On-demand Key Distribution for Cloud Networks

Paladi, Nicolae; Tiloca, Marco; Nikbakht Bideh, Pegah; Hell, Martin

Published in:
Proceedings of the 24th Conference on Innovation in Clouds, Internet and Networks

DOI:
10.1109/ICIN51074.2021.9385528

2021

On-demand Key Distribution for Cloud Networks

Nicolae Paladi
RISE and Lund University
Isafjordsgatan 22
Kista, Sweden
Email: nicolae.paladi@eit.lth.se

Marco Tiloca
RISE Research Institutes of Sweden
Isafjordsgatan 22
Kista, Sweden
Email: marco.tiloca@ri.se

Pegah Nikbakht Bideh and Martin Hell
Lund University
John Ericsons Väg 3
Lund, Sweden
Email: {pegah.nikbakht_bideh, martin.hell}@eit.lth.se

***Abstract*—Emerging fine-grained cloud resource billing creates incentives to review the software execution footprint in virtual environments. Operators can use novel virtual execution environments with ever lower overhead: from virtual machines to containers, to unikernels and serverless functions. However, the execution footprint of security mechanisms in virtualized deployments has either remained the same or even increased. In this demo, we present a novel key provisioning mechanism for cloud networks that unlocks scalable use of symmetric keys and significantly *reduces* the related computational load on network endpoints.**

I. Introduction

Increasingly fine-grained billing for computation [1] and network resources [2] is a notable recent trend. Such fine-grained billing creates strong incentives to develop and deploy applications that utilize a minimum amount of computing and network resources. This calls for a novel approach in application development and deployment, allowing to trim down the use of computing and network resources without compromising the security of the deployment.

Network architectures designed for cloud computing, such as Software-defined networking (SDN), help in network management and configuration. While capabilities introduced by SDN allow new key distribution mechanisms, public key cryptography is usually preferred to symmetric key cryptography in cloud and virtualised environments, especially to support the setup of secure associations and secure communication channels.

However, public key cryptography is CPU-expensive and adds additional communication costs. These costs and overhead can be reduced by relying on symmetric key cryptography instead. On the other hand, symmetric key cryptography leads to challenges including secure key provisioning and authentication.

To address these challenges, we demonstrate a novel, on-demand symmetric key distribution mechanism between endpoints in SDN deployments. The mechanism reduces the number of steps for providing symmetric keys to the endpoints, the computational load on the endpoints and the required time to setup the secure communication. The approach leverages existing channels established during the deployment orchestration and used for endpoint monitoring and patching. It *does not* establish new communication channels to enable the solution, leaving the attack surface practically unchanged. Our mechanism enables *flow-specific symmetric keys* and is compatible with (D)TLS v1.2 [3][4] and v1.3 [5][6].

https://doi.org/10.1109/ICIN51074.2021.9385528 ©2021 IEEE

II. Architecture

We consider the network architecture in Figure 1, which includes: a Client endpoint C; a Server endpoint S; an SDN Controller for establishing and managing network flows; and a network Switch capable to forward traffic between C and S, according to the established flows.

The endpoints C and S are under the control of a common *logical* Controller, but they do not necessarily belong to the same network. Note that the architecture does not preclude the scenario of a *distributed* controller, which acts as a single logical Controller but is distributed across several instances [7]. The Controller is assumed to have three pre-established secure communication channels, i.e. one with C, one with S, and one with the Switch; these can be, for instance, secure (D)TLS sessions. As highlihgted above, we reuse connections that are commonly in place for operational purposes, such as endpoint monitoring or updates.

Note that the proposed mechanism is also compatible with alternative, serverless architectures. In this case, the two endpoints will act as initiator and responder, instead of client and server.

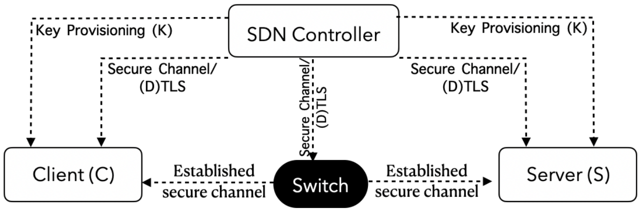

Figure 1: High level network architecture and components

In the considered setup, C wishes to securely communicate with S, for instance through a secure (D)TLS session. To speed up the secure communication setup and improve performance as a whole, this abstract focuses on C and S as establishing a secure communication session using a *symmetric* key K. This typically requires pre-sharing K, whose distribution requires prior knowledge of the communication pattern expected from C and S, as well as of the whole network topology, which is impractical.

Instead, the approach taken in this demo enables the fast distribution of K on-demand as coordinated by the Controller, at the time when the network flow between C and S is established. This allows the two endpoints to have a different symmetric key per flow, each used by a different application to establish a secure session over that flow.

The result is the effective separation of the security domains of different flows from one another, with benefits for the different applications running over those different flows. That is, other than being computationally cheap, establishing per-flow keys has the advantage that compromising the symmetric key of one flow does not affect the security of other flows and their respective applications, hence ensuring separation of security domains with minimal overhead.

Also, the considered approach overall simplifies the distribution and management of key material in the network, which in turn simplifies the maintenance of hardware and software for cryptographic operations and key generation.

A. *Workflow Summary*

The key K is distributed as follows. After C has sent a first packet addressed to S, the Switch does not find any matching flow rule in its flow table and sends a control message to the Controller.

Then, the Controller generates a new flow rule for the traffic between C and S, as well as a symmetric key K associated to that flow. The Controller uses a secret seed with sufficient entropy and a secure Key Derivation Function to derive the symmetric key K. The Controller essentially generates as many keys as new flows it establishes.

After that, the Controller separately provides K to both C and S, over the respective secure communication channel. Finally, the Controller provides the new flow rule to the Switch, that can now forward the packet from C to S.

Having received K, both C and S install it as the key associated to this network flow and use it to establish a secure communication session, for example through the (D)TLS Handshake protocol. C and S communicate over that flow and the established secure session. While the demo focuses on the implementation with (D)TLS, our approach is compatible with other protocols for secure session establishment.

The symmetric key K has an expiration time of its own, after which the Controller can accordingly terminate the flow according to an administrative policy, and the endpoints can later request to start a new flow if needed. If C and S close the secure session but the flow still exists, the two endpoints can open a new session over it before the key expires, without notifying the Controller.

III. Innovative Concepts and Relevance

The demonstrated mechanism introduces *flow-specific encryption keys* and describes a key provisioning mechanism that leverages the use of symmetric encryption keys in virtualised deployments within an administrative domain. Flow-specific encryption keys allow using distinct key material to protect traffic on different flows between two same endpoints.

This is a novel construct specifically designed for software-defined networks and especially relevant for virtualised enterprise networks. While this is in line with a defence-in-depth security approach, it does not on its own introduce a performance overhead, assuming that network traffic encryption is already used. This is because, instead of using the same cryptographic material to protect all network communication between two endpoints, every flow uses an independent set of cryptographic material.

Likewise, the approach *does not* trade computational overhead for network communication overhead. Instead, it practically enables the use of symmetric key encryption, embedding the payload in the flow establishment communication. This is different from the approaches typically used for PSK distribution, since it is agnostic of network topology and traffic requirements, and it occurs at run-time and on-demand, with no need for early information on peers and expected flows.

The rationale for shifting the task of key generation from endpoints to a dedicated component integrated in the network control is as follows. Centralised network management and network virtualisation are commonly used to implement a consistent set of policies in enterprise networks. Our demonstrated approach allows implementing network-wide consistency in key generation while minimising endpoint code-base and resource usage by removing the need for key generation code in endpoints. In turn, this can help reduce operational costs for the network infrastructure without compromising security. Moreover, keys can be pre-generated and cached whenever the CPU load on the controller host is at its minimum, thus trading storage for computation. The use of symmetric keys is tightly coupled with the implementation of flow-specific encryption keys and is essential in achieving high scalability and - in certain scenarios - reducing the computational overhead of network traffic encryption.

In an enterprise computation setting, flow-specific encryption using symmetric keys allows deploying robust and flexible network traffic encryption throughout the network domain. This helps to protect at scale network traffic between lightweight computation environments (such as Docker or Linux containers, unikernels or serverless functions), with minimal computational overhead. In turn, this allows organisations to benefit from the trend towards ever more fine-grained cloud resource billing and match their costs with the computational resources used.

Finally, key generation in a dedicated component allows to make full use of hardware support for both *key generation* and *key management*. Key generation can be facilitated with improved entropy sources such as a hardware entropy source, while hardware security modules are often used to facilitate the implementation of key management policies.

IV. Demonstration Functions and Features

On a high level, our approach is explained in the basic scenario introduced below[1]: two endpoints running in a virtualised network environment establish for the first time a

[1] Recorded video: https://vimeo.com/431482962

communication session over an OpenFlow switch. Since there is no flow rule matching this network pattern, the switch upstreams the first packet (a TCP SYN in this scenario) to the Controller for inspection.

An application of the Controller generates a symmetric key and deploys it to both endpoints. After that, the Controller returns the upstreamed packet to the data plane and the communication session establishment continues. Once the endpoints established a TCP session, they start using the deployed symmetric key and proceed to wrap the communication in a TLS session.

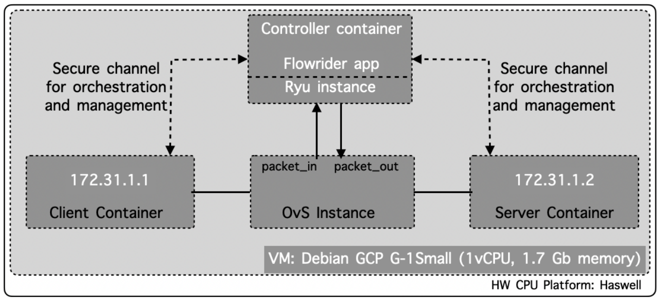

Figure 2: System test bed

The test bed, illustrated in Figure 2, is implemented in four Docker containers with the following roles: (a) Client; (b) Server; (c) Controller; (d) Open vSwitch (OvS). The endpoints communicate over TLS 1.3 [5] implemented with GnuTLS [8], version 3.6.5. The Controller container runs Ryu 3.12 and a custom Python application, that defines packet matching and subsequently generates and delivers keys to the endpoints. The OvS container runs an instance of Open vSwitch that routes packets between endpoints and forwards predefined packet types to the Controller.

The specific features shown by this demonstration are:

1) Deployment of new symmetric keys to the end-points for each flow;
2) Establishment of a secure communication channel using the deployed symmetric keys;
3) Formal verification of the approach using ProVerif.

In order to verify the security of the proposed solution, we modelled it with ProVerif[2]. Throughout the modeling, we maintained the assumption of a pre-established secure channel between the Client and the network Controller, as well as between the Server and the network Controller. The channels were securely pre-established using different key material, assumed inaccessible for the adversary. The Client, the Server, the Switch and the network Controller are each modelled as independent, top-level processes.

Then, we verified the following security properties:

1) The secure provisioning and resulting secrecy of key K, i.e. the key associated to the flow between the Client and the Server;
2) The secure possession of key K by Client, Server and Network Controller.

The demo shows how cloud network deployments can leverage the use of per-flow keys, using pre-shared symmetric keys. This improves both security - as different flows stemming from different applications are separated - as well as efficiency since symmetric key operations require less overhead than their asymmetric counterparts. Moreover, the demonstrator shows how the key distribution can be incorporated into the widely used TLS protocol. Thus, with only minor modifications to an existing cloud deployment, we can achieve this increase in security and efficiency. Users can interact with the demo by downloading the provided implementation and deploying it on their local setup.

V. Future Work

The demonstrator is under active development and we intend to expand this work in several directions. A first target is to adapt the key sharing mechanism to other widely used network security protocol suites (primarily IPsec). Other future works include verifying the performance in hyperscale settings and integrating with cloud orchestration software.

VI. Acknowledgements

This work was supported in part by the Swedish Foundation for Strategic Research, grant RIT17-0035; by VINNOVA and the Celtic-Next project CRITISEC; by the H2020 projects SIFIS-Home and ASCLEPIOS (Grant agreements 952652 and 826093); and by the Wallenberg AI, Autonomous Systems and Software Program (WASP).

References

[1] Y. Zhu, J. Ma, B. An, and D. Cao, "Monitoring and Billing of A Lightweight Cloud System Based on Linux Container," in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. New York, NY, USA: IEEE, 2017, pp. 325–329.

[2] H. Jin, X. Wang, S. Wu, S. Di, and X. Shi, "Towards optimized fine-grained pricing of iaas cloud platform," *IEEE Transactions on cloud Computing*, vol. 3, no. 4, pp. 436–448, 2014.

[3] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246 (Proposed Standard), RFC Editor, Fremont, CA, USA, pp. 1–104, Aug. 2008, updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905, 7919. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5246.txt

[4] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2," RFC 6347 (Proposed Standard), RFC Editor, Fremont, CA, USA, pp. 1–32, Jan. 2012, updated by RFCs 7507, 7905. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6347.txt

[5] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, RFC Editor, Fremont, CA, USA, Aug. 2018.

[6] E. Rescorla and H. Tschofenig and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3," May 2020, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-ietf-tls-dtls13-38

[7] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. R. Kompella, "ElastiCon; an elastic distributed SDN controller," in *2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 2014, pp. 17–27.

[8] N. Mavrogiannopoulos and S. Josefsson and D. Ueno and C. Latze and A. Pironti and T. Zlatanov and A. McDonald, *GnuTLS Reference Manual*. London, GBR: Samurai Media Limited, 2015.

[2] https://anonymous.4open.science/r/bb45538c-2e5d-4f70-a919-f1dedb9d158a