HSS: An On-board Hardware Security Subsystem for Trustworthy Cryptographic Services in Satellites

Daniel Fortún Sánchez, Martín Bárez Alonso
GMV, Germany

Richard Mitev, Patrick Jauernig
SANCTUARY Systems GmbH, Germany

Abstract— **Fuelled by New Space economy, there is a critical demand for enhanced cybersecurity in satellites, a necessity increasingly recognised yet challenging to meet due to resource constraints, complexity, and maintenance costs. Robust security implementations across each satellite subsystem traditionally require specialised cryptographic expertise, complicating key management and increasing overhead.**

To overcome these issues, this work introduces a novel satellite component, the Hardware Security Subsystem (HSS), designed to centralise and simplify advanced security onboard satellites. The HSS combines Commercial Off-The-Shelf (COTS) components with a hardware-based Trusted Platform Module (TPM) as a secure cryptographic anchor. Distinctively, the HSS offers comprehensive cryptographic and security functionalities over a secure CAN interface, accessed via a high-level API greatly simplifying subsystem integration. We implemented a prototype using the Xilinx Zynq UltraScale+ MPSoC using an Infineon SLB9672 TPM.

Index Terms*— *Small satellites, Satellites, Space technology, Embedded systems, Computer security, Security, TPM, COTS, Space applications, Space security, Cryptography.

I. Introduction

The space ecosystem is rapidly evolving, driven by the New Space paradigm, which emphasizes the use of commercial off-the-shelf (COTS) components and more powerful, reconfigurable payloads. This shift gives more opportunities for missions to adapt and enhance their capabilities in orbit.

Satellites are increasingly vulnerable to a wide range of cyberattacks, as highlighted by recent studies and real-world incidents [1]. These attacks can target various subsystems, including the satellite bus, payload, and communication links, leading to severe consequences such as data manipulation, service disruption, and even permanent loss of control. The reliance on COTS components, which are not originally designed for the stringent security requirements of space, exacerbates these vulnerabilities. Additionally, the interconnected nature of satellite systems [2], including ground stations and communication channels, further expands the attack surface, making satellites susceptible to both external and internal threats.

The criticality of these attacks cannot be overstated. For instance, attacks on the satellite bus can grant adversaries full control over the satellite's core operations, while payload attacks can disrupt mission-critical functions such as Earth observation or communication services. Network-based attacks, such as signal spoofing or jamming, can lead to denial of service or unauthorized command execution. These threats are not merely theoretical; they have been demonstrated in real-world scenarios, where even attackers with limited resources have successfully compromised satellite systems. The potential impact of such attacks extends beyond individual satellites, affecting global communication, navigation, and surveillance systems that rely on these space assets [3]. To address these challenges, this work presents the implementation of a dedicated Hardware Security Subsystem (HSS) that leverages a Trusted Platform Module (TPM) to provide centralized security services to other satellite subsystems over CAN bus. Through this interface, subsystems can access cryptographic functions and secure storage capabilities offered by the TPM, which serves as a hardware root of trust. This architecture enables secure key management, attestation, and data protection.

Our HSS can provide a secure environment for cryptographic operations, ensuring that sensitive data, such as encryption keys and authentication credentials, are protected from unauthorized access. By integrating an HSS into the satellite's architecture, critical operations like command and control, data transmission, and software updates can be safeguarded against tampering and exploitation. Furthermore, key management systems can ensure that cryptographic keys are securely generated, stored, and rotated, reducing the risk of key compromise and enhancing the overall security posture of the satellite.

II. Contributions

Our HSS is a novel secure subsystem for spacecraft that is the first to provide a comprehensive suite of security services to other subsystems via the satellite bus:

- Secure communication between subsystems: TLS encryption of the CAN bus of the satellite, ensuring inter-subsystem confidential, authenticated, and tamper-proof communication, critical for transmitting sensitive data such as telemetry, command sequences, and payload information, protecting the satellite from eavesdropping, message injection, and replay attacks.
- Cryptographic key management: secure generation, storage and management of secret keys will be used for encryption, digital signatures and authentication, essential for maintaining the integrity and confidentiality of the satellite's data.
- Secure updates: check integrity of firmware during boot process to only execute trustworthy software.
- Remote attestation: verification of the integrity of a satellite’s software and hardware configuration, crucial for detecting potential compromises during the satellite's operational lifecycle.

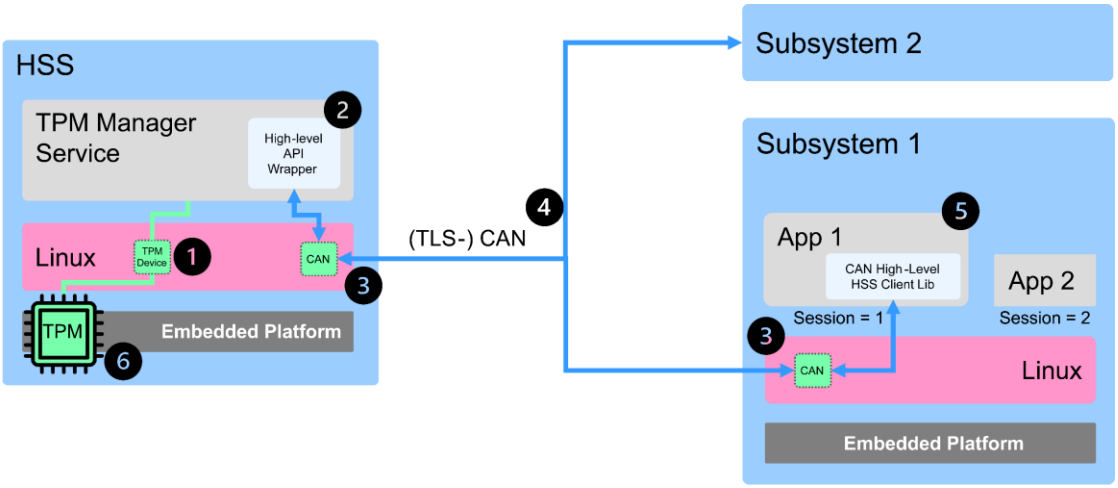

Figure 1 High-level overview of HSS architecture

- Payload data protection: Payloads, such as scientific instruments, often generate sensitive data that must be securely stored and transmitted. The HSS provides secure storage for payload data and ensures that it is encrypted before transmission over the CAN bus or downlink to ground stations. This ensures that payload data remains confidential and protected from unauthorized access throughout its lifecycle.

III. System Model

This section outlines the system model, introducing the basic satellite system model used throughout the paper. This system model addresses the security challenges specifically encountered in SmallSats and CubeSats, which commonly follow a classical satellite architecture. In this architecture, various critical subsystems—such as the Command and Data Handling System (CDHS), Electrical Power System (EPS), Attitude Determination and Control System (ADCS), and Communication Systems—are interconnected via a satellite communication bus, typically implemented using the Controller Area Network (CAN) protocol. Additionally, we assume the presence of a separate payload bus, dedicated to mission-specific components, including Payload Data Handling Systems (PDHS) and payload communication modules responsible for Telemetry, Tracking, and Command (TT&C) functions. Reflecting common practice in modern satellite missions, we assume these systems employ robust cryptographic mechanisms such as the CCSDS-defined Space Data Link Security (SDLS) for both encryption and authentication, particularly to secure ground communication.

A. Adversary Model

Our adversary model comprises a powerful attacker capable of arbitrary code execution on any single subsystem of the satellite, with the notable exception of the proposed Hardware Security Subsystem (HSS). Following the widely adopted Dolev-Yao threat model [4], the attacker can intercept, read, manipulate, inject, or suppress messages on the broadcast communication bus linking the satellite's subsystems. Specifically, the attacker has full visibility into all bus communication, can impersonate legitimate subsystems, or indefinitely delay the transmission of messages. However, the attacker cannot break cryptographic primitives. Furthermore, this work does not address attacks explicitly targeting the HSS itself, as defending such attacks is considered orthogonal to our scope. The HSS constitutes a classical embedded system interfaced via a CAN bus, and thus, protecting it against targeted intrusion is already covered extensively in existing embedded-system security literature. Finally, we exclude availability guarantees for the HSS, as CAN bus is prone to attacks [7].

IV. Architecture

The Hardware Security Subsystem (HSS) is a novel satellite subsystem that is, to the best of our knowledge, the first to provide advanced security and cryptographic services over the CAN bus. The HSS interfaces directly with a hardware-based TPM, leveraging its capabilities as the trust anchor for the entire satellite. The HSS exposes a high-level API over a secured CAN channel, enabling other satellite subsystems, each running a client application, to request cryptographic operations, key management functions, and higher-level services such as SDLS encryption, decryption, and remote attestation.

By centralizing security-critical tasks in the HSS, the design ensures that cryptographic keys remain protected within the hardware TPM, maintaining strict boundaries between sensitive material and client subsystems. Each client subsystem uses a library to interact with the HSS, requesting operations such as symmetric or asymmetric encryption, digital signing, or key generation. For advanced security use cases, the HSS supports remote attestation services not only for its own firmware and configuration, but also for compliant applications on client subsystems in case they integrate a Root of Trust for Measurement (ROTM) mechanism. This enables end-to-end authenticity and integrity checks of application code before its execution and is a central building block for upcoming supply chain security measures.

The high-level architecture of the HSS subsystem, as illustrated in Figure 1, depicts the interconnection between the HSS and the client subsystems, providing a structured representation of the system's functional components, comprising the following modules:

1) The TPM device and its software stack
2) The HSS API wrapper library
3) The CAN bus driver and wrapper library including an HSS API to CAN message serializer

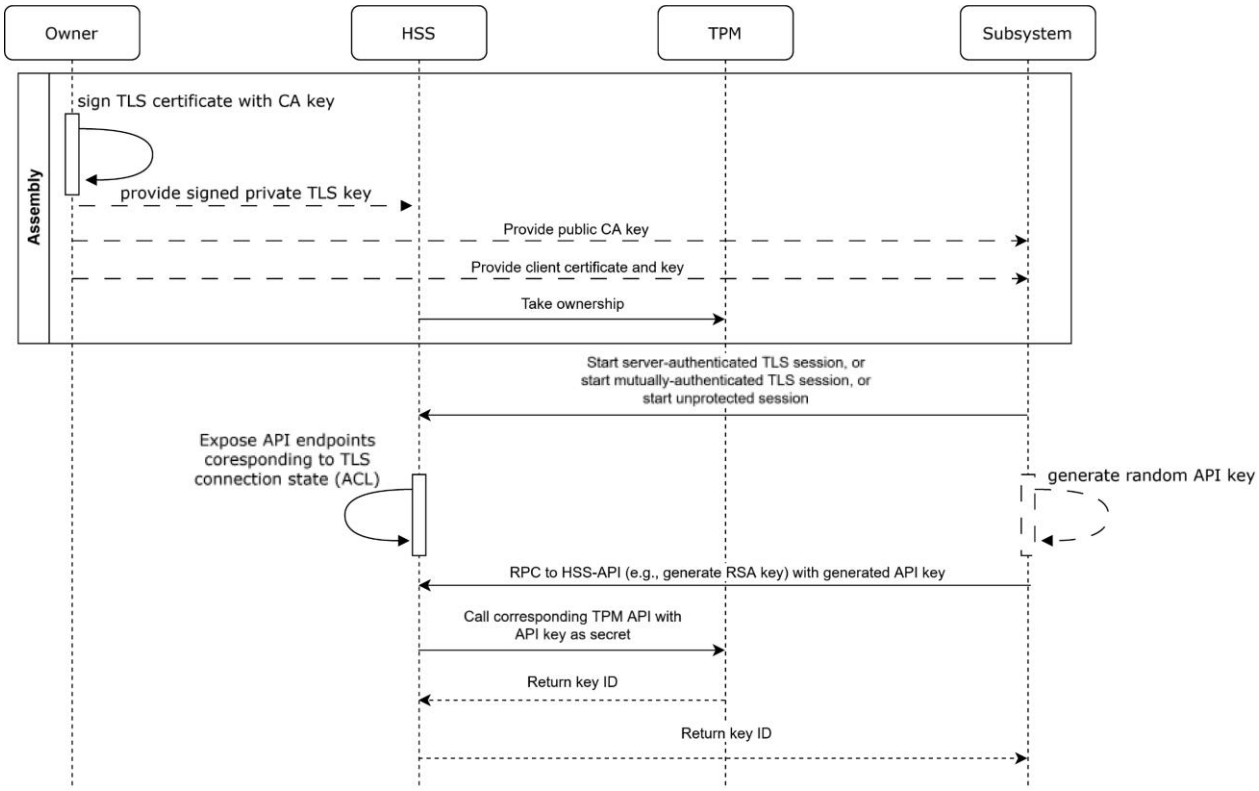

Figure 2 On-boarding of a subsystem to the HSS

4) The TLS secure channel over CAN bus library on both HSS and each subsystem

5) The subsystem client HSS API wrapper library including a (software) ROTM

6) The ROT of the embedded platform in combination with the TPM for Secure Boot

The HSS constitutes a critical component within the system architecture, primarily responsible for interfacing with the hardware TPM and facilitating communication with various subsystems. The design is organized into three principal components: the high-level API wrapper, security services, and TPM management. The high-level API wrapper establishes the HSS's interface by translating high-level function calls into lower-level service invocations that interact either directly with the TPM or with dedicated security service modules designed to implement more complex functionalities.

Security services extend the basic capabilities of the TPM by providing operations that surpass its inherent cryptographic primitives. These services are integral for tasks such as validating and encrypting or decrypting SDLS communication frames, as well as conducting remote attestation of external subsystems. By leveraging TPM-protected key material and utilizing cryptographic primitives offered by the TPM when feasible, the security service modules ensure that advanced security functionalities are executed with a high degree of hardware-rooted trust.

Finally, whenever possible, a TLS-secured CAN bus connection can be used by the client subsystems to interact with the services offered by the HSS. CAN bus is widely used on satellites, but it could be susceptible to attacks, for example message injection, eavesdropping or replay attacks.

B. HSS API

The HSS API is designed to provide a high-level interface for subsystems to access the TPM's security functionalities. The API is intended to be lightweight and user-friendly, abstracting the complexity of the underlying TPM commands. The API exposes only the necessary functionalities required by the use cases, while unnecessary or redundant TPM commands are hidden to reduce the attack surface.

The API includes functionality for:

- Space Data Link Security (SDLS) protocol for encryption/decryption of Telemetry, Tracking and Command (TT&C) packets.
- Key Management: Generation, storage, and retrieval of cryptographic keys.
- Cryptographic Functions: Symmetric and asymmetric encryption/decryption, creation & verification of digital signatures, hashing and HMAC operations.
- Attestation: Local and remote attestation.
- Random Number Generation: Generation of high-quality random numbers.
- Secure Storage: storage of data for subsystems, assuring that it can only be accessed by the owner.
- Session Management: Allows subsystems to interact with the HSS without needing to handle session creation, maintenance, or termination.

The design of the HSS API abstracts the TPM 2.0 interface, further refining and encapsulating TPM functionalities into a more restricted and accessible interface. This additional layer of abstraction is essential to ensure that access to security-critical operations is tightly controlled, while also simplifying the complexity of cryptographic processes for client subsystems. This approach not only

enhances usability but also significantly reduces the potential for misconfiguration or misuse, thereby strengthening the overall security of the system.

C. Initial HSS setup and subsystem onboarding

Figure 2 shows the initial HSS setup at assembly time including pre-sharing of secrets as well as the onboarding process of the subsystems. At assembly time, the owner determines the communication model between each subsystem and the HSS over the CAN bus. In one option, the subsystem connects without TLS security, meaning that no pre-shared keys or certificates are required. Another option introduces a server-authenticated TLS session, where the subsystem must possess the public CA key of the owner. A third alternative uses mutual authentication through TLS, in which case the subsystem requires both the public CA key of the owner and a signed private client certificate.

The satellite owner holds a CA private key, and the public portion is shared with the subsystems as needed. The HSS library uses an X.509 TLS certificate, which is signed by this CA private key. The HSS library takes ownership of the TPM by setting an authorization value that is later discarded, since the HSS is directly connected with low locality. Once this has been established, subsystems gain access to the HSS API over the CAN bus, with the option of (server or mutually authenticated) TLS-based communication.

Depending on the type of connection, the HSS enforces its Access Control List (ACL) to determine which calls are authorised. Some subsystems may issue unprotected requests without any authorization or authentication, such as a call to the *getrandom* function. Other subsystems connect through TLS and identify themselves by presenting their client certificate, thereby enabling ACL-managed requests (e.g., quotas per subsystem). Subsystems that require further authentication for HSS API calls register an API key in the form of a secret password and forward it inside authenticated requests. TLS protection safeguards this key against interception, since the HSS is always verified through the CA public key (i.e., no unprotected API calls that include an API key are allowed). The HSS API then provides the API key to the TPM, treating it as a password whenever a TPM API call requires. Therefore, when keys are saved or generated inside the TPM the API Key acts as a password for locking the key, or when a key is accessed the API key is used as a password for accessing it. Therefore, a subsystem is authenticated (but not necessarily identified) by passing the correct API key.

V. Implementation

For our implementation, we selected an Infineon SLB 9672, as it provides the most comprehensive list of cryptographic algorithms. As a development board, we selected an ALINX Xilinx Zynq UltraScale+, which is commonly used in space activities. The supported operating system is PetaLinux, for which we enabled secure boot, but also authenticated boot which fills the TPM's PCR and is needed for remote attestation. The high-level wrapper in the HSS server is responsible for implementing the previously defined HSS API functions by internally invoking the corresponding low-level wrapper functions. For these, wolfTPM [6] was used as it already provides abstraction. This design abstracts TPM-specific details from the external interface, presenting a clean and unified API for higher-layer components. To secure the CAN bus, we use TLS – depending on the subsystem this can be either server-side only authentication or a mutual authentication ("*mTLS*"). We implemented TLS for the CAN bus using wolfSSL [5] on top of CAN ISO-TP as a transport layer.

For the actual RPC interaction, we used Cap'n Proto, which is similar to the commonly used protobuf, but much more efficient. Cap'n Proto provides a schema-defined, type-safe mechanism for remote procedure calls with integrated access control. At its core, Cap'n Proto relies on an interface definition language (IDL) schema to formally specify the structure of RPC interfaces and associated message formats. Access control for these functions is tightly integrated into the RPC layer and is enforced based on the security context established during the TLS session. Depending on the level of authentication achieved, clients are granted access to different subsets of the available API functions. In the absence of TLS, access is strictly limited to a minimal set of operations, such as retrieving random values. When the client is authenticated through a server-side-only TLS handshake, access is slightly expanded but remains restricted. In the case of mutual

Figure 3 Infineon TPM attached to Xilinx Zynq UltraScale+

authentication, where both client and server present valid certificates, the system allows for fine-grained, certificate-based access control. The specific list of permitted functions under mutual TLS can be tailored to mission requirements, enabling differentiated permissions per client identity. This structure ensures that sensitive operations are only available to authorised entities, in accordance with both cryptographic trust and policy configuration. Finally, for each RPC function exposed by the HSS, there is a client-side stub that marshals the function arguments and checks for return values and errors in the RPC return message. The client library includes a regular CPP header that can be used as an interface to the complete high-level API of the HSS, also abstracting all CAN bus communication or TLS details.

The implementation of the SDLS protocol within the HSS is based on NASA's CryptoLib, a C-based, software-only implementation conforming to the CCSDS SDLS standard. The software also includes a work-in-progress extension for the SDLS Extended Procedures (SDLS-EP), intended to enhance the security of communication links between spacecraft flight software and terrestrial ground stations. For CryptoLib, we added respective backends for key (un)loading and cryptographic operations. Secure boot ensures firmware

and software integrity of the HSS by combining the SoC's native capabilities with a TPM, which cryptographically records boot measurements. The Boot ROM, FSBL, U-Boot, and all subsequent components are measured before execution, with each stage extending its hash into the TPM. Optional decryption and authentication steps provide configurable security for different mission requirements.

VI. Security Considerations

Given the capabilities of the adversary, defined in Section III.A, attacks on the satellite bus threaten full takeover of core operations, while payload-level compromises can disrupt Earth observation or communications services.

The model explicitly precludes cryptographic primitive breakage, preserving the theoretical integrity of algorithms; nonetheless, real-world exploitation of implementations, side-channels or flawed key management can still undermine security. Attacks directed at the HSS itself fall outside the present scope, as existing protections for the TPM and its host are assumed to be orthogonally addressed.

Integration of an HSS centred on a TPM provides a hardware root of trust that fundamentally alters the attacker's cost-benefit calculus. All cryptographic operations, encryption, digital signing, hashing and secure key storage, are confined within the TPM boundary, accessible only via the HSS API over a TLS-secured CAN channel. The TLS-based security of the communication channel enforces message confidentiality and authenticity, preventing eavesdropping, injection and replay, effectively mitigates Dolev-Yao attacks (note that we exclude availability of the HSM due well-known attacks on CAN bus). The API key further restricts the accessible key hierarchy and functions that can be used such that a potentially compromised subsystem cannot use a key hierarchy of another subsystem.

The HSS itself enacts secure boot and firmware-integrity checks at startup, ensuring that only authorised code populates the TPM's Platform Configuration Registers. Finally, physical tampering of the TPM or side-channel analysis could expose sensitive information, yet are not commonly encountered on satellites in space, and thus, have been excluded in the adversary model.

VII. Related Work

In this section, we discuss two highly related approaches that make a TPM available as a shared resource.

xTSeH [8] propagates TPM assurances from a single, TPM-equipped smart-embedded device to non-TPM peers by tunnelling TPM commands through an encrypted channel managed by a shadow TPM kernel module. Its Trusted-Booting, Remote-Verification and Node-Authentication protocols achieve measured-boot and attestation on Raspberry Pi hardware. In contrast to HSS, xTSeH does not provide a comprehensive set of security services to the clients, omitting, e.g., encrypting or signing data. Further, HSS introduces a general concept of authentication including access control to key material and TPM functions.

Khan et al. [9] anchor TLS sessions for grid-connected power converters in an TPM hosted on an ARM gateway, capturing tamper-proof telemetry with minimal latency overhead. Trust, however, remains centralised in the gateway, and the design does not expose key-management, sealed-storage or attestation services to other controllers on the field bus.

Both papers show that off-loading cryptographic primitives to commodity TPMs strengthens resource-constrained devices, but both do not expose a broader set of security services on a shared satellite bus.

VIII. Conclusions and Roadmap

This paper introduced a novel Hardware Security Subsystem (HSS), a dedicated on-board component designed to significantly enhance the cybersecurity of satellites, particularly addressing the growing threats associated with the New Space paradigm and the reliance on Commercial Off-The-Shelf (COTS) components. The presented HSS employs an Infineon SLB9672 Trusted Platform Module (TPM) integrated into a Xilinx Zynq UltraScale+ platform, providing comprehensive cryptographic and security services to satellite subsystems via a secure, TLS-protected CAN bus. The centralised architecture of the HSS simplifies the traditionally complex integration of cryptographic functions, reducing subsystem overhead and substantially mitigating the risk of data breaches, unauthorised access, and command injection attacks. Through secure boot procedures leveraging cryptographic measurements recorded in the TPM, the integrity and authenticity of firmware and software components are ensured from initial power-on. Future work aims to enhance the presented solution along several key dimensions. One direction involves integrating advanced cryptographic algorithms, including post-quantum cryptography (PQC), implemented in the FPGA's programmable logic to augment the TPM's native functionality. Another focus is the development of a custom, lightweight security stack, designed to replace existing dependencies on WolfSSL and WolfTPM with a mission-specific implementation optimised for the constraints of spaceborne systems. To improve interoperability, support for additional spacecraft communication buses beyond CAN is planned, enabling broader applicability across heterogeneous mission architectures. Lastly, experimental evaluation of TPM behaviour under radiation will be conducted to assess its resilience and inform the design of a protective packaging that enhances radiation tolerance in space environments.

References

[1] James Pavur, Ivan Martinovic, Building a launchpad for satellite cyber-security research: lessons from 60 years of spaceflight, *Journal of Cybersecurity*, Volume 8, Issue 1, 2022, tyac008, https://doi.org/10.1093/cybsec/tyac008

[2] Falco, G. (2020). When Satellites Attack: Satellite-to-Satellite Cyber Attack, Defense and Resilience. *ASCEND 2020*, doi.org/10.2514/6.2020-4014.

[3] Charlotte Van Camp, Walter Peeters, A World without Satellite Data as a Result of a Global Cyber-Attack, Space Policy, Volume 59, 2022, 101458, ISSN 0265-9646, doi.org/10.1016/j.spacepol.2021.101458.

[4] D. Dolev and A. Yao, "On the security of public key protocols," IEEE Transactions on Information Theory, vol. 29, no. 2, pp. 198–208, 1983.

[5] wolfSSL, "wolfSSL," GitHub repository, GitHub. [Online]. Available: github.com/wolfSSL/wolfssl. Accessed: June 27, 2025.

[6] wolfSSL, "wolfTPM," GitHub repository. [Online]. Available: github.com/wolfSSL/wolfTPM. Accessed: June 27, 2025.

[7] Rajapaksha, Sampath, et al. "CAN-MIRGU: a comprehensive CAN bus attack dataset from moving vehicles for intrusion detection system evaluation." Network and Distributed System Security (NDSS), 2024.

[8] Lu, Di, et al. "xTSeH: A trusted platform module sharing scheme towards smart IoT-eHealth devices." IEEE Journal on Selected Areas in Communications 39.2 (2020): 370-383.

[9] Khan, Ammar, et al. "Integrating Trusted Platform Modules in Power Electronics." 2020 IEEE CyberPELS (CyberPELS). IEEE, 202