Atomos: Constant-Size Path Validation Proof

Anxiao He, Kai Bu* *Member, IEEE,* Yucong Li, Eikoh Chida, Qianping Gu, and Kui Ren *Fellow, IEEE*

***Abstract*—Path validation has been explored as an indispensable security feature for the future Internet. Motivated by the Path-Aware Networking Research Group (PANRG) under the Internet Engineering Task Force (IETF) and Internet Research Task Force (IRTF), it gives end-hosts more control over packet forwarding and ensures that the forwarding history is verifiable. The main idea is to require that routers add proofs in packet headers for other routers to verify. We identify linear-scale proofs as the essential efficiency barrier of existing path validation solutions. In this paper, we propose Atomos to validate network paths with constant-size proofs. To this end, we construct a noncommutative homomorphic asymmetric-key encryption scheme. Asymmetric cryptography minimizes the number of proofs needed and saves time in processing proofs. The homomorphism we design yields constant-size proofs. It limits the header-space overhead and outperforms existing linear-scale counterparts when the path length exceeds a value that is usually small. Furthermore, the proposed encryption scheme is noncommutative so that any deviation from the forwarding path can be detected. We explore a series of design strategies for security and efficiency. The evaluation results show that Atomos yields not only shorter proofs but also faster validation than existing solutions.**

***Index Terms*—Path validation, constant-size proof, homomorphic encryption.**

I. Introduction

Motivated by the Path-Aware Networking Research Group (PANRG) [1], [2] under the Internet Engineering Task Force (IETF) and Internet Research Task Force (IRTF), path validation has been explored as an indispensable security feature for the future Internet [3]–[7]. Path validation makes the forwarding history of packets verifiably visible to both end-hosts and routers. In the current Internet, end-hosts have no control over packet forwarding. Once a packet enters the Internet, its forwarding process is agnostic to end-hosts. A router may not know any other routers that have processed the packet besides the one from its previous hop. Such invisibility in packet forwarding can be exploited to degrade service quality and security. For example, some Internet service providers (ISPs) collaborate on traffic forwarding [8]. Consider, for example, a case in which a customer signs up with ISP1 with a higher service fee and a higher expected service quality than ISP2. If ISP1 transmits the customer's traffic via the slower ISP2, the customer may experience worse performance than he agreed to. This performance degradation is, however, unknown to the customer because of the invisibility of traffic forwarding. Furthermore, such forwarding invisibility may make a security breach unnoticeable. For example, service requests made to the provider usually go through a series of security checks [9], [10]. Only traffic verified as genuine is expected to reach the provider. However, if attacking traffic deviates from the expected forwarding path and circumvents security checks, it may create various security issues for the provider (e.g., DDoS attacks [10]). With path validation, end-hosts are empowered to enforce forwarding preferences along specific paths. Network entities, including end-hosts and routers, can also verify whether packets have followed the specified forwarding path. In this way, path validation offers a verifiable method of protecting service quality and security.

Path validation introduces both enforcement and verification for packet forwarding [3], [11]. Unlike the current Internet, the path-validation-augmented Internet allows end-hosts to select specific forwarding paths for specific traffic for quality or security purposes. For example, end-hosts can select paths with less frequent traffic congestion and packet loss in order to achieve high quality and reliability. They can also select paths without censorship or filtering to protect communication security and privacy [12]. Enforcing forwarding preferences follows a similar strategy as source routing. The source either includes the specified path in packet headers [3] or passes it to en-route routers at the time of session creation [4]. To verify whether a packet has followed the specified path, each router should add its proof to the packet header. The proof is generated using cryptography to prevent forging. Each router should prove itself to all its downstream routers. Since existing path validation solutions use symmetric-key encryption for speed, a straightforward design of such pairwise proof methods introduces $\mathcal{O}(n^2)$ proof fields in the packet header for an n-hop forwarding path [11].

Along with security, efficiency in terms of proof length and processing speed remains a major design goal. These two performance metrics are proportional. That is, given a path validation protocol, the fewer proof fields it introduces into a packet header, the faster a router can process the packet. For example, ICING [3] uses the aggregate message authentication code (MAC) technique [13] to condense all proofs to the same router as a single proof field. This decreases the number of proof fields from $\mathcal{O}(n^2)$ [11] to $\mathcal{O}(n)$. A root cause of the high overhead of ICING is its strong robustness against malicious nodes. Specifically, ICING assumes no trust between any pair of nodes [3]. When a stronger security assumption is used, less computation is necessary. For example, origin and path trace (OPT) [4] uses a dynamically recreatable key (DRKey) technique and lets a trusted source precompute critical information that each en-route router needs for path validation.

A. He, K. Bu*, Y. Li, and K. Ren are with the College of Computer Science and Technology, School of Cyber Science and Technology, Zhejiang University, Hangzhou 310027, China. E-mail: {zjuhax, kaibu, yucongli, kuiren}@zju.edu.cn

E. Chida is with the Department of Engineering for Future Innovation, National Institute of Technology, Ichinoseki College, Ichinoseki 021–8511, Japan. E-mail: chida@ichinoseki.ac.jp

Q. Gu is with the School of Computer Science, Simon Fraser University, Burnaby, British Columbia V5A 1S6, Canada. E-mail: qgu@sfu.ca

*Corresponding Author: Kai Bu.

EDICS: NET-SPRO Security protocols.

OPT not only shortens each of the $\mathcal{O}(n)$ proof fields but also speeds up the verification process to only $\mathcal{O}(1)$ computations. Following the same security assumptions as OPT, orthogonal sequence verification (OSV) also assumes a trusted source and uses orthogonal matrices as cryptographic keys to promote higher efficiency [14], [15]. To further accelerate path validation, probabilistic packet validation (PPV) [16] enables routers to probabilistically tag packets with proofs. Each packet is tagged by at most two routers. PPV, however, cannot guarantee forwarding enforcement at the granularity of packets. It requires that coflow packets jointly demonstrate the forwarding behavior of en-router routers. In this paper, we focus on a generic framework for path validation methods requiring pairwise trust and packet granularity, such as ICING.

We observe several limitations that prevent an increase in path validation efficiency. The common reason for all of them lies in the use of verification proofs that are of linear size in terms of path length. First, as the path lengthens, its proof takes up more header space as well as network bandwidth. Including more proofs in the traffic means having less payload to transmit, given the fixed link capacity. It may therefore decrease network throughput. Second, since a longer proof takes more time to generate and verify, more packets tend to be queued on routers or even dropped after queues become full. Third, the symmetric-key encryption adopted by path validation solutions requires each router to generate a different proof for its downstream routers. This yields many more proofs than are necessary. Essentially, a router requires only one proof to prove itself to all other routers. Redundant proofs cost more time to compute (because of, e.g., additional generation, verification, and update steps) and lead to further slowdown.

In this paper, we present Atomos to boost path-validation efficiency using a constant-size validation proof. To this end, we construct a noncommutative homomorphic asymmetric-key encryption scheme. Using asymmetric cryptography, a router requires only one proof to be generated using its secret key. All other routers can verify the proof using the router's public key. This minimizes the number of proofs and saves time in processing proofs. We design an additive magma for aggregating proofs, a multiplicative magma for verifying proofs, and a homomorphic mapping from the additive magma to the multiplicative magma. All these yield a constant-size proof. It limits the header-space overhead and outperforms existing linear-scale counterparts when the path length exceeds a value that is usually small, e.g., 4 or 5 hops. Finally, the proposed encryption scheme is noncommutative so that any deviation from the forwarding path can be detected. We explore a series of design strategies for security and efficiency. For example, we adopt a hash function with a certain property that resists forging attacks.

Based on the observation that path validation uses cryptography to avoid forgeability rather than to protect confidentiality, we propose a dynamic rekeying technique to shorten proofs without sacrificing security. The evaluation results show that Atomos yields not only shorter proofs but also faster validation than existing solutions do.

In summary, we make the following major contributions:

- Identify linear-scale proofs as the essential barrier to increasing the efficiency of path validation. Linear-scale proofs based on symmetric cryptography increase both header space and processing time as path length increases (Section II).
- Propose a noncommutative homomorphic asymmetric-key encryption scheme that offers a constant-size proof (Section III).
- Design a path validation solution—Atomos—based on the proposed encryption scheme. We explore a series of design strategies (e.g., dynamic rekeying) to improve efficiency without sacrificing security (Section IV).
- Prove the security properties (e.g., proof unforgeability) of Atomos and discuss its nonsecurity goals compared with those of existing solutions (Section V).
- Implement Atomos using the Click router (Section VI) and validate its performance (Section VII). Compared with existing solutions with linear-scale proofs, Atomos yields not only shorter proofs but also faster validation.

II. Problem

In this section, we review the evolution of path validation solutions with the goal of promoting efficiency. Although the symmetric-key cryptography they adopt supports fast validation, it enforces pairwise trust among routers through pairwise-shared keys. A router needs a different key to prove itself to each of the other routers. This also necessitates a linear-scale validation proof in terms of path length. A long path therefore induces a long validation proof that costs more packet-processing time and network bandwidth. We propose addressing all these limitations using the constant-size proof in Section III.

A. Changes to Internet Architecture

Path validation necessitates a series of changes to the current Internet architecture. The main entities of interest are the source, destination, and routers. In the current Internet, the source simply sends packets to the destination, and the routers simply follow the longest-prefix match to forward the packets to the destination. The current Internet simplifies the packet forwarding process for, at best, the source, destination, and routers. It ensures only that packets are forwarded to the correct destination rather than via a certain forwarding path. However, such simplicity prevents the end-hosts, such as the source and destination, from seeing the packet forwarding information. This leaves various forwarding misbehavior undetected, which may affect service quality, security, and privacy [3], [4], [12]. To empower end-hosts with forwarding visibility, path validation needs to change the working logic of end-hosts, routers, and the entire Internet architecture, as described below.
End-host change. Path validation requires the source and destination to 1) specify a forwarding path, 2) share keys with each other as well as with routers on the specified forwarding path, and 3) encapsulate packets with path-related proofs and verify such path proofs. Requirement 1 further requires a change to the current **Internet architecture**. That is, the Internet should be empowered to compute paths requested by end-hosts and authorize these paths to end-hosts and routers [3]. However, it is unlikely that there could be a centralized

authority with global knowledge of routing paths between any two end-hosts. ISPs can collectively achieve the function of such a centralized authority. Specifically, they can share their internal topology with other ISPs to compute all interdomain routing paths. Essentially, path validation should be provided to end-hosts as a service. Since all Internet services should be built upon the underlying Internet architecture, ISPs can install corresponding routing policies as they authorize certain forwarding paths to end-hosts. For requirement 2, end-hosts need to share keys among themselves as well with routers. The former ability can easily be supported in the current Internet via, for example, session keys. The latter, however, requires architectural updates, which may be performed in several ways. First, the centralized authority can directly share per-path shared keys for end-hosts and on-path routers. Second, end-hosts can individually communicate with on-path routers to exchange shared keys. Third, routers can derive keys shared with the source from packets on the fly [4]. Requirement 3 simply requires service clients on end-hosts to perform certain functions, and the computation capabilities needed, such as cryptography, are readily available.

Router change. At the heart of path validation implementation are routers' packet processing logics. Conventionally, routers simply match packet headers to forwarding entries in routing tables and follow the longest-prefix-match protocol. To support path validation, routers should perform more computation over packet headers. They need to correlate a packet with the specific forwarding path authorized to the packet. They should first verify the packet-carried proof generated by upstream on-path routers and then, if the verification passes, update the proof by incorporating their own proof. Proof verification and update require shared keys among routers; this may induce further changes to router communication. Routers would no longer follow the longest-prefix match to forward packets. Instead, the forwarding decision would already be dictated by the authorized forwarding path. All the previous packet processing logics should be updated in router software [4].

Deployment. It may be challenging to upgrade routers all at once to support path validation. We suggest a feasible method for incremental deployment. Given that path validation is considered a service, ISPs that opt for providing such a service can modify their routers accordingly. The modified routers would act as waypoints. However, conventional routers would still follow traditional longest-prefix matching for packet forwarding. Therefore, a necessary change is that the source should replace the destination with the first waypoint router in its packets. Similarly, the first waypoint router would set the destination of these packets to the second router. This is essentially a coarse-grained version of path validation, because only some routers along the forwarding path would enforce path validation. We consider the deployment of path validation to be beyond the scope of this paper and focus mainly on the design of path validation principles.

B. The Road to Efficient Path Validation

Path validation enables routers to verify whether a packet has transited the specified path in the correct order. This goal is achieved through two subgoals: enforcement and verification. Enforcement is similar to the idea of source routing [17], [18]. It allows the source to select forwarding paths for its traffic. Path validation solutions have two options to enforce a forwarding path between end-hosts. One is to embed the path information in packet headers [3]; the other is to provide the path information to en-route routers when a connection is created [4]. To verify whether a packet has followed the enforced path, each router should add its proof to the packet header. These proofs are used by subsequent routers to verify the packet's forwarding history. If it conforms to the enforced path, the packet is deemed valid. Otherwise, the packet may have traversed an unspecified path via untrusted routers. It should be dropped to protect service quality and security [3].

Given that any router may be compromised to circumvent path validation, initial solutions require pairwise trust among routers [3], [11]. They enforce per-hop verification in that every en-route router verifies whether a packet has visited all its upstream routers in the correct order. To enable verification at each hop, each en-route router should embed proofs in the packet header, each of which proves the router to a downstream router. Symmetric-key cryptography is adopted because of its speed. Every pair of routers exchanges a shared secret key, which is used to compute and verify proofs. A straightforward solution would yield $\mathcal{O}(n^2)$ proofs in the header of a packet along an n-hop path. A representative solution, ICING [3], leverages the aggregate MAC technique [13] to shorten proofs. It condenses all $\mathcal{O}(n^2)$ pairwise proofs into $\mathcal{O}(n)$ fields. Each field is dedicated to a router. When a packet reaches a router, the router's corresponding field should have been computed by all proofs from the router's upstream routers. If the field passes verification, the router then embeds proofs for all its downstream routers in the respective fields and sends the updated packet to the next hop.

Assuming a trusted source, a router can verify and update only $\mathcal{O}(1)$ fields for path validation [4], [14], [15]. A typical solution of this type is OPT [4]. It introduces a PVF field and a series of OPV fields in a packet header. The PVF field is initialized by the source. It is then used for verification by en-route routers and is updated if verification succeeds. The source also initializes one OPV field per en-route router, such that $\mathrm{OPV}_i = f(\mathrm{PVF}_{i-1})$, where $f(\cdot)$ is a verification function shared among all routers. Since the source is trusted by the other routers, it can precompute all the expected OPV fields. Each router does not need to compute a different proof for each downstream router because they share no keys. For services where the destination is skeptical of the trustworthiness of the source, en-route routers should share keys with the destination and compute proofs for it [4]. More generally, if all routers, including the source and destination, can be compromised or misbehave, pairwise key sharing and proof computation become necessary [7]. This increases the computational complexity back to $\mathcal{O}(n^2)$.

Weakening the security of path validation can further improve efficiency. For example, PPV [16] does not require a packet to be validated by all routers it visits. Instead, each router probabilistically embeds its proof in the packet header. A packet is marked by only two adjacent routers. The destination will verify the marked packet. With each packet verifying the forwarding behavior of two adjacent routers, all coflow packets may jointly verify the forwarding behavior of all links and thus the entire path. Since a packet is not verified by all the routers, it is possible that some attacking packets will evade security checks. PPV is suitable for services where one or a few attacking packets can have little impact [3].

C. Barriers to an Increase in Efficiency

In this paper, we focus on the generic form of path validation that enforces pairwise trust proofs among all routers on every packet. Notwithstanding progressive efficiency gains, existing path validation solutions face several barriers to making another leap of efficiency. A linear-size verification proof in terms of path length imposes a high overhead on the source for construction, the routers for verification and update, and the destination for verification. Packets with longer proofs cost more network bandwidth. This will likely lead to an additional end-to-end delay, which may outweigh the computation speedup on routers. Furthermore, with pairwise-shared keys, each node may need to prove itself to every other node using an individual provenance assurance. One such assurance, however, should be sufficient for a node to prove it had processed a packet.

Expansive verification proof. Existing path validation solutions enforce an $\mathcal{O}(n)$-size verification proof for an n-hop path. The proof size is linear in the path length. For example, ICING includes a verification proof with one 336-bit verifier field per router to guarantee an 80-bit security level [3]. Similarly, OPT requires an individual 128-bit OPV field for each router [4], while OSV requires a much shorter 16-bit OV field per router [14], [15], satisfying a 128-bit security level. Previous measurements show that it is common for a path to span more than 15 hops; some may reach 30 or 40 hops [19], [20]. A longer path leads to a longer verification proof. The more bandwidth such proofs cost, the less is left for payload transmission. This will likely increase the end-to-end delay. Furthermore, a packet with a longer verification proof requires more processing time. A router may thus have more queued packets than it would have had if the packets had shorter verification proofs. Both slower processing and longer queuing further inflate the end-to-end delay.

Expensive proof construction. Given an $\mathcal{O}(n)$-size verification proof, the source needs to precompute all $\mathcal{O}(n)$ of its fields. ICING requires that the source share a different key with each en-route router. Then, the source uses each key to compute a different verifier to prove itself to the corresponding router. Since ICING also requires pairwise shared keys among other routers, the source need not precompute verifiers for other routers. OPT and OSV, however, impose more computation on the source because they only use shared keys between the source and en-route routers. In other words, when en-route routers share no keys and have no means to verify each other, the source needs to precompute for each node the aggregate assurance of all its upstream routers. An untrusted source may also forge verification proofs [4]. From both efficiency and security perspectives, the source should be restricted to having only the secrets needed for generating its own provenance assurance. The source requires only one assurance at minimum to prove itself to all other nodes, so proving itself to each node in a different way is unnecessary.

TABLE I
PER-SOURCE, PER-ROUTER, AND PER-DESTINATION COMPLEXITY COMPARISON OF ATOMOS WITH EXISTING SYMMETRIC-KEY ENCRYPTION-BASED SOLUTIONS [3], [4], [14], [15].

Source Complexity:

Solution	Space		Computation		
	key	proof	initialization	verification	update
ICING	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(0)$	$\mathcal{O}(0)$
OPT	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(0)$	$\mathcal{O}(0)$
OSV	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(0)$	$\mathcal{O}(0)$
Atomos	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(0)$	$\mathcal{O}(0)$

Router Complexity:

Solution	Space		Computation		
	key	proof	initialization	verification	update
ICING	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(0)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
OPT	$\mathcal{O}(1)^*$	$\mathcal{O}(n)$	$\mathcal{O}(0)$	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$
OSV	$\mathcal{O}(1)^*$	$\mathcal{O}(n)$	$\mathcal{O}(0)$	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$
Atomos	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(0)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Destination Complexity:

Solution	Space		Computation		
	key	proof	initialization	verification	update
ICING	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(0)$	$\mathcal{O}(n)$	$\mathcal{O}(0)$
OPT	$\mathcal{O}(1)^*$	$\mathcal{O}(n)$	$\mathcal{O}(0)$	$\mathcal{O}(1)^*$	$\mathcal{O}(0)$
OSV	$\mathcal{O}(1)^*$	$\mathcal{O}(n)$	$\mathcal{O}(0)$	$\mathcal{O}(1)^*$	$\mathcal{O}(0)$
Atomos	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(0)$	$\mathcal{O}(1)$	$\mathcal{O}(0)$

*Notes: For OPT [4] and OSV [14], [15], the marked complexities of $\mathcal{O}(1)$ directly follow the conclusion of their designs as given. However, as discussed in Section II-B, the $\mathcal{O}(1)$ complexity increases to $\mathcal{O}(n)$ if pairwise trust is enforced as in ICING [3] and Atomos (in this paper).

Redundant provenance assurance. To generalize the previous observation, we find that current solutions require intermediate routers to generate more provenance assurances than necessary. Take ICING for example. As with the source, each intermediate router generates a provenance assurance for every downstream router. Each provenance assurance uses a different pairwise-shared key. The complexity of updating a verification proof on each node is thus $\mathcal{O}(n)$. Subsequent solutions such as OPT and OSV further demonstrate the $\mathcal{O}(n)$ update complexity. If a node does not trust the source, it uses keys agnostic to the source for generating provenance assurances for its downstream routers [4]. Using the same shared key with two or more routers is vulnerable to assurance forgery. A router thus requires $\mathcal{O}(n)$ keys, each shared with a different downstream router. Then, a node needs to generate $\mathcal{O}(n)$ different provenance assurances. Essentially, however, one assurance is sufficient for a router to prove to all others.

III. MOTIVATION

In this section, we motivate our construction of a noncommutative homomorphic asymmetric-key encryption scheme to address the preceding limitations (Section II-C). It provides a constant-size validation proof. Constant-size proofs not only economize bandwidth usage and increase network throughput but also enable validation of particularly long paths. Table I demonstrates how constant-size proofs significantly increase efficiency through our path validation solution Atomos.

To address the limitations of current path validation solutions (Section II-C), we advocate that path validation proofs be constructed by a noncommutative homomorphic asymmetric-key encryption scheme. Our scheme has the following properties: asymmetry, aggregation of proofs and noncommutativity. First, asymmetry minimizes the number of proof keys per router to one. This decreases the cost of exchanging and maintaining keys and creating proofs at each router. Second, our scheme aggregates multiple proofs into one, enabling a constant-size proof throughout the whole validation process. Third, noncommutativity enforces that different aggregation sequences yield different proofs. This guarantees that packets carrying valid proofs visit all the routers on the specified path in the correct order. Together, these three properties promise an efficient and fast path validation solution.

A. Asymmetry

We adopt asymmetric-key encryption to minimize the number of proofs each router needs in order to prove itself to other routers. Existing solutions, however, use symmetric-key encryption for the sake of speed. A router then proves itself to another using their shared key. To prevent forging, routers should share secret keys in a pairwise way, necessitating $\mathcal{O}(n^2)$ keys [3]. Otherwise, any other router holding a router's secret key can impersonate it and forge its proof [4]. Such a pairwise key-sharing requirement imposes much more overhead than necessary. Using asymmetric-key encryption, a router maintains only one pair of public and private keys, requiring only $\mathcal{O}(n)$ keys. Its proof generated using the private key can be verified by anyone holding the public key. A single proof is thus sufficient for one router to prove itself to others.

A major design challenge raised by asymmetric-key encryption is speed. The concern stems from the intrinsic speed gap between asymmetric- and symmetric-key encryption. We compensate for the otherwise slow speed of asymmetric-key encryption in our scheme with three design strategies. The first is directly inherited from the property of asymmetric-key encryption. That is, a single proof computed using the private key can prove a router to all other routers. A router thus computes only one proof for all its downstream routers, instead of one for each as in current solutions. The second design strategy, to be detailed shortly, is to aggregate multiple proofs into one constant-size proof. A router then needs to verify only one proof instead of one for each upstream router. Finally, we propose a dynamic rekeying technique to further shorten proofs without sacrificing security.

B. Aggregation of Proofs

Our scheme creates a single constant-size proof. To achieve this, we design a noncommutative additive magma for aggregating proofs into one, a noncommutative multiplicative magma for verifying an aggregated proof, and a homomorphic mapping from the additive magma to the multiplicative magma to apply a simple asymmetric-key encryption scheme for creating and verifying proofs. Given a single proof, both verification and update operations are computed over the only field. This helps greatly accelerate the otherwise slow asymmetric-key encryption. Furthermore, the constant-size proof is a major contribution by Atomos. Constant-size proofs avoid the explosion of packet sizes as path lengths increase. The median length of network paths is approximately 16 hops, while long paths can exceed 30 hops [21]. Consider, for example, ICING with 42 bytes per proof field [3]. A 16-hop path requires a proof of $42 \times 16 = 672$ bytes. Together with other header fields, they take up to nearly half of the 1500-byte MTU limit. This indicates a bandwidth utilization of 50%, as only half of the traffic comprises packet payloads. When the path length exceeds 30 hops, header fields may take up the whole packet space, possibly leaving no space for payloads. With constant-size proofs, Atomos can address all the preceding limitations. Consider, for example, using a 1,024-bit group size for asymmetric-key encryption. The generated proof field is 1,433 bits, that is, approximately 180 bytes (Section IV-F). Under the same security level, an ICING proof becomes longer than an Atomos proof when a path is longer than 5 hops. Atomos economizes bandwidth usage and increases network throughput. More importantly, Atomos enables path validation to be applied in networks with particularly long paths.

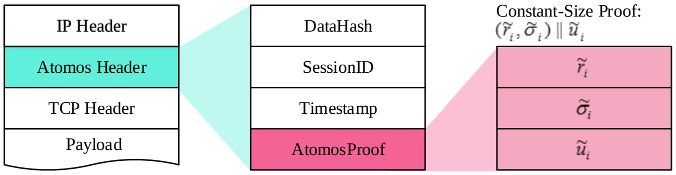


Fig. 1. Atomos header architecture. Among the four fields, DataHash, SessionID, and Timestamp follow existing designs, while AtomosProof is newly designed for a constant-size path validation proof. All header fields need to be initialized by the source. The subsequent routers need only verify and update AtomosProof, which consists of three fields.

C. Noncommutativity

Finally, the noncommutative operations in our scheme for aggregating and verifying proofs enforce a forwarding order. Path validation accepts forwarding only if a packet goes through all en-route routers in the correct order. Our scheme not only supports constant-size proofs but also enforces order checking inside proofs.

IV. Design

In this section, we present Atomos to validate network paths in an extremely efficient way. It uses our newly constructed noncommutative homomorphic public-key encryption scheme to overcome the aforementioned efficiency barriers of path validation. Atomos guarantees a constant-size verification proof. We also explore various design choices to accelerate Atomos.

A. Header Architecture

As with existing path validation solutions [4], [14]–[16], the Atomos header in a packet header is located between the IP header and the TCP header (Figure 1). Given the fixed length of the IP header, it is very convenient for routers to extract the Atomos header while parsing packet headers. There may be other options for the insertion of the path validation header. For example, we may append the Atomos header to the payload [10]. However, given various payload lengths, routers may have to look up various lengths of payloads until they find the appended Atomos header. This is less convenient than when the Atomos header directly follows an IP header with a fixed length. Furthermore, we consider a generalized scenario in line with existing path validation solutions; that is, all routers along the forwarding path support path validation. Otherwise, if only selective routers along the forwarding path support path validation, we need to prefix the IP header with the IP Tunnel header [5]. Note that the IP Tunnel header helps to direct packets to routers that are capable of validation. It does not enforce validation itself. Instead, after it guides a packet to the destined router indicated in its destination field, the destined router still needs to use the Atomos header for packet verification. Without loss of generality, we consider the generalized scenario here—inserting the Atomos header between the IP header and the TCP header without using the IP Tunnel header [3], [4], [14]–[16].

As shown in Figure 1, the Atomos header consists of four fields: DataHash, SessionID, Timestamp, and AtomosProof. The first three follow the design of existing solutions, while the fourth AtomosProof field differs from existing proof designs. Specifically, we construct a constant-size AtomosProof using our newly proposed noncommutative homomorphic asymmetric-key encryption scheme (Section III). All four fields are initialized by the source router, while only AtomosProof is verified and updated on subsequent routers.

DataHash. Given a packet with payload P, its DataHash field is set as the payload's hash value $H(P)$. We mainly use DataHash to verify packet integrity. The hash function H is shared on all routers. If an attacker cannot compromise H, it will not be able to forge a packet with a valid DataHash. Simply including DataHash in the packet header is insufficient to protect against packet forgery. Since H is shared among routers, a compromised or misbehaving router may compute a valid $H(P)$ for any P. Therefore, a path validation solution should also include $H(P)$ in the computation of validation proofs. As each router has unique credentials for computing proofs, a router cannot forge valid proofs of packets forged by other routers without knowing their credentials.

Following current Internet principles, packet integrity verification is usually performed at the end-host. For verification, a host first computes the hash value of the packet payload $H(P)$. Then, it compares $H(P)$ with the DataHash carried in the packet. If they match, integrity verification succeeds. Otherwise, the packet is deemed altered and should be dropped.

SessionID. For each session, the source and destination agree on a SessionID to track subsequent traffic and may also negotiate a session-specific configuration. Following OPT [4], we let the source and destination negotiate the path that traffic in a session will take at the creation of the session. Each en-route router locally records pairs of SessionIDs and corresponding paths. In this way, a packet carries only the SessionID instead of the specific path. This not only saves bandwidth but also protects path privacy [22].

The SessionID should also be included in AtomosProof computation. Otherwise, a misbehaving or compromised router can replace the SessionID and deviate the packet to the corresponding path. Consider two session IDs with corresponding paths having overlapping segments. For example, SessionIDa corresponds to the path (r_1, r_2, r_3) and SessionIDb corresponds to the path (r_1, r_2, r_4). Router r_2 may replace SessionIDa with SessionIDb. When router r_4 receives the redirected packet, since its SessionIDb leads it on a path to r_4, it is deemed valid. We can address such packet deviation by including the SessionID in AtomosProof computation. If router r_1 computes its proof using SessionIDa, r_4 will not validate the proof if SessionIDa is replaced with SessionIDb.

Timestamp. This records the time when the source creates a packet. It indicates packet freshness and helps avoid replay attacks in a session [4]. If Timestamp verification is not enforced, an attacker can replay a recorded valid packet with the Timestamp updated to the latest value. Therefore, the Timestamp should be included in AtomosProof computation as well.

AtomosProof. As shown in Figure 1, AtomosProof consists of three fields—$\tilde{r}_i$, $\tilde{\sigma}_i$, and $\tilde{u}_i$. We compute them following the encryption scheme in Section IV-C for a constant-size proof. Proof computation and verification require router identifiers and keys as well as other information (i.e., the DataHash, SessionID, and Timestamp) in the Atomos header. Algorithm 1 highlights the key functions for Atomos path validation. We will shortly detail them alongside the Atomos principles in Section IV-B, Section IV-C, and Section IV-D.

B. Validation Rationale

Noncommutative homomorphic asymmetric-key encryption. We now present the construction of our noncommutative homomorphic asymmetric-key encryption scheme. It is the key building block for Atomos to enable constant-size path validation proofs.

We first construct an additive magma and a multiplicative magma. Let $p > 2$ be a large prime number, $\mathbb{Z}_p = \{0, 1, ..., p-1\}$ and $\mathbb{Z}_p^* = \{1, 2, ..., p-1\}$. Let $X \times Y$ be the Cartesian product of sets X and Y.

Definition 1. Additive Magma $(\mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}, \star)$**:** We define a binary operation $\star : \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1} \to \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}$ as:

$$(a_1, b_1) \star (a_2, b_2) = (a_1 + a_2, a_1 + b_1 + b_2) \bmod (p-1).$$

The operation $\star$ is closed in $\mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}$, and we call $(\mathbb{Z}_{p-1}, \mathbb{Z}_{p-1}, \star)$ an additive magma.

Definition 2. Multiplicative Magma $(\mathbb{Z}_p^* \times \mathbb{Z}_p^*, \odot)$**:** We define a binary operation $\odot : \mathbb{Z}_p^* \times \mathbb{Z}_p^* \to \mathbb{Z}_p^* \times \mathbb{Z}_p^*$ as:

$$(a_1, b_1) \odot (a_2, b_2) = (a_1 a_2, a_1 b_1 b_2) \bmod p.$$

The operation $\odot$ is closed in $\mathbb{Z}_p^* \times \mathbb{Z}_p^*$, and we call $(\mathbb{Z}_p^* \times \mathbb{Z}_p^*, \odot)$ a multiplicative magma.

Next, we construct a homomorphic mapping F from the additive magma to the multiplicative magma.

Definition 3. We define a function $F : (\mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}, \star) \to (\mathbb{Z}_p^* \times \mathbb{Z}_p^*, \odot)$ as follows:

$$F(a, b) = (f(a), f(b)) = (g^a, g^b),$$

where we have a function $f : \mathbb{Z}_{p-1} \to \mathbb{Z}_p^*$ defined as $f(x) = g^x \bmod p$ and g is a generator of $\mathbb{Z}_p^*$.

We prove that 1) both the additive magma and the multiplicative magma are noncommutative in Lemma 1 (Appendix A) and 2) the function F is a noncommutative homomorphic function in Lemma 2 (Appendix A). From Lemma 1 and Lemma 2, we have the following result, the proof of which is in Appendix A:

Theorem 1. *A noncommutative homomorphic asymmetric-key encryption scheme can be constructed from the additive magma* $(\mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}, \star)$*, the multiplicative magma*

$(\mathbb{Z}_p^* \times \mathbb{Z}_p^*, \odot)$*, and the homomorphic function* $F : (\mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}, \star) \rightarrow (\mathbb{Z}_p^* \times \mathbb{Z}_p^*, \odot)$*.*

Validation rationale. The validation rationale is based on Theorem 1. Specifically, the noncommutative homomorphic asymmetric-key encryption scheme constructs a constant-size proof per packet. While the packet transits its specified path, each en-route router integrates its own proof into the proof. Furthermore, it validates the proof to check whether all its upstream routers have done so in the correct order. Consider an l-hop path $(r_1, r_2, ..., r_i, ..., r_l)$, where r_i, for $1 \leq i \leq l$, denotes the identifier of the ith en-route router. To construct the proof of router r_i, we introduce a two-tuple (r_i, σ_i). The first item uses the router identifier to specify the ownership of proof. The second item, to be elaborated shortly in Section IV-C, is computed using r_i's secret key and can be verified using its public key. Building on the homomorphism of function F in Equation 17, we construct (r_i, σ_i) over the magma $(\mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}, \star)$. Then, the aggregate constant-size proof for r_i to verify is as follows:

$$(\tilde{r}_i, \tilde{\sigma}_i) = (r_1, \sigma_1) \star ... \star (r_i, \sigma_i), \tag{1}$$

where

$$\tilde{r}_i = \sum_{1 \leq j \leq i} r_j, \text{ and } \tilde{\sigma}_i = \sum_{1 \leq j \leq i} \sigma_j + \sum_{1 \leq j \leq i-1} \sum_{1 \leq k \leq j} r_k, \tag{2}$$

according to Definition 1. Following Equation 17 in Lemma 2, we can verify the proof of router r_i if

$$F(\tilde{r}_i, \tilde{\sigma}_i) = F(r_1, \sigma_1) \odot F(r_2, \sigma_2) \odot ... \odot F(r_i, \sigma_i). \tag{3}$$

A notable aspect of this new design is that we do not have to know all specific $F(r_j, \sigma_j)$'s for $1 \leq j \leq i$ on the right side of Equation 3 for validation. In current symmetric-key encryption-based solutions, a router needs to hold other routers' proofs for validation. For example, ICING exchanges pairwise keys among routers, and a router can recompute the proof from another router using their shared key. Similarly, OPT lets a trusted source precompute proofs of en-route routers. Unlike existing solutions, Atomos adopts asymmetric-key encryption. A router now computes its proof using its secret key, which should be unknown to other routers. We also cannot simply carry router proofs in the packet header, as this violates the constant size requirement. Next, we detail how we design the proof item σ_i such that it enables the constant-size proof to be validated using only public keys and other available information carried in packet headers.

C. Proof Construction

Initialization.

Key generation and exchange among routers are outlined in lines 1-11 of Algorithm 1. Note that we follow the security assumption of ICING [3] with pairwise shared keys required. In this way, we do not need to assume that all the sources, destinations, and routers trust each other. Since Atomos is based on asymmetric-key encryption, each router r_i should first initialize a pair of secret keys k_i^{s} and public keys $k_i^{\mathrm{p}} = f(k_i^{\mathrm{s}}) = g^{k_i^{\mathrm{s}}} \bmod p$ (lines 8-9), where function f is defined in Definition 3 and g is a generator of $\mathbb{Z}_p^*$ (lines 4-6). To promote speedup, we choose a prime p such that $p-1$ has only two prime factors, and the generator g can be precomputed by line 6 [23]. The identifier r_i and corresponding public key k_i^{p} are public to other routers (line 11).

Algorithm 1: Atomos Path Validation

```
1  Function Initialization:
2      //key generation and exchange among routers;
3      //step 1: generator generation
4      $q \leftarrow$ random prime number such that $2q+1$ is prime;
5      $p \leftarrow 2q+1$;
6      generator $g \leftarrow$ random number such that $g^2 \not\equiv 1 \pmod p$ and $g^q \not\equiv 1 \pmod p$;
7      //step 2: secret and public key generation
8      $k_i^{\mathrm{s}} \leftarrow$ random prime number;
9      $k_i^{\mathrm{p}} = f(k_i^{\mathrm{s}}) = g^{k_i^{\mathrm{s}}} \bmod p$;
10     //step 3: key exchange
11     Router $r_i$ sends $(r_i, k_i^{\mathrm{p}})$ to other routers;
12 Function Construction:
13     if router $r_i$ is source $r_1$ then
14         SessionID $\leftarrow$ identifier of the current session;
15         Timestamp $\leftarrow$ creation time of the packet with payload $P$;
17         DataHash $\leftarrow H(P)$;
18         //construction of AtomosProof
19         $\tilde{r}_1 \leftarrow r_1$;
20         $h = \mathrm{DataHash}||\mathrm{SessionID}||\mathrm{Timestamp}$;
21         $c_1 \leftarrow$ random number;
22         $\tilde{\sigma}_1 \leftarrow k_1^{\mathrm{s}} + c_1 H(h) \bmod (p-1)$;
23         $\tilde{u}_1 \leftarrow u_1 = f(c_1) = g^{c_1} \bmod p$;
24         AtomosProof = $\mathrm{Proof}_1 \leftarrow (\tilde{r}_1, \tilde{\sigma}_1)||\tilde{u}_1$;
25     else
26         //$r_i$ is intermediate and updates only AtomosProof
27         $h = \mathrm{DataHash}||\mathrm{SessionID}||\mathrm{Timestamp}$;
28         $c_i \leftarrow$ random number;
29         $\sigma_i \leftarrow k_i^{\mathrm{s}} + c_i H(h) \bmod (p-1)$;
30         $u_i \leftarrow f(c_i) = g^{c_i} \bmod p$;
31         $(\tilde{r}_i, \tilde{\sigma}_i) \leftarrow (\tilde{r}_{i-1}, \tilde{\sigma}_{i-1}) \star (r_i, \sigma_i)$;
32         $\tilde{u}_i \leftarrow \tilde{u}_{i-1} u_i \bmod p$;
33         AtomosProof = $\mathrm{Proof}_i \leftarrow (\tilde{r}_i, \tilde{\sigma}_i)||\tilde{u}_i$;
34 Function Verification:
35     //$\mathrm{Proof}_i = (\tilde{r}_i, \tilde{\sigma}_i)||\tilde{u}_i$ is verified by router $r_{i+1}$;
36     $\delta(r_i) \leftarrow \sum_{1 \leq j \leq i-1} \sum_{1 \leq k \leq j} r_k$;
37     $left \leftarrow g^{\tilde{\sigma}_i} \bmod p$;
38     $right \leftarrow g^{\delta(r_i)} (\tilde{u}_i)^{H(h)} \prod_{1 \leq j \leq i} k_j^{\mathrm{p}} \bmod p$;
39     if $left == right$ then
40         Accept the packet and update the proof;
41     else
42         Drop the packet;
```

Proof construction at the source. With the DataHash, SessionID, and Timestamp generated following Section IV-A (lines 14-18), the source continues by generating the AtomosProof as follows (lines 19-24). As mentioned in Section IV-B, router r_i includes its proof of a two-tuple (r_i, σ_i). The item σ_i should be computed using r_i's secret key such that it can be verified by other routers using r_i's public key. Normally, asymmetric-key encryption enforces exponentiation that is much slower than polynomial computation. To promote speedup, we use r_i's secret key k_i^{s} with an addition instead of a slow exponentiation. Specifically, the source router r_1 computes σ_1 as:

$$\sigma_1 = k_1^{\mathrm{s}} + c_1 H(h) \bmod (p-1),$$

where c_1 is a random number to hide k_1^{s} and

$$h = \mathrm{DataHash}||\mathrm{SessionID}||\mathrm{Timestamp}$$

to verify packet integrity. Note that c_1 should be randomly generated for each packet. If it were a constant, a series of packets would lead to a system of equations that would render both k_1^{s} and c_1 solvable.

Solely using (r_1, σ_1) as the source router r_1's proof, however, cannot enable verification by subsequent routers. This is because it uses a newly generated random number c_1 to hide k_1^{s}. Without c_1, computation using k_1^{s} can be verified by its corresponding public key k_1^{p}. On the basis of this property of asymmetric-key encryption, we regard the random number c_1 as another one-time secret key of the source router r_1. Then, we compute its corresponding public key u_1 as follows:

$$u_1 = g^{c_1} \bmod p. \tag{4}$$

We send u_1 together with the two-tuple (r_1, σ_1) to the next-hop router r_2. Using u_1 and r_1's public key k_1^{p}, r_2 can verify the proof generated using their secret counterparts c_1 and k_1^{s}.

Definition 4. Proof at the Source Router r_1: The proof constructed by the source router r_1 is defined as follows:

$$\mathrm{Proof}_1 = (\tilde{r}_1, \tilde{\sigma}_1)||\tilde{u}_1,$$

where we have

$$(\tilde{r}_1, \tilde{\sigma}_1) = (r_1, \sigma_1), \text{ and } \tilde{u}_1 = u_1.$$

To further accelerate the proof construction process, we can precompute a set of (c_1, u_1) pairs and store it on router r_1. Then, we can save the time cost of the random number generation for generating c_1 and of the exponentiation for generating u_1 in Equation 4. We delete a pair after using it to compute the proof of a packet. The set can be replenished through precomputation while the router is less busy.

Proof construction on intermediate routers. One major design choice in constructing the proof of an intermediate router is how to integrate its proof into the upstream routers' such that a constant proof size is maintained. An intermediate router updates only AtomosProof (lines 27-33). The other three fields—DataHash, SessionID, and Timestamp—are unmodified. As with the source router r_1, router r_i first generates its pair (r_i, σ_i) as follows (lines 27-29):

$$\sigma_i = k_i^{\mathrm{s}} + c_i H(h) \bmod (p-1). \tag{5}$$

Following Equation 4, we compute the corresponding u_i of the random c_i as (line 34):

$$u_i = g^{c_i} \bmod p. \tag{6}$$

Based on Equation 1, we use the magma addition in Definition 1 to compute a constant-size proof.

Definition 5. Proof of Router r_i for $1 < i \leq l$: The proof constructed by the en-route router r_i, for $1 < i \leq l$, is defined as:

$$\mathrm{Proof}_i = (\tilde{r}_i, \tilde{\sigma}_i)||\tilde{u}_i,$$

where we have

$$(\tilde{r}_i, \tilde{\sigma}_i) = (\tilde{r}_{i-1}, \tilde{\sigma}_{i-1}) \star (r_i, \sigma_i), \text{ and}\tilde{u}_i = \tilde{u}_{i-1} u_i \bmod p.$$

Router r_i does not immediately compute Proof_i after receiving Proof_{i-1}. Instead, it first verifies Proof_{i-1}. Only if Proof_{i-1} is valid does r_i need to construct Proof_i according to Definition 5 and forward the valid packet. An invalid Proof_{i-1} fails path validation, and r_i simply drops the invalid packet.

D. Proof Verification

As mentioned above, we leverage noncommutative homomorphism for proof verification following Equation 3:

$$F(\tilde{r}_i, \tilde{\sigma}_i) = F(r_1, \sigma_1) \odot F(r_2, \sigma_2) \odot ... \odot F(r_i, \sigma_i).$$

The input of the left side—$(\tilde{r}_i, \tilde{\sigma}_i)$—is readily available in the received packet header. The items on the right side, however, are unknown to other routers r_j for $j \notin [1, i]$, as they have been integrated into a single constant-size proof. They also cannot be recomputed by r_j, because each router computes its σ using its secret key. The Atomos proof following Definition 5 can be verified without the specific values of the individual items on the right side of the above equation. Its validity can be determined using only publicly known information such as router identifiers r_i and corresponding public keys k_i^{p} as well as information carried in received packets such as $\tilde{u}_i$.

We present the verification process of an Atomos proof (lines 34-42) in Theorem 2 and prove the soundness of the method.

Theorem 2. $\mathrm{Proof}_i = (\tilde{r}_i, \tilde{\sigma}_i)||\tilde{u}_i$ *is valid if the following equation holds:*

$$g^{\tilde{\sigma}_i} \bmod p = g^{\delta(r_i)} \Big(\prod_{1 \leq j \leq i} k_j^{\mathrm{p}}\Big)(\tilde{u}_i)^{H(h)} \bmod p, \tag{7}$$

where $\delta(r_i)$ is defined as follows:

$$\delta(r_i) = \begin{cases} 0, & \textit{if } i = 1; \\ \sum_{1 \leq j \leq i-1} \sum_{1 \leq k \leq j} r_k, & \textit{if } 1 < i \leq l. \end{cases}$$

Proof. A valid Proof_i satisfies the following equation:

$$F(\tilde{r}_i, \tilde{\sigma}_i) = F(r_1, \sigma_1) \odot F(r_2, \sigma_2) \odot ... \odot F(r_i, \sigma_i).$$

The left side of this equation can be derived as:

$$F(\tilde{r}_i, \tilde{\sigma}_i) = (g^{\tilde{r}_i}, g^{\tilde{\sigma}_i}). \tag{8}$$

The right side can be derived as:

$$\begin{aligned} & F(r_1, \sigma_1) \odot F(r_2, \sigma_2) \odot ... \odot F(r_i, \sigma_i) \\ = & \ (g^{\sum_{1 \leq j \leq i} r_j}, g^{\delta(r_i)} g^{\sum_{1 \leq j \leq i} \sigma_j}) \\ = & \ (g^{\tilde{r}_i}, g^{\delta(r_i)} (\tilde{u}_i)^{H(h)} \prod_{1 \leq j \leq i} k_j^{\mathrm{p}}). \end{aligned} \tag{9}$$

To derive Equation 9 from its preceding line, the first item follows Equation 2 and the second item is derived as follows:

$$\begin{aligned} & g^{\delta(r_i)} g^{\sum_{1 \leq j \leq i} \sigma_j} \\ = & \ g^{\delta(r_i)} (\prod_{1 \leq j \leq i} g^{k_j^{\mathrm{s}}}) (\prod_{1 \leq j \leq i} g^{c_j H(h)}) \\ = & \ g^{\delta(r_i)} (\prod_{1 \leq j \leq i} k_j^{\mathrm{p}}) (\tilde{u}_i)^{H(h)}. \end{aligned}$$

Comparing the pairs in Equation 8 and Equation 9, we observe that they share the same first item. Therefore, the equality of the pairs holds if their second items are equal, that is, if $g^{\tilde{\sigma}_i} \bmod p = g^{\delta(r_i)} (\tilde{u}_i)^{H(h)} \prod_{1 \leq j \leq i} k_j^{\mathrm{p}} \bmod p$. This equation exactly resembles Equation 7. □

By Theorem 2, we can verify an Atomos proof following Equation 7 using the proof items $\tilde{\sigma}_i$, generator g, router identifiers r_i, proof items $\tilde{u}_i$, and packet digest $H(h)$. Among this information, $\tilde{\sigma}_i$ and $\tilde{u}_i$ are carried in the packet, while g and r_i are locally stored. To further speed up the validation process, each router can precompute $g^{\delta(r_i)}$ and $\prod_{1 \leq j \leq i} k_j^{\mathrm{p}}$.

E. Selection of the Hash Function

The property required by Atomos for the hash value h is that h does not have an inverse in $\mathbb{Z}_{p-1}$. As shown in Appendix B, if h has an inverse in $\mathbb{Z}_{p-1}$, then it is known that an attacker can forge a valid proof in polynomial time in the number of bits of p. Since no even integer has an inverse in $\mathbb{Z}_{p-1}$, we can select a hash function H that always creates an even hash value h.

F. Proof Shortening by Dynamic Rekeying

We now address the concern of proof length due to the large discrete logarithm group size (i.e., $\log_2 p$, the size/length of the modulo base p in, for example, Definitions 1-2) of asymmetric-key encryption. By Definition 5, a proof consists of three fields: $\tilde{r}_i$, $\tilde{\sigma}_i$, and $\tilde{u}_i$. We first analyze the length of each field.

- Field $\tilde{r}_i$ is the summation of the identifiers of en-route routers prior to router r_i (Equation 2). It should be no greater than the summation of the identifiers of all in-network routers. Let m denote the number of routers in the network. It suffices to use $\log_2 m$ bits as router identifiers. Moreover, the summation of all router identifiers is $\frac{m(m-1)}{2}$, which can be represented by at most $\log_2 \frac{m(m-1)}{2} \approx 2 \log_2 m$ bits. Given that the number of routers in the Internet may be less than the number of connected devices and that each device connects to the Internet via an IP address, we can determine an upper bound of 2^{32} for the number of routers currently in the IPv4 Internet. Therefore, we consider 32-bit router identifiers in what follows. The length of the field $\tilde{r}_i$ has an upper bound of $2 \log_2 2^{32} = 64$ bits.

TABLE II
TYPICAL DISCRETE LOGARITHM GROUP SIZES AND CORRESPONDING SECURITY LEVEL [24] VERSUS PROOF SIZE.

Group Size (bit)	1,024	2,048	3,072	7,680
Secret Key Size (bit)	160	224	256	384
Security Level (bit)	80	112	128	192
Proof Size (bit)	1,415	2,503	3,559	8,295

- Field $\tilde{\sigma}_i$ comprises two factors: $\sum_{1 \leq j \leq i} \sigma_j$ and $\sum_{1 \leq j \leq i-1} \sum_{1 \leq k \leq j} r_k$ (Equation 2). Based on the respective lengths of the parameters involved, the first factor dominates the overall length. In the first factor, σ_i comprises two further factors, k_i^{s} and $c_i H(h)$, where the second factor dominates the overall length. Let $\log_2(c_i H(h))$ denote the length of $c_i H(h)$. Then, the length of $\sum_{1 \leq j \leq i} \sigma_j$ is bounded by $\log_2(c_i H(h)) + \log_2 i$. Given that i corresponds to the hop index of a router, it can be upper bounded by the path length. Let us consider a path of sufficient length for practical purposes, of 1,024 hops. Then, we can estimate the length of field $\tilde{\sigma}_i$ as $\log_2(c_i H(h)) + 10 = \log_2 k_i^{\mathrm{s}} + \log_2 H(h) + 10$ bits, because the random number serves as a secret key and is bounded by the size of k_i^{s}.
- Field $\tilde{u}_i$ is the result of multiplying a series of u_i values, by Definition 5. According to Equation 6, u_i is a large number computed using the modulo base p. Its length can be upper bounded by the length of p, which is $\log_2 p$.

Therefore, the size of an Atomos proof is estimated as:

$$|\mathrm{Proof}| = \log_2 k_i^{\mathrm{s}} + \log_2 H(h) + 10 + \log_2 p + 2 \log_2 m.$$

Table II lists typical lengths of the group size (i.e., $\log_2 p$) and secret key size (i.e., $\log_2 k_i^{\mathrm{s}}$) and their corresponding security levels [24] and proof sizes. Given a commonly used 128-bit security level, an Atomos proof is 445 bytes. However, simply using a smaller group size weakens the security level. For example, a 1,024-bit group size yields a much shorter 177-byte proof and a much lower 80-bit security level. We need to find a design strategy that shortens the Atomos proof without sacrificing security.

We propose a dynamic rekeying technique to shorten Atomos proofs while preserving security. The key idea is twofold. First, we use a smaller group size to obtain a shorter proof. Second, we dynamically change the keys of routers. The dynamic rekeying technique increases both security and efficiency for the two reasons below.

First, path validation protocols use encryption for unforgeability rather than confidentiality. When encryption is used to protect the confidentiality of messages, a strong security level is required such that encrypted messages can only be cracked by an attacker after a long period of time (e.g., hundreds of years). However, validation proofs are not secrets to protect. The major purpose of encryption is to prevent forgeability. That is, they should be difficult for an attacker to forge without all the keys for computing them. Since the keys are kept on routers and the lifecycle of a router may span dozens of years at most, the keys can be shortened accordingly while unnecessarily preserving a security level that will take hundreds of years to circumvent.

Second, once a packet arrives at the destination with a validated proof, we no longer require the proof to be secure. Given that the lifecycle of a packet, a session, or a connection is much shorter than that of a router, we can further shorten the keys. However, we cannot simply configure router keys such that they are only robust within the lifecycle of a specific connection. Routers usually need to handle a steady stream of traffic. Once their keys are cracked, an attacker may easily forge validation proofs for packets thereafter. This motivates us to dynamically change router keys. If the rekeying frequency guarantees that the keys are changed before they can be cracked, Atomos is resistant against forging attacks. For example, if a router key is configured to be crackable in 10 hours, we can change it every 9 hours.

The rekeying frequency depends on the security level determined by the group size. Our asymmetric-key encryption scheme is an application of Diffie-Hellman. Breaking it reduces to solving the discrete logarithm problem [25]. The best known algorithm for solving discrete logarithms is the number field sieve, whose computational complexity can be quantified by the general number field sieve (GNFS) as [26]:

$$\exp\left(\left(\sqrt[3]{\frac{64}{9}}+o(1)\right)(\ln p)^{\frac{1}{3}}(\ln\ln p)^{\frac{2}{3}}\right)=L_p\left[\frac{1}{3},\sqrt[3]{\frac{64}{9}}\right].$$

A derivation of the preceding equation can estimate the security level corresponding to group size $n=\log_2 p$ as [27]:

$$\text{security level}=\frac{1.923*\sqrt[3]{n*\ln 2}*\sqrt[3]{[\ln(n*\ln 2)]^2}-4.69}{\ln 2}.$$

In estimating the time to break an encryption, a known cracking time for a specific security level is usually used as a reference. A recent reference dates back to 16 June 2016, when a 768-bit prime module of a discrete logarithm was cracked after 6,600 core years since February 2015 [28]. A core year indicates a year of time spent by a single core. According to Moore's law, the hardware speed per dollar doubles every eighteen months. As of July 2019 upon the submission of this paper, we can estimate the hardware speedup since February 2015 as $2^3=8$ times. Therefore, using a current computer to break the 768-bit prime module would take $6,600/8=825$ core years. Given that a 768-bit prime module delivers a 66-bit security level, an x-bit security level would take $825\times 2^{x-66}/c$ core years to break using a c-core computer. Table III provides the estimated time needed to break Atomos with various group sizes by a 10,000-core attacker. A 600-bit group size would take such a powerful attacker approximately two days to crack. This time period is already far longer than the time span of most network connections. It is therefore feasible to set the rekeying frequency as once every two days.

TABLE III
ESTIMATES OF THE TIME NEEDED TO BREAK ATOMOS WITH VARIOUS GROUP SIZES BY A 10,000-CORE ATTACKER. WE ALSO COMPUTE THE NUMBER OF HOPS THAT AN ICING PROOF OF THE SAME SIZE [3] CAN SUPPORT.

Group Size (bit)	500	600	768	1,024
Security Level (bit)	56	62	66	80
Cracking Time	42.3 min	1.9 days	30.1 days	1351 years
Proof Size (bit)	843	955	1,131	1,415
ICING Hop Count	3	4	4	5

We further compare the proof size of Atomos and ICING in Table III. Specifically, when we use dynamically changed short keys for Atomos, we should compare it with existing symmetric-key-based solutions integrated with dynamic rekeying. Using shorter symmetric keys, the proofs also become shorter. If a constant-size proof is still much longer than a linear one given a quite long forwarding path, its necessity diminishes. An ICING proof consists of three fields: the router metadata, proof of consent (PoC), and proof of provenance (PoP). The router metadata include a NodeID, which is used as the public key of elliptic curve cryptography (ECC), and its size determines the ICING security level and a corresponding 32-bit tag. The PoC is a 16-bit expiration time indicator. The PoP comprises a 96-bit proof using PRF-96 and a 32-bit hardener using PRF-32. Except for the NodeID, the sizes of the fields are fixed and account for 22 bytes. Now we have

$$|\text{ICING Proof Size}|=|\text{NodeID}|+22\text{bytes}.$$

As shown in Table III, even when Atomos uses small group sizes, the size of an ICING proof already exceeds that of an Atomos proof when the forwarding path to validate is more than 4 hops. This demonstrates the potential of Atomos's constant-size proof in saving bandwidth and computation.

G. Speedup

Throughout proof construction and verification, we have explored various design choices to accelerate Atomos.

- Atomos requires routers to exchange only their public keys. Such key exchange needs be done only once after rekeying, rather than sessionwise, as in existing solutions.
- Atomos uses the SessionID in packet headers to indicate forwarding paths. By not including on-path router identifiers in packet headers, we can decrease the time for routers to encapsulate and parse packets.
- To accelerate proof construction, each router generates several (x,y) pairs, precomputes g^δ and $g^\delta y$, and stores the information in a table. After routers change the key, it can be directly fetched from the stored keys. This saves initialization time.
- To accelerate proof construction, each router precomputes and stores the (c,u) pairs. After a (c,u) is used, it can be fetched directly from the stored pairs. This saves the time of generating c and performing exponentiation on u.
- To accelerate proof verification, each router precomputes and stores $g^{\delta(r_i)}$ and $\prod_{1\le j\le i} k_j^{\mathrm{p}}$. Both values can be directly used for verifying packets in the same session.
- Atomos requires that each router update only valid proofs. That is, Atomos first verifies the proofs of received packets, and only if the verification succeeds will a router update the proof by integrating its own. This avoids time wasted on updating invalid proofs.

V. SECURITY

In this section, we prove the proof unforgeability property of Atomos and its robustness against other attacks such as packet alteration, injection, and deviation. As with existing path validation solutions, we also discuss the nonsecurity goals of Atomos, such as packet dropping and traffic monitoring.

A. Proof Unforgeability

As with existing path validation solutions, the major security goal of Atomos is proof unforgeability. This guarantees that an attacker cannot forge a valid proof to pass validation. This ultimately enforces packet forwarding along specified paths.

Theorem 3. *If a probabilistic polynomial-time-bounded attacker cannot compromise the secret keys of routers, Atomos robustness against forgeability depends on the difficulty of solving the discrete logarithm problem (DLP). Atomos adopts a discrete algorithm that has no polynomial-time solution. An attacker cannot forge a valid proof to pass Atomos with a probability higher than that of random guessing.*

Proof. Given a generator g, a prime number p, router identifiers r_i, router public keys k_i^{p}, and a valid $\mathrm{Proof}_{i-1} = (\tilde{r}_{i-1}, \tilde{\sigma}_{i-1})||\tilde{u}_{i-1}$, we represent the forged proof $\acute{\mathrm{Proof}}_i$ as:

$$\acute{\mathrm{Proof}}_i = (\acute{r}_i, \acute{\sigma}_i)||\acute{u}_i,$$

where $\acute{\sigma}_i$ and $\acute{u}_i$ are forged in polynomial time, while $\acute{r}_i$ can be easily computed by $\acute{r}_i = \tilde{r}_{i-1} + r_i$. By Theorem 2, as parts of a valid proof, $\acute{\sigma}_i$ and $\acute{u}_i$ should satisfy:

$$g^{\acute{\sigma}_i} = g^{\delta(r_i)}(\acute{u}_i)^{H(h)} \prod_{1 \le j \le i} k_j^{\mathrm{p}}. \tag{10}$$

Additionally, let $\acute{\sigma}_i = \tilde{\sigma}_{i-1} + \alpha + \tilde{r}_{i-1}$. Then, we can derive $g^{\acute{\sigma}_i}$ in a different way, as follows.

$$\begin{aligned} g^{\acute{\sigma}_i} &= g^{\tilde{\sigma}_{i-1} + \alpha + \tilde{r}_{i-1}} \\ &= g^{\delta(r_i)}(\tilde{u}_{i-1})^{H(h)} g^{\alpha} \prod_{1 \le j \le i-1} k_j^{\mathrm{p}}. \end{aligned} \tag{11}$$

Comparing Equation 10 and Equation 11, we have

$$(\acute{u}_i)^{H(h)} = (\tilde{u}_{i-1})^{H(h)} (k_i^{\mathrm{p}})^{-1} g^{\alpha}. \tag{12}$$

Given that the hash function $H(\cdot)$ is noninverse, the attacker cannot exploit an inverse hash to forge a valid proof (Appendix B). To make the forged proof valid, the attacker needs to forge both $\acute{u}_i$ and α that satisfy Equation 12. According to the Atomos design, they should follow the representations of $\alpha = k_i^{\mathrm{s}} + cH(h)$ and $\acute{u}_i = \tilde{u}_{i-1} g^{cH(h)}$. Therefore, whether the forged proof $\acute{\mathrm{Proof}}_i$ can be computed in polynomial time depends on the difficulty of solving α in Equation 12, which resembles a DLP. If we configure Atomos with a discrete algorithm that has no polynomial-time solution, the attacker cannot forge a valid proof in polynomial time. □

B. Validation Robustness

Although an attacker cannot forge valid proofs, the attack may try to circumvent validation in various ways, such as by altering or deviating packets. We now discuss common attacks addressed by existing solutions and investigate how Atomos guarantees robustness against them. The security goal is that an invalid proof must fail verification.

Packet alteration. A modified payload results in a different DataHash than that carried in the packet header. Payload alteration can thus be easily detected. If the attacker exploits a compromised or misbehaving router, the hash function used for computing the DataHash can also be exploited. With the hash function, the attacker can compute a valid DataHash for any modified payload. This cannot circumvent Atomos either, because an altered DataHash in the packet header fails proof verification. Specifically, the source constructs its proof using the DataHash of the original packet. After the DataHash is modified by an attacker, the attacker needs to forge the source's proof using the modified DataHash. Since the source uses its secret key for proof construction, the attacker cannot forge the source's proof unless it compromises its secret key.

Packet injection. An attacker may inject packets with crafted payloads. To make the injected packets pass verification, the attacker has to forge valid proofs. This cannot be achieved by a polynomial-time-bounded attacker, by Theorem 3. The attacker may also try recording packets with valid proofs and then injecting these valid packets later. This resembles a replay attack. The SessionIDs and Timestamps carried in packet headers are powerful in preventing replay attacks [4]. Routers can drop packets carrying the SessionIDs of long completed sessions or Timestamps beyond a reasonable delay. However, if a packet is replayed soon after it is captured, routers cannot detect it simply using its SessionID and Timestamp. Detecting such types of replayed packets is a challenge facing any network [29]. It is orthogonal to the task of path validation. A feasible solution is for routers to log the packet digests of current sessions for a certain time period [29]. Upon receiving a packet, a router first computes its digest and compares it with logged ones. Finding a match indicates a replay attack.

Packet deviation. By manipulating a compromised or misbehaving router, an attacker may deviate packets from their specified paths. Take, for example, a packet with SessionID1 and assigned path (r_a, r_b, r_c). Consider a case in which the router r_b deviates the packet to an off-path router, say r_d. Since r_d is not on the path corresponding to SessionID1, it has no locally recorded path coupled with SessionID1. Then, r_d can easily tell that it is not supposed to be a router that forwards the received packet. Therefore, r_d drops the packet and suppresses the packet deviation attack. The attacker could replace SessionID1 with, for example, SessionID2, which does correspond to a forwarding path including r_d. In this case, r_d is an intersection of the two paths of SessionID1 and SessionID2. Let (r_e, r_d, r_f) denote SessionID2's corresponding path. For the deviated packet to pass proof verification on r_d, r_b has to forge a proof that r_e has validated the packet. Because of the proof nonforgeability property (Theorem 3), r_b cannot forge r_e's proof without compromising r_e's secret key.

C. Nonsecurity Goals

As with existing solutions [3], [4], Atomos does not address the following packet forwarding and processing misbehaviors:

- A misbehaving router may arbitrarily drop packets. This is usually addressed by fault localization protocols that detect erroneous routers [30].
- A misbehaving router may copy and forward packets of interest to certain servers for traffic analysis. Such misbehaviors can be stealthy and hard to detect.

- A misbehaving router may simply verify and update proofs without providing the process it is supposed to perform on the packet. For example, when a compromised router is connected to a middlebox, the router may forward packets without submitting them to the middlebox for security checks [31].
- A misbehaving router may disturb key distribution. The design focus of path validation is essentially that, given shared keys and allocated paths, end-hosts and routers should collaborate to detect and discard packets that have not followed the allocated paths. We refer to countermeasures in the literature to secure the key distribution process [30], [32]; this is orthogonal to path validation.

VI. Implementation

We implement Atomos using Click routers [33]. A Click router comprises a number of packet processing modules called elements. The packet validation logics of Atomos function through a new element, `Atomos`. Following Algorithm 1, the three key functions we implement are `initialization`, `construction`, and `verification`. Since these functions involve computation over large numbers (e.g., 1,024 bits), we use the `BIGNUM` structure and related operations from the OpenSSL toolkit. During `initialization`, we aim to generate router keys. We first use `BN_new()` to allocate and initialize `BIGNUM` structures to store the large numbers generated later. Then, we call `BN_generate_prime()` to generate the random prime p and private keys. The prime p is then used to compute the generator g. All random numbers that will be used can be generated by calling `BN_rand()`. For both `construction` and `verification`, we need to include the DataHash, SessionID, and Timestamp in the computation. One note of caution is that we use `BN_mod_mul()` for large-number multiplication without overflow. Following the standard element framework, these functions should be called by the `simple_action` function, which is the default function in an element to process packet headers and payloads.

To enable Atomos, we need to call the `Atomos` element in the configuration file. For ease of implementation, we inherit it from the configuration of `fake-iprouter` in Click. Since path validation only processes packet headers, we put the `Atomos` element after the `CheckIPHeader` element. Specifically, a packet first goes through the `Strip` element to remove the Ethernet header and then the `CheckIPHeader` element to check header correctness. According to Figure 1, the Atomos header is located between the IP header and TCP header. Therefore, we call the `Atomos` element after `CheckIPHeader`. If the proof validation succeeds, we update the proof. Both verification and update are defined in the `Atomos` element. Packets failing verification are dropped while verified packets are forwarded to the next-hop router. To forward a valid packet to the next hop, we call the `rt` element, which performs static IP lookup.

VII. Evaluation

In this section, we evaluate Atomos in comparison with ICING [3] under a generic path-validation framework requiring pairwise trust and packet granularity. We compare Atomos with ICING [3], OPT [4], and OSV [14], [15]. We, however, do not compare Atomos with PPV because PPV does not enforce per-packet validation as all the other solutions do. We use a 4-core server with Intel Xeon Skylake 6146 CPUs (3.2 GHz) and 8 GB memory. Test packets are generated using the `InfiniteSource` function integrated in Click. All reported statistics are averaged over 100 runs. The results demonstrate that Atomos provides short proofs and fast validation.

A. Proof Size

In addition to the preliminary statistics of proof size in Section IV-F, we present a more comprehensive measurement and analysis. As shown in Figure 2(a)-(d), Atomos guarantees a constant-size proof, which becomes increasingly shorter than that of ICING as the forwarding path exceeds a fairly small value (e.g., 3 hops under a 56-bit security level and 5 hops under an 80-bit security level). The header space saved increases with path length (Figure 2(e)-(h)). For example, given a median length of network paths of approximately 16 hops [21], Atomos saves 81.71% of the header space in comparison with ICING under a 56-bit security level, while the saved header space still accounts for as much as 73.66% under an 80-bit security level. This demonstrates the potential of Atomos in shortening the packet size and, in turn, saving network bandwidth.

Although OPT and OSV do not enforce pairwise trust among all end-hosts and routers, we investigate the overhead induced by their extra validation proofs. The proof field of OPT consists of the DataHash, SessionID, Timestamp, PVF, and OPVs. The number of OPVs is determined by the number of hops in the forwarding path, and each OPV field is 16 bytes. All of these parts are 16 bytes except the Timestamp, which is 4 bytes; the proof size of OPT is $(52+16n)$ bytes, where n is the number of hops. According to Table II, at the same security level, Atomos has a fixed proof size of 3,559 bits. When the path length exceeds 25 hops, OPT has a longer proof.

The OSV header is slightly more complicated than that of OPT; it consists of 12-byte eigenvalues (version, header length, unused, credential length, user id, row index, and matrix index), a 16-byte credential c, an 80-byte PVF, and 2 bytes for each OV. Its total proof size is $(108+2n)$ bytes, where n represents the number of nodes in the path. At a security level of 128 bits, it would take 168 hops for Atomos to reach OSV. However, as we discussed in Section IV-F, Atomos uses dynamic rekeying so that we can maintain the same security level while using shorter public keys. In this way, we can drastically reduce the communication overhead. For example, if Atomos uses a 1,024-bit public key and a 160-bit private key, the total proof size would be compressed to 1,415 bits, and it would only need 8 hops to outperform OPT and 35 hops to reach OSV.

For storage overhead, each router in Atomos needs to store its own private key and all the public keys of other on-path routers in a local table. ICING and OPT both calculate the symmetric key in the forwarding process so that they avoid storing all the symmetric keys locally. However, this causes

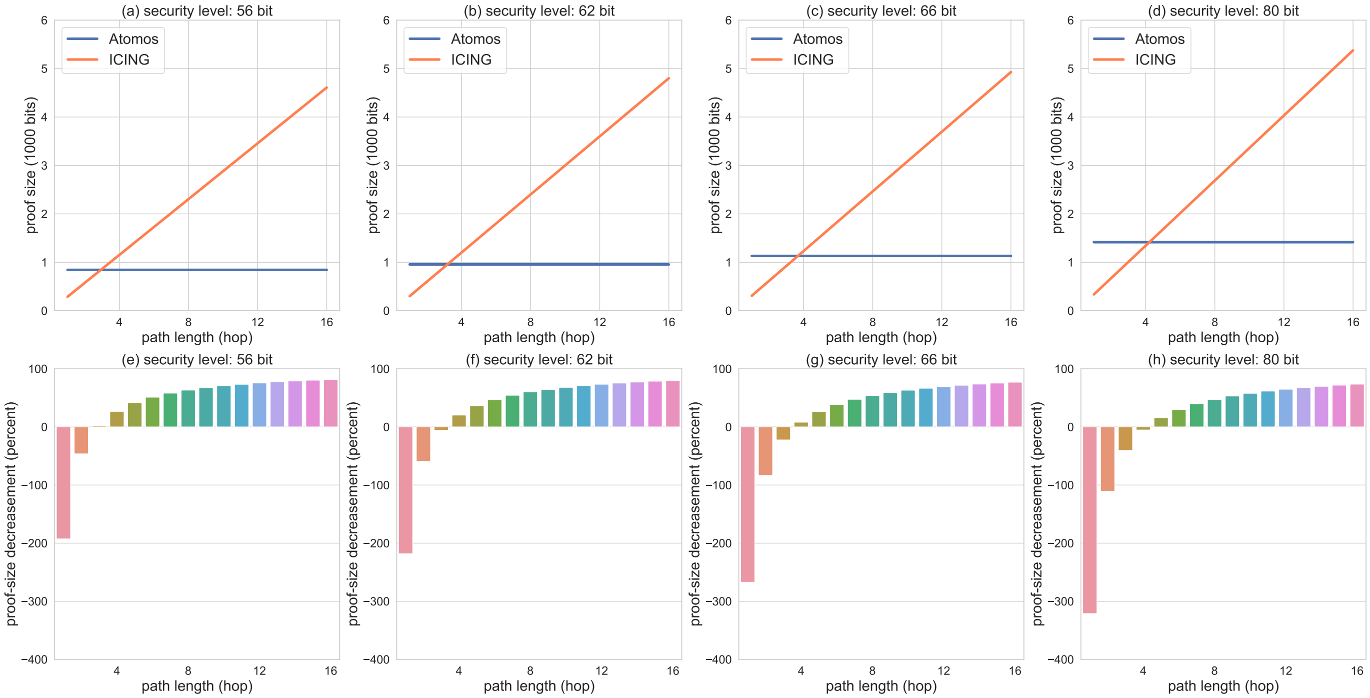


Fig. 2. Comparison of proof sizes between Atomos and ICING.

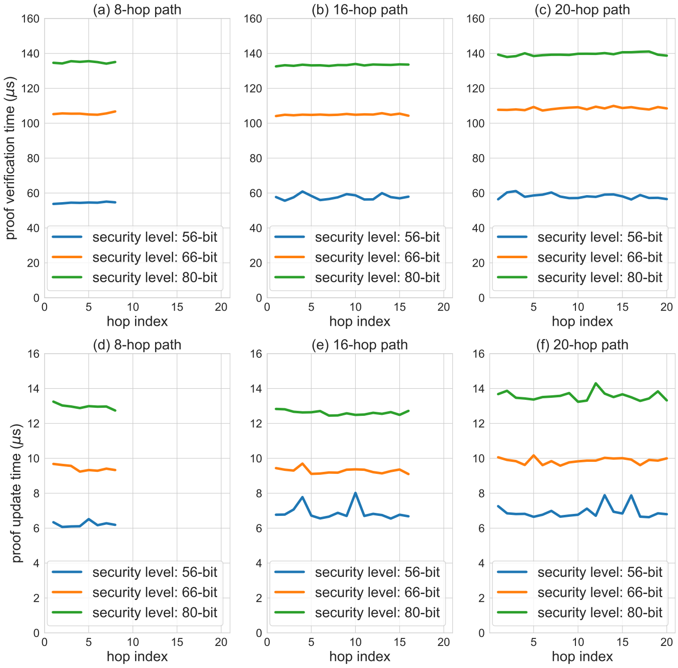


Fig. 3. Proof verification and update time with varying path lengths and security levels.

delays in packet processing and forwarding. Moreover, they also need to store other data, for example, computation primers such as ECC for ICING and a public/private key pair for OPT. OSV is similar to Atomos in this regard. It also needs to store secret keys locally. In addition, the source node of OSV needs to store an entire $n \times n$ Hadamard matrix locally, while other nodes need to store the corresponding matrix rows. This is a large storage overhead. ICING requires every router to exchange secret keys to guarantee safety, which also costs considerable storage overhead. OPT and OSV make a stronger security assumption to avoid this. Although Atomos may cause larger storage overhead, this is necessary to save bandwidth for data and increase the delivery efficiency.

TABLE IV
PROOF CREATION TIME (μS) AT THE SOURCE WITH VARYING SECURITY LEVELS. THE LEVEL IS INSENSITIVE TO PATH LENGTH.

Security Level (bit)	56	62	66	80	112	128
Creation Time (μs)	1.18	1.28	1.37	1.42	1.72	2.54

TABLE V
PROOF CREATION TIME (μS) AT THE SOURCE WITH VARYING PATH LENGTHS.

Path length (hops)	16	20
OPT Creation Time (μs)	11.5	13.2
OSV Creation Time (μs)	14.0	23.0

B. *Processing Time*

Atomos promises not only shorter proofs but also faster validation, especially for long forwarding paths. The processing time of a proof is determined by three parts: the creation time, verification time, and update time.

Creation time. As shown in Table IV, Atomos can deliver extremely fast proof creation. This is attributed to two properties of Atomos. First, using asymmetric-key encryption, Atomos enables each router to generate only one proof for all other routers. Therefore, the source need not compute different proofs for each of the en-route routers, as required by ICING. This also makes the creation time insensitive to path length. Second, we perform as much precomputation as possible to promote speedup (Section IV-G). Precomputed data need not be recomputed when processing different packets. Since the proof contains fields related to the group size p (Section IV-C), the time for proof creation increases with the security level.

Verification and update time. A router verifies a proof with Equation 3 using $\tilde{\sigma}_{i-1}$ and $\tilde{u}_{i-1}$. These two parameters are generated by the router's previous hop and are carried in the packet header. If the verification succeeds, the router updates the proof following Definition 5. Figure 3(a)-(c) report the time for proof verification given various security levels on an 8-,

TABLE VI
VERIFICATION AND UPDATE TIME (μS) FOR 16 HOPS.

Node ID	1	4	8	12	16
OPT (μs)	5.06	4.57	4.79	4.66	4.81
OSV (μs)	1.58	1.58	1.58	1.58	1.58

TABLE VII
VERIFICATION AND UPDATE TIME (μS) FOR 20 HOPS.

Node ID	1	4	8	12	16	20
OPT (μs)	5.21	4.66	4.66	4.68	4.63	4.67
OSV (μs)	2.51	2.51	2.51	2.51	2.51	2.51

16-, and 20-hop path, respectively. Figure 3(d)-(f) report the corresponding update time. A higher security level requires more processing time, and which hop a router is located at in a path has little effect. As Atomos uses a constant-size proof, each hop processes a proof of the same scale. The proof processing time is therefore relatively constant across the forwarding path. In contrast, the verification time of ICING is proportional to the path length. A router with hardware acceleration takes $(2.6x + 24.4)\mu s$ to verify a proof at an 80-bit security level, where x is the hop index of the router. Our software-based implementation of Atomos is faster when, for example, a forwarding path exceeds 15 hops.

C. Comparison with OPT and OSV

We further compare the time cost of Atomos with that of OPT and OSV, which do not enforce pairwise trust. As with the performance comparison of OPT and OSV with ICING, we expect that Atomos will be slower because pairwise trust incurs more computation for verifying proofs. However, we do not consider this a fair comparison; we provide the comparison solely to demonstrate the extent to which relaxed security can improve performance.

Creation time. As Table V shows, both OPT and OSV show sensitivity to path length, and OSV is much slower, as it needs to generate an Hadamard matrix at this stage. Both of them are far slower than Atomos, which does not require the source to compute proofs for other routers and is insensitive to path length. For example, at the same security level of 128 bits and the same path length of 16 hops, Atomos only needs 2.54 μs to create a packet, while OPT needs 11.5 μs and OSV needs 14.0 μs. This shows the benefit of the simplified precomputation of Atomos.

Verification and update time. OPT and OSV do not have the same security design as ICING and Atomos. Routers in ICING and Atomos do not need to trust each other, while OPT and OSV assume otherwise to avoid possible inefficiencies and vulnerabilities. If they followed the same security requirement as ICING and Atomos, every pair of routers would need to exchange secret keys; this would cost additional resources and decrease efficiency. This is why OPT and OSV are faster than Atomos. For a 16-hop path (Table VI), OPT takes 4.7 μs on average, and OSV only needs 1.58 μs. The results on a 20-hop path show the same trend (Table VII). In contrast, Atomos needs approximately 60 μs to process a node. However, since Atomos has a stronger security guarantee than ICING does, it can be used in a more suspicious environment where end-hosts and routers do not necessarily trust each other.

VIII. DISCUSSION

A. Router Implementation and Internet Deployment

We temporarily use Click routers [33] to emulate routers following ICING [4] and do not integrate them into the real Internet. To the best of our knowledge, all existing path validation solutions do not use real Internet environments for experiments due to conceivable challenges, although they may implement hardware routers. For example, ICING [3] is also implemented on a field-programmable gate array (FPGA) board. We find that OSV [14], [15] provides a feasible way to create a small and enforceable deployment of path validation in the Internet. Specifically, by jointly considering router implementation and Internet deployment, we can use the ExoGENI [34] platform, featuring routers across the globe. ExoGENI is an evolved global environment for network innovations (GENI) testbed to support open cloud computing and dynamic circuit fabrics [34]. Once granted access, one can deploy packet processing logics to the accessed routers.

B. Network Dynamics

The process of handling network dynamics can be divided into two phases. The first is allocating paths based on network dynamics. The second is validating packets forwarded along the newly allocated paths. The latter focuses on path validation designs themselves. That is, no matter how forwarding paths are allocated, existing path validation protocols simply take the allocated paths as input and enforce packet forwarding along them. In other words, handling network dynamics is not the technical focus of path validation. Instead, it focuses more on making routing policies, especially for the first phase—path allocation. This should be orthogonal to path validation.

Path validation has two ways to react to network dynamics during packet forwarding. Specifically, if the allocated forwarding path expires when some packets are still en route on it, we can either 1) allow the expired forwarding path to be valid for the en-route packets as well the current session or 2) let the destination discard the en-route packets upon receiving them and let the source retransmit them. The first method is straightforward to enforce. When the current session ends and the next session begins, the source simply follows the new forwarding path for building path proofs. The packets will also be forwarded along the new path. The second method is slightly more complex, since it involves synchronization between the source and destination as well as packet retransmission. Specifically, once the destination is notified of a newly allocated path, it discards received packets that follow the original path. If the transmission protocol does not require the destination to send acknowledgments of non-discarded packets to the source, the destination may use additional packet-drop feedback. Upon receiving such feedback, the source retransmits the dropped packets embedded with new path proofs.

Clearly, path changes due to network dynamics impose overhead on path validation. Short sessions are usually recommended to limit the impact of network dynamics on performance [7]. This is because, when network dynamics enforce path changes, end-hosts can quickly adapt to new paths. A short gap between the time a new path is issued and the time it takes effect has little impact on performance.

C. Applicability of Path Validation

For any path validation solution [3], [4], [14]–[16], efficiency enhancements bring greater benefit to networks with relatively long forwarding paths on average. For end-to-end communication, especially for routers between end-hosts, it is common that a forwarding path spans from several to dozens of hops. The evaluation results in Section VII show that Atomos becomes more efficient than ICING when the forwarding path exceeds several hops (e.g., 3 or 4).

However, the efficiency benefit of a path validation solution may not be obvious for services with extremely short path lengths. Consider the content delivery network (CDN), for example. To speed up content distribution, CDN providers usually deploy many content delivery servers across various ISPs. These servers hold replicated content so that users can obtain the requested content from the nearest content delivery server. According to a recent measurement study [35], more than 60% of end-user prefixes are directly connected to large content providers. In other words, these end-users are just one hop away from the content delivery servers. We consider path validation to be unsuitable for such scenarios. Traditional authentication suffices for the security needs of such one-hop end-to-end communication.

IX. Conclusion

We have studied how to overcome the efficiency barriers of network path validation using constant-size proofs. Existing solutions use linear-scale proofs in terms of path length. The longer a forwarding path is, the larger the proofs that routers need to handle and packets need to carry. This brings heavy overhead in terms of both packet-header space and processing time. To address these limitations, we construct a noncommutative homomorphic asymmetric-key encryption scheme that offers constant-size validation proofs. We design and implement Atomos based on the proposed encryption scheme. Various optimization techniques are explored to gain efficiency without sacrificing security. Both analytical and experimental results show that Atomos provides not only shorter proofs but also faster validation than existing solutions. In future work, we plan to augment Atomos with hardware acceleration [3] and adapt it to multipath routing [36].

Acknowledgment

This work is supported in part by the National Science Foundation of China under Grant 61402404, the Natural Science Foundation of Zhejiang Province under grant No. LY19F020050, the National Natural Science Foundation of under Grant No. 61772236, Zhejiang Key RD Plan under Grant No. 2019C03133, the Alibaba-Zhejiang University Joint Institute of Frontier Technologies, Research Institute of Cyberspace Governance in Zhejiang University, and Leading Innovative and Entrepreneur Team Introduction Program of Zhejiang. The authors would also like to thank the Editors and Reviewers of IEEE Transactions on Forensics and Security and AJE Editors for their review efforts and helpful feedback.

References

[1] IRTF, "Path aware networking research group panrg," 2019. [Online]. Available: https://irtf.org/panrg.
[2] IETF, "Path aware networking rg (panrg)," 2019. [Online]. Available: https://datatracker.ietf.org/rg/panrg/about/.
[3] J. Naous, M. Walfish, A. Nicolosi, D. Mazières, M. Miller, and A. Seehra, "Verifying and enforcing network paths with icing," in *CoNEXT*, 2011.
[4] T. H.-J. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig, "Lightweight source authentication and path validation," in *ACM SIGCOMM*, vol. 44, no. 4, 2014, pp. 271–282.
[5] D. Barrera, L. Chuat, A. Perrig, R. M. Reischuk, and P. Szalachowski, "The scion internet architecture," *Communications of the ACM*, 2017.
[6] B. Trammell, J.-P. Smith, and A. Perrig, "Adding path awareness to the internet architecture," *IEEE Internet Computing*, vol. 22, no. 2, pp. 96–102, 2018.
[7] K. Bu, Y. Yang, A. Laird, J. Luo, Y. Li, and K. Ren, "What's (not) validating network paths: A survey," *arXiv:1804.03385*, 2018.
[8] E. Zmijewski, "You can't get there from here." 2008. [Online]. Available: https://dyn.com/blog/you-cant-get-there-from-here-1/
[9] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig, "Hornet: High-speed onion routing at the network layer," in *CCS*, 2015.
[10] Z. Liu, H. Jin, Y.-C. Hu, and M. Bailey, "Middlepolice: Toward enforcing destination-defined policies in the middle of the internet," in *ACM CCS*, 2016, pp. 1268–1279.
[11] K. L. Calvert, J. Griffioen, and L. Poutievski, "Separating routing and forwarding: A clean-slate network layer design," in *IEEE BROADNETS*, 2007, pp. 261–270.
[12] D. Levin, Y. Lee, L. Valenta, Z. Li, V. Lai, C. Lumezanu, N. Spring, and B. Bhattacharjee, "Alibi routing," in *ACM SIGCOMM*, 2015.
[13] J. Katz and A. Lindell, "Aggregate message authentication codes," *Topics in Cryptology–CT-RSA 2008*, pp. 155–169, 2008.
[14] H. Cai and T. Wolf, "Source authentication and path validation with orthogonal network capabilities," in *INFOCOM WKSHPS*, 2015.
[15] ——, "Source authentication and path validation in networks using orthogonal sequences," in *IEEE ICCCN*, 2016, pp. 1–10.
[16] B. Wu, K. Xu, Q. Li, Z. Liu, Y.-C. Hu, M. J. Reed, M. Shenk, and F. Yang, "Enabling efficient source and path verification via probabilistic packet marking," in *IWQoS*, 2018.
[17] P. K. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, D. Wetherall *et al.*, "Improving the reliability of internet paths with one-hop source routing." in *OSDI*, 2004.
[18] Z. Li, D. Levin, N. Spring, and B. Bhattacharjee, "Internet anycast: performance, problems, & potential." in *ACM SIGCOMM*, 2018.
[19] A. Fei, G. Pei, R. Liu, and L. Zhang, "Measurements on delay and hop-count of the internet," in *GLOBECOM*, 1998.
[20] F. Begtasevic and P. Van Mieghem, "Measurements of the hopcount in internet," in *PAM*, 2001.
[21] V. Paxson, "End-to-end routing behavior in the internet," in *ACM SIGCOMM*, 1996, pp. 25–38.
[22] B. Sengupta, Y. Li, K. Bu, and R. H. Deng, "Privacy-preserving network path validation," Cryptology ePrint Archive, Report 2019/407, 2019, https://eprint.iacr.org/2019/407.
[23] R. Motwani and P. Raghavan, *Randomized algorithms*. Cambridge university press, 1995.
[24] Keylength - NIST Report on Cryptographic Key Length and Cryptoperiod (2016). [Online]. Available: https://www.keylength.com/en/4/
[25] Diffie–Hellman key exchange - Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange
[26] General number field sieve - Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/General_number_field_sieve
[27] C. NIST, "Implementation guidance for fips pub 140-2 and the cryptographic module validation program, may. 2019."
[28] T. Kleinjung, "Computation of a 768-bit prime field discrete logarithm," 2016. [Online]. Available: https://listserv.nodak.edu/cgi-bin/wa.exe?A2=NMBRTHRY;a0c66b63.1606
[29] T. Lee, C. Pappas, A. Perrig, V. Gligor, and Y.-C. Hu, "The case for in-network replay suppression," in *ACM AsiaCCS*, 2017, pp. 862–873.
[30] C. Basescu, Y.-H. Lin, H. Zhang, and A. Perrig, "High-speed inter-domain fault localization," in *IEEE S&P*, 2016, pp. 859–877.
[31] K. Bu, Y. Yang, Z. Guo, Y. Yang, X. Li, and S. Zhang, "Flowcloak: Defeating middlebox-bypass attacks in software-defined networking," in *IEEE INFOCOM*, 2018, pp. 396–404.
[32] B. Wu, K. Xu, Q. Li, B. Liu, S. Ren, F. Yang, M. Shen, and K. Ren, "Rfl: Robust fault localization on unreliable communication channels," *Computer Networks*, vol. 158, pp. 158–174, 2019.

[33] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.

[34] I. Baldine, Y. Xin, A. Mandal, P. Ruth, C. Heerman, and J. Chase, "Exogeni: A multi-domain infrastructure-as-a-service testbed," in *Testbeds and Research Infrastructure. Development of Networks and Communities*, 2012, pp. 97–113.

[35] Y.-C. Chiu, B. Schlinker, A. B. Radhakrishnan, E. Katz-Bassett, and R. Govindan, "Are we one hop away from a better internet?" in *IMC*, 2015, pp. 523–529.

[36] W. Xu and J. Rexford, "Miro: multi-path interdomain routing," in *ACM SIGCOMM*, 2006.

Appendix

A. Proofs of Function Properties

Lemma 1. *Both the additive magma $(\mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}, \star)$ and the multiplicative magma $(\mathbb{Z}_p^* \times \mathbb{Z}_p^*, \odot)$ are noncommutative given operands (a_1, b_1) and (a_2, b_2) with $a_1 \neq a_2$.*

Proof. The additive magma is noncommutative if $(a_1, b_1) \neq (a_2, b_2)$, and we have $(a_1, b_1) \star (a_2, b_2) \neq (a_2, b_2) \star (a_1, b_1)$ for some values of a_1 and a_2. According to the definition of $\star$ in Definition 1, we have:

$$(a_1, b_1) \star (a_2, b_2) = (a_1 + a_2, a_1 + b_1 + b_2). \tag{13}$$

$$(a_2, b_2) \star (a_1, b_1) = (a_1 + a_2, a_2 + b_1 + b_2). \tag{14}$$

Given the constraint of $a_1 \neq a_2$ in the statement, we can derive the inequality between the right sides of Equation 13 and Equation 14 because $(a_1+b_1+b_2) \neq (a_2+b_1+b_2)$. Therefore, we demonstrate that $(a_1, b_1)\star(a_2, b_2) \neq (a_2, b_2)\star(a_1, b_1)$ and prove the noncommutativity of the additive magma.

Similarly, we prove the noncommutativity of the multiplicative magma $(\mathbb{Z}_p^* \times \mathbb{Z}_p^*, \odot)$ as follows:

$$(a_1, b_1) \odot (a_2, b_2) = (a_1 a_2, a_1 b_1 b_2). \tag{15}$$

$$(a_2, b_2) \odot (a_1, b_1) = (a_1 a_2, a_2 b_1 b_2). \tag{16}$$

Given the constraint of $a_1 \neq a_2$, we can derive the inequality between the right sides of Equation 15 and Equation 16 because $a_1 b_1 b_2 \neq a_2 b_1 b_2$. Therefore, we demonstrate that $(a_1, b_1) \odot (a_2, b_2) \neq (a_2, b_2) \odot (a_1, b_1)$ and prove the noncommutativity of the multiplicative magma. □

Lemma 2. *The function F defined in Definition 3 is a noncommutative homomorphic function.*

Proof. First, we prove the homomorphism of function F. A homomorphic function h should satisfy the property:

$$h(x \ \mathrm{op}_1 \ y) = h(x) \ \mathrm{op}_2 \ h(y),$$

where x and y denote two inputs and op_1 and op_2 denote two operations that may or may not be identical. Based on the definition of F, we prove its homomorphism using the fact that it satisfies the following equation:

$$F((a_1, b_1) \star (a_2, b_2)) = F(a_2, b_2) \odot F(a_1, b_1). \tag{17}$$

Equation 17 can be derived as follows:

$$\begin{aligned} F((a_1, b_1) \star (a_2, b_2)) &= F(a_1 + a_2, a_1 + b_1 + b_2) \\ &= (g^{a_1+a_2}, g^{a_1+b_1+b_2}) \\ &= F(a_1, b_1) \odot F(a_2, b_2). \end{aligned} \tag{18}$$

Second, we prove the noncommutativity of function F by showing that $F((a_1, b_1) \star (a_2, b_2)) \neq F((a_2, b_2) \star (a_1, b_1))$. Since $F((a_1, b_1)\star(a_2, b_2))$ was already derived in Equation 18, we now derive $F((a_2, b_2) \star (a_1, b_1))$ as follows:

$$\begin{aligned} F((a_2, b_2) \star (a_1, b_1)) &= F(a_2 + a_1, a_2 + b_2 + b_1) \\ &= (g^{a_2+a_1}, g^{a_2+b_2+b_1}) \\ &= (g^{a_2} g^{a_1}, g^{a_2} g^{b_2} g^{b_1}) \\ &= F(a_2, b_2) \odot F(a_1, b_1). \end{aligned} \tag{19}$$

By Lemma 1, the multiplicative magma $\mathbb{Z}_p^* \times \mathbb{Z}_p^*, \odot)$ is noncommutative. This directly yields the inequality between the right sides of Equation 18 and Equation 19, that is, $F(a_1, b_1) \odot F(a_2, b_2) \neq F(a_2, b_2) \odot F(a_1, b_1)$. This further yields the following inequality between their left sides:

$$F((a_2, b_2) \star (a_1, b_1)) \neq F((a_2, b_2) \star (a_1, b_1)). \tag{20}$$

With Equation 20, we prove the noncommutativity of F. □

From Lemma 1 and Lemma 2, we have the following result:

Theorem 1. *A noncommutative homomorphic asymmetric-key encryption scheme can be constructed from the additive magma $(\mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}, \star)$, multiplicative magma $(\mathbb{Z}_p^* \times \mathbb{Z}_p^*, \odot)$, and homomorphic function $F : (\mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}, \star) \to (\mathbb{Z}_p^* \times \mathbb{Z}_p^*, \odot)$.*

Proof. We construct an encryption scheme as follows:

- A proof $(a_i, b_i) \in \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}$ is created by each user (e.g., a router) r_i.
- To aggregate the proofs of $r_1, ..., r_i$, $(\tilde{a}_i, \tilde{b}_i) = (a_1, b_1) \star ... \star (a_i, b_i)$ is computed.
- To verify the aggregated proof $(\tilde{a}_i, \tilde{b}_i)$, the equality $F((\tilde{a}_i, \tilde{b}_i)) = F((a_1, b_1)) \odot ... \odot F((a_i, b_i))$ is checked to see whether it holds.

The construction yields an asymmetric-key encryption scheme with $F((a_i, b_i)) = (f(a_i), f(b_i)) = (g^{a_i}, g^{b_i})$ as the public key and (a_i, b_i) as the secret key of r_i. From Lemma 1 and Lemma 2, the encryption scheme is noncommutative homomorphic if integers $a_1, ..., a_i$ are selected such that for any two integers a_{j_1} and a_{j_2} with $1 \leq j_1 \neq j_2 \leq i$, we have $a_{j_1} \neq a_{j_2} \bmod (p-1)$. This can be easily satisfied. For example, routers have different identifiers, and such router-specific information can be used to form the integers $a_1, ..., a_i$. □

B. Attack on the Inverse Hash

An inverse of h in $\mathbb{Z}_{p-1}$ can be computed in polynomial time in $b(p)$, the number of bits in p. We sketch the proof by constructing a polynomial-time attack using $H(\cdot)$ with an inverse in $\mathbb{Z}_{p-1}$. The generic attack goal is to have router r_i be the attacker. Upon receiving a valid proof $(\tilde{r}_{i-1}, \tilde{\sigma}_{i-1})||\tilde{u}_{i-1}$ from its previous-hop router r_{i-1}, r_i can follow the attack to forge a valid proof to its next-hop router r_{i+1}. Specifically, the attack consists of five steps.

Step 1: Select an arbitrary $\alpha \in \mathbb{Z}_{p-1}$.

Step 2: Compute the inverse $(H(h))^{-1}$ of $H(h)$ in $\mathbb{Z}_{p-1}$.

Step 3: Compute $\beta = (g^{\alpha}(k_i^{\mathrm{p}})^{-1})^{(H(h))^{-1}}$.

Step 4: Compute $\acute{\sigma}_i = \alpha + \tilde{\sigma}_{i-1} + \tilde{r}_{i-1}$, $\acute{u}_i = \beta \tilde{u}_{i-1}$.

Step 5: Compute $\tilde{r}_i$ using $\tilde{r}_{i-1}$ following Equation 2.

The attacker r_i then sends the forged proof $(\tilde{r}_{i-1}, \acute{\sigma}_i)||\acute{u}_i$ to r_{i+1}. By Theorem 2, the forged proof can pass verification if:

$$g^{\acute{\sigma}_i} \bmod p = g^{\delta(r_i)}(\prod_{1 \leq j \leq i} k_j^{\mathrm{p}})(\acute{u}_i)^{H(h)} \bmod p. \tag{21}$$

The left side of Equation 21 can be derived as:

$$g^{\acute{\sigma}_i} = g^{\alpha + \tilde{\sigma}_{i-1} + \tilde{r}_{i-1}} = g^{\alpha + \sum_{1 \leq j \leq i-1} \sigma_j + \delta(r_i)}.$$

The right side of Equation 21 can be derived as:

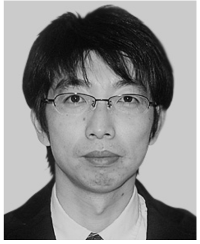
$$g^{\delta(r_i)}(\prod_{1 \leq j \leq i} k_j^p)(\acute{u}_i)^{H(h)} = g^{\delta(r_i)}(\prod_{1 \leq j \leq i-1} k_j^p) k_i^p (\beta \tilde{u}_{i-1})^{H(h)}$$
$$= g^{\delta(r_i)}(\prod_{1 \leq j \leq i-1} k_j^p)(\tilde{u}_{i-1})^{H(h)} k_i^p \beta^{H(h)}$$
$$= g^{\delta(r_i) + \alpha + \sum_{1 \leq j \leq i-1} \sigma_j}.$$

The left side is equal to the right side. This proves that Equation 21 holds and the forged proof can pass verification.

Anxiao He is currently pursuing the undergraduate degree with the College of Computer Science and Technology, Zhejiang University. His research interests include network security.

Kai Bu received the B.Sc. and M.Sc. degrees in computer science from the Nanjing University of Posts and Telecommunications, Nanjing, China, in 2006 and 2009, respectively, and the Ph.D. degree in computer science from The Hong Kong Polytechnic University, Hong Kong, in 2013. He is currently an Associate Professor with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. His research interests include network security and computer architecture. He is a member of the ACM, the IEE, and the CCF. He was a recipient of the Best Paper Award of IEEE/IFIP EUC 2011 and the Best Paper Nominee of IEEE ICDCS 2016.

Yucong Li is currently pursuing the undergraduate degree with the College of Computer Science and Technology, Zhejiang University. His research interests include network security.



Eikoh Chida received his B.Eng. degree in information engineering and M.S. and Ph.D. degrees in information science from Tohoku University, Japan, in 1993, 1995 and 1998, respectively. He is a professor the Department of Electrical Engineering, Ichinoseki National College of Technology. His research interests include cryptology and related mathematics.

Qianping Gu received his Ph.D. degree in computer science from Tohoku University, Japan. He is a professor at the School of Computing Science, Simon Fraser University, Canada. His research interests include algorithms and computation, network communications and algorithms, parallel and distributed computing.

Kui Ren received the Ph.D. degree from the Worcester Polytechnic Institute. He is currently a Professor with the Institute of Cyberspace Research, Zhejiang University, and the Director of the UbiSeC Laboratory, State University of New York at Buffalo (UB). He has published 200 papers in peer-reviewed journals and conferences. His current research interest spans cloud and outsourcing security, wireless and wearable systems security, and mobile sensing and crowdsourcing. He is a Distinguished Lecturer of the IEEE, a member of ACM, and a past Board Member of the Internet Privacy Task Force, State of Illinois. He received several best paper awards, including IEEE ICDCS 2017, IWQoS 2017, and ICNP 2011. He received the IEEE CISTC Technical Recognition Award in 2017, the UB Exceptional Scholar Award for Sustained Achievement in 2016, the UB SEAS Senior Researcher of the Year Award in 2015, the Sigma Xi/IIT Research Excellence Award in 2012, and the NSF CAREER Award in 2011. He currently serves on the editorial boards of the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, the IEEE TRANSACTIONS ON SERVICE COMPUTING, the IEEE TRANSACTIONS ON MOBILE COMPUTING, the IEEE WIRELESS COMMUNICATIONS, the IEEE INTERNET OF THINGS JOURNAL, and the SpingerBriefs on Cyber Security Systems and Networks.