



**NANYANG  
TECHNOLOGICAL  
UNIVERSITY**  
SINGAPORE

# Achilles Heel in Secure Boot: Breaking RSA Authentication and decrypted bitstream recovery from Zynq-7000 SoC

Prasanna Ravi

**Arpan Jati**

Shivam Bhasin

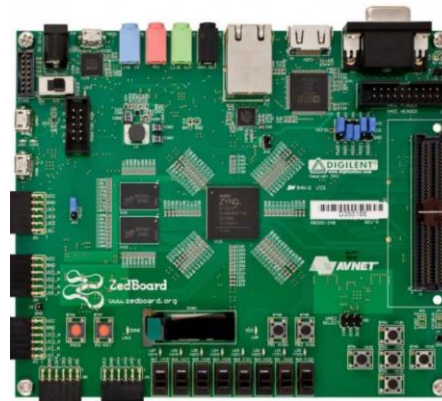
*Temasek Labs, NTU Singapore*

*14<sup>th</sup> March, 2024*



# Introduction

- ❑ The Zynq-7000 is widely deployed series of FPGA+SOC device from Xilinx/AMD.
- ❑ Used for defense, aerospace and medical applications.
- ❑ We have identified critical security vulnerability in RSA authentication process of Secure Boot of Zynq 7000 SoC.
- ❑ The flaw is present in FSBL (first stage boot loader), which can be exploited by constructing a malicious boot image to boot unauthorized applications.
- ❑ Once we get root access, we demonstrate a novel attack to recover the encrypted bitstream and applications.
- ❑ We also extend our analysis to the critical BootROM.



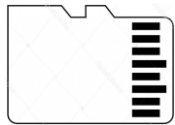
# Outline

- ❑ Introduction
- ❑ RSA Authentication Attack on Zynq-7000
  - ❑ **Background: Attack Model, Secure Boot and RSA Authentication**
  - ❑ Vulnerability in FSBL
  - ❑ Attack Implementation: Using SD Card Switcher Board
- ❑ Starbleed on Zynq
  - ❑ Introduction and Working
  - ❑ Experimental Results
- ❑ Analyzing SD Card Data Transfer
- ❑ BootROM
  - ❑ Possible BootROM Vulnerabilities
  - ❑ PHT Transfer Analysis
  - ❑ BootROM Data Transfer Analysis
- ❑ Conclusion and Future Works

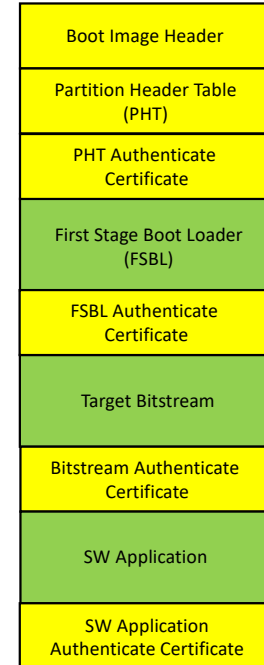
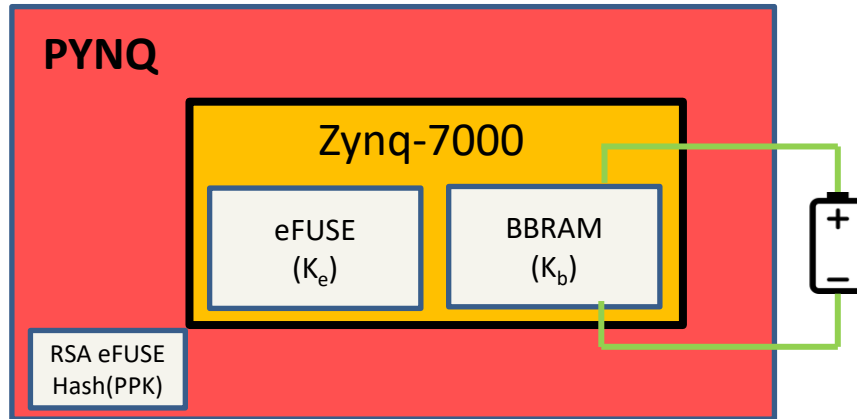


# Attack Model

- ❑ Attacker has access to the target Zynq-7000 device.
- ❑ He/She can obtain the valid secure boot image used in the target device.
- ❑ RSA Authentication mandatory for secure boot (RSA Enable eFUSE is burnt).



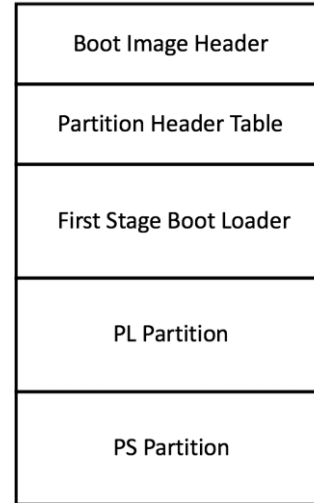
SD card  
Boot



■ - Unencrypted  
■ - Encrypted

# Secure Boot of Zynq-7000 SoC

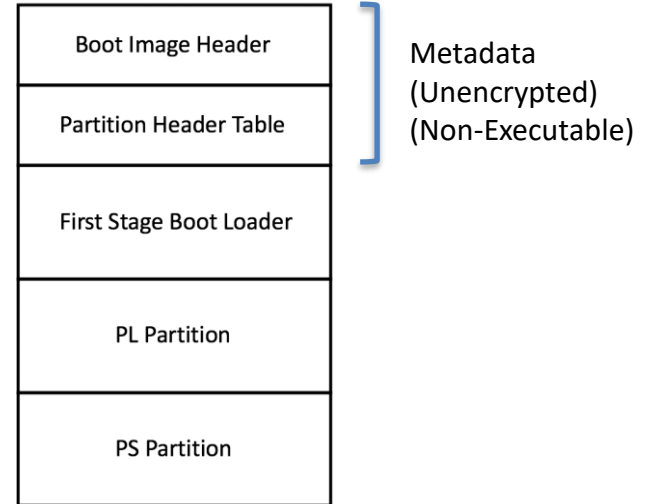
- ❑ **Secure Boot Image** is typically stored in a non-volatile memory such as SD Card, NOR or NAND flash.
- ❑ It has information about:
  - ❑ Different HW and SW components to be loaded on the Zynq device.
  - ❑ Where and how each and every partition has to be loaded during the secure boot procedure.
- ❑ It contains multiple partitions:
  - ❑ BootROM header (**BIH**)
  - ❑ Partition Header Table (**PHT**)
  - ❑ First Stage Boot Loader (**FSBL**)
  - ❑ PL partition/s (bitstream)
  - ❑ PS partition/s (Standalone Application or Operating System)
- ❑ Each partition can be **AES encrypted**, **HMAC authenticated** and **RSA authenticated**



Typical Boot Image Structure

# Secure Boot of Zynq-7000 SoC

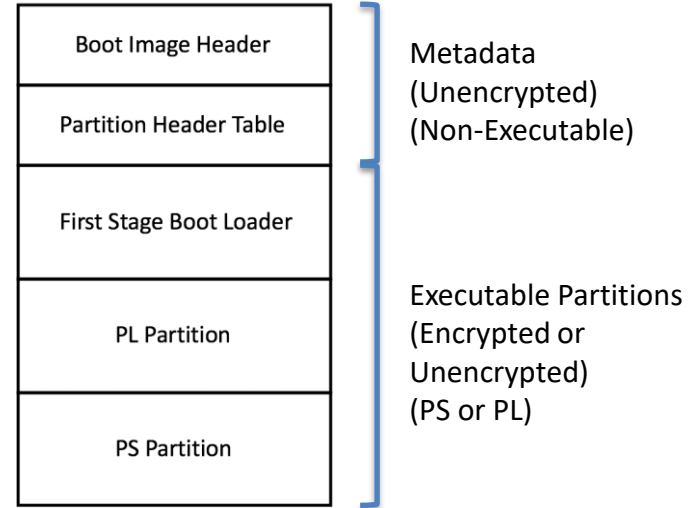
- ❑ **Secure Boot Image** is typically stored in a non-volatile memory such as SD Card, NOR or NAND flash.
- ❑ It has information about:
  - ❑ Different HW and SW components to be loaded on the Zynq device.
  - ❑ Where and how each and every partition has to be loaded during the secure boot procedure.
- ❑ It contains multiple partitions:
  - ❑ BootROM header (**BIH**)
  - ❑ Partition Header Table (**PHT**)
  - ❑ First Stage Boot Loader (**FSBL**)
  - ❑ PL partition/s (bitstream)
  - ❑ PS partition/s (Standalone Application or Operating System)
- ❑ Each partition can be **AES encrypted**, **HMAC authenticated** and **RSA authenticated**



Typical Boot Image Structure

# Secure Boot of Zynq-7000 SoC

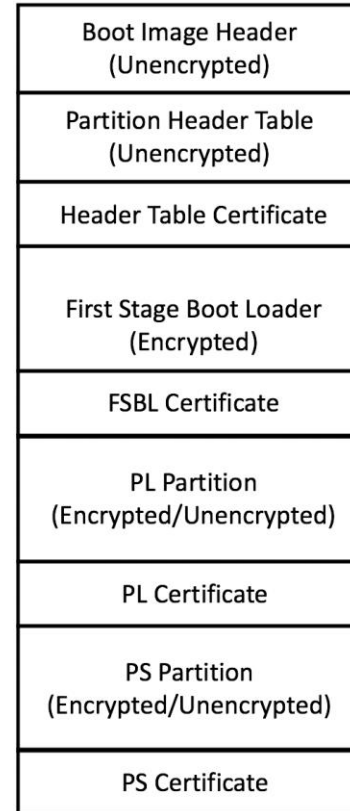
- ❑ **Secure Boot Image** is typically stored in a non-volatile memory such as SD Card, NOR or NAND flash.
- ❑ It has information about:
  - ❑ Different HW and SW components to be loaded on the Zynq device.
  - ❑ Where and how each and every partition has to be loaded during the secure boot procedure.
- ❑ It contains multiple partitions:
  - ❑ BootROM header (**BIH**)
  - ❑ Partition Header Table (**PHT**)
  - ❑ First Stage Boot Loader (**FSBL**)
  - ❑ PL partition/s (bitstream)
  - ❑ PS partition/s (Standalone Application or Operating System)
- ❑ Each partition can be **AES encrypted**, **HMAC authenticated** and **RSA authenticated**



Typical Boot Image Structure

# Secure Boot of Zynq-7000 SoC

- ❑ **Secure Boot Image** is typically stored in a non-volatile memory such as SD Card, NOR or NAND flash.
- ❑ It contains multiple partitions:
  - ❑ BootROM header (**BIH**)
  - ❑ Partition Header Table (**PHT**)
  - ❑ First Stage Boot Loader (**FSBL**)
  - ❑ PL partition/s (bitstream)
  - ❑ PS partition/s (Standalone Application or Operating System)

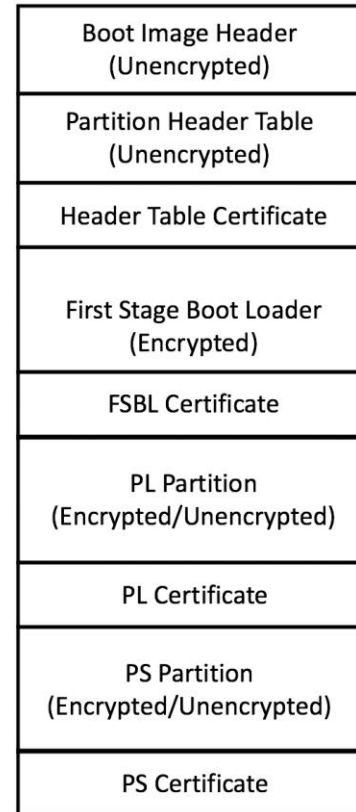


**Boot Image (Authenticated)**



# Secure Boot of Zynq-7000 SoC

- ❑ **Secure Boot Image** is typically stored in a non-volatile memory such as SD Card, NOR or NAND flash.
- ❑ It contains multiple partitions:
  - ❑ BootROM header (**BIH**)
  - ❑ Partition Header Table (**PHT**)
  - ❑ First Stage Boot Loader (**FSBL**)
  - ❑ PL partition/s (bitstream)
  - ❑ PS partition/s (Standalone Application or Operating System)

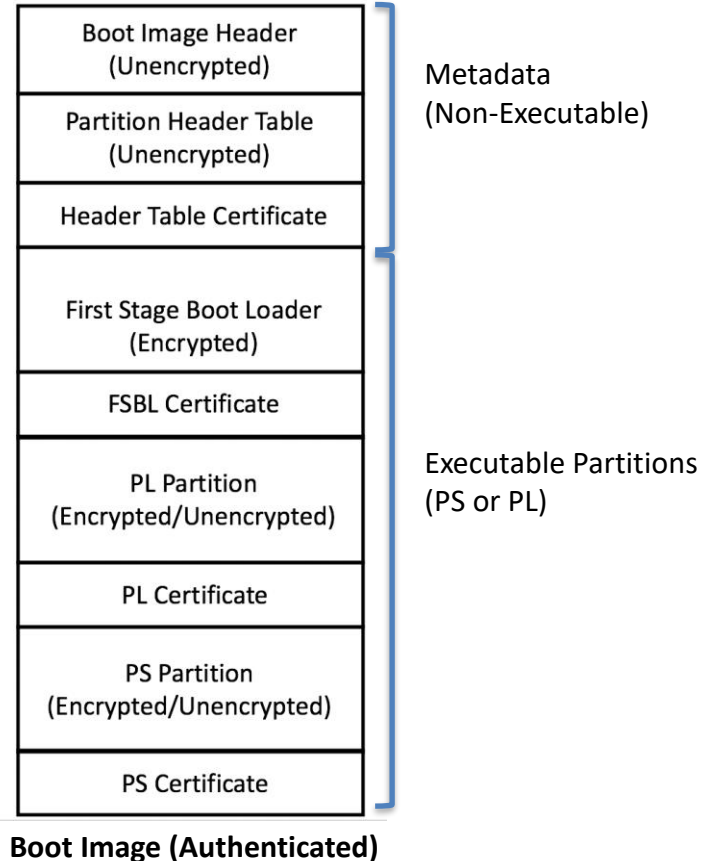


Metadata  
(Non-Executable)

**Boot Image (Authenticated)**

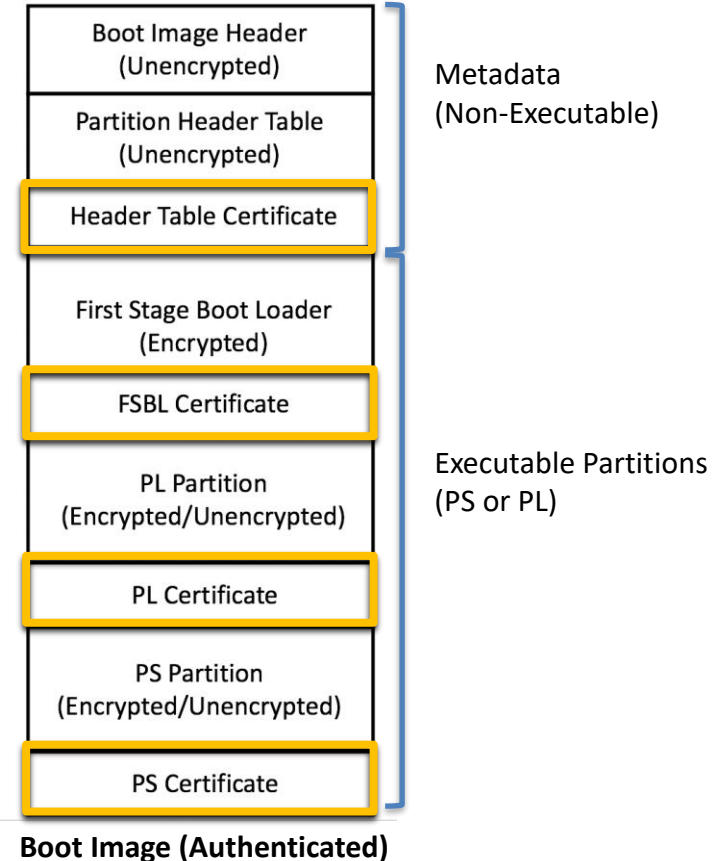
# Secure Boot of Zynq-7000 SoC

- ❑ **Secure Boot Image** is typically stored in a non-volatile memory such as SD Card, NOR or NAND flash.
- ❑ It contains multiple partitions:
  - ❑ BootROM header (**BIH**)
  - ❑ Partition Header Table (**PHT**)
  - ❑ First Stage Boot Loader (**FSBL**)
  - ❑ PL partition/s (bitstream)
  - ❑ PS partition/s (Standalone Application or Operating System)



# Secure Boot of Zynq-7000 SoC

- ❑ **Secure Boot Image** is typically stored in a non-volatile memory such as SD Card, NOR or NAND flash.
- ❑ It contains multiple partitions:
  - ❑ BootROM header (**BIH**)
  - ❑ Partition Header Table (**PHT**)
  - ❑ First Stage Boot Loader (**FSBL**)
  - ❑ PL partition/s (bitstream)
  - ❑ PS partition/s (Standalone Application or Operating System)



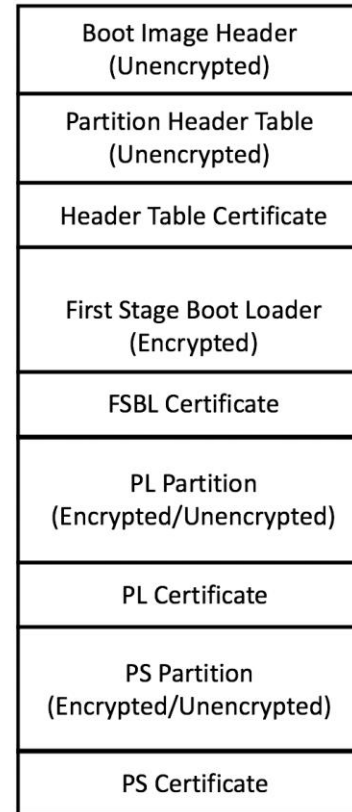
# RSA Authentication in Zynq-7000 SoC

- ❑ Based on the well-known RSA-2048 signature scheme
- ❑ Authentication of each partition is done using two types of keys:
  - ❑ Primary Key - Primary Public Key (PPK), Primary Secret Key (PSK)
  - ❑ Secondary Keys - Secondary Public Key (SPK), Secondary Secret Key (SSK)
- ❑ Primary Key (PPK, PSK) is fixed for a given device - Hash of PPK is burnt into RSA eFUSE (One-Time Programmable)
- ❑ Secondary Key (SPK, SSK) is specific to each partition and can be different for each.
- ❑ **Two-Step Authentication** (For Each Partition):
  - ❑ Primary Keys authenticate the Secondary Keys
  - ❑ Secondary Keys authenticate the Partition



# Boot Image Header (BIH): Intro

- ❑ Contains metadata about the boot image.
  - ❑ Encryption Status of Boot Image
  - ❑ Size of Boot Image
  - ❑ Location of FSBL
  - ❑ Load address of FSBL
- ❑ Upon reset, **BootROM** is the first piece of code executed
- ❑ **BootROM Execution:**
  - ❑ CRC check of BootROM is carried out
  - ❑ BIH is read from the SD card
  - ❑ FSBL is read from SD card (Along with certificate)
  - ❑ It is authenticated and decrypted based on the eFuse
  - ❑ If successful, control transferred to FSBL



→ Read by BootROM

# Partition Header Table (PHT) ): Intro

- ❑ Provides metadata info. about each partition in boot image.
- ❑ Each partition has a 64-byte entry in the PHT, read by FSBL.
- ❑ Each entry contains information about the partition such as:
  - ❑ **Partition Encryption status** (AES encrypted or not)
  - ❑ **Partition Authentication status** (RSA authenticated or not)
  - ❑ **Partition Length etc.**
- ❑ PHT also has a certificate which is verified by First Stage Boot Loader (FSBL).
- ❑ PHT is central to the secure boot process.
- ❑ If we tamper PHT, we can load any application of our choice!!!

Boot Image Header (Unencrypted)
Partition Header Table (Unencrypted)
Header Table Certificate
First Stage Boot Loader (Encrypted)
FSBL Certificate
PL Partition (Encrypted/Unencrypted)
PL Certificate
PS Partition (Encrypted/Unencrypted)
PS Certificate

→ Read by FSBL



- ❑ Gathers information about the individual partitions from the Partition Header Table.



# FSBL

- ❑ Gathers information about the individual partitions from the Partition Header Table.
- ❑ All partitions are loaded into a temporary location in the DDR memory from the NVM memory.

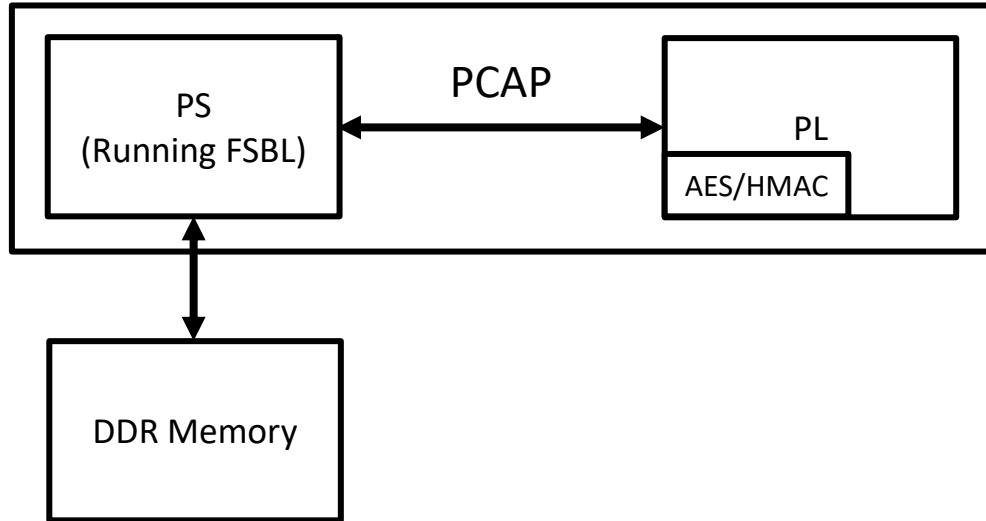
# FSBL

- ❑ Gathers information about the individual partitions from the Partition Header Table.
- ❑ All partitions are loaded into a temporary location in the DDR memory from the NVM memory.
- ❑ They are then suitably decrypted and authenticated before being used for configuration appropriately (Configure PL using PL bitstream and transfer control to last PS partition).

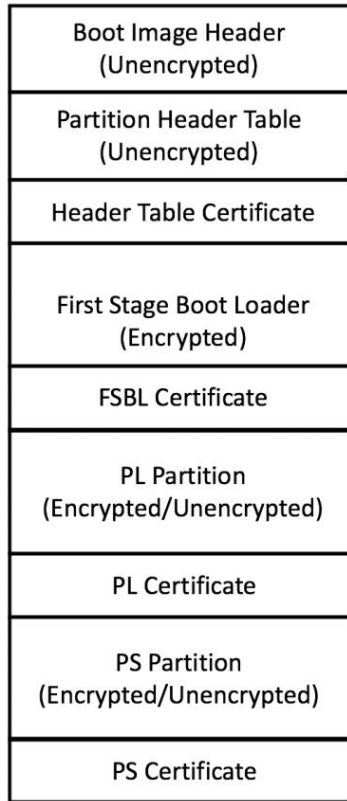


# Overview of FSBL Operation

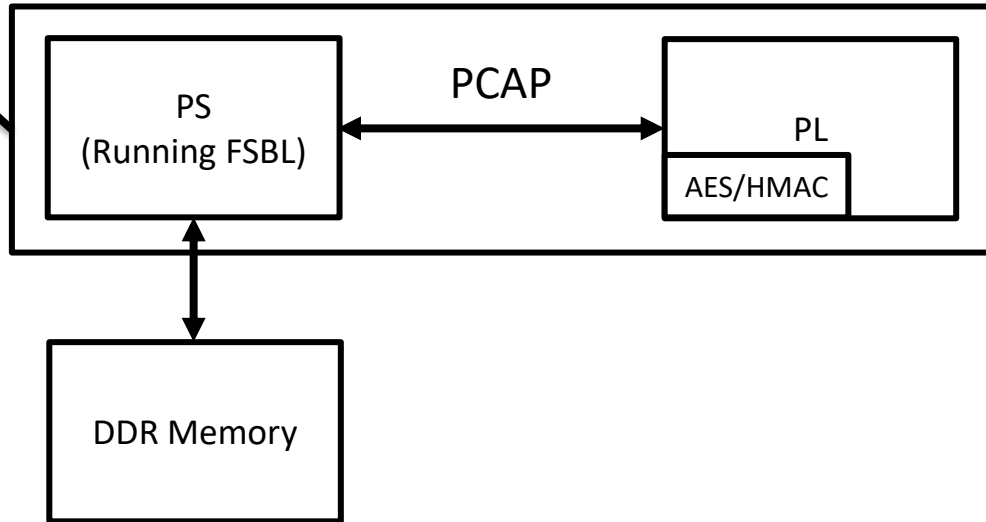
Boot Image Header (Unencrypted)
Partition Header Table (Unencrypted)
Header Table Certificate
First Stage Boot Loader (Encrypted)
FSBL Certificate
PL Partition (Encrypted/Unencrypted)
PL Certificate
PS Partition (Encrypted/Unencrypted)
PS Certificate



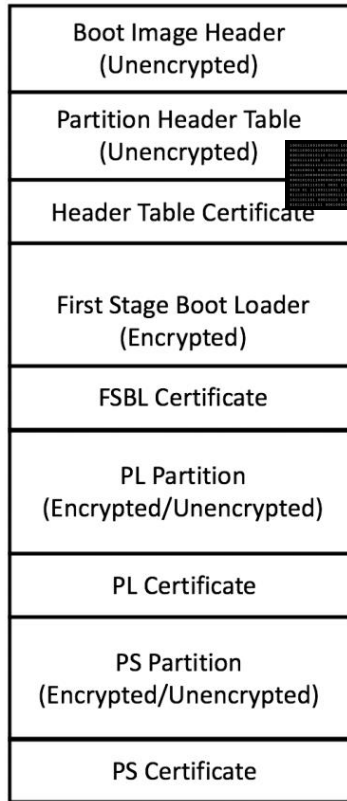
# Overview of FSBL Operation



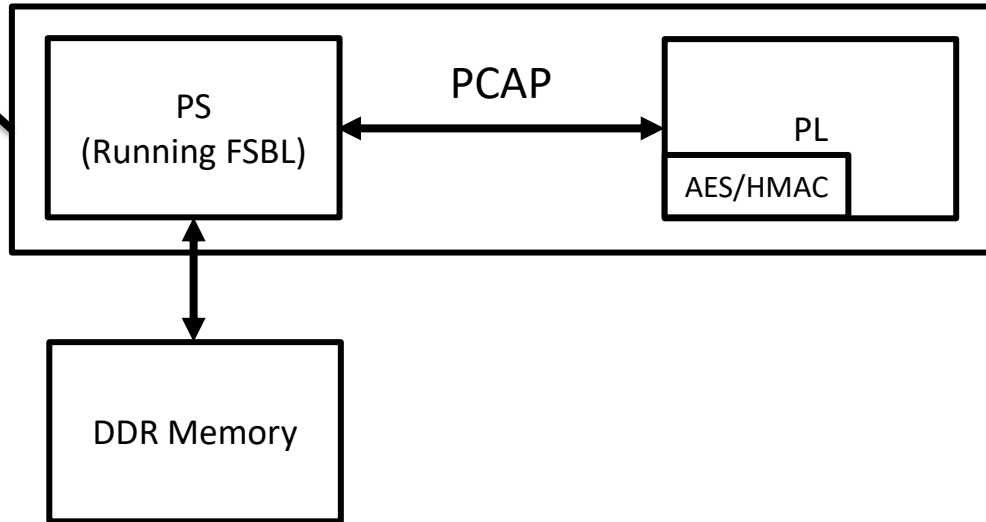
**Step 1:** FSBL Authenticates PHT certificate and proceeds (uses PHT) if successful



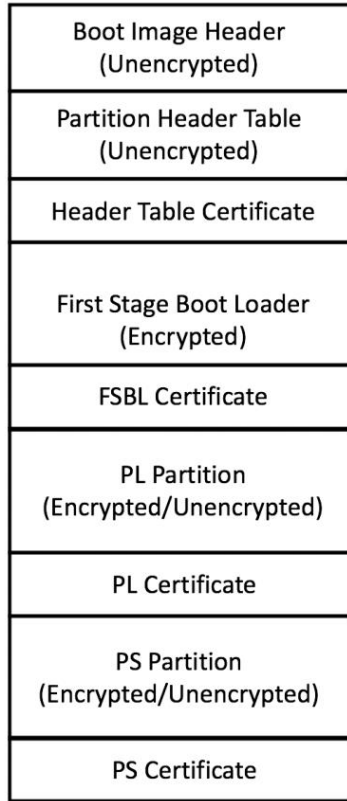
# Overview of FSBL Operation



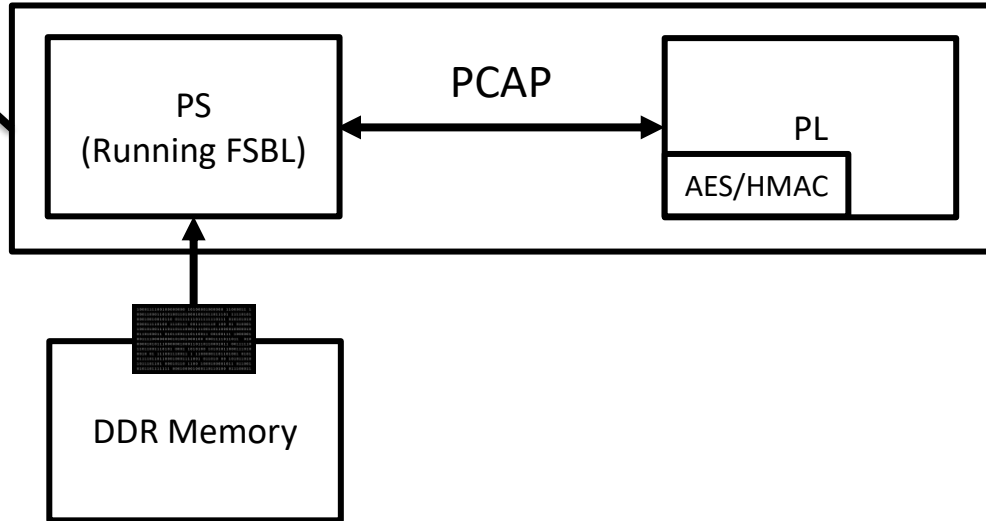
**Step 1:** FSBL Authenticates PHT certificate and proceeds (uses PHT) if successful



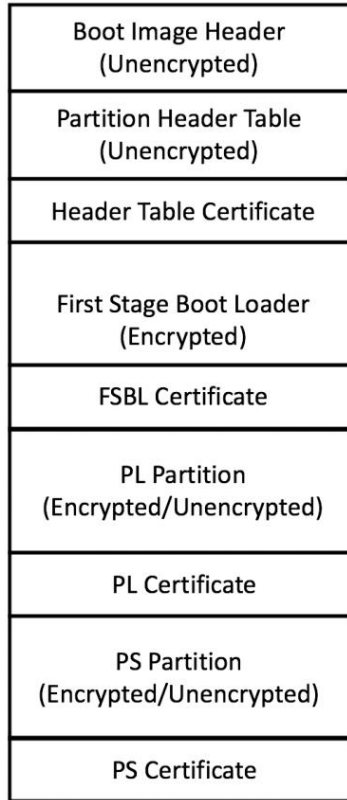
# Overview of FSBL Operation



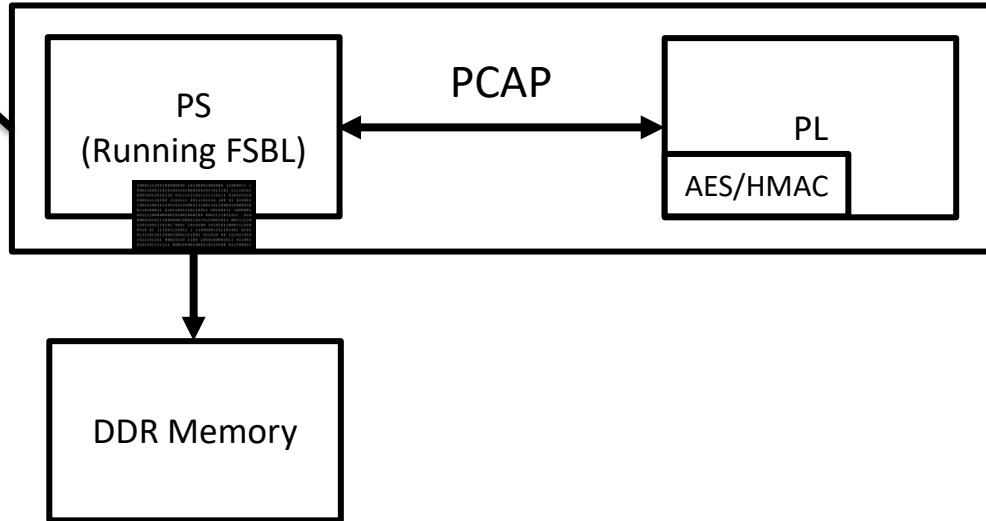
**Step 1:** FSBL Authenticates PHT certificate and proceeds (uses PHT) if successful



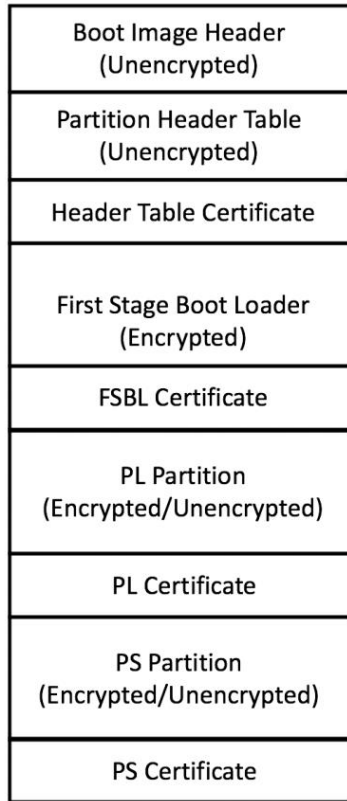
# Overview of FSBL Operation



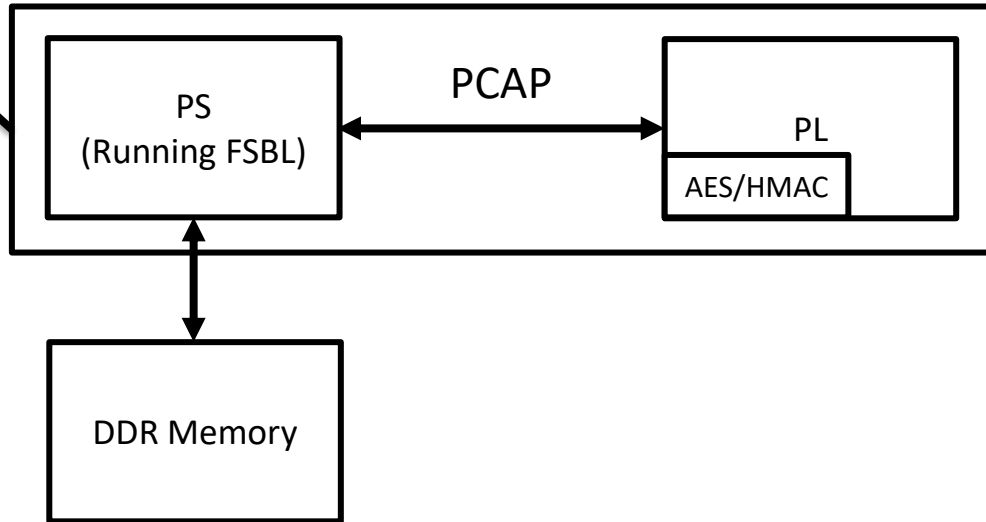
**Step 1:** FSBL Authenticates PHT certificate and proceeds (uses PHT) if successful



# Overview of FSBL Operation



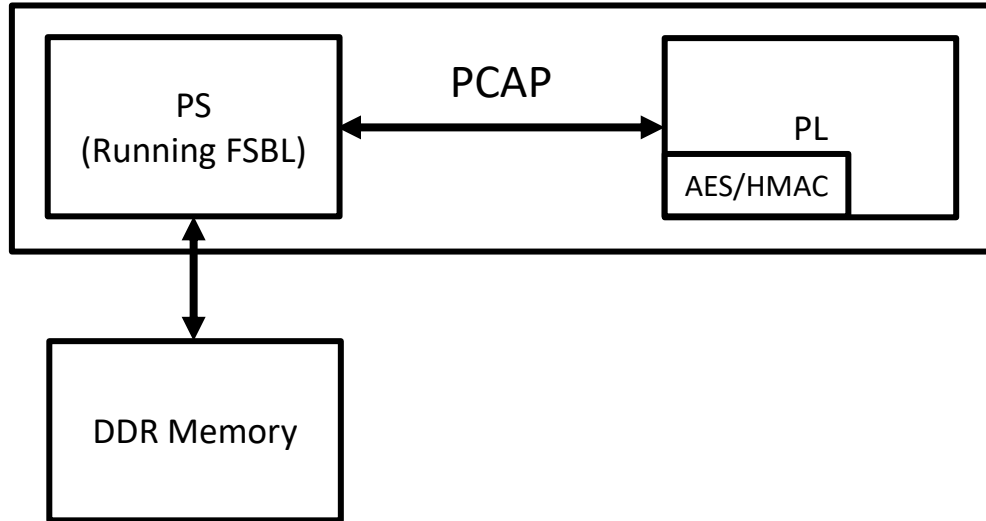
**Step 1:** FSBL Authenticates PHT certificate and proceeds (uses PHT) if successful



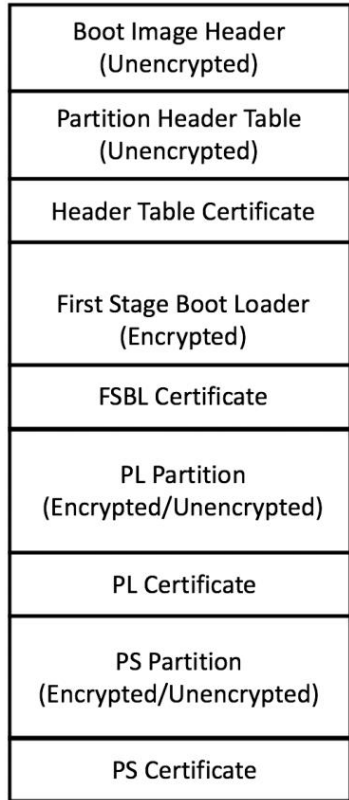


# Overview of FSBL Operation

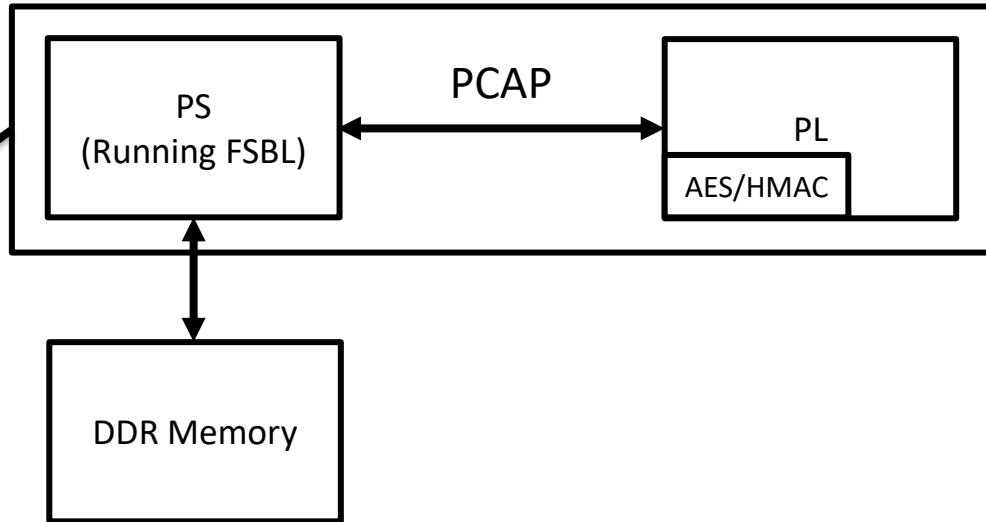
Boot Image Header (Unencrypted)
Partition Header Table (Unencrypted)
Header Table Certificate
First Stage Boot Loader (Encrypted)
FSBL Certificate
PL Partition (Encrypted/Unencrypted)
PL Certificate
PS Partition (Encrypted/Unencrypted)
PS Certificate



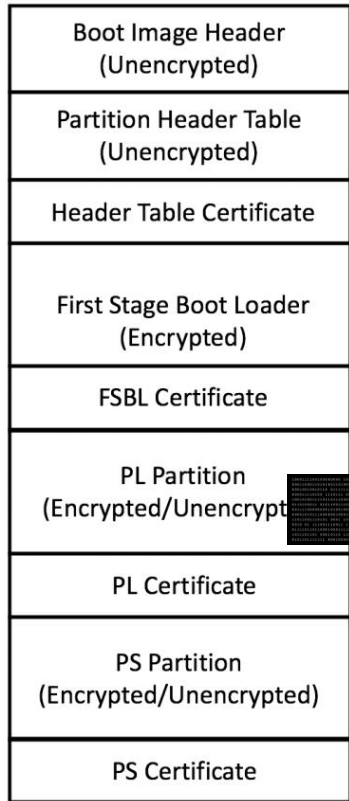
# Overview of FSBL Operation



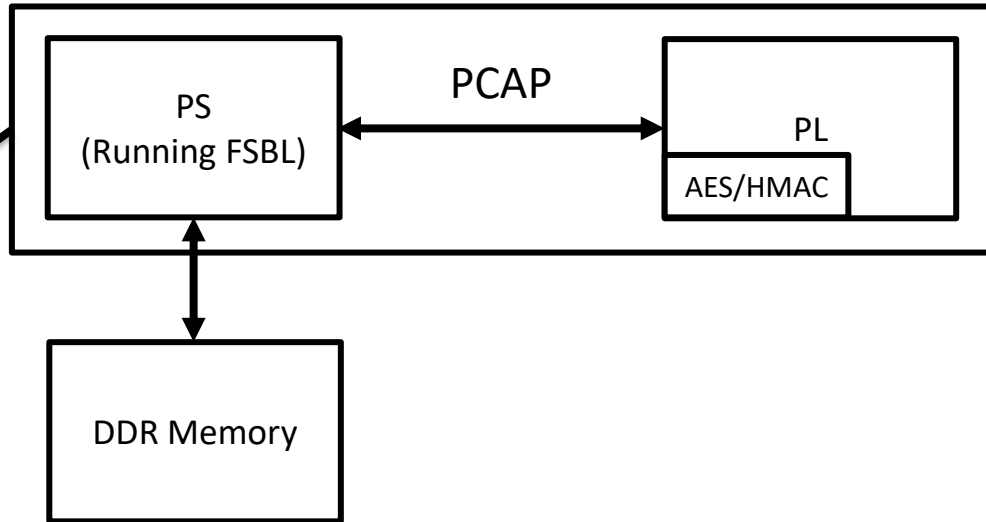
**Step 2:** Authenticate RSA signature of PL bitstream (if required)



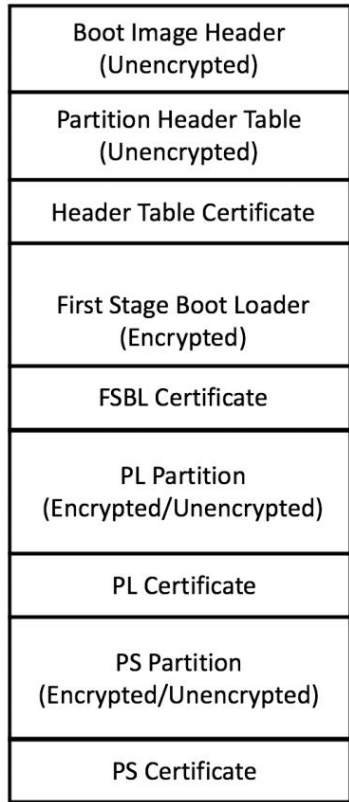
# Overview of FSBL Operation



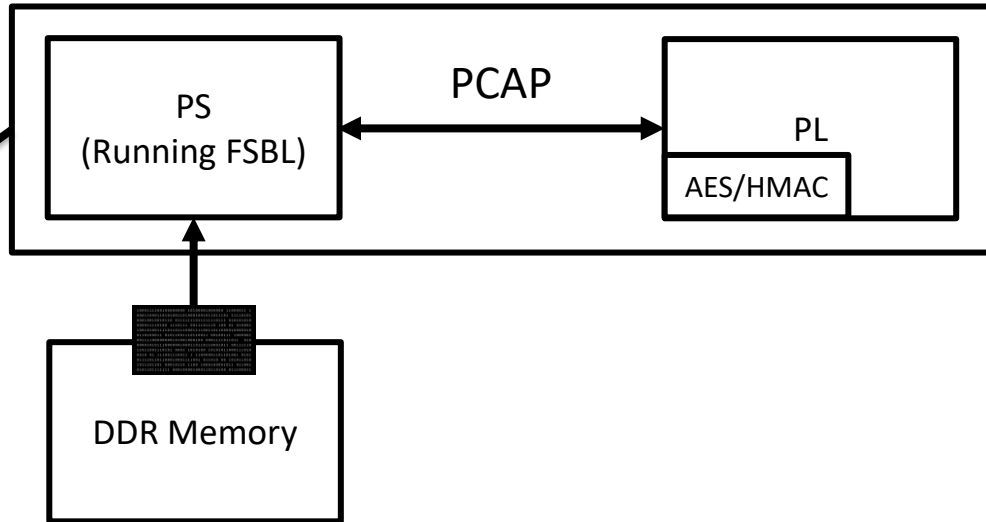
**Step 2:** Authenticate RSA signature of PL bitstream (if required)



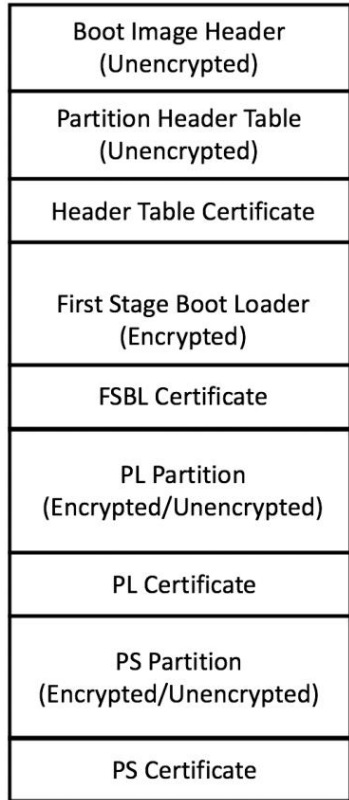
# Overview of FSBL Operation



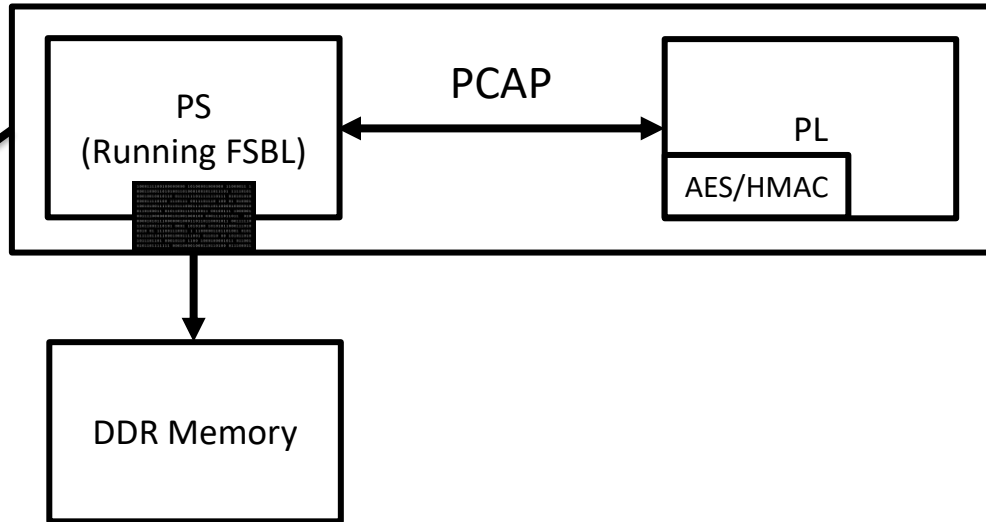
**Step 2:** Authenticate RSA signature of PL bitstream (if required)



# Overview of FSBL Operation

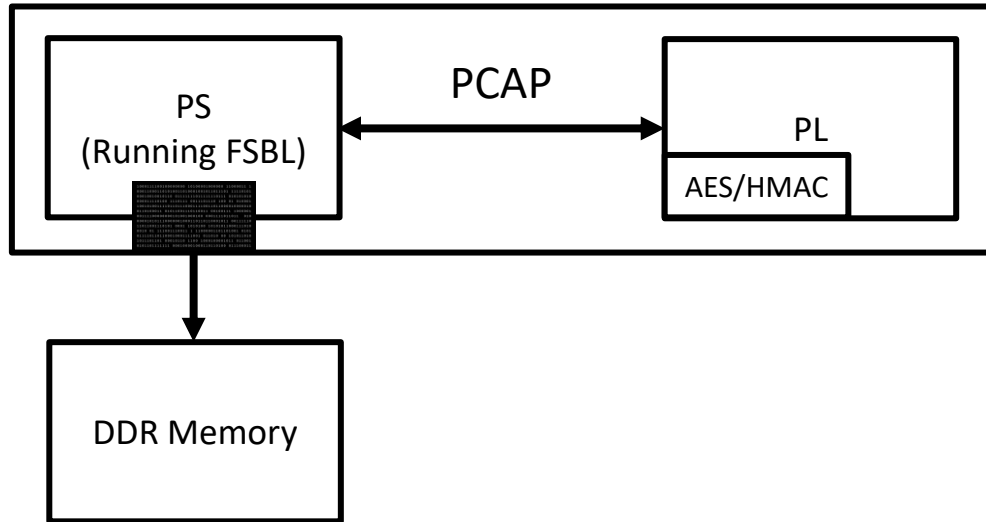


**Step 2:** Authenticate RSA signature of PL bitstream (if required)



# Overview of FSBL Operation

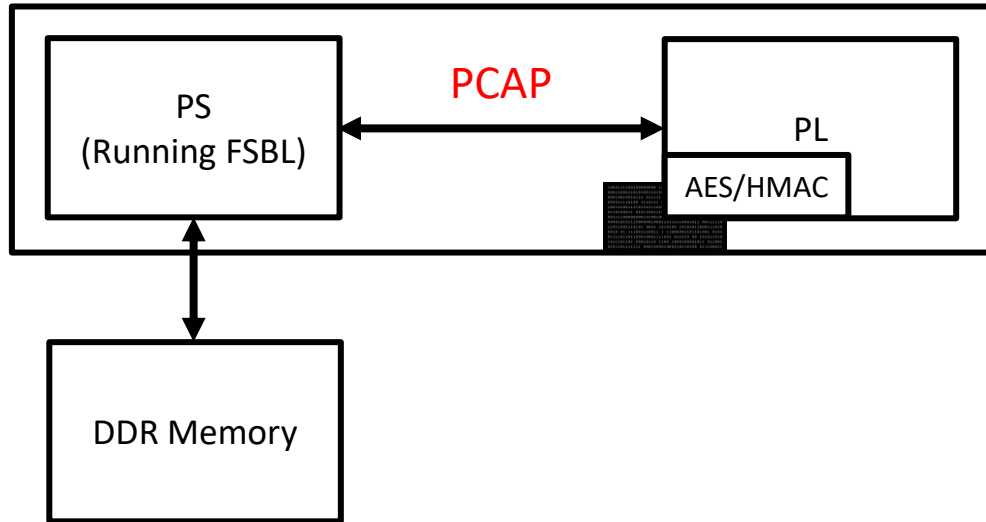
Boot Image Header (Unencrypted)
Partition Header Table (Unencrypted)
Header Table Certificate
First Stage Boot Loader (Encrypted)
FSBL Certificate
PL Partition (Encrypted/Unencrypted)
PL Certificate
PS Partition (Encrypted/Unencrypted)
PS Certificate



# Overview of FSBL Operation

Boot Image Header (Unencrypted)
Partition Header Table (Unencrypted)
Header Table Certificate
First Stage Boot Loader (Encrypted)
FSBL Certificate
PL Partition (Encrypted/Unencrypted)
PL Certificate
PS Partition (Encrypted/Unencrypted)
PS Certificate

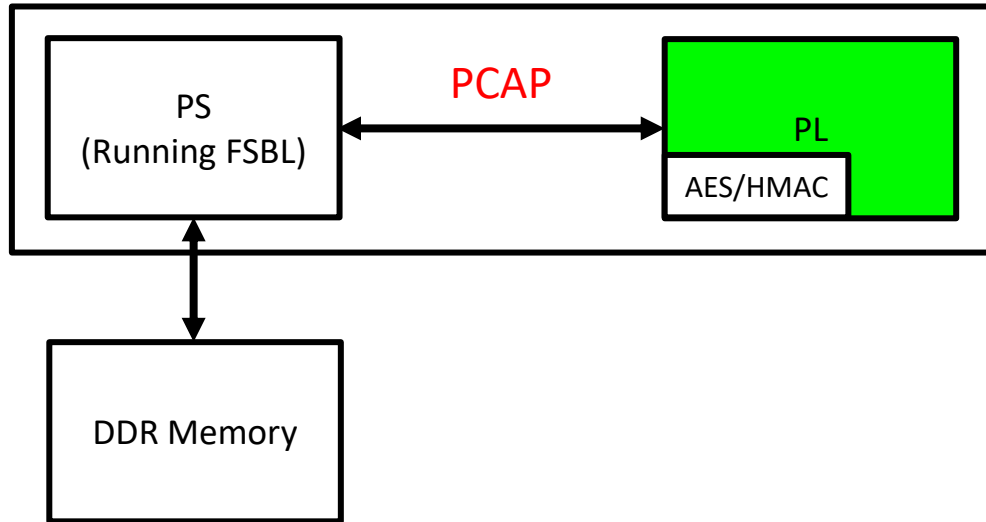
## Step 3: Decrypt and authenticate bitstream (if required)



# Overview of FSBL Operation

Boot Image Header (Unencrypted)
Partition Header Table (Unencrypted)
Header Table Certificate
First Stage Boot Loader (Encrypted)
FSBL Certificate
PL Partition (Encrypted/Unencrypted)
PL Certificate
PS Partition (Encrypted/Unencrypted)
PS Certificate

**Step 3:** Decrypt and authenticate bitstream (if required)

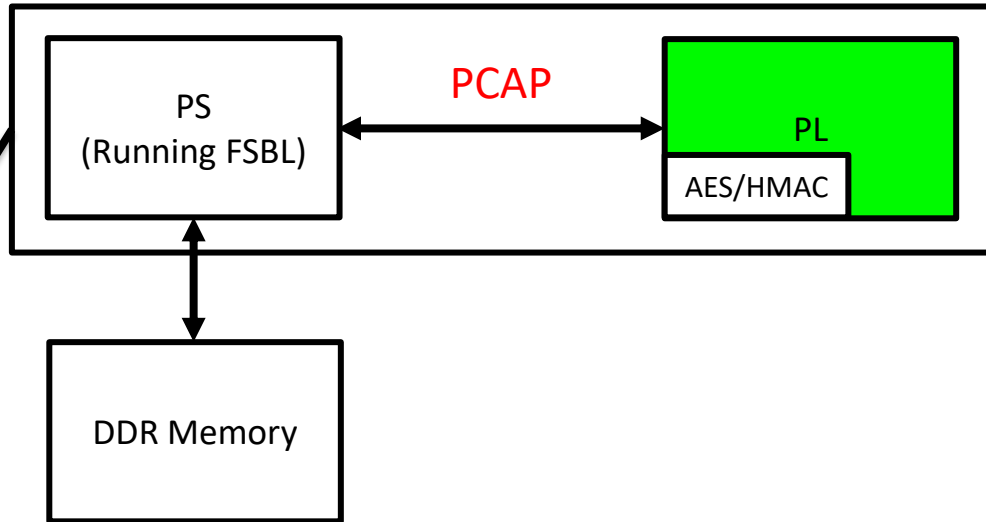




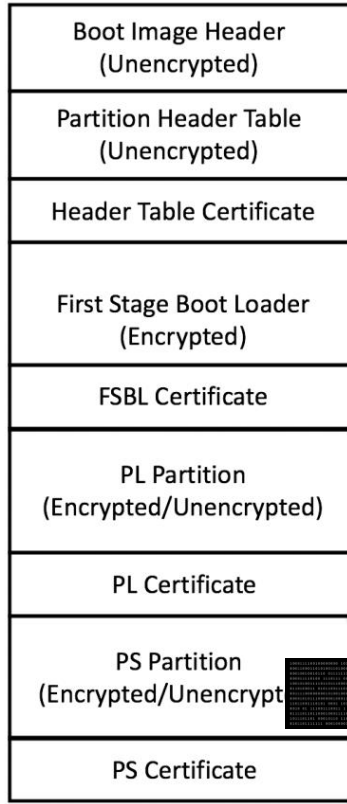
# Overview of FSBL Operation

Boot Image Header (Unencrypted)
Partition Header Table (Unencrypted)
Header Table Certificate
First Stage Boot Loader (Encrypted)
FSBL Certificate
PL Partition (Encrypted/Unencrypted)
PL Certificate
PS Partition (Encrypted/Unencrypted)
PS Certificate

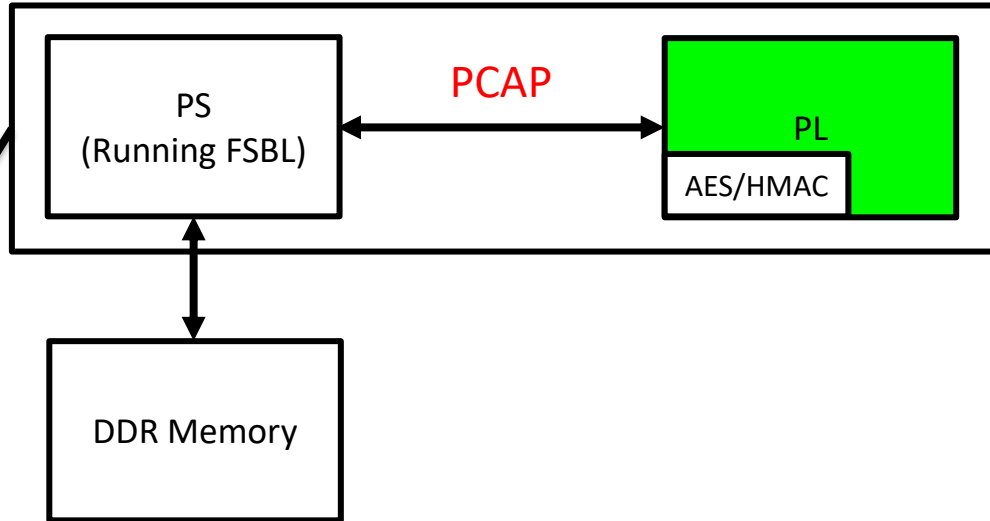
**Step 4:** Authenticate RSA signature of PS application(if required)



# Overview of FSBL Operation



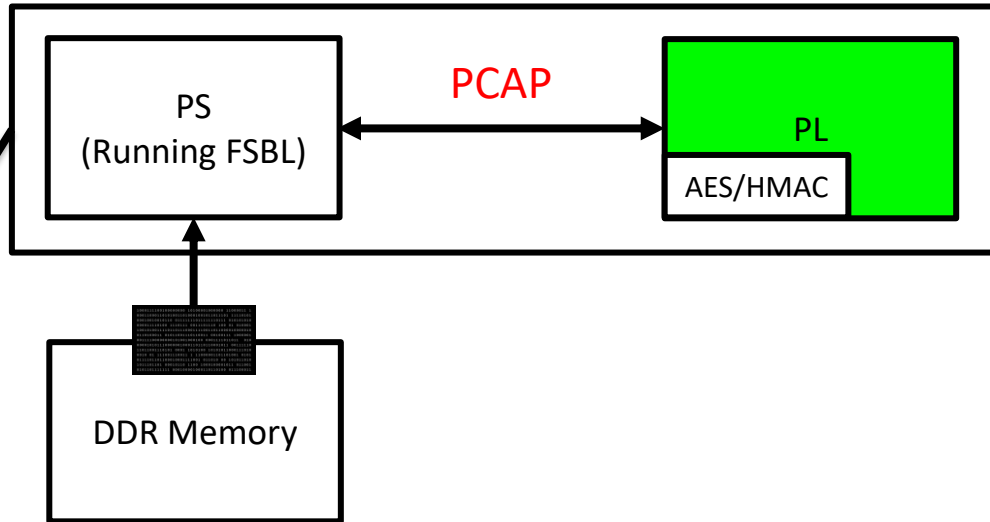
**Step 4:** Authenticate RSA signature of PS application(if required)



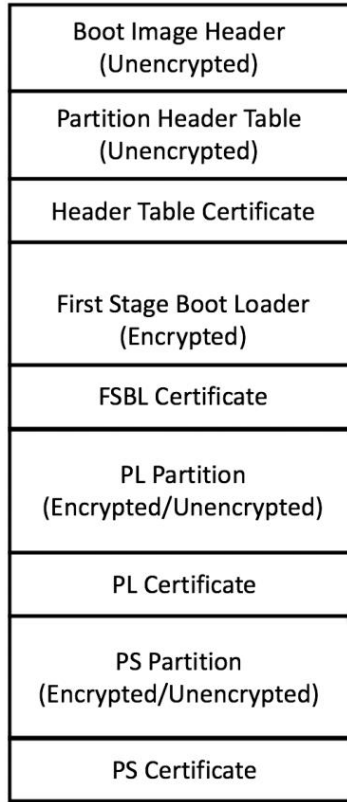
# Overview of FSBL Operation

Boot Image Header (Unencrypted)
Partition Header Table (Unencrypted)
Header Table Certificate
First Stage Boot Loader (Encrypted)
FSBL Certificate
PL Partition (Encrypted/Unencrypted)
PL Certificate
PS Partition (Encrypted/Unencrypted)
PS Certificate

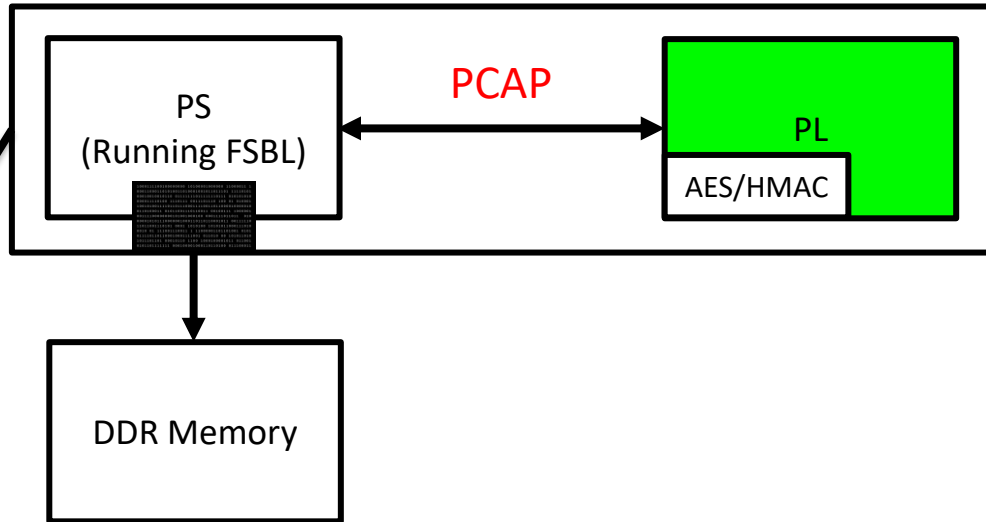
**Step 4:** Authenticate RSA signature of PS application(if required)



# Overview of FSBL Operation



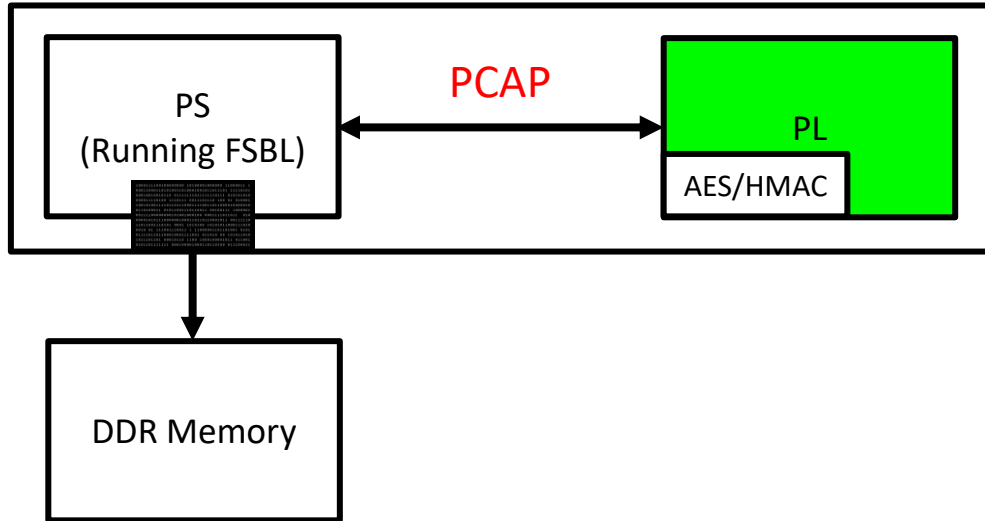
**Step 4:** Authenticate RSA signature of PS application(if required)



# Overview of FSBL Operation

Boot Image Header (Unencrypted)
Partition Header Table (Unencrypted)
Header Table Certificate
First Stage Boot Loader (Encrypted)
FSBL Certificate
PL Partition (Encrypted/Unencrypted)
PL Certificate
PS Partition (Encrypted/Unencrypted)
PS Certificate

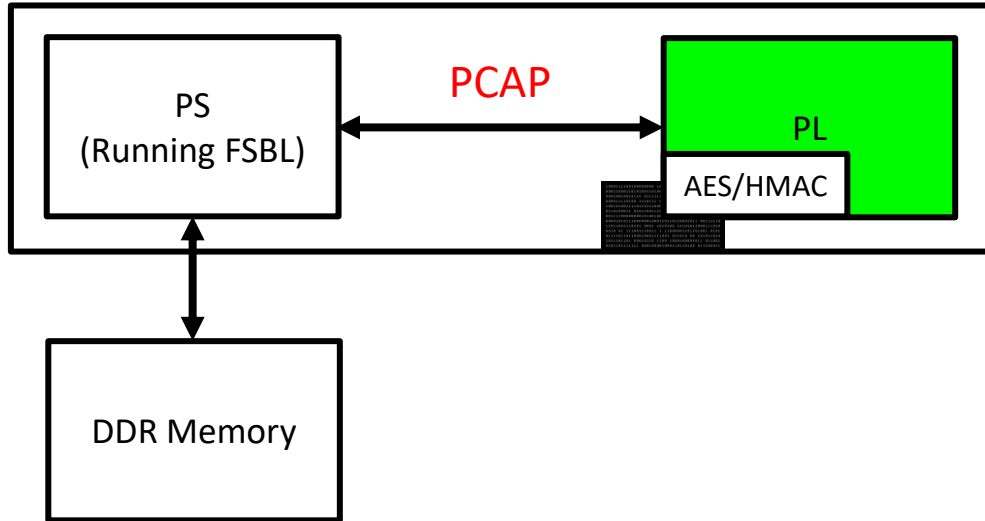
**Step 5:** Decrypt and authenticate PS application (if required)



# Overview of FSBL Operation

Boot Image Header (Unencrypted)
Partition Header Table (Unencrypted)
Header Table Certificate
First Stage Boot Loader (Encrypted)
FSBL Certificate
PL Partition (Encrypted/Unencrypted)
PL Certificate
PS Partition (Encrypted/Unencrypted)
PS Certificate

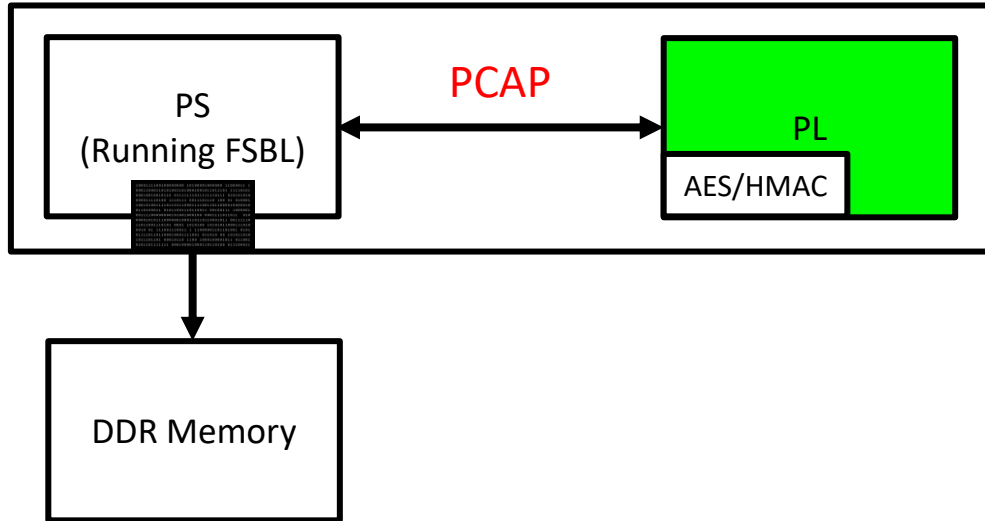
**Step 5:** Decrypt and authenticate PS application (if required)



# Overview of FSBL Operation

Boot Image Header (Unencrypted)
Partition Header Table (Unencrypted)
Header Table Certificate
First Stage Boot Loader (Encrypted)
FSBL Certificate
PL Partition (Encrypted/Unencrypted)
PL Certificate
PS Partition (Encrypted/Unencrypted)
PS Certificate

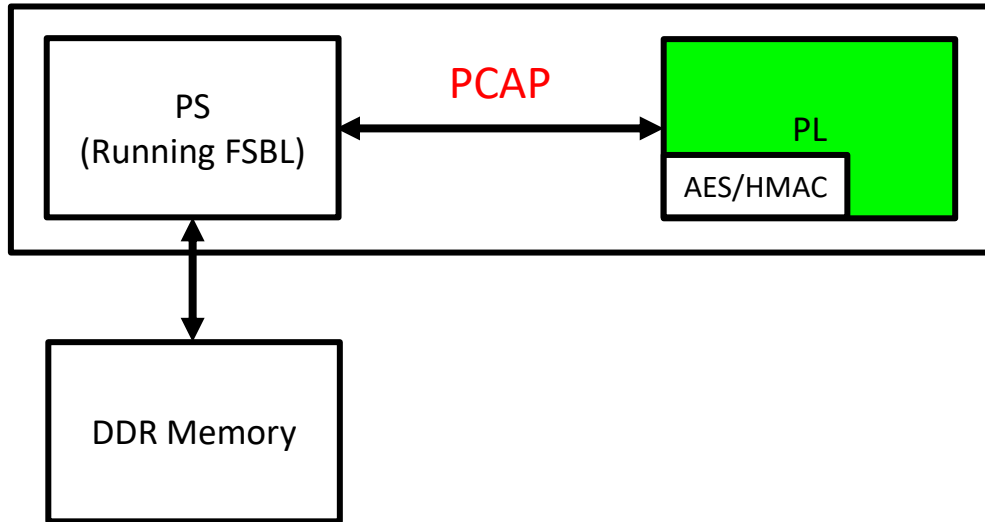
**Step 5:** Decrypt and authenticate PS application (if required)



# Overview of FSBL Operation

Boot Image Header (Unencrypted)
Partition Header Table (Unencrypted)
Header Table Certificate
First Stage Boot Loader (Encrypted)
FSBL Certificate
PL Partition (Encrypted/Unencrypted)
PL Certificate
PS Partition (Encrypted/Unencrypted)
PS Certificate

**Step 5:** Decrypt and authenticate PS application (if required)

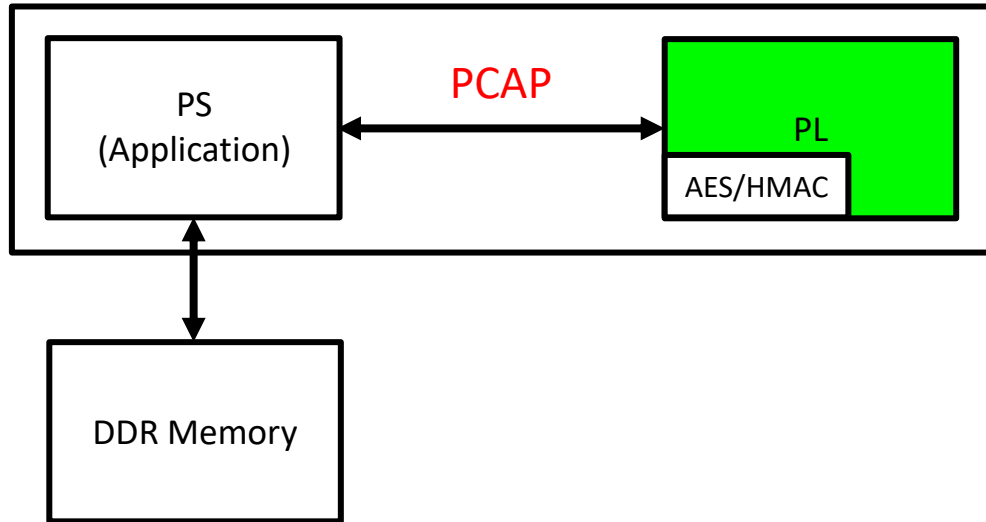




# Overview of FSBL Operation

Boot Image Header (Unencrypted)
Partition Header Table (Unencrypted)
Header Table Certificate
First Stage Boot Loader (Encrypted)
FSBL Certificate
PL Partition (Encrypted/Unencrypted)
PL Certificate
PS Partition (Encrypted/Unencrypted)
PS Certificate

## Step 6: Hand over control to PS application

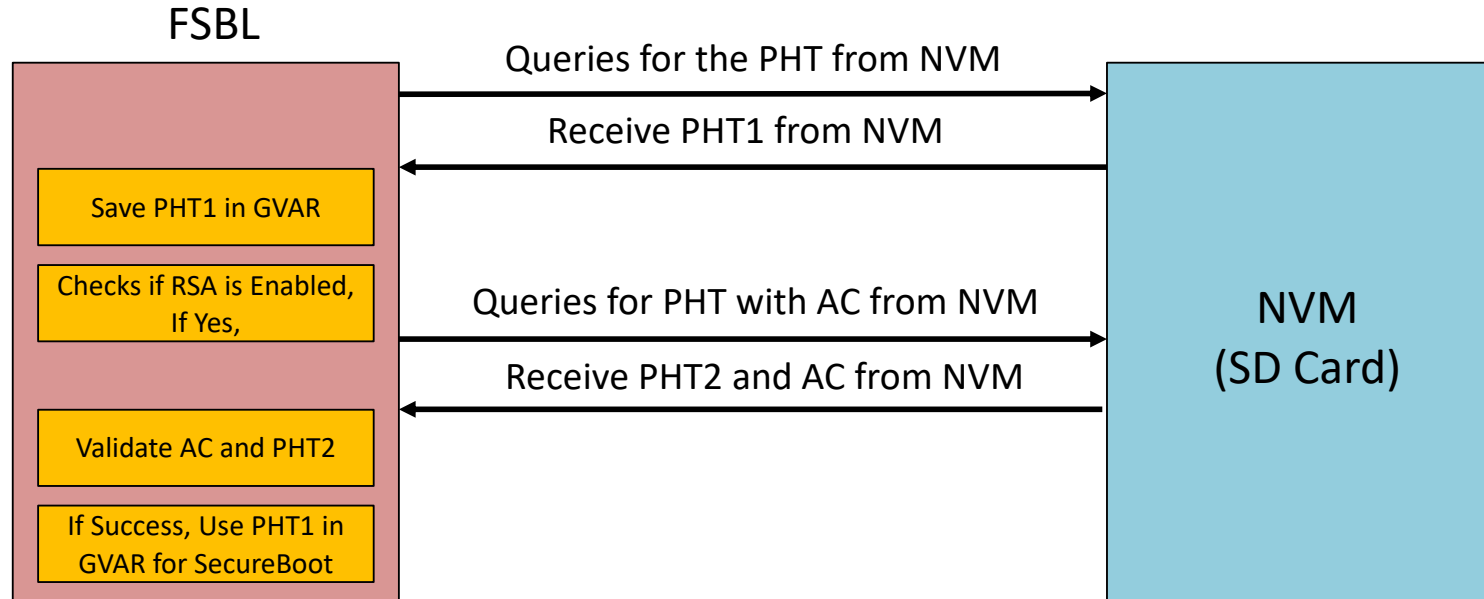


# Outline

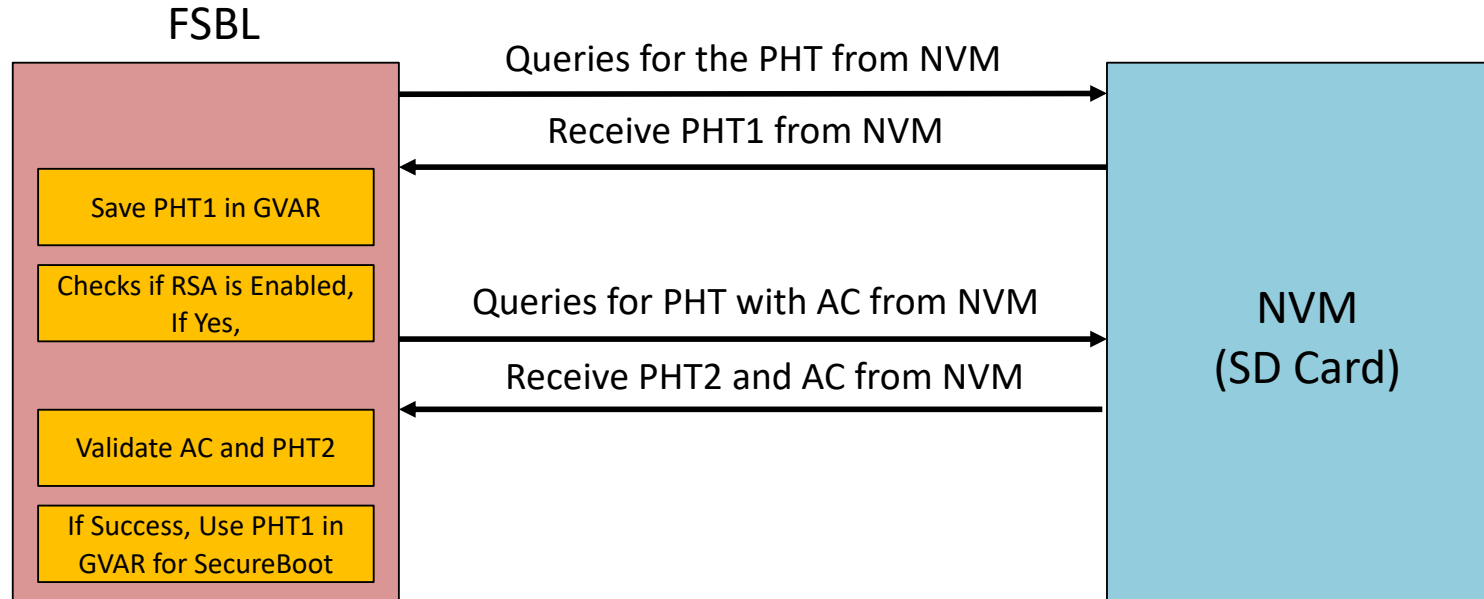
- ❑ Introduction
- ❑ RSA Authentication Attack on Zynq-7000
  - ❑ Background: Attack Model, Secure Boot and RSA Authentication
  - ❑ **Vulnerability in FSBL**
  - ❑ Attack Implementation: Using SD Card Switcher Board
- ❑ Starbleed on Zynq
  - ❑ Introduction and Working
  - ❑ Experimental Results
- ❑ Analyzing SD Card Data Transfer
- ❑ BootROM
  - ❑ Possible BootROM Vulnerabilities
  - ❑ PHT Transfer Analysis
  - ❑ BootROM Data Transfer Analysis
- ❑ Conclusion and Future Works



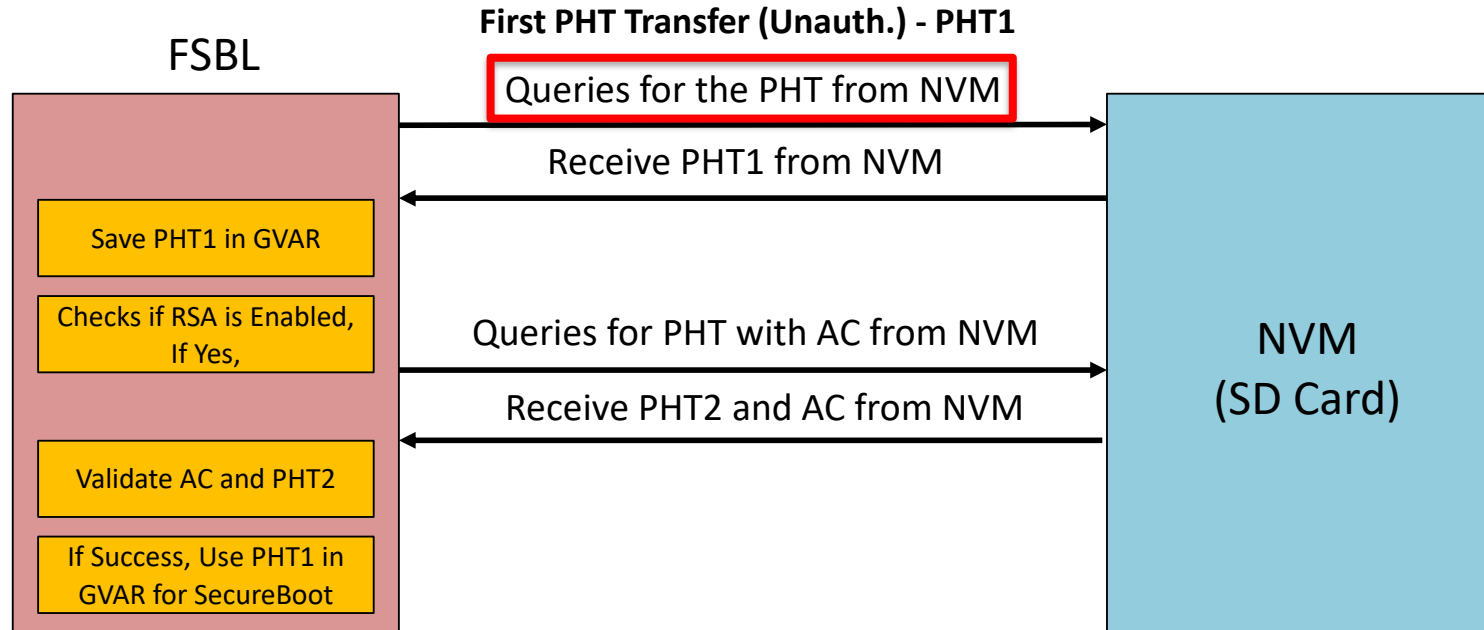
# PHT Authentication by FSBL



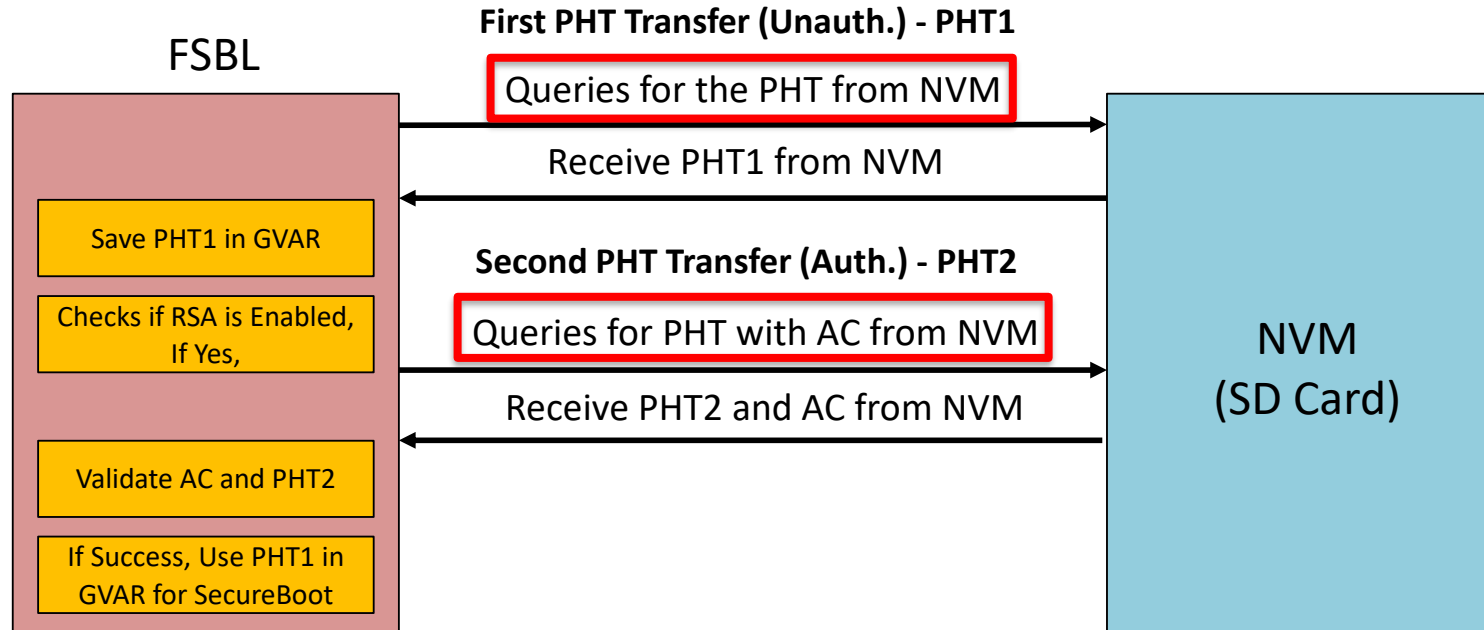
# PHT Authentication by FSBL



# PHT Authentication by FSBL



# PHT Authentication by FSBL



# Vulnerability: Redundant PHT Transfer

- ❑ **Key Observation:** FSBL authenticates **PHT2**, but uses **PHT1** for secure boot
- ❑ **Flaw:** FSBL uses the unauthenticated **PHT1** for secure boot.
- ❑ **Attack Idea:**
  - ❑ Present Tampered PHT1 to device
  - ❑ Present Valid PHT2 to device: PHT2 will be authenticated successfully by device
  - ❑ Flaw: Tampered PHT1 will be used for secure boot
  - ❑ Attack application mounted successfully on target!!



# RSA Attack using SD card Multiplexer

- ❑ **Requirement:** Manipulate data coming from SD card (PHT1  $\neq$  PHT2)
- ❑ **Idea:** Multiplexer to switch between two SD cards (SD Card 1 and SD Card 2)
- ❑ **Attack Steps:**
  - ❑ SD Card 1 sends tampered PHT1
  - ❑ Switch!!!
  - ❑ SD Card 2 sends valid PHT2
  - ❑ Device boots based on tampered PHT1
  - ❑ Attack application loaded from SD Card 2
- ❑ **Caveat:** Switch should be done oblivious to the target Zynq device
- ❑ **Solution:** We designed an SD card switcher board



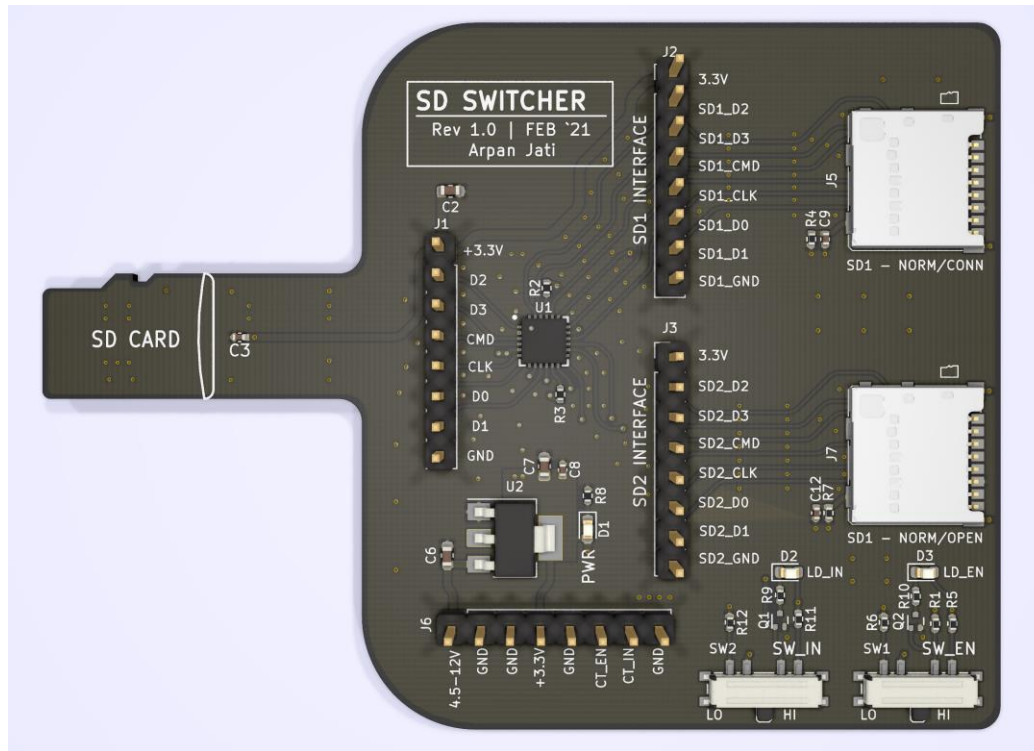


# Outline

- ❑ Introduction
- ❑ RSA Authentication Attack on Zynq-7000
  - ❑ Background: Attack Model, Secure Boot and RSA Authentication
  - ❑ Vulnerability in FSBL
  - ❑ **Attack Implementation: Using SD Card Switcher Board**
- ❑ Starbleed on Zynq
  - ❑ Introduction and Working
  - ❑ Experimental Results
- ❑ Analyzing SD Card Data Transfer
- ❑ BootROM
  - ❑ Possible BootROM Vulnerabilities
  - ❑ PHT Transfer Analysis
  - ❑ BootROM Data Transfer Analysis
- ❑ Conclusion and Future Works



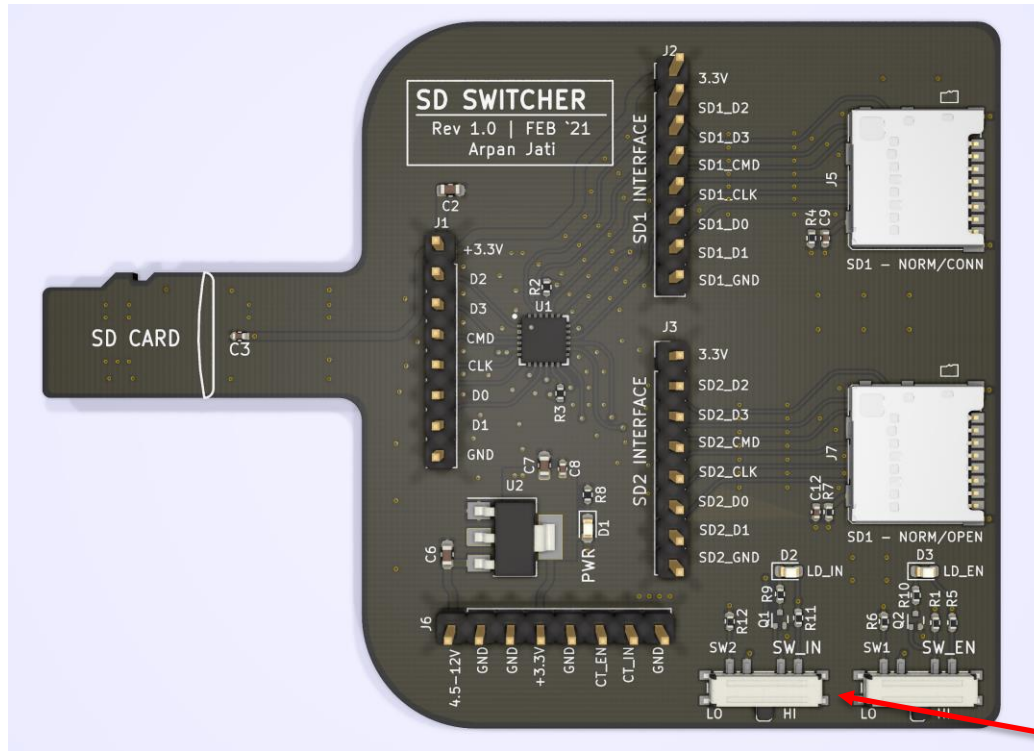
# SD Card Switcher Board:



SD Card 1

SD Card 2

# SD Card Switcher Board:

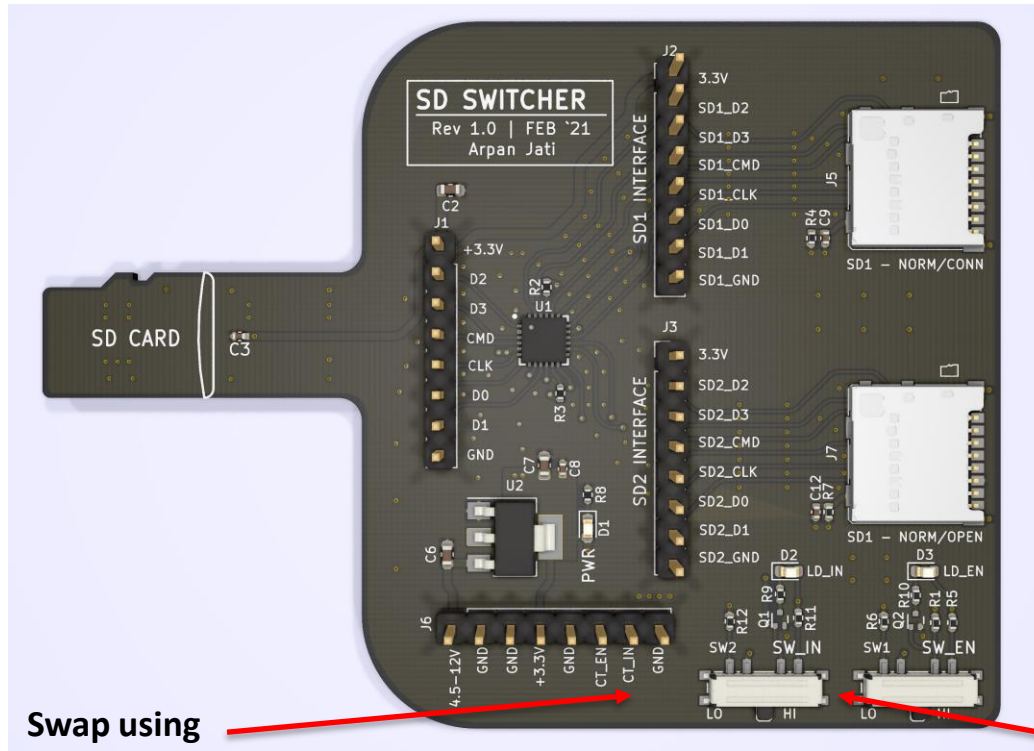


SD Card 1

SD Card 2

Manual  
Switch for  
SD card swap

# SD Card Switcher Board:



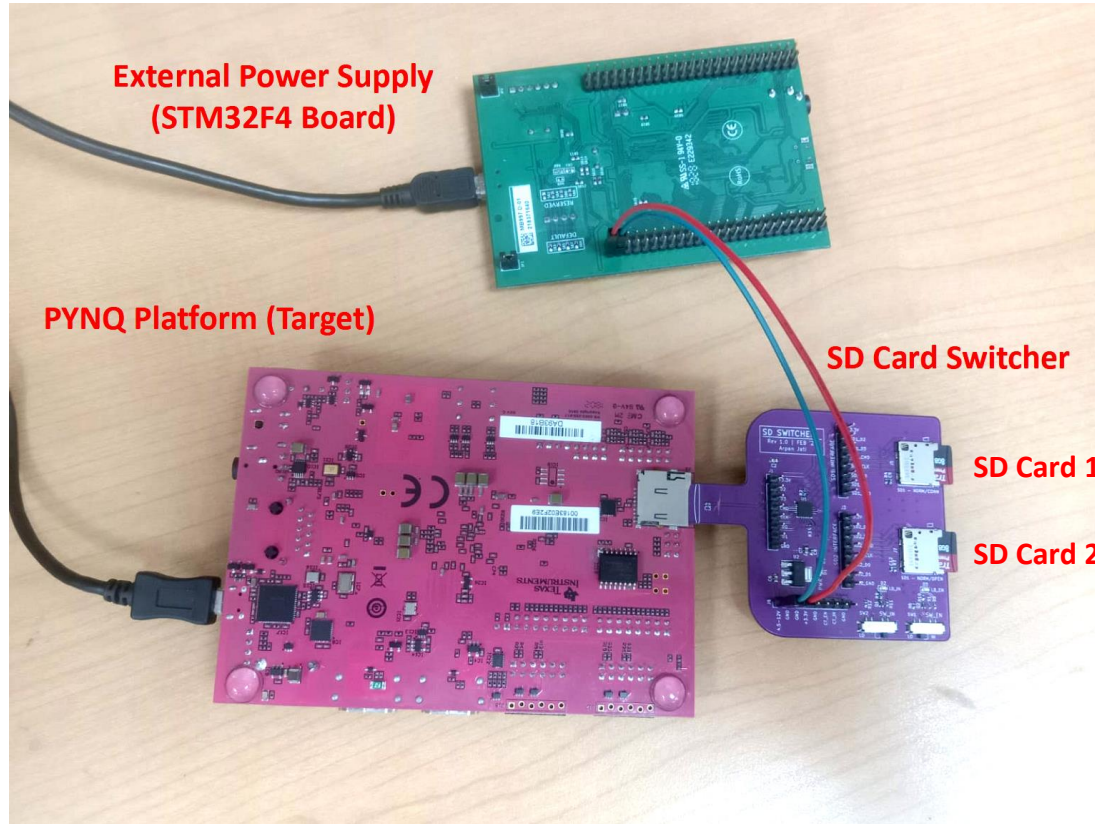
SD Card 1

SD Card 2

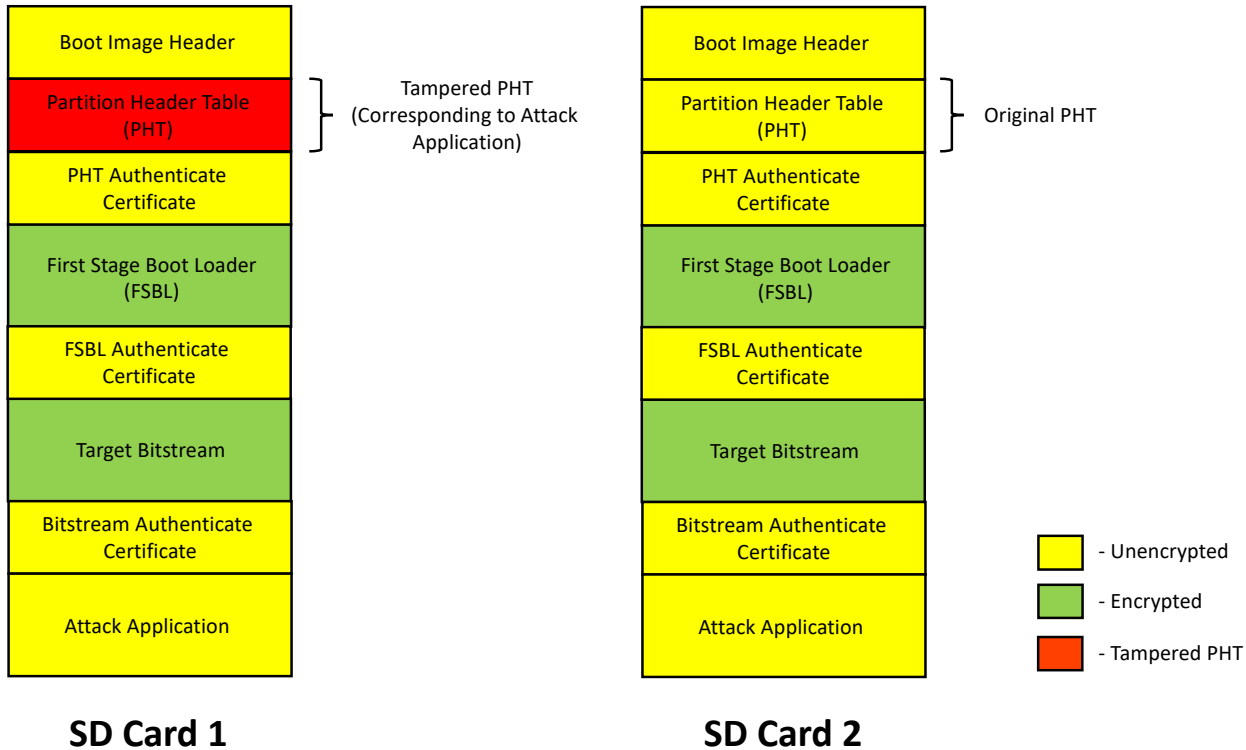
Swap using  
GPIO Trigger

Manual  
Switch for  
SD card swap

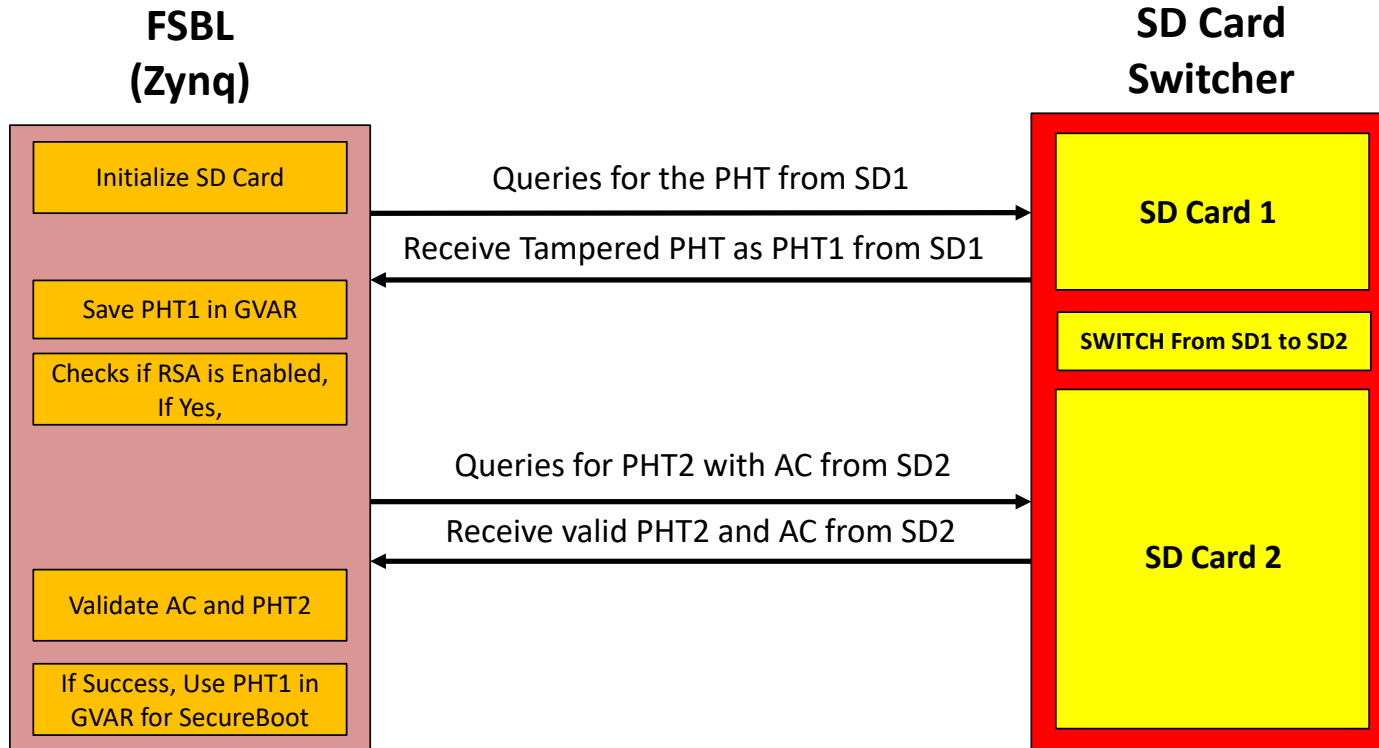
# Attack Setup:



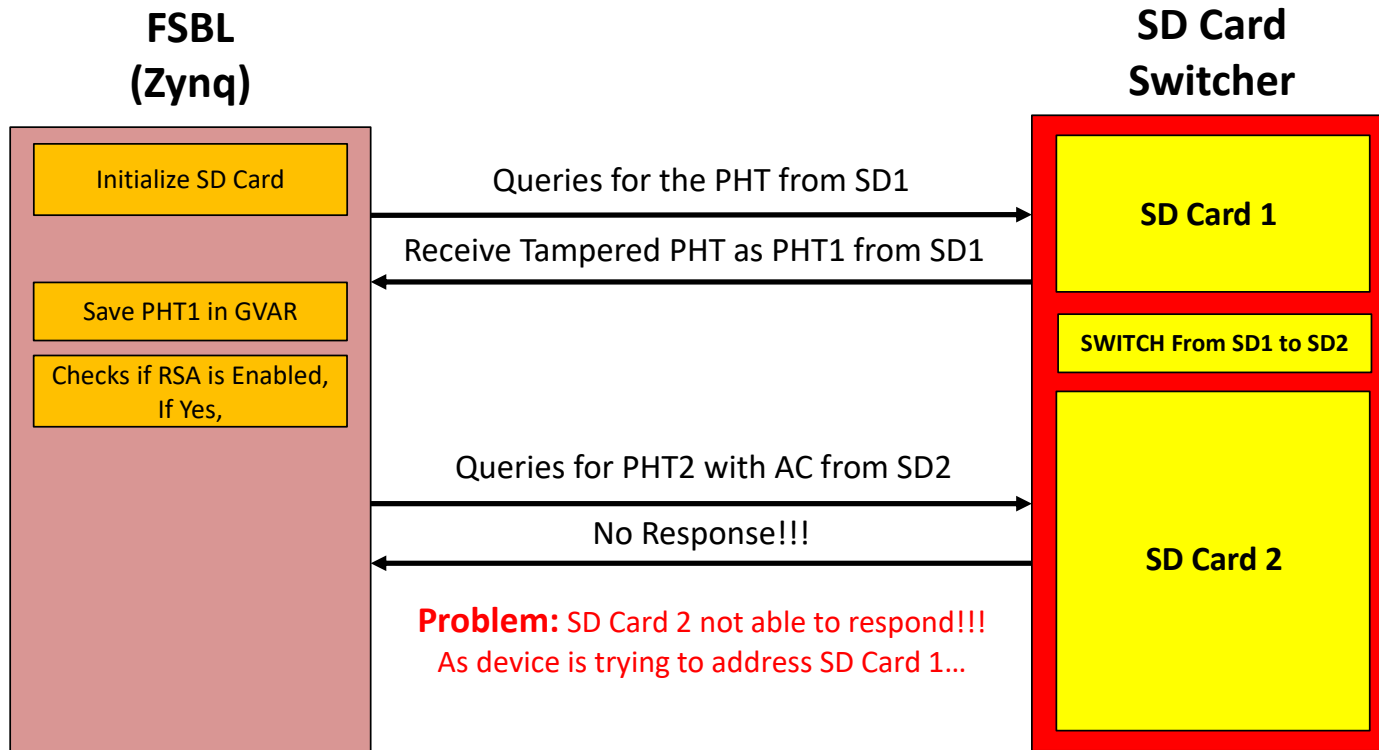
# Attack Setup:



# Attack (Expected):

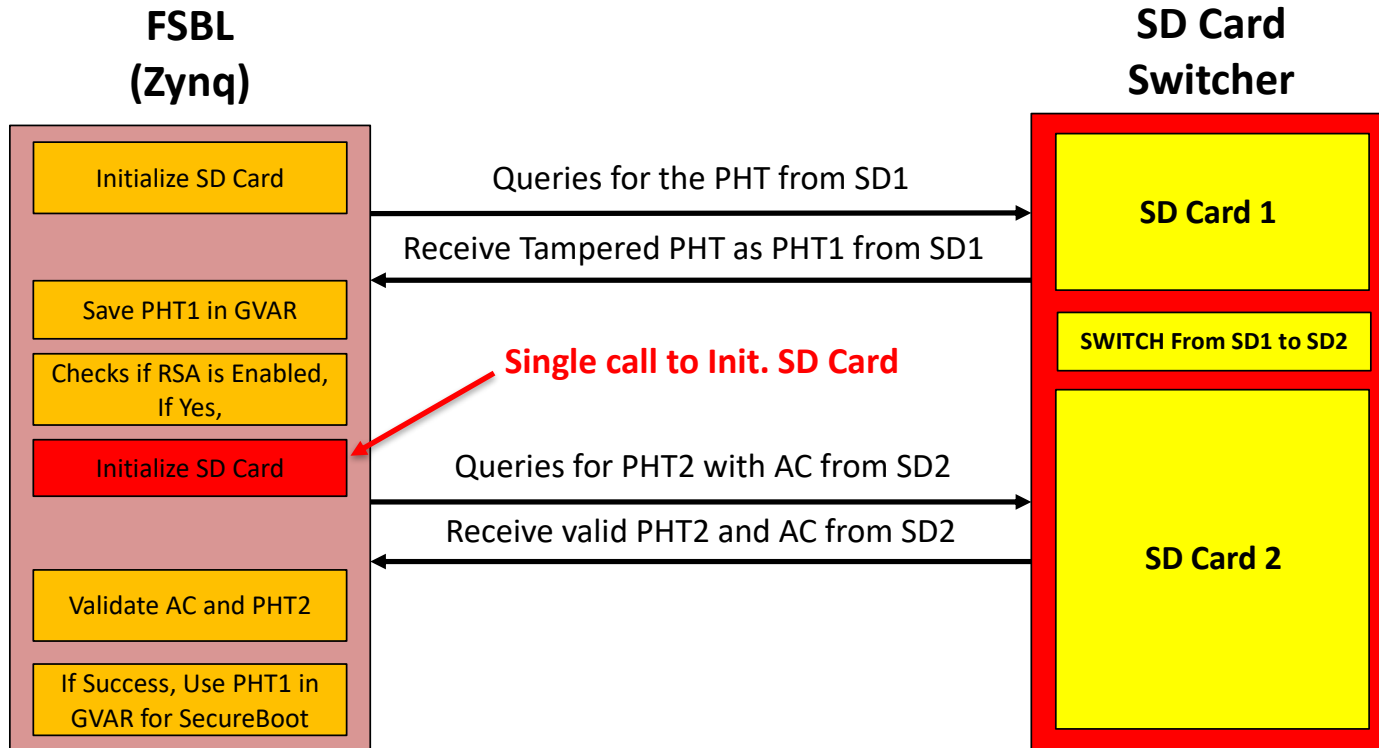


# Attack (Observed):



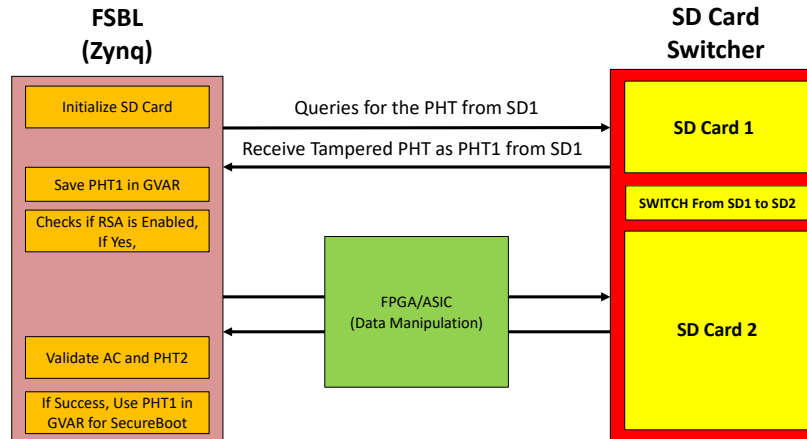


# Attack:

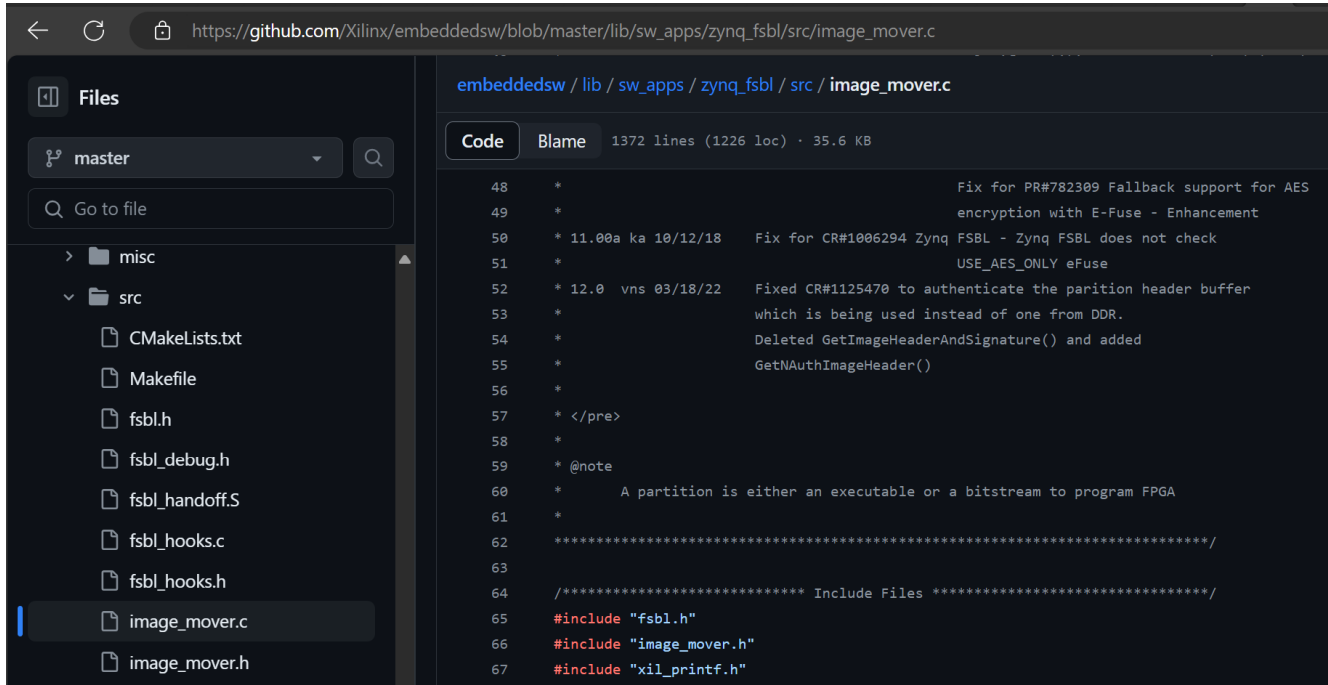


# The Attack, practicality and CVE

- ❑ Attack requires to make a very minor modification in the FSBL: To Initialize the SD Card 2
  - ❑ Very close to practical, but not yet fully practical
  - ❑ Modification in FSBL not related to the identified vulnerability !!!
- ❑ However, our attack concretely demonstrates presence of flaw in FSBL.
  - ❑ **Xilinx confirmed its a serious issue and patched the vulnerability**
  - ❑ **They also published a CVE 2022/23822 (Dated April 27, 2024)**
- ❑ For real-world practical attack, we need to design specialized hardware (using ASIC/FPGA)



# Software Patch by Xilinx



```
embeddedsw / lib / sw_apps / zynq_fsbl / src / image_mover.c
Code Blame 1372 lines (1226 loc) · 35.6 KB
48 *                               Fix for PR#782309 Fallback support for AES
49 *                               encryption with E-Fuse - Enhancement
50 * 11.00a ka 10/12/18   Fix for CR#1006294 Zynq FSBL - Zynq FSBL does not check
51 *                               USE_AES_ONLY eFuse
52 * 12.0 vns 03/18/22   Fixed CR#1125470 to authenticate the partition header buffer
53 *                               which is being used instead of one from DDR.
54 *                               Deleted GetImageHeaderAndSignature() and added
55 *                               GetNAuthImageHeader()
56 *
57 * </pre>
58 *
59 * @note
60 *   A partition is either an executable or a bitstream to program FPGA
61 *
62 * *****/
63 *
64 * /***** Include Files *****/
65 #include "fsbl.h"
66 #include "image_mover.h"
67 #include "xil_printf.h"
```

## Patch Note:

This patch fixes the secure vulnerability of partition header(PH) authentication, in existing code the actual buffer used and authenticated are different, this patch fixes the issue by considering the actual used buffer of partition header while calculating the SHA2 digest

March 25, 2022



# Outline

- ❑ Introduction
- ❑ RSA Authentication Attack on Zynq-7000
  - ❑ Background: Attack Model, Secure Boot and RSA Authentication
  - ❑ Vulnerability in FSBL
  - ❑ Attack Implementation: Using SD Card Switcher Board
- ❑ Starbleed on Zynq
  - ❑ **Introduction and Working**
  - ❑ Experimental Results
- ❑ Analyzing SD Card Data Transfer
- ❑ BootROM
  - ❑ Possible BootROM Vulnerabilities
  - ❑ PHT Transfer Analysis
  - ❑ BootROM Data Transfer Analysis
- ❑ Conclusion and Future Works



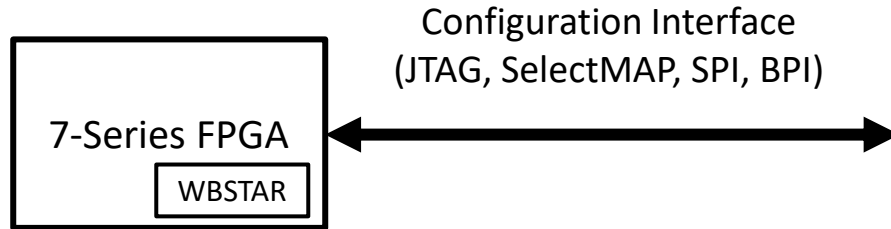
# Introduction to Starbleed on Zynq

- ❑ The starbleed attack introduced by Ender et.al on a standalone FPGA works in the following manner:



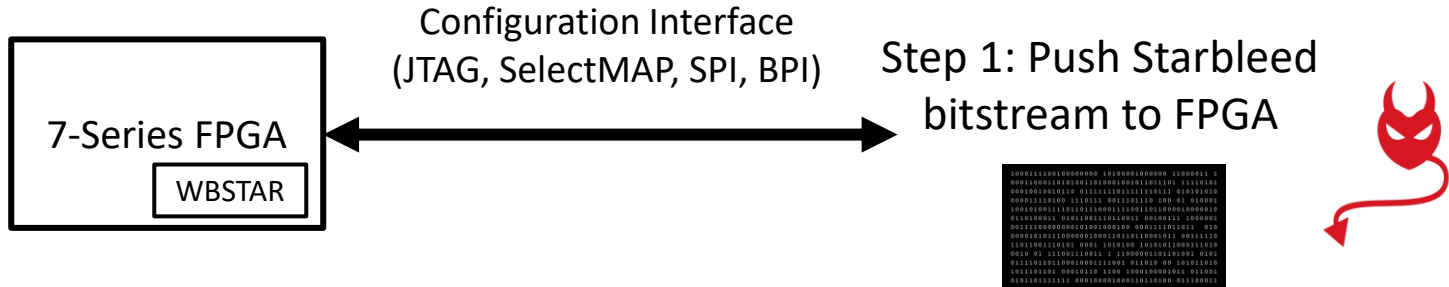
# Introduction to Starbleed on Zynq

- ❑ The starbleed attack introduced by Ender et.al on a standalone FPGA works in the following manner:



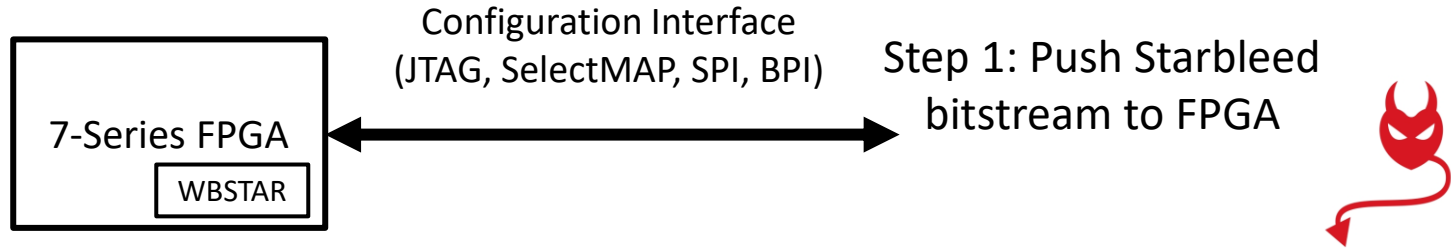
# Introduction to Starbleed on Zynq

- ❑ The starbleed attack introduced by Ender et.al on a standalone FPGA works in the following manner:



# Introduction to Starbleed on Zynq

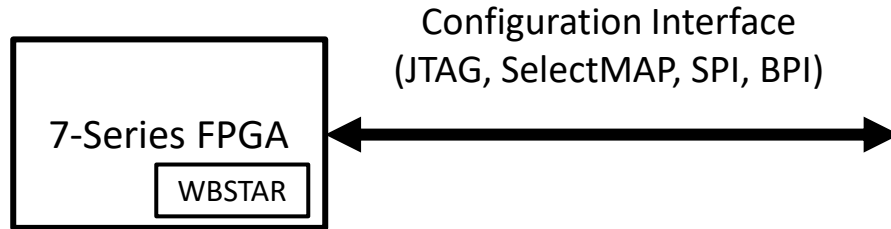
- ❑ The starbleed attack introduced by Ender et.al on a standalone FPGA works in the following manner:





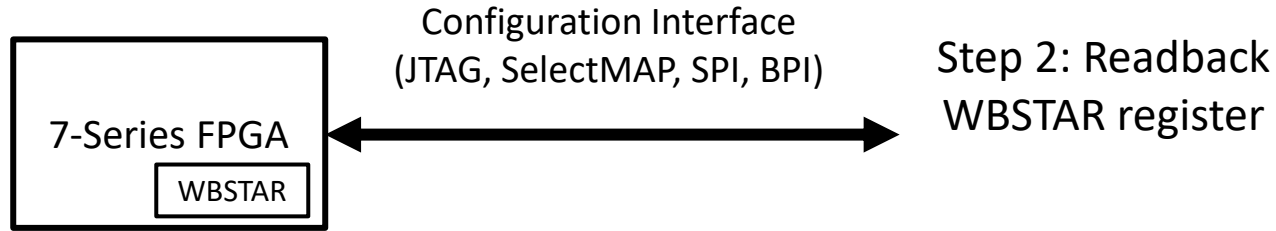
# Introduction to Starbleed on Zynq

- ❑ The starbleed attack introduced by Ender et.al on a standalone FPGA works in the following manner:



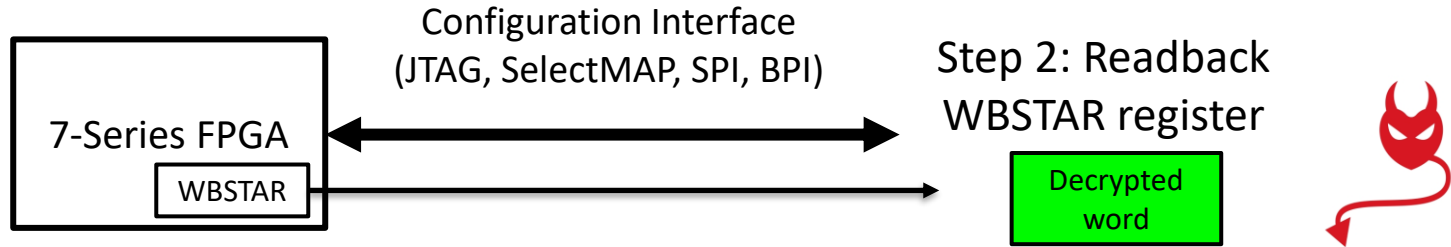
# Introduction to Starbleed on Zynq

- The starbleed attack introduced by Ender et.al on a standalone FPGA works in the following manner:



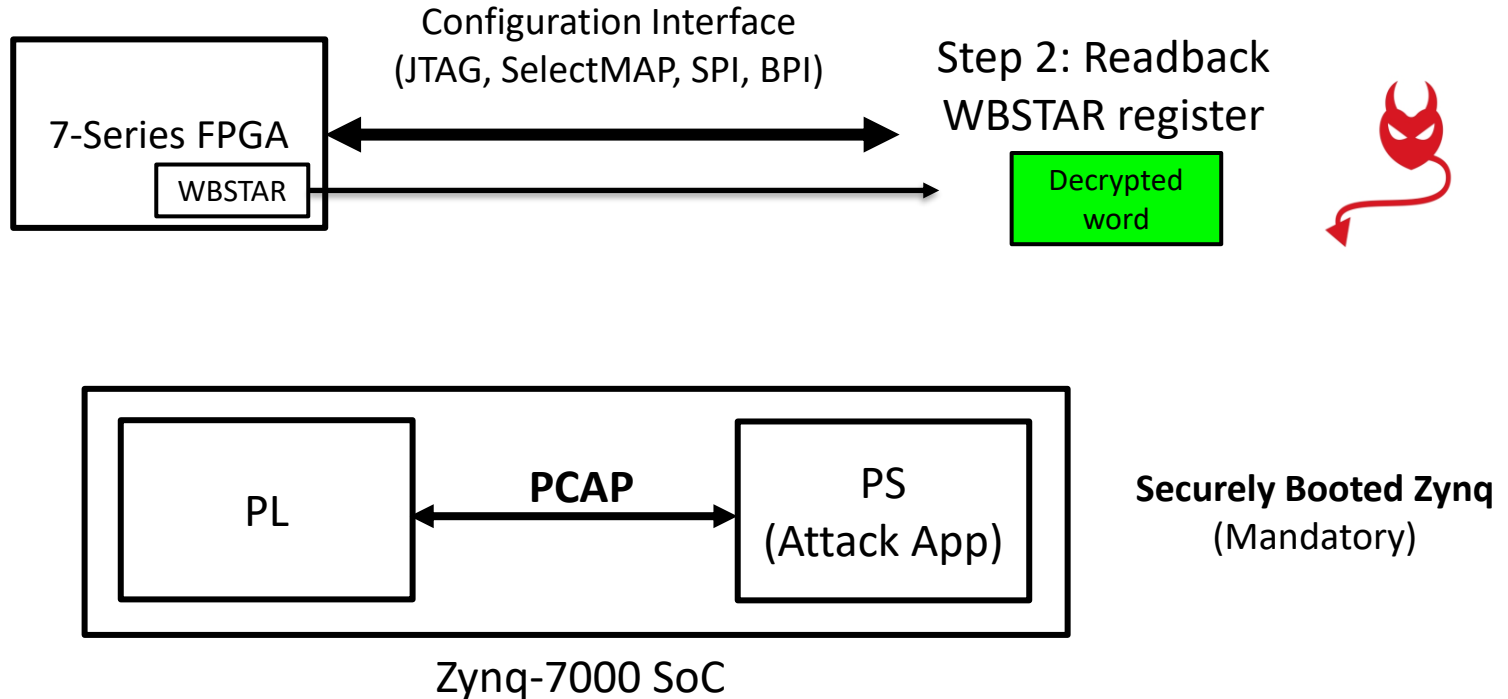
# Introduction to Starbleed on Zynq

- The starbleed attack introduced by Ender et.al on a standalone FPGA works in the following manner:



# Introduction to Starbleed on Zynq

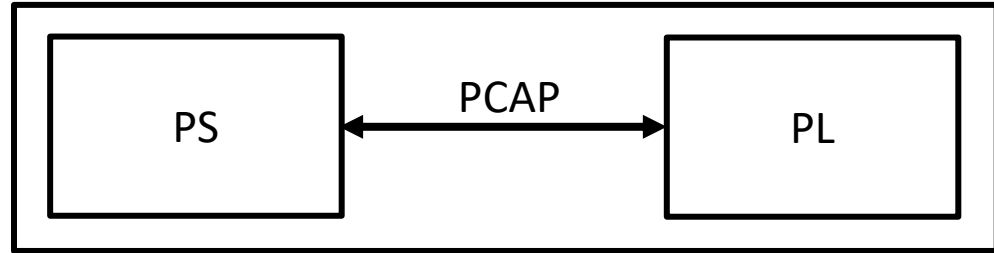
- The starbleed attack introduced by Ender et.al on a standalone FPGA works in the following manner:



# Attack Methodology: Starbleed on Zynq

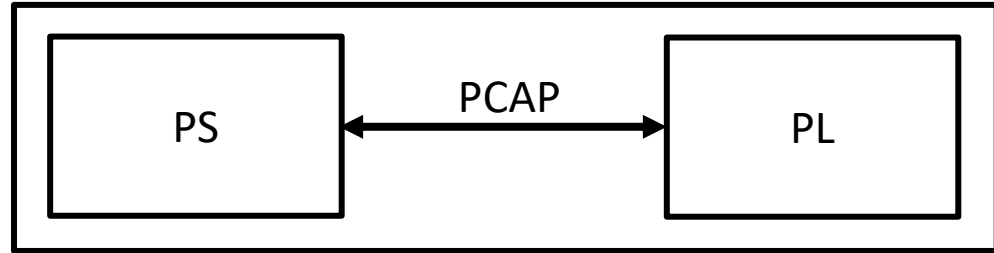


# Attack Methodology: Starbleed on Zynq

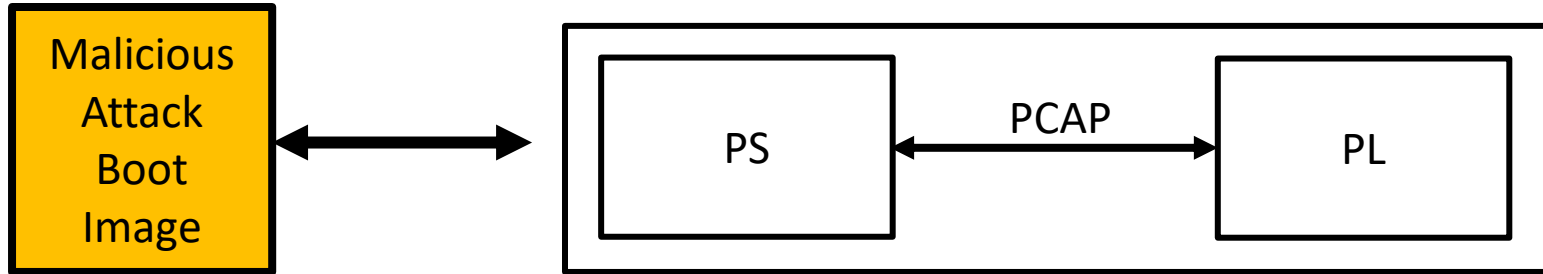


# Attack Methodology: Starbleed on Zynq

Malicious  
Attack  
Boot  
Image

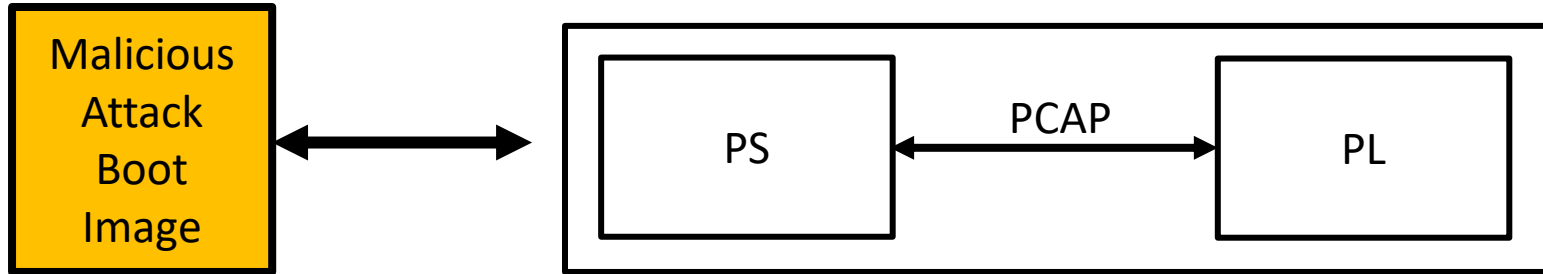


# Attack Methodology: Starbleed on Zynq

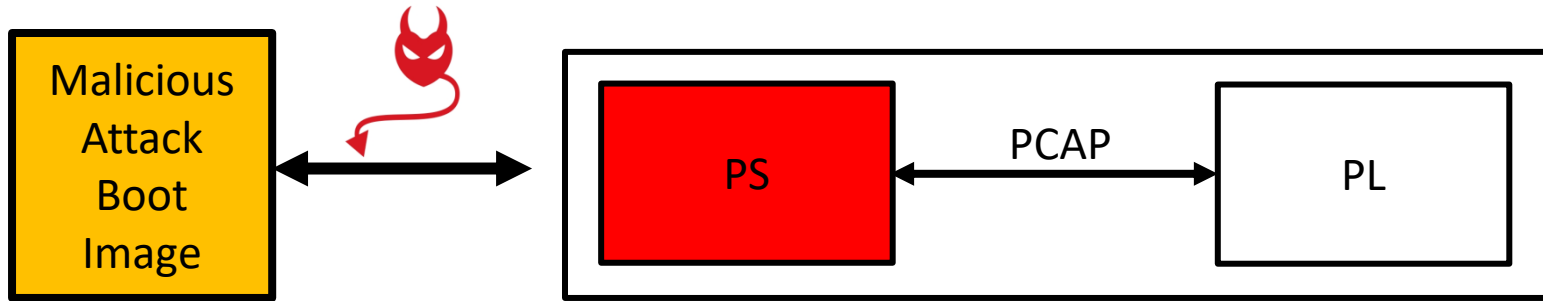




# Attack Methodology: Starbleed on Zynq

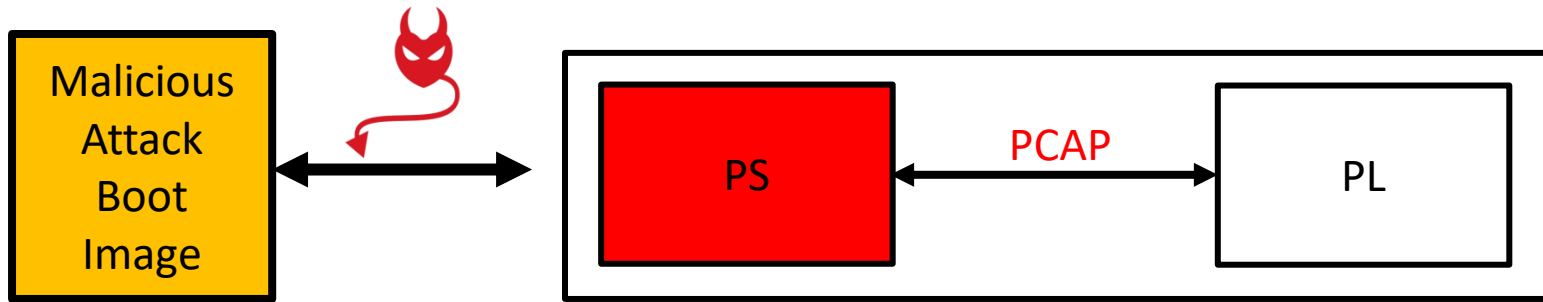


# Attack Methodology: Starbleed on Zynq



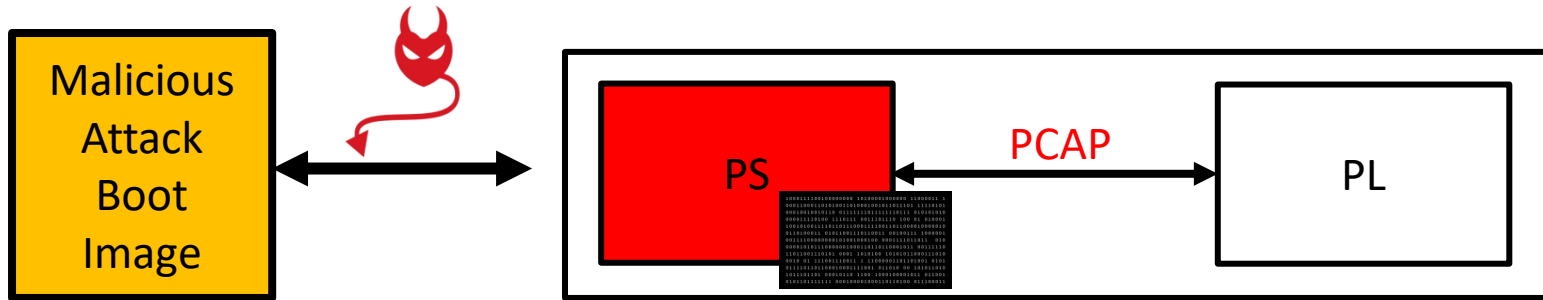
Secure boot **OK!!**

# Attack Methodology: Starbleed on Zynq



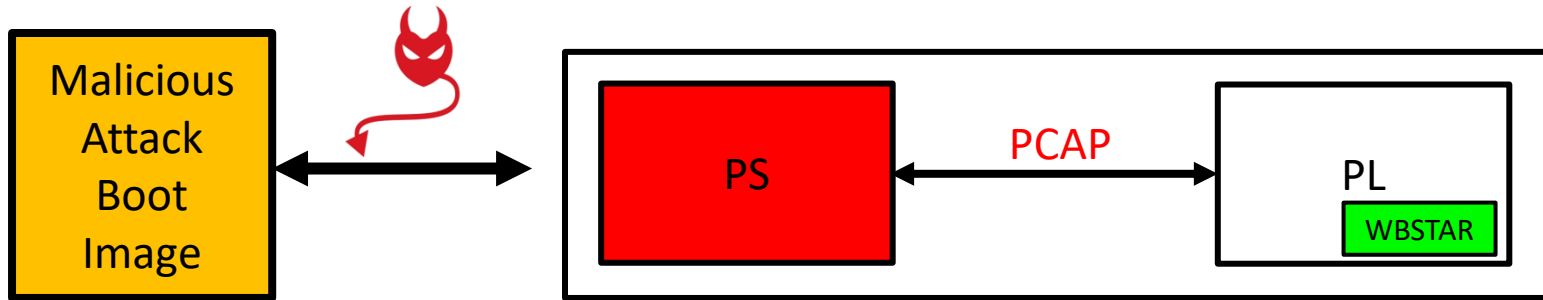
Secure boot **OK!!**

# Attack Methodology: Starbleed on Zynq



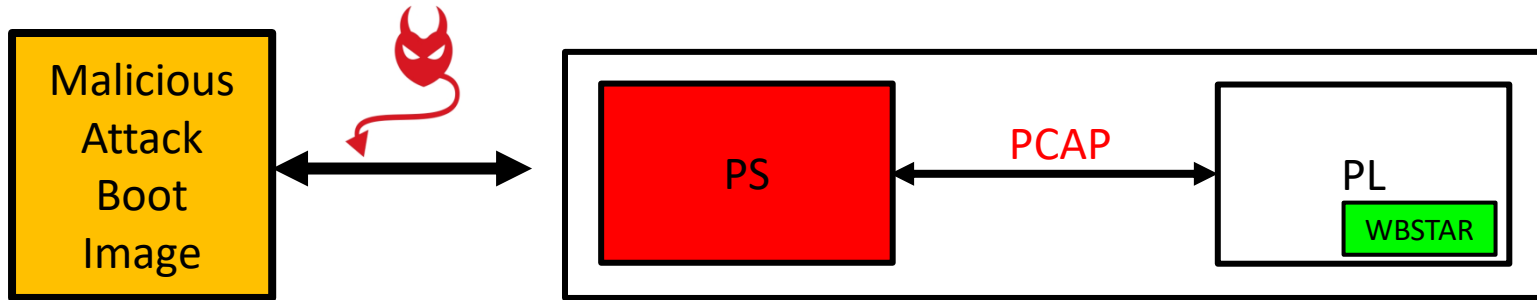
Secure boot **OK!!**

# Attack Methodology: Starbleed on Zynq



Secure boot **OK!!**

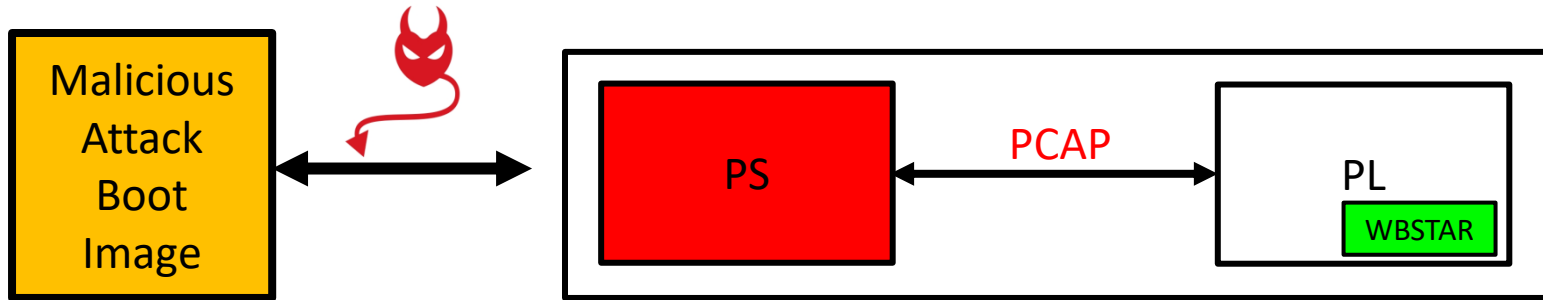
# Attack Methodology: Starbleed on Zynq



Secure boot **OK!!**

- ❑ Main Challenges:

# Attack Methodology: Starbleed on Zynq

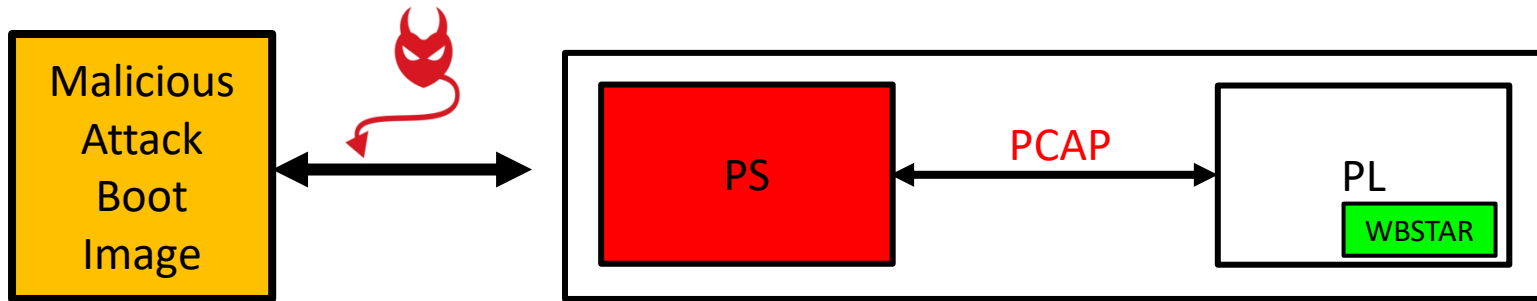


Secure boot **OK!!**

## ❑ Main Challenges:

- ❑ Implementing a working Starbleed attack on standalone FPGA

# Attack Methodology: Starbleed on Zynq



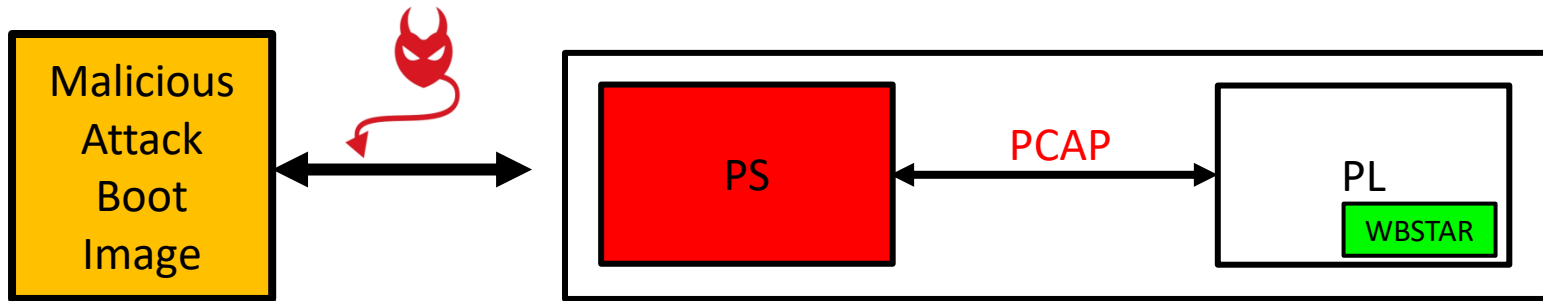
Secure boot **OK!!**

## ❑ Main Challenges:

- ❑ Implementing a working Starbleed attack on standalone FPGA
- ❑ Construction of secure attack boot image (without knowledge of key)



# Attack Methodology: Starbleed on Zynq

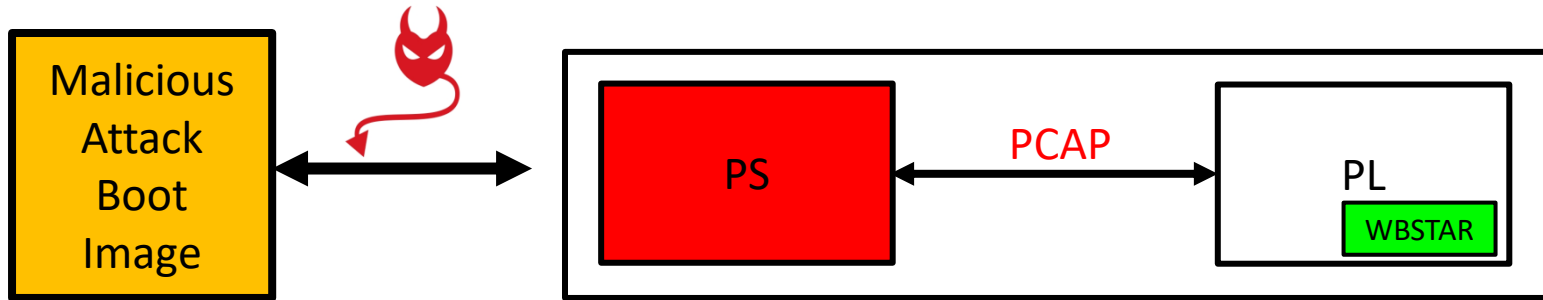


Secure boot **OK!!**

## ❑ Main Challenges:


- ❑ Implementing a working Starbleed attack on standalone FPGA
- ❑ Construction of secure attack boot image (without knowledge of key)
- ❑ Development of attack application to carry out the starbleed attack (PCAP interface)

# Attack Methodology: Starbleed on Zynq

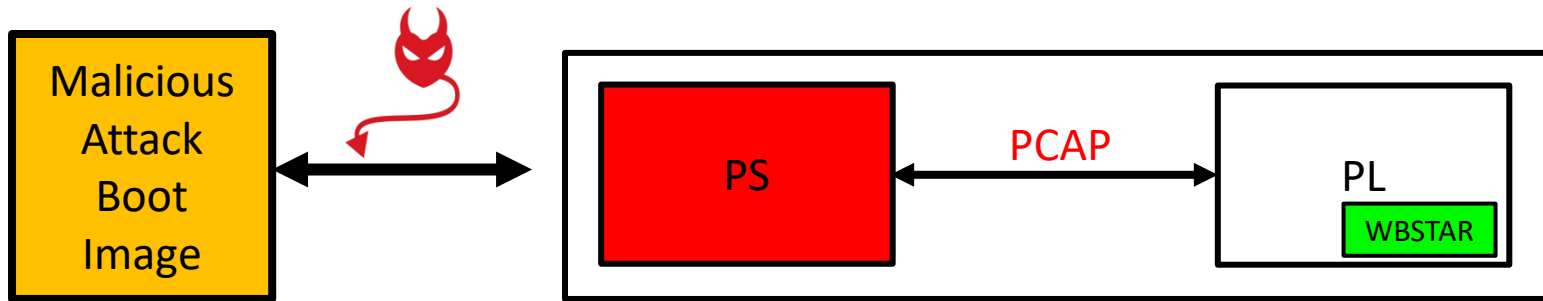


Secure boot **OK!!**

## ❑ Main Challenges:



- ❑ Implementing a working Starbleed attack on standalone FPGA 
- ❑ Construction of secure attack boot image (without knowledge of key)
- ❑ Development of attack application to carry out the starbleed attack (PCAP interface)

# Attack Methodology: Starbleed on Zynq

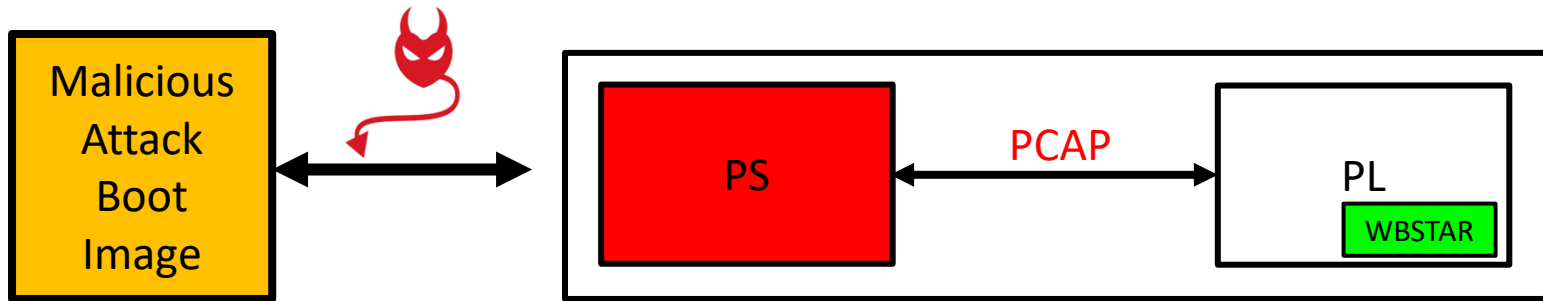


Secure boot **OK!!**

## ❑ Main Challenges:



- ❑ Implementing a working Starbleed attack on standalone FPGA 
- ❑ Construction of secure attack boot image (without knowledge of key) 
- ❑ Development of attack application to carry out the starbleed attack (PCAP interface)

# Attack Methodology: Starbleed on Zynq



Secure boot **OK!!**

## ❑ Main Challenges:

- ❑ Implementing a working Starbleed attack on standalone FPGA 
- ❑ Construction of secure attack boot image (without knowledge of key) 
- ❑ Development of attack application to carry out the starbleed attack (PCAP interface)

# Starbleed Bitstream Creation: Tool

SelectMAP Programmer & Bitstream Browser | SAKURA-X | March 2023 | Arpan Jati

FILE TOOLS

I/O Inspector Exporter

Binary File C:\Users\Arpan\Downloads\starbleed\_attack\_newKey.bin

NKY File C:\Users\Arpan\Downloads\all.nky

Recovery Data

Position 0 Value (Hex) 00000000    Attack Bitstream  Force Fault  With Config Trailer  Byte Swapped

Bitstream Browser

	Decrypted Bitstream			Fault Locations
0	FF FF FF FF X	FF FF FF FF X	FF FF FF FF X	FF FF FF FF X
16	FF FF FF FF X	FF FF FF FF X	FF FF FF FF X	FF FF FF FF X
32	00 00 00 BB X	11 22 00 44 X	FF FF FF FF X	FF FF FF FF X
48	AA 99 55 66 X	20 00 00 00 T1 NOP	30 03 E0 01 T1 W BSPI 1	00 00 00 0B X
64	30 00 80 01 T1 W CMD 1	00 00 00 12 CMD BSPI_READ	20 00 00 00 T1 NOP	30 00 C0 01 T1 W MASK 1
80	00 00 00 40 X	30 00 A0 01 T1 W CTL0 1	00 00 00 40 X	30 01 C0 01 T1 W COR1 1
96	00 00 00 00 X	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP
112	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP
128	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP
144	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP	30 01 60 04 T1 W CBC 4
160	00 01 02 03 X	04 05 06 07 X	08 09 0A 0B X	0C 0D 0E 0F X
176	30 03 40 01 T1 W DWC 1	00 00 01 28 X		
184	F6 77 84 35	D2 53 90 11	EE 6F AC 2D	CA 4B 88 09
200	36 36 36 36	36 36 36 36	36 36 36 36	36 36 36 36

Ready Binary Recovery File does NOT exist.

- The tool works by adding faults in the required places (done manually).
- Removing extra superfluous code (makes the attack much faster)

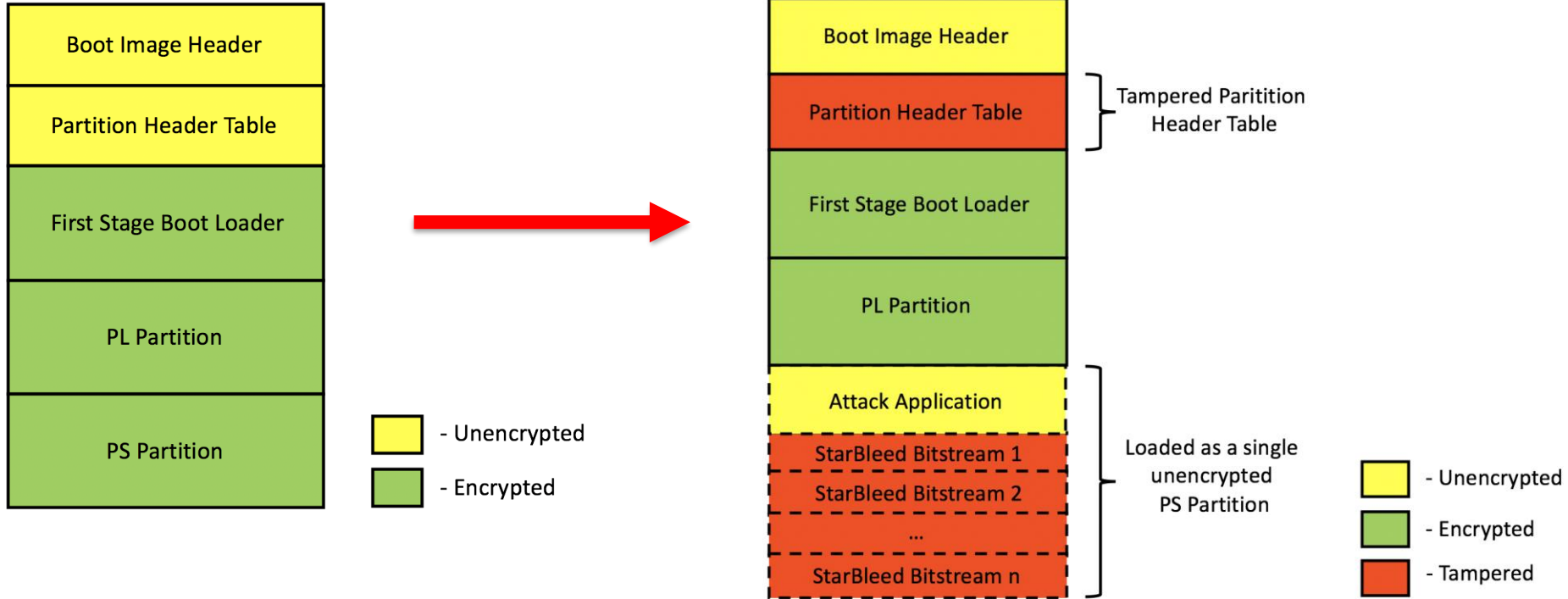
# Starbleed Bitstream Creation: Tool

Decrypted Bitstream				
0	FF FF FF FF X	FF FF FF FF X	FF FF FF FF X	FF FF FF FF X
16	FF FF FF FF X	FF FF FF FF X	FF FF FF FF X	FF FF FF FF X
32	00 00 00 BB X	11 22 00 44 X	FF FF FF FF X	FF FF FF FF X
48	AA 99 55 66 X	20 00 00 00 T1 NOP	30 03 E0 01 T1 W BSP1 1	00 00 00 0B X
64	30 00 80 01 T1 W CMD 1	00 00 00 12 CMD BSP1_READ	20 00 00 00 T1 NOP	30 00 C0 01 T1 W MASK 1
80	00 00 00 40 X	30 00 A0 01 T1 W CTL0 1	00 00 00 40 X	30 01 C0 01 T1 W COR1 1
96	00 00 00 00 X	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP
112	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP
128	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP
144	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP	30 01 60 04 T1 W CBC 4
160	00 01 02 03 X	04 05 06 07 X	08 09 0A 0B X	0C 0D 0E 0F X
176	30 03 40 01 T1 W DMC 1	00 00 01 28 X		
184	F6 77 B4 35	D2 53 90 11	EE 6F AC 2D	CA 4B 88 09
200	36 36 36 36	36 36 36 36	36 36 36 36	36 36 36 36
216	36 37 34 35	32 33 30 31	3E 3F 3C 3D	3A 3B 38 39
232	3F DB 4B 21	BC C2 24 FF	48 76 2C 94	D0 FE 8B CF

Decrypted Bitstream				
232	3F DB 4B 21	BC C2 24 FF	48 76 2C 94	D0 FE 8B CF
248	30 02 20 01 T1 W TIMER 1	00 00 00 00 X	30 02 00 26 T1 W WBSTAR 38	00 00 00 00 X
264	30 00 80 01 T1 W CMD 1	00 00 00 00 CMD NULL	20 00 00 00 T1 NOP	30 00 80 01 T1 W CMD 1
280	00 00 00 07 CMD RCRC	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP	30 02 60 01 T1 W [U/D] 1
296	00 00 00 00 X	30 01 20 01 T1 W COR0 1	02 00 3F E5 X	30 01 C0 01 T1 W COR1 1
312	00 00 00 00 X	30 01 80 01 T1 W IDCODE 1	03 64 C0 93 X	30 00 80 01 T1 W CMD 1
328	00 00 00 09 CMD SWITCH	20 00 00 00 T1 NOP	30 00 C0 01 T1 W MASK 1	FF FF FF FF X
344	30 00 A0 01 T1 W CTL0 1	00 00 05 49 X	30 00 C0 01 T1 W MASK 1	FF FF FF FF X
360	30 03 00 01 T1 W CTL1 1	00 01 E3 D0 X	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP
376	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP	20 00 00 00 T1 NOP
392	EF C2 FE A8 X	FE 10 7C BA X	F4 03 4F E4 X	D3 3F 44 62 X
408	30 00 80 01 T1 W CMD 1	00 00 00 01 CMD WCFG	20 00 00 00 T1 NOP	30 02 00 08 T1 W WBSTAR 8
424	50 19 85 70 T2 W 1672560	00 00 00 00 X	00 00 00 00 X	00 00 00 04 X
440	21 A1 16 93 T1 NOP	DC D9 DC 06 X	7F 99 A0 56 X	48 D1 E1 AC T2 R 13754796
456	00 00 00 00 X	00 00 00 00 X	00 00 00 00 X	00 00 00 00 X
472	00 00 00 00 X	00 00 00 00 X	00 00 00 00 X	00 00 00 00 X



# Building Secure Boot Image from Victim Boot Image



# Building Attack Application:

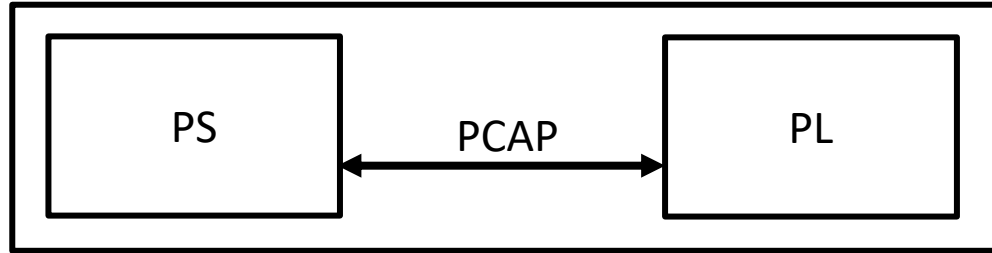
- ❑ **Main Tasks:**
  - ❑ **Task 1:** Fetch starbleed bitstreams from DDR memory and push them to PL through PCAP interface.
  - ❑ **Task 2:** Perform readback of the WBSTAR register through the PCAP interface.
- ❑ We utilized Xilinx Software Development Kit (**XSDK**) from Xilinx to develop the attack application running on the PS.
- ❑ We are able to successfully use the PCAP interface to configure PL with valid bitstreams.



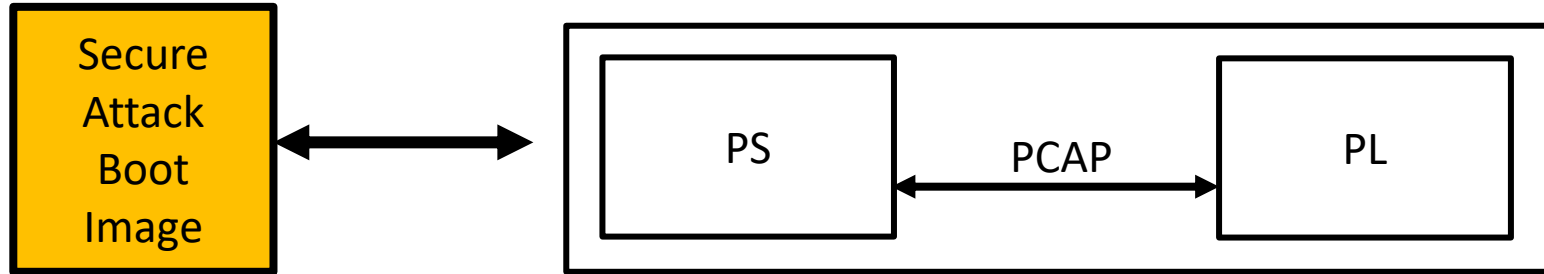


# Experimental Observations: tampered PL

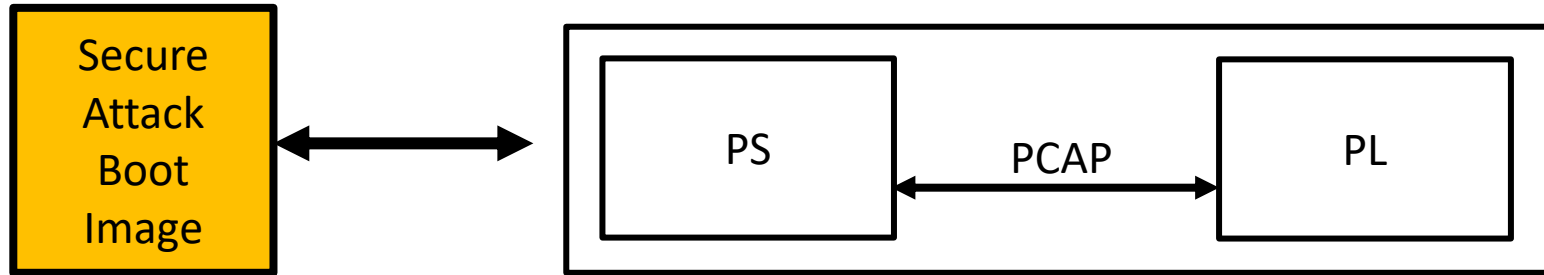
Secure  
Attack  
Boot  
Image



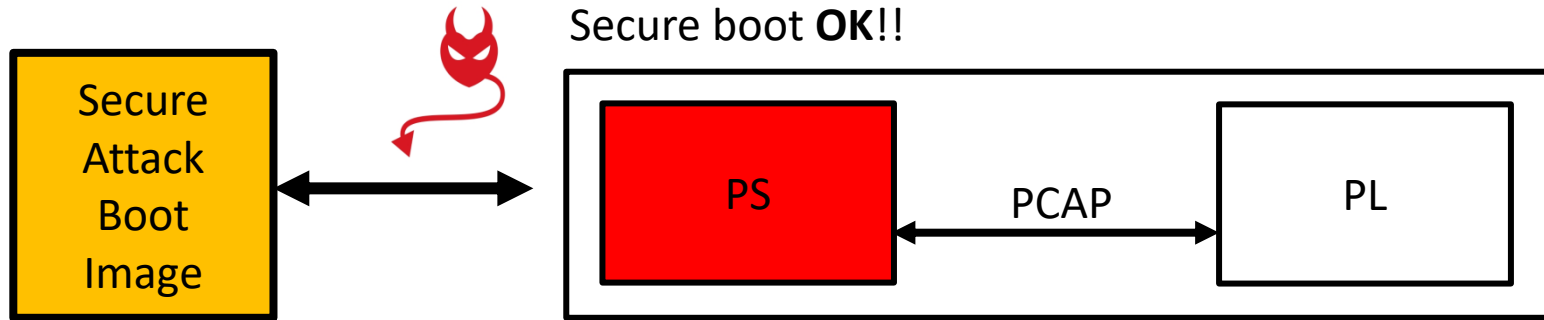
# Experimental Observations: tampered PL



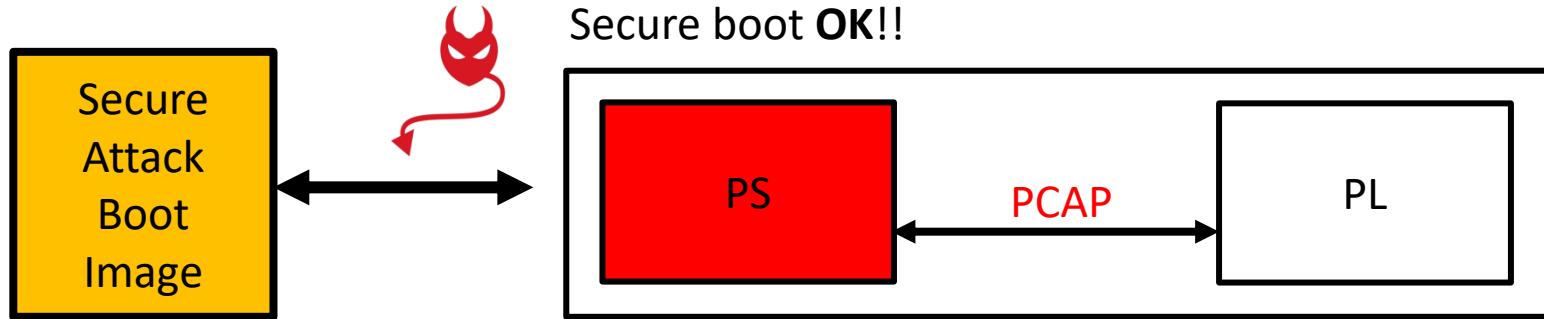
# Experimental Observations: tampered PL



# Experimental Observations: tampered PL

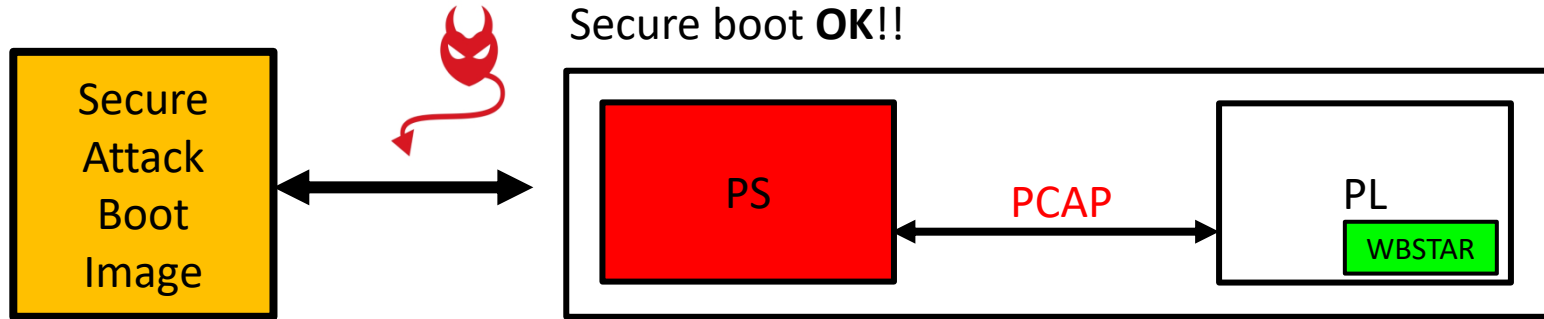


# Experimental Observations: tampered PL

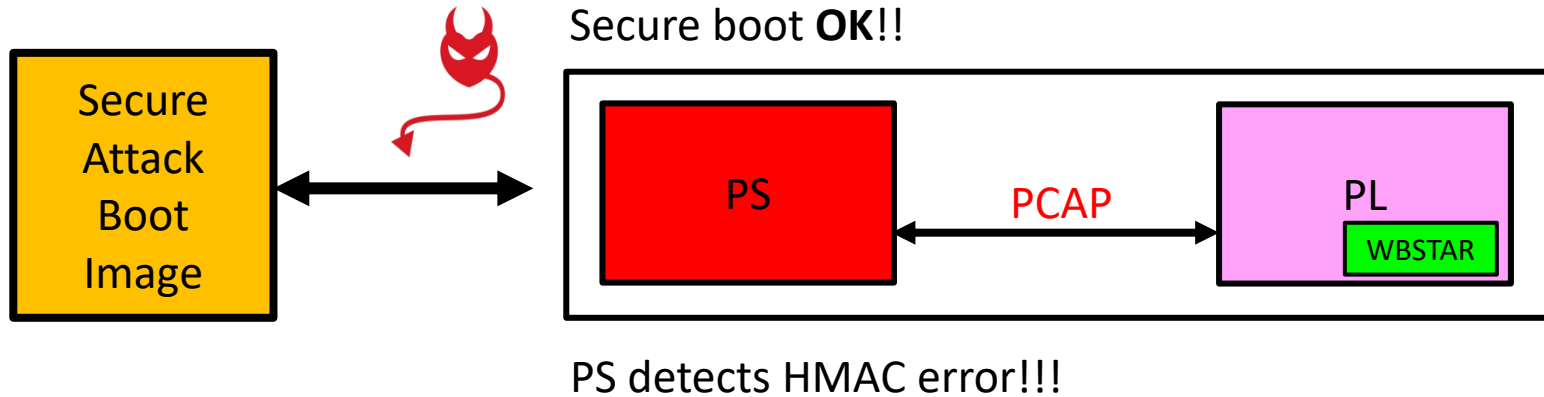




# Experimental Observations: tampered PL

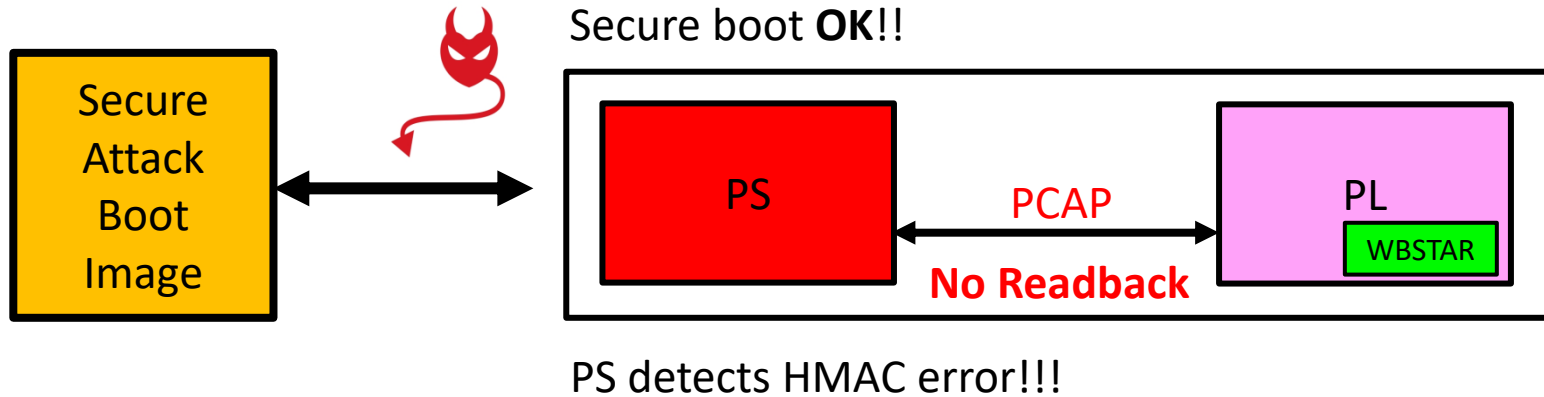


# Experimental Observations: tampered PL

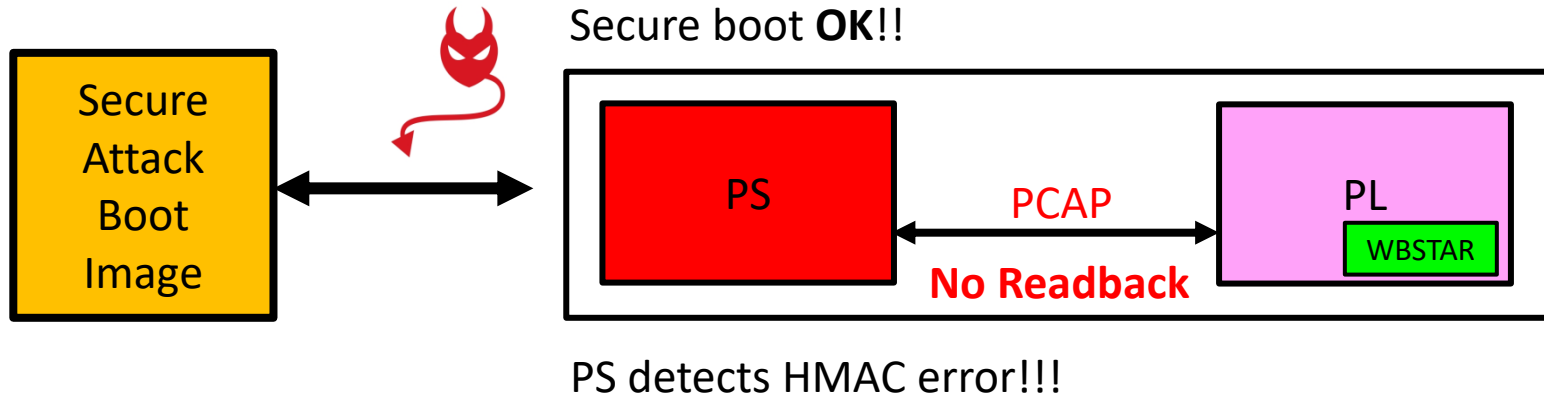




# Experimental Observations: tampered PL

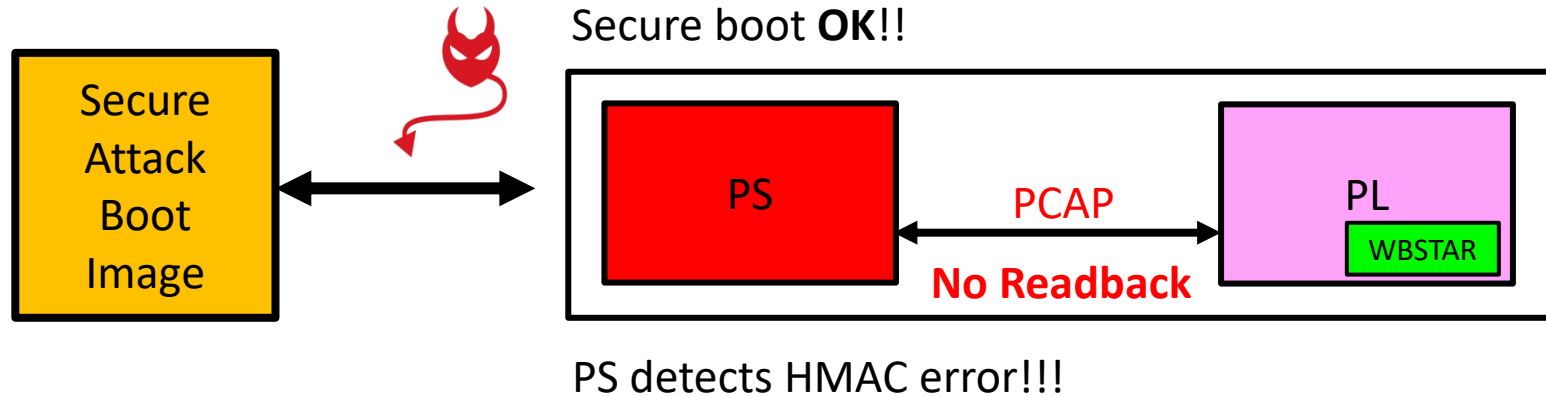


# Experimental Observations: tampered PL



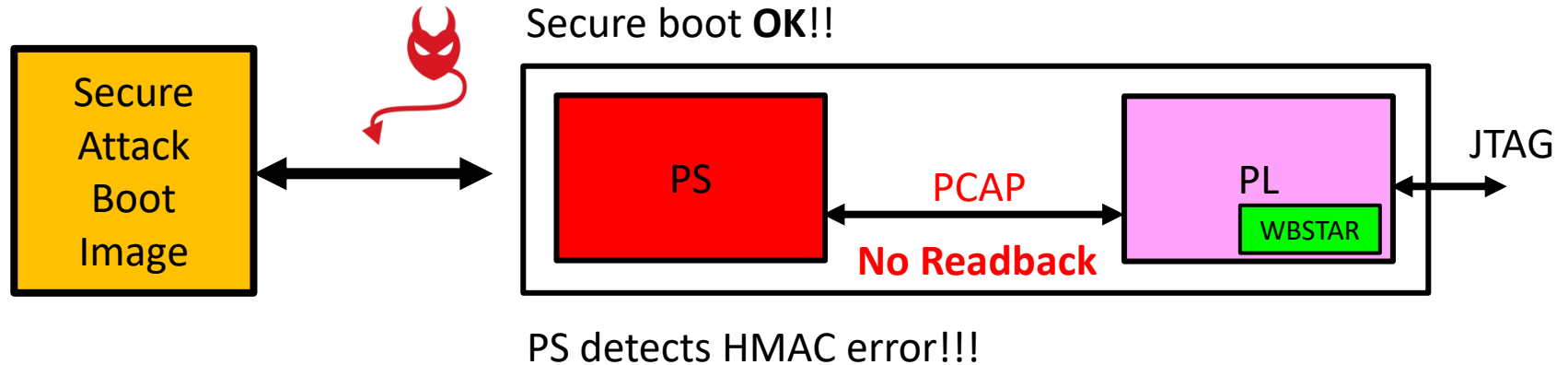
- ❑ Readback through PCAP is only possible when the PL is fully configured with a valid bitstream.

# Experimental Observations: tampered PL



- ❑ Readback through PCAP is only possible when the PL is fully configured with a valid bitstream.
- ❑ So, we attempted to use the JTAG interface to perform readback.

# Experimental Observations: tampered PL



- ❑ Readback through PCAP is only possible when the PL is fully configured with a valid bitstream.
- ❑ So, we attempted to use the JTAG interface to perform readback.
- ❑ We were able to read the correct decrypted word in the WBSTAR register through JTAG interface.

# Can we rely on JTAG for Starbleed Attack

- ❑ JTAG is an external interface which might not be exposed on a deployed device.
- ❑ There is a fuse control bit that can permanently disable JTAG: **XSK\_EFUSEPK\_DISABLE\_JTAG\_CHAIN**



# Can we rely on JTAG for Starbleed Attack

- ❑ JTAG is an external interface which might not be exposed on a deployed device.
- ❑ There is a fuse control bit that can permanently disable JTAG: `XSK_EFUSEPK_DISABLE_JTAG_CHAIN`
- ❑ Can we perform the attack using just the PCAP interface (**without relying on external interface**)?

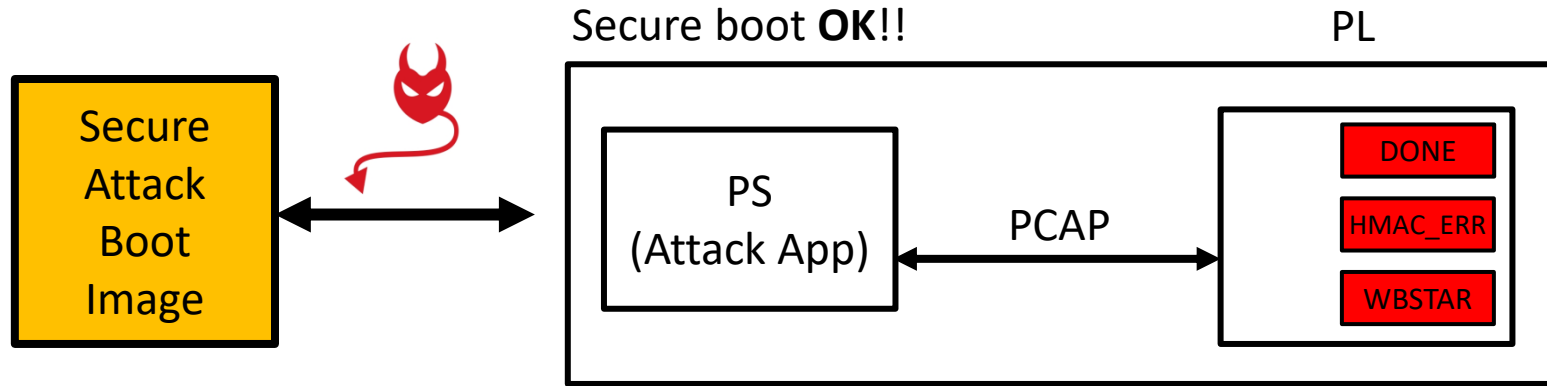


# Can we rely on JTAG for Starbleed Attack

- ❑ JTAG is an external interface which might not be exposed on a deployed device.
- ❑ There is a fuse control bit that can permanently disable JTAG: `XSK_EFUSEPK_DISABLE_JTAG_CHAIN`
- ❑ Can we perform the attack using just the PCAP interface (**without relying on external interface**)?
- ❑ We identified a “hack” to perform the starbleed attack only using the PCAP interface.



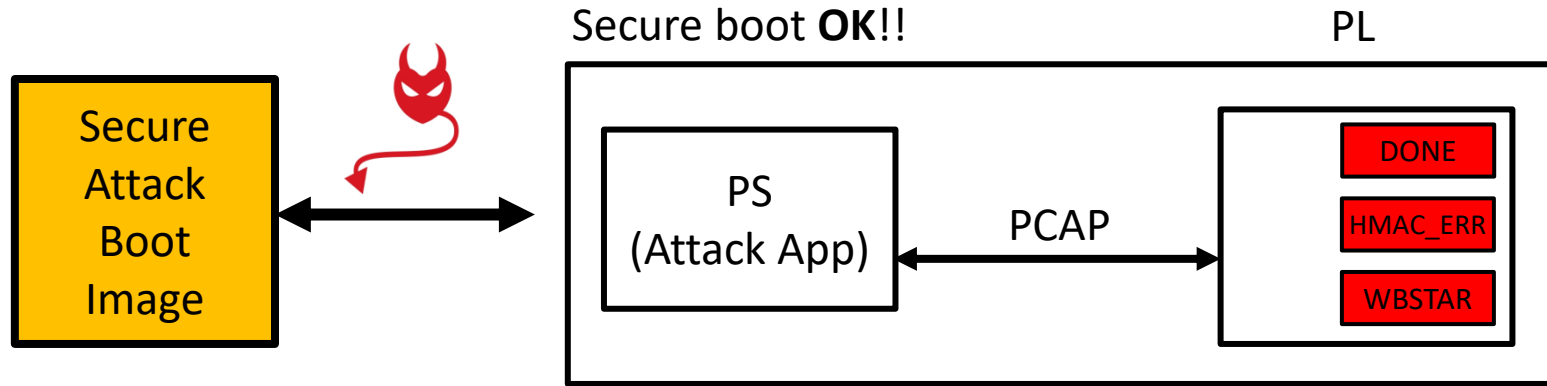
# Starbleed Attack (using PCAP)



- ❑ **Limitation:** PCAP readback possible only when PL is properly configured (PL Done High)

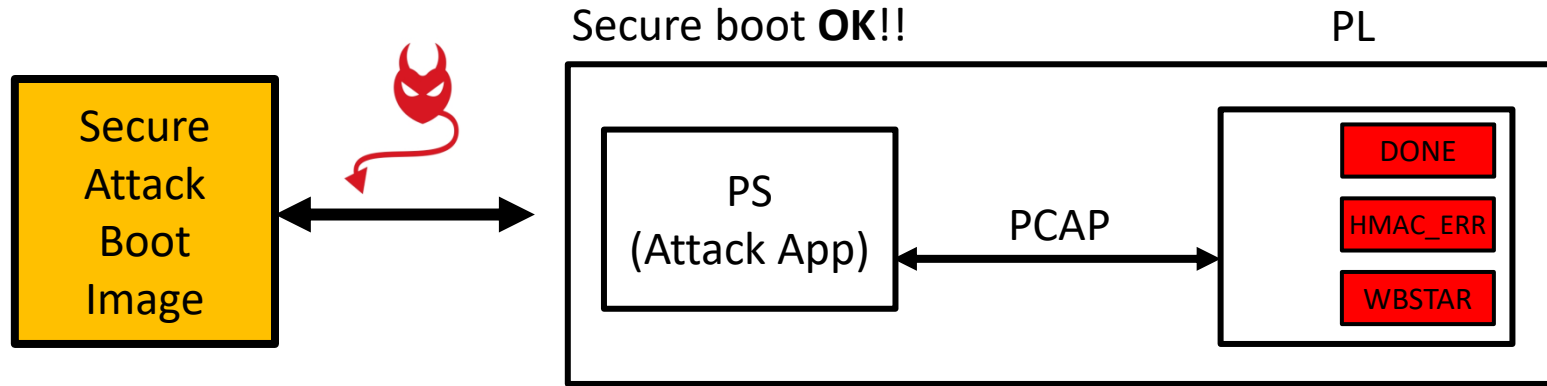


# Starbleed Attack (using PCAP)



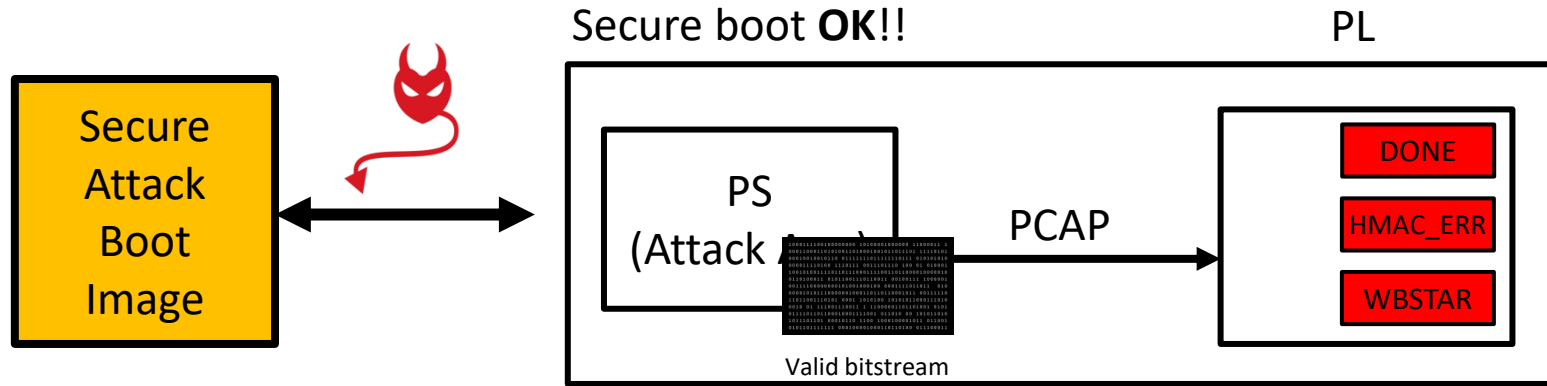
- ❑ **Limitation:** PCAP readback possible only when PL is properly configured (PL Done High)
- ❑ **Attack Steps:**

# Starbleed Attack (using PCAP)



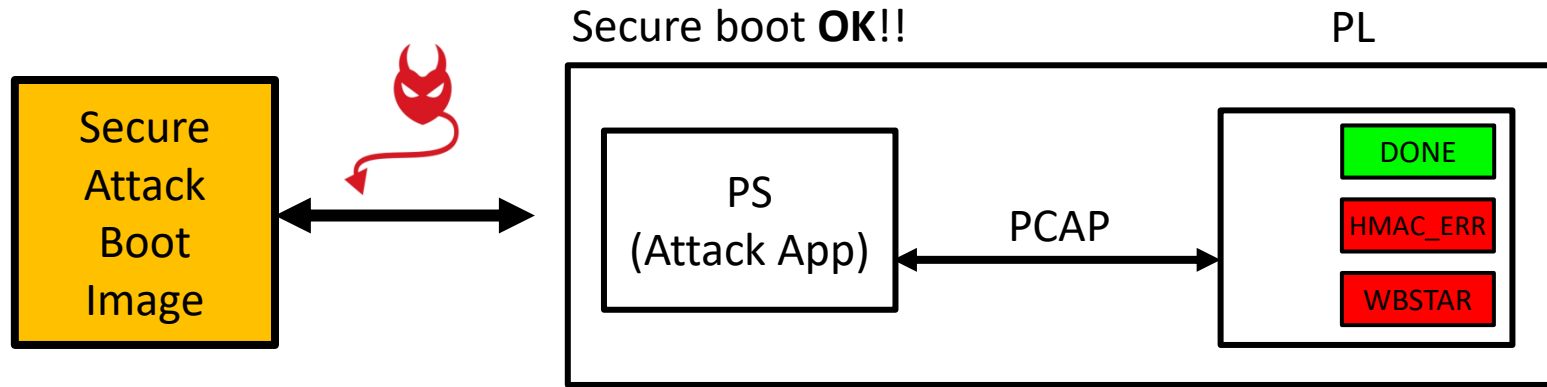
- ❑ **Limitation:** PCAP readback possible only when PL is properly configured (PL Done High)
- ❑ **Attack Steps:**
  - Step-1: We push a valid bitstream (encrypted bitstream in victim image) and fully configure the PL (PL DONE high)

# Starbleed Attack (using PCAP)



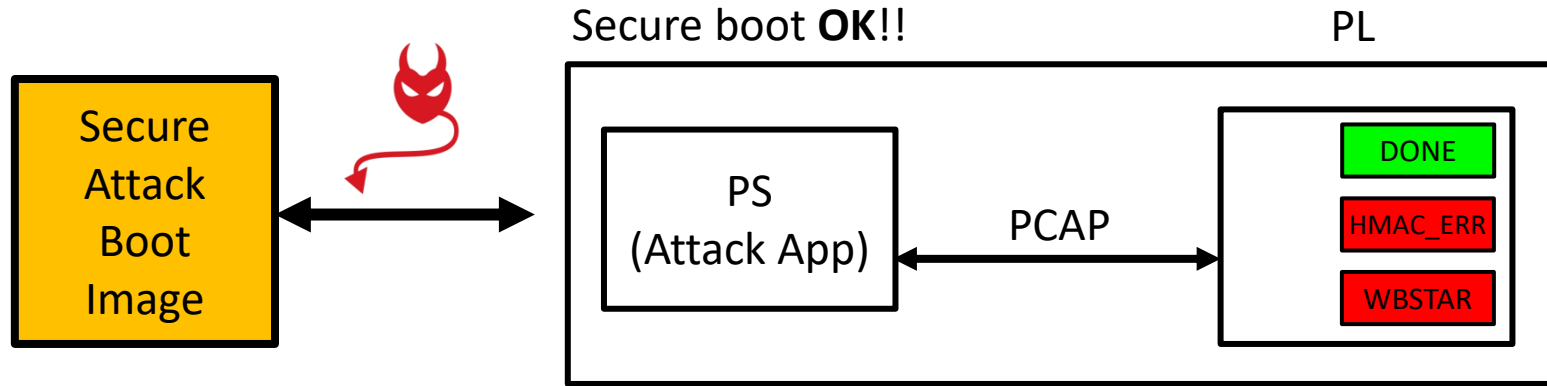
- ❑ **Limitation:** PCAP readback possible only when PL is properly configured (PL Done High)
- ❑ **Attack Steps:**
  - Step-1: We push a valid bitstream (encrypted bitstream in victim image) and fully configure the PL (PL DONE high)

# Starbleed Attack (using PCAP)



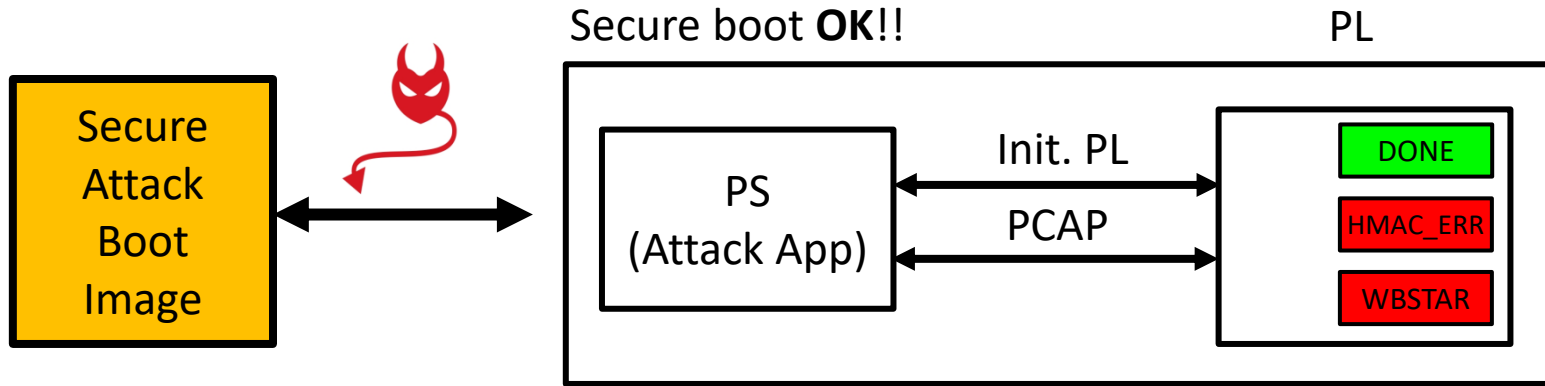
- ❑ **Limitation:** PCAP readback possible only when PL is properly configured (PL Done High)
- ❑ **Attack Steps:**
  - Step-1: We push a valid bitstream (encrypted bitstream in victim image) and fully configure the PL (PL DONE high)

# Starbleed Attack (using PCAP)



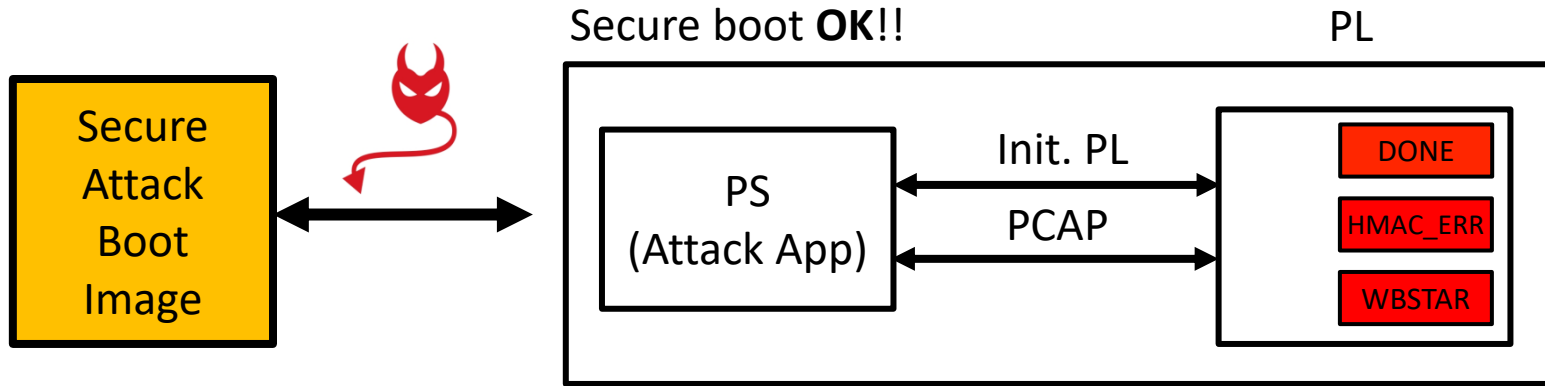
- ❑ **Limitation:** PCAP readback possible only when PL is properly configured (PL Done High)
- ❑ **Attack Steps:**
  - Step-1: We push a valid bitstream (encrypted bitstream in victim image) and fully configure the PL (PL DONE high)
  - Step-2: Without initializing the PL, we send in the Starbleed bitstream (not recommended)

# Starbleed Attack (using PCAP)



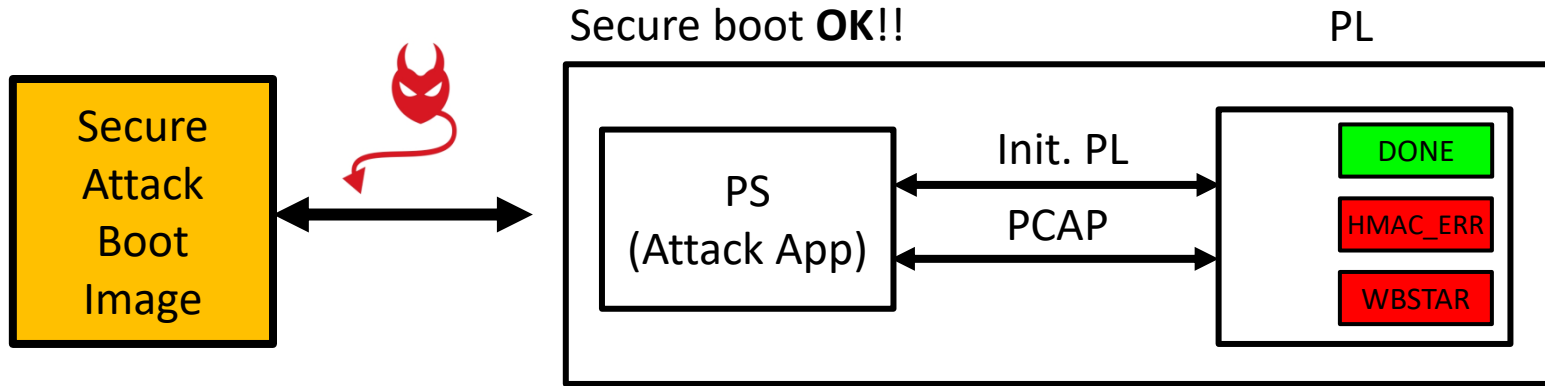
- ❑ **Limitation:** PCAP readback possible only when PL is properly configured (PL Done High)
- ❑ **Attack Steps:**
  - Step-1: We push a valid bitstream (encrypted bitstream in victim image) and fully configure the PL (PL DONE high)
  - Step-2: Without initializing the PL, we send in the Starbleed bitstream (not recommended)

# Starbleed Attack (using PCAP)



- ❑ **Limitation:** PCAP readback possible only when PL is properly configured (PL Done High)
- ❑ **Attack Steps:**
  - Step-1: We push a valid bitstream (encrypted bitstream in victim image) and fully configure the PL (PL DONE high)
  - Step-2: Without initializing the PL, we send in the Starbleed bitstream (not recommended)

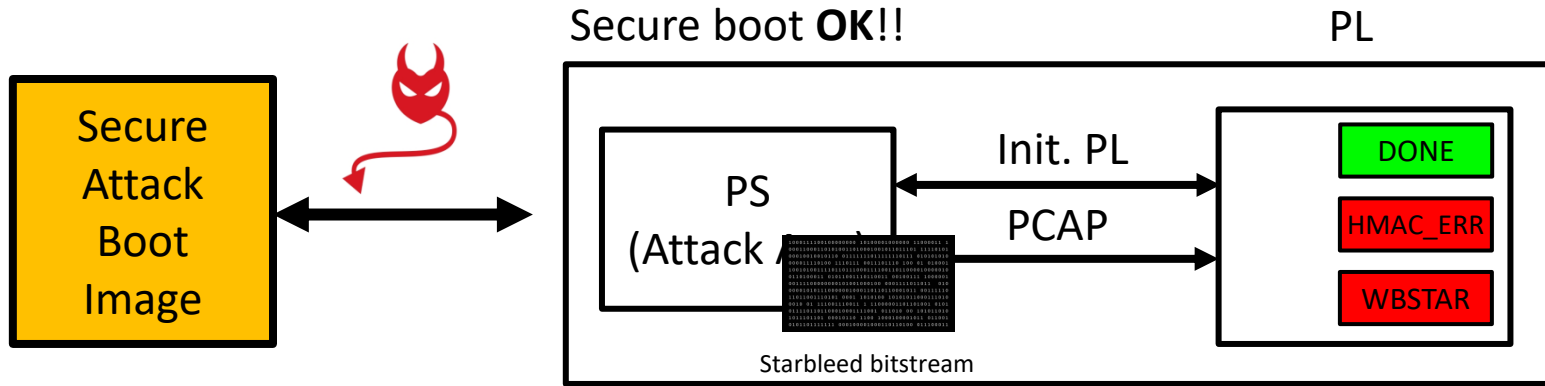
# Starbleed Attack (using PCAP)



- ❑ **Limitation:** PCAP readback possible only when PL is properly configured (PL Done High)
- ❑ **Attack Steps:**
  - Step-1: We push a valid bitstream (encrypted bitstream in victim image) and fully configure the PL (PL DONE high)
  - Step-2: Without initializing the PL, we send in the Starbleed bitstream (not recommended)

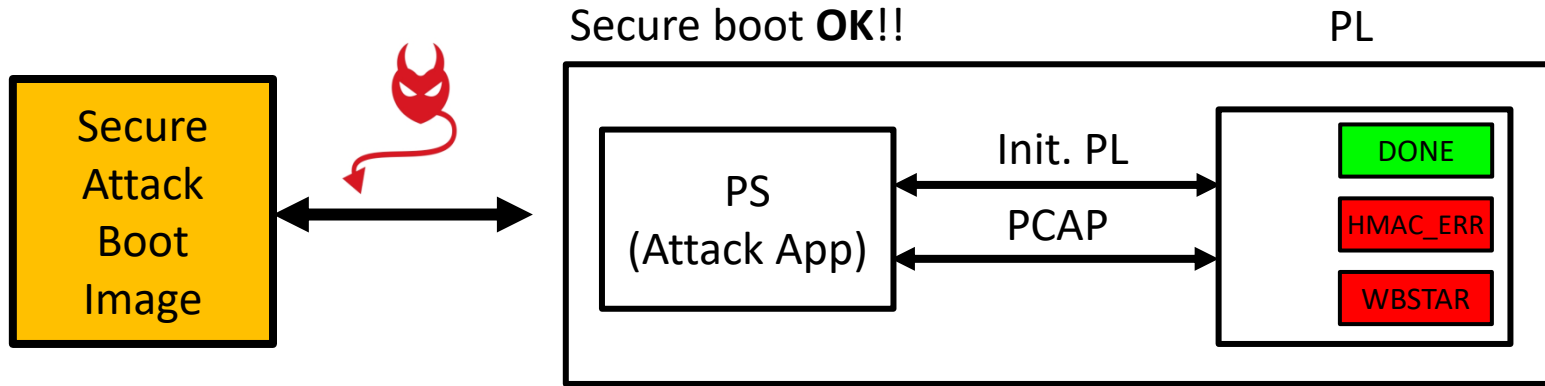


# Starbleed Attack (using PCAP)



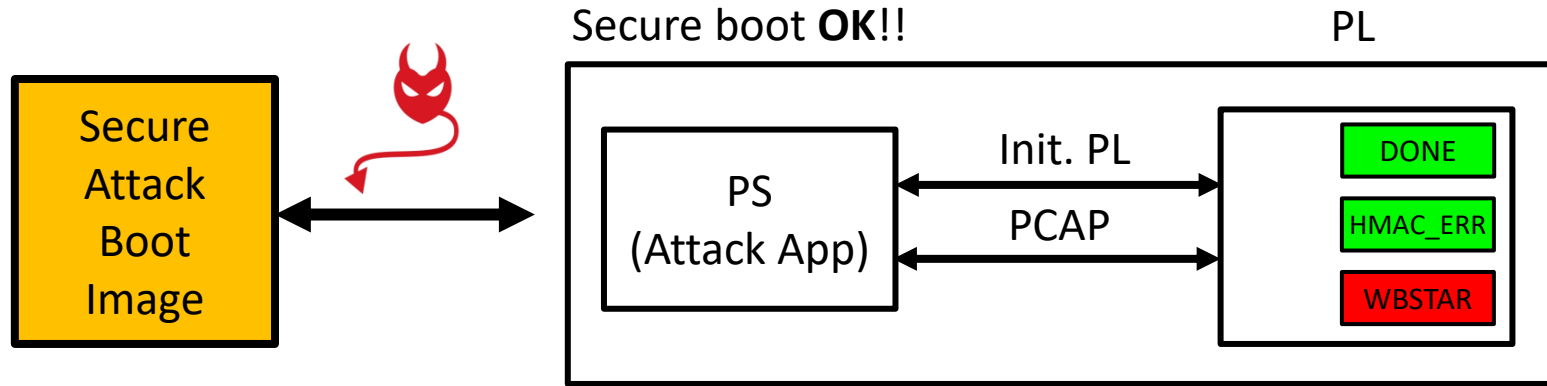
- ❑ **Limitation:** PCAP readback possible only when PL is properly configured (PL Done High)
- ❑ **Attack Steps:**
  - Step-1: We push a valid bitstream (encrypted bitstream in victim image) and fully configure the PL (PL DONE high)
  - Step-2: Without initializing the PL, we send in the Starbleed bitstream (not recommended)

# Starbleed Attack (using PCAP)



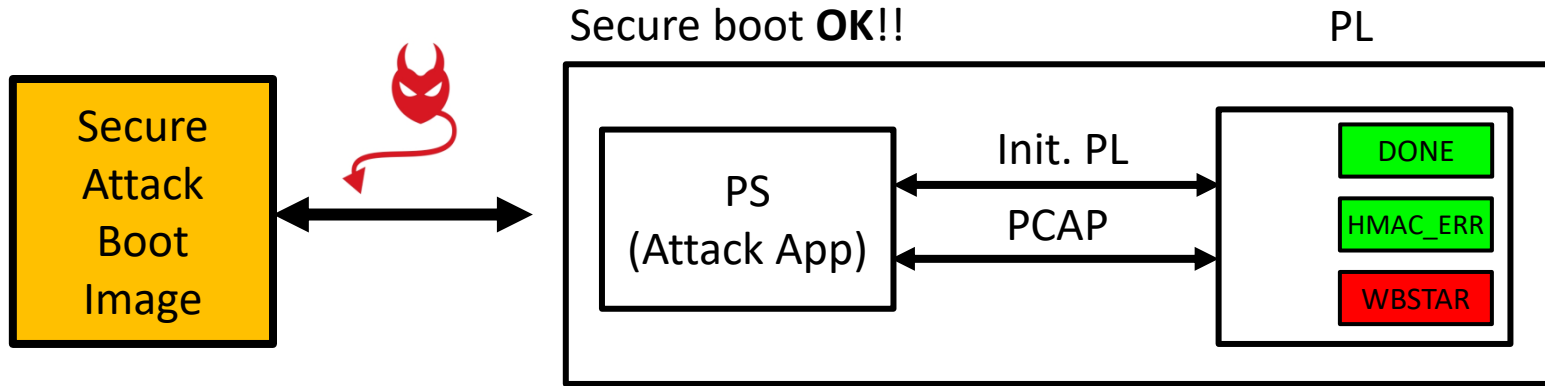
- ❑ **Limitation:** PCAP readback possible only when PL is properly configured (PL Done High)
- ❑ **Attack Steps:**
  - Step-1: We push a valid bitstream (encrypted bitstream in victim image) and fully configure the PL (PL DONE high)
  - Step-2: Without initializing the PL, we send in the Starbleed bitstream (not recommended)

# Starbleed Attack (using PCAP)



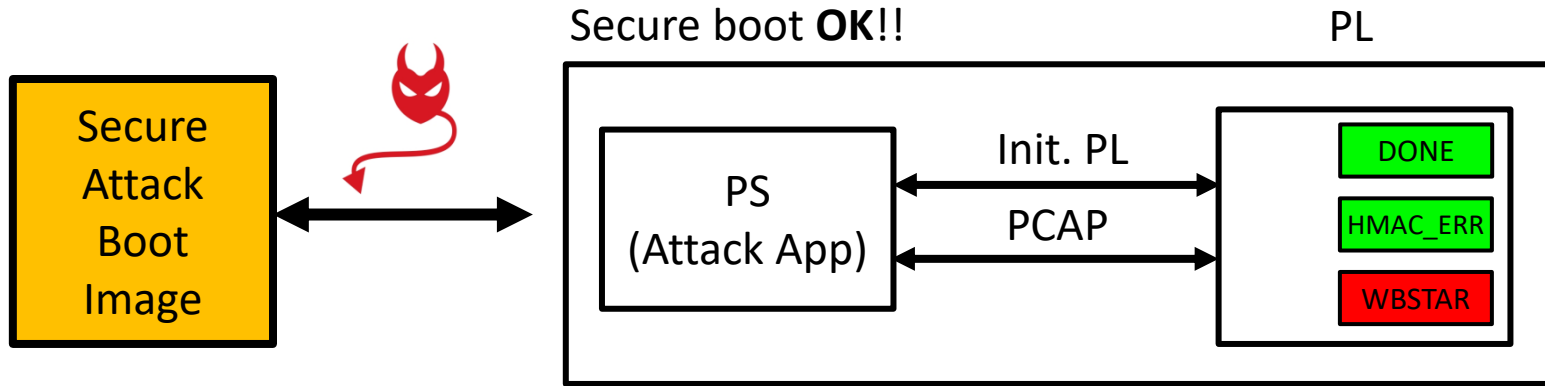
- ❑ **Limitation:** PCAP readback possible only when PL is properly configured (PL Done High)
- ❑ **Attack Steps:**
  - Step-1: We push a valid bitstream (encrypted bitstream in victim image) and fully configure the PL (PL DONE high)
  - Step-2: Without initializing the PL, we send in the Starbleed bitstream (not recommended)

# Starbleed Attack (using PCAP)



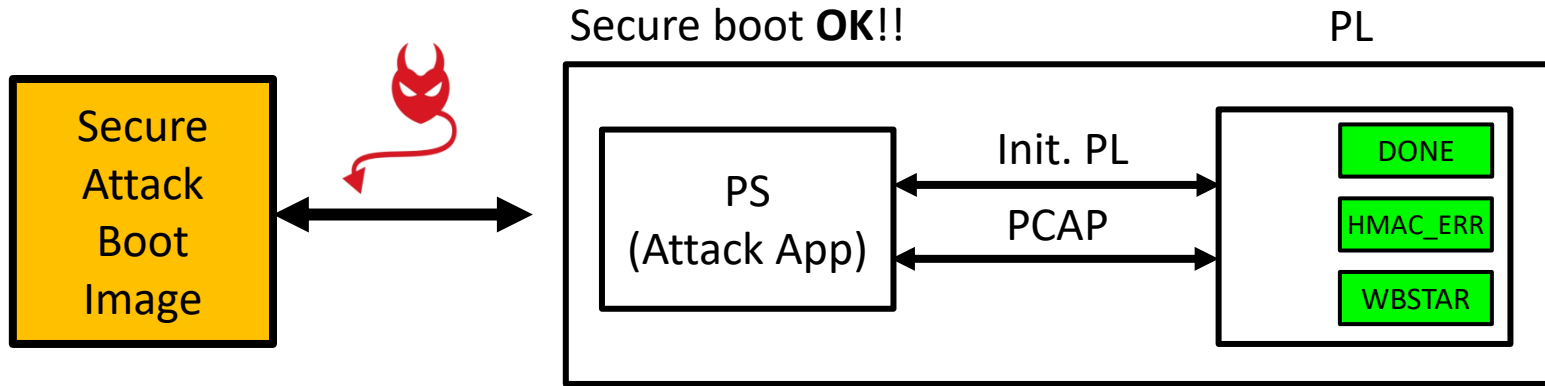
- ❑ **Limitation:** PCAP readback possible only when PL is properly configured (PL Done High)
- ❑ **Attack Steps:**
  - Step-1: We push a valid bitstream (encrypted bitstream in victim image) and fully configure the PL (PL DONE high)
  - Step-2: Without initializing the PL, we send in the Starbleed bitstream (not recommended)
    - We then observe an HMAC error and the DONE LED is still high (**FPGA still fully configured**)

# Starbleed Attack (using PCAP)



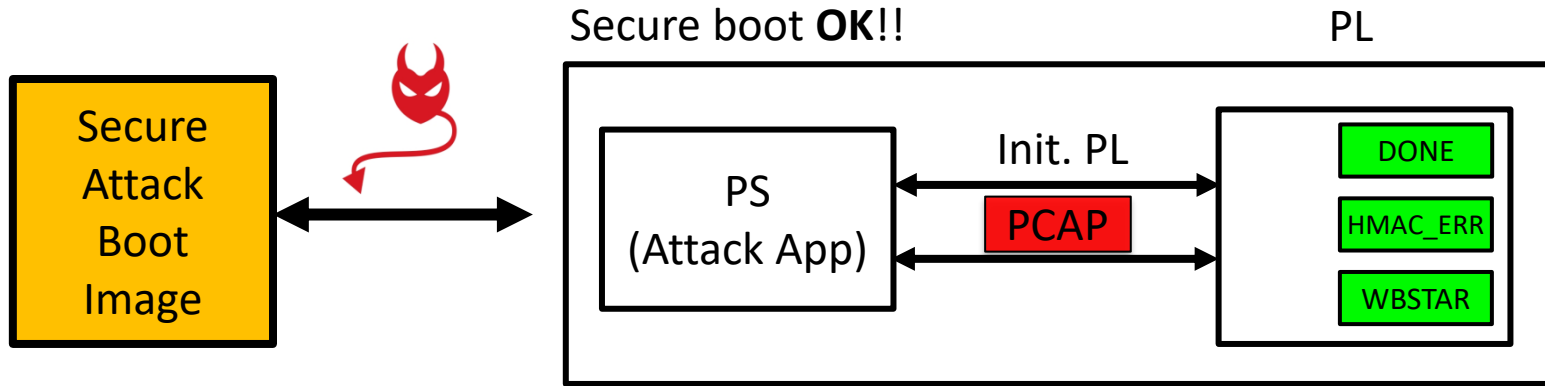
- ❑ **Limitation:** PCAP readback possible only when PL is properly configured (PL Done High)
- ❑ **Attack Steps:**
  - Step-1: We push a valid bitstream (encrypted bitstream in victim image) and fully configure the PL (PL DONE high)
  - Step-2: Without initializing the PL, we send in the Starbleed bitstream (not recommended)
    - We then observe an HMAC error and the DONE LED is still high (**FPGA still fully configured**)
  - Step-3: We read the WBSTAR register through PCAP interface - We get the decrypted codeword!!!

# Starbleed Attack (using PCAP)



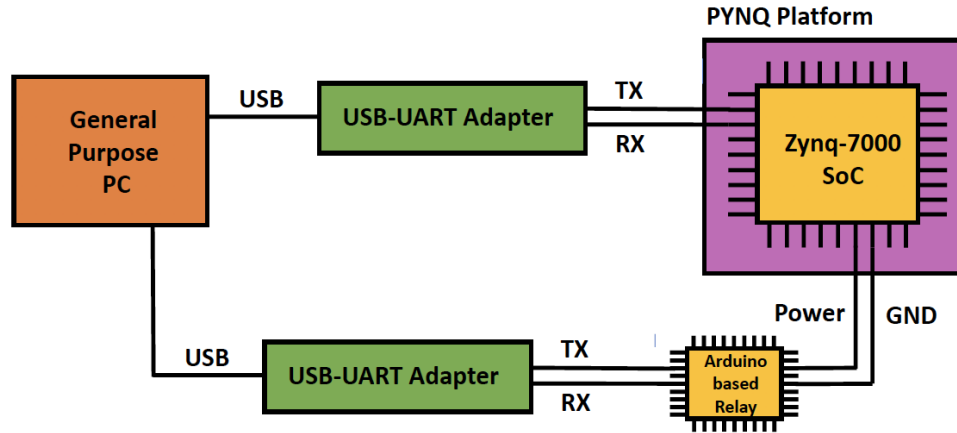
- ❑ **Limitation:** PCAP readback possible only when PL is properly configured (PL Done High)
- ❑ **Attack Steps:**
  - Step-1: We push a valid bitstream (encrypted bitstream in victim image) and fully configure the PL (PL DONE high)
  - Step-2: Without initializing the PL, we send in the Starbleed bitstream (not recommended)
    - We then observe an HMAC error and the DONE LED is still high (**FPGA still fully configured**)
  - Step-3: We read the WBSTAR register through PCAP interface - We get the decrypted codeword!!!

# Starbleed Attack (using PCAP)



- ❑ **Limitation:** PCAP readback possible only when PL is properly configured (PL Done High)
- ❑ **Attack Steps:**
  - Step-1: We push a valid bitstream (encrypted bitstream in victim image) and fully configure the PL (PL DONE high)
  - Step-2: Without initializing the PL, we send in the Starbleed bitstream (not recommended)
    - We then observe an HMAC error and the DONE LED is still high (**FPGA still fully configured**)
  - Step-3: We read the WBSTAR register through PCAP interface - We get the decrypted codeword!!!
  - Step-4: PCAP goes into unknown state – **unresponsive** - requires a POR reset

# Automating Starbleed Attack (using PCAP)



- ❑ Starbleed bitstreams need to be created adaptively (based on knowledge of previously retrieved words)
- ❑ Since we have control of attack application, we use UART interface to communicate with target
- ❑ New bitstreams are fed to the Zynq device through the UART interface (then used by PS for the attack)
- ❑ We have an Arduino based relay to perform automatic POR reset of the target



# Outline

- ❑ Introduction
- ❑ RSA Authentication Attack on Zynq-7000
  - ❑ Background: Attack Model, Secure Boot and RSA Authentication
  - ❑ Vulnerability in FSBL
  - ❑ Attack Implementation: Using SD Card Switcher Board
- ❑ Starbleed on Zynq
  - ❑ Introduction and Working
  - ❑ **Experimental Results**
- ❑ Analyzing SD Card Data Transfer
- ❑ BootROM
  - ❑ Possible BootROM Vulnerabilities
  - ❑ PHT Transfer Analysis
  - ❑ BootROM Data Transfer Analysis
- ❑ Conclusion and Future Works



# Starbleed on Zynq: Experimental Results

- ❑ We are able to retrieve a single decrypted bitstream word in approx. 1 second.
- ❑ An encrypted bitstream of size 3.85 MB can be retrieved in 46 days.
- ❑ Attacker needs access to the target device for this duration.
- ❑ Maximum time spent in POR reset (target device goes through secure boot for every bitstream word)



# Optimizing Starbleed Attack on Zynq

- ❑ Faster bitstream recovery is possible with dedicated PCB and faster relay
- ❑ Can have multiple target devices to speed-up the attack as well.
- ❑ **Main Bottleneck:** POR reset requirement ( when using secure boot )
- ❑ Can we perform attack without requiring POR reset?
  - ❑ Once we recover the HMAC key, we can create authenticated starbleed bitstreams (No HMAC error)
  - ❑ PCAP could potentially be retained in a working state
  - ❑ Complete bitstream recovery might be possible without POR reset for every recovered word
  - ❑ Some sound strategies may not work because of some unknown reason as well.



# Attack Demo

4/19/2024

40





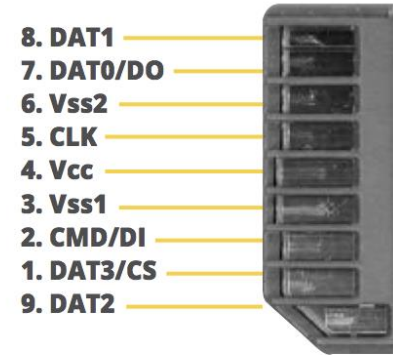
# Outline

- ❑ Introduction
- ❑ **RSA Authentication Attack on Zynq-7000**
  - ❑ Background: Attack Model, Secure Boot and RSA Authentication
  - ❑ Vulnerability in FSBL
  - ❑ Attack Implementation: Using SD Card Switcher Board
- ❑ Starbleed on Zynq
  - ❑ Introduction and Working
  - ❑ Experimental Results
- ❑ **Analyzing SD Card Data Transfer**
- ❑ BootROM
  - ❑ Possible BootROM Vulnerabilities
  - ❑ PHT Transfer Analysis
  - ❑ BootROM Data Transfer Analysis
- ❑ Conclusion and Future Works

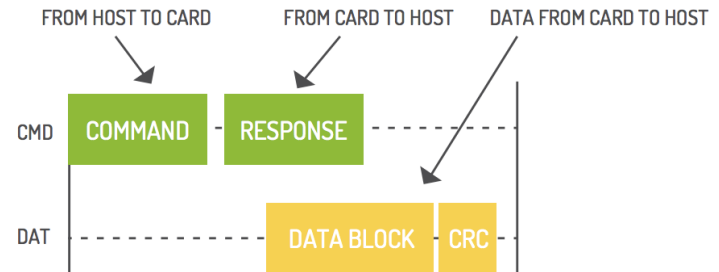


# SD Card Interface: Background

- ❑ 9 wire interface:
  - ❑ CMD (Command)
  - ❑ CLK (Clock)
  - ❑ DAT0-DAT3 (4 data lines)
- ❑ Commands and Response are exchanged over CMD line
  - ❑ In the form of Packets
- ❑ SD card contains a few information registers:
  - ❑ Control and Status of SD Card interface
- ❑ Reading/Writing in blocks of 512 bytes
- ❑ Important commands for read:
  - ❑ CMD17 - To read single block
  - ❑ CMD18 - To read multiple blocks



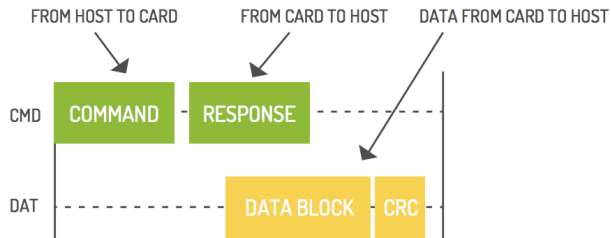
Single block read transfer, SD mode



# SD Card Interface

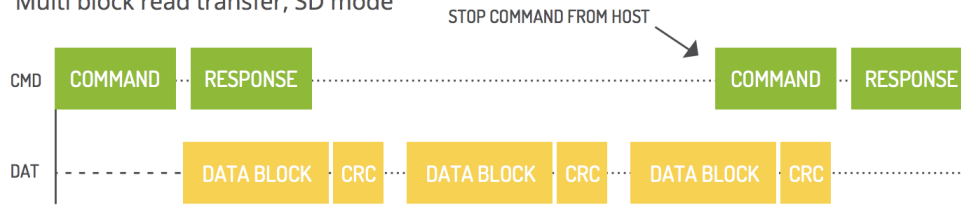
## CMD17:

Single block read transfer, SD mode



## CMD18:

Multi block read transfer, SD mode

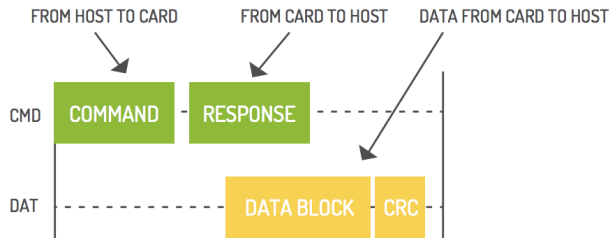




# SD Card Interface

## CMD17:

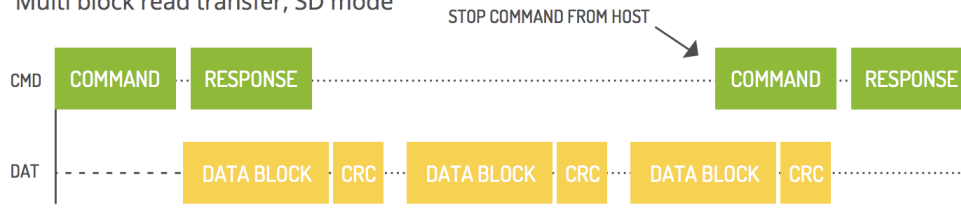
Single block read transfer, SD mode



Idea:

## CMD18:

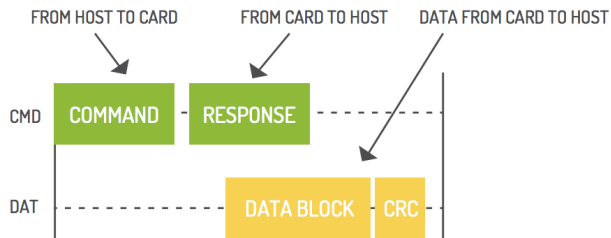
Multi block read transfer, SD mode



# SD Card Interface

## CMD17:

Single block read transfer, SD mode

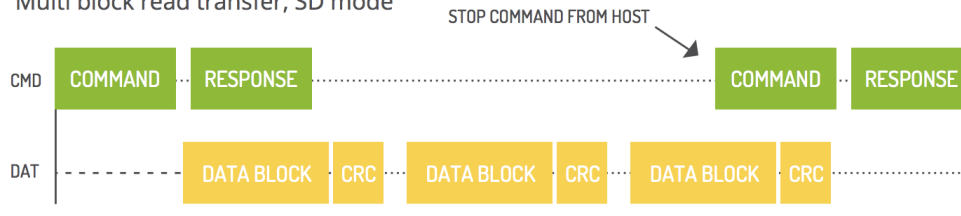


**Idea:**

☐ Monitor the number of CMD17, CMD18 calls

## CMD18:

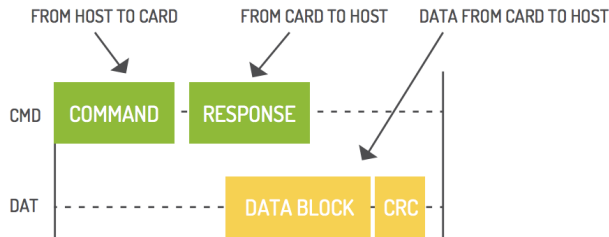
Multi block read transfer, SD mode



# SD Card Interface

## CMD17:

Single block read transfer, SD mode

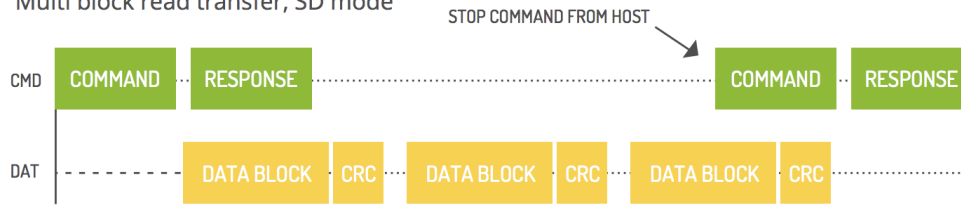


## Idea:

- ❑ Monitor the number of CMD17, CMD18 calls
- ❑ Gives us information about data blocks read by BootROM/FSBL over SD interface...

## CMD18:

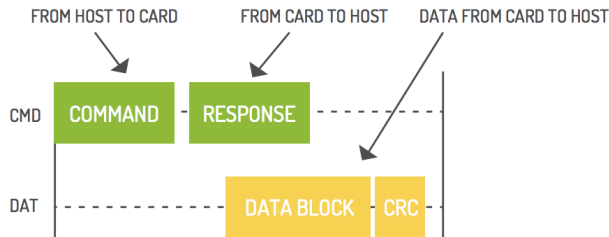
Multi block read transfer, SD mode



# SD Card Interface

## CMD17:

Single block read transfer, SD mode



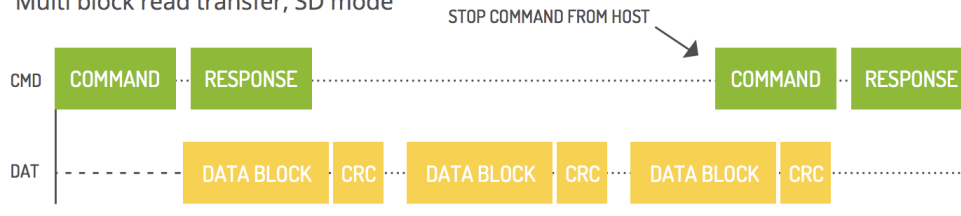
## Idea:

- ❑ Monitor the number of CMD17, CMD18 calls
- ❑ Gives us information about data blocks read by BootROM/FSBL over SD interface...

**Use Logic Analyzer to analyze SD Card Interface**

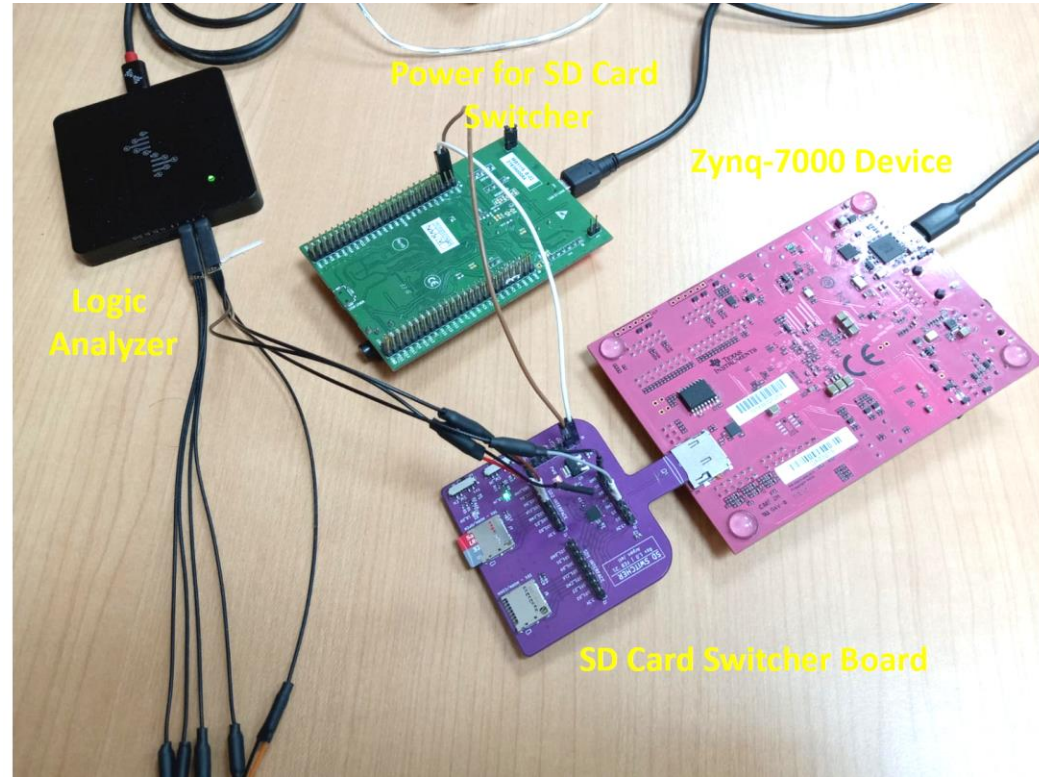
## CMD18:

Multi block read transfer, SD mode



# Logic Analyzer for SD Card Interface

- ❑ **DS Logic Plus Analyzer:**
  - ❑ 400 MHz (16 channels)
  - ❑ SDIO protocol decoder
- ❑ **Analysis of the following signals:**
  - ❑ CMD
  - ❑ CLK
  - ❑ DAT3 (Can be any other data line)



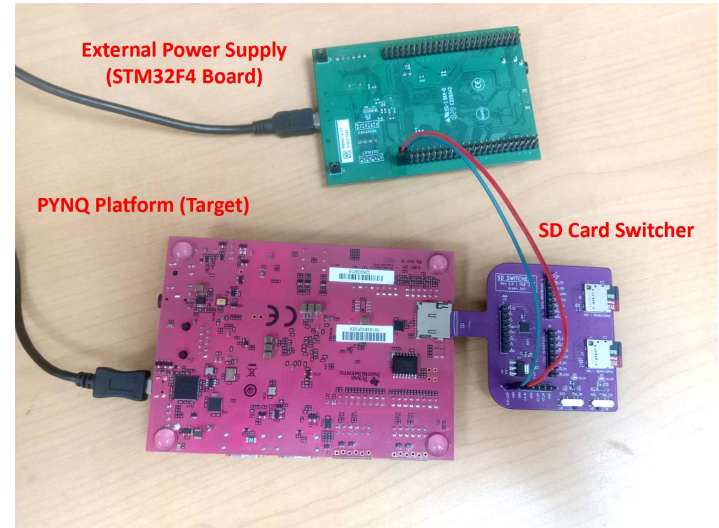
# Outline

- ❑ Introduction
- ❑ RSA Authentication Attack on Zynq-7000
  - ❑ Background: Attack Model, Secure Boot and RSA Authentication
  - ❑ Vulnerability in FSBL
  - ❑ Attack Implementation: Using SD Card Switcher Board
- ❑ Starbleed on Zynq
  - ❑ Introduction and Working
  - ❑ Experimental Results
- ❑ Analyzing SD Card Data Transfer
- ❑ BootROM
  - ❑ **Possible BootROM Vulnerabilities**
  - ❑ PHT Transfer Analysis
  - ❑ BootROM Data Transfer Analysis
- ❑ Conclusion and Future Works

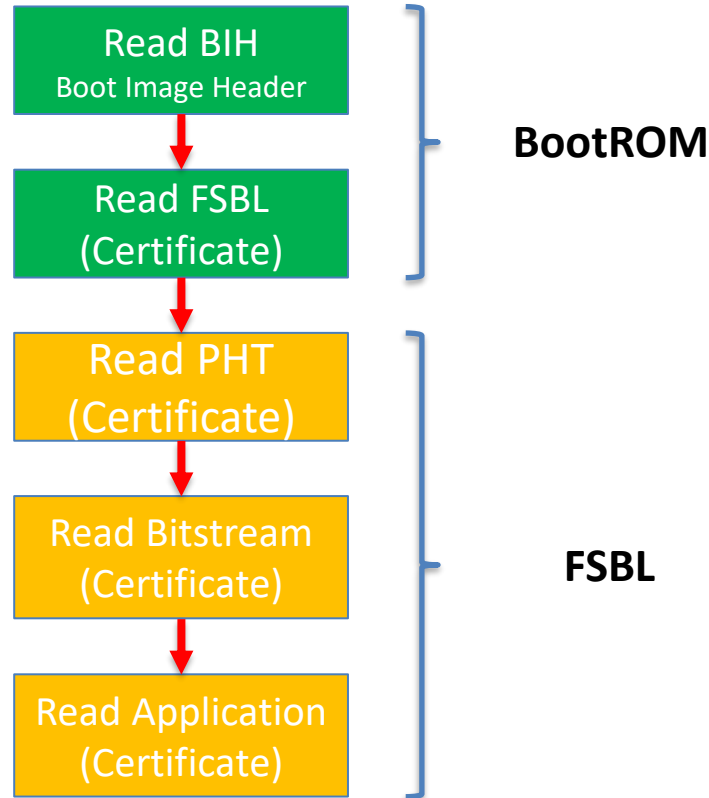


# Possible BootROM Vulnerabilities

- ❑ What about **BootROM**?
  - ❑ RSA Authentication, Decryption of FSBL (SD Card)
  - ❑ Any vulnerabilities in BootROM?
- ❑ **Challenges:**
  - ❑ BootROM code is not available (hard-coded on chip)
  - ❑ BootROM code cannot be changed
- ❑ **In this work:**
  - ❑ Black Box Vulnerability analysis of BootROM
  - ❑ Updates on our previous attack on FSBL
- ❑ Probe the SD Card Interface between SoC and SD Card

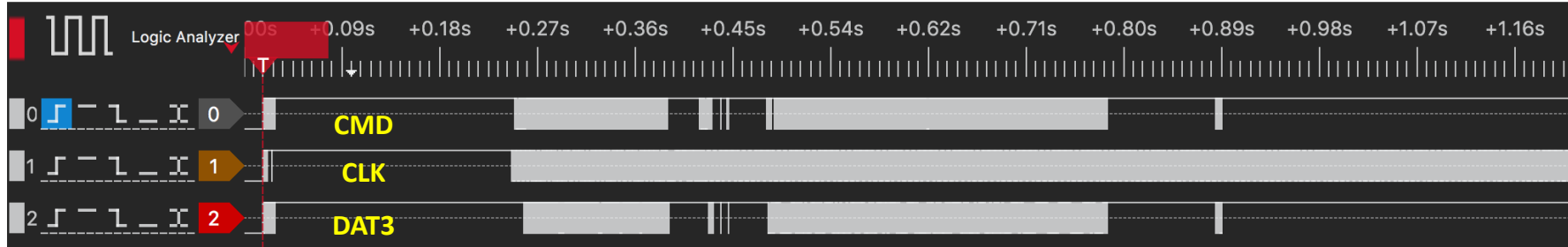


# Analysis of SD Card Interface during Bootup



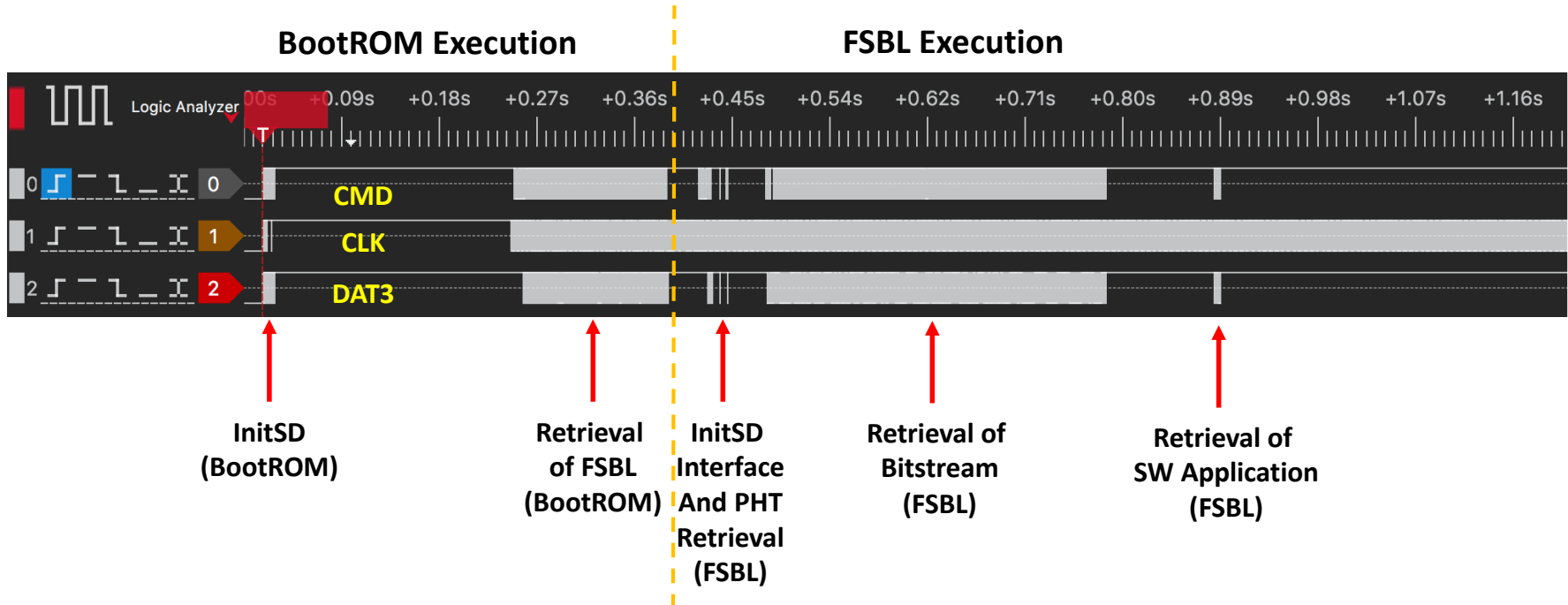


# Full Boot up:

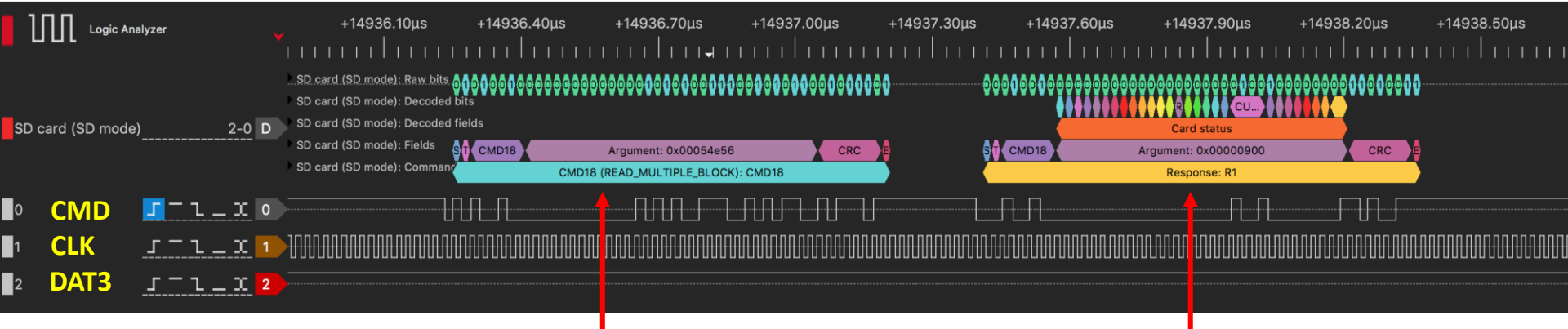


- ❑ How to differentiate between BootROM transfers and FSBL transfers?
  - ❑ **Observation:** FSBL is controllable software
  - ❑ **Idea:** Insert varying delays within the FSBL software and observe how the transfers are perturbed.
    - ❑ Insert delay just after start of FSBL
    - ❑ Insert delay after PHT transfer
    - ❑ Insert delays after bitstream transfer

# Full Boot up:



# Full Boot up:



**CMD18 from Zynq Device (Host)  
(Read Multiple Blocks at Address: 0x54e56)**

**Response from SD Card  
(Acknowledgement)**

# Outline

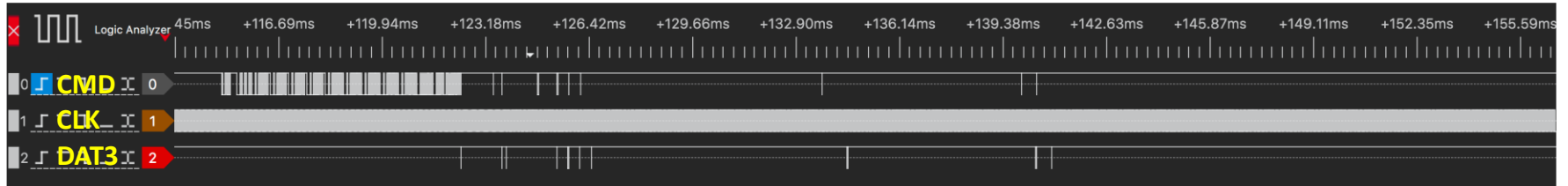
- ❑ Introduction
- ❑ RSA Authentication Attack on Zynq-7000
  - ❑ Background: Attack Model, Secure Boot and RSA Authentication
  - ❑ Vulnerability in FSBL
  - ❑ Attack Implementation: Using SD Card Switcher Board
- ❑ Starbleed on Zynq
  - ❑ Introduction and Working
  - ❑ Experimental Results
- ❑ Analyzing SD Card Data Transfer
- ❑ BootROM
  - ❑ Possible BootROM Vulnerabilities
  - ❑ **PHT Transfer Analysis**
  - ❑ BootROM Data Transfer Analysis
- ❑ Conclusion and Future Works



# PHT Transfer by FSBL

- ❑ We know there are two PHT transfers (PHT1 and PHT2)
- ❑ **To identify PHT1:** Put an infinite while loop after PHT1

## FSBL Execution

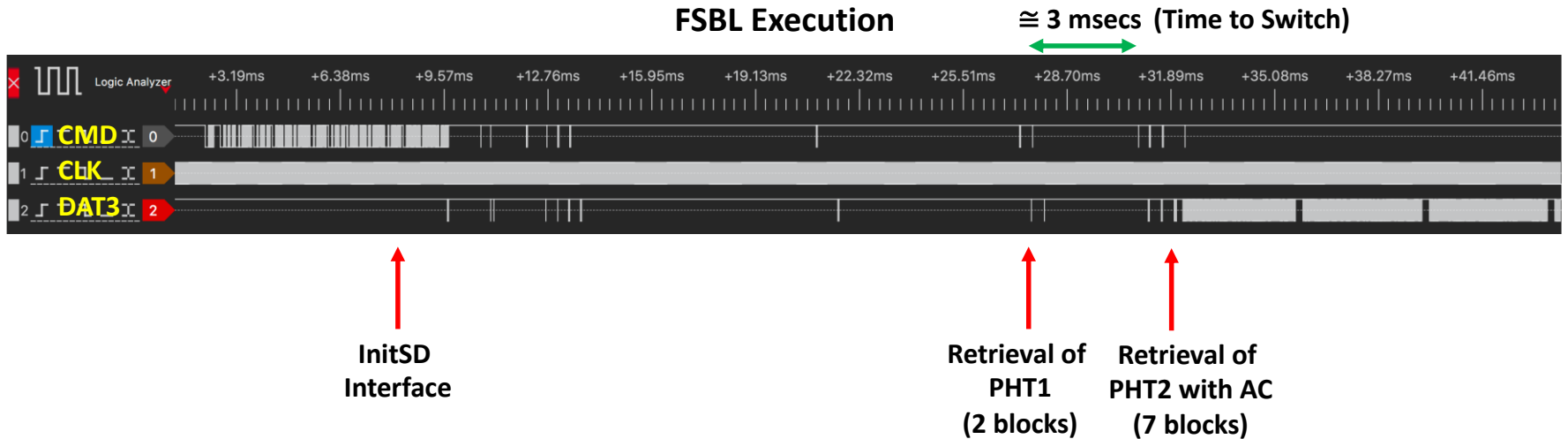


↑  
**InitSD  
Interface**

↑  
**Retrieval of  
PHT1  
(2 blocks)**

# PHT Transfer by FSBL

- ❑ We know there are two PHT transfers (PHT1 and PHT2)
- ❑ **To identify PHT2:** Put an infinite while loop after PHT2



- ❑ PHT1 transfer: **445** msecs (from first clock edge on CMD line)
- ❑ PHT2 transfer: **448.5** msecs (from first clock edge on CMD line)

# Outline

- ❑ Introduction
- ❑ RSA Authentication Attack on Zynq-7000
  - ❑ Background: Attack Model, Secure Boot and RSA Authentication
  - ❑ Vulnerability in FSBL
  - ❑ Attack Implementation: Using SD Card Switcher Board
- ❑ Starbleed on Zynq
  - ❑ Introduction and Working
  - ❑ Experimental Results
- ❑ Analyzing SD Card Data Transfer
- ❑ BootROM
  - ❑ Possible BootROM Vulnerabilities
  - ❑ PHT Transfer Analysis
  - ❑ **BootROM Data Transfer Analysis**
- ❑ Conclusion and Future Works



# Analyzing BootROM Behaviour

- ❑ **Area of Interest:** data blocks transferred during FSBL authentication
- ❑ We consider three cases:
  - ❑ Non-secure Boot (**Nsec**)
  - ❑ Secure with only encryption (**Sec\_Encrypt**)
  - ❑ Secure with both encryption and authentication (**Sec\_Auth\_Encrypt**)





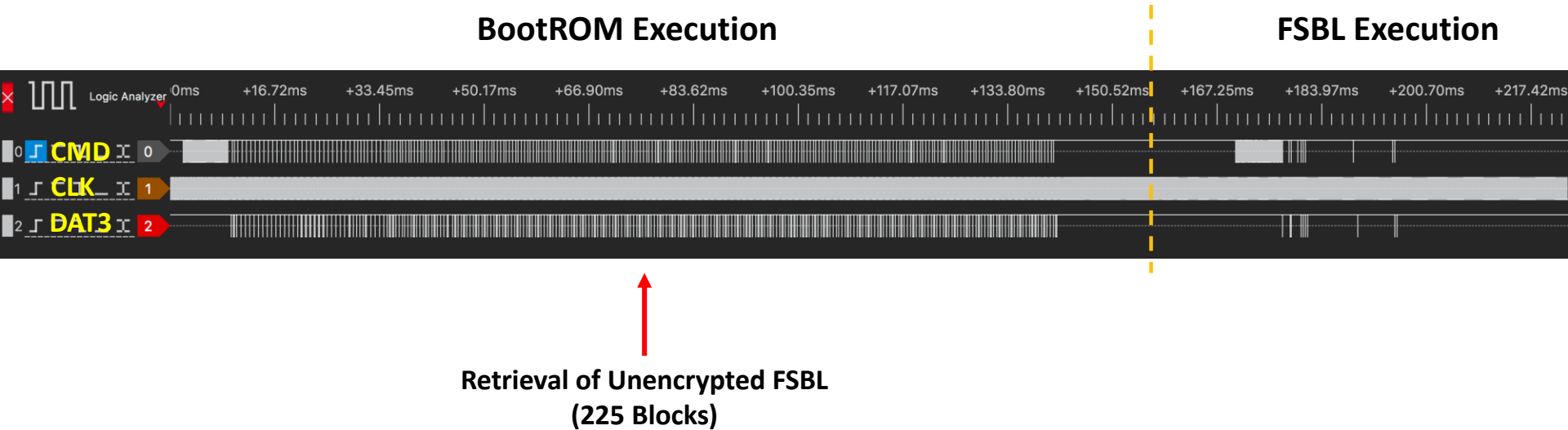
# BootROM Behaviour: Nsec Image

- ❑ Read unencrypted FSBL
- ❑ 114.5 KB = 225 Blocks



# BootROM Behaviour: Nsec Image

- ❑ Read unencrypted FSBL
- ❑ 114.5 KB = 225 Blocks



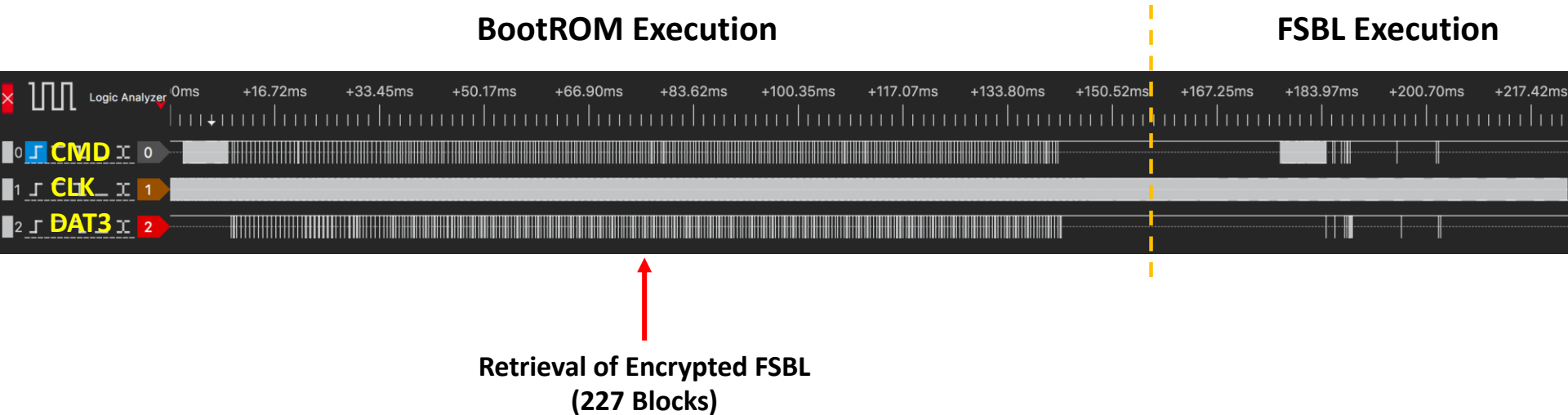
# BootROM Behaviour: Sec\_Encrypt Image

- ❑ Read encrypted FSBL
- ❑ 115.5 KB = 227 Blocks



# BootROM Behaviour: Sec\_Encrypt Image

- ❑ Read encrypted FSBL
- ❑ 115.5 KB = 227 Blocks



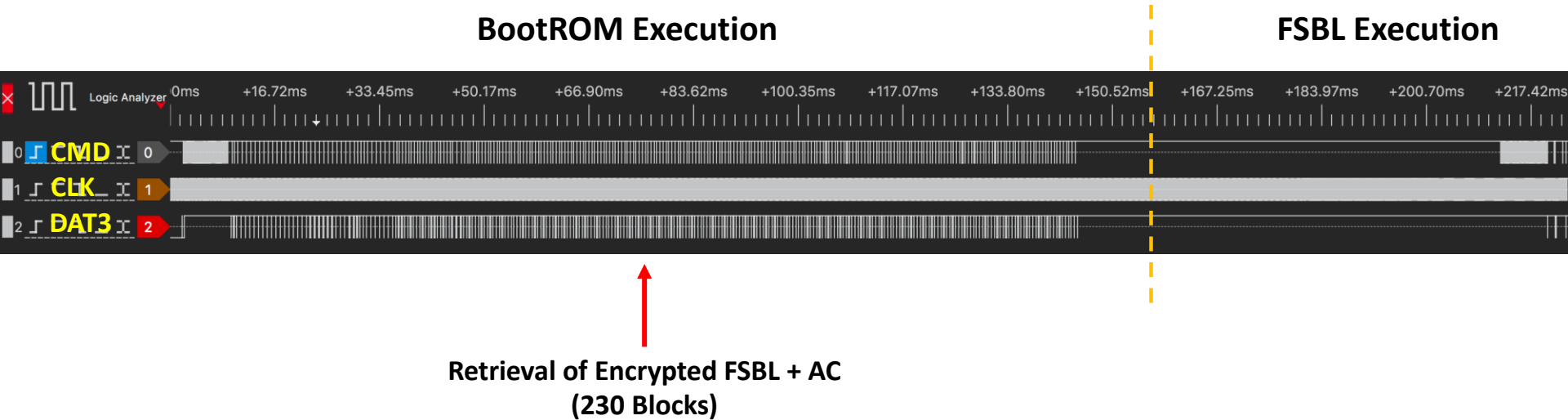
# BootROM Behaviour: Sec\_Auth\_Encrypt Image

- ❑ Read encrypted FSBL + Certificate
- ❑ 116.8 KB = 230 Blocks



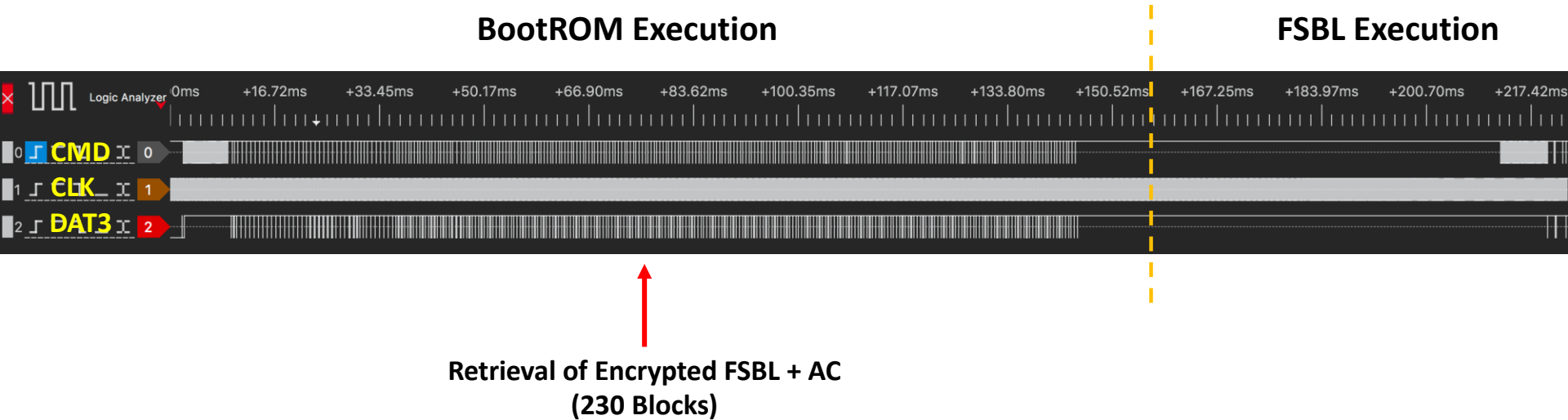
# BootROM Behaviour: Sec\_Auth\_Encrypt Image

- ❑ Read encrypted FSBL + Certificate
- ❑ 116.8 KB = 230 Blocks



# BootROM Behaviour: Sec\_Auth\_Encrypt Image

- ❑ Read encrypted FSBL + Certificate
- ❑ 116.8 KB = 230 Blocks



- ❑ **Inference:** There are no duplicate data transfers of the FSBL data during BootROM execution...

# Conclusion:

- ❑ We have identified a critical security flaw in the Zynq-7000 FSBL software, due to mishandling of the PHT data.
- ❑ We experimentally validated exploitation of the flaw, using an SD card switcher board.
- ❑ A very minor modification to the FSBL is required to demonstrate successful attack with existing hardware.
- ❑ For real world attack, we need a specialized hardware between the target and the SD card switcher board.
- ❑ Xilinx/AMD has acknowledged the presence of the critical flaw to bypass RSA Authentication.
- ❑ A software patch for the FSBL is provided Xilinx to remove the vulnerability.
  - ❑ **But all unpatched devices in the wild face recovery of unencrypted bitstream and application files.**
- ❑ We performed the first vulnerability analysis of the BootROM software of Zynq-7000 SoC
  - ❑ We used a logic analyzer to probe the SD card interface during FSBL and BootROM execution
  - ❑ BootROM Analysis: showed that there is no duplicate transfer of FSBL during BootROM execution





# Future Works:

- ❑ After patching the PHT authentication vulnerability, are more attacks still possible??
- ❑ Fault Vulnerability Analysis of the FSBL, BootROM



**Thank you!!!**

