

## Note-taking of *Benchmarking the Security Protocol and Data Model (SPDM) for component authentication*, Renan C. A. Alves, Bruno C. Albertini, Marcos A. Simplicio Jr

### Introduction

- At every stage of the supply chain, there is a risk that malicious entities may inconspicuously tamper with or substitute components.
- The usual protection techniques that operate at the operating system level (e.g., antivirus software) or techniques that act at the infrastructure level (e.g., firewalls) are oblivious to such threats.
- SPDM was recently proposed to address such low-level security challenges.
- Firmware measurements enable system components to be verified, ensuring they have not been victim of tampering, while establishing sessions keys avoids passive eavesdropping by malicious components attempting to steal data.
- Since SPDM is a relatively recent proposal, there is a lack of studies evaluating its performance impact on real-world applications.
- <https://github.com/rcaalves/spdm-benchmark>
- The goal of this paper is to assess the overhead introduced by each phase of the SPDM message flow, in an emulated environment.

### SPDM overhead assessment

- They used a Random Number Generator (RNG) device.
- Each of execution step was performed 100 times, aiming to obtain statistical confidence.
- As expected, most of the overhead was due to messages related to the authentication process.
- The *GetCertificate* procedure is expected to be slow since:
  - it may require several messages to finish
  - by the end of it, the retrieved certificate must be verified for correctness, which requires a few signature verifications.
- *KeyExchange*, in turn, involves the generation of a symmetric key pair by means of a Diffie-Hellman key exchange.
  - The usage of Pre-Shared Keys (PSK) considerably reduces the burden of establishing session keys.
- Retrieving measurements all at once is slightly faster than retrieving measurements one by one.
- SPDM led to a 6.4-fold increase in terms of time.
  - RNG was designed to be extremely fast, so even cryptographic operations that are quite lightweight in absolute numbers, like symmetric encryption, become comparatively expensive.
- The responder handles *GetCertificate* messages faster than *KeyExchange*. The reason behind this behavior is that most of the cryptographic processing of *GetCertificate* remains at the requester side.
  - The largest overhead observed at the responder, however, was the time to load certificates from the disk.

## Hard drive use case

- The driver fills the role of the SPDM requester, while the hard drive takes the role of SPDM responder.
- They used a few widely employed tools and benchmarking utilities to assess hard drive performance: *dd*, *hdparm*, *ioping*, *bonnie++*, *fio*.
- *dd*: SPDM caused a  $\approx 68\%$  slowdown in writing speed.
- *hdparm*: provides “an indication of how fast the drive can sustain sequential data reads under Linux, without any filesystem overhead.”
  - Without SPDM, average read speed observed was 3.9 GB/s.
  - With SPDM, average read speed observed was 28 kB/s, which translates to a 99.3% speed degradation.
- *ioping*: confidence intervals were very large, making it hard to draw statistically relevant conclusions.
- *bonnie++*: both reading and writing speed drop to the same order of magnitude, since the system bottleneck is the same in both tests: the processing cost of encrypting/decrypting every transaction.
- *fio*: performance degradation becomes less prominent when randomness is introduced.
  - Bottleneck shifting from the cryptographic operations to the physical disk operations, (frequent seek operations to address the random request locations).