2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION

UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 24-27, 2025

Guardians of the Chip: Mastering Next-Gen Security for SoCs and IPs

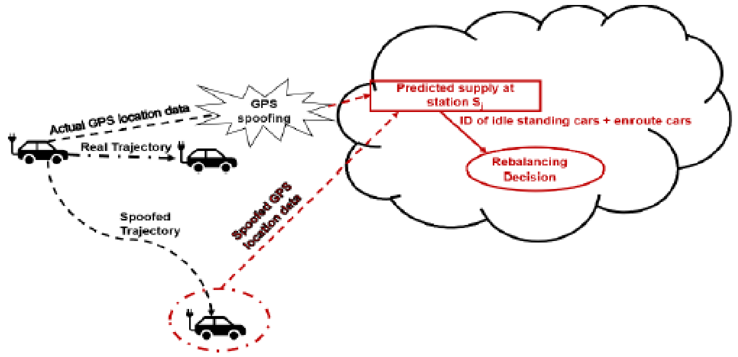
Jagata Sridevi, Cadence Design Systems, India
Vishnu Prasad K V, Cadence Design Systems, India
Deep Mehta, Cadence Design Systems, India

accellera SYSTEMS INITIATIVE

The Silent Security Flaw: The Error That No One Sees—Until It's Too Late

What Happens When an Invisible Error Gets Through?

- AI models fail to predict correctly
- Cloud workloads corrupt sensitive data
- Autonomous vehicles misinterpret surroundings.


Actual GPS location data
GPS spoofing
Real Trajectory
Spoofed Trajectory
Spoofed GPS location data
Predicted supply at station S_j
ID of idle standing cars + enroute cars
Rebalancing Decision

The Problem:

- A **single verification blind spot** can break encryption, corrupt transactions, and disrupt entire systems

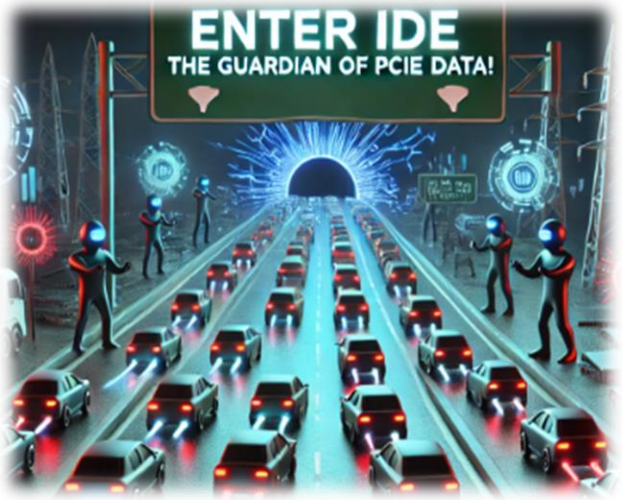
"How do we verify that security flaws in high-speed SoCs and IPs are caught before deployment?"

What Verification Engineers Can Take Away

This Presentation at a Glance

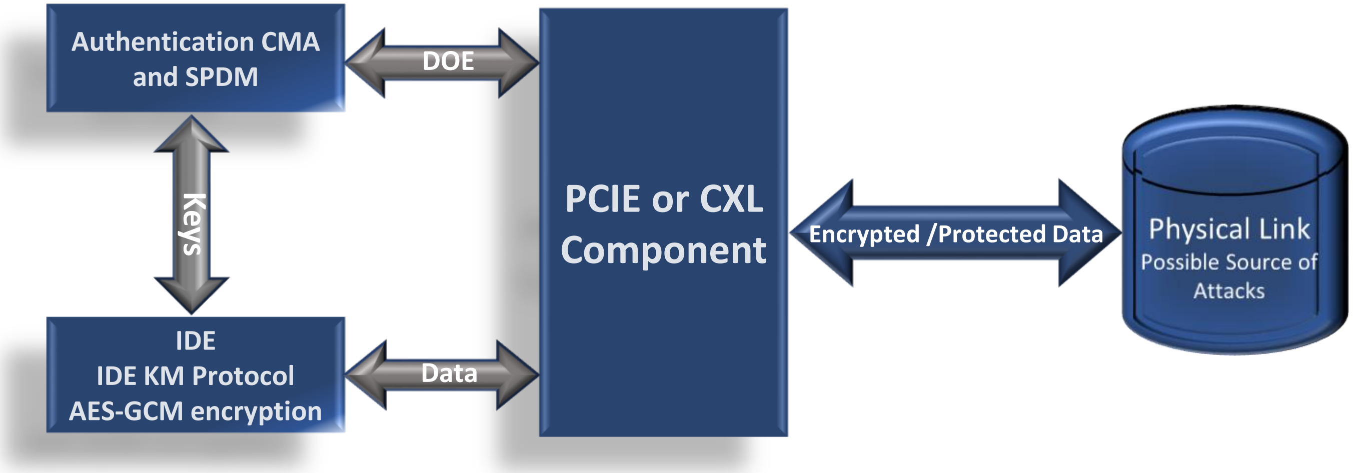
- Unveiling **critical blind spots in high-speed protocol verification**
- Introducing **five techniques** that enhance security & debug efficiency
- Learning how these methods **bridge gaps in security verification**

- **What You'll Take Away Today**?
    - A fresh way to **identify and fix hidden verification blind spots.**
    - These techniques **can be applied to other high-speed protocols** and SoC/IP verification.
    - Smarter debugging, stronger security, and **a higher verification confidence level.**
        - By recognizing and addressing **hidden blind spots**, engineers can improve **design robustness** and **accelerate debugging**



What is IDE in PCIe?

- IDE is a security feature in PCIe that ensures **data integrity** and **confidentiality** over the PCIe link for TLP’s.
  - Protect data from **tampering** (integrity).
  - Prevent **unauthorized access** (encryption)



PCIe/CXL with IDE and Authentication??

- Physical Link as a Possible Source of Attacks and
- The PCIe or CXL component, handles encrypted data transmission.
- Uses AES-GCM for data confidentiality.
- Key Management: Handles secure key generation, exchange, and refresh.
- Using CMA and SPDM, we establish a secure communication channel and Prevents unauthorized access.
- DOE to exchange security-related messages, including authentication and key exchange processes.

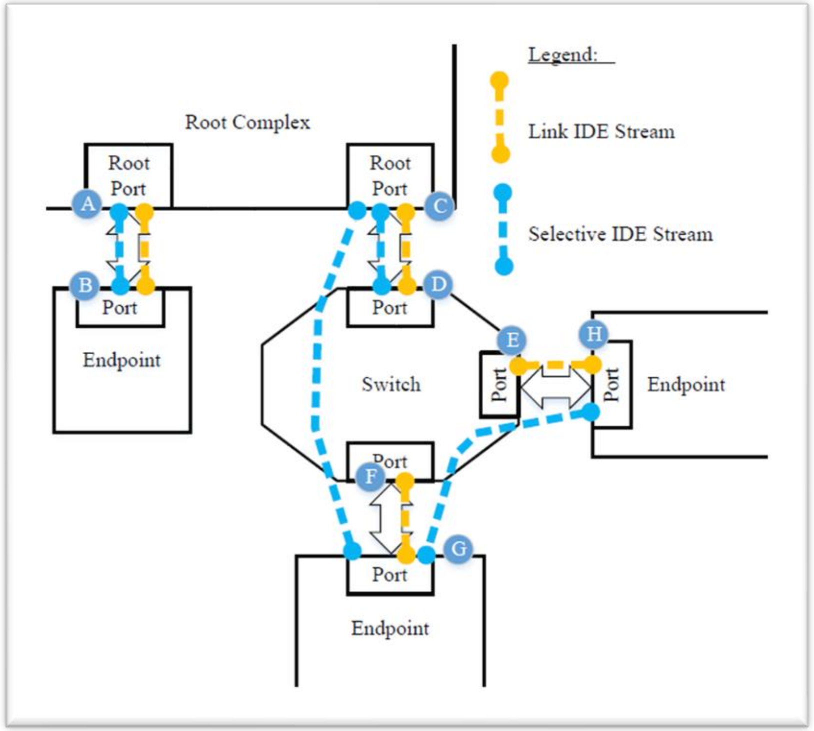
IDE Streams and Data Flow

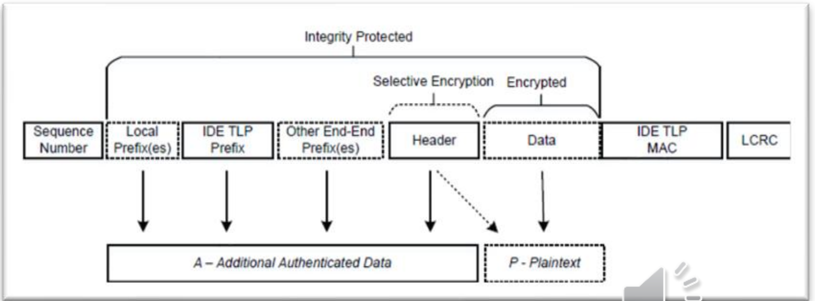
Link IDE Stream (Yellow)

- Encrypts all traffic over a PCIe/CXL link,
- This approach **adds encryption overhead**, which may impact performance

Selective IDE Stream (Blue)

- Secures only specific traffic (e.g., based on address or transaction type).
- This helps balance performance and security, allowing non-sensitive data to flow unencrypted.
- More flexible, as encryption is applied only to selected streams, optimizing performance.



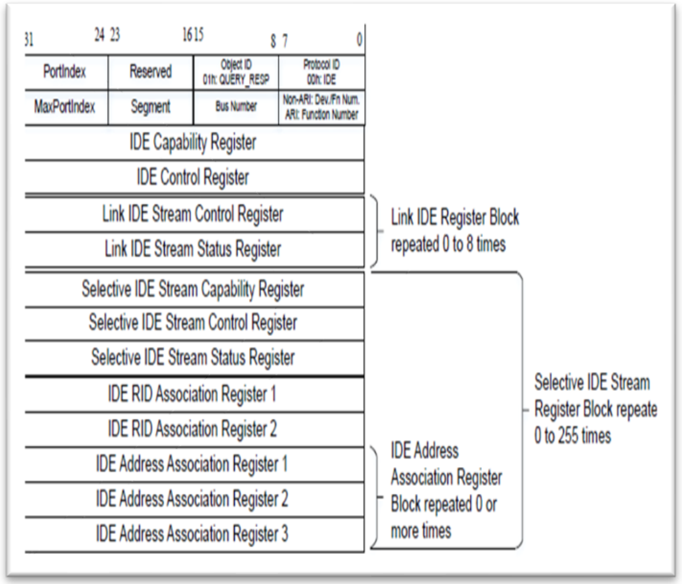


IDE Capability Discovery & Register Structure

How is IDE Capability Advertised and Discovered?

- Devices supporting IDE **announce their capability** in **PCIe/CXL Configuration Space**.
- The **IDE register structure includes repeating blocks**, which allow
  - **Link IDE Register Blocks** - **8 times**
  - **Selective IDE Stream Register Blocks** - **255 times**
  - **IDE Address Association Blocks** to be repeated **as needed** (allowing dynamic mapping of encryption).

Hosts can dynamically associate addresses or RIDs with IDE for flexibility and efficiency.


31
24 23
16 15
8 7
0
PortIndex
Reserved
Object ID 01h: QUERY_RESP
Protocol ID 00h: IDE
MaxPortIndex
Segment
Bus Number
Non-ARI: Dev./Fn Num. ARI: Function Number
IDE Capability Register
IDE Control Register
Link IDE Stream Control Register
Link IDE Stream Status Register
Link IDE Register Block repeated 0 to 8 times
Selective IDE Stream Capability Register
Selective IDE Stream Control Register
Selective IDE Stream Status Register
IDE RID Association Register 1
IDE RID Association Register 2
IDE Address Association Register 1
IDE Address Association Register 2
IDE Address Association Register 3
IDE Address Association Register Block repeated 0 or more times
Selective IDE Stream Register Block repeate 0 to 255 times

Layered IDE Security Framework and Techniques

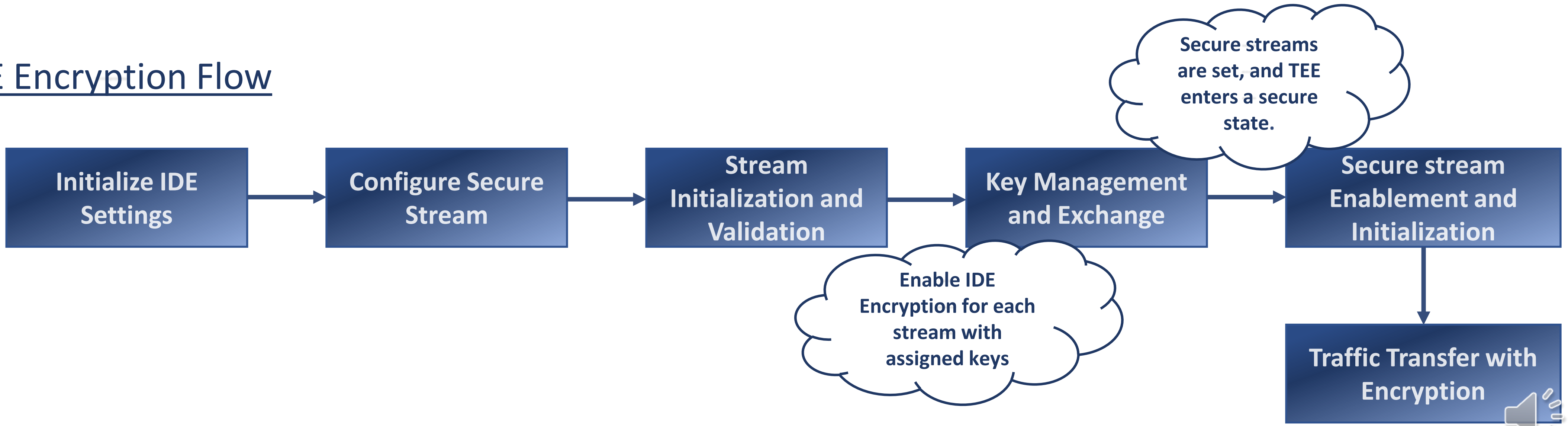
Layered IDE Packet Security Framework

Initialize IDE Settings and Configure Secure Stream – Establish foundational security parameters and Define conditions for secure data exchange.

Key Management and Exchange – Establish, refresh, and distribute encryption keys securely

Secure Stream Enablement and Initialization –TEE enters a secure state

IDE Encryption Flow

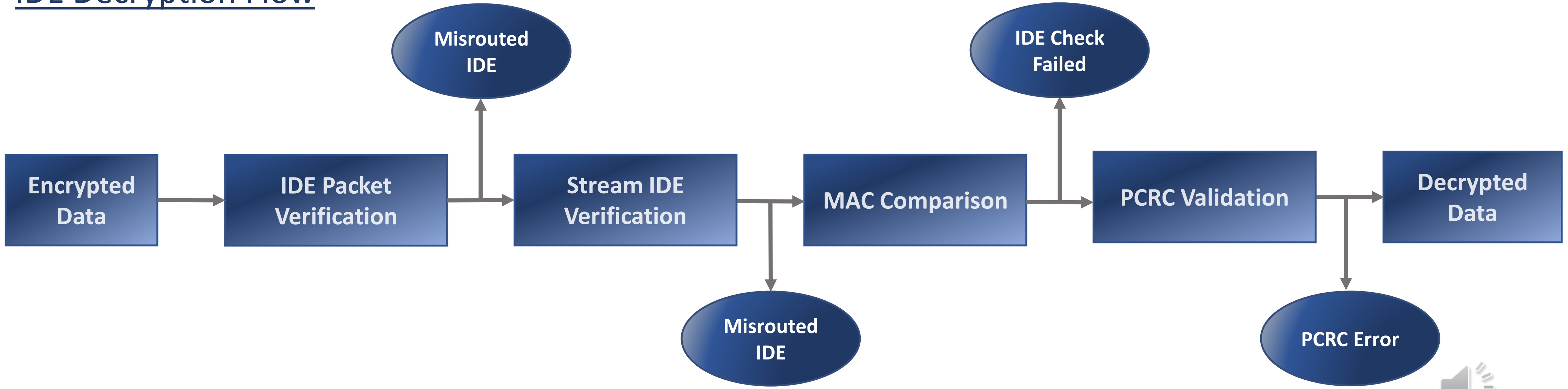

Initialize IDE Settings
Configure Secure Stream
Stream Initialization and Validation
Key Management and Exchange
Secure stream Enablement and Initialization
Traffic Transfer with Encryption
Secure streams are set, and TEE enters a secure state.
Enable IDE Encryption for each stream with assigned keys

Layered IDE Packet Security Framework(Contd.)

IDE Packet Verification – Determines whether the packet is IDE encrypted.

Stream Verification, PCRC and MAC – Ensures the packet belongs to an authenticated stream and Checks data integrity using Message Authentication Code.

IDE Decryption Flow



1)State Machine for IDE Capability Discovery

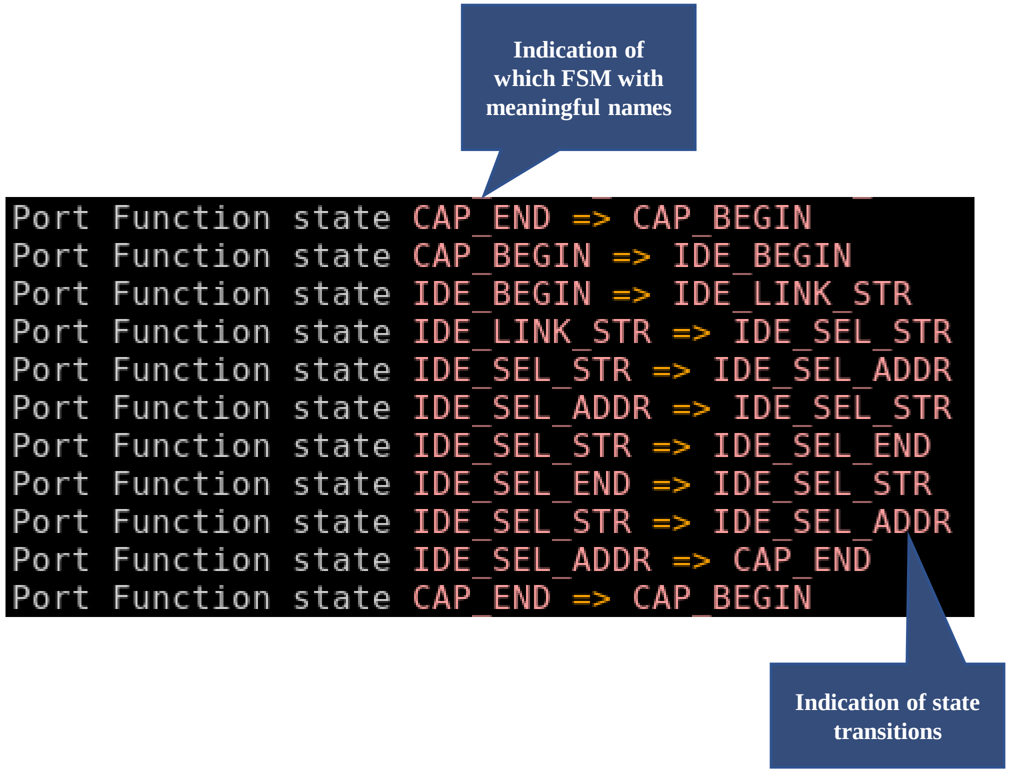
A structured **Finite State Machine (FSM)** approach to streamline **IDE capability discovery** in complex configurations like link-based and selective streams.

☑ **Structured & Predictable** – Organizes IDE discovery into distinct states.

☑ **Automation & Efficiency** – FSM automates IDE checks, reducing manual intervention.

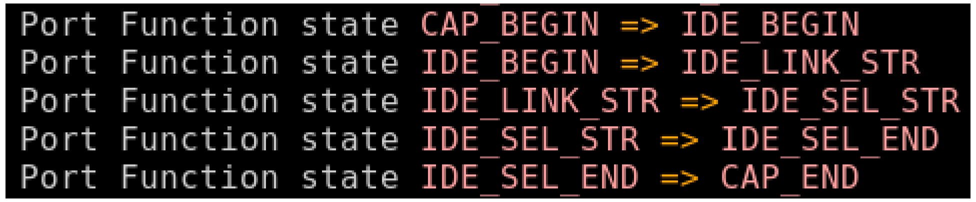
☑ **Improved Verification Accuracy** – Ensures compliance with the IDE specification.

☑ **Enhanced Team Alignment** – Provides a clear framework for verification teams.


Indication of which FSM with meaningful names
Indication of state transitions

```
Port Function state CAP_END => CAP_BEGIN
Port Function state CAP_BEGIN => IDE_BEGIN
Port Function state IDE_BEGIN => IDE_LINK_STR
Port Function state IDE_LINK_STR => IDE_SEL_STR
Port Function state IDE_SEL_STR => IDE_SEL_ADDR
Port Function state IDE_SEL_ADDR => IDE_SEL_STR
Port Function state IDE_SEL_STR => IDE_SEL_END
Port Function state IDE_SEL_END => IDE_SEL_STR
Port Function state IDE_SEL_STR => IDE_SEL_ADDR
Port Function state IDE_SEL_ADDR => CAP_END
Port Function state CAP_END => CAP_BEGIN
```

State Machine for IDE Capability Discovery(Contd.)



```
Port Function state CAP_BEGIN => IDE_BEGIN
Port Function state IDE_BEGIN => IDE_LINK_STR
Port Function state IDE_LINK_STR => IDE_SEL_STR
Port Function state IDE_SEL_STR => IDE_SEL_END
Port Function state IDE_SEL_END => CAP_END
```

Capability Discovery Process

- The FSM moves through **CAP_BEGIN → IDE_BEGIN → IDE_LINK_STR → IDE_SEL_STR → IDE_SEL_ADDR** and so on, meaning it's discovering what IDE capabilities exist for the given port or function.
- **IDE_SEL_STR (Selective Stream Mode) and IDE_SEL_ADDR (Address-Based Selection Mode)** appear multiple times, suggesting that the FSM is dynamically evaluating different modes.

Looping Nature of the FSM

- The FSM loops back from CAP_END to CAP_BEGIN, indicating that capability discovery for control, status and Address association blocks is a recurring process rather than a one-time setup.
- Transitioning back to CAP_END suggests that once all possible configurations are evaluated, the FSM finalizes the IDE settings.

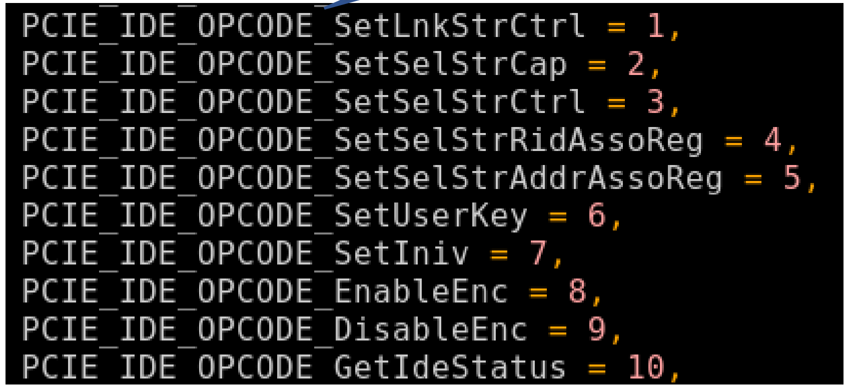
2)Operation Codes in IDE Context

Issue: Multiple APIs for IDE operations → **Inefficiency & fragmentation**

Solution: Unified OpCode-Based API

- ☑**Simplified API** – One structure for all operations
- ☑**Scalable & Flexible** – Easy future expansion
- ☑**Faster Debugging** – OpCodes streamline testing
- ☑**Improved Readability** – Clear & structured approach

Standard Prefix and meaningful for all OPCODE's

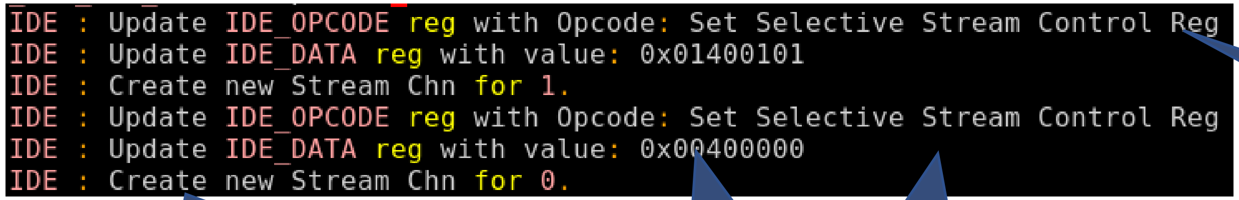


```
PCIE_IDE_OPCODE_SetLnkStrCtrl = 1,
PCIE_IDE_OPCODE_SetSelStrCap = 2,
PCIE_IDE_OPCODE_SetSelStrCtrl = 3,
PCIE_IDE_OPCODE_SetSelStrRidAssoReg = 4,
PCIE_IDE_OPCODE_SetSelStrAddrAssoReg = 5,
PCIE_IDE_OPCODE_SetUserKey = 6,
PCIE_IDE_OPCODE_SetIniv = 7,
PCIE_IDE_OPCODE_EnableEnc = 8,
PCIE_IDE_OPCODE_DisableEnc = 9,
PCIE_IDE_OPCODE_GetIdeStatus = 10,
```

- OpCodes mapped to specific IDE operations.
- Each operation has a unique identifier prefixed with "PCIE_IDE_OPCODE_"
- Instead of using multiple APIs, each function is assigned a unique opcode for streamlined execution.

Operation Codes in IDE Context (Contd.)

- The below execution shows **how the defined OpCodes are being used in real operations**.
- The system updates different registers like IDE_OPCODE reg, IDE_DATA reg using OpCodes.
- Also, opcodes are used dynamically at runtime to manage IDE operations efficiently.
- Data Written to the Opcode indicates that values are being assigned to registers based opcode operations.



```
IDE : Update IDE_OPCODE reg with Opcode: Set Selective Stream Control Reg
IDE : Update IDE_DATA reg with value: 0x01400101
IDE : Create new Stream Chn for 1.
IDE : Update IDE_OPCODE reg with Opcode: Set Selective Stream Control Reg
IDE : Update IDE_DATA reg with value: 0x00400000
IDE : Create new Stream Chn for 0.
```

sets the Selective Stream Control Register

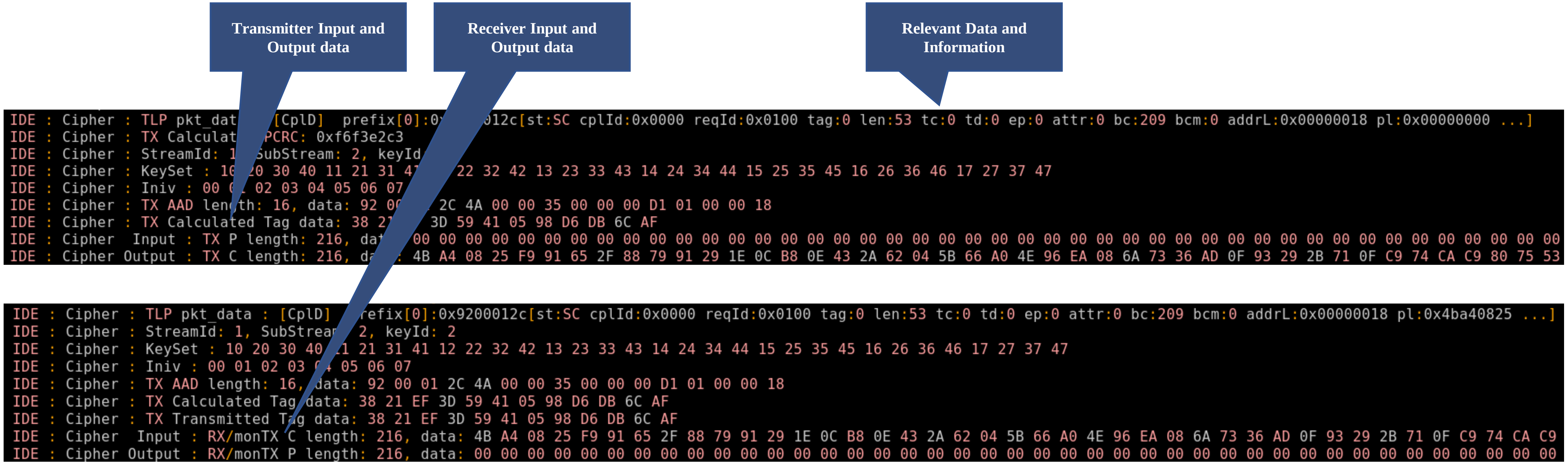
Indicates the creation of a new IDE stream

Data Written to the Opcode

Operation Performed via OPCODE

3)White Box Logging

- The below execution shows **how the defined OpCodes are being used in real operations**.


Transmitter Input and Output data
Receiver Input and Output data
Relevant Data and Information

```
IDE : Cipher : TLP pkt_dat   [CplD]  prefix[0]:0x    012c[st:SC cplId:0x0000 reqId:0x0100 tag:0 len:53 tc:0 td:0 ep:0 attr:0 bc:209 bcm:0 addrL:0x00000018 pl:0x00000000 ...]
IDE : Cipher : TX Calculat   PCRC: 0xf6f3e2c3
IDE : Cipher : StreamId: 1  SubStream: 2, keyId
IDE : Cipher : KeySet : 10 20 30 40 11 21 31 4    22 32 42 13 23 33 43 14 24 34 44 15 25 35 45 16 26 36 46 17 27 37 47
IDE : Cipher : Iniv : 00 0  02 03 04 05 06 07
IDE : Cipher : TX AAD length: 16, data: 92 00    2C 4A 00 00 35 00 00 00 D1 01 00 00 18
IDE : Cipher : TX Calculated Tag data: 38 21    3D 59 41 05 98 D6 DB 6C AF
IDE : Cipher  Input : TX P length: 216, dat   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
IDE : Cipher Output : TX C length: 216, da    4B A4 08 25 F9 91 65 2F 88 79 91 29 1E 0C B8 0E 43 2A 62 04 5B 66 A0 4E 96 EA 08 6A 73 36 AD 0F 93 29 2B 71 0F C9 74 CA C9 80 75 53
```

```
IDE : Cipher : TLP pkt_data : [CplD]   refix[0]:0x9200012c[st:SC cplId:0x0000 reqId:0x0100 tag:0 len:53 tc:0 td:0 ep:0 attr:0 bc:209 bcm:0 addrL:0x00000018 pl:0x4ba40825 ...]
IDE : Cipher : StreamId: 1, SubStrea   2, keyId: 2
IDE : Cipher : KeySet : 10 20 30 40  1 21 31 41 12 22 32 42 13 23 33 43 14 24 34 44 15 25 35 45 16 26 36 46 17 27 37 47
IDE : Cipher : Iniv : 00 01 02 03 0  05 06 07
IDE : Cipher : TX AAD length: 16, data: 92 00 01 2C 4A 00 00 35 00 00 00 D1 01 00 00 18
IDE : Cipher : TX Calculated Tag data: 38 21 EF 3D 59 41 05 98 D6 DB 6C AF
IDE : Cipher : TX Transmitted Tag data: 38 21 EF 3D 59 41 05 98 D6 DB 6C AF
IDE : Cipher  Input : RX/monTX C length: 216, data: 4B A4 08 25 F9 91 65 2F 88 79 91 29 1E 0C B8 0E 43 2A 62 04 5B 66 A0 4E 96 EA 08 6A 73 36 AD 0F 93 29 2B 71 0F C9 74 CA C9
IDE : Cipher Output : RX/monTX P length: 216, data: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

White Box Logging (Contd.)

✗Limitations of Black Box Logging Only captures inputs and outputs without revealing what happens internally, So debugging is difficult as there's no visibility into intermediate steps

👍White Box Logging is an advanced debugging approach that provides deep visibility into internal operations

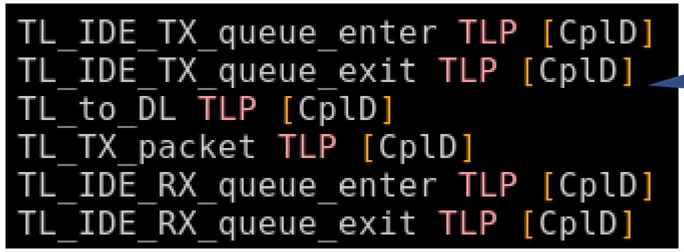
- ☑**Faster Debugging** – Trace security & performance issues easily
- ☑ **Data Integrity Validation** – Ensures correctness of transformations
- ☑ **Reliable Tracking** – Essential for high-security applications
- ☑ **Enhanced Error Detection** – Combine with protocol checks for compliance

White Box Logging ensures full transparency, making security debugging faster, accurate, and reliable!

4) Modular Debug Hooks

Modular debug hooks provide structured **traceability and transparency** within **layered architectures** like **PCIe and CXL**, enhancing **debug efficiency** while maintaining minimal overhead.

- Captures transactions at **entry & exit**
- Logs movement from **TL to DL layer**.
- Simulates issues like **corrupted TLPs & misrouted data**.
- Reduces **manual log inspection** by correlating transaction points.



```
TL_IDE_TX_queue_enter TLP [CplD]
TL_IDE_TX_queue_exit TLP [CplD]
TL_to_DL TLP [CplD]
TL_TX_packet TLP [CplD]
TL_IDE_RX_queue_enter TLP [CplD]
TL_IDE_RX_queue_exit TLP [CplD]
```

IDE Layer callback for Error Injection or can be used as Debug hooks

5) Flexible Verification Methods for IDE Functionality

This technique provides alternative verification approaches when the **DUT lacks full IDE support**, such as authentication and key management.

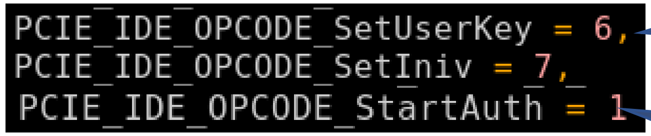
Authentication Bypass (Front-Door Approach)

- VIP **initiates authentication independently** using a **front-door opcode mechanism**.

Backdoor Key Management (KM Bypass)

- Directly **injects IDE keys** into the environment, bypassing standard KM protocols.

Allows full IDE testing even with **partial feature availability, and** Works **independently of DUT's IDE maturity**, accelerating verification.



```
PCIE_IDE_OPCODE_SetUserKey = 6,
PCIE_IDE_OPCODE_SetIniv = 7,
PCIE_IDE_OPCODE_StartAuth = 1
```

We can start front-door authentication process

We can set user key to start KM process

(a) Handling Invalid Routing ID and Error Classification

- The RX side may not have a well-defined behavior for handling invalid Routing IDs, leading to security risks or misrouted packets.
- Incorrect error classification can occur if multiple fields (e.g., Routing ID, PCRC, MAC) are corrupted simultaneously due to the dependency

Mitigation Strategy:

- Inject invalid Routing IDs and verify consistent RX behavior.
- Use fine-grained error injection to isolate field-specific corruption and prevent misdiagnosed.

Risk:

🚫A PCIe Root Complex forwards TLPs with an incorrect Routing ID. The Endpoint doesn’t have a clear rule—should it discard, flag, or forward them? If mishandled, this can lead to unauthorized memory accesses or silent data corruption in an SoC

(b)Handling Pending Packets in Secure-to-Insecure Transitions

- Pending IDE packets or non-IDE Packets may be misrouted or processed when transitioning from secure to insecure mode which compromise security
- Selective and Link IDE Streams precedence issues may result in unprotected transactions when both are active.

Mitigation Strategy:

- Ensure pending TLPs and Non-IDE TLP’s are properly flushed or reclassified before security mode changes to Insecure.
- Define clear Selective vs. Link IDE precedence rules in the verification plan as per spec’s.

Risk:

🚫 A PCIe in an SoC downgrades to Insecure mode during link reinitialization. If pending encrypted TLPs aren’t flushed, some TLPs may be exposed in plaintext before reaching the host, leading to potential data leaks.

(c)Credit and Counter Management Under Error Conditions

- Improper credit tracking during error handling may cause protocol deadlocks.
- Underflow conditions must be managed, and overflow testing (2^1024 TLPs) is impractical**.**

Mitigation Strategy:

- Use white-box logging to monitor credit updates during error-handling scenarios.
- Implement independent RX and TX counters on **both** DUT and VIP sides to validate underflow conditions and overflow conditions to avoid performance overhead as well.

Risk:

🚫 A PCIe switch handling IDE-encrypted TLPs discards a corrupted packet but forgets to update the credit counter. This eventually leads to a transaction stall, as the root complex believes credits are exhausted.

(d) Multi-Stream IDE Verification in Mixed Traffic Classes

- Verifying transactions across multiple Stream IDs with a mix of link and selective encryption.
- Ensuring encryption enforcement, integrity checks, and protocol compliance across random traffic class (TC) transactions

Mitigation Strategy (Extended):

- The recommendation here would be to initiate transactions for different traffic classes in a random fashion with a mix of encrypted and unencrypted TLPs for good path testing.

Risk:

🛇 Unencrypted TLPs may be incorrectly forwarded in encrypted paths which Potential data leakage and integrity failures.

(e)IDE Key Management and Synchronization Issues

- Key Mismatch During Transitions: When switching between encryption key sets (e.g., from k=0 to k=1), there may be desynchronization between the sender and receiver, causing decryption failures.
- If the encryption key is expired or incorrectly rotated, TLPs may be unintentionally dropped or misrouted.

Mitigation Strategy:

- Implement controlled key switching mechanisms and track key synchronization events.
- Verify that both the DUT and VIP handle key mismatches gracefully without silent data loss.

Risk:

🚫 A PCIe endpoint key shifts for encryption, but the RC fails to update the decryption key in time. As a result, the host discards all incoming encrypted TLPs, breaking the data flow.

Addressing Blind Spots with Targeted Techniques

Technique	Testbench Application	SoC/IP Verification Impact	Challenges Addressed
State Machine for IDE Capability Discovery	Ensures seamless state transitions, preventing security misconfigurations.	Ensures correct IDE activation across multiple IPs and maintains consistency in security enforcement.	Handling complex IDE state transitions and preventing misalignment between security states.
Operation Codes in IDE Context	Validates secure OpCode processing, eliminating protocol violations and security risks.	Detects opcode misrouting and prevents unauthorized access or manipulation.	Ensuring protocol compliance and preventing side-channel attacks due to incorrect OpCode handling.
White Box Logging	Provides deep visibility into IDE transactions, tracking encryption integrity in real-time.	Enables real-time monitoring of IDE transactions.	Tracking encrypted transactions without exposing sensitive data and maintaining real-time traceability.
Modular Debug Hooks	Enables targeted debugging and anomaly detection, ensuring correct IDE behavior.	Allows selective activation of debugging features for runtime security verification.	Providing efficient debugging mechanisms without interfering with the secure execution of IDE transactions.
Flexible Verification Methods for IDE Functionality	Adapts to dynamic test conditions and directed testing	Ensures IDE security remains robust.	Maintaining verification scalability.

Eliminating Blind Spots in IDE Verification for PCIe & CXL

Bug 1: Protocol Flit Transmission

- Flits from the next MAC Epoch transmitted **before TMAC flit**, violating spec.

Bug 2: Mixing IDE & Non-IDE TLPs in the Same Flit

- Selective Streams allowed IDE & Non-IDE TLPs together but lacked clear differentiation.

Bug 3: Address Association Block Register Reconfiguration Issue

- Address Association Block Register locked post-initialization, reducing runtime flexibility.

Bug 4: Flushing queue (non-IDE + IDE traffic in pending queue)

- Handling buffer resets without data loss or security issues when both encrypted (IDE) and non-encrypted traffic are in the queue

Innovative Verification Methodology & Impact

- **Targeted Blind Spot Elimination** – Applied state-aware validation, protocol-specific error injection, and adaptive debugging.
- **97% IDE Verification Coverage** – Achieved using ML-driven regression, improving test efficiency and accuracywithin a very less span of Verification Timelines.
- **Significant Debugging & Manual Effort Reduction** – Improved testbench automation and real-time error traceability.
- **Scalability Beyond PCIe & CXL** – Techniques extend to Ethernet, HDMI, MIPI, and future high-speed protocols.

Conclusion

- These five techniques enhance **multi-layered IDE verification** by enabling **state-aware security validation, deep transaction visibility, and adaptive debugging in testbenches**.
- They ensure **seamless encryption-decryption flows, error resilience, and robust security compliance** across SoC and IP verification.
- They ensure preventing **security blind spots** in **PCIe and CXL** environments

Questions