Hindawi
Security and Communication Networks
Volume 2022, Article ID 1942097, 7 pages
https://doi.org/10.1155/2022/1942097

Research Article

An Identity-Based Encryption Method for SDN-Enabled Source Routing Systems

Bander Alzahrani[1] **and Shehzad Ashraf Chaudhry**[2]

[1]*College of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia*
[2]*Department of Computer Engineering, Faculty of Engineering and Architecture, Istanbul Gelisim University, Istanbul, Turkey*

Correspondence should be addressed to Bander Alzahrani; baalzahrani@kau.edu.sa and
Shehzad Ashraf Chaudhry; sashraf@gelisim.edu.tr

Received 6 January 2022; Accepted 16 March 2022; Published 13 April 2022

Academic Editor: Muhammad Arif

Copyright © 2022 Bander Alzahrani and Shehzad Ashraf Chaudhry. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In this study, we consider endpoints communicating over a software-defined networking (SDN)-based architecture using source routing, i.e., packets are routed through a path selected by the packet sender, and we provide a security solution that enforces the selected path. In particular, our solution allows a sender to select the path that a packet should go through using a constant-size cryptographic construction which is referred to as the authenticator. A recipient can examine an authenticator and verify that the received packet has followed the path selected by the sender. Additionally, any intermediate "programmable" switch can verify whether or not it is included in the path of a packet. Our solution can be used even for paths that include multiple recipients (e.g., multicast paths), as well as multiple parallel paths (e.g., multipath transmissions). We implement our solution by leveraging identity-based encryption (IBE), so it can be used by any sender that knows the identifiers of the links that compose the desired path, i.e., information that the sender usually already knows as part of the source routing protocol. Our solution is realistic since it can be implemented over a variety of platforms with tolerable overhead.

1. Introduction

Software-defined networking (SDN) is a popular communication technology, which is commonly adopted for intradomain traffic engineering. SDN uses programmable switches that can dynamically make packet switching decisions, as well as do changes to forwarded packets. Those switches can act independently based on predefined forwarding rules or by working with a centralized component called the SDN controller. This entity, among other things, is responsible for configuring switches, as well as helping them to make forwarding decisions.

SDN facilitates the deployment of custom routing protocols, which are leveraged by conventional or even by Next-Generation Internet architecture. For example, clean-slate Internet architecture paradigms, such as the Information-Centric Networking (ICN) [1], can be overlayed on top of SDN (see, for example, [2]), enabling this way the experimentation and integration with existing systems.

A common collection of routing protocols that benefit from SDN is known as source routing protocols. Using such kind of protocols, a sender can choose the path a packet should travel. The path creation procedure is dependent on several factors, including improved quality of service, network cache occupancy, and security constraints, among others. In the great majority of cases, however, the senders trust the programmable switches to perform the necessary switching decisions, ensuring that the path selection is followed and the final path is used. In this study, we design and implement a cryptographic mechanism that allows packet receivers to check that a packet has been forwarded through the intended path. Our solution can be used in cases where untrusted or even opportunistic programmable switches are considered (e.g., SDN over mobile devices acting as programmable switches). This is an emerging pattern which is considered in modern, nontraditional networks, such as vehicular networks [3], mobile ad-hoc networks [4], or even IoT technology [5]. An important

aspect of this paradigm is that it allows novel, effective security mechanisms [6]. For example, by combining SDN with other related technologies, efficient security solutions are enabled, such as intrusion detection systems, firewalls, authentication, and authorization mechanisms [6]. In these environments, our solution brings great value, since it allows detection of malfunctioning and/or malicious programmable switches.

From a high-level perspective, our solution operates as follows. We consider an SDN network where each link is uniquely identified by an identifier. The senders are aware of the network topology (hence, they know the corresponding link identifiers), and they are able to calculate network paths, composed of multiple links, using a source routing protocol. Each path is expressed with a path identifier inserted at the packet header and switches are able to make forwarding decisions (autonomously or with the help of the controller) using that path identifier. Additionally, senders include in the packet header an "authenticator," i.e., a cryptographic construction. Every switch receives a packet, extracts the in-packet authenticator, performs a small computation, and, as a result, alters the authenticator based on a prespecified algorithm, which can be executed correctly only by the switches alongside the path selected by the sender. The purpose of the authenticator is to allow (i) legitimate switches to identify if the arrived packet comes from a link that is not part of the delivery tree/path and (ii) receivers to check that the packet delivered through the selected tree.

Our solution does not require from the senders of the packet to know any additional information apart for the link identifiers, i.e., information already known due to the source routing protocol. Furthermore, our solution supports paths with multiple recipients (e.g., multicast), as well as paths that use multiple links in parallel (e.g., multipath). With our solution, the packet recipient does not have to be aware neither of link identifiers nor of any information related to the path; in other words, the only information a packet recipient learns is that "the packet followed the correct path." In case of multicast, each recipient can independently take the action about the arrived packet.

The remainder of this study is organized as follows. In Section 2, we present the design of our solution. In Section 3, we present the implementation of our solution, based on a particular IBE algorithm, as well as its performance and security evaluation. We discuss related work in Section 4, and we conclude our paper in Section 5.

2. Design

2.1. System Entities and Notation. The main building block of our solution is identity-based encryption (IBE). IBE is an asymmetric encryption scheme where arbitrary strings can be used as public keys [7]. Public keys, referred to as identities in the context of IBE, are associated with a secret key. In the following, we refer to the secret key that corresponds to an identity "A" as SK_A and to the encryption of a message "M" using "A" as the public key as $Enc_A(M)$. Therefore, $Enc_A(M)$ can only be decrypted using SK_A. Secret keys are generated by a centralized trusted entity known as the "Private Key Generator" (PKG). In our solution, the role of the PKG is held by the SDN controller.

Our solution considers a typical SDN architecture, which is composed of an SDN controller, programmable switches, senders, and recipients. Each link in the network is identified by a unidirectional identifier referred to as link_{ID}. Our solution is agnostic to the particular source routing protocol; nevertheless, it has been designed using the solution presented in [8] as a reference architecture. That solution allows packet senders to select the path that a packet should follow by encoding link_{ID} that forms the path in a Bloom filter [9]. Additionally, this solution leverages standard OpenFlow [10] forwarding rules in order to achieve Bloom filter-based forwarding without any modification to the programmable switches. An example of the solution of [8] is illustrated in Figure 1. In this example, a "sender" is connected to three other nodes. The "sender" wants to send a message to nodes "A" and "C." Therefore, it constructs a Bloom filter that includes the link identifiers of the appropriate links (illustrated with the green bold line then figure).

The SDN controller is always aware of the network topology; i.e., for each programmable switch, it knows its neighbors. This can be achieved for example using a topology discovery protocol or even out-of-band using a configuration file. Similarly, our solution considers that senders are aware of the network topology and the corresponding link identifiers and can construct arbitrary packet forwarding paths. Furthermore, it is assumed that senders and recipients have already created an end-to-end encryption channel (e.g., using TLS); hence, the transmitted data are secured from any possible disclosure or change.

2.2. System Setup. Our system considers a setup phase during which link_{ID} identifiers are created. Each link_{ID} is selected by the SDN controller and all link_{ID} identifiers are treated as an IBE public key; the corresponding secret keys are also generated by the controller. During this phase, for each outgoing link, each switch is configured (by the controller) with the corresponding SK_{linkID}, i.e., an IBE secret key.

This setup process is illustrated in Figure 2. In this figure, a switch, switch *A*, detects a new link with another switch, switch *B*. It communicates with the SDN controller (message 1), and it receives back a link identifier and the corresponding secret key (message 2). The switch stores the received secret key and the link becomes active.

When all switches communicate with the controller, all links will be associated with an identifier, and all switches will know the corresponding secret key. For example, in Figure 3, switch A is configured with SK_{0001}, switch B with SK_{0010}, and switch C with SK_{0100}.

2.3. Path Authenticator Construction and Usage. A sender can construct an authenticator for a unicast forwarding path composed of links 1 to m as follows (Figure 3):

(i) Let IDx be link_{ID} of the xth link, i.e., link_{ID} of the first link is ID1 and link_{ID} of the second link is ID2

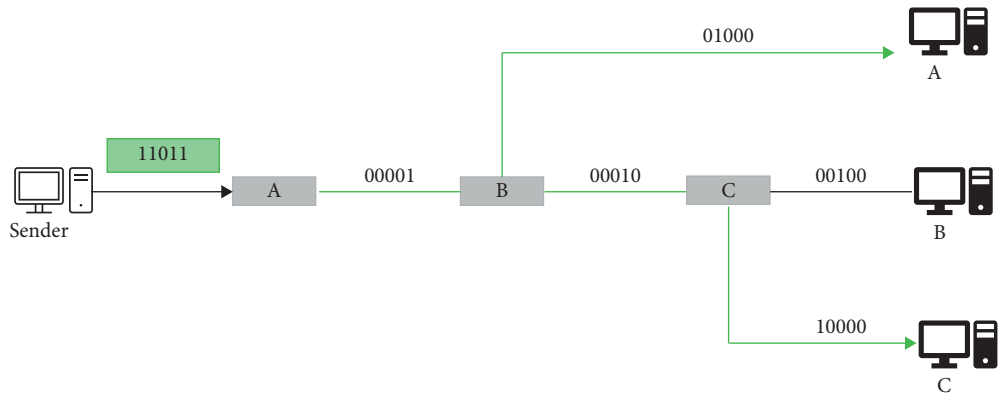

Figure 1: Bloom filter-based forwarding.

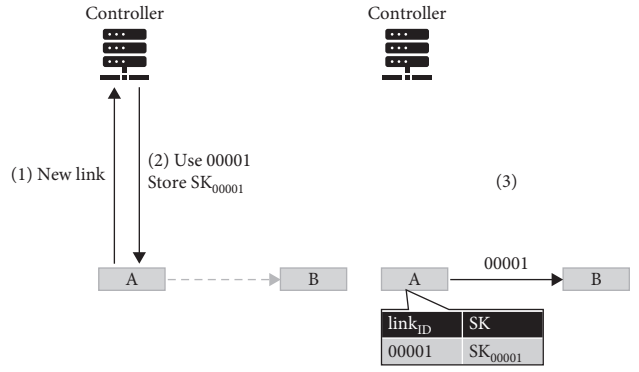

Figure 2: SK distribution.

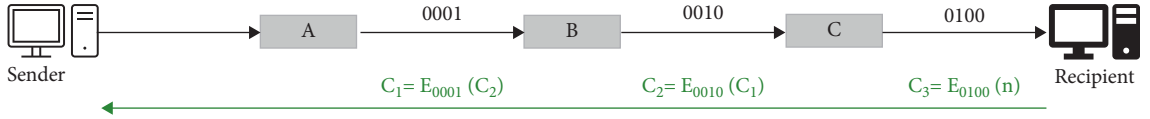

Figure 3: Constructing an authenticator for a unicast path.

(ii) The sender creates a nonce (n)

(iii) Beginning from the link m of the delivery tree, $C_m = Enc_{IDM}(n)$ is created by the sender, i.e., an IBE encryption of the nonce using the $link_{ID}$ as the public key

(iv) Then, the sender creates $C_{m-1} = Enc_{IDM-1}(C_m)$, $C_{m-2} = Enc_{IDM-2}(C_{m-1})$, . . ., $C_1 = Enc_{ID1}(C_2)$

(v) C_1 is the path authenticator

The sender includes the path authenticator in the transmitted packet header. Furthermore, it includes the nonce n in the (encrypted) packet payload. From a high-level perspective, the authenticator is the nonce encrypted with m layers of encryption (where m is the path length). Each switch that forwards a packet tries to "remove" the topmost encryption layer as follows:

(i) The switch determines link_{ID} of the next-hop link.

(ii) Then, it retrieves the corresponding SK.

(iii) It tries to decrypt the authenticator using the retrieved key.

(iv) If the decryption succeeds it forwards the packet including the new authenticator. Otherwise, the packet has been forwarded through a wrong switch and it is rejected.

If the packet is forwarded through all the appropriate switches, the final value of the authenticator will be the nonce n. Therefore, the receiver has simply to check if n matches the nonce included in the encrypted payload. This process is illustrated in Figure 4. As it can be seen, the sender sends the authenticator and the encrypted payload, which contains the data and the nonce n. As this packet is forwarded, the authenticator is being replaced. Finally, the last switch includes in the authenticator field of the nonce n.

2.4. Authenticator for Multiple Receivers. For paths that include "branches" (e.g., the multicast path of Figure 5), the authenticator construction is modified as follows. The sender constructs an authenticator for each branch. At the point of the path where two or more branches are merged, the sender creates a new authenticator which is the IBE

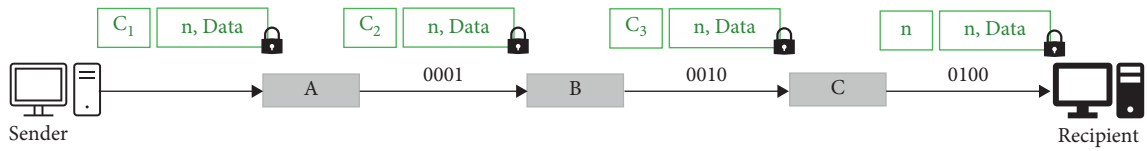

Figure 4: The forwarding process of the authenticator.

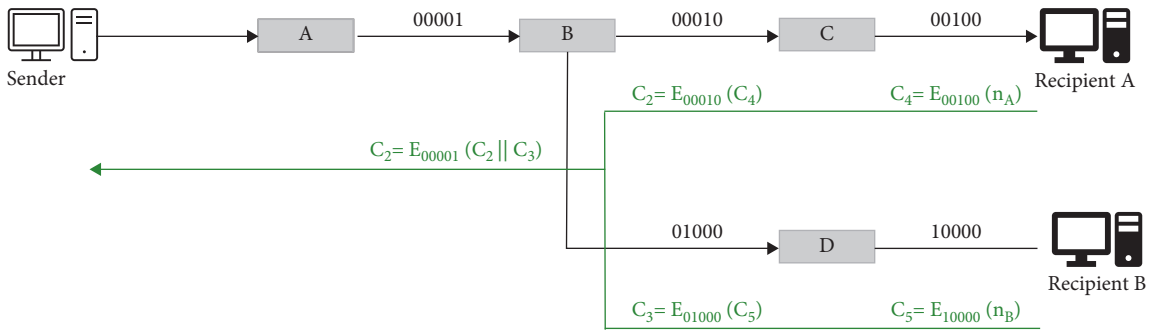

Figure 5: Constructing an authenticator for a path with branches.

encryption of the concatenation of the corresponding branch authenticators. For example, Figure 5 illustrates a path that includes two branches. The sender creates an authenticator for each branch, i.e., $C2$ and $C4$. At switch B, the two paths are encoded into a single link with $link_{ID} = 0001$; hence, the sender creates the authenticator $C5 = \mathrm{Enc}_{ID5}(C2 \| C4)$, where $\|$ denotes concatenation.

Following the reverse process, if a packet is forwarded correctly when it reaches switch B, the authenticator will be of the form $C2 \| C4$. Then, the switch B will split the authenticator into two parts (i.e., $C2$ and C4) and will try to decrypt each part with SK that corresponds to each next-hop link, and if the decryption process is successful, it will forward the corresponding authenticator to the appropriate branch. It should be noted that the sender can use different nonce for each recipient.

3. Implementation and Evaluations

3.1. Implementation. We implemented our solution using the "BB2 IBE scheme" presented in [7]. The used IBE scheme is based on bilinear groups and maps. A map is a function that maps two elements belonging to two cyclic groups $G1$ and $G2$, of prime order p, into an element of another group Gt. $G1$ and $G2$ can be the same (so the rest of this section will be simply referred to as G), and they are elements of a "pairing friendly" elliptic curve. As it will be seen in the rest of this section, all cryptographic operations have as input and outputs, numbers that are elements of G, or elements of Gt, or elements of Zp (where p is the order of G). For this reason, we consider three hash functions: $h1$ that maps any bit string to an element of G, $h2$ that maps any bit string to an element of Gt, and $h3$ that maps any bit string to an element of Zp.

The SDN controller in our solution also holds the role of the private key generator (PKG). Initially, the SDN controller generates a master secret key (MSK) and some system parameters (SP), as described in [7]. The MSK is used for generating all subsequent secret keys, and thus, it is kept secret by the SDN controller. Using an out-of-band mechanism, all programmable switches and end users learn the system parameters SP. These parameters are used as an input to both encryption and decryption algorithms, and they are public and PKG/controller specific. Since SP are public, they can be publicly published.

Whenever a programmable switch joins the network, it initially contacts the SDN controller. The controller selects a $link_{ID}$ for each outgoing link of the switch, it generates the corresponding IBE secret keys $SK_{\mathrm{link}ID}$ and securely transmits them to the switch. The key derivation algorithm (defined in [7]) requires as input an element of Zp; for this reason, the SDN controller uses $h3(\mathrm{link}_{ID})$ as an input to this algorithm. $SK_{\mathrm{link}ID}$ can be deleted from the SDN controller (since the controller can regenerate any key any time it wants), and it is safely kept by the programmable switch. $SK_{\mathrm{link}ID}$ is used for decrypting corresponding IBE ciphertexts.

The encryption algorithm defined in [7] accepts the plaintext, the target identifier, and a random number r. The formed ciphertext includes three parameters $\{A, B, C\}$. A is an element of Gt, and its value depends on the input plaintext, B is an element of G and its value depends on the random number r, and C is also an element of G and its value depends on a number $C\prime$ raised to the power of the target identifier. $C\prime$ is an element of G and its value depends on r. Therefore, two encryptions of two different plaintexts, using the same random r, will output the same B and $C\prime$ values. We leverage this property in order to construct constant-size authenticators.

Any time an end-user terminal wishes to create an authenticator, it follows the process described in Section 2. The encryption algorithm defined in [7] requires that plaintexts are elements of the group Gt; for this reason, the terminal selects at random an element of this group to hold

the role of the nonce n. Similarly, the terminal selects a random number r which is required by the encryption algorithm. The authenticator will be created after a series of encryptions that will use the same random number r. The first encryption uses as plaintext the nonce n and as target identifier the identifier of the mth link, i.e., $C_m = Enc_{IDm}(n)$. It is reminded that C_m will be of the form $\{A, B, C\}$. The second encryption uses as a plaintext the number A of the ciphertext produced by the first encryption and as the target identifier the identifier of the $(m-1)$th link, i.e., $C(m-1) = Enc_{IDm-1}(C_m(A))$. Similarly, $C(m-2) = Enc_{IDm-2}(C_{m-1}(A))$, ..., $C_1 = Enc_{ID1}(C_2(A))$. The path authenticator is C_1, A, and C'. This process is illustrated in Figure 6. As it can be seen, the sender starts by encrypting the nonce n and proceeds with encrypting As, always using the same random number r.

The path authenticator is included in the transmitted packet payload. All programmable switches that forward a packet decrypt the authenticator, by raising $C\prime$ to the identifier of the link in which the packet is forwarded and by using the IBE decrypt algorithm described in [7].

3.2. Evaluation

3.2.1. Performance Evaluation. The size of the the cryptographic parameters is determined based on the selected group G (which is also associated with Gt). Initially, we present some analytical results, and then, we present some numeric results based on a specific pairing friendly elliptic curve. Let S_g be the size in bits of an element of G and S_{gt} be the size in bits of an element in GT. Table 1 shows the size of some main parameters used in our solution.

For the type A elliptic curve of the PBC library [11], S_g = 512bits and S_{gt} = 1024bits. Therefore, using these settings, the size of an authenticator for a unicast path is 2048 bits. When it comes to a multicast delivery tree, the size of the authenticator for the shared path is 2048bits × the number of brunches.

For the same type of the curve, Table 2 presents the time needed to calculate the main cryptographic operations in an Ubuntu 20.4 machine with 4 GB of RAM and an Intel i5 processor, using the *Python*3-based charm crypto library [12].

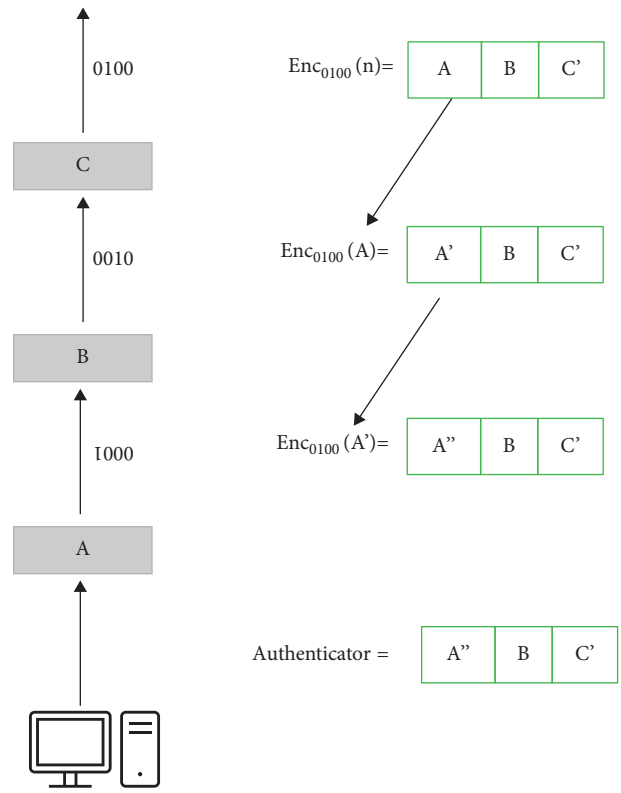

Figure 6: Detailed view of the authenticator construction process.

Table 1: Size of key components.

Component	Size
SP	$3 \times S_g + S_{gt}$
Secret key	$2 \times S_g$
C_{id}	$2 \times S_g + S_{gt}$
Authenticator for unicast path	$2 \times S_g + S_{gt}$

Table 2: Overhead of cryptographic operations.

Operation	Time in ms.
MSK and SP generation	19
SK formation	3.8
Encryption	3.9
Decryption	4.7

3.2.2. Security Evaluation. One of the significant advantages of our solution is that a sender can create a path authenticator using information given by the underlay source routing protocol. Furthermore, by using IBE, link identifiers can be used as an encryption key; therefore, there it is not required to keep separate public keys along with the link identifiers. An additional advantage of our solution is that it allows receivers to check if a packet was forwarded through the legitimate determined path, without the need to know that path: this not only makes our solution easier to deploy but also protects the sender's privacy. Moreover, our solution produces authenticators of constant and predictable size. Finally, IBE has been actively researched for almost two decades, and there are efficient tools and libraries for a plethora of platforms. For this reason, we believe that our solution can be integrated into existing systems.

Our solution "borrows" the disadvantages of IBE. Firstly, IBE suffer from the so-called key-escrow problem, i.e., there is a centralized entity, the Private Key Generator, that knows all secret keys. However, in our system, this role is held by the SDN controller which already has full control of the network. Additionally, IBE does not allow key rotation: if a secret key is lost or breached, then the corresponding identity must be changed. Our solution copes with this by changing the corresponding link identifier every time key rotation is required. From the network side, this issue is treated as a link has gone offline and a fresh one has been activated. In more detail, an IBE secret key cannot be

rotated, i.e., it is not possible to change a secret key that corresponds to an identifier. The only way to achieve this is by modifying the MSK and consequently the corresponding SP. Nevertheless, this is a cumbersome process since in that case all secret keys have to be updated, and all active entities have to be notified about this change. In order to avoid having this happening often, we require from the SDN controller to not reuse the same link_{ID}: whenever a SDN switch needs a new secret key, it will simply receive a new link_{ID}.

4. Related Work

Legnet et al. [13] proposed "EPIC" an authentication system for interdomain paths. The pathways are at the AS level, and the authenticators in [13] are based on symmetric key encryption. For the transmitter and recipient nodes to know the related symmetric keys, EPIC requires the usage of additional protocols. Our method uses identity-based encryption, which only requires senders to provide link IDs (which may be collected as part of the routing protocol) and no further information from receivers.

Kim et al. [14] also provide an HMAC-based path validation method. The work in [15] may be used at both the AS and router levels, and it requires the usage of a trusted entity to create certificates for each verifying entity. All certifications should be accessible by both senders and receivers. Receivers do not require any additional information to use our service. Furthermore, we employ link IDs instead of certificates since link identifiers are provided by default in routing protocols, making them easier to distributed than digital certificates.

The work in [16] brought another solution with similar goals to ours called ICING. Using self-generated public keys, ICING provides proofs of provenance. ICING, on the contrary, requires a specialized server per node known as the consent server to validate entities' desire to distribute these keys with their direct connections. Our solution does not require switches to exchange any information with each other. Furthermore, our solution does not require any additional entity other than the standard entities of an SDN network.

In [17], the authors introduce a secure source routing approach to the problem with assumptions that are different from our solution: senders are considered untrustworthy, and the network tries to defend itself against malicious senders. The proposed mechanism allows senders to include proof in their packets that confirms they are "authorized" to send a packet in this manner.

Platypous proposed a work in [18] that is an extension of [17]: like in [17], it allows senders to add a stamp that demonstrates they are "policy compliant." In our solution, senders are considered trusted. For this reason, our solution protects senders against misbehaving forwarding nodes. Nevertheless, our solution does not prevent solutions similar to [17, 18]: both solutions can be used in parallel.

Other research that employs TEE, "Trusted Execution Environment," to attain similar goals. TEEs are hardware modules that have been cryptographically validated to run a certain piece of code as intended. The work of [15], for instance, builds secure routing protocols using Intel's SGX. Our solution does not require any specialized hardware, and it is secured against forwarding nodes that do not execute our protocol correctly: any protocol deviation will be detected, and the corresponding packets will be discarded.

Wang et al. [19] employed a technique that is similar to ours in terms of goals. It aims to protect SDN architecture by recognizing and categorizing harmful network traffic. The solution of [19], on the contrary, is less fine-grained than ours: our approach does per-packet checks, whereas solution in [19] performs per group of flows.

In Sasaki et al. [20], cryptographic proofs are also used to try to secure SDN architectures. Their approach, in particular, encrypts each packet and attaches a cryptographic proof to it, which can then be used to verify that it followed the correct path. The key difference between solution in [20] and our method is that, in [20], the verification process is handled by the SDN controller, whereas our solution's verification process is handled by the packet receiver. Receivers are unable to employ the technique in [20] because it requires the verifying entity to disclose a secret with the intermediate switches, which has security and scalability issues (since every possible receiver should share a secret with the corresponding switches). In our system, senders and recipients are not obligated to expose any secrets to anyone.

The authors in [21] stated that, Alibi routing adopts per-packet cryptographic proofs to protect the routing process. Nonetheless, their approach is used to establish that a packet did not cross a certain area, whereas our technique is used to verify that a packet followed a specific path.

5. Conclusions

In this study, we presented a solution used for securing source routing forwarding in SDN networks. Our solution allows senders to create and attach in transmitted packets a cryptographic authenticator that "authorized" SDN switches can modify predictably. Based on the final value of the authenticator, a recipient can determine whether or not a packet has been forwarded through the appropriate path. Our authenticators are of constant size, no matter the length of the path. Additionally, they can be created and verified using information that is already available using the underlay routing protocol. Forwarding an authenticator using our solution requires less than 5 ms. Although this is not a significant overhead, it prevents data transmission at line speed. Nevertheless, pairing-based cryptography is a very active research area where hardware-assisted implementation can be anticipated.

Future work in this area involves solutions for handling mobility. In particular, we will investigate solutions that will handle both recipient mobility, as well as dynamic networks. Furthermore, we will investigate various optimizations, such as not requiring all switches to perform authenticator verification, or not, including an authenticator in all packets of the same flow. Towards this direction, we will examine security-performance tradeoffs.

Data Availability

No data were used to support the findings of the study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The Deanship of Scientific Resarch (DSR) at King Abdulaziz University, Jeddah, Saudi Arabia, has funded this project, under grant no. RG-13-611-42.

References

[1] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, 2012.

[2] N. Fotiou, D. Mendrinos, and G. C. Polyzos, "Edge-assisted traffic engineering and applications in the iot," in *Proceedings of the Workshop on Mobile Edge Communications, MECOMM'18*, pp. 37–42, ACM, New York, NY, USA, 2018.

[3] X. Duan, Y. Liu, and X. Wang, "Sdn enabled 5g-vanet: adaptive vehicle clustering and beamformed transmission for aggregated traffic," *IEEE Communications Magazine*, vol. 55, no. 7, pp. 120–127, 2017.

[4] K. Poularakis, G. Iosifidis, and L. Tassiulas, "Sdn-enabled tactical ad hoc networks: extending programmable control to the edge," *IEEE Communications Magazine*, vol. 56, no. 7, pp. 132–138, 2018.

[5] M. Ojo, D. Adami, and S. Giordano, "A sdn-iot architecture with nfv implementation," in *Proceedings of the IEEE Globecom Workshops (GC Wkshps)*, pp. 1–6, IEEE, Washington, DC, USA, December 2016.

[6] I. Farris, T. Taleb, Y. Khettab, and J. Song, "A survey on emerging sdn and nfv security mechanisms for iot systems," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 812–837, 2019.

[7] D. Boneh and X. Boyen, "Efficient selective identity-based encryption without random oracles," *Journal of Cryptology*, vol. 24, no. 4, pp. 659–693, 2011.

[8] M. J. Reed, M. Al-Naday, N. Thomos, D. Trossen, G. Petropoulos, and S. Spirou, "Stateless multicast switching in software defined networks," in *Proceedings of the IEEE International Conference on Communications (ICC)*, pp. 1–7, Kuala Lumpur, Malaysia, May 2016.

[9] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[10] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using openflow: a survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 493–512, 2014.

[11] S. University, "Pbc cryptography library homepage," 2020, https://crypto.stanford.edu/pbc/.

[12] J. A. Akinyele, C. Garman, I. Miers et al., "Charm: a framework for rapidly prototyping cryptosystems," *Journal of Cryptographic Engineering*, vol. 3, no. 2, pp. 111–128, 2013.

[13] M. Legner, T. Klenze, M. Wyss, C. Sprenger, and A. Perrig, "{EPIC}: every packet is checked in the data plane of a path-aware internet," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pp. 541–558, 2020.

[14] T. H.-J. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig, "Lightweight source authentication and path validation," in *Proceedings of the ACM Conference on SIGCOMM*, pp. 271–282, August 2014.

[15] S. Kim, Y. Shin, J. Ha, T. Kim, and D. Han, "A first step towards leveraging commodity trusted execution environments for network applications," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, pp. 1–7, November 2015.

[16] J. Naous, M. Walfish, A. Nicolosi, D. Mazieres, M. Miller, and A. Seehra, "Verifying and enforcing network paths with icing," in *Proceedings of the 7th Conference on Emerging Networking Experiments and Technologies*, pp. 1–12, 2011.

[17] B. Raghavan and A. C. Snoeren, "A system for authenticated policy-compliant routing," *ACM SIGCOMM - Computer Communication Review*, vol. 34, no. 4, pp. 167–178, 2004.

[18] B. Raghavan, P. Verkaik, and A. C. Snoeren, "Secure and policy-compliant source routing," *IEEE/ACM Transactions on Networking*, vol. 17, no. 3, pp. 764–777, 2008.

[19] T. Tao Wang and H. Hongchang Chen, "Sguard: a lightweight sdn safe-guard architecture for dos attacks," *China Communications*, vol. 14, no. 6, pp. 113–125, 2017.

[20] T. Sasaki, C. Pappas, T. Lee, T. Hoefler, and A. Perrig, "Sdnsec: forwarding accountability for the sdn data plane," in *Proceedings of the 25th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–10, IEEE, Waikoloa, HI, USA, August 2016.

[21] D. Levin, Y. Lee, L. Valenta et al., "Alibi routing," *ACM SIGCOMM - Computer Communication Review*, vol. 45, no. 4, pp. 611–624, 2015.