39] L. Nguyen, "Accumulators from bilinear pairings and applications," in *Proc. Cryptographers' Track RSA Conf.*, 2005, pp. 275–292.
[40] K. Argyraki, P. Maniatis, and A. Singla, "Verifiable network-performance measurements," in *Proc. 6th Int. Conf.*, Nov. 2010, pp. 1–12.
[41] P. Kocher et al., "Spectre attacks: Exploiting speculative execution," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 1–19.
[42] A. Ahmad, B. Joe, Y. Xiao, Y. Zhang, I. Shin, and B. Lee, "OBFUSCURO: A commodity obfuscation engine on Intel SGX," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2019, pp. 1–15.
[43] R. C. Merkle, "A certified digital signature," in *Proc. CRYPTO*, 1989, pp. 218–238.
[44] B. Choi, J. Chae, M. Jamshed, K. Park, and D. Han, "DFC: Accelerating string pattern matching for network applications," in *Proc. NSDI*, 2016, pp. 551–565.
[45] X. Wang et al., "Hyperscan: A fast multi-pattern regex matcher for modern CPUs," in *Proc. NSDI*, 2019, pp. 631–648.
[46] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, Aug. 2000.
[47] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. ASIACRYPT*, 2008, pp. 90–107.
[48] M. A. Almaiah, "A new scheme for detecting malicious attacks in wireless sensor networks based on blockchain technology," in *Artificial Intelligence and Blockchain for Future Cybersecurity Applications*. Berlin, Germany: Springer, 2021, pp. 217–234.
[49] A. Ali et al., "An industrial IoT-based blockchain-enabled secure searchable encryption approach for healthcare systems using neural network," *Sensors*, vol. 22, no. 2, p. 572, Jan. 2022.
[50] A. I. Siam et al., "Secure health monitoring communication systems based on IoT and cloud computing for medical emergency applications," *Comput. Intell. Neurosci.*, vol. 2021, pp. 1–23, Dec. 2021.

Xiaoli Zhang received the Ph.D. degree from Tsinghua University in 2020 under the supervision of Prof. Jianping Wu. Currently, she is a Lecturer with the College of Computer Science, Zhejiang University of Technology. Her research interests include trusted computing, verifiable computation, cloud security, and network security.

Wei Geng received the B.Eng. degree from Tsinghua University in 2020, where he is currently pursuing the M.E. degree with the Institute of Network Science and Cyberspace. His research interests include the IoT security and cloud security.

Yiqiao Song is currently pursuing the M.E. degree with the Zhejiang University of Technology, Hangzhou, China. Her research interests include blockchain, cryptography, and information security.

Hongbing Cheng (Member, IEEE) received the Ph.D. degree from the Nanjing University of Posts and Telecommunications. He is currently a Professor with the College of Computer Science, Zhejiang University of Technology. He has published numerous research papers in high-quality international journals and conferences. His research interests include blockchain, cryptography, privacy-preserving and information security, computer communications, and cloud computing security. He served as an invited editor for several international journals and conferences. He has been invited to give keynote speeches and chairs committee, where he reviewed papers for many international journals and conferences.

Ke Xu (Senior Member, IEEE) received the Ph.D. degree from Tsinghua University, Beijing, China. He is currently a Full Professor with the Department of Computer Science, Tsinghua University. He has published more than 200 technical articles and holds 11 U.S. patents. His research interests include next generation internet, blockchain systems, the Internet of Things, and network security. He is a member of ACM. He serves as the Steering Committee Chair for IEEE/ACM IWQoS. He has guest-edited several special issues for IEEE and Springer journals. He is an Editor of IEEE INTERNET OF THINGS JOURNAL.

Qi Li (Senior Member, IEEE) received the Ph.D. degree from Tsinghua University. He is currently an Associate Professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include internet and cloud security, mobile security, and big data security. He is an Editorial Board Member of the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING and *Digital Threats: Research and Practice* (ACM).