Proteção em hardware com Security Protocol and Data Model

Gustavo C. Bastos[1], **Marcos A. Simplício Jr.**[1], **Bruno C. Albertini.**[1]

[1]Escola Politécnica - Universidade de São Paulo (USP)

gustavocerq5@usp.br
mjunior@larc.usp.br, balbertini@usp.br

Abstract. *A critical security threat to modern computer systems involves hardware-level attacks, such as firmware manipulation. These attacks are particularly dangerous because they enable unauthorized data access at the bus level, making eavesdropping or tampering difficult to detect. One possible countermeasure is the Security Protocol and Data Model (SPDM), an industry standard for hardware component mutual attestation and secure channel establishment. In this work, we present the design, implementation, and execution of SPDM on an FPGA (Field-Programmable Gate Array). Our test environment consists of a RISC-V SoC and an Ethernet card, where the bootcode (BIOS) authenticates the peripheral (Ethernet card) using SPDM.*

Resumo. *Uma ameaça crítica à segurança de sistemas computacionais modernos envolve ataques em nível de hardware, como a manipulação de* firmware*. Esses ataques são particularmente perigosos pois permitem acesso não autorizado a dados no nível do barramento, dificultando a detecção de* eavesdropping *ou adulteração de dados. Uma possível contramedida é o* Security Protocol and Data Model *(SPDM), um padrão da indústria para atestação mútua de componentes de hardware e estabelecimento seguro de canais. Neste trabalho, apresentamos o projeto, a implementação e a execução do SPDM em um FPGA (*Field-Programmable Gate Array*). Nosso ambiente de teste consiste em um SoC RISC-V e uma placa Ethernet, onde o código de inicialização (BIOS) autentica o periférico (placa Ethernet) usando o SPDM.*

1. Introdução

A atual cadeia de produção de sistemas computacionais é descentralizada, envolvendo componentes de hardware e software desenvolvidos por múltiplos fabricantes especializados. É comum um único componente ser desenvolvido e fabricado por mais de um fabricante, terminando com um integrador. Este cenário permite que alguns fabricantes em posições privilegiadas na cadeia de produção explorem a confiança da cadeia para fins maliciosos. Ações como a manipulação de componentes de hardware ou de seu *firmware* já foram relatadas na literatura [Cui et al. 2013] neste cenário. A preocupação com este tipo de ataque executado em baixo nível agrava-se pois as ferramentas tradicionais de segurança, como *firewalls* e antivírus, operam acima do nível do sistema operacional e, portanto, são ineficazes [Choi et al. 2016].

A detecção de modificações no *firmware* não é um problema novo e está relativamente madura na literatura [Basnight 2013, Wang et al. 2015, Choi et al. 2016, Brown et al. 2021, Zhang et al. 2023]. O SPDM (*Security Protocol and Data Model*) [DTMF 2022] diferencia-se das propostas pois propõe um conjunto de mecanismos e

formatos para atestação de componentes de hardware e *firmware*, aliado a protocolos para acordo de chaves criptográficas e, consequentemente, o estabelecimento de sessão confidencial no nível do barramento. O protocolo impõe uma camada de segurança através da utilização desses algoritmos criptográficos que são consolidados na literatura, adicionando uma complexidade para o atacante, já que os algoritmos precisariam ser quebrados ou terem suas chaves vazadas para obter acesso aos dados de interesse. A adoção do SPDM em relação a outros métodos se dá exatamente pela proteção dos dados em trânsito no barramento, evitando possíveis ataques em baixo nível [Mehta et al. 2020]. A especificação do SPDM segue padrões industriais e de uso geral, especificando o protocolo e algoritmos criptográficos com análise de segurança descrita e aceita na literatura [Cremers et al. 2023, Cremers et al. 2024]. A implementação de referência em *software* é de código aberto [DMTF 2023].

Apesar de ser um padrão aberto e focado na segurança em nível de *hardware*, as implementações conhecidas do SPDM são simulações em *software* [Alves et al. 2022, Nvidia 2022]. Para análises de segurança, a literatura baseia-se em uma biblioteca de código aberto `LibSPDM` [DMTF 2023]. Há também algumas implementações de baixo nível, como a parcial em [Belwafi et al. 2024], onde a parte que inicia a comunicação do protocolo está implementada em um FPGA e o restante em software executando em um Raspberry Pi, com a comunicação entre as partes sobre um barramento I2C.

Um sistema computacional do tipo SoC (*System-on-Chip*) caracteriza-se por um elemento principal de processamento conectado a outros dispositivos, considerados periféricos. A conexão é realizada por um barramento e, no caso dos SoCs, todos os componentes estão no mesmo circuito integrado. É comum na indústria a utilização de componentes de prateleira conhecidos como IP (*Intelectual Property*), pois o projetista pode focar na implementação do SoC e licenciar projetos de periféricos, barramentos ou até mesmo de processadores. No entanto, a confiança em soluções prontas assemelha-se ao problema de confiança na cadeia de produção. Há poucas iniciativas de proteção no nível de barramento, seja este interno ou externo ao circuito integrado, e o SPDM é uma delas.

O método de desenvolvimento usando SoC em detrimento de métodos de emulação ou em software se deve a possibilidade de melhor desempenho em métricas como o tempo de execução. Além disso, uma abordagem completa em hardware realiza o motivo da existência do SPDM: uma mudança do nível de abstração da segurança, ou seja, diminui-se de um nível mais alto de abstração (*software*) para um nível mais baixo (*hardware*), o barramento. Apesar da motivação e das vantagens quanto a métricas de tempo de execução, este tipo de abordagem pode impor uma carga no consumo energético e de área de hardware que variam a depender do nível de segurança desejado. Há então um balanceamento entre o nível de segurança desejado no barramento e os objetivos de custos e desempenho.

Este trabalho tem como objetivo implementar funções de SPDM dentro do contexto de uma FPGA, isto é, uma implementação física do protocolo no formato de sistema em um circuito integrado. A ferramenta desenvolvida é capaz de reconhecer e executar funções da `LibSPDM` dentro do escopo de um *firmware* (*requester*) e de uma placa Ethernet (*responder*) sobre o barramento interno AXI (*Advanced eXtensible Interface*). Nossa implementação considera o *firmware* como o primeiro código executado por um processador de uso geral, portanto também pode ser usada em conjunto com um sistema

operacional, desde que o SPDM inicie a sua execução no código de *boot* (e.g. BIOS).

O processo de desenvolvimento do SoC é descrito na Seção 2, a demonstração de execução do protocolo na Seção 3 e, por fim, apresentamos as considerações finais e perspectivas de trabalhos futuros na Seção 4.

2. Metodologia e Desenvolvimento

A metodologia de trabalho para construção da ferramenta foi dividida em duas partes, *hardware* e *software*. O *hardware* segue o modelo SoC, onde o elemento computacional é uma implementação aberta da arquitetura RISC-V [SiFive 2017]. O barramento usado, assim como os periféricos de suporte e para o teste de SPDM, também foram escolhidos para serem livremente modificáveis. O software consiste na integração da `LibSPDM` no código de boot (BIOS) do SoC, viabilizando a integração *software-hardware*, e no suporte ao SPDM dentro do *kernel* do sistema operacional.

2.1. Hardware

Desde a concepção da ferramenta, visávamos o suporte SPDM para um periférico de rede pois, se atacado, este tipo de periférico é o que tem o maior potencial para vazamento de dados. Neste sentido, definimos a placa de suporte como a NetFPGA-Sume [Digilent 2024], por conter os conectores e *transceivers* necessários para operação com rede, além de um FPGA Virtex-7 690T com capacidade para o SoC, incluindo memórias, células lógicas e circuitos de suporte ao *clock*.

Para a implementação do processador principal do SoC, usamos uma implementação aberta conhecida como Rocket Chip [Asanović et al. 2016]. Esta implementação é suportada por diversas ferramentas como compiladores e geradores de código, além de sistemas operacionais conhecidos como o Linux. Dentre as ferramentas, este processador também é suportado pelo Litex [Kermarrec et al. 2020], um gerador de plataformas. O Litex é capaz de gerar o código sintetizável de uma plataforma SoC baseado em parâmetros.

Além do processador, um SoC possui ao menos um barramento e os periféricos. Escolhemos o barramento AXI por motivos de compatibilidade com modificações externas. Para os periféricos, selecionamos os seguintes: UART (*Universal Asynchronous Receiver / Transmitter*), LEDs, placa de rede Ethernet (SFP+), ROM (*Read Only Memory*), SRAM e SDCard. A UART e os LEDs foram selecionados para serem os pontos de entrada e saída de dados, ou seja, utilizados para ser possível ordenar comandos e visualizar respostas. Em relação às memórias, a ROM é responsável por armazenar a BIOS e a SRAM atua como memória de instruções e de dados (inclui-se aqui *bootloader* e Sistema Operacional). O controlador de SD Card é uma alternativa para armazenamento não volátil de baixo desempenho (e.g. sistemas de arquivos) e, por fim, a placa de rede Ethernet é o alvo do SPDM.

Para todos os periféricos, usamos implementações abertas disponíveis publicamente e compatíveis com o Litex. A placa de rede, alvo do SPDM, foi substituída por uma implementação própria com suporte ao SPDM, mantendo a compatibilidade com a placa original disponível no Litex. O periférico recebeu a inclusão de um banco de registradores mapeados em memória e um processador Microblaze [AMD 2024] com acesso ao barramento AXI e memória interna própria. Este processador faz o papel de elemento

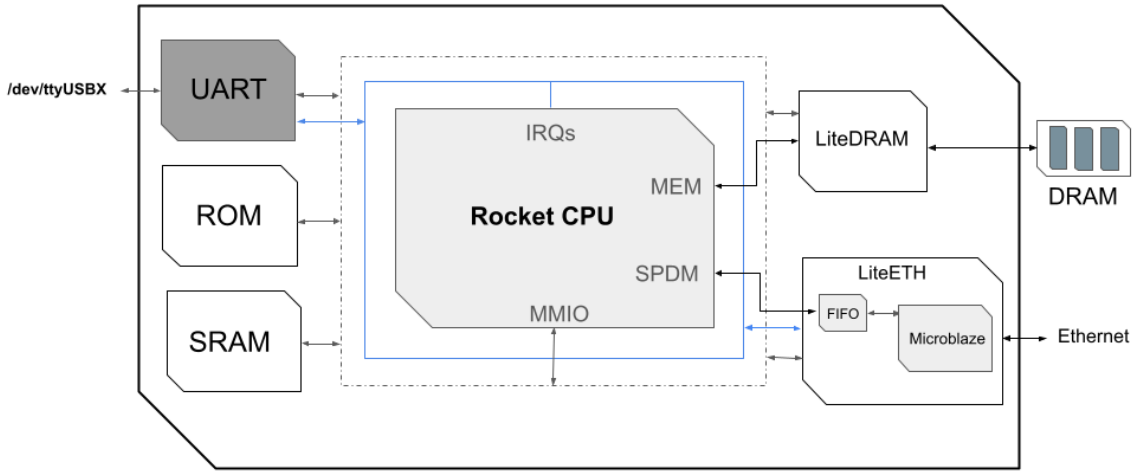

Figura 1. Esquema de Hardware do SoC, adaptado de [Kermarrec et al. 2020, Hub 2023]

computacional da placa de rede, executando um *firmware* onde, além das funções de placa de rede comuns, foi adicionado o SPDM. As trocas de dados com o processador principal do SoC ocorrem por meio do banco de registradores e de uma fila FIFO implementada em hardware. A placa continua suportando o acesso mapeado em memória sem o SPDM, mas responde a todas as fases do protocolo SPDM caso o *software* (e.g. BIOS) a acesse desta forma. A Figura 1 mostra uma visão geral da arquitetura proposta.

2.2. Software

Para o software, usamos a versão pública da `LibSPDM` como ponto de partida, modificando-a para a arquitetura proposta. Disponibilizamos a nossa versão publicamente e o *toolchain* usado é também de código aberto [riscv collab 2023].

No ciclo de *boot*, a BIOS é o primeiro código a ser executado pelo processador principal do SoC, portanto é a melhor candidata para autenticar os dispositivos antes da sua utilização. Para efeitos de segurança, consideramos a BIOS como uma ROM somente de leitura, portanto a cadeia de segurança assume que a BIOS não pode ser comprometida sem ataque físico e é confiável pelo fabricante. A nossa implementação de BIOS suporta a `LibSPDM` integralmente, ou seja, implementa as funções *spdm_requester* e *spdm_responder*, responsáveis por acionar os mecanismos do SPDM e colocar a BIOS como uma autenticadora (*requester*) ou uma autenticada (*responder*). Na prática, a BIOS pode iniciar a autenticação de outros componentes de hardware ou pode ser verificada por um componente externo (e.g. TPM (*Trusted Platform Module*)).

Uma vez que a BIOS autentique os componentes de hardware, cabe a ela continuar o ciclo de *boot*, transferindo o fluxo de execução para o *bootloader*. Escolhemos o *bootloader* da OpenSBI [RISC-V 2020] por suportar tanto a arquitetura do SoC (RISC-V) quanto o sistema operacional (Linux). Finalmente, o *bootloader* termina o ciclo de *boot* transferindo o controle para o *kernel* do sistema operacional.

3. Resultados e Demonstração

Com o *firmware* e o hardware devidamente integrados, montamos uma plataforma SoC capaz de executar e reconhecer o protocolo SPDM. A ferramenta foi inserida no código da BIOS e foi reconhecida pela placa de rede Ethernet, que foi devidamente autenticada.

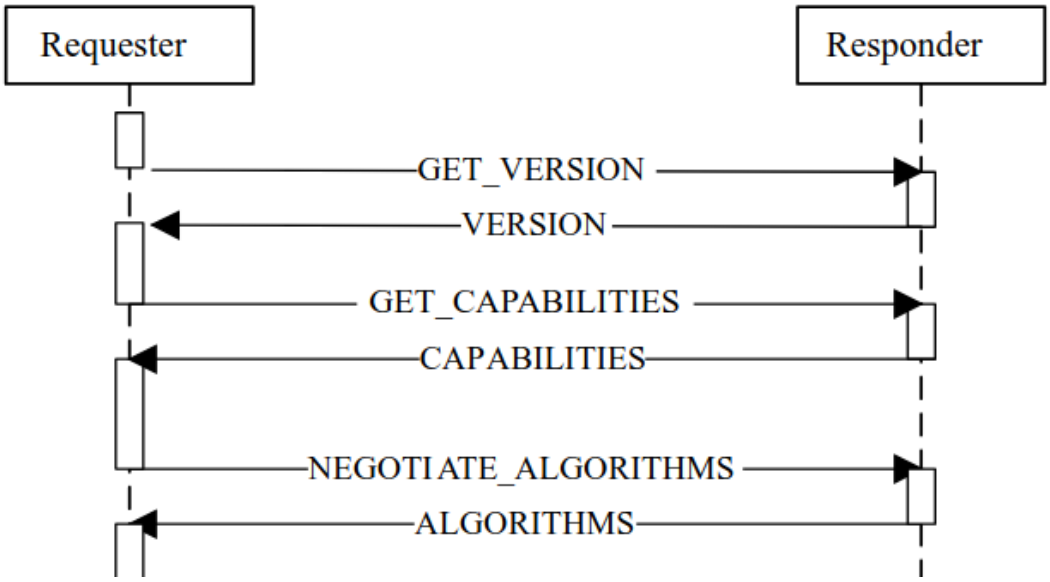

Figura 2. Execução como *Requester* pela BIOS [DTMF 2022]

A Figura 2 mostra o fluxo de execução do protocolo, que pode ser acompanhado no terminal da Figura 3 para a função de *spdm_requester*. No terminal, podemos observar todas as etapas do protocolo: (i) uma alocação dinâmica de memória de um *buffer* e para o contexto SPDM; (ii) os parâmetros SPDM do periférico, como algoritmos suportados; e (iii) a chamada da função para envio dos pacotes. Na Figura 3, pode-se observar os pacotes do SPDM, em primeiro plano os pacotes enviados pelo *requester*, seguidos dos pacotes enviados pelo *responder*. O protocolo completa sua execução de maneira correta ao entregar o código `0x0` em *libspdm_init_connection*. Os pacotes são enviados e recebidos pelo barramento do SoC e armazena-se o pacote na FIFO até o recebimento pelo outro componente.

Em caso análogo, o SPDM pode detectar mudanças maliciosas no *firmware*. A Figura 4 demonstra um caso exemplo em que o *firmware* da Ethernet sofreu um ataque. Neste caso, o algoritmo de *hash* sofreu um *downgrade* no qual o atacante conseguiu subverter os algoritmos utilizados pelo SPDM para que o *responder* tente utilizar um algoritmo MD5, que possui vulnerabilidades conhecidas em assinatura de certificados [Lenstra et al. 2005]. O atacante tentou negociar um algoritmo com segurança comprometida ao invés de outros algoritmos estáveis e conhecidos pela literatura, como o SHA3-512. Note que *libspdm_init_connection* retornou um código de erro (valor diferente de `0x0`) e o *boot* do sistema foi interrompido. A placa de rede, neste caso, não pode ser utilizada com autenticação ou comunicação segura, e esta informação é repassada para os demais componentes do ciclo de *boot* (e.g. *bootloader*, sistema operacional), que podem decidir se continuam o *boot* de forma insegura ou não. É possível executar o SPDM em qualquer fase, mas consideramos a BIOS como ponto crítico pois, caso com-

```
litex> spdm_requester
Memory used for rContext Initialzed
Address is 0x11001038.
Memory used for requester buffer (in bytes): 18432
Address scrath_buffer is 0x11006ae8.
LIBSPDM_DATA_SPDM_VERSION - 0x0
LIBSPDM_DATA_SECURED_MESSAGE_VERSION - 0x0
LIBSPDM_DATA_CAPABILITY_CT_EXPONENT - 0x0
LIBSPDM_DATA_CAPABILITY_FLAGS - 0x0
LIBSPDM_DATA_MEASUREMENT_SPEC - 0x0
LIBSPDM_DATA_BASE_ASYM_ALGO - 0x0
LIBSPDM_DATA_BASE_HASH_ALGO - 0x0
LIBSPDM_DATA_DHE_NAME_GROUP - 0x0
LIBSPDM_DATA_AEAD_CIPHER_SUITE - 0x0
LIBSPDM_DATA_REQ_BASE_ASYM_ALG - 0x0
LIBSPDM_DATA_KEY_SCHEDULE - 0x0
LIBSPDM_DATA_OTHER_PARAMS_SUPPORT - 0x0

Message size spdm_requester_send_message 5
SPDM Message from requester: 0x5 0x10 0x84 0x0 0x0

Message size spdm_requester_receive_message 9
SPDM Message from responder: 0x5 0x10 0x4 0x0 0x0 0x0 0x1 0x0 0x12
```

Figura 3. Pacote do SPDM em transmissão

prometida, compromete toda a cadeia de *boot*. Como mencionado anteriormente, a BIOS pode ser verificada externamente por um TPM caso o nível de segurança requeira.

```
Message size spdm_requester_send_message 49
SPDM Message from requester: 0x5 0x12 0xe3 0x4 0x0 0x30 0x0 0x1 0x2 0x90 0x0 0x0 0x0
1b 0x0 0x3 0x20 0x6 0x0 0x4 0x20 0xf 0x0 0x5 0x20 0x1 0x0

Message size spdm_requester_receive_message 5
SPDM Message from responder: 0x5 0x12 0x7f 0x1 0x0

libspdm_init_connection - 0x8001000a
Error, connection failed
```

Figura 4. Boot interrompido devido ao código de erro do SPDM

A implementação do SPDM em FPGA se encontra na modalidade código aberto. A documentação, manuais e o código-fonte estão disponíveis em `https://github.com/GustavsC/FPGA-SPDM`. O vídeo de configuração e demonstração está em `https://youtu.be/7Gst2-HgR14`. A demonstração no Salão de Ferramentas será o SPDM em execução pela BIOS e Ethernet na FPGA. Para a demonstração é necessário a FPGA NetFPGA-Sume (nota para como a placa recebe energia) e de um computador com os itens instalados: biblioteca LiteX 2024.04 e Vivado 2023.1, com uma licença para a Virtex-7.

4. Conclusão e trabalhos futuros

O SPDM é um protocolo desenvolvido para troca segura de mensagens entre dois pontos, garantindo que os dados trocados não sofreram algum tipo de modificação. Além da integridade e confidencialidade, o protocolo é capaz de autenticar um *firmware*, garantindo que não houve modificações através de assinaturas por certificados. No entanto, a utilização de protocolos deste tipo em nível de barramento não deve impor um gargalo de desempenho que inviabilize sua utilização em sistemas embarcados.

O presente trabalho demonstra uma implementação real em hardware do SPDM em um SoC materializado em um FPGA. A ferramenta apresentada é personalizável e mostramos um exemplo de implementação onde o software (BIOS) autentica o hardware (Ethernet). Esperamos que esta ferramenta sirva de ponto de partida para a implementação de SPDM em outros periféricos e para a análise de desempenho do protocolo em implementações reais. Para trabalhos futuros, consideramos expandir o protocolo para todos os periféricos e coletar métricas de consumo de energia e área de hardware, estimando assim o impacto da adição de segurança no nível de barramento.

Agradecimentos:

Este trabalho foi financiado em parte pela FAPESP (processo 2020/09850-0), CNPq (processo 307732/2023-1), e Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) – (Código de Financiamento 001).

Referências

Alves, R. C., Albertini, B. C., and Simplicio, M. A. (2022). Securing hard drives with the security protocol and data model (spdm). In *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 446–447. IEEE.

AMD (2024). Amd microblaze processor - a flexible and efficient soft processor. Acessado em Setembro/2024.

Asanović, K., Avizienis, R., Bachrach, J., Beamer, S., Biancolin, D., Celio, C., Cook, H., Dabbelt, D., Hauser, J., Izraelevitz, A., Karandikar, S., Keller, B., Kim, D., Koenig, J., Lee, Y., Love, E., Maas, M., Magyar, A., Mao, H., Moreto, M., Ou, A., Patterson, D. A., Richards, B., Schmidt, C., Twigg, S., Vo, H., and Waterman, A. (2016). The rocket chip generator. Technical Report UCB/EECS-2016-17, EECS Department, University of California, Berkeley.

Basnight, Z. H. (2013). Firmware counterfeiting and modification attacks on programmable logic controllers. Technical report, AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH GRADUATE SCHOOL OF ....

Belwafi, K., Shoufan, A., Alsafi, M., Ahmed, A., and Han, K. (2024). Hardware/software implementation of a chip-to-chip communication protocol based on spdm. *IEEE Access*, 12:194962–194972.

Brown, D., Walker, T. O., Blanco, J. A., Ives, R. W., Ngo, H. T., Shey, J., and Rakvic, R. (2021). Detecting firmware modification on solid state drives via current draw analysis. *Computers Security*, 102:102149.

Choi, B.-C., Lee, S.-H., Na, J.-C., and Lee, J.-H. (2016). Secure firmware validation and update for consumer devices in home networking. *IEEE Transactions on Consumer Electronics*, 62(1):39–44.

Cremers, C., Dax, A., and Naska, A. (2023). Formal analysis of spdm: security protocol and data model version 1.2. In *Proceedings of the 32nd USENIX Conference on Security Symposium*, SEC '23, USA. USENIX Association. Acessado em Julho/2024.

Cremers, C., Dax, A., and Naska, A. (2024). Breaking and provably restoring authentication: A formal analysis of SPDM 1.2 including cross-protocol attacks. Cryptology ePrint Archive, Paper 2024/2047. Acessado em Julho/2024.

Cui, A., Costello, M., and Stolfo, S. (2013). When firmware modifications attack: A case study of embedded exploitation.

Digilent (2024). Netfpga-sume (legacy). Acessado em Julho/2024.

DMTF (2023). Libspdm. Acessado em Maio/2025.

DTMF (2022). Security protocol and data model (spdm) specification. Acessado em Março/2025.

Hub, L. (2023). Linux on litex with a rv64gc rocketchip cpu. Acessado em Julho/2025.

Kermarrec, F., Bourdeauducq, S., Le Lann, J.-C., and Badier, H. (2020). Litex: an open-source soc builder and library based on migen python dsl. Acessado em Maio/2025.

Lenstra, A., Wang, X., and de Weger, B. (2005). Colliding x.509 certificates. Cryptology ePrint Archive, Paper 2005/067. Acessado em Julho/2025.

Mehta, D., Lu, H., Paradis, O. P., M. S., M. A., Rahman, M. T., Iskander, Y., Chawla, P., Woodard, D. L., Tehranipoor, M., and Asadizanjani, N. (2020). The big hack explained: Detection and prevention of pcb supply chain implants. *J. Emerg. Technol. Comput. Syst.*, 16(4). Acessado em Julho/2025.

Nvidia (2022). Nvidia/spdm: Implementation of the spdm protocol. Acessado em Março/2025.

RISC-V (2020). opensbi. Acessado em Abril/2025.

riscv collab (2023). Risc-v gnu compiler toolchain. Acessado em Abril/2024.

SiFive (2017). The -march, -mabi, and -mtune arguments to risc-v compilers. Acessado em Maio/2024.

Wang, X., Konstantinou, C., Maniatakos, M., and Karri, R. (2015). Confirm: Detecting firmware modifications in embedded systems using hardware performance counters. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 544–551. IEEE.

Zhang, Y., Li, Y., and Li, Z. (2023). Aye: A trusted forensic method for firmware tampering attacks. *Symmetry*, 15(1).