The KISS Principle in Software-Defined Networking: A Framework for Secure Communications

Diego Kreutz | University of Luxembourg and Federal University of Pampa
Jiangshan Yu | University of Luxembourg and Monash University
Paulo Esteves-Veríssimo | University of Luxembourg
Cátia Magalhães | KCS iT
Fernando M.V. Ramos | University of Lisbon

The pace of adoption of secure mechanisms in software-defined networking (SDN) has been slow, largely due to traditional solutions' performance overhead and their support infrastructure's complexity. To address these challenges, we propose KISS, a secure SDN control plane communications architecture that includes innovative solutions in the context of key distribution and secure channel support.

In software-defined networking (SDN), network control is separated from the forwarding devices and logically centralized in a controller. This separation is achieved by means of a protocol (typically, OpenFlow) that enables the SDN controller to remotely populate the forwarding tables of network switches. The OpenFlow standard includes Transport Layer Security (TLS; see IETF RFC 5246) as an optional security feature for authenticating forwarding devices and controllers and for encrypting the communication channel. However, to date most reported deployments still use TCP for control traffic, and SDN controllers and switching hardware with TLS support are still rare.[1] This makes the control plane communication vulnerable to different attacks.[1,2]

Four fundamental issues can slow down the rate of adoption of secure mechanisms in SDN. First, securing communications has a non-negligible cost in terms of increased communication latency and reduced performance. Several recent studies have analyzed this overhead in various contexts.[3] Second, the computing capabilities of commodity switches are typically weak. The typical SDN switch is equipped with a single- or dual-core CPU running at approximately 1 GHz, which compares unfavorably with the multicore CPUs found in typical commodity servers. Imposing the additional cost of TLS to these computing-constrained networking devices is a problem. Third, poor choice of cryptographic primitive implementations can also have a significant impact on the performance of the control plane communications handled by the controller. Finally, the public-key infrastructure (PKI) on which TLS relies is complex and thus vulnerability prone,[4] opening a large surface for successful attacks.[5]

To meet these challenges, we propose a modular secure SDN control plane communications architecture KISS, which aims to increase the robustness of control communications while enhancing their performance, by decreasing the complexity of the support infrastructure, as an alternative to current approaches based on classic configurations of TLS and PKI. (For more information on related work, see the sidebar.)

A core novel component of our architecture is the integrated device verification value (iDVV), a deterministic but indistinguishable-from-random secret code generation protocol. The concept was inspired by the iCVVs (integrated card verification values) used in credit cards to authenticate and authorize transactions

Related Work

There are several feasible attacks against the SDN control plane.[2] Most of them explore vulnerabilities such as the lack of authentication, authorization, and other essential security properties. However, almost no attention has been paid to the security requirements of control plane associations and communication between devices. For instance, only recently, the use of secrecy through obscurity has been proposed to protect SDN controllers from DoS attacks.[14] In this case, the switch authentication ID is hidden in a specific field in the IP protocol. It is assumed that the devices share a lookup table and unique IDs. However, in spite of being capable of mitigating DoS attacks, this technique does not address the security issues of control plane communications. As another example, the Open Network Foundation (ONF) defines TLS only as optional for OpenFlow-based SDNs. This can make the network vulnerable and therefore affect the overall security of the enterprise. Interestingly, even if we use TLS and PKI, we will most likely face security, performance, or complexity issues. Recent research has been trying to identify and recommend ways to mitigate these issues.[12,13,15] Among the alternatives to reduce communication latency, clean slate approaches, using alternative libraries such as NaCL, have been proposed.[12]

in a secure and inexpensive way. We develop and extend the idea for SDN, proposing a flexible method of generating iDVVs by adapting proven one-time password-like techniques. iDVV codes allow the safe decentralized generation/verification of keys at both ends of the channel, at will, even on a per-message basis.

To understand and minimize the cost of security, we quantify the impact of secure primitives on the performance and scalability of control plane communications through a performance study of different implementations of TCP versus TLS, complemented by a deeper study of underlying hashing and message authentication code (MAC) primitives. Those experiments confirm our intuition that the choice of protocols and primitives used in secure communication may well be one strong reason behind the slow adoption of these mechanisms in SDN.

KISS Architecture

KISS is a modular secure control plane communications architecture for SDN, offering alternatives to classic configurations of secure channel and authentication protocols and subsystems followed in TLS and PKI. We assume a typical SDN architecture, as illustrated in Figure 1, composed of controllers and forwarding devices. We further assume that device registration and association services are in place. For lack of space, we do not discuss them in detail, but for self-containment, we discuss some properties and their interface below.

The two components encapsulated by the KISS boxes are the crucial components of the architecture, and the main subject of our study: a secure channel protocol suite, composed of a judicious choice of state-of-the-art mechanisms and protocols, which we dub *SC* for convenience of description, and iDVV, a novel deterministic but indistinguishable-from-random secret code generation protocol.

We considered using TLS implementations (for instance, OpenSSL) as the baseline protocol for SC. However, our experiments alerted us to the sheer performance cost of cryptographic communication and the further impact of suboptimal choices of cryptographic primitives. This motivated us to adopt NaCl,[6] a high-performance yet secure cryptographic library, as the substrate of SC, complemented by the MAC and strong hash primitives with best performance according to our experiments—Poly1305 and SHA512 OpenSSL. SHA-512 is used by the iDVV generator while Poly1305 is a fast MAC algorithm.

The iDVV, a novel component we propose, helps to further enhance the security of SC through strong crypto material generated at a low cost (for instance, one-time keys, per-message authentication, and authorization codes) to be used by NaCl ciphers. The indistinguishability-from-random allied to the determinism allows the safe decentralized generation and verification of per-message keys at both ends of the channel.

System and Threat Model

For simplicity and without loss of generality, we assume that the controllers and forwarding devices are registered and associated through a secure and robust key distribution service provided by a key distribution center (KDC), which is out of the scope of this article, but can be readily secured by state-of-the-art KDCs like Kerberos Key Distribution Center.

The device registration process is by default invoked by network administrators to the KDC to register new devices. As the result of device registration, the device

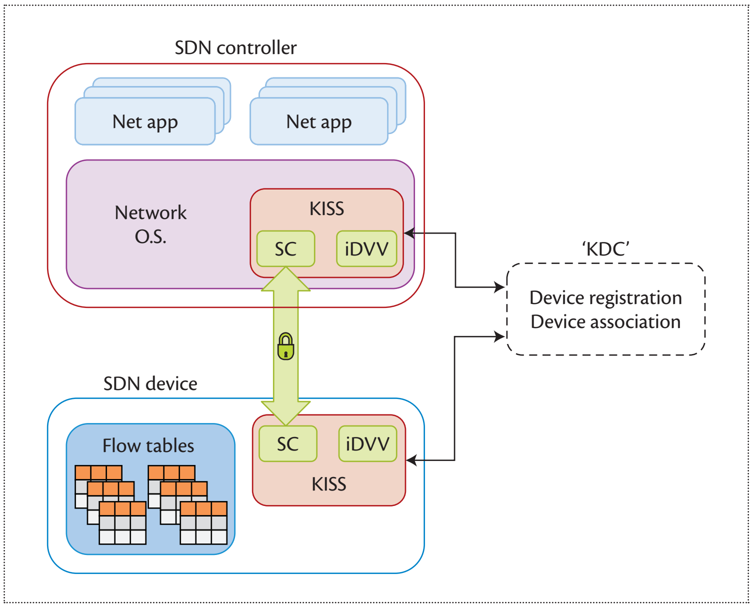

Figure 1. KISS general architecture.

and the KDC securely share a symmetric key. We denote K_{kc} the shared key between the KDC authority and a registered controller, and K_{kf} the shared key between the KDC authority and a registered forwarding device.

Registered controllers and forwarding devices must be securely associated, also through the KDC authority, as a precondition to communicate securely. The most common case is a forwarding device f_i requesting an association to a controller c_j, through the KDC. After associating, a controller and a forwarding device share two symmetric secrets (of size 256 bits), namely a $seed_{ij}$ and a key_{ij}. The key is generated by the KDC, and the seed is generated by the KDC in cooperation with the controller. These secrets will be used to bootstrap the iDVV module.

As threat model, we consider a Dolev-Yao style attacker, who has complete control of the network; the attacker logs all messages and can arbitrarily delay, drop, reorder, insert, or modify messages. In addition, this strong attacker can compromise any network device (for instance, a controller or a forwarding device) at any time. We assume the security of the used cryptographic primitives, including MAC (that is, Poly1305), hash function (that is, SHA-512), and symmetric encryption algorithm (for instance, AES).

Security Goals

The main goal of KISS is to provide security properties including authenticity, integrity, and confidentiality for control plane communications while minimizing cost and complexity.

The secure communication between participants can be easily guaranteed when a secure encryption algorithm is used, as long as the shared secret key is kept secure. To provide a robust SDN system, we focus on advanced security guarantees for the situation when the shared key is exposed to an attacker, as this might happen in practice. In particular, if an attacker has compromised a device and learned its shared keys, then we are aiming to provide *perfect forward secrecy* (PFS) of communications. That is, the secrecy of a device's past communications should be protected when the device is compromised and its shared keys are exposed to an attacker. It is important to emphasize that PFS is an essential requirement for SDN. The lack of it can lead to information disclosure—that is, reveal different aspects of the network's state and the controller's strategy (for example, proactive or reactive flow setup).

Established KDC technologies like Kerberos have robust implementations and are intensely used by industry, which makes us consider the logical single point of failure they present as moderate, and an acceptable option for the current state of the art. Even though, as we said, the KDC is out of the scope of the article, we present mitigation measures to achieve PFS in case of compromise of the KDC. We also plan, as future work, to investigate the development of SDN KDCs resilient to accidental and malicious faults, drawing from fault and intrusion tolerance techniques.[7]

On the devices side, we make no claim about their sheer resilience, since this is largely dependent on vendors. More precisely, when a controller and/or a forwarding device is compromised, we consider that the attacker can obtain all knowledge of the victim devices, including all stored secrets and the session status. However, our goal is to guarantee the confidentiality of all past communications through measures that allow us to achieve perfect forward secrecy.

iDVV: Keep It Simple and Secure

iDVVs are sequentially generated to protect and authenticate requests between two networking devices. The generator is conceived so that its output sequence has the indistinguishability-from-random and determinism properties. In consequence, the same sequence of random-looking secret values is generated on both ends of the channel, allowing the safe decentralized generation and verification of per-message keys at both ends. However, if the seed and key initial values and the state of the generator are kept secret, there is no way an adversary can know, predict, or generate an iDVV.

In other words, an iDVV is a unique secret value generated by a device A (for instance, a forwarding device), which can be locally verified by another device B (for instance, a controller). The iDVV generation is made flexible to serve the needs of SDN. iDVVs can therefore be generated:

- on a per message basis,
- for a sequence of messages,
- for a specific interval of time, and
- for one communication session.

The main advantages of iDVVs are their low cost and the fact that they can be generated locally—that is, without having to establish any previous agreement.

Different from standard KDF algorithms such as HKDF, which assumes that keying material is not uniformly random or pseudorandom, our keying material (that is, seed and key) comprises random symmetric secrets (each of size 256 bits) generated by the KDC, with high entropy. In such cases, a strong hash function can be safely used to derive a key (RFC 4880). As shown by the results of our experiments (presented below), iDVV generation is simpler and faster than standard KDF algorithms such as HKDF (RFC 5869) and similar solutions.

iDVV Bootstrap

As discussed earlier, the association between two SDN devices—for example, forwarding device f_i and controller c_j—happens through the help of KDC, under the protection of the long-term secret keys obtained from registration (K_{kf}, resp. K_{kc}). The outcome of the association protocol is the distribution of two random secrets to both devices: a seed $seed_{ij}$, and an association key key_{ij}. The iDVV mechanism is bootstrapped by installing these two secret values in both the controller and the switch to animate the iDVV generation algorithms, which we describe next.

Note that the setup and generation of the iDVV values are performed in a deterministic way, so that they can be done locally at both ends. However, as iDVVs will be used as keys by cryptographic primitives such as MAC or encryption functions, they have to be indistinguishable from random. Hashing primitives are a natural choice for our algorithms, since they provide indistinguishable-from-random values if one or more of the input values are known only by the sender and the receiver. This explains why it is crucial that the seed and association key are sent encrypted and therefore known only to the communicating devices. Moreover, to prevent information leakage, all variables *seed*, *key*, and *idvv* in the algorithms below should have the same length, which we chose to be 256 bits in our design. This length is commonly considered robust. From our experiments, the hashing primitive to be used is SHA512, which yields 512 bits, of which we will use the most significant q bits if we need to reduce the output length to q (as recommended by IETF RFC 4880). For example, we use the most significant 256 bits of the SHA512 output as the key for symmetric ciphers.

The initial iDVV value is deterministically created at both ends of the association between two devices by calling function `idvv_init`, which performs hashing on the concatenation of the initial *seed* and *key*, as illustrated by the algorithm below. (For readability, we omit the device-identifying subscripts in the variables.) After setup, the generator is ready for first use, as described in the following section.

```
1: idvv_init()
2: idvv ← H(seed || key)
```

iDVV Generation

After the bootstrap with the initial *idvv* value, the `idvv_next` function is invoked on demand (again, synchronously at both ends of the channel) to autonomously generate authentication or encryption keys that will be used for securing the communications, as illustrated by the algorithm below.

```
1: idvv_next()
2: seed ← H(seed || idvv)
3: idvv ← H(seed || key)
```

The *key* remains the only constant shared secret between the devices. The *seed* evolves to a new indistinguishable-from-random value each time `idvv_next` is invoked to generate a new iDVV. The new seed is the outcome of a hashing primitive H over the current *seed* and current *idvv* (line 2). The new *idvv*, output of function `idvv_next`, is the outcome of a hashing primitive H over the concatenation of the *new seed* and association key *key*.

iDVV Synchronization

The iDVV mechanism is agnostic with regard to secure communication protocols and can be used in a number of ways, in a number of protocols, as a key-per-message or key-per-session, and so on. The only key issue about iDVV generation is to keep it synchronized in both ends of the channel. A thorough analysis of use cases and recommendations can be found in our extended report.[8]

The most general style of iDVV use is *Indexed iDVV*: iDVVs are indexed by the generation number, and they are operated in "one key per direction" mode—that is, at each end, one iDVV is generated for each communication direction. This way, they support competitive, non-synchronized correspondents. This mode also supports unreliable, connectionless protocols like UDP. Each iDVV generated is indexed by a sequence number (the initial iDVV being $idvv^0$), and the sequence number is included in the message where the respective *idvv* is used. This way, each receiving end (this works in either direction, as we have two pairs of iDVVs) can know the exact *idvv* number that should be used and, for example, detect and recover from omissions, by generating *idvvs* the necessary number of times to resynchronize.

iDVVs can get out of sync for several reasons, such as speed differences, omission errors, or even DoS attacks. When de-synchronization happens, a baseline technique consists of advancing the iDVV of the "slower" end to catch up. The process is made robust by two techniques. First, communication should be authenticated (encrypt-then-MAC recommended) such that any messages failing crypto (decryption or MAC verification) can be simply discarded. Second, when say, $idvv^k$ is advanced to $idvv^l$ ($k < l$) to re-synchronize, and the operation is not successful (crypto fails), the old $idvv^k$ is restored (and the message motivating the recovery is discarded, as per above). This restoration does not affect the PFS of communications because the $idvv^k$ (or newer) has not yet been used to secure the traffic between the two communicating devices. Finally, in the case of attacks, these robustness techniques also help to foil them, since the attacker cannot mimic valid crypto, so the message is discarded and the node returns to the original iDVV state.

iDVV Implementation and Application

iDVVs require minimal resources, which means that they can be implemented on any device, from a simple and very limited smart card to most existing devices. In other words, they are a simple and viable solution that can be embedded in any networking device. Just three values per association have to be securely stored—the seed, the association key, and the iDVV itself—to use iDVV continuously. Furthermore, only hash functions—simple to implement and with a very small code base—are required to generate iDVVs. Such kind of resource is already available on all networking devices that support traditional network protocols and basic security mechanisms.

iDVVs are inexpensive and, as a result, can be used on a per-message basis to secure communication. It is worth emphasizing that, from a security perspective, one fresh iDVV per message makes it much harder for attacks such as key recovery and advanced side-channel attacks, among other general HMAC attacks, to succeed. In fact, the one-time key approach was initially used for generating MACs. Yet, it was set aside (that is, replaced by keys with a longer lifetime) due to performance reasons. However, as the iDVV generation has a low cost, we incur a lower penalty.

Finally, iDVVs can have further practical applications. For instance, the TLS handshake can be used to bootstrap the iDVV. After that, iDVVs can be used as session keys, that is, in security mechanisms such as encrypt-then-MAC.

On the Cost of Security

Here we provide a quantitative analysis of the impact of cryptographic primitives on control plane communication. Although the number of use cases is expanding, SDN has been mainly targeting data centers. As such, SDN controllers must be capable of dealing with the challenging workloads of these large-scale infrastructures. In these environments, new flows can arrive at a given forwarding device every 10 µs, with a great majority of mice traffic lasting less than 100 ms.[9] (In spite of the fact that there are several definitions of flow in SDN,[1] we equate SDN flow with TCP flow for the sake of simplicity.) This means that current data centers need to handle peak loads of tens of millions of new flows/s. The control plane has to meet both the network latencies and throughputs required to sustain these high rates. Current controllers are capable of achieving a throughput of up to 20M flows/s using TCP.[1]

Any effort to systematically secure control plane communications has to meet these challenges. In the following, we try to put the problem in perspective by analyzing the effect of including even the most basic security primitives to ensure authenticity, confidentiality, and integrity when considering peak loads of this magnitude.

The Cost of Secure Channels

Our first experiments assess the compared average latency of TCP and TLS on control plane communication. We analyze the latency of connection setup and of OpenFlow `PACKET_IN`/`FLOW_MOD` messages. The OpenFlow `PACKET_IN` message is used by switches to send packets to the controller (for example, when there is no rule matching the packet received in the switch). `FLOW_MOD` messages allow the controller to modify the state of an OpenFlow switch.

The connection setup time for TLS is two orders of magnitude higher than for TCP, since TLS has a more elaborate handshake protocol between the devices.[8] Also, PolarSSL (a library used in systems from companies such as Gemalto, ARM, and Linksys) induces nearly twice the overhead of OpenSSL. However important, a high connection cost can be amortized by maintaining persistent connections. As such, we focus on the communications cost. Figure 2 shows the latency of `FLOW_MOD` messages, averaged over 10k messages. The results with `PACKET_IN` messages were similar so we omit them for clarity. The costs of TCP, OpenSSL, and PolarSSL grow nearly linearly with the number of forwarding devices. OpenSSL latency is approximately three times higher than TCP. This is explained

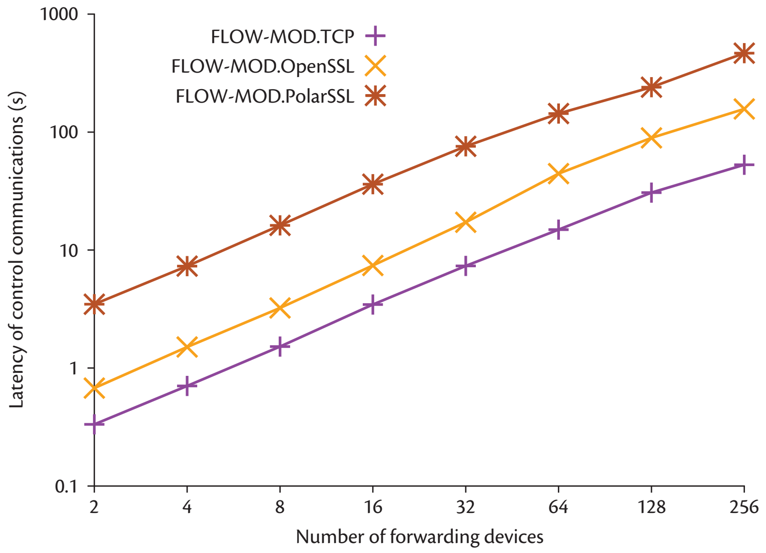

Figure 2. FLOW_MOD latency (in log scale).

by the high overhead of cryptographic primitives, as we further analyze in the next section. PolarSSL is significantly worse, increasing the latency by up to seven times when compared with TCP.

Conclusions. The main findings of this analysis can be summarized in two points. First, different implementations of TLS present very different performance penalties. Second, the additional computation required by the cryptographic primitives used in TLS leads to a non-negligible performance penalty in the control plane. In consequence, we turn to lightweight cryptographic libraries, such as NaCl[6] and TweetNaCl,[10] which are starting to be used in different applications. NaCl has been designed to be secure[6,11] and to be embedded in any system,[10] taking a clean slate approach and avoiding most of the pitfalls of other libraries (for example, OpenSSL misuse issues). First, it exposes a simple and high-level API, with a reduced set of functions for each operation. Second, it uses high-speed and highly secure primitives, carefully implemented to avoid side-channel attacks. Third, NaCl is less error prone because low-security options are eliminated, and it also provides a limited number of cryptographic primitives. In other words, users do not need deep knowledge regarding security to use it correctly. This is one of the major differences between it and other libraries such as OpenSSL. For instance, it has been recurrently shown that developers have been using OpenSSL in incorrect ways, leading to several security issues. Fourth, it has already been shown that secure and high-performance network protocols that outperform OpenSSL can be designed and implemented using NaCl.[12]

A Closer Look at the Cost of Cryptography

To understand in more detail the cause of the previous findings, we now perform a fine-grained analysis of two main classes of security primitives used in secure channel protocols: hashing and MAC.

We analyze the performance of nine hashing primitives. The results are presented in Figure 3. The red bars represent primitives that are provided by OpenSSL, while white bars (BLAKE and KECCAK) indicate the original implementation of primitives that are not part of OpenSSL. From Figure 3, we observe that the primitives with smaller digest sizes (SHA-1 and MD5) achieve better performance, as expected. The stronger versions of the SHA and BLAKE families achieve comparable performance (slightly slower), with higher security guarantees. Interestingly, SHA-512 outperforms SHA-256. This behavior is explained by the fact that on a 64-bit processor, each round can process twice as much data (64-bit words instead of 32-bit words). However, SHA-256 is faster on a 32-bit processor.

To understand the variance between different implementations, we present in Figure 4 the costs of the five hashing primitives for which different implementations were available. The OpenSSL implementation shows the best performance for hashing primitives. With the exception of RIPEMD160, the PolarSSL implementation always presented higher message latencies.

Finally, Figure 5 shows the results of the latency analysis of six MAC primitives. It is clear that Poly1305 outperformed all other primitives, being approximately two times faster than OpenSSL's HMAC-SHA1, and close to four times faster than HMAC-SHA512, for instance.

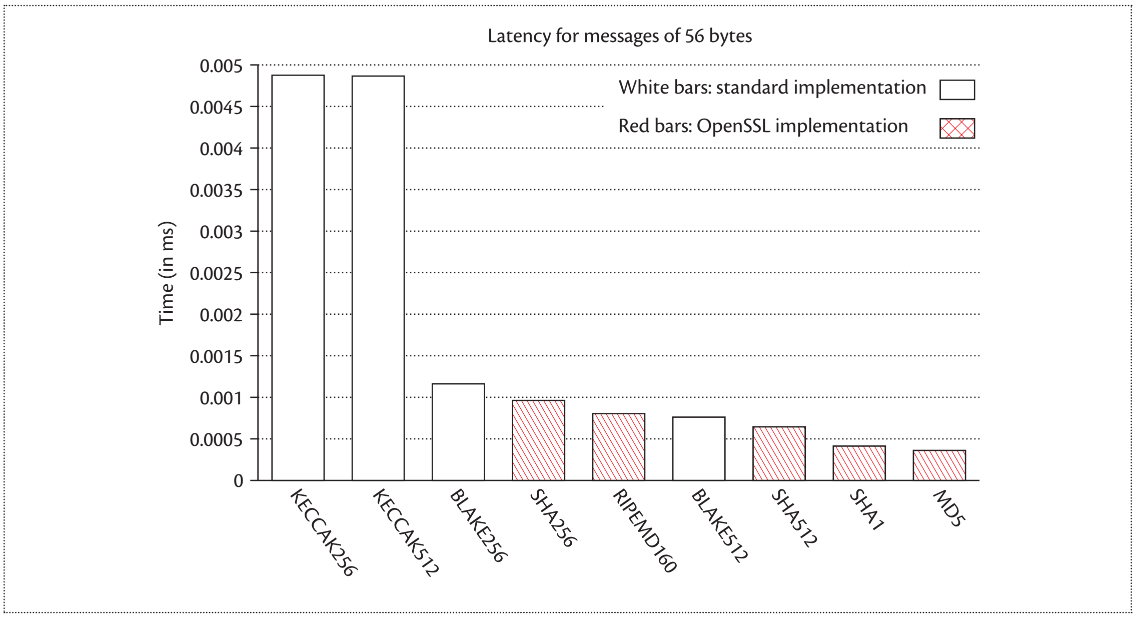

Figure 3. Hashing primitives.

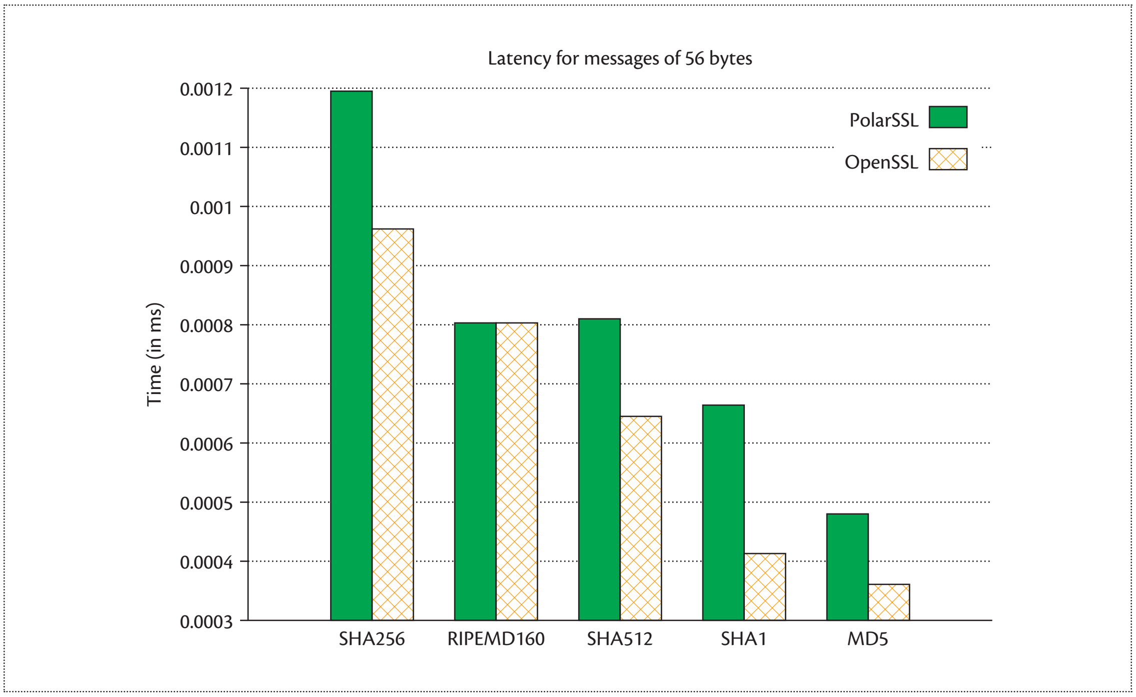

Figure 4. Implementations of hashing primitives.

Conclusions. From the results of Figure 5, considering the MAC primitive with best performance in the analysis (Poly1305 with 0.001 ms per message), around 20 dedicated cores are needed to compute a MAC to maintain a rate of 20M flows/s. To understand the importance of judiciously selecting the security primitives' implementation, the HMAC-SHA512 OpenSSL (worst case performance in the analysis) would require more than three times more cores (up to 65) to compute MACs at these rates. From the hashing primitive analysis in Figures 3 and 4, of the strong primitives (that is, all except SHA1 and MD5), SHA-512 performs the best. However, concerning MAC primitives, the performance of HMAC-SHA512 disappoints, and it is clear

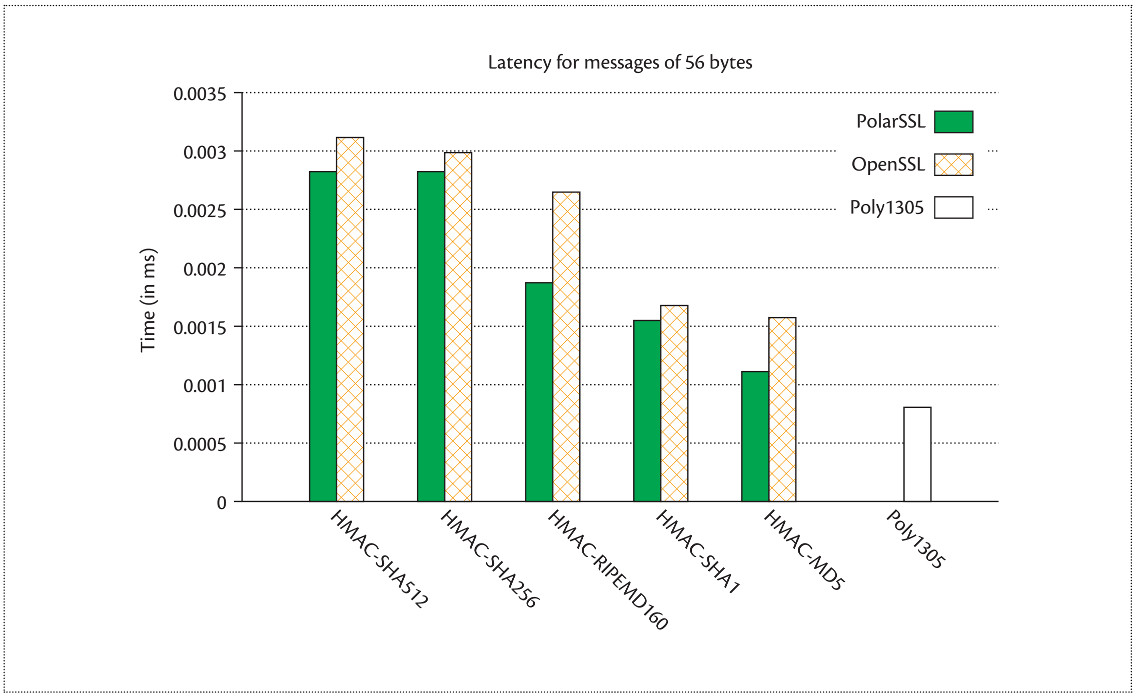

Figure 5. MAC primitives.

that Poly1305 outperformed all other primitives, providing security with high speed and low per-message overhead.

Figure 6 shows the performance of different primitives for generating cryptographic material. We compare the iDVV generator using SHA512 (iDVV-S5) to an implementation of a common key derivation function (KDFx) with different values for the exponent *c* (128, 64, 32, and 16, respectively), the Diffie-Hellman implementation used by OpenSSL (DH-OSSL), and the `randombytes()` function (NaCl-R) provided by NaCl. The latencies of the several primitives are significantly higher than iDVV. Even the *randombytes()* primitive of NaCl, the second fastest after iDVV, still presents a latency at least 2.6 times higher.

In summary, our findings in this section indicate that

- the inclusion of cryptographic primitives results in a non-negligible performance impact on the latency and throughput of the control plane, and
- a careful choice of the primitives used and their respective implementations can significantly contribute to reduce this performance penalty and enable feasible solutions in certain scenarios.

Taking the outcome of our analysis into consideration and given the benefits of NaCl described earlier, we selected the NaCl lightweight cryptographic library, and the MAC and strong hash primitives with best performance—Poly1305 and SHA512 OpenSSL—as the baseline SC secure channel component technologies. NaCl is complemented in our architecture with the iDVV mechanism to generate crypto material (for instance, keys) used by NaCl ciphers. Taken together they provide, as per our evaluation, the best tradeoff between security and performance for control plane communications in SDN.

Discussion

Next, we further discuss the security, robustness, and cost of iDVVs. Specifically, we explain how we ensure perfect forward secrecy for keys shared between any two devices, how we improve robustness by reducing the threat surface, and how we achieve higher performance than the alternatives while offering stronger security guarantees.

On the Security of iDVV

The secrecy of iDVVs is ensured from initialization and as long as the KDC, controller, and forwarding device are not compromised. Our scheme also achieves perfect forward secrecy in the face of compromise of any of these three elements of the architecture. In short, when the KDC is compromised, then the attacker would be able to obtain all the shared secrets (between the authority and registered devices), decrypt the past

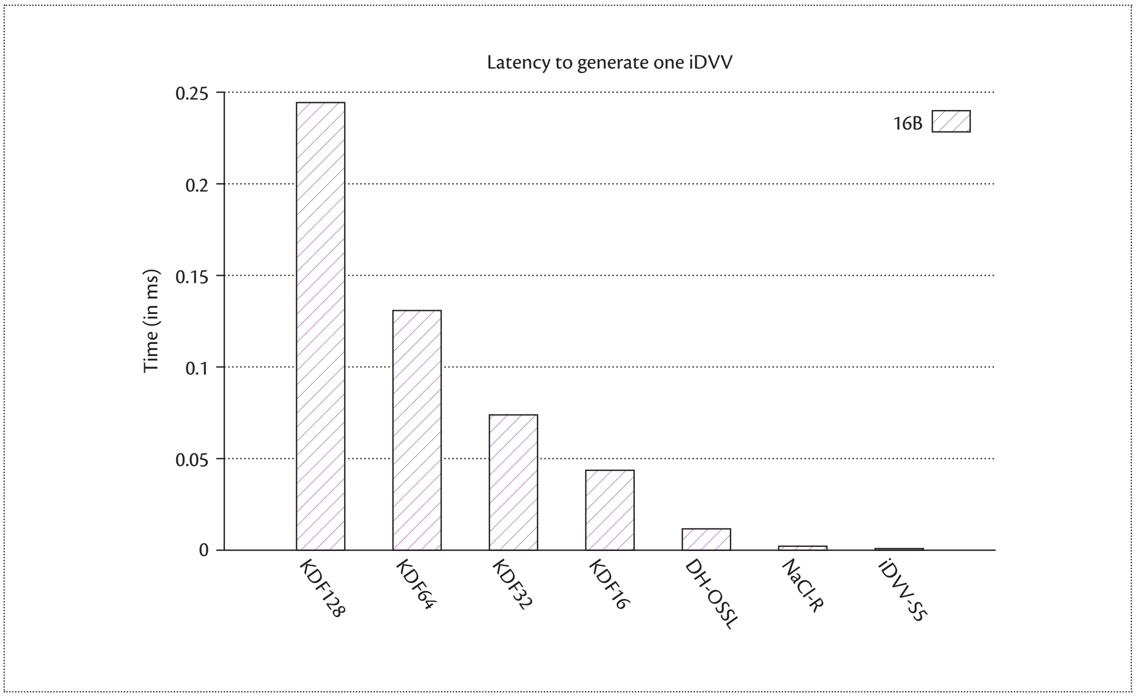

Figure 6. Latency to generate keys.

communication that delivered the initial *seed* and *key* to the associated devices, regenerate iDVVs, and, in consequence, decrypt past conversations.

We provide a simple mechanism for providing PFS even when the authority is compromised: we update the shared key each time a forwarding device is associated with a controller. The key is updated as follows: $K_{kc} \leftarrow H(K_{kc})$ and $K_{kf} \leftarrow H(K_{kf})$. This way, a captured shared key cannot decrypt any past messages, since they have been encrypted with previous generations of that key, which have been "forgotten" in the system, given the irreversible nature of hashes.

As far as devices are concerned, when they are compromised, the current values of *seed*, *key*, and *idvv* are captured. Note that *key* remains the original secret, but *seed* is rolled forward every time a new iDVV is generated. So, an attacker will be unable to synthesize any past iDVVs since day one and so cannot decrypt past conversations, achieving PFS, as we desired.

On the Solution's Robustness

Our proposal compares well with traditional solutions such as EJBCA (http://www.ejbca.org) and OpenSSL, two popular implementations of PKI and TLS, respectively.

The first interesting takeaway is that our solution has nearly one order of magnitude less LOC (85k) and uses four times fewer external libraries and only four programming languages. This makes a huge difference from a security and dependability perspective. For instance, formally proving more than 717k LOC (OpenSSL + EJBCA) is a tremendous challenge, and it gets considerably worse if we take into account 80 external libraries and 11 development languages. Moreover, it is worth emphasizing that libraries such as OpenSSL suffer from different fundamental issues such as too many legacy features accumulated over time, too many alternative modes as result of tradeoffs made in the standardization, and too much focus on web and DNS names.

Second, OpenSSL is complex and highly configurable. This has been also the source of many security incidents—developers and users frequently use the library inappropriately. It has also been shown that the majority of the security incidents are still caused by errors and misconfiguration of systems. Lastly, recent research has uncovered new vulnerabilities in TLS implementations.[13]

In contrast, our proposed architecture exhibits gains in both performance and robustness, contributing to solving the dilemma we outlined in the introduction. By having less LOC, we significantly reduce the threat surface—by one order of magnitude—and by combining NaCl and the iDVV mechanism, we provide a potentially equivalent level of security, but increased performance/robustness product, as keys can be rolled even on a per-message basis.

On the Cost of iDVV

Similarly to iCVVs, iDVVs are a low-overhead solution that requires minimal resources. This solution is thus feasible to be integrated into compute-constrained devices as commodity switches. Our preliminary evaluation has revealed that the iDVV mechanism is faster than traditional solutions, namely, the key-exchange algorithms embedded in the OpenSSL implementation. Considering a setup with 128 switching devices, sending `PACKET_IN` messages to and receiving `FLOW_MOD` messages from the controller, our results show our proposed solution (iDVV + NaCl's ciphers) to be more than 30 percent faster than an OpenSSL-based implementation using AES256-SHA (the most common high-performance cipher suite, used by IT companies such as Google, Facebook, Microsoft, and Amazon). Importantly, we were able to outperform OpenSSL-based deployments while still providing the same security properties: authenticity, integrity, and confidentiality. In addition, we achieved this result not only while offering the same properties, but also with stronger security guarantees: the tests were made by generating one iDVV per packet, while the OpenSSL-based implementation uses a single key (for symmetric ciphering) for the entire communication session.

An extended report of this work can be found in "The KISS Principle in Software-Defined Networking: An Architecture for Keeping It Simple and Secure,"[8] extending discussion of the iDVV synchronization, performance, forward secrecy, randomness, and proofs of its security properties. As ongoing work, we are extensively evaluating the NaCl-iDVV compound. Our results are clear in showing the solution to outperform OpenSSL on control plane communications.

We believe that this is a first step toward lightweight but effective security for control plane communication and, potentially, for SDN in general. We make a "call to arms" to foster developments on securing SDN communications without impairing performance, a fundamental precondition for widespread adoption by future SDN deployments. ■

Acknowledgments

We would like to thank the anonymous reviewers and Christian Esteve Rothenberg and Marcus Völp for the insightful comments. This work was partially supported by the Fonds National de la Recherche Luxembourg (FNR) through PEARL grant FNR/P14/8149128, by European Commission funds through the H2020 programme, namely by funding of the SUPERCLOUD project, ref. H2020-643964, by Fundação para a Ciência e a Tecnologia (FCT—Portugal), namely by funding of LaSIGE Research Unit, ref. UID/CEC/00408/2013 and the uPVN project, ref. PTDC/CCI-INF/30340/2017, and by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq—Brazil) through grant number 202104/2012-5.

References

1. D. Kreutz et al., "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, Jan. 2015.
2. S. Scott-Hayward, S. Natarajan, and S. Sezer, "A Survey of Security in Software Defined Networks," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, 2016, pp. 623–654.
3. D. Naylor et al., "The Cost of the 'S' in HTTPS," *Proceedings of the Tenth ACM Conference on Emerging Networking Experiments and Technologies* (CoNEXT 14), 2014, p. 7.
4. A. Wazan et al., "PKI Interoperability: Still an Issue? A Solution in the x.509 Realm," *Info. Assurance and Sec. Education and Training*, vol. 406, Springer, 2013.
5. N. van der Meulen, "DigiNotar: Dissecting the First Dutch Digital Disaster," *Journal of Strategic Security*, vol. 6, no. 2, 2013.
6. D. Bernstein, T. Lange, and P. Schwabe, "The Security Impact of a New Cryptographic Library," *Progress in Cryptology*, LATINCRYPT, LNCS 7533, Springer, 2012.
7. P. Verissimo et al., "Intrusion-Resilient Middleware Design and Validation," *Information Assurance, Security and Privacy Services*, vol. 4, Emerald Group Publishing Limited, May 2009, pp. 615–678.
8. D. Kreutz et al., "The KISS Principle in Software-Defined Networking: An Architecture for Keeping It Simple and Secure," ArXiv e-prints, June 2017; https://arxiv.org/abs/1702.04294.
9. T. Benson, A. Akella, and D.A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, (IMC 10), 2010, pp. 267–280.
10. D. Bernstein et al., "TweetNaCl: A Crypto Library in 100 Tweets," *Progress in Cryptology*, LATINCRYPT 2014, LNCS 8895, D.F. Aranha and A. Menezes, eds., Springer International Publishing, 2015, pp. 64–83.
11. J.B. Almeida et al., "Formal Verification of Side-Channel Countermeasures Using Self-Composition," *Science of Computer Programming*, vol. 78, no. 7, 2013, pp. 796–812.
12. W.M. Petullo et al., "MinimaLT: Minimal-Latency Networking through Better Security," *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, (CCS 13), 2013, pp. 425–438; http://doi.acm.org/10.1145/2508859.2516737.
13. B. Beurdouche, "A Messy State of the Union: Taming the Composite State Machines of TLS," *IEEE Symposium on Security and Privacy*, 2015, pp. 535–552.
14. O.I. Abdullaziz, Y.J. Chen, and L.C. Wang, "Lightweight Authentication Mechanism for Software Defined Network Using Information Hiding," *2016 IEEE Global Communications Conference* (GLOBECOM 16), 2016, pp. 1–6.
15. B. Agborubere and E. Sanchez-Velazquez, "OpenFlow Communications and TLS Security in Software-Defined Networks," *IEEE International Conference on Internet of Things* (iThings 17), and *IEEE Green Computing and*

Communications (GreenCom) and *IEEE Cyber, Physical and Social Computing* (CPSCom) and *IEEE Smart Data* (SmartData), 2017, pp. 560–566.

Diego Kreutz is a member of the CritiX group at the Interdisciplinary Centre for Security, Reliability and Trust (SnT), working toward a PhD degree in computer science at the University of Luxembourg, and an adjunct professor at Federal University of Pampa. His main research interests are in software-defined networking, intrusion tolerance, network security, security, and dependability. He is also member of learned societies such as IEEE, IEEE Computer Society, ACM, and SIGOPS. Contact him at kreutz@acm.org.

Jiangshan Yu is currently a lecturer at the Monash University and an honorary research fellow at the University of Birmingham, where he received his PhD in computer science. He was a research associate at SnT, University of Luxembourg. The focus of his research has been on applied cryptography, post-compromised security, cryptographic key management, and blockchains. Contact him at j.yu.research@gmail.com.

Paulo Esteves-Veríssimo is a professor of the University of Luxembourg and head of the CritiX research lab at UL's SnT Centre. Previously, he has been with the University of Lisbon. He is Fellow of IEEE and of ACM, chair of the IFIP WG 10.4 on Dependable Computing and Fault-Tolerance and vice chair of the Steering Committee of the DSN conference, as well as associate editor of the *IEEE Transactions on Computers*. He is author of over 180 peer-refereed publications and co-author of five books. Contact him at paulo.verissimo@uni.lu.

Cátia Magalhães received an MSc degree in computer science from the University of Lisbon, Faculty of Sciences. Currently, she is working in the digital marketing area as software developer. For the past two years, she has worked with Java technologies and has participated in development projects, mainly in banking and telcos areas. Contact her at catiamagalhaes27@gmail.com.

Fernando M.V. Ramos is an assistant professor in the Department of Computer Science and Engineering at the Faculty of Sciences of the University of Lisbon. His research interests lie at the intersection of networking, systems, and security. Ramos has a PhD in computer science from the University of Cambridge. He is a member of the ACM, the IEEE, and the MEF research council. Contact him at fvramos@ciencias.ulisboa.pt.