Received 27 April 2023, accepted 9 June 2023, date of publication 13 June 2023, date of current version 21 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3285630

RESEARCH ARTICLE

Light-Weight Security Protocol and Data Model for Chip-to-Chip Zero-Trust

ASHFAQ AHMED[1], **(Senior Member, IEEE), ABDULHADI SHOUFAN**[1,2], **(Member, IEEE), AND KAIS BELWAFI**[1], **(Member, IEEE)**

[1]Center for Cyber-Physical Systems (C2PS), Khalifa University, Abu Dhabi, United Arab Emirates
[2]Department of Electrical Engineering and Computer Science, Khalifa University, Abu Dhabi, United Arab Emirates

Corresponding author: Ashfaq Ahmed (ashfaq.ahmed@ku.ac.ae)

This work was supported by the Khalifa University of Science and Technology from the External Fund under Grant 8434000388.

ABSTRACT The semiconductor supply chain is vulnerable to multiple security attacks, such as hardware Trojan injection, intellectual property theft, and overproduction. The notion of zero-trust (ZT) – never trust, always verify – offers a promising opportunity for chip security by authenticating integrated circuits (ICs) when they are connected to critical computing systems. Before exchanging any data, the system establishes trust with the chip using industry security protocols. In this paper, we propose using the secure protocol and data model (SPDM) to establish chip-to-chip (C2C)-ZT communications. Furthermore, we present formal models for this solution and verify these models using state-of-the-art formal verification tools. The results show that the SPDM meets the requirements of the ZT architecture and can be used as a foundation for secure C2C interconnection.

INDEX TERMS Secure protocol and data model (SPDM), automatic verification of internet security protocols and applications (AVISPA), secure protocol animator (SPAN), formal verification (FV).

I. INTRODUCTION

Over the last few years, ZT has become the preeminent concept, drawing the attention of the cyber security community [1]. Many corporate giants have already deployed ZT networks, notably Google's yondCorp [2], [3] and Microsoft's self-ZT security model [4], [5]. Due to the significant interest of the research community in ZT, the National Institute of Standards and Technology (NIST) has recently drafted the terms, definitions, and network infrastructure components for ZT. It is anticipated that it will be an integral part of future systems to protect enterprises and critical infrastructures against cyber attacks. The ZT principle has been widely adopted in a wide range of applications, including but not limited to self-driving networks [6], cloud computing [7], device-to-device communications [8], medical and healthcare sector [9], [10], and internet of things (IoT) [11].

The ZT principle promotes the notion of never trusting and always verifying. Recently, the hardware community has shown keen interest in ZT to protect chips and embedded systems. Especially since the emergence of the IoT, the security of the enormous, tiny devices is vital, since they may serve as soft targets for cyber attackers [12]. Drones / unmanned aerial vehicles (UAVs) are another example of devices that can be flown away from control centers and then kidnapped, hacked, and operated by an unauthorized entity [13].

In this paper, we propose a C2C-ZT architecture that provides physical-level security for hardware systems by permitting communication between two chips only if they pass an authentication and attestation procedure. This solution aims to mitigate the security concerns associated with outsourcing chip production that may lead to vulnerable embedded systems [14], [15]. Physically unclonable function (PUF) is a relevant technology that provides a digital fingerprint for chips. To validate this fingerprint, the user must have access to additional information, namely the challenge-response table. So, authenticating a chip with PUF boils down to trusting this challenge-response table, which contradicts the ZT principle. SPDM, on the other hand, is based on trusting the certification authority, rather than any data generated in the chip supply chain. Therefore, tt the core of the proposed C2C-ZT architecture is the SPDM.

The associate editor coordinating the review of this manuscript and approving it for publication was Dominik Strzalka.

As SPDM becomes increasingly popular in the cyber security community, it is crucial to validate the protocol. Indeed, the prerequisite for attesting any security protocol is its formal verification (FV). If a security protocol successfully passes all security threat tests during the FV, it can be implemented in systems. There are numerous tools and techniques available for verifying security protocols. In this study, we use the widely used push-button verification tool, automatic verification of internet security protocols and applications (AVISPA)+secure protocol animator (SPAN), which is extensively utilized by the research community for FV [16], [17], [18], [19], [20]. The performance of AVISPA tool is evaluated in [21]. Multiple security protocols are utilized to test the tool, and it is determined that the tool is well-suited for validating the protocols.

A. LITERATURE REVIEW

The ZT security architecture has been extensively studied in the literature, and various proposals have been put forward to address different security threats. In this section, we systematically classify the current ZT security architecture based on different threats and survey them accordingly.

AUTHENTICATION AND CONFIDENTIALITY OF IoT DEVICES

To guarantee the authentication and confidentiality of IoT devices, blockchain-enabled information sharing under the ZT principle is proposed in [11]. A lightweight continuous device-to-device authentication (LCDA) protocol is proposed in [8] to protect inter-device communication on resource-constrained devices. The protocol specifies mutual and continuous authentication phases that include a dynamic secret key refreshing method including channel state information (CSI) modification.

INSIDER THREATS

A ZT-based framework for the IC design process to counter insider threats is proposed in [14]. Palmo et al. in [22] proposed a method to securely embed resource-limiting IoT devices within software-defined perimeter (SDP), a ZT model proposed by the cloud security alliance (CSA). The use of endogenous security concepts to improve the internal structure of the ZT security model is proposed in [23]. The suggested heterogeneous redundancy mechanism enhances the security of the ZT system against internal threats while maintaining universal applicability, ensuring reliable network services and effective security defense.

MALWARE AND TROJAN DETECTION

In [15], a novel ZT aligned method for detecting run-time Trojans in untrusted commercial off-the-shelf (COTS) processors is provided. A software-defined ZT architecture for sixth generation (6G) networks is proposed in [24], which has the potential to establish a flexible and scalable security framework. This architecture relies on adaptive collaboration among control domains to enable secure access control, and it

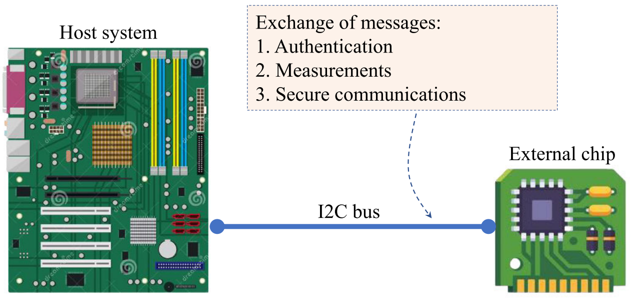

FIGURE 1. Use-case of chip-to-chip communications based on ZT principle.

provides effective protection against malicious access behaviors, including distributed denial of service (DDoS) attacks, malware propagation, and zero-day exploits.

HARDWARE SECURITY

In [25], a ProGun hardware/software co-design is proposed to mitigate the majority of security vulnerabilities caused by the lack of physical protection. It is built with multiple security layers to withstand cyber attacks. ProGun is a univeral serial bus (USB) dongle with a hardened and restricted operating system that can be used to protect any COTS computer from remote access. The dongle also includes logical interfaces and features for authentication and additional security. In [26], a satellite ground station front end processor (FEP) is examined to determine the degree of trust minimization in real-world scenarios. Using modern technologies and hardware-oriented methodologies, it is proven that trust minimization is attainable to a significant degree. It is anticipated that the code size will be reduced by a factor of 100, and that the level of protection against hardware backdoors will increase substantially.

COUNTERFEITING

Dutta et al. in [27] proposes and implements a ferroelectric field-effect transistor (FeFET)-based run-time reconfigurable camouflage logic technique. The solution concurrently conceals the design intellectual property (IP) from ZT foundry and an untrusted testing facility, thus eliminating the counterfeiting threats posed by reverse engineering.

Overall, these works address different security threats in the ZT security architecture and provide various solutions.

B. MOTIVATION AND CONTRIBUTIONS

The review of related work shows that substantial efforts have been made to protect devices, systems, and processes. To the best of our knowledge, however, there is no published literature on physical-layer authentication and secure communication for a C2C-ZT architecture. This physical-layer authentication of chips is highly needed to protect the system from exchanging information with counterfeit chips.

Under such a C2C-ZT architecture, no device is granted implicit access to the system, but the least possible privileges

TABLE 1. Description of the lightweight SPDM protocol in a C2C-ZT scenario.

Step	Host Chip	External Chip
1	Sends *GET_VERSION* message, which contains a list of all the versions of SPDM that are supported by the host	Receives the *GET_VERSION* message, and responds with a *VERSION* message that includes the latest and commonly supported SPDM version based on the list of supported versions from both the host and guest
2	Sends the *GET_CAPACITIES* message, which contains the SPDM version, the authentication, encryption, and key exchange algorithms, as well as the maximum message size	Receives the message *GET_CAPACITIES* and responds with *CAPACITIES* which is a message similar to the one received but includes the capabilities supported by the guest chip
3	Sends a *NEGOTIATE_ALGORITHMS* message, which includes a list of supported algorithms and a preferred order of priority	Selects the highest-priority algorithms that are supported by both the chips and sends a *ALGORITHMS* message to confirm the selected algorithms to be used for the rest of process.
4	Sends *GET_DIGESTS* message, which specifies the type of digest and the number of digests to be returned by the guest	Responds with *DIGESTS* message, which contains the hash of digital certificate, $Hash\{(KC1)_K\}$, where $KC1$ is the public key of the host which is encrypted with the private key K of the certification authority (CA)
5	Receives the $Hash\{(KC1)_K\}$ $if\{Hash\{(KC1)_K\}$ is not available in the memory$\}$ $\Longrightarrow$ Send *GET_CERTIFICATE* message, store $Hash\{(KC1)_K$ $else$ $\Longrightarrow$ Move to next step	Receives *GET_CERTIFICATE* message, and sends back the digital certificate, represented as $\{KC1\}_K$
6	Receives $\{KC1\}_K$, if not available in the memory. Sends a challenge message containing a nonce N_a	Receives the nonce N_a, generates its nonce N_b, and sends back $\{N_b, \{N_a\}_{inv\{K\}}\}$ as *CHALLENGE_AUTH* message, where $inv\{KC1\}$ represents the private key of the guest
7	Sends *GET_MEASUREMENTS* message to get measurements from a guest's firmware or hardware. The message includes a nonce N_a	Sends back the message *MEASUREMENTS*, which includes the requested measurements, the received nonce N_a, and its encrypted nonce $\{N_b\}_{inv\{KC1\}}$
8	A secure communication channel is established by exchanging Diffie-Hellman (DHE) key-pairs	

are granted after a rigorous authentication and attestation procedure. Fig. 1 shows a C2C use case in which an external chip is plugged into the host system through an inter-integrated circuit (I2C) bus. The host system initiates an authentication process with the chip. The external chip is only permitted to exchange data after the successful completion of this process.

To enable the C2C-ZT architecture, we propose using the SPDM between the host system and every chip in the system. The design of robust security protocols is knowingly complex since it should take into consideration all possible hostile behaviors and activities of attackers who could interfere with the protocol to gain unauthorized access [28]. Formal verification is a recognized approach that helps in checking the correctness of security protocols. To our knowledge, the SPDM protocol has not yet been formally verified. Therefore, a comprehensive FV of this protocol is demonstrated in this paper. This is to provide assurance about the security of this protocol in the hardware community. The protocol is specified in high-level protocol specification language (HLPSL) and a complete FV is performed using the AVISPA tool. The protocol is evaluated in several attack scenarios to show its robustness against these attacks and readiness for the C2C-ZT. The main contribution of this paper is the formal verification of the SPDM protocol, as well as providing open-source models for the community to reuse based on their specific requirements, with an emphasis on lightweightness. While the SPDM protocol is recommended by Intel,[1] its integration into chips is difficult due to the large size of the compiled library and limited resources of the chips. Therefore, it is critical to develop a lightweight version of the protocol to enable practical use in resource-constrained devices. Formal models play an essential role in ensuring that the lightweight versions of the protocol are secure.

C. PAPER ORGANIZATION

The threat model, followed by an overview of the SPDM protocol, is provided in Section II. The FV of SPDM is provided in Section III. This section includes an overview of the AVISPA tool and the HLPSL description of the protocol. The SPDM security validation is discussed in Section IV. The challenges involved in the formal verification of the SPDM protocol and the C2C-ZT are described in Section V. Finally, the work is concluded in Section VI.

[1]https://www.intel.com/content/www/us/en/newsroom/opinion/zero-trust-approach-architecting-silicon.html#gs.vthrcx

TABLE 2. Potential attacks and their countermeasure addressed by SPDM protocol.

Attack category	Description	Countermeasures
Information disclosure	Data flow sniffing	Encrypting the data flow
	Weak authentication schemes	Custom authentication scheme
Repudiation	Potential data repudiatioin by requester processor	Auditing to record the source, time, and summary of received data
Spoofing	Spoofing the requester processor process	Authentication mechanism to identify the destination process
Tampering	Collision attacks	Reassemble data before filtering it, and handle the data overlapping
	Replay attacks	Implement anti-reply techniques (investigates sequence numbers before timers), and strong integrity

II. THREAT MODEL AND AN OVERVIEW of SPDM PROTOCOL

The Microsoft threat modeling tool is utilized to identify potential attacks. The potential attacks are addressed by the SPDM protocol. The list of potential attacks in the C2C-ZT, and their potential countermeasures are provided in Table 2.

Recently published by the distributed management task force (DMTF) organization [29], the SPDM protocol is envisioned as a potential solution that adheres to the ZT's guiding principles. It provides means to authenticate and measure hardware devices and enable secure communications between system components. The SPDM relies on asymmetric cryptography and public-key certificates for authentication. A post-quantum design is proposed in [30].

The broader device community has widely embraced this protocol. It has been adopted by several other standard groups, including peripheral component interconnect (PCI) [31] and trusted platform module (TPM) [32]. In addition, the PCI express (PCIe) integrity and data encryption key management (IDE_KM) protocol [33] for link encryption and other application protocols are built on top of the SPDM protocol. It standardizes the authentication of hardware components and enables the establishment of secure communication channels among them [34]. The protocol allows endpoints to discover and negotiate each other's security capabilities and to retrieve each other's measurements, which include the configurations of the hardware and the firmware.

A. SPDM MESSAGES

We chose a lightweight approach to implementing the complex SPDM protocol in this work because the chip cannot execute the entire protocol due to limited on-chip resources. As a result, we carefully selected a subset of SPDM functions that guarantee ZT.

An overview of the exchanged messages in SPDM protocol is depicted in Fig. 2, and is explained in detail in Table 1. A secure connection is established between two end-nodes, referred to as a requester and a responder, after the exchange of multiple messages. The requester initiates the protocol by sending a *GET_VERSION* message to the responder. Typically, this message is used to either initiate a new SPDM session or reset the current session. In response to this message, the responder sends the *VERSION* message, which contains the version of SPDM it supports. Both the requester and responder may support multiple SPDM versions, but the requester always selects the most recent common version. Once the requester sends the *GET_VERSION* message, all active sessions are terminated and the associated data is discarded. After receiving a successful *VERSION* message response, the requester sends *GET_CAPABILITIES*. This message is used to retrieve a number of the capabilities of the responder, i.e., whether the requester supports digests and certificates, challenge authentication, and mutual authentication, as well as several capabilities pertaining to key exchange during the secure session. In SPDM, end-nodes can switch roles at any time. Consequently, during the exchange of *GET_CAPABILITIES* messages, the requester shares its capabilities with the responder.

The requester sends the *NEGOTIATE_ALGORITHMS* message after obtaining the capabilities of the responder. The objective of this message is to negotiate cryptographic algorithms. The requester informs the responder of all encryption algorithms supported. The responder selects the largest common encryption algorithm from the received list and responds via *ALGORITHMS* in a message. All SPDM session parameters are determined following a successful *ALGORITHMS* message response. The requester then transmits the *GET_DIGESTS* message if the responder supports digests and certificates. The responder sends the cryptographic hash values of its certificate chains in response to this message. The requester caches the received hash values and compares them to the previously cached digests. Certificates are only requested if they cannot be located in the local storage buffer.

If the received digests do not match the digests stored in the requester's cache, the requester sends the *GET_CERTIFICATE* message. Therefore, the *GET_DIGESTS* message can improve performance by avoiding the transfer of certificates that are already present at the requester. The complete certificate chain includes the first certificate, which is either signed by the root certificate or is itself a root certificate. The preceding certificates sign the subsequent certificates. The public key of the responder is contained in the leaf certificate. When receiving certificate chains,

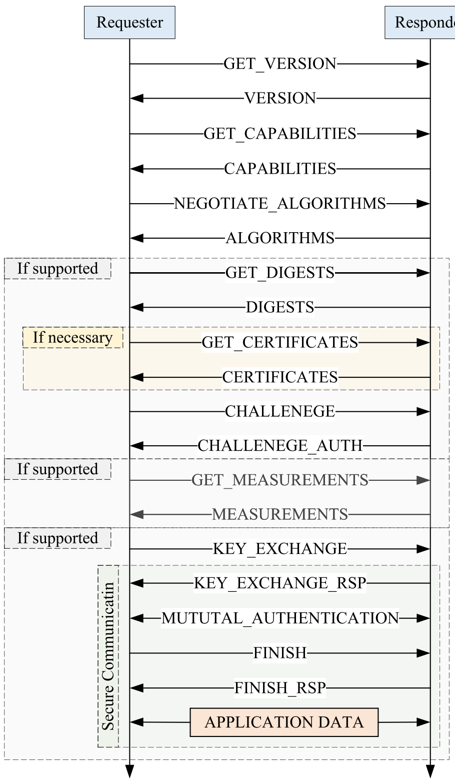

FIGURE 2. An overview of SPDM protocol.

the requester must at least save the public key of the leaf certificate. In addition, the requester must send multiple *GET_CERTIFICATE* requests to retrieve the chain. The requester then sends the *CHALLENGE* message to authenticate the responder. The requester sends a randomly generated *nonce* and the slot number of the certificate chain to be used for authentication. In response, the responder sends the message *CHALLENGE_AUTH*. The certificate chain hash is returned in this message. These hash values are used to validate that the certificate chain matches the requested one. Moreover, the responder generates signatures by encrypting the *nonce* received from the requester with its private key. The summary of the measurements' hash values is also included within the *CHALLENGE_AUTH* message. In contrast, the requester verifies the received signatures with the responder's public key; a successful verification proves the responder's authentication.

After authenticating the responder, the requester sends the *GET_MEASUREMENTS* message to determine the number of individual measurement blocks supported by the responder and to request either specific blocks or all available blocks. This message contains a *nonce* and the certificate chain index used for the authentication of measurements. In addition, the requester specifies whether all measurements, all available measurements, or specific measurements are required. Multiple request messages are used by the requester to obtain the measurements. In this situation, the requester does not request signatures for each individual message. Therefore, the responder simply sends back the measurements without the signature but creates a log for all received measurement request messages. On the other hand, the requester creates a log for all measurements received. The requestor will then request the signatures in the final measurements request message. Using its private key, the responder generates a signature on the measurement log if it includes an active flag for the measurement capability. The responder generates a *MEASUREMENTS* message that indicates any detected changes to the signed measurement log. It assists the requester in requesting the current measurements through another *GET_MEASUREMENTS* message. Furthermore, the number and length of measurement blocks are included in the response. Additionally, the message contains a *nonce* and the signatures of the measurements log.

This request message shall initiate a handshake between the requester and responder in order to authenticate the responder (or optionally both parties), negotiate cryptographic parameters (in addition to those negotiated in the previous *NEGOTIATE_ALGORITHMS / ALGORITHMS* exchange), and establish shared keying material. During this phase, the public keys generated by DHE [35] are exchanged between the two end-nodes. The responder generates a DHE secret using the private key of the responder's DHE key pair and the public key of the requester's DHE key pair as specified in the *KEY_EXCHANGE* request message. In a similar way, the requester generates a DHE secret using the private key of the DHE key pair of the requester and the public key of the DHE key pair of the responder, which are provided in the *KEY_EXCHANGE_RSP* response message. Both end-nodes generate identical DHE secrets, which are used to generate session secrets. The responder sends back a *KEY_EXCHANGE_RSP* message containing the DHE public key it has generated. The handshake is concluded when the requester and the responder send the *FINISH* and *FINISH_RSP* messages, respectively. The *FINISH* request and *FINISH_RSP* response messages serve to provide key confirmation, bind the identity of each party to the exchanged keys, and protect the entire handshake from active manipulation. The respondent returns the signature and hash-based message authentication code (HMAC) of the transcript. The application data is then exchanged between the end-nodes utilizing the authenticated encryption with associated data (AEAD) algorithm, which simultaneously provides data encryption and authentication.

III. FORMAL VERIFICATION OF SPDM PROTOCOL

This section discusses the AVISPA tool and how it is used to validate the SPDM protocol. FV, as opposed to simulations, demonstrates design correctness by using static analysis methods or model checkers for an equivalence check with a well-known referent or to demonstrate that the design properties and assertions are fulfilled. It eliminates the need for extensive simulations to get excellent coverage, and the verification results are independent of the quality of the test cases. Moreover, mathematical proofs are highly relevant in

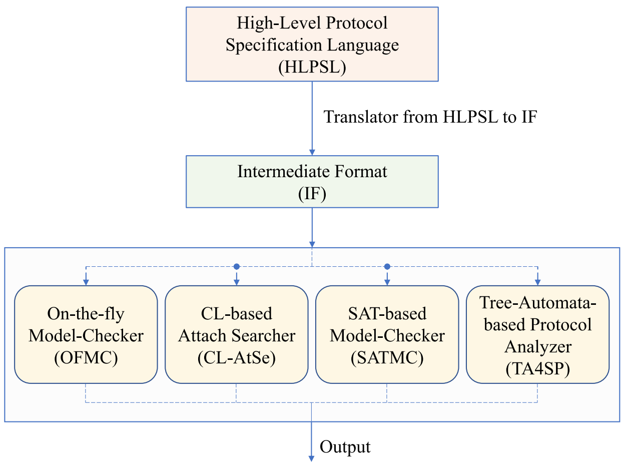
High-Level Protocol Specification Language (HLPSL)
Translator from HLPSL to IF
Intermediate Format (IF)
On-the-fly Model-Checker (OFMC)
CL-based Attach Searcher (CL-AtSe)
SAT-based Model-Checker (SATMC)
Tree-Automata-based Protocol Analyzer (TA4SP)
Output

FIGURE 3. AVISPA architecture.

the field of computer security, as they allow for the formal verification of security protocols to ensure their correctness and security. The AVISPA+SPAN toolkit embeds mathematical proofs as a core feature, allowing for the identification and mitigation of potential security vulnerabilities.

Given that SPDM is gaining popularity in the cyber security community, validation of the protocol is highly essential. Indeed, the FV of any security protocol is a prerequisite for the protocol's attestation. If the security protocol passes all tests for security threats during the FV, it can be implemented in the systems. Numerous tools and techniques exist for the FV of the security protocols. In this work, we are utilizing the AVISPA+SPAN tool, which is a push-button verification tool that is widely utilized by the research community for FV.

Given that SPDM offers a way to authenticate, compute the device's hardware identification, and allow secure communications between the devices. The FV of the SPDM protocol will increase hardware community trust in using the protocol for device authentication. The procedure typically involves the following steps:

1) Model the SPDM protocol using a HLPSL.
2) Implement the protocol in the tool and generate an initial model.
3) Specify the desired security properties to be verified, such as authentication or confidentiality.
4) Use the SPAN tool to simulate the protocol execution and generate an attack trace.
5) Analyze the attack trace to determine if the protocol is vulnerable to the specified security properties.
6) Modify the protocol or add security mechanisms if vulnerabilities are found.
7) Repeat the verification process until all desired security properties are satisfied.

A. AN OVERVIEW OF AVISPA TOOL

Figure 3 illustrates the structure of the AVISPA tool. As the initial step, the user specifies the security protocol in the HLPSL. Along with the protocol itself, the protocol's desired objectives are included in the protocol specification. The formal language HLPSL is expressive, modular, and based on roles. It describes complicated security aspects such as control-flow patterns, data structures, multiple cryptographic operators, and their corresponding algebraic properties. The HLPSL protocol specification, on the other hand, does not necessitate an initial protocol simplification because it has more robust features than weaker alternatives. The AVISPA tool converts a user-defined or standard security protocol described in HLPSL to a corresponding intermediate format (IF). The IF specifications are inputted into the tool's back-ends. The tool next explores the infinite-state transition system provided by the IF specifications to determine the state that signals attacks on the protocol's intended features. The current version of the AVISPA tool has four back-ends: on-the-fly model-checker (OFMC) [36], constraint-logic-based attack searcher (CL-AtSe) [37], SAT-based model-checker (SATMC) [38], and tree-automata-based protocol analyzer (TA4SP) [39]. The back-ends examine the protocols under the assumption of perfect cryptography and a network controlled by a Dolev-Yao intruder [40] for message exchanges. Even though the active intruder has control of the network, it is unable to breach the encryption. Furthermore, if the intruder has access to the corresponding keys, it can intercept and decrypt the messages. Following that, it forwards messages in the name of any other party that is generated based on its own knowledge. The analytical results are subsequently output by the backends using a standard and clearly defined output format. Based on the input goals, the outcome will indicate if the protocol is safe or unsafe. If a potential attack is detected, a trace is generated.

B. SPDM SPECIFICATION IN HLPSL

The protocol is comprised of two agents: the host system and the external chip. In our proposed method, the host system is the requester, and the external chip is the responder. We assume in our protocol that only the host system authenticates the external chip. This section contains HLPSL descriptions of the involved roles and sessions under consideration. In these HLPSL specifications, all receiving and transmitting channels adhere to the Dolev-Yao (DY) intruder model. In Fig. 4, an overview of the FV of SPDM in a C2C-ZT is displayed. The environment is created with two sessions. In addition, the goals for secrecy and authentication, as well as the knowledge of the intruder, are defined. In our model, the illegal chip is impersonated as an intruder.

1) HLPSL CODE FOR HOST

The HLPSL code for the role of the host is given below:

```
role host_chip (A, B: agent,
  Hash: hash_func,
  K: symmetric_key,
  SND, RCV: channel(dy))

played_by A
def=
```

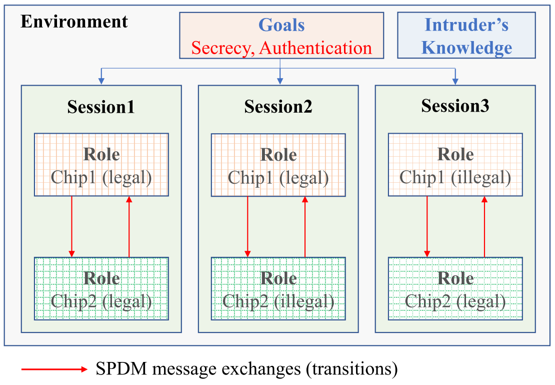

FIGURE 4. An overview of implemented FV on the AVISPA tool.

```

local
  State : nat,
  Get_Ver,Ver,Get_Cap,Cap : message,
  Negotiate_Algo,Algo : message,
  Get_Digests,Digests : message,
  Get_Cert : message,
  Get_Measurements,Measurements : message,
  Kc1 : public_key,
  KeyRing : (public_key) set,
  Na1,Na2,Na3,Na4,Na5 : text,
  Nb1,Nb2 : text

init
  State := 0

transition

% Sending GET_VERSION message
1.  State = 0    /\ RCV(start) =|>
    State' := 2  /\ SND(Get_Ver)

% Receiving VERSION message and Sending
  GET_CAPABILITIES message
2.  State = 2   /\ RCV(Ver') =|>
    State' := 4 /\ SND(Get_Cap)

% Receiving CAPABILITIES message and Sending
  NEGOTIATE_ALGORITHM message
3.  State = 4   /\ RCV(Cap') =|>
    State' := 6 /\ SND(Negotiate_Algo)

% Receiving ALGORITHMS message and sending
  GET_DIGESTS message, if certificates not
  available
4.  State = 6   /\ RCV(Algo') =|>
    State' := 8 /\ SND(Get_Digests)

% Receiving DIGESTS message and if certificate
  are not available
5.  State = 8 /\ RCV(Hash(Kc1'))
              /\ not(in(Hash(Kc1'),KeyRing)) =|>
    State' := 10 /\ SND(Get_Cert)
                 /\ KeyRing' := cons(Hash(Kc1'),
  KeyRing)
                 /\ request(A,B,auth_kc1,Kc1')

% Receiving DIGESTS message and if certificate
  are available
6.  State = 8    /\ RCV(Hash(Kc1'))
                 /\ in(Hash(Kc1'),KeyRing) =|>
    State' := 12 /\ request(A,B,auth_kc1,Kc1')


% Receiving CERTIFICATES and sending CHALLENGE (
  nonce)
7.  State = 10   /\ RCV({Kc1'}_K) =|>
    State' := 14 /\ Na1' := new()
                 /\ SND(Na1')
                 /\ request(A,B,auth_kc1,Kc1')
                 /\ secret(Na1',secrecy_Na,{A,B
  })

% Only sending CHALLENGE (nonce)
8.  State = 12 =|>
    State' := 14 /\ Na2' := new()
                 /\ SND(Na2')
                 /\ secret(Na2',secrecy_Na,{A,B
  })
                 /\ witness(A,B,auth_Na,Na2')

% Receiving CHALLENEGE and sending
  GET_MEASUREMENTS message
9.  State = 14   /\ RCV(Nb1'.{Na3'}_inv(Kc1))
  =|>
    State' := 16 /\ Na4' := new()
                 /\ SND(Na4')
                 /\ request(A,B,auth_Nb,Nb1')
                 /\ secret(Na4',secrecy_Na,{A,B
  })
% Receiving the MEASUREMENTS and sending the
  KEY_EXCHANGE message
10. State = 16   /\ RCV(Nb2'.{Na5'}_inv(Kc1))
  =|>
    State' := 18 /\ request(A,B,auth_Nb,Nb2')
                 /\ secret(Na5',secrecy_Na,{A,B
  })

end role
```

The role of the host chip initiates the protocol since it contains the statement *RCV(start)*, where *RCV* is the receiving channel of the role. It then sends the *Get_Ver* message on the *SND* channel to retrieve the SPDM version from the external chip. It receives the version message in the *Ver* local variable via the *RCV* channel. After this, the host system sends the *Get_Cap* message and updates the state. Likewise, after receiving the capabilities message in its local variable *Cap*, it modifies the state and transmits the *Negotiate_Algo* message over the *SND* channel. The host system updates its current state and requests the digests of the certificates using the *Get_Digests* message after obtaining the common algorithm in its local variable *Algo*. The message received contains the hash values of the available certificates of the external chip. The host system compares the hash certificate values received within its local buffer, *KeyRing*. If the hash values are present in the buffer, it means that the host does not need to request the certificates because they are already available. The next message in this case is to challenge the external chip. The host chip sends a nonce to the external chip. Alternatively, if the host chip is unable to locate the received certificate hash values in its local buffer, the host modifies its current state and simultaneously sends a *Get_Cert* message. Concurrently, the host chip stores the received certificate hash values in the local buffer. The host receives the public key of the chip in response to the *Get_Cert* message in its local variable *Kc1*. In practice, the chip sends the entire certificate chain, with the leaf certificate containing the public key. This public key is subsequently employed to create a secure channel. In this case, the exchange of public keys should be accomplished through the exchange of certificates.

At this stage, the host chip challenges the external chip through a nonce. In response to the challenge message, the host chip receives the nonce shared by the external chip as well as its transmitted nonce that has been encrypted using the external chip's private key. If the host decrypts the encrypted nonce with the external chip's public key, it signals that it is connecting with the legal external chip. At this point, the host updates its state and sends a message to obtain the external chip's measurements. This message includes a nonce for freshness. The host then requests the chip's firmware and hardware measurements. The nonce guarantees that the most recent and accurate measurements are received. In response to the message, the host receives the measurement as well as the nonce of the chip and its own nonce encrypted using the external chip's private key. For authentication purposes, the host decrypts its encrypted nonce using the public key of the external chip.

Finally, a secure communication link between the two devices is established. The SPDM protocol accomplishes the key exchange using a handshaking method that incorporates the ephemeral DHE key exchange. Both chips generate a pair of public and private keys during this phase. After that, they exchange their respective public keys. Finally, both endpoints construct secret keys using their private keys and the public keys they have received. In [41], the FV of the DHE algorithm is provided. Therefore, we limit our FV to identification and authentication in this work.

2) HLPSL CODE FOR EXTERNAL CHIP

The following is the HLPSL specification for the role of external chip:

```
role external_chip (
  A, B: agent,
  Hash: hash_func,
  K: symmetric_key,
  SND, RCV: channel(dy))
played_by B
def=

  local
    State : nat,
    Get_Ver,Ver,Get_Cap,Cap : message,
    Negotiate_Algo,Algo : message,
    Get_Digests,Digests : message,
    Get_Cert : message,
    Get_Measurements,Measurements : message,
    Na1,Na2,Nb1,Nb2 : text,
    Kc1 : public_key

  init
    State := 1

  transition

  % Sending VERSION message
  1.  State = 1    /\ RCV(Get_Ver') =|>
      State' := 3 /\ SND(Ver)

  % Sending CAPABILITIES message
  2.  State = 3    /\ RCV(Get_Cap') =|>
      State' := 5 /\ SND(Cap)

  % Sending ALGORITHMS message
  3.  State = 5    /\ RCV(Negotiate_Algo') =|>
      State' := 7 /\ SND(Algo)

  % Sending DIGESTS message
  4.  State = 7    /\ RCV(Get_Digests') =|>
      State' := 9 /\ SND(Hash(Kc1))
                  /\ witness(A,B,auth_kc1,Kc1)
                  /\ secret(Kc1,secrecy_kc1,{A,B})

  % Sending CERTIFICATES message
  5.  State = 9     /\ RCV(Get_Cert') =|>
      State' := 11 /\ SND({Kc1}_K)
                   /\ witness(A,B,auth_kc1,Kc1)
                   /\ secret(Kc1,secrecy_kc1,{A,B
  })

  % Sending CHALLENGE_AUTH message
  6.  State = 11    /\ RCV(Na1') =|>
      State' := 13 /\ Nb1' := new()
                   /\ SND(Nb1'.{Na1'}_inv(Kc1))
                   /\ witness(A,B,auth_Nb,Nb1')
                   /\ request(A,B,auth_Na,Na1')
                   /\ secret(Nb1',secrecy_Nb,{A,B
  })

  % Sending MEASUREMENTS message
  7.  State = 13    /\ RCV(Na2') =|>
      State' := 15 /\ Nb2' := new()
                   /\ SND(Nb2'.{Na2'}_inv(Kc1))
                   /\ witness(A,B,auth_Nb,Nb2')
                   /\ request(A,B,auth_Na,Na2')
                   /\ secret(Nb2',secrecy_Nb,{A,B
  })

end role
```

In the case under consideration, the external chip imitates a responder. Consequently, it always responds to the host system's messages. The first message the external chip receives is the *Get_Ver* message. It responds by updating its state and sending the *Ver* message. The chip then receives the *Get_Cap* message and sends the *Cap* response message to the host chip to convey its capabilities. The chip then gets a *Negotiate_Algo* message and, in response, selects a hash algorithm from the host's shared list of hash algorithms. Typically, the chip selects the greatest common hash algorithm and returns it to the host via an *Algo* message. It receives the *Get_Digests* message from the host in the subsequent state. The chip returns the hash values for all available certificates. In the proposed FV, the external chip's public key is hashed and returned to the host chip as part of the HLPSL description. The host then issues a challenge to the chip in the form of a nonce. The chip creates its own nonce and uses its private key to encrypt the received nonce. The created and encrypted nonce is then transmitted to the host. After the challenge authentication, the host sends the chip a measurements request that contains a nonce. The chip encrypts the received nonce and then creates a new nonce in response to the request. Then it returns its measurement blocks, as well as the encrypted host nonce and its newly created nonce.

3) HLPSL CODE FOR SESSION ROLE

The HLPSL code for the session role is given below:

```
role session(
  A, B : agent,
  K: symmetric_key,
  Hash : hash_func)
def=

  local
    SBA, RBA, SAB, RAB : channel (dy)

  composition
    host_chip (A, B, Hash, K, SAB, RAB)
    /\ external_chip (A, B, Hash, K, SBA, RBA)
end role
```

The session role combines the defined protocol participants. In this case, for example, the roles of the host chip and the external chip are glued together. Both roles are instantiated inside the session role. The session role defines local variables for all transmitting and receiving channels.

4) HLPSL CODE FOR ENVIRONMENT ROLE

Following is the HLPSL specification for the environment role:

```
role environment()
def=

  const   a, b : agent,
    h : hash_func,
    k: symmetric_key,
    auth_kc1, auth_Na, auth_Nb : protocol_id,
    secrecy_kc1, secrecy_Na, secrecy_Nb :
    protocol_id

  intruder_knowledge = {a,b}

  composition
    session(a,b,k,h) /\
    session(a,i,k,h)
end role

% -------------
goal
  authentication_on auth_kc1, auth_Na, auth_Nb,
    auth_Nc
  secrecy_of secrecy_kc1, secrecy_Na, secrecy_Nb
end goal

% -------------
environment()
```

This role manages the sessions. The environment role can accommodate numerous sessions. As previously mentioned, in this work, we define two sessions. Furthermore, the objectives for secrecy and authentication are specified. In this role, the protocol (including the initial knowledge of the intruder) and the scenario to be implemented, i.e., the instances of parallel sessions, are examined. The information given to the roles as parameters is constant, with the exception of the communication channels.

IV. SPDM SECURITY VALIDATION

SPDM provides a set of security mechanisms, including mutual authentication, data integrity, confidentiality, and replay attack protection, among others. The protocol is designed to be flexible and extensible to support various security requirements and use cases. The majority of these SPDM security features have been investigated in this research work. Replay attack prevention is one of the features provided by SPDM. It uses sequence numbers and timestamps to detect and prevent replay attacks. Besides replay attacks, the authentication of the guest chip is verified, and confidentiality and secrecy are also checked.

The AVISPA+SPAN tools are used to examine the following security features of the SPDM protocol.

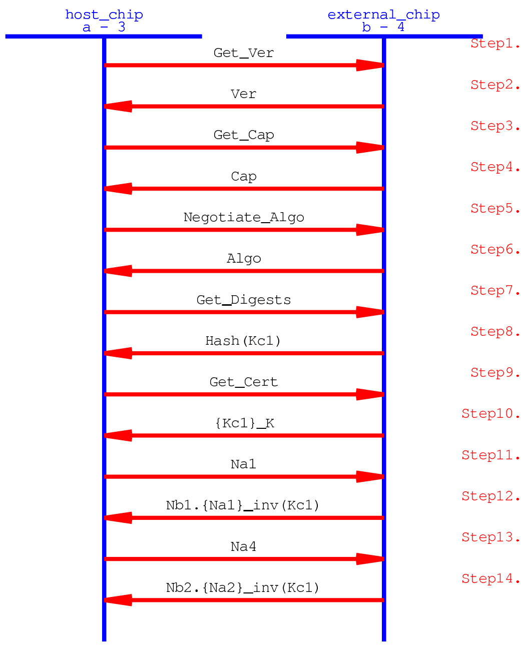

FIGURE 5. **SPDM protocol simulation.**

A. PROTOCOL SIMULATION

The flow of messages in the SPDM protocol is validated using the protocol simulation feature of SPAN. The simulation of the protocol is depicted in Fig. 5. The SPAN tool enables the visualization of the sequential message exchange of the SPDM protocol. Furthermore, the protocol simulation is useful for eliminating and correcting semantic errors in the HLPSL protocol description. The complete message exchange in Fig. 5 indicates that the HLPSL specification contains no semantic error.

B. AUTHENTICATION

In this step, the endpoints confirm that they are communicating with the desired entity. Two endpoints are involved in the SPDM protocol, namely the host chip and the external chip. In the HLPSL specification, the authentication property of a protocol can be examined using the *request* and *witness* clauses, which permit endpoints to declare that they want to be the peer and will agree on a value (variable) for authentication.

Here, the request-witness pair of the host chip authenticates the external chip using the nonce *Nb1*. This authentication is

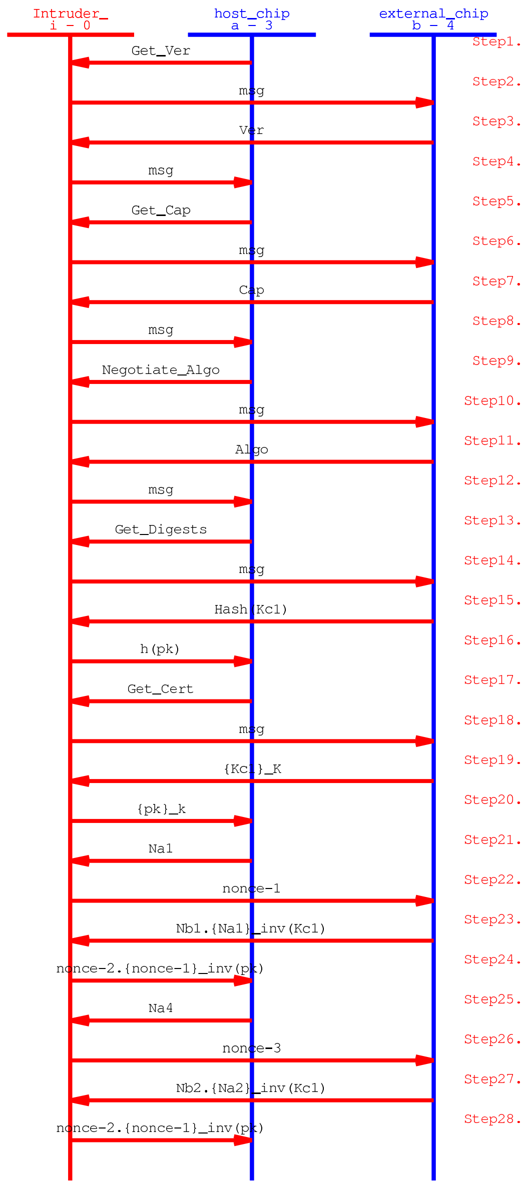

FIGURE 6. **Intruder simulation of SPDM protocol.**

declared as the authentication goal in the goal section of the HLPSL.

witness(A,B,auth_Nb,Nb1')

This indicates that the host chip expects the external chip to authenticate it on the *Nb1* protocol. Here, *auth_Nb* represents the protocol identifier that is later used to set the goal.

request(A,B,auth_Nb,Nb1')

which indicates that the external chip has requested that the host chip authenticate itself on *Nb1*. The authentication goal is finally defined in the *Environment* using the protocol identity *auth_Nb* as,

authentication_on auth_Nb

Correspondingly, we validated the authentication of several messages using HLPSL specifications.

C. CONFIDENTIALITY SECRECY

During the active SPDM protocol session, this feature indicates that confidential information is not disclosed to an unauthorized endpoint. SPDM maintains secrecy by exchanging messages through an encrypted channel. The confidentiality of secret values may be confirmed in AVISPA by declaring the *secrecy* clause in the role that generates secret information and by adding the secrecy goal of the HLPSL specification. For instance, we checked the secrecy of the external chip's public key by including the following statement in the session for the external chip:

secret(Kc1,secrecy_kc1,A,B)

where the first argument *Kc1* is the public key of the external chip, which is required to be kept secret. The second argument *secrecy_kc1* specifies the secrecy goal identifier, whilst the last argument indicates the agents with whom the secret value is exchanged. Here, we test the secrecy of *Kc1* between the host and external chips by establishing the following goal:

secrecy_of secrecy_kc1

Likewise, we have examined the secrecy of the encrypted nonce values.

D. REPLAY ATTACK

A replay attack happens when an unauthorized agent or intruder captures network communication and delays or repeats it deliberately while impersonating a legal agent. Messages can be made resistant to replay attacks by including a nonce or timestamp. SPDM resists this attack by including a nonce within the handshake nonce. In order to test against a replay attack, the following two protocol sessions were evaluated:

- *session(a,b,k,h)*: This session executes the protocol assuming that all agents participating in the session are legitimate. The simulation for this session is shown in Fig. 5.
- *session(a,i,k,h)*: This session simulates a situation in which an intruder imitates the external chip. The simulation for this session is shown in Fig. 8.

Table 3 summarizes the attack scenarios. The AVISPA tool is only compatible with the Dolev-Yao model. In the Dolev-Yao intruder model, the intruder has complete control of the network, meaning that all communications delivered by agents are routed to the intruder. The intruder may intercept, analyze, and/or modify the message and transmit any message to an agent posing as a legal agent, but without the key, he or she cannot encrypt or decrypt the communication.

Figure 6 depicts the intruder simulation of the *session(a,b,k,h)*. The outcome of the OFMC backend with intruder simulation demonstrates that the protocol is unsafe with two legitimate agents, *a* and *b* in the presence of an intruder *i*. The output trace of the intruder simulation of *session(a,b,k,h)* using the OFMC backend is as follows:

```
% OFMC
% Version of 2006/02/13
SUMMARY
  UNSAFE
DETAILS
  ATTACK_FOUND
PROTOCOL
  /home/span/span/testsuite/results/get_digests.if
GOAL
  authentication_on_auth_Na
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 0.05s
  visitedNodes: 19 nodes
  depth: 5 plies
ATTACK TRACE
i -> (b,3): x250
(b,3) -> i: dummy_msg
i -> (b,3): x279
(b,3) -> i: dummy_msg
i -> (b,3): x308
(b,3) -> i: dummy_msg
i -> (b,3): x337
(b,3) -> i: h(dummy_pk)
i -> (b,3): x366
(b,3) -> i: {dummy_pk}_k
i -> (b,3): x395
(b,3) -> i: Nb1(6).{x395}_inv(dummy_pk)


% Reached State:
%
% request(a,b,auth_Na,x395,3)
% secret(Nb1(6),secrecy_Nb,set_146)
% contains(a,set_146)
% contains(b,set_146)
% witness(a,b,auth_kc1,dummy_pk)
% secret(dummy_pk,secrecy_kc1,set_145)
% contains(a,set_145)
% contains(b,set_145)
% secret(dummy_pk,secrecy_kc1,set_144)
% contains(a,set_144)
% contains(b,set_144)
% state_external_chip(b,a,h,k,13,x250,dummy_msg,
    x279,dummy_msg,x308,dummy_msg,x337,dummy_msg,
    x366,dummy_msg,dummy_msg,x395,dummy_nonce,Nb1
    (6),dummy_nonce,dummy_pk,dummy_sk,dummy_nonce,
    dummy_nonce,set_144,set_145,set_146,set_147,3)
% state_host_chip(a,b,h,k,0,dummy_msg,dummy_msg,
    dummy_msg,dummy_msg,dummy_msg,dummy_msg,
    dummy_msg,dummy_msg,dummy_msg,dummy_msg,
    dummy_msg,dummy_pk,dummy_set,dummy_nonce,
    dummy_nonce,dummy_nonce,dummy_nonce,
    dummy_nonce,dummy_nonce,dummy_nonce,dummy_sk,
    dummy_nonce,dummy_nonce,set_119,set_120,
    set_121,3)
% witness(a,b,auth_Nb,Nb1(6))
```

This scenario will never occur in this work since the SPDM protocol is executed between two devices. However, Fig. 6 depicts the exchange of SPDM messages across three devices. In addition, the intruder can listen to the exchanged messages, but it cannot decrypt the external chip's public key, as only a symmetric key between the host chip and the external chip can be used to decrypt the key. This symmetric key is unknown to the intruder. However, the intruder chip imitates a host chip. The output trace for the intruder simulation of *session(a,b,k,h)* is shown as

```
(host_chip, 3) -> (Intruder_, 0) : x(Get_Ver,
    Listen_i)
(Intruder_, 0) -> (external_chip, 4) : x(msg,
    Get_Ver)
(external_chip, 4) -> (Intruder_, 0) : x(Ver,
    Listen_i)
(Intruder_, 0) -> (host_chip, 3) : x(msg,Ver)
(host_chip, 3) -> (Intruder_, 0) : x(Get_Cap,
    Listen_i)
(Intruder_, 0) -> (external_chip, 4) : x(msg,
    Get_Cap)
(external_chip, 4) -> (Intruder_, 0) : x(Cap,
    Listen_i)
(Intruder_, 0) -> (host_chip, 3) : x(msg,Cap)
(host_chip, 3) -> (Intruder_, 0) : x(
    Negotiate_Algo,Listen_i)
(Intruder_, 0) -> (external_chip, 4) : x(msg,
    Negotiate_Algo)
(external_chip, 4) -> (Intruder_, 0) : x(Algo,
    Listen_i)
(Intruder_, 0) -> (host_chip, 3) : x(msg,Algo)
(host_chip, 3) -> (Intruder_, 0) : x(Get_Digests,
    Listen_i)
(Intruder_, 0) -> (external_chip, 4) : x(msg,
    Get_Digests)
(external_chip, 4) -> (Intruder_, 0) : x(apply(
    Hash,Kc1),Listen_i)
(Intruder_, 0) -> (host_chip, 3) : x(apply(h,pk),
    apply(Test_Hash,Kc1))
(host_chip, 3) -> (Intruder_, 0) : x(Get_Cert,
    Listen_i)
(Intruder_, 0) -> (external_chip, 4) : x(msg,
    Get_Cert)
(external_chip, 4) -> (Intruder_, 0) : x(scrypt(K,
    Kc1),Listen_i)
(Intruder_, 0) -> (host_chip, 3) : x(scrypt(k,pk),
    scrypt(Test_K,Kc1))
(host_chip, 3) -> (Intruder_, 0) : x(Na1_new,
    Listen_i)
(Intruder_, 0) -> (external_chip, 4) : x(nonce-1,
    Na1)
(external_chip, 4) -> (Intruder_, 0) : x(pair(
    Nb1_new,crypt(inv(Kc1),Na1)),Listen_i)
(Intruder_, 0) -> (host_chip, 3) : x(pair(nonce-2,
    crypt(inv(pk),nonce-1)),pair(Nb1,crypt(inv(
    Test_Kc1),Na3)))
(host_chip, 3) -> (Intruder_, 0) : x(Na4_new,
    Listen_i)
(Intruder_, 0) -> (external_chip, 4) : x(nonce-3,
    Na2)
(external_chip, 4) -> (Intruder_, 0) : x(pair(
    Nb2_new,crypt(inv(Kc1),Na2)),Listen_i)
(Intruder_, 0) -> (host_chip, 3) : x(pair(nonce-2,
    crypt(inv(pk),nonce-1)),pair(Nb2,crypt(inv(
    Test_Kc1),Na5)))
```

The host chip can be connected, over the same bus, to more than two chips using, for example, the I2C interface, as shown in Fig 7. Therefore, the host chip, which is the master, may simultaneously establish the SPDM protocol between several chips. The master should orchestrate the data exchange with an external peripheral by sending the address of the appropriate slave. One of the expected scenarios is that one of the connected slaves is a malicious peripheral. So, the illegal peripheral can intercept the data on the bus, but it cannot decrypt the external chip's public key, as only a symmetric key between the host chip and the external chip can be used to decrypt the key. Figure 8 demonstrates the actual situation

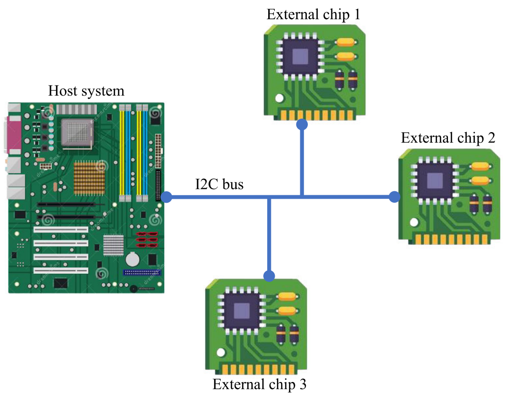

FIGURE 7. Use-case of multiple chip-to-chip communication over shared I2C bus.

involving an unauthorized external chip. In this scenario, the host chip initiates SPDM exchanges with the intruder (unauthorized external chip). As the exchange of the initial three messages occurs prior to authentication, there are no issues. As soon as the host chip requests the digests, the intruder is unable to supply the digests of the certificate, prompting the host chip to terminate subsequent message exchanges and implying that the external chip is illegitimate. The OFMC output trace for this simulation can be seen below.

```
% OFMC
% Version of 2006/02/13
SUMMARY
  SAFE
DETAILS
  BOUNDED\_NUMBER\_OF\_SESSIONS
PROTOCOL
  /home/span/span/testsuite/results/get\_digests.
    if
GOAL
  as\_specified
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 0.02s
  visitedNodes: \text{5}~nodes
  depth: \text{4}~plies
```

E. VERIFICATION RESULTS

The validation results for SPDM using the OFMC back-end are presented in Table 4. The table shows the cases that have been examined for potential security goal breaches.

The obtained results are reported in Table 4. These results are summarized by AVISPA using one of the following:

- **Safe** means that the protocol does not breach any of the security goals outlined in the HLPSL specification.
- **Unsafe** implies that the protocol for which an attack trace was identified contains a security vulnerability.
- **Inconclusive** refers to the fact that, due to fundamental difficulties, AVISPA is unable to examine the protocol.

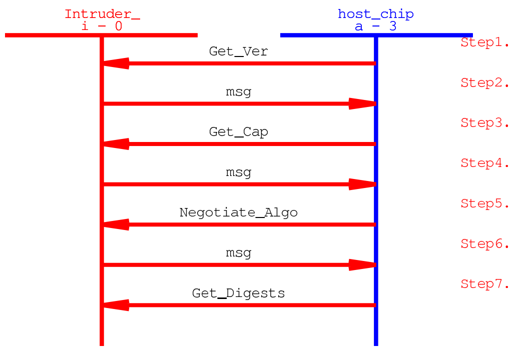

FIGURE 8. SPDM protocol simulation with illegitimate external chip.

TABLE 3. Attack scenarios.

Scenario	**Session** Configuration
1	*session(a, b, h)*
2	*session(a, i, h)*

F. COMPUTATIONAL OVERHEAD

The tool is efficient and consumes minimal memory and central processing unit (CPU) resources. We used the system monitor tool to estimate the CPU and memory usage and found that, on average, the tool uses approximately 1% of CPU resources and 30% of memory resources. It should be noted that the tool runs on an Ubuntu platform within a virtual machine, which is allocated 4096 MB of base memory and three processors by the host system. Despite these modest resources, the verification process completes within seconds, demonstrating that the verification process is both feasible and fast enough.

AVISPA+SPAN is a powerful tool for verifying security protocols, but it can be computationally intensive and may not scale well for very large systems or complex protocols. In particular, the verification time can increase exponentially with the number of protocol runs, making it impractical for large systems. However, there are techniques that can be used to improve the scalability of formal verification through AVISPA+SPAN, such as reducing the size of the protocol, optimizing the verification parameters, and parallelizing the verification process across multiple machines. We proposed a lightweight SPDM protocol that is not computationally intensive. As a result, the tool can traverse the required number of states quickly.

We estimated the computational complexity of the SPDM protocol in addition to that of the verification tool. The protocol is executed on two Raspberry Pi 4B devices, with one designated as the requester and the other as the responder module. The Raspberry Pi platform integrates a powerful Broadcom BCM2711 processor with a frequency of 1.8GHz and 4GB of internal memory. The total runtime of the protocol is approximately 29 seconds, including a 2-second initial communication stage, a 0.70-second *GET_DIGEST* stage, a 3-second *GET_CERTIFICATE* stage, a 10-second *CHALLENGE* phase, a 12-second *KEY_EXCHANGE* stage, and a 0.7-second secure communication session.

TABLE 4. Validation results with OFMC backend.

Scenario	Description	Result
session(a,b,k,h)	VisitedNodes: 5 nodes Depth: 4 plies SearchTime: 0.02s	Unsafe
session(a,i,k,h)	VisitedNodes: 19 nodes Depth: 5 plies SearchTime: 0.05s	Safe

V. CHALLENGES INVOLVED

The SPDM protocol provides a comprehensive set of tools and features to ensure the authentication, secrecy, and confidentiality of data and devices. This work focuses on C2C-ZT, where the host chip initiates the protocol and authenticates the guest chip using digital certificates. Major challenges in formally verifying the proposed lightweight SPDM protocol for C2C-ZT include:

1) We model the private key of the CA as a symmetric public key between the chips and use *inv()* to convert it into the private key. To ensure the confidentiality and secrecy of the CA's private key, we have implemented specific features. While this method replicates the concept of a public-key infrastructure (PKI), finding an exact solution to this problem remains a challenge.
2) While we assume that the guest chip contains a digital certificate, in reality, most hardware chips do not have digital certificates, and it is unclear whether they contain them. Implementing PKI for hardware chips requires the host chips to store the public keys of major CAs, which can be an overhead.
3) To minimize the amount of data exchanged between devices, SPDM incorporates the sharing of digital certificate hashes before the actual exchange takes place. Although this feature significantly reduces data exchange overhead, implementing it on the AVISPA+SPAN tool can be challenging due to the presence of *if-else* conditions. If the host chip detects the presence of digital certificate hashes, it skips the step of requesting the certificate and proceeds directly to sending the challenge. However, if the hashes are not found, it will request the digital certificates before proceeding.
4) A major challenge is to thoroughly study the SPDM protocol and develop the necessary features that guarantee the minimum requirements for ZT authentication while remaining lightweight.
5) The SPDM protocol relies on asymmetric cryptography and public-key certificates for authentication. While these operations can be resource-intensive, they are necessary for ensuring ZT. This means that chip manufacturers may need to provide more on-chip resources to support authentication. In the proposed lightweight SPDM, we suggest using lightweight digital certificates instead of the standard X.509 certificates [42]. These lightweight certificates are already being used by the research community for IoT devices and are more efficient because they use elliptic curve cryptography (ECC) instead of Rivest-Shamir-Adleman (RSA) for encryption. In summary, more resources than what are currently available on-chip may be required for achieving C2C-ZT. However, using lightweight digital certificates and ECC can help to reduce the resource requirements and make the authentication process more efficient. Still, we think that it is accepted that ZT authentication comes with additional costs.

VI. CONCLUSION

This paper describes a C2C architecture based on the ZT principles. The SPDM protocol, which allows the authentication of the external chip to the host chip, is described in detail. In addition, the protocol enables the establishment of a secure communication channel for the exchange of data between the two chips. A detailed formal validation of SPDM protocol is performed using the widely-used AVISPA+SPAN tool. Simulations are conducted for both an authorized and unauthorized external chip. We infer from the FV that the protocol meets the necessary security features (secrecy, authentication, and freshness) and is secure against active and passive attacks. This document also includes the HLPSL specification for SPDM.

Furthermore, we intend to use fuzzy testing to supplement formal verification and identify any potential vulnerabilities that may have been missed during formal verification.

REFERENCES

[1] N. F. Syed, S. W. Shah, A. Shaghaghi, A. Anwar, Z. Baig, and R. Doss, "Zero trust architecture (ZTA): A comprehensive survey," *IEEE Access*, vol. 10, pp. 57143–57179, 2022.

[2] J. L. Hardcastle. (Dec. 2022). *Google Brings Beyondcorp Zero-Trust Security to the Masses*. Accessed: Dec. 2022. [Online]. Available: https://www.sdxcentral.com/articles/news/google-productizes-beyondcorp-zero-trust-network-security/2020/04/

[3] R. Ward and B. Beyer, "BeyondCorp: A new approach to enterprise security," *Login*, vol. 39, no. 66, pp. 6–11, 2014. [Online]. Available: [Online]. Available: https://research.google/pubs/pub43231/

[4] *Implementing a Zero Trust Security Model at Microsoft*. Microsoft Corporation. Accessed: Dec. 2022. [Online]. Available: https://www.microsoft.com/en-us/insidetrack/implementing-a-zero-trust-security-model-at-microsoft

[5] T. Dawoud. (2021). *Zero Trust Deployment Guide for Microsoft Azure Active Directory*. Accessed: Dec. 2022. [Online]. Available: https://www.microsoft.com/en-us/security/blog/2020/04/30/zero-trust-deployment-guide-azure-active-directory/

[6] O. Hireche, C. Benzaíd, and T. Taleb, "Deep data plane programming and AI for zero-trust self-driven networking in beyond 5G," *Comput. Netw.*, vol. 203, Feb. 2022, Art. no. 108668.

[7] L. Ferretti, F. Magnanini, M. Andreolini, and M. Colajanni, "Survivable zero trust for cloud computing environments," *Comput. Secur.*, vol. 110, Nov. 2021, Art. no. 102419.

[8] S. W. Shah, N. F. Syed, A. Shaghaghi, A. Anwar, Z. Baig, and R. Doss, "LCDA: Lightweight continuous device-to-device authentication for a zero trust architecture (ZTA)," *Comput. Secur.*, vol. 108, Sep. 2021, Art. no. 102351.

[9] B. Chen, S. Qiao, J. Zhao, D. Liu, X. Shi, M. Lyu, H. Chen, H. Lu, and Y. Zhai, "A security awareness and protection system for 5G smart healthcare based on zero-trust architecture," *IEEE Internet Things J.*, vol. 8, no. 13, pp. 10248–10263, Jul. 2021.

[10] M. Sultana, A. Hossain, F. Laila, K. A. Taher, and M. N. Islam, "Towards developing a secure medical image sharing system based on zero trust principles and blockchain technology," *BMC Med. Informat. Decis. Making*, vol. 20, no. 1, pp. 1–10, Oct. 2020.

[11] Y. Liu, X. Hao, W. Ren, R. Xiong, T. Zhu, K. R. Choo, and G. Min, "A blockchain-based decentralized, fair and authenticated information sharing scheme in zero trust Internet-of-Things," *IEEE Trans. Comput.*, vol. 72, no. 2, pp. 501–512, Feb. 2023.

[12] A. N. Jahromi, H. Karimipour, A. Dehghantanha, and K. R. Choo, "Toward detection and attribution of cyber-attacks in IoT-enabled cyber–physical systems," *IEEE Internet Things J.*, vol. 8, no. 17, pp. 13712–13722, Sep. 2021.

[13] C. Pu, A. Wall, K. R. Choo, I. Ahmed, and S. Lim, "A lightweight and privacy-preserving mutual authentication and key agreement protocol for Internet of Drones environment," *IEEE Internet Things J.*, vol. 9, no. 12, pp. 9918–9933, Jun. 2022.

[14] A. Stern, H. Wang, F. Rahman, F. Farahmandi, and M. Tehranipoor, "ACED-IT: Assuring confidential electronic design against insider threats in a zero-trust environment," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 10, pp. 3202–3215, Oct. 2022.

[15] M. Hasan, J. Cruz, P. Chakraborty, S. Bhunia, and T. Hoque, "Trojan resilient computing in COTS processors under zero trust," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 10, pp. 1412–1424, Oct. 2022.

[16] S. Smaoui, F. Zarai, M. S. Obaidat, K. F. Hsiao, and L. Kamoun, "HIP_IKEv2: A proposal to improve internet key exchange protocol-based on host identity protocol," in *Proc. Intl. Conf. Simulation Modeling Methodolog. Technol. Appl. (SIMULTECH)*, 2013, pp. 404–411.

[17] V. Odelu, A. K. Das, and A. Goswami, "A secure biometrics-based multi-server authentication protocol using smart cards," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 9, pp. 1953–1966, Sep. 2015.

[18] M. Wazid, A. K. Das, V. Odelu, N. Kumar, and W. Susilo, "Secure remote user authenticated key establishment protocol for smart home environment," *IEEE Trans. Depend. Secure Comput.*, vol. 17, no. 2, pp. 391–406, Mar. 2020.

[19] A. Dua, N. Kumar, A. K. Das, and W. Susilo, "Secure message communication protocol among vehicles in smart city," *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 4359–4373, May 2018.

[20] S. Paliwal, "Hash-based conditional privacy preserving authentication and key exchange protocol suitable for industrial Internet of Things," *IEEE Access*, vol. 7, pp. 136073–136093, 2019.

[21] L. Viganò, "Automated security protocol analysis with the AVISPA tool," *Electron. Notes Theor. Comput. Sci.*, vol. 155, pp. 61–86, May 2006.

[22] Y. Palmo, S. Tanimoto, H. Sato, and A. Kanai, "Optimal federation method for embedding Internet of Things in software-defined perimeter," *IEEE Consum. Electron. Mag.*, early access, Sep. 19, 2022, doi: 10.1109/MCE.2022.3207862.

[23] J. Guo and M. Xu, "ZTESA—A zero-trust endogenous safety architecture: Gain the endogenous safety benefit, avoid insider threats," in *Proc. Int. Symp. Comput. Appl. Inf. Syst. (ISCAIS)*, May 2022, pp. 192–202.

[24] X. Chen, W. Feng, N. Ge, and Y. Zhang, "Zero trust architecture for 6G security," 2022, *arXiv:2203.07716*.

[25] K. Bicakci, Y. Uzunay, and M. Khan, "Towards zero trust: The design and implementation of a secure end-point device for remote working," in *Proc. Int. Conf. Inf. Secur. Cryptol. (ISCTURKEY)*, Dec. 2021, pp. 28–33.

[26] J. Lowdermilk and S. Sethumadhavan, "Towards zero trust: An experience report," in *Proc. IEEE Secure Develop. Conf. (SecDev)*, Oct. 2021, pp. 79–85.

[27] S. Dutta, B. Grisafe, C. Frentzel, Z. Enciso, M. S. Jose, J. Smith, K. Ni, S. Joshi, and S. Datta, "Experimental demonstration of gate-level logic camouflaging and run-time reconfigurability using ferroelectric FET for hardware security," *IEEE Trans. Electron Devices*, vol. 68, no. 2, pp. 516–522, Feb. 2021.

[28] M. Avalle, A. Pironti, and R. Sisto, "Formal verification of security protocol implementations: A survey," *Formal Aspects Comput.*, vol. 26, no. 1, pp. 99–123, Jan. 2014.

[29] "Security protocol and data model (SPDM) specification," Distrib. Manag. Task Force (DMTF), Portland, OR, USA, Tech. Rep. 1.2.1, 2022. [Online]. Available: https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.2.1.pdf

[30] J. Yao, K. Matusiewicz, and V. Zimmer, "Post quantum design in SPDM for device authentication and key establishment," *Cryptography*, vol. 6, no. 4, p. 48, Sep. 2022.

[31] "PCI express 7.0 specification," PCI-SG, Singapore, Tech. Rep., 2022. [Online]. Available: https://pcisig.com/specifications/pci-express-70-specification

[32] "Trusted platform module 2.0 library specification," TCG, Washington, DC, USA, Tech. Rep., 2022. [Online]. Available: https://trustedcomputinggroup.org/tpm-2-0-library-specification-approved-isoiec-international-standard/

[33] "PCI express base specification revision 6.0.1, version 1.0," TCG, Washington, DC, USA, Tech. Rep., 2022. [Online]. Available: https://members.pcisig.com/wg/PCI-SIG/document/18363

[34] R. C. A. Alves, B. C. Albertini, and M. A. Simplicio, "Securing hard drives with the security protocol and data model (SPDM)," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2022, pp. 446–447.

[35] R. C. Merkle, "Secure communications over insecure channels," *Commun. ACM*, vol. 21, no. 4, pp. 294–299, Apr. 1978.

[36] D. Basin, S. Mödersheim, and L. Viganò, "OFMC: A symbolic model checker for security protocols," *Int. J. Inf. Secur.*, vol. 4, no. 3, pp. 181–208, Jun. 2005.

[37] M. Turuani, "The CL-Atse protocol analyser," in *Proc. Int. Conf. Rewriting Techn. Appl.*, F. Pfenning, Ed. Seattle, WA, USA: Springer, 2006, pp. 277–286. [Online]. Available: https://link.springer.com/chapter/10.1007/11805618_21

[38] A. Armando and L. Compagna, "SAT-based model-checking for security protocols analysis," *Int. J. Inf. Secur.*, vol. 7, no. 1, pp. 3–32, Jan. 2008.

[39] Y. Boichut, P.-C. Héam, O. Kouchnarenko, and F. Oehl, "Improvements on the Genet and Klay technique to automatically verify security protocols," in *Proc. Int. Ws. Automated Verification Infinite-State Syst. (AVIS)*, Oct. 2022, pp. 1–11.

[40] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. Inf. Theory*, vol. IT-29, no. 2, pp. 198–208, Mar. 1983.

[41] S. Dey and A. Hossain, "Session-key establishment and authentication in a smart home network using public key cryptography," *IEEE Sensors Lett.*, vol. 3, no. 4, pp. 1–4, Apr. 2019.

[42] F. Forsby, M. Furuhed, P. Papadimitratos, and S. Raza, "Lightweight X.509 digital certificates for the Internet of Things," in *Interoperability, Saf. Secur. IoT, Int. Conf. IoT (InterIoT)*, 2018, pp. 123–133.

ASHFAQ AHMED (Senior Member, IEEE) received the M.S. and Ph.D. degrees from the Department of Electronics and Telecommunications, Politecnico di Torino, Turin, Italy, in 2010 and 2014, respectively. He is currently with the Center for Cyber-Physical Systems, Department of Electrical Engineering and Computer Science, Khalifa University (KU), Abu Dhabi, United Arab Emirates. His research interests include hardware security, security protocols, computational intelligence, evolutionary algorithms, convex optimization, resource allocation, applied optimization for 5G and beyond 5G applications, cloud computing, and physical layer wireless communication.

ABDULHADI SHOUFAN (Member, IEEE) received the Dr.-Ing. degree from Technische University Darmstadt, Germany, in 2007. He is currently an Associate Professor of electrical engineering and computer science with Khalifa University, Abu Dhabi. His research interests include drone security, safe operation, embedded security, cryptography hardware, learning analytics, and engineering education.

KAIS BELWAFI (Member, IEEE) received the M.Sc. degree in intelligent and communication systems from the Highest School of Engineering of Sousse, Tunisia, in 2012, and the Ph.D. degree in sciences and technology of information and communication from the University of Paris–Seine, Cergy-Pontoise, France, in 2017. He is currently a Research Scientist with the Electrical and Computer Engineering Department, Khalifa University. His main research interests include the security of embedded systems, drone security, brain–computer interfaces, machine learning, signal processing, embedded and real-time systems, and HW/SW co-design.