



Travail d'Étude et de Recherche (TER)

Device security and Security Protocol and Data Model (SPDM)

Master's degree in computer science - SIRIS program
Science et Ingénierie des Réseaux, de l'Internet et des Systèmes

Presented by
Léon GALL
leon.gall@etu.unistra.fr

Supervised by
Pierre DAVID
pda@unistra.fr

Contents

Introduction	2
1 Threat analysis	3
1.1 Context	3
1.2 Man In The Middle attack	3
1.3 Spoofing attack	4
1.4 Attack vectors	4
2 Traditional solutions	6
2.1 Partial solutions	6
2.1.1 White box and sluice	6
2.1.2 Disk encryption	7
2.1.3 Authentication in Internet of Things (IoT)	7
2.2 Virtualization	8
3 Security Protocol and Data Model (SPDM)	10
3.1 Context	10
3.2 The different phases of the protocol	10
3.2.1 Device initialization phase	10
3.2.2 Version-Capabilities-Algorithms (VCA) phase	11
3.2.3 Options phase	11
3.2.4 Session phase	11
3.3 Security features of SPDM	13
3.4 Limitations	13
3.4.1 Potential design pitfalls	14
3.4.2 Benchmark of SPDM	14
Conclusion	16

Introduction

In the ever-evolving landscape of digital technology, the security of devices stands as a critical concern. The proliferation of interconnected devices has brought in unprecedented convenience and efficiency, but it has also exposed systems to a myriad of potential threats. Within this context, the absence of a robust security protocol leaves peripherals vulnerable to exploitation and compromise, posing a significant challenge for both industry stakeholders and cybersecurity experts.

In this context, the Distributed Management Task Force (DMTF) emerges as a central figure, convening industry leaders such as Intel, Cisco, and Broadcom to address these challenges. Among the standards promulgated by the DMTF, the Security Protocol and Data Model (SPDM) is of particular importance. Originating in 2019 and currently evolving through version 1.3.0, SPDM embodies a concerted effort to strengthen authentication and data exchange within peripheral ecosystems, drawing inspiration from the principles of TLS 1.3.

As part of my *Travail d'Étude et de Recherche* (TER), I conducted an in-depth investigation into the security of peripheral devices with the following goal: to produce a comprehensive state-of-the-art report on securing access to peripheral devices. This included an investigation of the range of threats posed by the lack of a robust security protocol, an examination of the traditional solutions used before SPDM, an evaluation of the innovative solutions provided by SPDM, and an analysis of their collective impact on operating systems and drivers.

Under the guidance and supervision of Pierre DAVID, my work unfolded in three distinct phases, each of which constitutes a separate section of this report: threat analysis, exploration of traditional solutions, and an in-depth examination of SPDM. Through this meticulous examination, I aimed to contribute to the ongoing discourse surrounding device security and to provide insights that can inform future advancements in this critical field.

1 Threat analysis

1.1 Context

In the context of device security, and more generally in the context of computer systems, security encompasses several vital aspects: confidentiality, authentication, integrity, and availability. Failure to adhere to any of these security principles can lead to various threats and vulnerabilities. This brief introduction is intended to recall these principles.

Confidentiality ensures that transferred data remains unintelligible to unauthorized parties. This is achieved through symmetric or asymmetric cryptography. Symmetric cryptography uses a shared key between the sender and receiver, while asymmetric cryptography requires each party to have a unique public/private key pair. The sender encrypts data using the receiver's public key, ensuring that only the receiver with the corresponding private key can decrypt it.

Authentication verifies the identities of communicating parties, typically through certificates. Messages are signed with the sender's private key, allowing the receiver to authenticate the sender by decrypting the message with the sender's public key. This process also prevents message repudiation.

Integrity ensures that transmitted messages remain unaltered by third parties. This is achieved by including a hash of the initial message within the transmitted message, typically generated through a one-way function. Authentication and integrity can be combined using a Message Authentication Code (MAC, [1]).

While availability is crucial for web servers, ensuring they remain accessible even during heavy traffic or DDoS attacks, it falls outside the scope of this paper, which primarily focuses on device security.

As mentioned above, a threat may emerge as soon as one of the aspects of security is not respected. In the context of device security, there are two main types of threats: Man In The Middle attack and spoofing attack. These attacks may be initiated from a variety of entry points, which are referred to as attack vectors. The subsequent section will present these threats and the different attack vectors.

1.2 Man In The Middle attack

A man-in-the-middle (MITM) attack is a significant cybersecurity threat in which an attacker secretly positions himself between two communicating parties, intercepts messages, and relays them without either party's knowledge. This malicious tactic allows the attacker to eavesdrop on or modify the messages being exchanged, potentially compromising the integrity and confidentiality of the communication.

In the realm of device security, MITM attacks can manifest in various forms, such as keyloggers, discreetly positioned between the keyboard and the computer, clandestinely recording every keystroke, including sensitive passwords. Additionally, any device capable of intercepting data, commonly known as a packet sniffer, poses a significant risk. This vulnerability is particularly alarming in the context of autonomous cars, where compromised communication channels could have severe consequences, potentially compromising passenger safety and sys-

tem integrity.

To illustrate this kind of attack, consider the case of a keylogger. A malicious individual could insert such a device between the USB port of a computer and a keyboard. When a user presses a key, the keylogger receives the signal from the keyboard, records the key, and then forwards the signal to the computer. This operation is completely transparent to the user. Depending on the information entered by the user, the attacker can gain access to passwords or even credit card details.

In addition, another concerning manifestation of MITM attacks involves the compromise of the Controller Area Network (CAN) in vehicles. In this scenario, attackers exploit vulnerabilities in the vehicle's CAN bus, a network protocol used for communication among vehicle components such as sensors, actuators, and Electronic Control Units (ECUs). By inserting themselves into the CAN network, attackers can intercept and manipulate messages exchanged between vehicle systems. For instance, an attacker could alter messages related to braking or acceleration, which could have serious consequences such as loss of vehicle control.

Mitigating such attacks requires robust measures ensuring confidentiality, integrity, and critically authentication. Failure to uphold any of these security principles exposes the communication to potential threats and vulnerabilities.

1.3 Spoofing attack

In the context of device security, the second main threat is the spoofing attack. A spoofing attack is a malicious cybersecurity strategy in which an attacker impersonates a legitimate entity or device to perform unauthorized actions. These actions can range from gaining unauthorized access to sensitive information to manipulating system functionality for malicious purposes.

Spoofing attacks are often perpetrated by malicious USB devices or cables. For instance, a malicious USB key can act as a legitimate input device, such as a keyboard. Once connected to a computer, the rogue USB key can emulate keystrokes to access a command line terminal and execute arbitrary commands. This allows the attacker to download malicious files or run harmful scripts, compromising the computer's security without the user's knowledge. Similarly, a malicious cable could impersonate a keyboard and perform similar actions, as illustrated below.

Consider a scenario in which Alice generously lends a USB port on her computer to Mallory, who urgently needs to charge her smartphone. Unbeknownst to Alice, the seemingly innocuous charger Mallory has provided could be a malicious device carefully designed to compromise the security of Alice's computer once connected. In this scenario, the attacker, Mallory, could exploit vulnerabilities in the charger to inject malware directly into Alice's computer system. This malicious act has the potential to compromise sensitive data or covertly install malicious software, all without Alice's knowledge or consent.

Such incidents highlight the critical importance of implementing robust security measures, including device authentication and data integrity checks, to mitigate the risk of spoofing attacks.

1.4 Attack vectors

Device security encompasses a wide range of potential vulnerabilities, with almost any component susceptible to exploitation. While the previous section highlighted the USB port as an example of attack vector, it is important to recognize that every aspect of a device can serve as a potential entry point for malicious actors. These actors can use a variety of tactics, such as replacing legitimate components with compromised ones, inserting malicious devices, or even

directly compromising the device firmware itself.

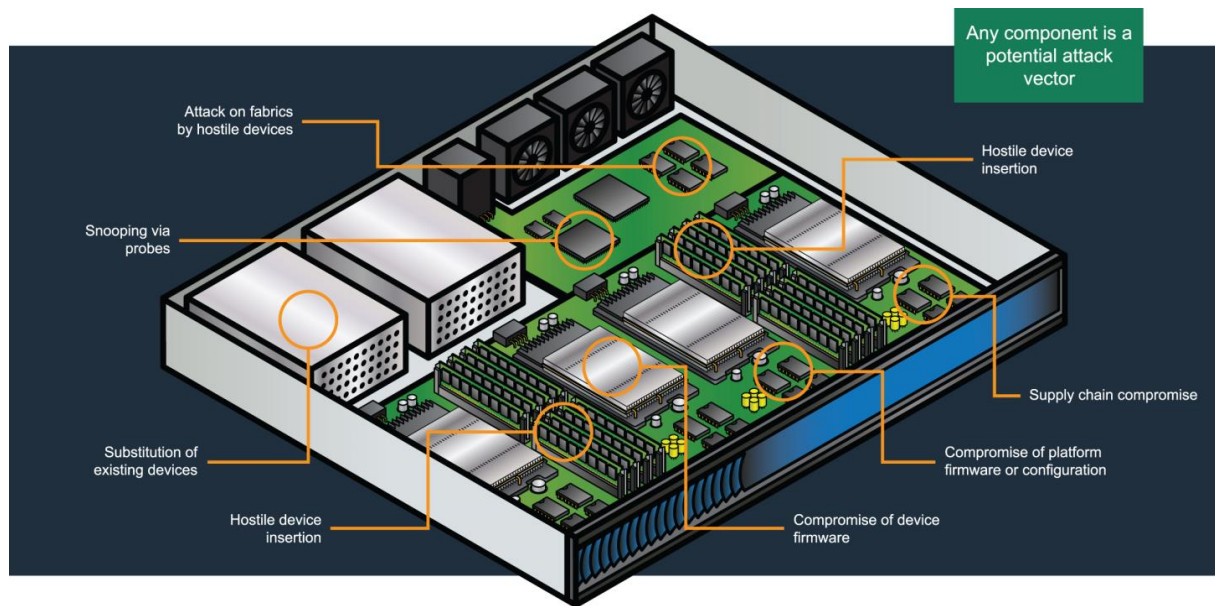


Figure 1.1: Potential attack vectors in a device. Extracted from [2].

In the figure 1.1, we can see the myriad of potential attack vectors within a single device. These include the replacement of existing components, the insertion of hostile devices, and the compromise of device firmware. We can note that physical access to components is required to perform these attacks.

The wide range of attack vectors underscores the need for security measures that address vulnerabilities at all levels of the device architecture.

2 Traditional solutions

In order to mitigate the threats presented in the previous chapter and to ensure the security of the devices, there are some solutions offered in the industry. Unfortunately, these solutions suffer from several limitations and are rarely implemented.

2.1 Partial solutions

2.1.1 White box and sluice

To address these challenges, the French Cybersecurity Agency (ANSSI) recommends implementing either a white box or sluice architecture [3].

A white box configuration involves a server or workstation isolated from the operational network and dedicated exclusively to anti-malware analysis of removable media such as USB keys and the data stored on them. Extensive logging is essential, with logs collected both locally and remotely.

As illustrated in the figure 2.1, when a user has to access a file from a USB stick, he needs to plug the USB stick into the white box. The white box then performs a rigorous security check on the file and transfers it to a secure USB key that is authenticated and signed. Personal computers can only access the data through this designated USB key, ensuring a controlled and secure transfer process.

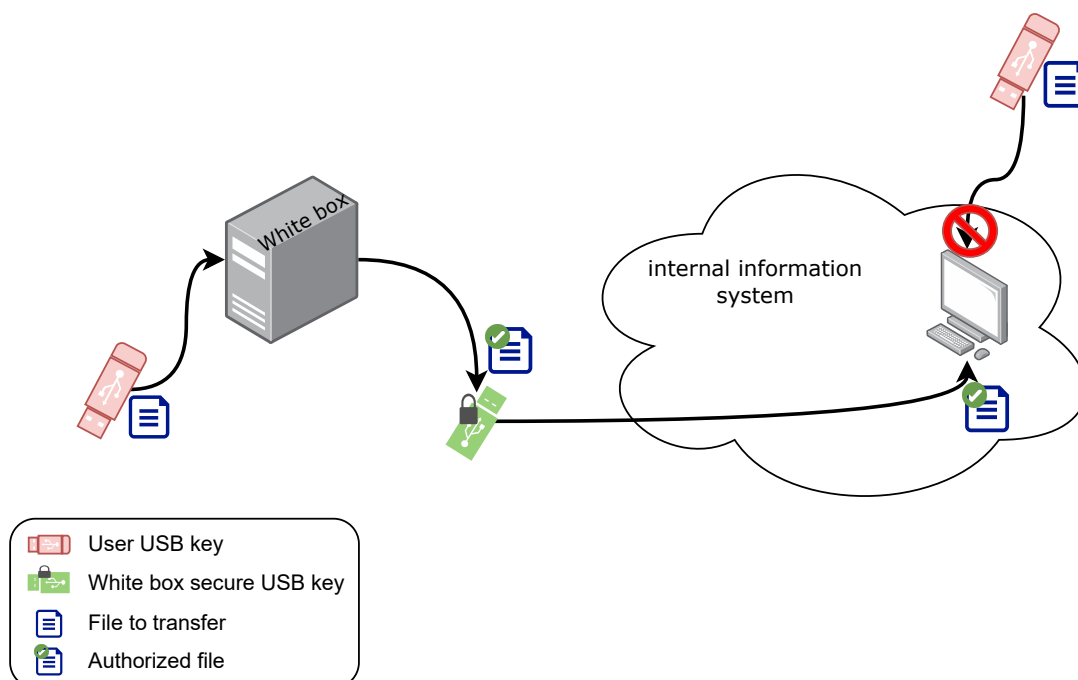


Figure 2.1: White box principle. Based on [3].

A sluice architecture operates similarly to a white box, but is specifically designed for a data insertion point that is physically isolated from the main network. As shown in Figure 2.2,

the sluice validates data before sending it to the data insertion point, where users retrieve the verified data.

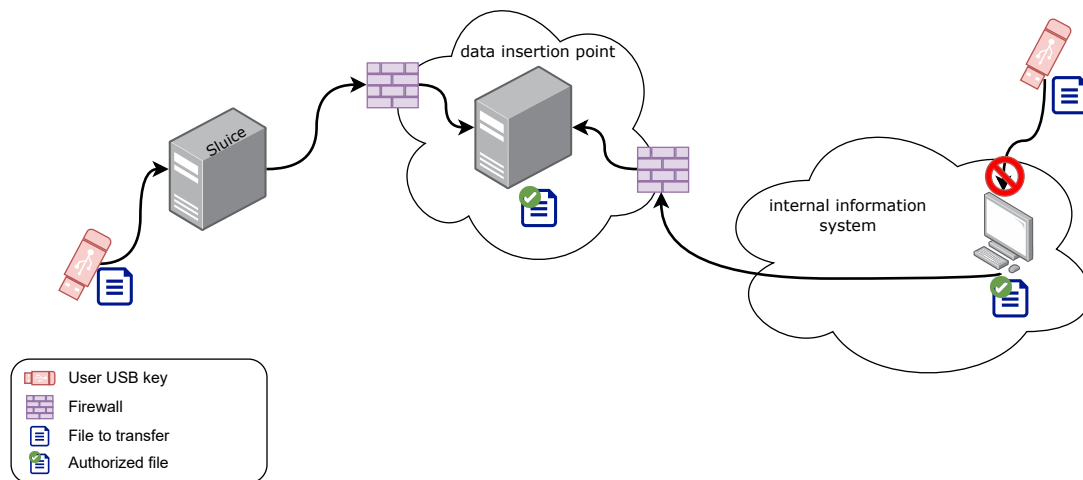


Figure 2.2: Sluice principle. Based on [3].

However, this system has its limitations. It focuses primarily on monitoring data transfers and is highly dependent on the sluice or white-box server. Furthermore, if the server is compromised, all transmitted data is at risk. Although rare, a breach in the Board Management Controller (BMC) could potentially bypass the server's security measures. Most importantly, this solution adds a great amount of complexity to data transfer, requiring the USB key to be passed through a server before the data can be retrieved from another USB key or exchange point.

2.1.2 Disk encryption

While the previous method effectively ensured data integrity and authenticated transmissions, it lacked mechanisms for confidentiality. This section presents a solution for providing confidentiality in storage.

Disk encryption is a security measure designed to protect the contents of storage devices such as hard drives by encrypting sensitive data to prevent unauthorized access. Two common encryption techniques are used: full disk encryption, which encrypts the entire storage device, and file-level encryption, which selectively encrypts specific files or folders.

To implement disk encryption, the disk driver uses cryptographic algorithms to encrypt blocks of data written to the disk and decrypt those read from the disk. Some disks also ensure data integrity by providing a hash of the data within each block. By performing encryption and decryption operations at the device driver level, data remains unintelligible in transit, effectively preventing unauthorized eavesdropping attempts.

However, it's important to note that while disk encryption provides robust data protection, it typically lacks authentication mechanisms, leaving a potential gap in overall security.

2.1.3 Authentication in Internet of Things (IoT)

While the solution presented in the previous section lacked authentication mechanisms, this section will explore various authentication solutions in the context of the Internet of Things (IoT).

The IoT is a network of interconnected devices equipped with sensors and various technologies

that enable them to collect and exchange data with each other. IoT devices often face resource constraints, particularly in terms of CPU, memory, and power consumption. Despite these limitations, they play a critical role in facilitating various functions such as data collection and analysis.

However, the widespread adoption of IoT devices has also brought with it a range of security challenges that are exacerbated by their inherent constraints. These devices are subject to the same threats as traditional computing devices, amplified by their wireless aspect. Because IoT devices often handle sensitive data, such as health information in the case of wearable devices like smartwatches, ensuring the security of their communications is critical.

In their article [4], B. Liao et al. propose a systematic review of the literature focusing on the use of mobile computing to enable secure communications with IoT devices. Mobile computing can facilitate communication between a server (e.g., a cloud server) and IoT devices or process data gathered from sensors. Furthermore, it can provide authentication for IoT devices through mechanisms such as QR codes for device authentication and possibly two-factor authentication (2FA), which includes biometric authentication and passwords, to identify the user of the mobile application that is relaying or processing the exchanged data. However, a drawback of relying on mobile computing is the need for a mobile device (such as a smartphone) to facilitate secure communications, which limits device-to-device interactions. In addition, reliance on a mobile intermediary introduces the vulnerability of a single point of failure.

E. Schiller et al. provide a landscape of IoT security in [5]. They argue that authentication in the IoT should be hardware-based, using physical device attributes as a fingerprint for authentication.

They further distinguish between token-based authentication, such as OAuth2, where a device uses a token provided by a server, and non-token-based authentication, where a device authenticates itself by presenting credentials at each data exchange. They also distinguish between one-way authentication, where only one device authenticates itself to the other, two-way authentication, which involves mutual authentication, and three-way authentication, where a central authority authenticates both parties. All of these can be used for IoT authentication.

However, the IoT ecosystem is made up of a wide variety of devices from different manufacturers, running various protocols. A remaining challenge consists to achieve interoperability and standardize authentication mechanisms across this heterogeneous landscape.

2.2 Virtualization

Methods such as white-box, sluice, and disk encryption provide a solution to secure file transfers, but threats remain, such as a keylogger inserted between a keyboard and a computer. This section presents an alternative solution to peripheral security concerns: the use of virtualization technology.

In their paper [6], T. Shinagawa et al. introduced BitVisor, a minimalist hypervisor explicitly designed to enhance the security of I/O devices. BitVisor takes a unique approach to minimize overhead by limiting its functionalities to handling only one virtual machine (VM). In addition, to optimize its Trusted Computing Base, BitVisor implements a parapass-through architecture that allows direct pass-through access for certain peripherals, such as sound cards or graphics devices, while mediating access for others that require security enforcement. This architecture, shown in Figure 2.3, optimizes resource utilization and enhances security by selectively monitoring and encrypting IO operations. However, BitVisor doesn't provide built-in mechanisms for device authentication.

The figure 2.3 provides a practical demonstration of BitVisor's functionality. It shows how

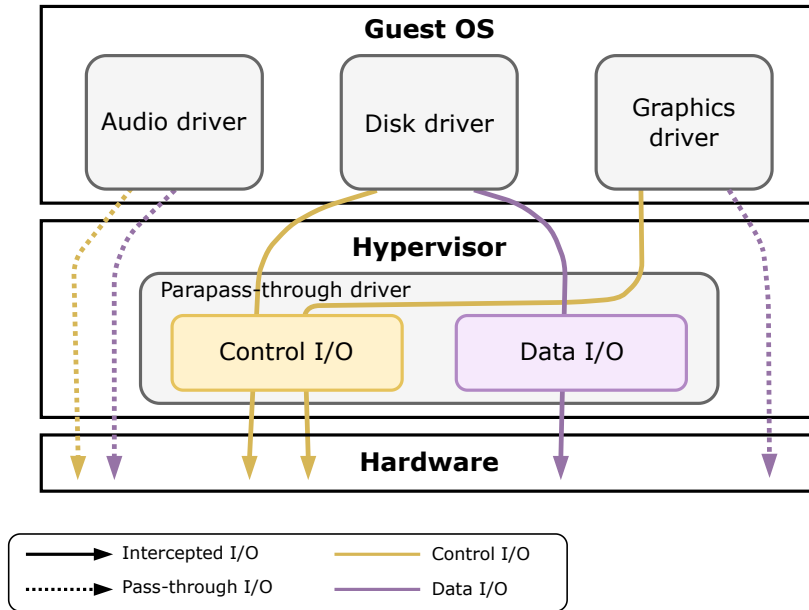


Figure 2.3: An example of the parapass-through architecture introduced in BitVisor. Based on [6].

BitVisor can intercept, control and encrypt the data exchanged between the guest operating system and the hardware devices, thereby enhancing security measures to prevent potential attacks against both the guest operating system and the underlying hardware. The control flow can also be verified and monitored to ensure that only authorized operations are allowed. In the use case shown in the figure, the communications between the guest OS and the hard disk are intercepted by BitVisor. We can imagine that the control flow check would verify the legitimacy of the locations, and the data flow check would implement data encryption in order to protect the disk.

In a separate study [7], M. Hirano et al. demonstrated the versatility of BitVisor by using it to detect ransomware using machine learning techniques. Leveraging BitVisor’s IO monitoring capabilities, the researchers set up an additional monitoring machine connected via a 10 GbE card to collect and classify data, focusing primarily on memory access patterns. This application is an example of how BitVisor can be used to strengthen device security by enabling sophisticated threat detection mechanisms.

Overall, BitVisor provides a compelling framework for enhancing device security through virtualization, demonstrating its potential for securing I/O operations and detecting malicious activity, although with certain limitations regarding device authentication.

3 Security Protocol and Data Model (SPDM)

3.1 Context

In order to address the limitations of traditional solutions and to provide a scalable solution that ensures secure access to peripherals, the Security Protocol and Data Model (SPDM) protocol was developed by a working group of the DMTF, and specified in [8].

The Distributed Management Task Force (DMTF) is a nonprofit association that develops new industry standards, which are recognized by both the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO). Its working teams include major hardware industry players such as Broadcom Inc., Cisco, Dell Technologies, Intel Corporation, and others.

SPDM is a protocol designed to enhance the security of communications over the wire and device attestation. The DMTF released the initial version of the protocol in 2019, and the latest version, SPDM version 1.3, was published in 2023.

The protocol has some similarities with the Internet Engineering Task Force's (IETF) TLS v1.3 [9], including mechanisms for cipher suite negotiation, certificate-based or pre-shared key authentication, confidentiality through key exchange, and key update protocols. However, formal analysis by Cremers et al. [10] suggests that SPDM's state machine complexity surpasses that of TLS. The authors of this formal analysis segment the protocol into four phases, as outlined in the next section.

3.2 The different phases of the protocol

3.2.1 Device initialization phase

The SPDM specification [8] mandates this phase to occur in a "secure and trusted environment", such as at a device manufacturer's facility. This phase, which is performed outside the protocol, consists of initializing the device with a unique device identifier, SPDM implementation, supported versions of the protocol, capabilities, and cryptographic algorithms. Vendors may also define "vendor defined functionalities", by implementing their own requests and responses.

To secure future communications, a device must receive at least some pre-shared symmetric keys or pre-shared public keys in accordance with another device. An example of using Pre-Shared Keys (PSK) could be between the screen (or keyboard) and the CPU of a laptop, since they often communicate with each other, and are soldered to the same board as soon as they are manufactured. In addition to the PSK, or instead of using the PSK, an asymmetric key pair, public key certificates in X.509 format (defined in [11]), and a root of trust to validate the certificates may also be used. If the specification does not set a limit on the number of shared keys, it limits the number of certificates to 8, with only the first forced to be set during initial setup (slot 0). The others slots can be filled or modified during a secure session with a certificate signing request (GET_CSR) and the response SET_CERTIFICATE.

3.2.2 Version-Capabilities-Algorithms (VCA) phase

This phase, akin to TLS, begins upon device connection. First, the Requester and the Responder exchange the versions of SPDML that they support, and decide which version of the protocol to run. The specification requires this version to be the highest supported by each party.

Once the parties have agreed on a version of the protocol, they must exchange their capabilities. A capability is a specified operation supported by the device, such as mutual authentication and encryption of communications. Both parties maintain the common subset of their capabilities.

To complete this phase, the parties exchange the cryptographic algorithms they support. Creemers et al. point out in [10] that the standard does not impose any mechanism for choosing among the supported algorithms, but that it seems obvious to take the most secure one.

During this phase, the Responder and the Requester maintain a transcript of their conversation. A transcript consists of a concatenation of the messages in a specific order. This transcript is completed and used in the next phases to verify the integrity of the exchange and even to derive the session and authentication keys.

3.2.3 Options phase

This phase enables device attestation (proof of firmware integrity and device identity) and one-way authentication of the Responder. As the name suggests, this phase is optional: the Requester can send a `KEY_EXCHANGE` request immediately after the VCA phase to directly jump to the session phase.

To authenticate the Responder, the Requester must first obtain a public key and corresponding certificate from the Responder. The Requester must then pick a random nonce and challenge the Responder by demanding a signature of the transcript. The response to this challenge asserts knowledge of the private key associated with the public key of the certificate, while the nonce prevents the replay attack. Instead, the Requester can ask the Responder to provide a digest of the certificate, if the certificate was cached from a previous session. The SPDML specification [8] recommends to perform this unilateral authentication before device attestation (because the transcripts are slightly different after attestation).

To attest the device, the Responder must send measurements to the Requester. A measurement is a cryptographic fingerprint of a piece of firmware (e.g., a hash of the firmware code). This can be used to verify the integrity of the device's firmware.

3.2.4 Session phase

The session phase begins with a handshake to allow encrypted and/or authenticated communication. By default, the key exchange provides only one-way authentication of the Responder (obtained in the option phase), which can set the *MutAuthRequested* flag to achieve two-way (mutual) authentication.

The Requester and Responder can establish the session key by using a Diffie-Hellman-based method or a pre-shared key. From this session key, the parties then derive their own encryption key and the (different) decryption key. The Requester's encryption key is the Responder's decryption key, and vice versa. Another type of key is derived, called the finished-key, which is used to authenticate the transcript. There are also two finished-keys between the Responder and the Requester.

In Diffie-Hellman-based schemes, authentication is accomplished either by requesting the other party's certificate, by using a cached certificate from a previous authentication, or by using a

pre-shared public key. In the case where the session key is derived using a pre-shared symmetric key, the authentication is implicit since they know the shared key.

Once the handshake is established, the Requester and Responder can exchange application data, possibly encrypted and/or authenticated using an Authenticated Encryption with Associated Data (AEAD) session. Periodically (the specification does not specify a frequency, only that it should be done regularly), they should perform a key update. For this purpose, the main secret is derived from the previous one using a key derivation function (HKDF), and then the new encryption key is derived from the new main secret using a HMAC function. These secrets can be updated for both the Responder and the Requester, or only for the sender of the KEY_UPDATE request.

All secrets are removed from memory when the session ends (request END_SESSION).

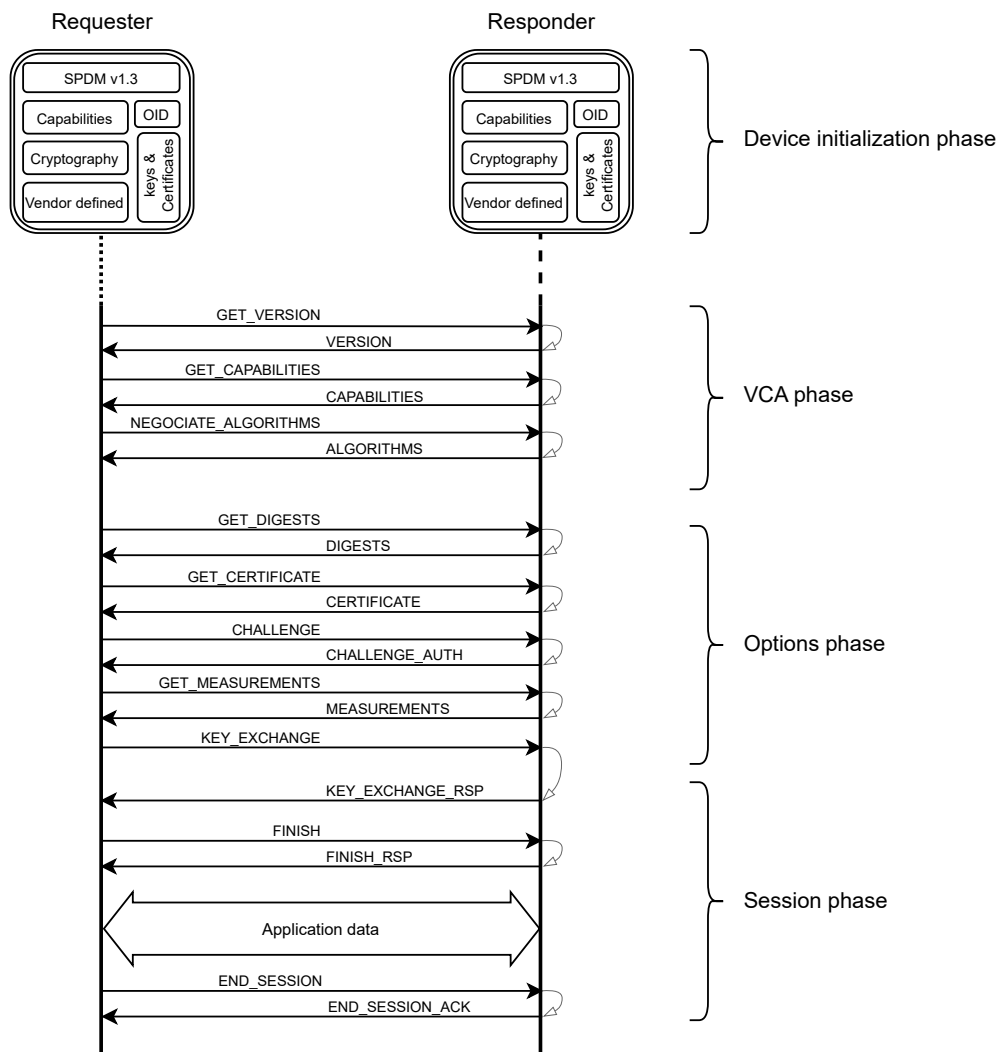


Figure 3.1: Messaging flow of SPDML

To sum up, the figure 3.1 illustrates the four phases of SPDML, with some associated messages. It's particularly enlightening to delve into the requests exchanged during the options phase. As previously outlined, this phase is optional and ends with a KEY_EXCHANGE request. Moreover, in this example, this phase starts with the requester retrieving the digests of any certificates stored during a previous session. Since the GET_CERTIFICATE request was issued after the fact, we can notice that no certificate was previously stored. Finally, the Responder is challenged on its knowledge of the key associated with the certificate ; the Responder is now au-

thenticated. Before completing this phase, measurements are requested to ensure the integrity of the Responder's system.

No mutual authentication is requested during the session phase, so the `FINISH` request is issued immediately. In the presented example, no key updates occur, and the session concludes after the exchange of application data encrypted using negotiated algorithms and exchanged keys.

3.3 Security features of SPDML

As we saw in the previous section, SPDML ensures secure communications over the wire by providing confidentiality through the encryption of application data and integrity through the transcripts and Message Authentication Codes (MACs) sent with the communications. The key update mechanism using a key derivation function strengthens the confidentiality of communications by reducing the risk of key leakage when this mechanism is performed regularly. In addition, the key derivation function provides forward secrecy by creating a chain of keys that prevents an attacker from computing the previously used keys from a leaked key because this function is a hash function.

Furthermore, SPDML provides device attestation and authentication through measurements and certificates. All of this allows SPDML to mitigate the different threats presented in the chapter 1.

Cremers et al. [10] used the formal prover Tamarin to analyze and prove the main security properties of SPDML, such as device attestation, certificates, pre-shared asymmetric and symmetric keys. However, they found some potential pitfalls, which are discussed in the next section.

3.4 Limitations

SPDML has several limitations, primarily due to its relative newness in the realm of security protocols. Despite its promising potential, there is currently a limited amount of research detailing its advantages and drawbacks. In addition, widespread implementation remains rare, with notable exceptions such as the NVIDIA Connect X7 Infinity Band network card [12]. However, since 2023, SPDML has been incorporated into the Mobile Industry Processor Interface (MIPI) Security Framework [13], which is designed to enhance security measures in various automotive applications, particularly in the context of *MIPI CSI-2*-based image sensors used in Advanced Driver Assistance Systems (ADAS) and Autonomous Driving Systems (ADS). This adoption underscores SPDML's adaptability and effectiveness in addressing critical security concerns, as exemplified by its ability to mitigate the man-in-the-middle attack scenario described in section 1.2.

This makes it difficult to estimate the performance of such a protocol in real-world applications, or the target audience for such a protocol. On the latter point, we can imagine that this protocol is aimed more at high-end products with sufficient computing power to manage cryptographic operations quickly. Also, the only SPDML implementations currently available are aimed at this target (Connect X7).

Furthermore, the protocol specification is not very clear on certain points, notably the storage of encryption keys. Is an enclave such as a Trusted Platform Module (TPM) required? Obviously, if the keys used by the protocol were known (for example, because they were stored in an insecure manner), the protocol would no longer provide any security guarantees and would be useless.

3.4.1 Potential design pitfalls

First, the protocol allows counters to be used instead of nonce at some stages. However, if the device is reset, the counter can be reused, which could lead to a replay attack. To address this issue, the SPDm v1.3 specification [8] specifies that "the counter shall not be reset for the lifetime of the device".

Second, because the Responder nonce is optional in PSK mode (justified in [8] by the fact that "the Responder can be a constrained device that cannot generate a nonce"), the anti-replay protection may depend on the unicity of the two-byte session ID. Thus, due to the small range of the session ID, an attacker could replay the initial messages in a session using the same session ID, in order to establish a session using the same session keys. However, the specification requires the Responder to use a nonce when possible.

Another challenge associated with PSK usage is authentication. While the shared key knowledge authenticates the involved parties, a distinction arises between device and key authentication, particularly when multiple devices share the same key. To address this concern, the specification mandates the inclusion of a unique object identifier (OID) in certificates. This OID serves to differentiate devices, bolstering authentication integrity. However, the certificates are not necessarily used in PSK mode, since authentication is based on knowledge of the key.

Similar to TLS, this protocol is susceptible to downgrade attacks, wherein an adversary compels a party to employ a compromised algorithm to compromise cryptography. As the negotiated cryptographic algorithm is the strongest mutually supported by both parties, attackers may resort to inducing the use of outdated algorithms susceptible to exploitation. To mitigate this threat, parties must undergo updates to eliminate compromised algorithms. However, updating devices presents a challenge, as the Initialization phase necessitates execution within a trusted environment.

3.4.2 Benchmark of SPDm

In their study [14], R. C. A. Alves et al. showcase the application of SPDm v1.1 in securing communications between a device and a hard drive within an emulated environment. The aim of their paper was to provide a proof of concept of SPDm, and to present the modifications required on the host operating system and on the device firmware. The researchers employed the *libspdm*¹ library, an open-source implementation of SPDm proposed by the DMTF, to modify the device firmware and write a matching driver.

Additionally, they delve into the performance analysis of SPDm through a benchmarking study, elaborated in [15], employing a distinct simulation setup.

In the latter study, the authors emulate a random number generator (RNG) device using QEMU KVM as the SPDm Responder and meticulously measure the overhead incurred by SPDm for each message individually. Their findings reveal that the `CERTIFICATE` response, predominantly due to disk access, has the highest execution time, approaching 10 milliseconds. Similarly, the `KEY_EXCHANGE` operation is in the same order of magnitude. Conversely, the PSK exchange demonstrates faster completion, taking only 1 millisecond, significantly reducing the workload associated with establishing session keys. Overall, the integration of SPDm results in a 6.4-fold increase in time for acquiring a random number after establishing the connection (application data of figure 3.1). Given the inherently swift nature of RNG, cryptographic operations become proportionally more resource-intensive.

Furthermore, the authors utilize diverse tools such as *dd* and *hdparm* to assess the overall per-

¹*libspdm* is an open-source implementation of SPDm, proposed by the DMTF. <https://github.com/DMTF/libspdm>.

formance of SPDM within a hard drive use case. Their observations reveal a 68% decrease in write speed with *dd*, and a substantial 99.3% reduction in read speed with *hdparm*.

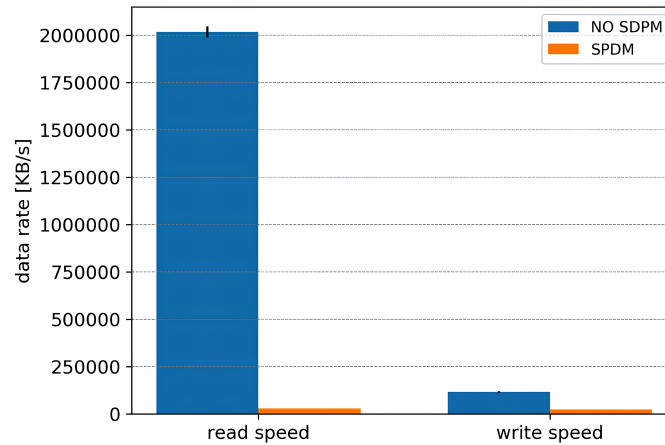


Figure 3.2: Read and write throughput of a SPDM-enabled hard drive, measured with *bonnie++*. Extracted from [15].

The figure 3.2, excerpted from [15], vividly illustrates the decline in performance attributable to SPDM. These results, garnered via *bonnie++*, closely align with those derived from *dd* and *hdparm*. Furthermore, as the encryption and decryption of each message constitute the bottleneck, both read and write operations drop down to approximately 25 MB/s, manifesting within the same order of magnitude.

Further analysis using the *fio* tool indicates that performance degradation becomes negligible when randomness is introduced into write and read address operations, effectively shifting the bottleneck to physical disk operations.

The reliance on simulation rather than real-world device-to-device communication can potentially skew the results. Unlike an emulated hard drive running on a computer’s CPU, a physical hard drive may lack comparable computational power, potentially impacting SPDM’s performance. Consequently, the benchmark results obtained in the simulated environment may not accurately reflect real-world scenarios, possibly overestimating SPDM’s efficacy.

Furthermore, the necessity for a high-performance processor to prevent significant computer slowdown situates SPDM’s target demographic primarily within premium device categories, such as the NVIDIA Connect X7.

Conclusion

In summary, the realm of device security encompasses a broad spectrum of challenges and considerations, each of which requires careful attention and proactive measures. Throughout this paper, we have undertaken a comprehensive exploration of device security, dissecting common attack vectors such as man-in-the-middle and spoofing attacks to illuminate the intricacies of today's threats.

We covered a range of traditional solutions and security architectures, from the pragmatic approaches of white box and sluice to the fundamental importance of disk encryption in protecting sensitive data. We also covered IoT device authentication as a key aspect of ensuring secure communications across interconnected networks. In addition, we explored the evolving device security landscape through the lens of virtualization-based solutions, highlighting their potential to enhance security measures in a variety of contexts.

At the heart of this report, we focused on SPDML, a nascent protocol that promises to revolutionize device security with its device attestation and secure communication capabilities. However, as with any emerging technology, questions remain regarding its widespread adoption and the computational resources required for its efficient operation. As such, future research efforts could profitably focus on evaluating SPDML's performance in real-world scenarios beyond the confines of virtualized environments. In addition, exploring the potential impact of quantum computing on the effectiveness of SPDML represents another compelling avenue for further investigation.

With cybersecurity an everyday issue, the future will tell if SPDML will be implemented in every device, just as most websites now use HTTPS instead of HTTP. For this to happen, the impact on the operating system corresponding with the device must be minimal, meaning the device must provide good performance.

Bibliography

- [1] “Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher,” standard, International Organization for Standardization, Geneva, CH, Mar. 2011. Edition: 2.
- [2] “Enabling Platform Integrity in a Common Way by Utilizing DMTF’s SPDm Standard,” Jan. 2024. DMTF Technical Note.
- [3] “Profil de fonctionnalités et de sécurité - Sas et station blanche (réseaux non classifiés),” July 2020. ANSSI.
- [4] B. Liao, Y. Ali, S. Nazir, L. He, and H. U. Khan, “Security Analysis of IoT Devices by Using Mobile Computing: A Systematic Literature Review,” *IEEE Access*, vol. 8, pp. 120331–120350, 2020.
- [5] E. Schiller, A. Aidoo, J. Fuhrer, J. Stahl, M. Ziörjen, and B. Stiller, “Landscape of IoT security,” *Computer Science Review*, vol. 44, p. 100467, 2022.
- [6] T. Shinagawa, H. Eiraku, K. Tanimoto, K. Omote, S. Hasegawa, T. Horie, M. Hirano, K. Kourai, Y. Oyama, E. Kawai, K. Kono, S. Chiba, Y. Shinjo, and K. Kato, “BitVisor: a thin hypervisor for enforcing i/o device security,” in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE '09*, (New York, NY, USA), pp. 121–130, Association for Computing Machinery, 2009. event-place: Washington, DC, USA.
- [7] M. Hirano and R. Kobayashi, “Machine Learning-based Ransomware Detection Using Low-level Memory Access Patterns Obtained From Live-forensic Hypervisor,” in *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*, pp. 323–330, 2022.
- [8] “DMTF. SPDm v1.3.0 specification (DSP0274),” June 2023.
- [9] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3,” Aug. 2018. Issue: 8446 Num Pages: 160 Series: Request for Comments Published: RFC 8446.
- [10] C. Cremers, A. Dax, and A. Naska, “Formal Analysis of SPDm: Security Protocol and Data Model version 1.2,” 2022. Published: Cryptology ePrint Archive, Paper 2022/1724.
- [11] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” May 2008. Issue: 5280 Num Pages: 151 Series: Request for Comments Published: RFC 5280.
- [12] “NVIDIA: Using RecordFlux and SPARK to Implement SPDm for Secure Computing,” *AdaCore Technical Paper*, Feb. 2023.
- [13] “MIPI Security Framework.” <https://www.mipi.org/specifications/mipi-security>. [Accessed 29-04-2024].
- [14] R. C. A. Alves, B. C. Albertini, and M. A. Simplicio, “Securing hard drives with the Security Protocol and Data Model (SPDM),” in *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 446–447, 2022.
- [15] R. C. A. Alves, B. C. Albertini, and M. A. S. J. au2, “Benchmarking the Security Protocol and Data Model (SPDM) for component authentication,” 2023. _eprint: 2307.06456.