The KISS principle in Software-Defined Networking: An architecture for Keeping It Simple and Secure

Diego Kreutz, Paulo Esteves-Verissimo
SnT, University of Luxembourg, Luxembourg
{Diego.Kreutz,Paulo.Verissimo}@uni.lu

Cátia Magalhães, Fernando M. V. Ramos
LaSIGE/FCUL, University of Lisboa, Portugal
catiamagalhaes27@gmail.com, fvramos@ciencias.ulisboa.pt

Abstract—Security is an increasingly fundamental requirement in Software-Defined Networking (SDN). We estimate the slow pace of adoption of secure mechanisms to be a consequence of the overhead of traditional solutions and of the complexity of the support infrastructure required. In this paper we address these two problems as a first step towards defining a secure SDN architecture. First, we investigate the impact of essential security primitives on control plane performance. Our analysis sheds light on the importance of a judicious choice of security primitives (and of their implementations) to obtain security at reasonable cost. Based on these findings, we propose a lightweight security architecture specifically tailored to SDN. While offering the same security properties of standard TLS and PKI implementations, our solution is less complex, prone to be more robust, with a code footprint one order of magnitude smaller, and finally, outperforming reference alternatives such as OpenSSL, by around 30%.

I. INTRODUCTION

In Software-Defined Networking (SDN), network control is separated from the forwarding devices and logically centralised 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 TLS 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]. Moreover, most deployments communicate control plane traffic in-band with data traffic to reduce the infrastructure required, making control plane communication vulnerable to different attacks [1]. For instance, a single malicious forwarding device can intercept control traffic and become a dangerous threat to the SDN infrastructure [2].

Three 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 communications latency and reduced performance. Several recent studies have analysed this overhead in various contexts [3], [4], [5]. Second, the computing capabilities of commodity switches are typically weak. The typical SDN switch (e.g., [6], [7], [8]) is equipped with a single or dual-core CPU running at approximately 1GHz, which compares unfavourably with the multi-core CPUs found in typical commodity servers. Imposing the additional cost of TLS to these computing-constrained networking devices is a problem. Finally, the Public Key Infrastructure (PKI) on which TLS relies is complex [9], leading to a large surface for vulnerabilities which makes it a recurring target of successful attacks [10], [11].

In face of these challenges, in this paper we make three contributions. First, we quantify the impact of security primitives on the performance and scalability of control plane communications (Section II). This is a necessary first step not only to understand one of the main reasons of the slow adoption of these mechanisms in SDN, but also to provide guidelines for the security architecture we propose next. When evaluating the costs of security, we focus on the impact of hashing and message authentication codes (MACs), as they form the basis of nearly all security protocols. We compare different implementations of these primitives on both a modern high-end server system and on an embedded system comparable to those found in commodity switches.

Our second contribution is an identity-based authentication architecture to replace the classic PKI in SDN contexts (Section III). Its main goal is to increase the robustness of SDN control communications by decreasing the complexity of the support infrastructure. The design we propose is centred in an anchor of trust that “logically centralises” security-related services. Our solution includes a novel component – the integrated device verification value (iDVV) – our third contribution (Section IV). The concept was inspired by the iCVVs (integrated card verification values) used in credit cards to authenticate and authorize transactions 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 techniques and efficient cryptographic primitives (drawing from the analysis in Section II). iDVVs can safely replace existing key-exchange protocols and key derivation mechanisms, such as those used in TLS. As a result, the iDVV, while achieving the same functional level of security, is simpler, leading to a higher level of implementation robustness by vulnerability reduction. In fact, we estimate the proposed security architecture footprint to be smaller than traditional PKI alternatives by an order of magnitude, in terms of the number of lines of code (LOC). This metric is the one typically used to assess the robustness of software systems.

II. ON THE COST OF SECURITY

In this section we provide a quantitative analysis of the impact of cryptographic primitives on control plane communication. We focus our analysis on hashes and MACs as they are the essential primitives for authenticity and integrity of communication. We compare the performance of 50+ hashing and MAC primitives, including different implementations.
such as those provided by OpenSSL (version 1.0.0) and PolarSSL (version 1.3.9), two of the most widely used SSL libraries. We evaluate these primitives using a hardware platform that includes two quad-core Intel Xeon E5620 2.4GHz, with 2x4x256KB L2 / 2x12MB L3 cache, 32GB SDIMM at 1066MHz, with hyper-threading enabled and overclocking and dynamic CPU frequency scaling disabled. These machines ran Ubuntu Server 14.04 LTS and were connected via Gigabit Ethernet. To measure the latency of control plane communication we used Linux’s resource usage system call (getrusage()) to get the user CPU execution time. This function is commonly used to measure the performance of cryptographic primitives [12].

A. Connection and communication costs

Our first experiments assess the average latency of TCP and TLS on control plane communication. We analyse 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 (e.g., 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. One of the two nodes of the evaluation platform emulates the controller, whereas the other assumes the role of the forwarding devices. The emulated controllers and forwarding devices are implemented in C, using the OpenSSL and PolarSSL TLS implementations in their standard configuration (i.e., no library-specific optimizations were applied). Figures 1 and 2 show the median of the measured latency over 40k executions. The standard deviation is below 3% so we do not include it in the figures.

Figure 1 shows the connection setup time (per forwarding device). The higher costs of the two TLS implementations are due to the execution of a more elaborate handshake protocol between the devices. While TCP uses a simple three-way handshake, TLS requires a twelve message handshake for mutual authentication of the communicating entities. As expected, the overhead increases with the number of forwarding devices. Interestingly, our results also suggest that the choice of implementation has a non-negligible performance impact. For connection setup, PolarSSL induces nearly twice the overhead of OpenSSL.

Although important, a high connection cost can be amortized by maintaining persistent connections. As such, the communications cost is usually considered more relevant. Figure 2 shows the latency of FLOW_MOD messages (56 bytes, as specified in OpenFlow 1.4 [13]), averaged over 10k messages. The results with PACKET_IN messages (32 bytes) 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 3x higher than TCP. This is explained by the high overhead of cryptographic primitives, as we further analyse in the next section. PolarSSL is significantly worse, increasing the latency by up to 7x when compared with TCP.

The main findings of this analysis can be summarised in two points. First, the additional computation required by the cryptographic primitives used in TLS leads to a non-negligible performance penalty in the control plane. Second, different implementations of TLS present very different performance penalties.

B. The impact of cryptographic primitives

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

To measure the overhead of these primitives we disabled hyper-threading, in order to remove noise and randomness due to the implied resource sharing. As commodity switching devices do not implement direct cache access, we have ensured that the data to be hashed resides in main memory. This avoids artificial performance boosts when operating on cached data. To mimic the behaviour of a switch, we circulated over an input buffer that is twice as large as the last-level cache (L3) to ensure that every read resulted in a cache miss. The numbers in the following graphs represent the median of 1M executions, with a standard deviation below 3%.

We analyse the performance of nine hashing primitives. The results are presented in Figure 3. The red bars represent

---

1With cached data, we observed artificial gains of up to 20% for hashing and of 12% for MAC primitives.
primitives that are provided by OpenSSL, while blue 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 the former performs more calculations (80 rounds of its compression function) over 1024 bits of data at a time, while the latter executes 64 rounds over just 512 bits of data. In the case of KECCAK the difference in performance is due to the additional computational complexity of the mechanisms employed. This solution requires 24 rounds of permutation on each compression step, while BLAKE requires up to 16 rounds.

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. OpenSSL implementations are the ones showing the best performance for hashing primitives. Curiously, this does not hold for other primitives such as HMAC, as we show later in Figure 5. With the exception of RIPEMD160, the PolarSSL implementation always presented higher message latencies. In addition to OpenSSL and PolarSSL, we included EVP, a library that provides a high-level interface to cryptographic functions. Its main purpose is the ability to replace cryptographic algorithms without having to modify applications. The added flexibility comes with a cost, as we can observe in the results. The same OpenSSL primitives used through an EVP interface experience a penalty between 3% and 15%.

Finally, Figure 5 shows the results of the latency analysis of six MAC primitives. It is clear that Poly1305-AES outperformed all other primitives. This MAC was designed to provide security with high speed and low per-message overhead. Our results confirm that it fulfils this objective, being approximately two times faster than OpenSSL’s HMAC-SHA1, and close to four times faster than HMAC-SHA512, for instance. For MAC primitives, the choice of specific implementations remains relevant. Curiously, in this case the PolarSSL implementation always outperformed the equivalent OpenSSL implementation. The reason may lie on the fact that OpenSSL does not provide native HMAC implementations, but rather highly configurable HMACs through EVP interfaces. These primitives thus carry the overhead of EVP and the extra costs of configurability.

C. The impact of securing the control plane

Although the number of use cases is expanding, SDN has been mainly targeting data centers. As such, SDN controllers have to 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 mouse traffic lasting less than 100ms [14]. 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], [15]. In the following we try to put the results we obtained in perspective, by analysing the effect of including even the most basic security primitives to ensure authenticity and integrity when considering peak loads of this scale.

Considering the MAC primitive with best performance in the analysis (Poly1305-AES), around 20 dedicated cores are needed to compute a MAC in order to maintain a rate of
20M flows/s. To understand the importance of judiciously selecting the security primitive implementation, the HMAC-SHA512 OpenSSL (worst case performance in the analysis) would require up to 65 cores to compute MACs at these rates.

In summary, our findings indicate that (i) the inclusion of cryptographic primitives results in a non-negligible performance impact on the latency and throughput of the control plane; and that (ii) 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, we have selected the primitives with best performance – Poly1305-AES and SHA512 OpenSSL – to implement the various mechanisms of the architecture we propose next (Section III). Taken together, they provide the best trade-off between security and performance for control plane communications in SDN.

III. KISS ARCHITECTURE

In this section we present our proposal of a modular security architecture for SDN, as an alternative to a classic PKI. Its main goal is to provide security properties including authenticity, integrity, and confidentiality for control plane communications, while minimizing cost and complexity. The architecture is composed of three main components, as illustrated in Figure 6: controllers, forwarding devices, and an anchor of trust. A cryptographic toolbox (CryptoT), comprised of two modules – the iDVV and NaCl – is included in both the controller and the switches. NaCl [16], which is a high performance yet secure cryptographic library, replaces the asymmetric and symmetric ciphering of TLS implementations (such as OpenSSL). The iDVV is a novel component we propose that helps to ensure the security of control plane communications through authentication and authorization codes generated at a low cost. The anchor of trust (AoT) provides two security-related services: device registration and device association. The objective of proposing a modular design centered in an anchor of trust is to reduce the complexity of forwarding devices and controllers by “logically centralizing” security-related services.

The device registration service can be seen as a trust path mechanism for bootstrapping the essential security requirements of network infrastructures. A specific protocol is provided to network administrators to register new devices. During registration, each device receives a unique ID and two secrets used to bootstrap the iDVV module, which will be subsequently required for ensuring secure communication between networking devices and the AoT.

The association service is used by any two networking devices to establish a new association between them. The most common case is a forwarding device requesting an association to a controller. This association service then prepares the iDVV mechanism (explained next) by generating and installing two secret values indistinguishable from random (assocID and seed), in both the controller and the switch. These secrets – an initial seed and a unique association ID – are therefore known only to the communicating devices. It is important to emphasize that the AoT does not keep any sensitive data regarding the association between devices. After the seed and association ID are installed on the devices, the eventual compromise of the AoT will not endanger the communications between devices.

A. Why do size and complexity matter?

The more complex the system, the higher the probability of having vulnerabilities and hence a broader attack surface. Nowadays, this is still one of the major problems faced by the technology industry. Specialized security reports have recurrently highlighted the complexity and size of systems as one of the most important security challenges [17]. The time for re-thinking the security of communication channels may have come, and that is also the position we take in this paper.

Renowned cryptographers and security experts have been claiming that simplicity is one of the keys in securing computer systems [16], [18], [19], [20]. In fact, the trusted computing community has been advocating simple interfaces and and concerned with the size and complexity of components for a long time [21], [22], [23], [24]. These positions have in essence been pointing to the KISS principle, i.e., keep it simple, stupid.

Another example are cryptographic libraries, such as NaCl [16], which are starting to appear and be used in different applications. Libraries such as NaCl are proven to be secure and can be embedded in any system. They take a clean slate approach, avoiding most of the pitfalls of other libraries (e.g., OpenSSL – misuse issues). One of the most important take aways of NaCl is its reduced size and its minimal complexity in terms of interface (less error prone) and sub-protocols (if any). Moreover, TweetNaCl was announced as the smallest implementation of high-security cryptographic library, which makes it a good candidate for inclusion into trusted code bases of computer systems. However, a library such as NaCl, by itself, is not enough. We complement it in our architecture with the iDVV mechanism, described ahead, to generate the security sensitive data (e.g., keys) used by NaCl ciphers, for instance.

B. On the cost of the infrastructure

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 take away is that our solution has nearly one order of magnitude less LOC (85k) and uses four times less external libraries and only four programming languages. This makes a huge difference from a security and dependability perspective. For instance, to formally prove more than 717k LOC is by itself a tremendous challenge. And it gets considerably worse if we take into account eighty external libraries and eleven development languages.

Second, OpenSSL is complex and highly configurable. This has been the source of many security incidents, i.e., developers and users frequently use the library in an inappropriate way [25], [26], [27]. It has also been shown that the majority of the security incidents are still caused by errors and misconfiguration of systems [28], [29]. Lastly, recent research has uncovered new vulnerabilities on TLS implementations and also shown that the whole protocol, including the extensive cryptography, is very unlikely to be verified in a near future [30].

In contrast, our proposed architecture exhibits gains in both performance and robustness, contributing to solving the dilemma we enunciated in the introduction. By having less LOC, we significantly reduce the threat surface – by one order of magnitude – and by combining NaCl, the anchor of trust, and the iDVV mechanism, we provide a potentially equivalent level of security, but quite increased performance: the bootstrap process (device registration and association) is much faster and our connection latency is also significantly lower.

IV. KEEP IT SIMPLE AND SECURE

iCVVs are used in credit cards to authenticate and authorize transactions in a secure and inexpensive way. An iCVV is a unique value generated by the credit card which can be verified by the bank. Similarly, iDVVs are sequentially generated to authenticate and authorize requests between two networking devices. However, iDVV generation is made flexible to serve the needs of SDN. iDVVs can therefore be generated: (a) on a per message basis; (b) for a sequence of messages; (c) for a specific interval of time; and (d) for one communication session. The main advantages of iDVVs are their low cost and the fact that they can be generated off-line, i.e., without having to establish any previous agreement.

As iDVVs can be used by cryptographic primitives such as MAC functions, they have to be indistinguishable from random. Hashing primitives are natural choices since they provide indistinguishable-from-random values if one or more of the input values are known only by the sender and the receiver. Our iDVV generation is based on two internal secrets, which means that without knowing both values an attacker will not be able to generate a valid iDVV. To ensure this property, we use the two values generated during the association phase that are known only by the associated devices, the unique association identifier assocID and the evolving seed [2]. These values are used by the iDVV to autonomously generate on-demand authentication and authorization keys that will be used for securing the communications. The assocID and the seed are initialized by a resilient pseudo-random generator provided by the anchor of trust (see Section III). The seed evolves to an indistinguishable-from-random value each time a new iDVV is generated. The new seed is the outcome of an XOR operation between the current iDVV and the output of a hashing primitive H. As both values used by the XOR operation are outputs of hashing primitives, the evolution of the seed follows a scheme similar to the PRF of TLS [31]. The next iDVV is the outcome of a hashing primitive H using as input the newly generated seed and the association ID, which are values known only by the communicating devices.

From a security perspective, one iDVV per message makes it much harder for attacks such as key recovery [32], advanced side channel attacks [33], among other general HMAC attacks [34], to succeed. In fact, the one-time key approach was part of the original MAC proposal and was only removed (i.e., replaced by keys with a longer life time) for performance reasons. However, we can incur that penalty because the iDVV generation has low cost (see Section IV-B).

A. Synchronization

The iDVV solution is able to cope with message losses on the communication channel without requiring the use of a reliable transport channel such as TCP. This allows the use of unreliable – but lower overhead – UDP channels. When a packet loss occurs, the receiver is not able to verify the authenticity and integrity of the next received message. In this case, the iDVV is advanced a number of times (pre-defined) until it is able to verify the incoming message. However, when packet loss is high the system may reach a point when advancing the iDVV by the pre-defined amount is not enough. In this case, the receiver initiates the iDVV synchronization phase to speed up the process. The synchronization step significantly hardens the system by setting a new starting point for the iDVV generators.

This process is depicted in Algorithm 1. The synchronization starts with one of the devices, called the initiator (I). This device generates a verification value sv, which is the result of a HMAC primitive using a special purpose secret S, of at least 32 bytes, which is initialized during the device registration phase), shared between the two devices, and the initiator’s id (I_ID) plus a nonce x that identifies the current synchronization request and helps avoid replay attacks. Both values sv and x are sent by the initiator to the receiver (R).

Algorithm 1: iDVV synchronization

| I → R | HMAC(S, I_ID || x) Generates verification value sv |
| I → R | HMAC(S, R_ID || x) Generates verification value sv' |
| I → R | HMAC(S, I_ID || x) Generates commit value sv'' |
| I → R | HMAC(S, sv'' || x) Commit new seed value |

The second device generates its own value sv' using his id (R_ID) and the received nonce. This value sv' is sent to the initiator as an ack of the synchronization. After receiving and verifying the ack, the initiator generates a new value sv'' which is sent to the receiver. The new seed is the resulting output of an HMAC primitive, using as input the shared secret S, the value sv' and the nonce x. This newly generated seed is used to update the internal state of the iDVV mechanism.
B. On the cost of iDVVs

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 (by an order of magnitude), namely, the key-exchange algorithms embedded in the OpenSSL implementation. Considering a setup with 128 switching devices, our results show our proposed solution (iDVV + NaCl’s ciphers) to be more than 30% 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.

V. CONCLUDING REMARKS

This paper explored the possible reasons behind a slower than expected adoption of security mechanisms in SDN, and based on those findings, proposed some solutions, in the way of a novel modular security architecture for SDN.

We started by investigating the impact of essential cryptographic primitives and TLS on the control plane performance. We showed that whilst even the most basic security primitives add a non-negligible degradation of performance, a judicious choice of these primitives and their specific implementations can mitigate the penalty significantly. This is particularly important for the typical SDN scenario that resorts to commodity hardware.

The second problem we explored in this paper was the complexity of the support infrastructure for secure communications. We proposed a simple and robust architecture centered on an anchor of trust and on novel mechanisms for sharing the secrets necessary for secure communications between network devices. The modularity of our solution and the logical centralization of the fundamental security services is important to minimize the complexity introduced in switches and controllers.

Our results are encouraging in terms of an increase of performance (30% improvement over OpenSSL) and robustness (an order of magnitude reduction in the number of LOC). We believe that this is one first step towards lightweight but effective security for control plane communication. We make a “call to arms” to foster developments on securing control plane communications without impairing performance, a fundamental pre-condition for widespread adoption by future SDN deployments.

REFERENCES

[1] D. Kreutz, F. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” Proceedings of the IEEE, vol. 103, no. 1, Jan 2015.

[2] M. Antikainen, T. Aura, and M. Särelä, “Spook in your network: Attacking an SDN with a compromised openflow switch,” in Secure IT Systems, ser. Lecture Notes in Computer Science, K. Bernsmed and S. Fischer-Hübner, Eds. Springer International Publishing, 2014, pp. 229–244. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-11599-3_14

[3] C. Shen, E. Nahum, H. Schulzrinne, and C. P. Wright, “The impact of TLS on SIP server performance: Measurement and modeling.” IEEE/ACM Trans. on Networking, vol. 20, no. 4, Aug 2012.

[4] J. Schonwalder and V. Marinov, “On the impact of security protocols on the performance of SNMP,” IEEE Trans. on Net. and Service Management, vol. 8, no. 1, 2011.

[5] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafo, K. Papagiannaki, , and P. Steenkiste, “The cost of the “S” in HTTPS,” in Proceedings of the Tenth ACM Conference on Emerging Networking Experiments and Technologies, ser. CoNEXT ’14. New York, NY, USA: ACM, 2014, p. 7. [Online]. Available: http://doi.acm.org/10.1145/2535372.2535416

[6] Hewlett-Packard Development Company, L.P., “HP 3800 switch series,” 2015. [Online]. Available: http://h17078.www1.hp.com/docs/products/4aa3-7115enw.pdf

[7] Advantech Co., Ltd., “Freescale introduces a new breed of intelligent switching for the SDN era,” 2015. [Online]. Available: http://www.advantech.com

[8] NEXCOM International Co., Ltd., “NSA 5640,” 2015. [Online]. Available: http://www.nexcom.com/ [9] A. Wazan, R. Laborde, F. Barrere, A. Benzekri, and D. Chadwick, “PKI interoperability: Still an issue? a solution in the x.509 realm,” in Info. Assurance and Sec. Education and Training. Springer, 2013, vol. 406.

[10] N. van der Meulen, “DigiNotar: Dissecting the first dutch digital disaster,” Journal of Strategic Security, vol. 6, no. 2, 2013.

[11] D. Kaminsky, M. Patterson, and L. Sassaman, “PKI layer cake: New collision attacks against the global X.509 infrastructure,” in Financial Crypto. and Data Sec., ser. L. N. in CS. Springer, 2010, vol. 6052.

[12] VAMPIRE, “SUPERCOP,” 2015, http://bench.cr yp.to/super cop.html. [Online]. Available: http://bench.cr yp.to/super cop.html

[13] ONF, “Openflow switch specification (version 1.4.0),” October 2013. [Online]. Available: http://goo.gl/d1Djxw6

[14] T. Benson, A. Akella, and D. A. Maltz, “Network traffic characteristics of data centers in the wild,” in Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, ser. IMC ’10. New York, NY, USA: ACM, 2010, pp. 267–280. [Online]. Available: http://doi.acm.org/10.1145/1879141.1879175

[15] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, “Maple: Simplifying sdn programming using algorithmic policies,” in ACM SIGCOMM Computer Communication Review, vol. 43, no. 4. ACM, 2013, pp. 87–98.

[16] D. Bernstein, T. Lange, and P. Schwabe, “The security impact of a new cryptographic library,” in Progress in Cryptology - LATINCRYPT, ser. L. N. in CS. Springer, 2012, vol. 7533.

[17] Cisco, “Annual security report,” 2014. [Online]. Available: https://www.cisco.com/web/offer/gist_ty2_asset/Cisco_2014 ASR.pdf

[18] M. Green, “The anatomy of a bad idea,” 2012. [Online]. Available: http://blog.cryptographyengineering.com/2012/12/the-anatomy-of-bad-idea.html

[19] Stanford University, “Stanford cybersecurity expert: Dan Boneh,” 2015. [Online]. Available: https://www.youtube.com/watch?v=H-YGdcNFBJk

[20] B. Preneel, “System security after Snowden,” 2015. [Online]. Available: http://2015.dsn.org/keynote-speakers/H-YGdcNFBJk

[21] G. Heiser, T. Murray, and G. Klein, “It’s time for trustworthy systems,” IEEE/ACM Trans. on Networking, vol. 20, no. 4, Aug 2012.

[22] M. Antikainen, T. Aura, and M. Särelä, “Spook in your network: Attacking an SDN with a compromised openflow switch,” in Secure IT Systems, ser. Lecture Notes in Computer Science, K. Bernsmed and S. Fischer-Hübner, Eds. Springer International Publishing, 2014, pp. 229–244. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-11599-3_14

[23] C. Shen, E. Nahum, H. Schulzrinne, and C. P. Wright, “The impact of TLS on SIP server performance: Measurement and modeling.” IEEE/ACM Trans. on Networking, vol. 20, no. 4, Aug 2012.
“TrustVisor: Efficient TCB reduction and attestation,” in *Security and Privacy (SP)*, 2010 IEEE Symposium on, May 2010, pp. 143–158.

[24] U. Steinberg and B. Kauer, “NOVA: A microhypervisor-based secure virtualization architecture,” in *Proceedings of the 5th European Conference on Computer Systems*, ser. EuroSys ’10. New York, NY, USA: ACM, 2010, pp. 209–222. [Online]. Available: http://doi.acm.org/10.1145/1755913.1755935

[25] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, “An empirical study of cryptographic misuse in Android applications,” in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS ’13. New York, NY, USA: ACM, 2013, pp. 73–84. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516693

[26] S. H. Kim, D. Han, and D. H. Lee, “Predictability of Android OpenSSL’s pseudo random number generator,” in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS ’13. New York, NY, USA: ACM, 2013, pp. 659–668. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516706

[27] S. Fahl, M. Harbach, T. Maders, L. Baumgärtner, B. Freisleben, and M. Smith, “Why eve and mallory love Android: an analysis of Android SSL (in)security,” in *Proceedings of the 2012 ACM conference on Computer and communications security*, ser. CCS ’12. New York, NY, USA: ACM, 2012, pp. 50–61. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382205

[28] Verizon, “2015 data breach investigations report,” Verizon, Tech. Rep., 2015. [Online]. Available: http://www.verizonenterprise.com/DBIR/2015/

[29] J. Zhang, Z. Durumeric, M. Bailey, M. Lia, and M. Karir, “On the mismanagement and maliciousness of networks,” in *Symposium on Network and Distributed System Security (NDSS)*, 2014.

[30] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue, “A messy state of the union: Taming the composite state machines of TLS,” in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 535–552.

[31] T. Dierks and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” RFC 5246 (Proposed Standard), IETF, Aug. 2008, updated by RFCs 5746, 5878, 6176.

[32] H. Handschuh and B. Preneel, “Key-recovery attacks on universal hash function based mac algorithms,” in *Advances in Cryptology - CRYPTO*, ser. L. N. in CS. D. Wagner, Ed. Springer, 2008, vol. 5157.

[33] O. Benoit and T. Peyrin, “Side-channel analysis of six sha-3 candidates,” in *Cryptographic Hardware and Embedded Systems, CHES 2010*, ser. L. N. in CS. Springer, 2010, vol. 6225, pp. 140–157.

[34] J. Kim, A. Biryukov, B. Preneel, and S. Hong, “On the security of HMAC and NMAC based on HAVAL, MD4, MD5, SHA-0 and SHA-1 (extended abstract),” in *Security and Cryptography for Networks*, ser. L. N. in CS. Springer, 2006, vol. 4116.