Cultivating Security: Debug Authentication for Ensuring the Security of SoC's Root-of-Trust

Arash Vafaei, Sujan Kumar Saha, Mark Tehranipoor, and **Farimah Farahmandi**
ECE Department, University of Florida

Outline

1 **Motivation and Background**

- Hardware Assisted Debug
- Attacks using Debug Infrastructure
- Secure Debug Requirements
- Threat Model

2 **Proposed Secure Debug**

- Certificate-Based Authentication
- System-on-Chip with Security Engine
- Secure Debug Architecture
- Multi-Level Privilege Model

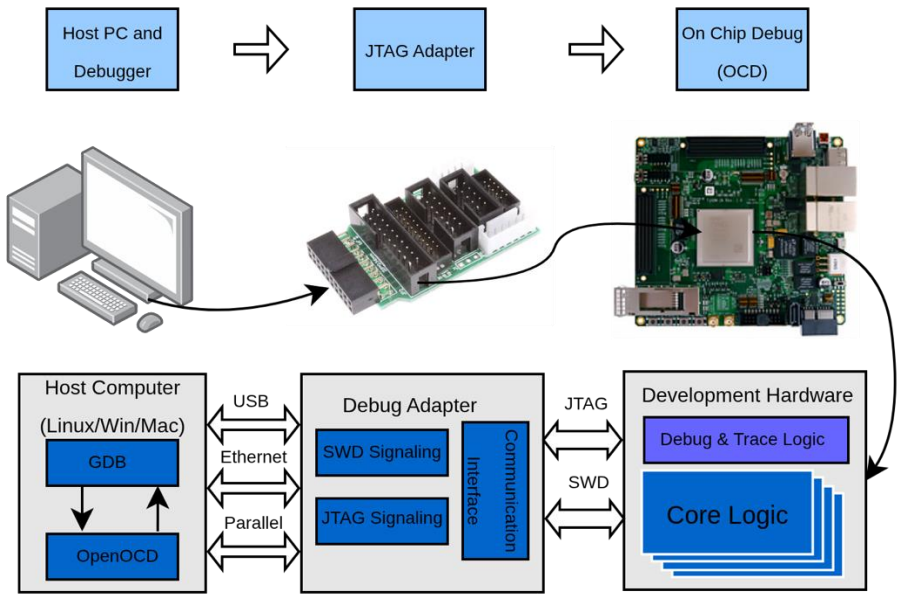
3 **RISC-V Secure Debug Implementation**

- Secure Debug Implementation
- Debug Register Protection Unit

4 **Conclusion**

Hardware-assisted Debug

- Hardware-assisted debugging allows engineers to observe internal behaviors of SoC in real time.
- JTAG and Scan Chain allow direct access to IP internal signals, enable arbitrary execution, permit full memory extraction.
- Enhance visibility of the system for efficient debugging.
- So far So Good..



Overview of Hardware-assisted Debug Infrastructure

But these introduce new attack surface.

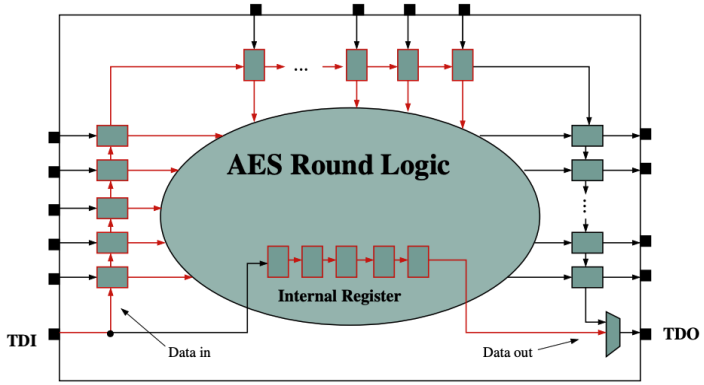
Attacks using Debug Infrastructure

Fact

Debug infrastructures are misused by attackers to leak encryption keys, tamper memory content, inject malicious payload.

❑ Examples:

- AES and RSA modules are vulnerable to scan-chain-based attacks [1,2].
- Attackers pause execution and inject instructions that can result in privilege escalation or unauthorized memory access in ARM infrastructure [3].
- Attackers extract firmware for reverse engineering or patch compromised firmware or bypass security checks [4].



AES data leaked through Scan registers [1]

Existing Solutions

❑ Encrypting Data

- Encrypting debug data path (scan chain data) using stream or block ciphers to protect data in transit [5,6].

❑ Debugger Authentication

- Password based debug control [7] lacks strong identity binding.
- PUF-based authentication methods allows binding debuggers to unique hardware fingerprints [8].

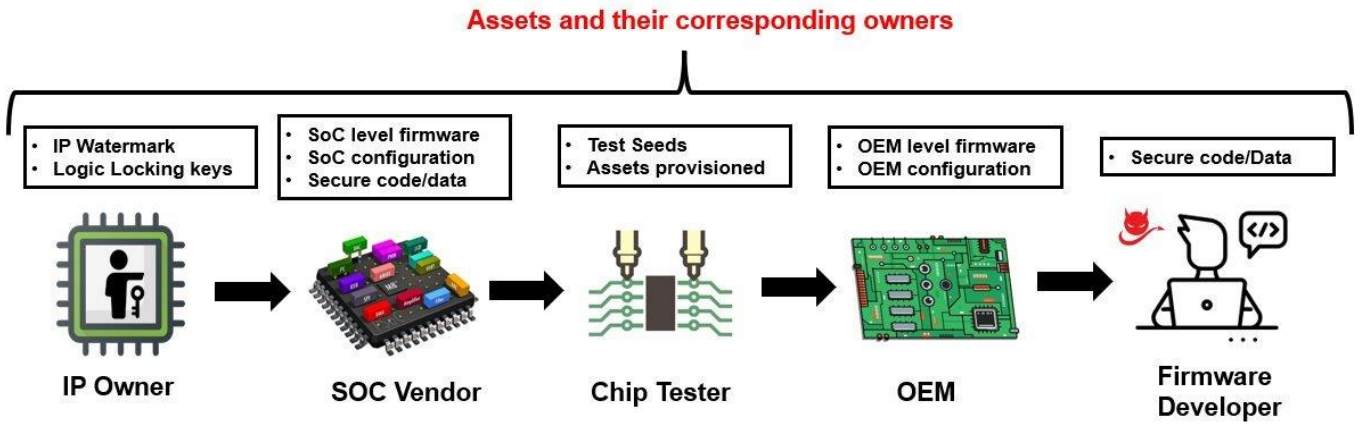
❑ Debug Interface Disable

- Permanently disable debug interface after manufacturing and testing using e-fuses [3], hence not possible in-field debugging.

The methods come with trade-offs in area, performance, flexibility and new vulnerabilities..!!

Secure Debug requirements

- **Requirements:**
  - SoCs have assets at different life-cycle stages.
  - Different stakeholders such as test engineers, field service providers, security analyst etc. require varying levels of debug access.
  - Mutual authentication protocols are required.
  - Certificate based authentication is a must.
  - Multiple privilege levels are needed for users on different operational domains.



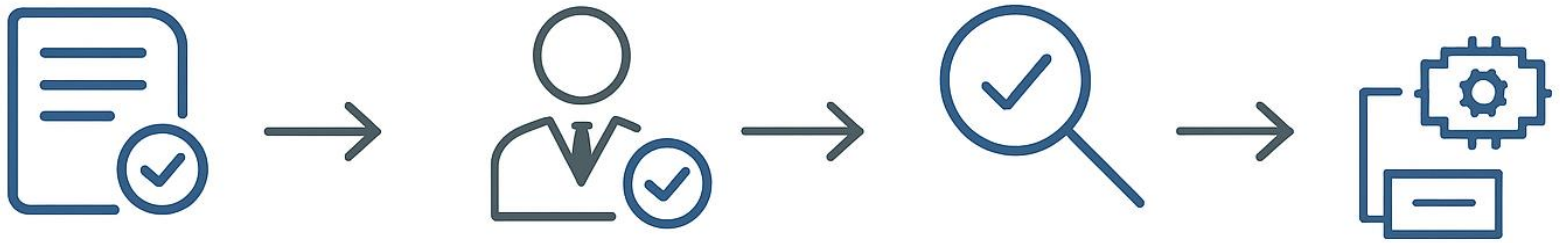
Various entities using debug infrastructure and the attack targets (assets) at each stage

Threat Model

- Non-secure developers may access secure programs or data even in test and debug mode.
- Debuggers may access secure data without proper access control.
- Remote debuggers can transmit private code and configurations to potentially unauthorized SoCs.
- Our threat model considers both developers and debuggers perspective.

Certificate-Based Authentication

Certificate-Based Authentication Method



Certificate Structure

- Contains public key, issuer ID, and digital signature
- Includes custom fields (e.g., SoC ID, debugger privilege level)

Signature Generation

- Hash certificate content (excluding signature field)
- Encrypt hash using authority's private signing key
- Attach signature to certificate

Validation Process

- Recompute hash from received certificate
- Decrypt signature using authority's public key
- Compare hashes → if matched, certificate is authentic

Authentication Outcome

- Verifies identities of both debugger and SoC
- Prevents impersonation and tampering in debug framework

Secure SoC Architecture with Security Engine

❖ Purpose & Role

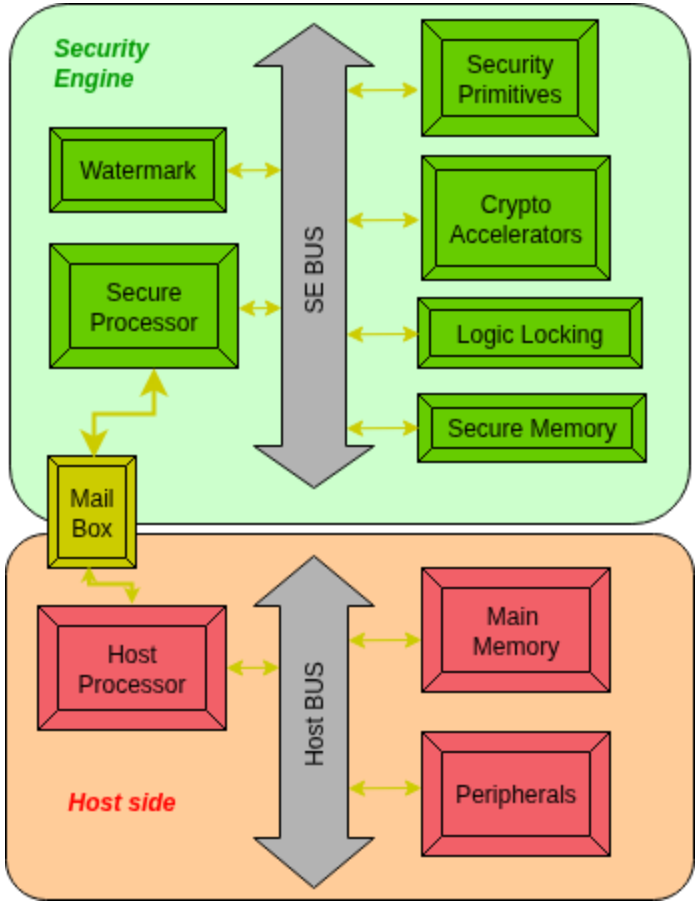
- Provides hardware-based process isolation to contain attacks and protect critical assets
- Ensures confidentiality, integrity, and authenticity even if OS/hypervisor is compromised
- Operates as an isolated hardware environment, retaining assets or offering secure services (e.g., encrypted transfers)

❖ Examples

- Intel SGX: Enclave isolation & attestation
- ARM TrustZone: Secure/normal world separation
- RISC-V Keystone: Memory encryption & remote attestation
- Extended with supply-chain protections (watermarking, logic locking, secure boot)

❖ Vulnerabilities

- Debug port links secure & non-secure domains → potential attack surface
- Disabling debug or weak authentication = impractical / insecure
- Highlights need for secure debug mechanisms in SE-equipped SoCs



System-on-Chip with Security Engine

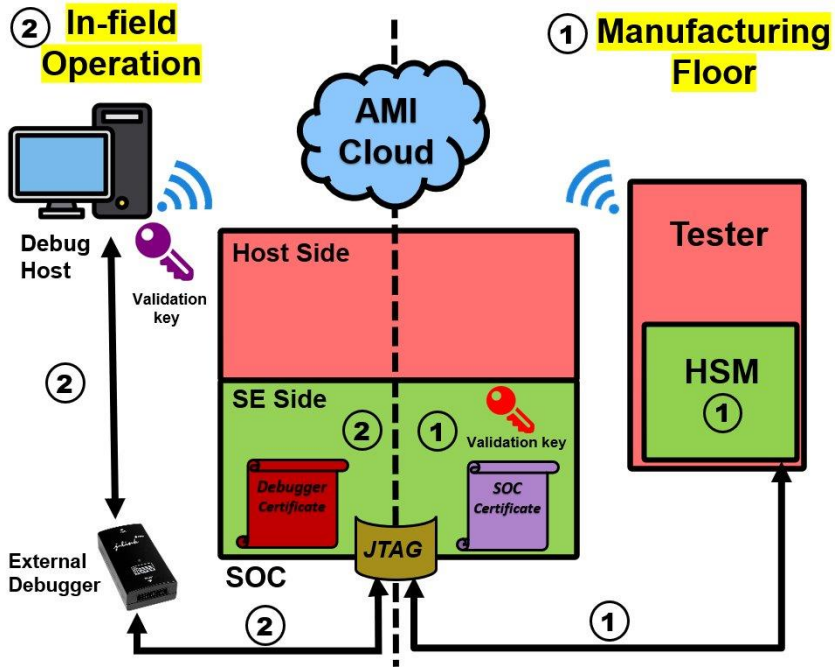
Asset Provisioning and Security Engine

1. Manufacturing Phase

- SoC-specific assets are provisioned:
  - Debugger certificate validation key
  - SoC certificate
- Hardware Secure Module (HSM) enables secure provisioning and trust establishment between Asset Management Infrastructure (AMI) and untrusted tester
- AMI (Certificate Authority):
  - Generates key pairs & SoC-specific certificates
  - Maintains database indexed by SoC IDs
  - Each SoC securely stores keys & certificate in tamper-proof memory

2. In-Field Debug Phase

- Debug host contacts AMI to request a certificate for specific SoC ID
- Host authenticates via standard internet protocols
- AMI signs a debugger certificate using SoC's private key, embedding:
  - Debug privilege level
  - Public encryption key for secure communication
- Security Engine (SE) in the SoC validates the debugger certificate using pre-provisioned validation key



Asset provisioning of SoC with AMI, Debugger and Tester

Outcome: Enables authenticated and secure debug access between the SoC and external debugger

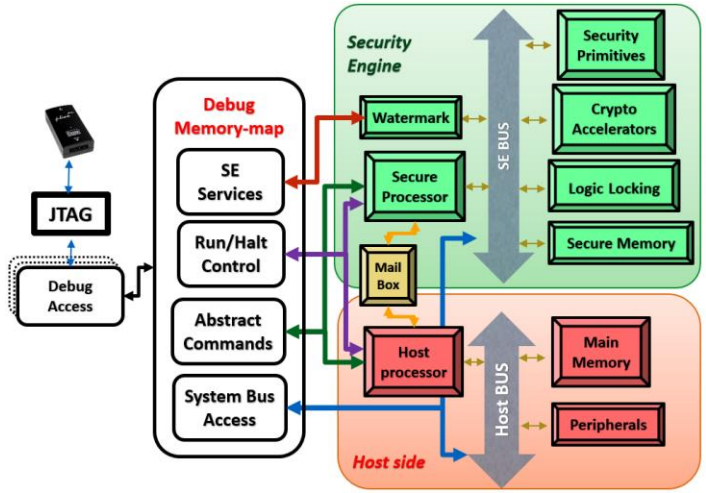
Secure Debug

Overview

- Defines a privilege-based debug access model independent of ISA
- Ensures isolation and secure access between the Security Engine (SE) and host SoC
- Enables secure debugging while minimizing architectural changes

Debug Architecture

- Debug access via memory-mapped interfaces routed to SE modules through debug port registers
- Allows trusted debug access for secure firmware development and IP verification
- Host-side debug covers multi-core processors, buses, and memory hierarchies — protected by SE-based encryption and memory protection



Debug platform services and connections in the presence of a security engine in SoC

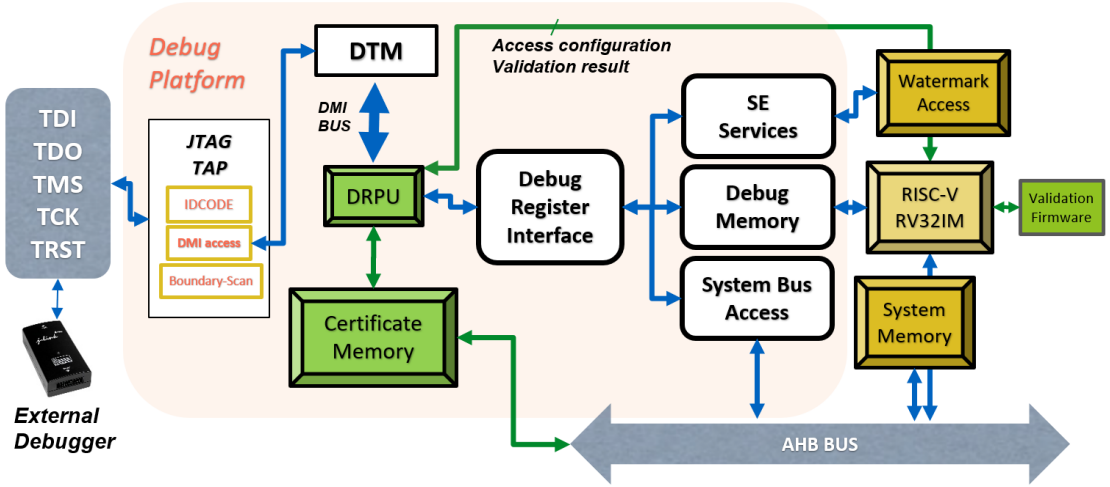
Multi-Level Privilege Model for Secure Debug

Level	Access Privilege & Purpose
Level 0	Reserved for IP owners; allows secure IP watermark extraction or key validation without host involvement
Level 1	Full SE access for SoC owners and secure firmware developers; can run code and access SE registers (except IP-owner services)
Level 2	Restricted SE access; disables run/halt, abstract commands, and SE bus; allows only secure mailbox-based communication
Level 3	Complete SE isolation; mailbox disabled to block SE communication during vulnerabilities; for general developers
Level 4	Host-specific restrictions; isolates SE, but may limit or allow bus/memory access per SoC architecture

Key Takeaway:
- Defines minimum privilege zones for safe debug access in SE-integrated SoCs
- Supports fine-grained policy enforcement for secure and flexible debug operations

Secure Debug Implementation

- Flexible, feature-rich interface for **hardware-assisted debugging** of RISC-V processors
- Supports **independent hart control**, register/CSR access, and **memory read/write**
- **Debug initialization** from the first instruction via debug reset interface
- Core features: **breakpoints**, **single-stepping**, **custom instruction execution**, and **non-intrusive register access**
- **Debug path:** JTAG → Debug Transport Module (DTM) → Debug Register Protection Unit (DRPU)→ Debug Module Interface (DMI) → Debug Module (DM)
- **DM functions:** run/halt control, abstract commands, and system bus access through register interface
- **Key registers:**
  - `dmcontrol` → hart selection
  - `command`, `data0–11`→ instruction/data exchange
  - `sbaddress`, `sbdata` → system bus interface
- Centralized register access enables fine-grained monitoring and privilege-based access control



RISC-V debug platform and with its connection to a simplified RISC-V SoC with a 32-bit core and a small memory for storing firmware

Debug Register Protection Unit

Key Functions:

1. Certificate Channel Support:
   - Adds new authentication opcode to DMI read/write path
   - Loads debugger certificate via JTAG into certificate buffer
   - Manages access through address decoding
2. Access Control Enforcement:
   - Secure firmware configures permissions post-certificate validation
   - Enforces restrictions during normal debug operations
3. Status Reporting:
   - Provides real-time authentication and debug state feedback

Prototype Implementation:

- Based on CVA6 RISC-V core (RV32IM)
- Policies enforced:
  - Block SE-related hart access
  - Restrict SE-protected bus regions
  - Prevent external access to watermark registers

TABLE I Power and Area Overhead of Secure Debug IPs

	Secure Debug	RISC-V (CVA6)	RISC-V Debug	Overhead for Debug	Overall Overhead
Area (mm^2)	2608.24	245871.38	49440.84	5%	0.8%
Dynamic Power (uW)	311.19	8861.1	2311.8	13%	2%

Conclusion

- **Challenge:** Debug interfaces are vital but expose serious security risks in open SoC architectures (e.g., RISC-V).
- **Proposed Solution:** Layered framework with
  - Certificate-based authentication
  - Fine-grained privilege control
  - Debug Register Protection Unit (DRPU) integrated into RISC-V debug path
- **Key Strength:** Enforces secure access without modifying the RISC-V debug specification
- **Results:** Hardware prototype shows feasible, low-overhead security enforcement in real SoC environments

References

1. Ali, Sk Subidh, Ozgur Sinanoglu, and Ramesh Karri. "Test-mode-only scan attack using the boundary scan chain." In 2014 19th IEEE European test symposium (ETS), pp. 1-6. IEEE, 2014.
2. Wang, Weizheng, Xiangqi Wang, Jin Wang, Neal N. Xiong, Shuo Cai, and Peng Liu. "Ensuring cryptography chips security by preventing scan-based side-channel attacks with improved DFT architecture." IEEE Transactions on Systems, Man, and Cybernetics: Systems 52, no. 3 (2020): 2009-2023.
3. Ning, Zhenyu, and Fengwei Zhang. "Understanding the security of arm debugging features." In 2019 IEEE Symposium on Security and Privacy (SP), pp. 602-619. IEEE, 2019.
4. Vasile, Sebastian, David Oswald, and Tom Chothia. "Breaking all the things—A systematic survey of firmware extraction techniques for IoT devices." In International Conference on Smart Card Research and Advanced Applications, pp. 171-185. Cham: Springer International Publishing, 2018.
5. Valea, Emanuele, Mathieu Da Silva, Giorgio Di Natale, Marie-Lise Flottes, and Bruno Rouzeyre. "A survey on security threats and countermeasures in IEEE test standards." IEEE Design & Test 36, no. 3 (2019): 95-116.
6. Valea, Emanuele, Mathieu Da Silva, Marie-Lise Flottes, Giorgio Di Natale, and Bruno Rouzeyre. "Encryption-based secure JTAG." In 2019 IEEE 22nd International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), pp. 1-6. IEEE, 2019.
7. Lee, Kuen-Jong, Zheng-Yao Lu, and Shih-Chun Yeh. "A secure JTAG wrapper for SoC testing and debugging." IEEE Access 10 (2022): 37603-37612.
8. Kumar, Sudeendra, Saurabh Seth, Sauvagya Sahoo, Abhishek Mahapatra, Ayas Kanta Swain, and Kamalakanta Mahapatra. "PUF-based secure test wrapper for SoC testing." In 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 672-677. IEEE, 2018.

Q & A