UNC CHARLOTTE UNIVERSITY OF NORTH CAROLINA

Hardware Security and Trust Challenges in Emerging IoT Systems and Applications

Prof. Fareena Saqib (fsaqib@uncc.edu) ECE DepT, UNCC

HOST TUTORIAL 2018

Agenda Layout

1. Introduction to Internet of Things and Cybersecurity
2. Security challenges in Automotive Security
3. Future directions in research
4. Automotive Ethernet
5. Architectural and Hardware security
6. Trusted platform module
7. Hardware demo of secure communication
8. FPGA as ECU platform
9. Secure boot
10. Conclusion and Q&A session

Internet of Things: An Era of Smart

Amazon Alexa

Holus is a triangular holographic chamber

Amazon Fire TV Stick

3D-Printer build a house

Virtual Reality
Oculus Rift and Omni treadmill

Internet of Things: IoT Characteristics

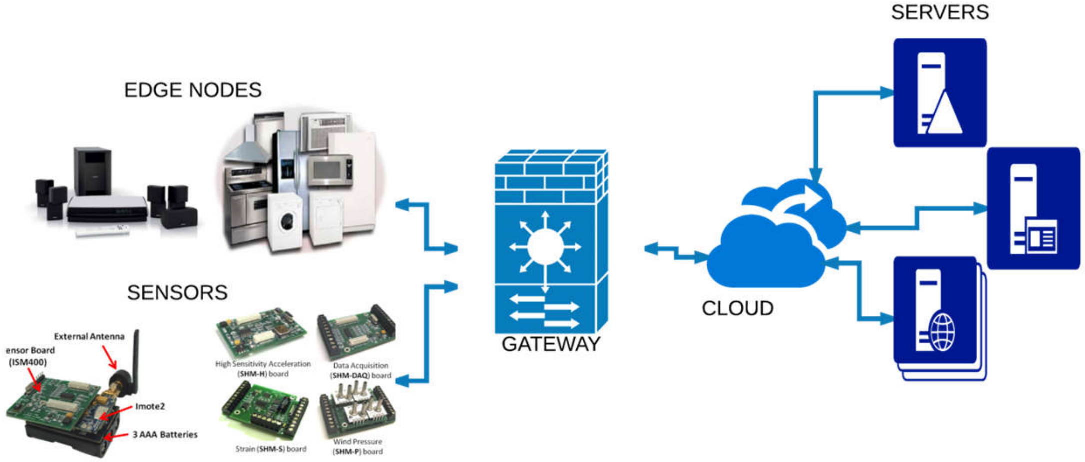
- IoT, a major shift in computing.
- IoT is a major shift of consumer interacts with the technology and interface with the Internet.
- IoT making progress in initiatives such as smart grid, and intelligent vehicles.
- Computing devices are becoming distributed, unsupervised, and physically exposed

IOT Challenges

- Connecting devices, exchanging data with the other nodes and server/cloud.
- Delivering value through smart interfaces and user experience

IOT Security Issues

- Long life cycles of IoTs
- Provisioning keys and key management life cycle
- Security assessment of equipment connected via gateways, that were never intended to be connected.
- Device identification for device-to-device communication
- Availability and system resilience.
- Scalability



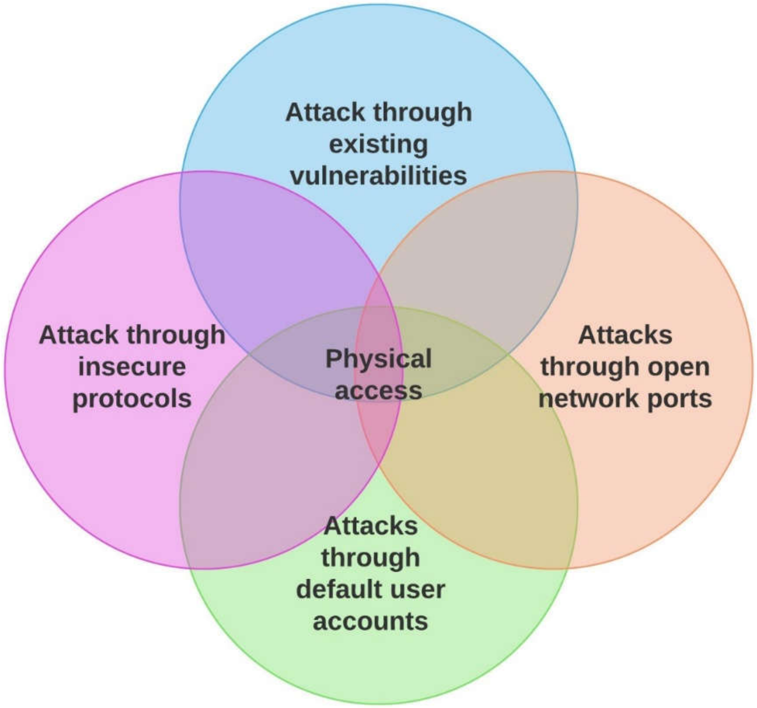
Requires holistic view of device to gateway to cloud and the communication between them.

IOT Security Concerns

- Privacy Concerns
  - Collect personal information
  - Unencrypted transmission across networks.
- Authentication/Authorization mechanism
  - Weak passwords
- Firmware Updates
  - Unencrypted software and firmware updates
- Network Encryption
  - Use of insecure and unencrypted network services.
  - ZigBee, Bluetooth, Ethernet, Wireless sensor networks and Internet
- Web Interfaces
  - Poor session management, cookies
  - Persistent cross site scripting.
- Device Security
  - Hardware lacks key management system, and no established root of trust

IOT Attack Surface

- The data and control paths and I/O ports of the devices.
- The processes that protects these paths.
- All valuable data used in the device, including secrets and keys.
- The external cryptographic functions that protects these data.



IOT Attacks

25th April: Finnish researchers are able to clone hotel master keys using $300 RFID card reader and an expired keycard.

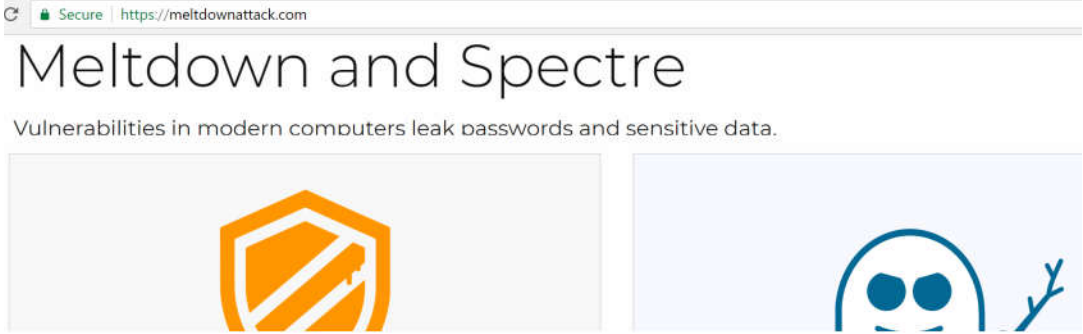
In 60 seconds, security researchers can clone the master hotel-room keys for 140,000 hotels in 160 countries

26th April: Alexa's built-in JavaScript library can turn on listening and bypass the requirement for trigger words.

Security researchers can turn Alexa into a transcribing, always-on listening device

24th April: Hackers exploited USB based rescue mode to overtake Nintendo Switch console.

4th January: Spectre and Meltdown were first publicly reported.

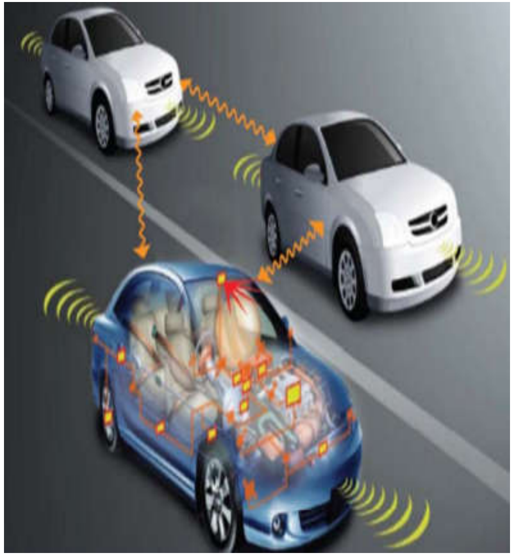


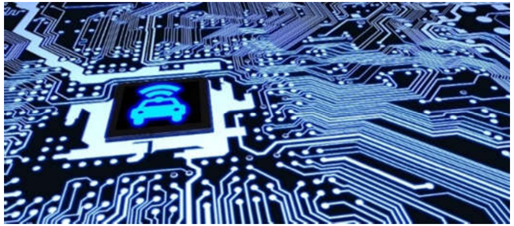
IOT Security and Trust

- There is no silver bullet for security.
- Devices needs to be built with security in the design flow.
- Develop key life cycle management.
- Communications paths for security events and encryption.
- Foresee the issues by applying analytics.
- Secure framework design and integration with IoT.

Automotive Class of IoTs

- Electronics as an innovation driver.
- Safety – Air bags.
- Advanced driver assistance system **ADAS**.
- Self Driving Cars.
- Smart charging – A key to successful E-Mobility and autonomous charging.





Automotive: Security and Trust Threats and Attacks

Security experts reveal $40 device that would allow thieves to wirelessly unlock nearly every Volkswagen made since 1995

- Audi, VW, and Škoda among models at risk of being wirelessly hacked
- One attack pulls a shared key value from one vehicle that opens others
- The other hack uses eight rolling codes to discover the owner's codes
- Both attacks are conducted with a battery-powered radio and RF module

ANDY GREENBERG SECURITY 07.24.15 12:30 PM

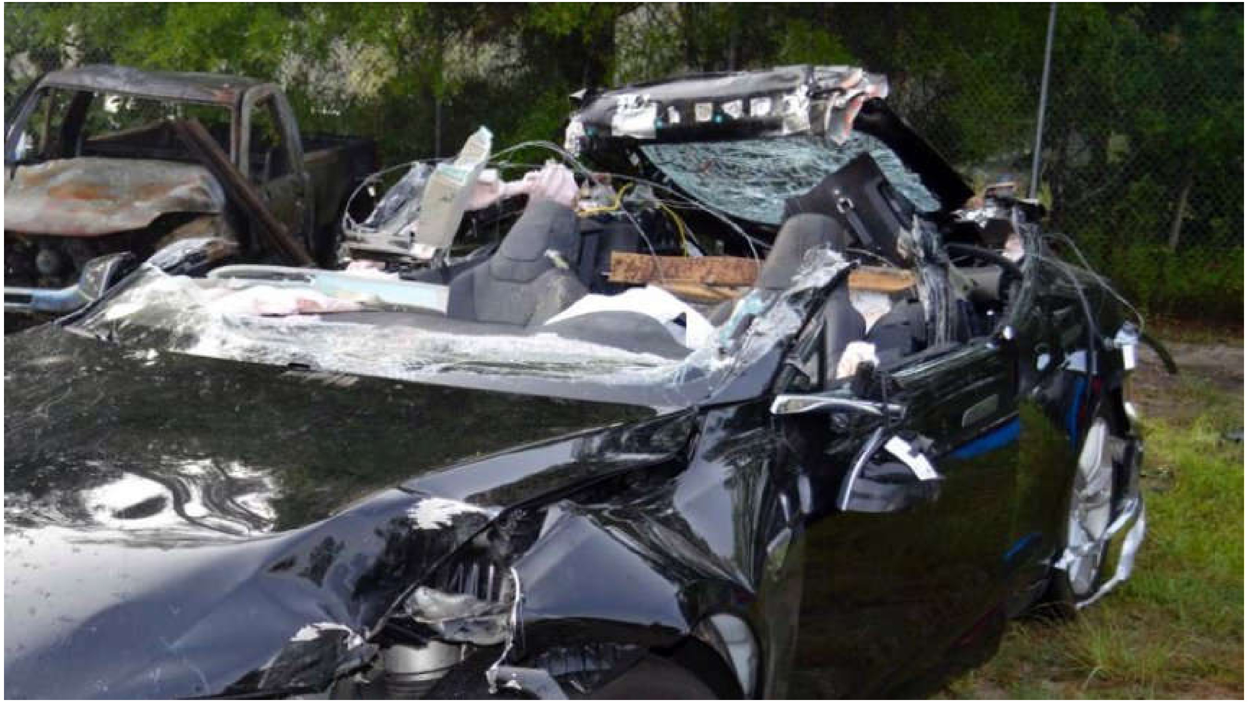
AFTER JEEP HACK, CHRYSLER RECALLS 1.4M VEHICLES FOR BUG FIX

RADIO ATTACK LETS HACKERS STEAL 24 DIFFERENT CAR MODELS

http://www.dailymail.co.uk/sciencetech/article-3737375/Security-experts-reveal-40-device-allow-thieves-wirelessly-unlock-nearly-Volkswagen-1995.html

https://www.wired.com/2016/03/study-finds-24-car-models-open-unlocking-ignition-hack/

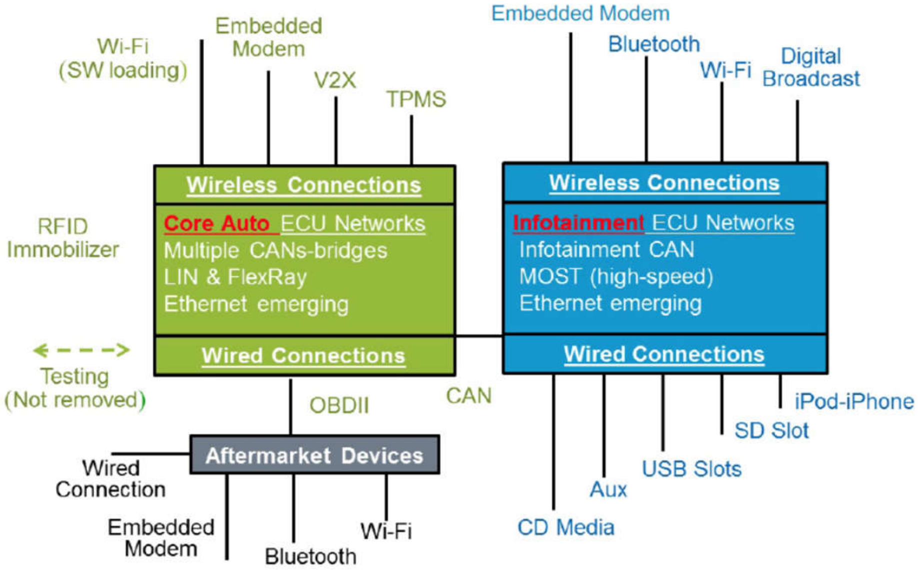
Automotive: Safety Threats



A Tesla Model S that crashed while in self driving mode which resulted in the death of Joshua Brown on May 7, 2016. FLORIDA HIGHWAY PATROL

Automotive: Communication as a Attack Surface

- In-vehicle systems communicate with the outside world in multiple ways
- Vehicles can be hacked with physical or remote access through Bluetooth, OBD-II , RF signals, and more

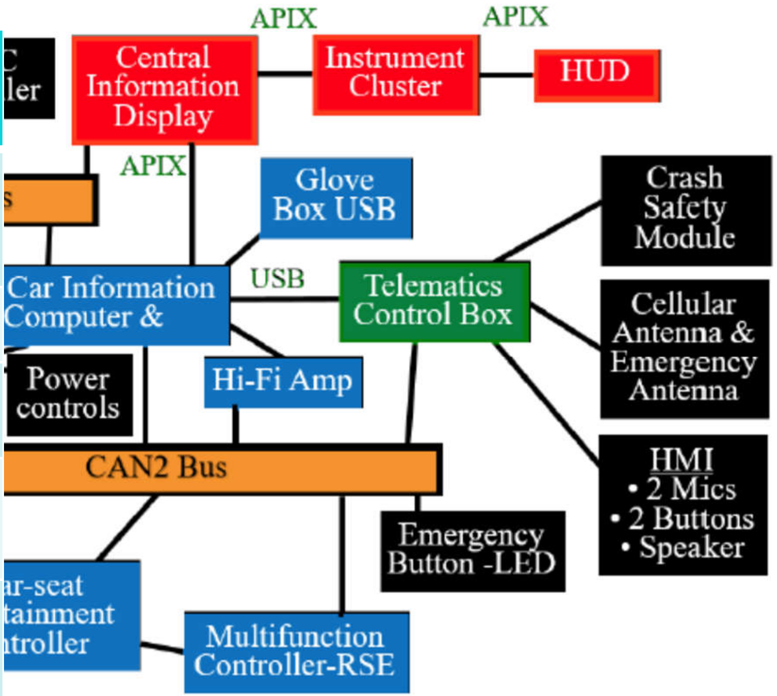


Automotive: V2X Communication

- 2002: American Society of Testing and Materials (ASTM) published WLAN based V2X communication standard ASTM E-2213.
- 2004: IEEE released initial standard for 802.11p standard.
- 2007: IEEE introduced 1609.X standard. It is based on 802.11p standard and provides Layer 3 and Layer 4 of the OSI. Also known as WAVE (Wireless Access in Vehicular Environments)
- 2012-2013: In Japan, Association of Radio Industries and Business released the ARIB STD-T109 standard. It provides V2V and V2I communication on 700MHz.
- 2017: 3GPP releases LTE-V based V2I and V2X communication physical layer standards.

Automotive: Communication

Domain	Description	End-to-End Latency Requirements	Bandwidth Requirements
Powertrain	Controls the components that generate power and transmit to the road	<10μs	Low
Chassis	Controls steering, brakes, suspension	<10μs	Low
Body and Comfort	Radio, A/C, Window, Seat, and light controls	<10ms	Low
Driver Assistance and Driver Safety	Controls systems designed to increase safety	<250μs or <1ms Depending on the system	20-100 Mbps per camera
Human-Machine Interface	Controls displays and other interfaces that interface with the driver or passengers	<10ms	Varies by system, the requirements are increasing



Automotive Communication

1983 : CAN (Controller Area Network)

CAN is a shared serial bus running at up to 1Mbps. It was developed by Bosch and standardized in multiple ISO standards.
It has the disadvantages of relatively low bandwidth and being a shared media.
CAN is used in powertrain, chassis, and body electronics.

2001: LIN (Local Interconnect Network)

It is a serial bus. It runs at 19,200 baud and requires only one shared wire (instead of the 2 for CAN).
LIN is a master-slave architecture.
Used for body electronics (mirrors, power seats, accessories).

2005: FlexRay

FlexRay is a shared serial bus running at up to 10Mbps.
It has the advantage of having higher bandwidth than CAN, but the disadvantage of higher cost and being a shared media.
FlexRay is used in high-performance powertrain and safety (drive-bywire, active suspension, adaptive cruise control).

2012: CANFD

First CAN-FD controller available in 2013.
Similar costs as for classic CAN
Higher bandwidth
Small impact on current SW and applications
Physical layer and structure of topologies can be maintained

2001: MOST (Media Oriented Systems Transport)
MOST has a ring architecture running at up to 50Mbps using either fiber or copper interconnects. Each ring can contain up to 64 MOST devices

Used in camera and video connections.

1994: LVDS (Low Voltage Differential Signaling) –
LVDS has been gaining use in the automotive market as a replacement for most

Automotive Communication

Low data rate control

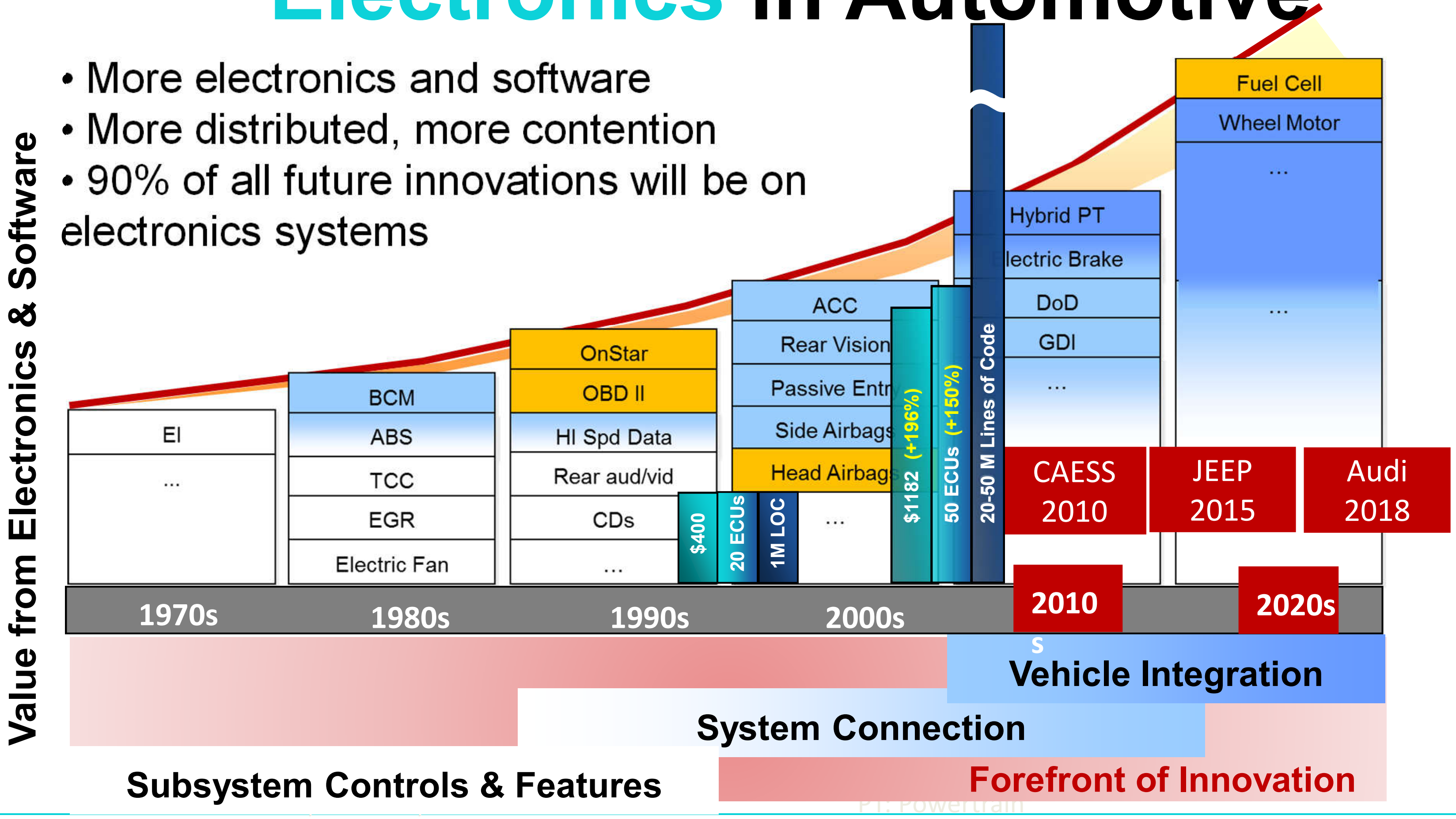
Technology	Data Rate	IP Ownership	Media	Topology	Usage
LIN	40kbps	LIN Consortium	Single wire	P2P	Body electronics
CAN	1Mbps	ISO-11898 Bosch	UTP	Shared	Power train (Engine, transmission, ABS)
CAN-FD	2.5Mbps	Bosch	UTP	Shared	Power train (Engine, transmission, ABS)
FlexRay	10Mbps	ISO-17458 FlexRay Consortium	UTP	Shared	High-perf power train (safety, drive by wire, active suspe nsion, ACC)

High cost/proprietary

Technology	Data Rate	IP Ownership	Media	Topology	Usage
MOST	150Mbps	SMSC	POF	Ring	Infotainment
FPDLink LVDS	655Mbps-3 Gbps	TI/National	Shield coax	P2P	Camera/display

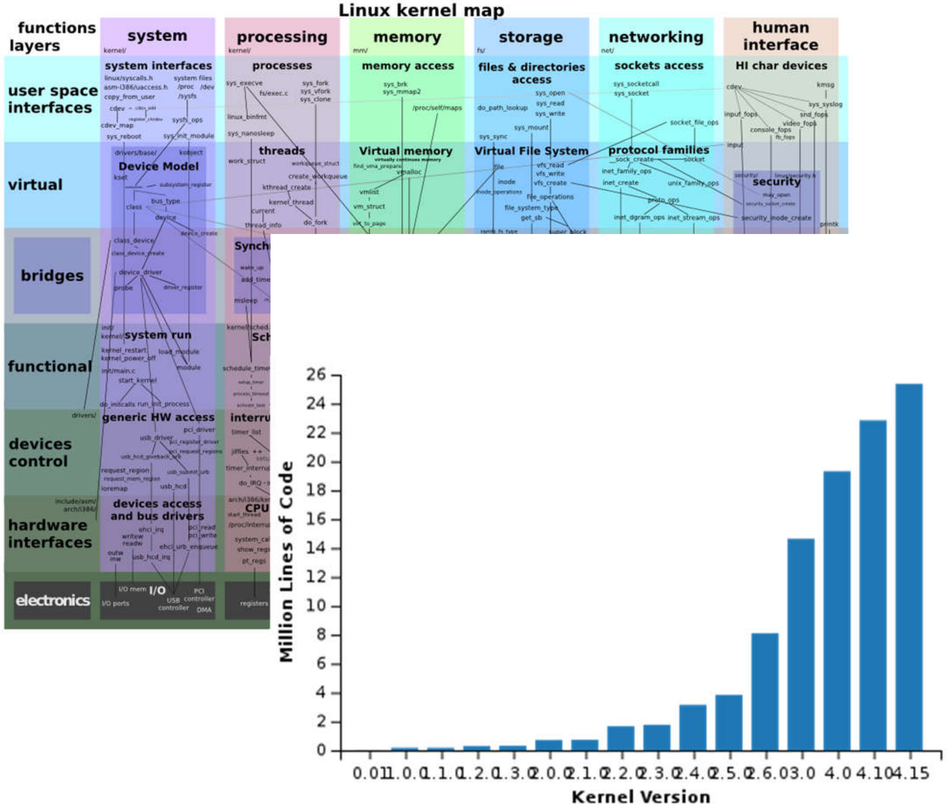
Electronics in Automotive

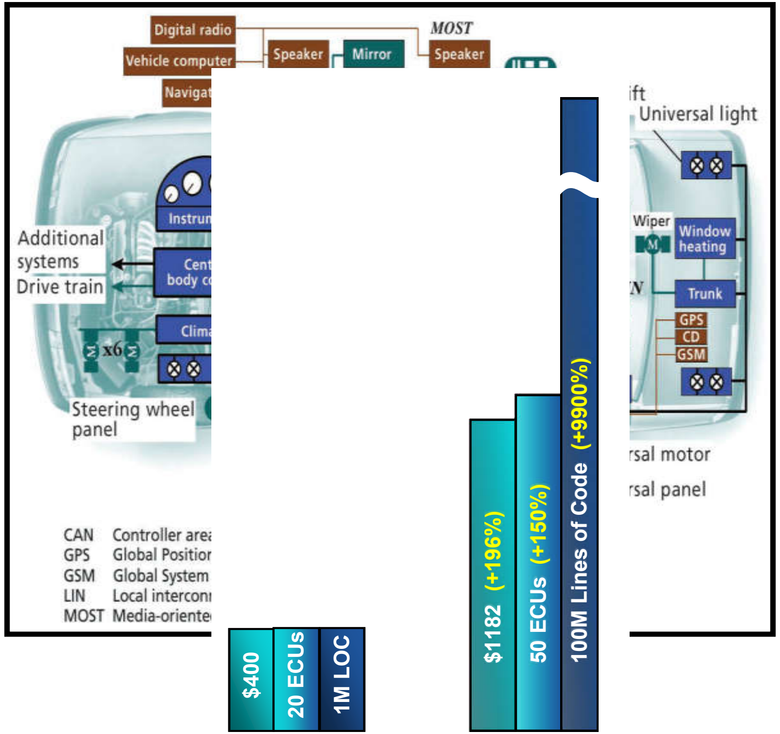
- More electronics and software
- More distributed, more contention
- 90% of all future innovations will be on electronics systems



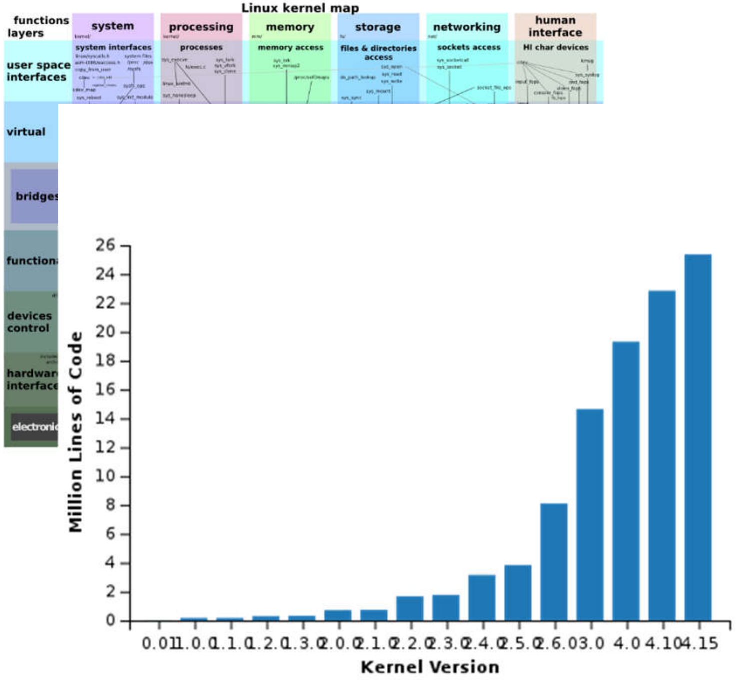
[source: Qi Zhu,ISPD]

Challenges in Automotive

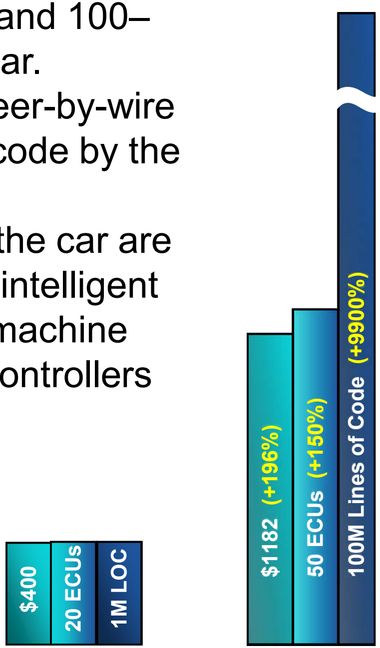




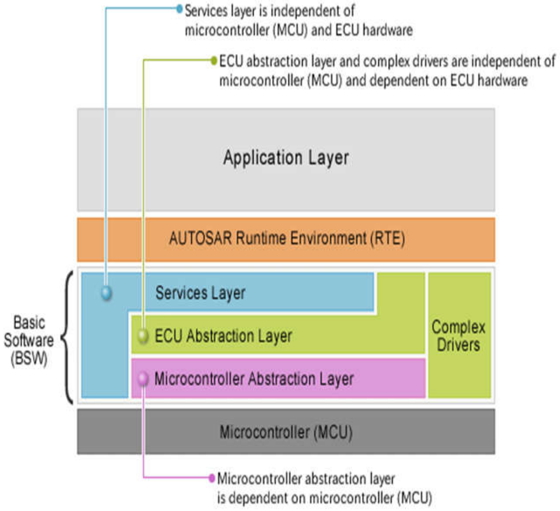
Challenges in Automotive



- Average of 50–60 ECUs, 80 chips, and 100–300 MBs of binary code in today's car.
- Some systems like safety critical steer-by-wire already feature 3–4 million lines of code by themselves.
- The highest powered computers in the car are no longer infotainment systems but intelligent ECUs powering sensor fusion and machine learning and consolidated domain controllers

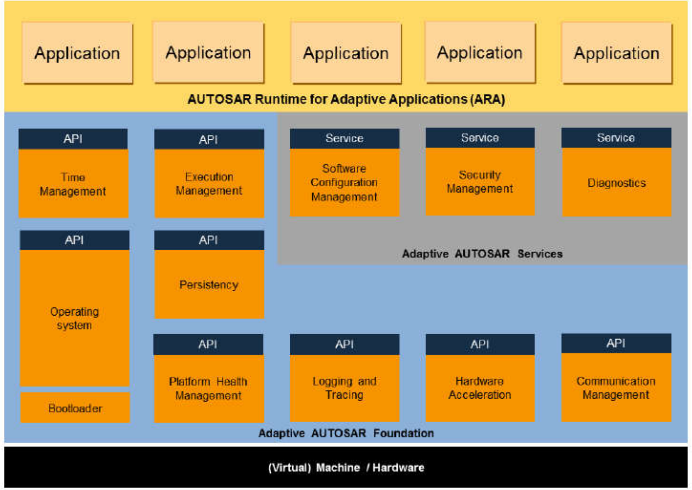


AUTOSAR Architecture: Standards Methodologies



AUTOSAR (Automotive Open System Architecture)

- Highest real-time requirements
- Lowest computing power requirements



AUTOSAR (Automotive Open System Architecture)
Execution Management
•Persistency
•Communication Management
•Platform Health Management
•Diagnostics
Highest computing power requirements

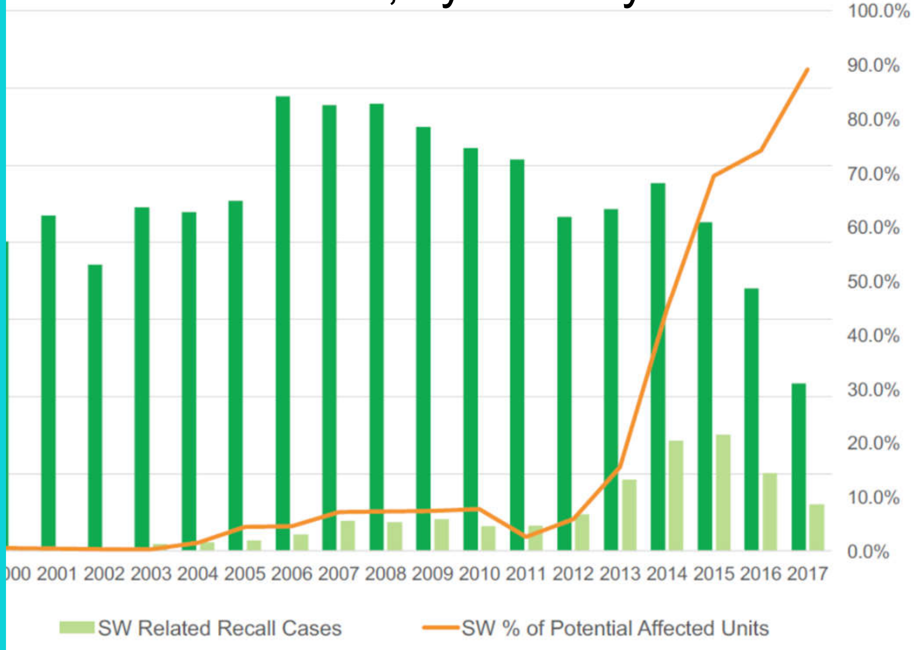
Infotainment Systems and c onnectivity OS

- Low safety criticality
- No real-time requirements
- High computing power
- Runs on android or linux

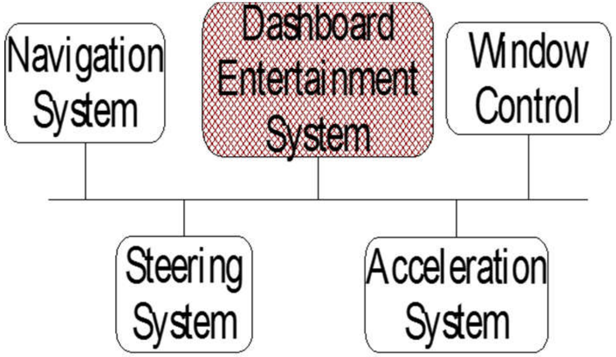
Challenges in Automotive

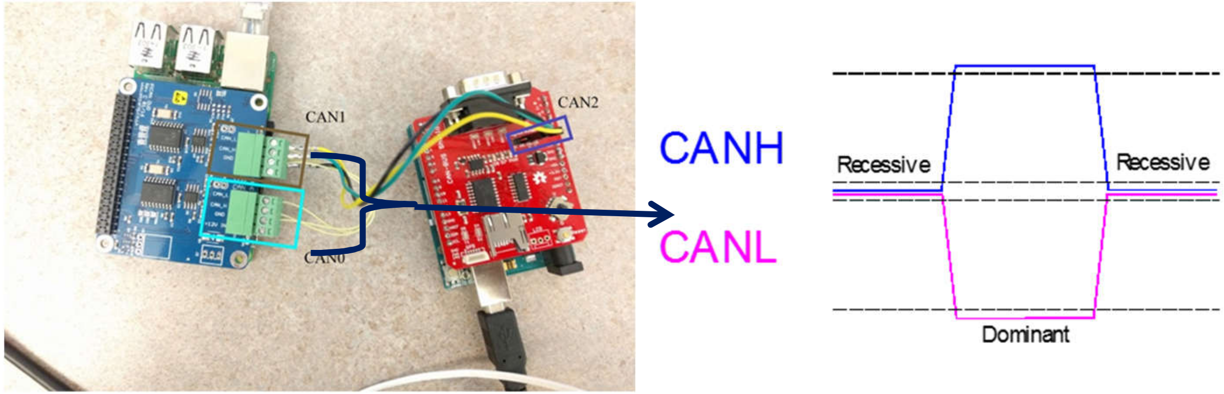
- More problems in vehicle electronic systems
  - Recalls related to electronic systems tripled in past 30 years.
  - Hard to diagnose: Debugging the ECUs is difficult, more than 50% of the failed ECUs passed the testi ng phase.
- **Secure Over-the-Air (OTA) software updates**
- Need for Standard Methodologies and tools
  - Modeling, analyzing and verifying complex system behavior with formal models.
  - Optimizing performance metrics such as, reliability, cost, security, energy, extensibility.

Total recall cases, by model year



Threat model of CAN Bus

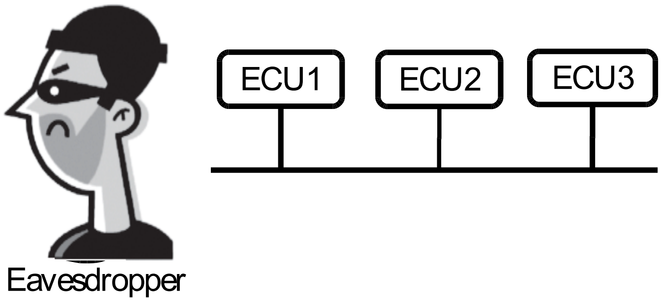




ECUs are composed of a processing element connecting to an actuation and a telemetry interface of a component.

- Hitting the brakes pedal should tell the braking system to actuate the brake disks.
- The interactive dashboard system controlling the climate of the car.

Threat model of CAN Bus

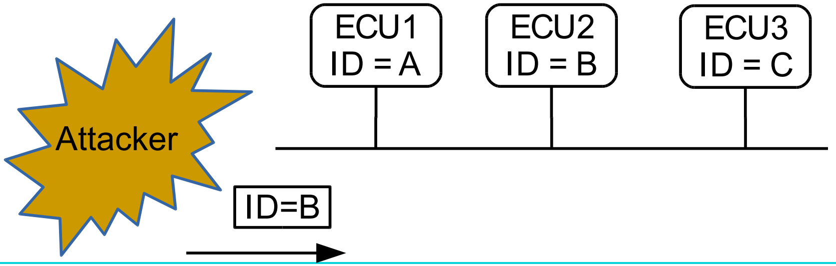


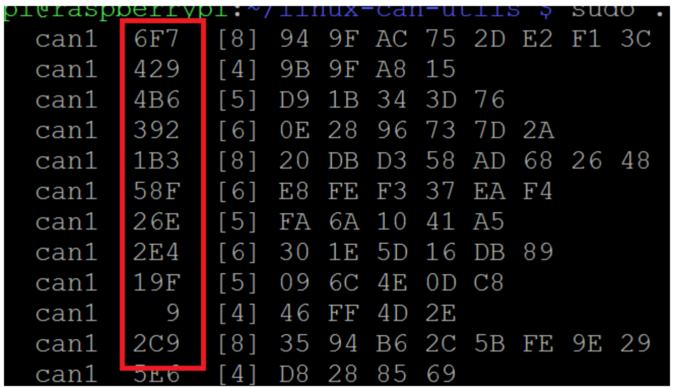
Eavesdropping

- A device broadcasts its message to the entire network.
- No encryption
- No effort to eavesdrop the communication.

Stealing Identifiers

- A device broadcasts its message to the entire network.
- No encryption
- No effort to eavesdrop the communication.





CAN Bus Based Attacks - Denial of Service Attack

- Malicious CAN node, CAN2 can interrupt the legitimate communication
- between CAN 0 and CAN1. CAN0 is forced to terminate its transmission

```
pi@raspberrypi:~/linux-can-utils $ sudo ./cangen can0
write: No buffer space available
pi@raspberrypi:~/linux-can-utils $
```

```
can1  7DF  [8]  02 01 05 00 00 00 00 00
can1  7DF  [8]  02 01 05 00 00 00 00 00
can1  7DF  [8]  02 01 05 00 00 00 00 00
can1  7DF  [8]  02 01 05 00 00 00 00 00
can1  7DF  [8]  02 01 05 00 00 00 00 00
can1  7DF  [8]  02 01 05 00 00 00 00 00
can1  7DF  [8]  02 01 05 00 00 00 00 00
can1  7DF  [8]  02 01 05 00 00 00 00 00
can1  7DF  [8]  02 01 05 00 00 00 00 00
can1  7DF  [8]  02 01 05 00 00 00 00 00
can1  7DF  [8]  02 01 05 00 00 00 00 00
can1  7DF  [8]  02 01 05 00 00 00 00 00
can1  7DF  [8]  02 01 05 00 00 00 00 00
can1  7DF  [8]  02 01 05 00 00 00 00 00
can1  7DF  [8]  02 01 05 00 00 00 00 00
```

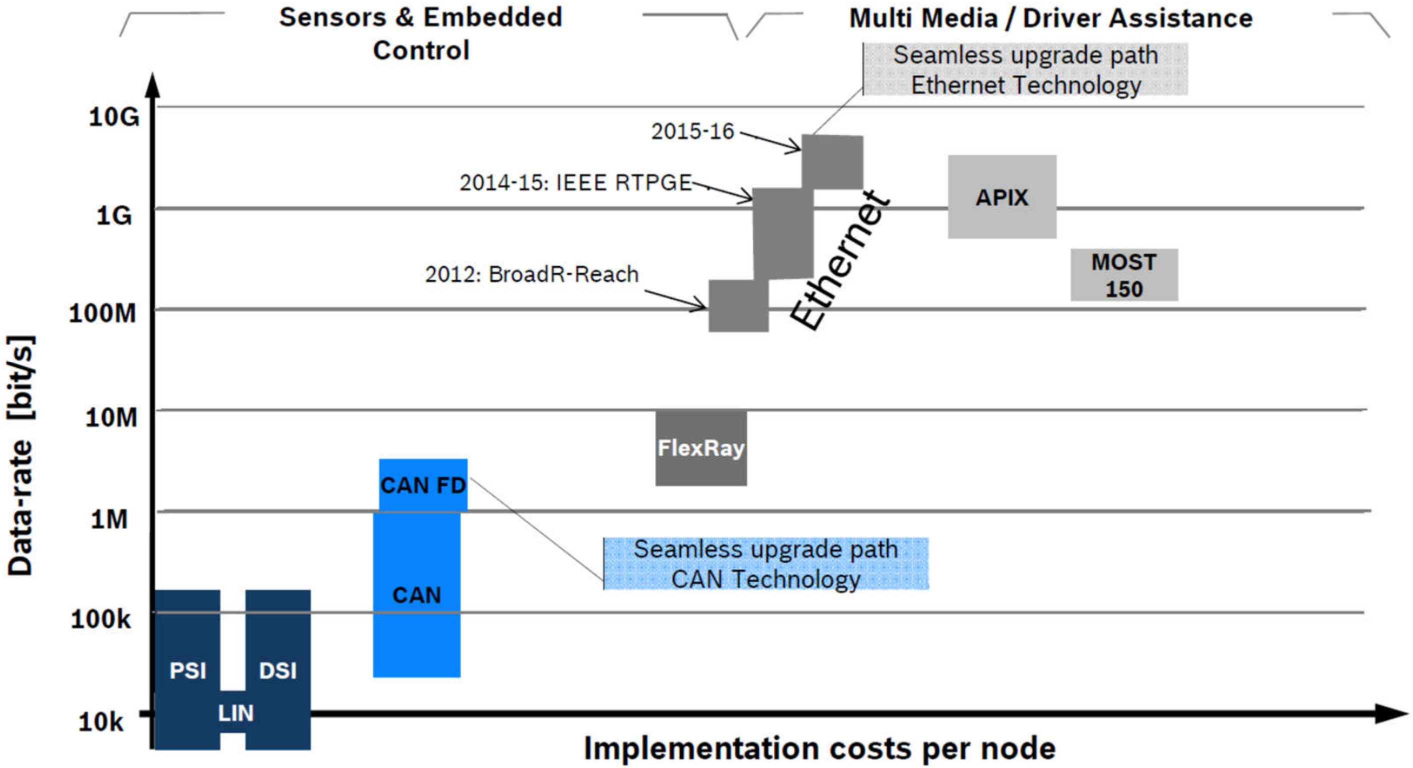
Automotive Security -Research Directions

- New capabilities in hardware and systems
  - Accelerated product development drives the need for early detection of problems
  - Market demand for connected vehicles and mobile applications requires use of new technology and development practices
  - Quality, security and safety become key concerns for developers.
  - Reconfigurable functional units
  - Automotive Ethernet.
  - Secure Architecture
- Hardware security capabilities
  - Accuracy and speedy insight into quality defects and security vulnerabilities.
  - Identification
  - Encrypted communication
  - secure boot
  - Bandwidth
  - PUF technologies for authentication.
  - Trusted platform module TPM
- Automation
  - All manual processes introduce avoidable delays and opportunity for human failure
  - Automation enable focus on quality, security and safety into System development life cycle (SDLC).
  - Continuously test with depth and speed
  - Implement features securely more than adding on security features.
  - Secure code while developers work, rather than after they're done – Specific to software.
- Future of CARS
  - Connected Cars
  - Autonomous vehicles

Automotive Security -Research Directions

- To get a competitive edge, intelligent vehicle manufacturers must meet demanding communication requirements, including safety, resilience, security, scalability, fault tolerance, and fast data transmission.
- Security should be part of the architecture design, embedded in multiple system layers.
- Develop open, flexible architectures for security, safety and mission-critical applications in un-armed v ehicles UAVs

Automotive Future Communication System Landscape

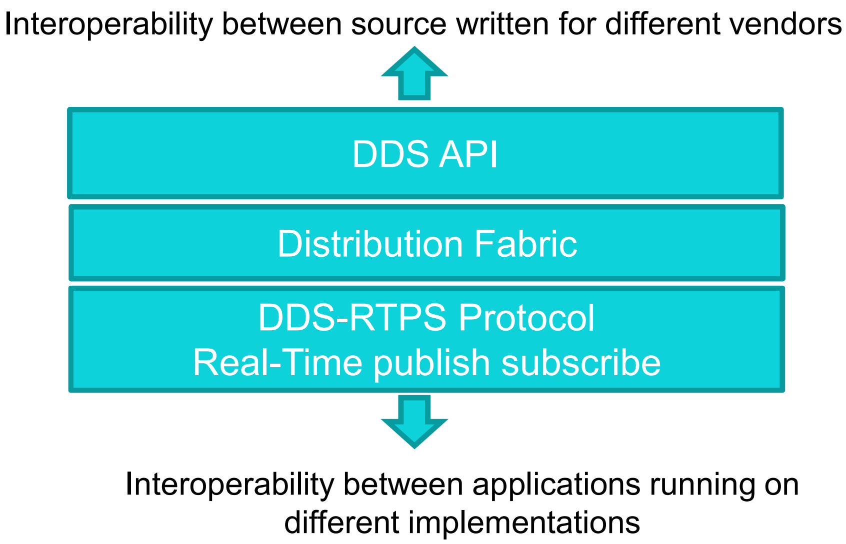


Automotive Future Communication System Landscape

	1st Generation	2nd Generation	3rd Generation
	Independent Subsystems and Diagnostics, ECU Flashing, Rear-view Cameras	Infotainment and Driver Assistance Systems	Ethernet as network backbone
Higher Layer Protocols	Diagnostics over IP (ISO 13400)	Energy Efficient Ethernet (IEEE 802.3ez) A/V Bridging Gen 1 Video comm Interface	Partial Networking A/V Bridging Gen 2 Time-triggered Ethernet
Data Link Layer Physical Layer	100 Base-Tx Fast Ether net (IEEE 802.3u)	BroadR-Reach (OPEN Alliance)	Reduced Pair 1 Gigabit (IEEE)
	2010	2015	2020

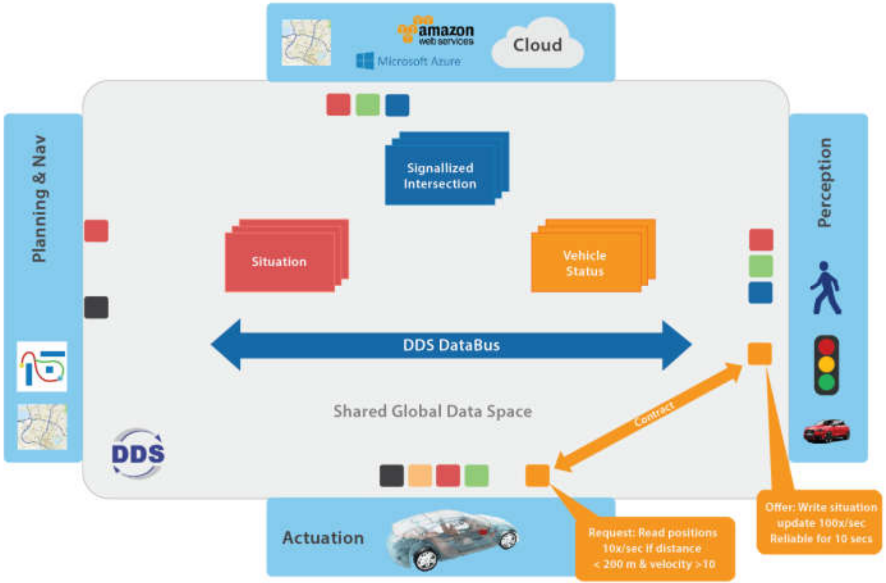
Architecture: Data distribution service

- A data centric middleware
  - Data is the interface.
  - Data centricity enables interoperation, scale and integration
- Instead of message centric system
  - Point to point
  - Client /server
  - Publish/subscribe
  - Queueing
- The Data Distribution services is the proven data connectivity standard for the IoT.



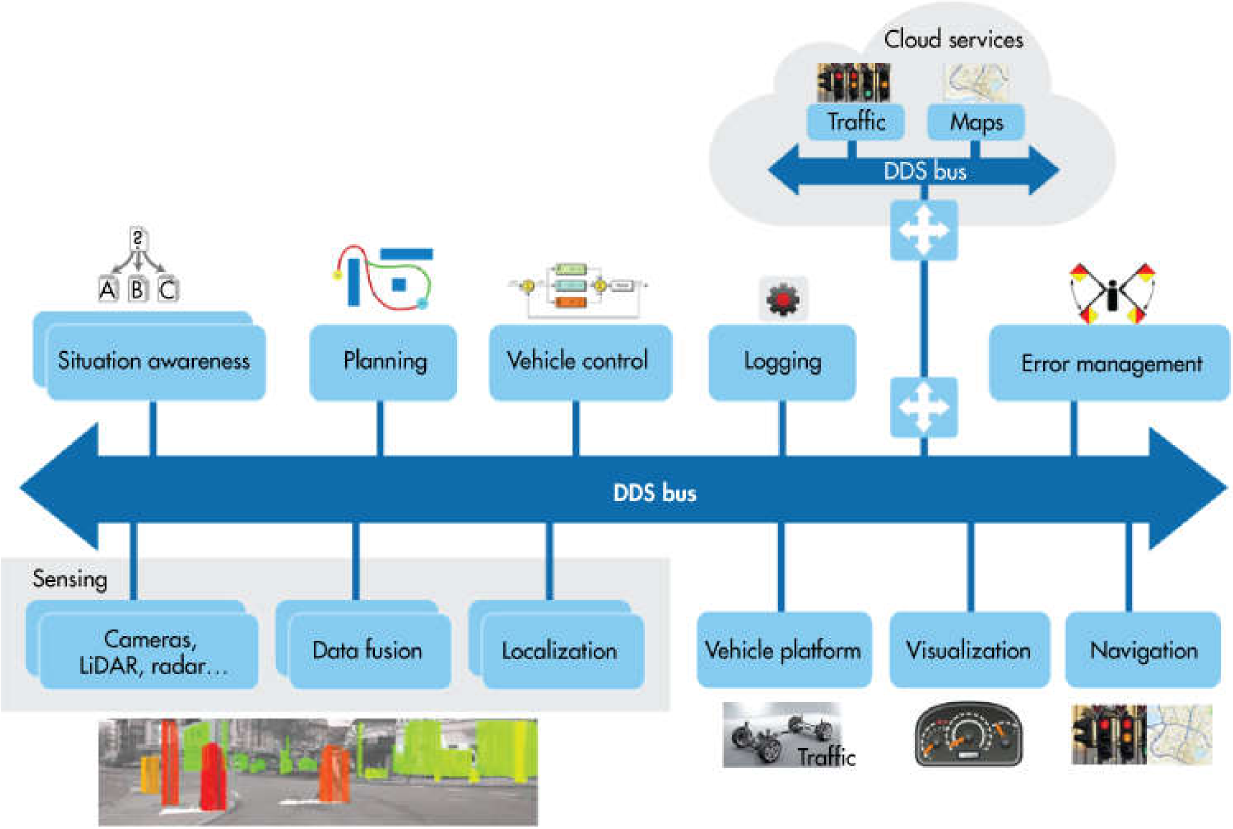
Architecture: Data distribution **service**

- Global data space
  - Automatic discovery
  - Read and write data in any OS, language, transport
  - Type aware
  - Redundant sources/sinks/nets
- No Servers
- QoS control
  - Timing, reliability, redundancy ordering filtering security.
- Connect vehicles to clouds and infrastructure.
- Performance/scale
  - Measure in ms or micro seconds
  - Or scale > 20+ applications or 10+ teams?
  - Or 10k+ data values?



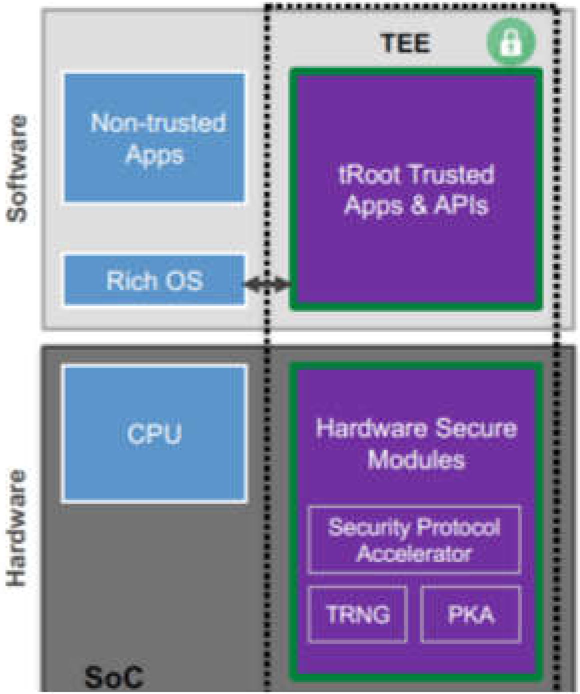
Architecture: System Integration

- Build security in from the start.
- Data flow level security
  - Control read and write access to each data item for each function
  - Ensures proper dataflow operation
- Complete protection
  - Discovery authentication
  - Data-centric access control
  - Cryptography
  - Tagging and logging
  - Non-repudiation
  - Secure multicast



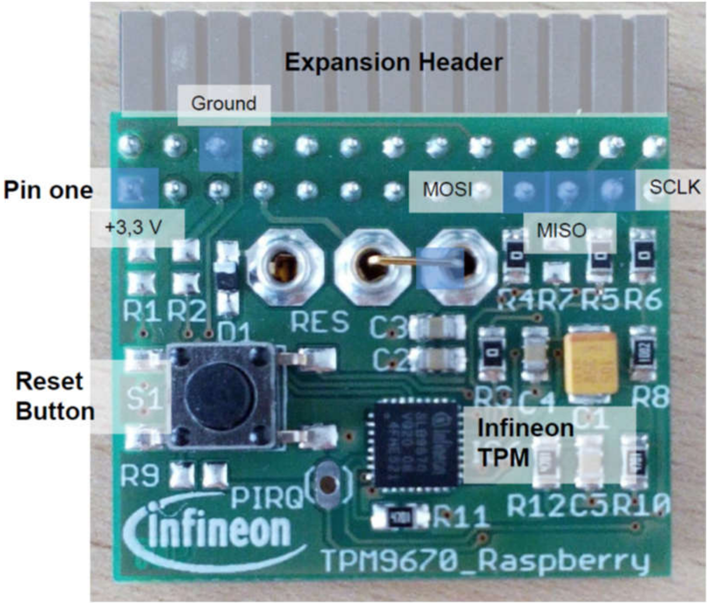
Hardware Support: Trusted Platform Module

- Standard hardware secure modules with root of trust provides an execution environment to
  - Root of trust hardware proveds SoCs with a unique identity.
  - Securely create, store and manage secrets
  - Extend trust to other internal and external entities
  - Multistage secure boot validates software and data integrity.
  - Secure authentication/ updates/ storage / debug enable in-the-field device manage ment.
  - Key management and crypto APIs provide secure access to cryptographic keys



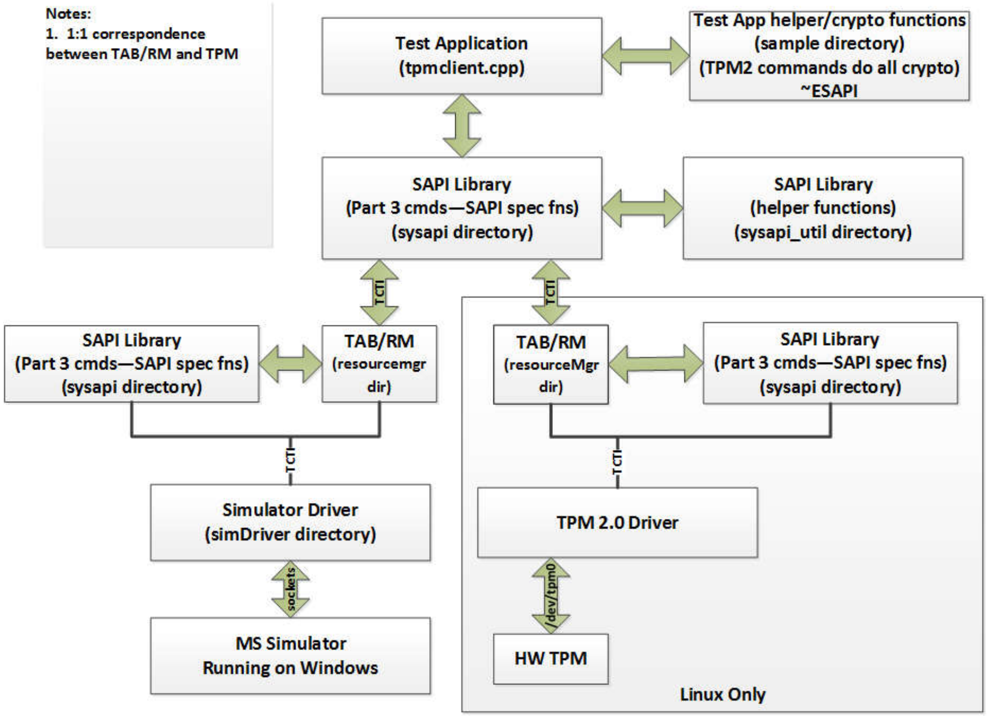
Hardware Support: Trusted Platform Module

- Trusted platform modules are cryptographic processors.
- Supports security functions such as, key generation, storage, symmetric and asymmetric encryption engine and hash algorithms.
- TPM integration into a platform can be found in the specifications of the Trusted Computing Group (TCG).
- TPM provides three groups to hold objects. Each hierarchy serves a   different use case:
    - Owner – Intended to be used by the IT dept. of an enterprise or the end user
    - Endorsement – Privacy sensitive area, to hold certification keys.
    - Platform – To be used by the platform ma nufacturer or the   vendor.

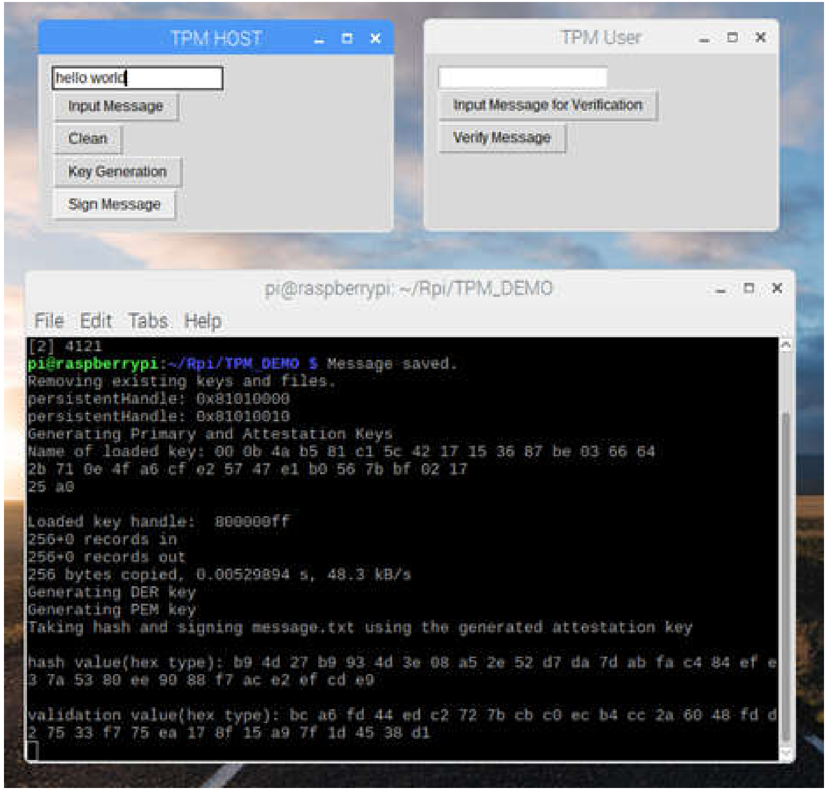


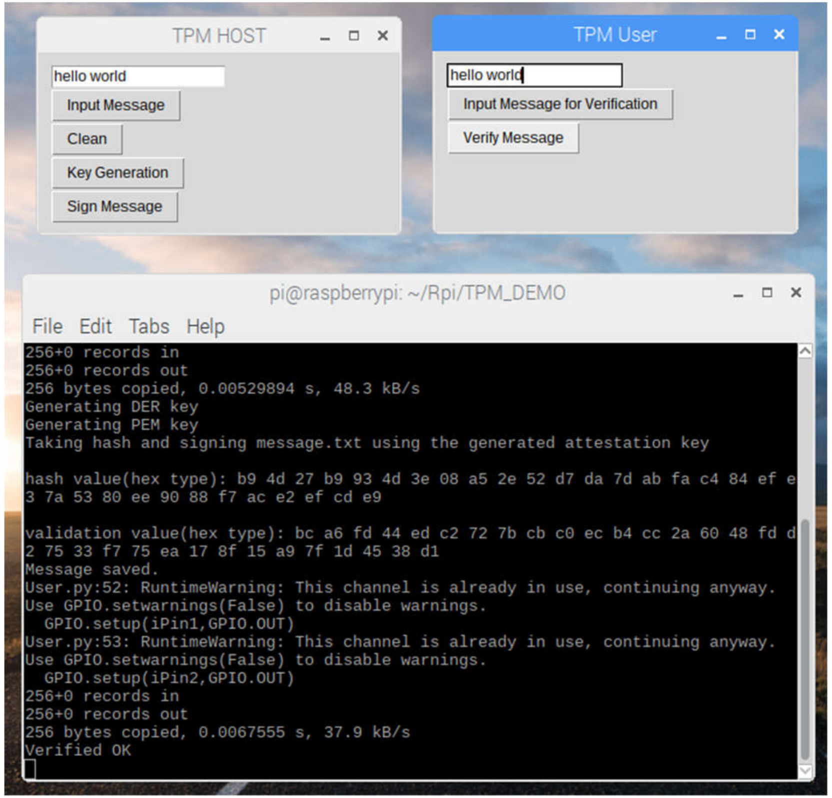
Hardware Support: TPM Software Stack

- TPM TCG Software Stack (TPM2-TSS) is an open source software stack that pro vides a System API (SAPI) to the TPM commands defined in the specifications.
- TPM-TSS is implemented as a library in C language and is composed of function calls that can be used by client code.
  - The software library
  - User land device resource manage ment daemon.
  - Tool implementation for TPM struct ures.

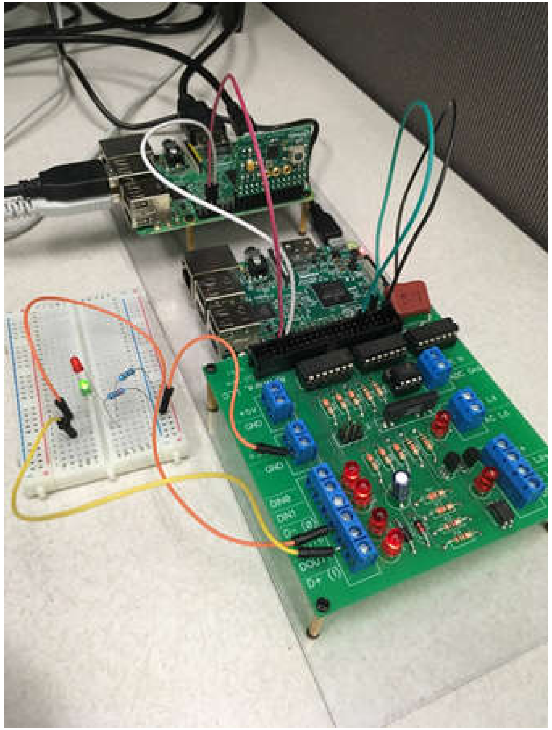


Signing and Data Verification Tool



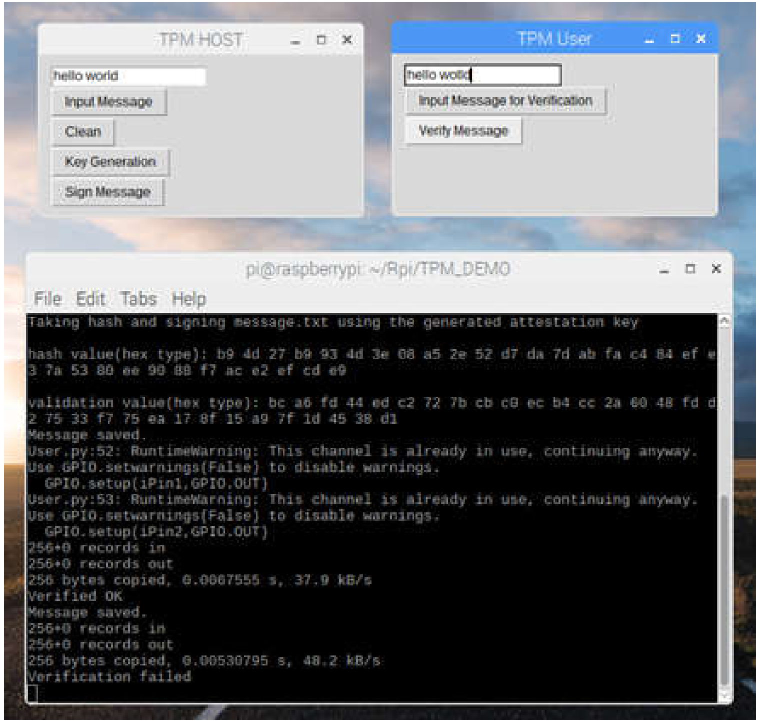


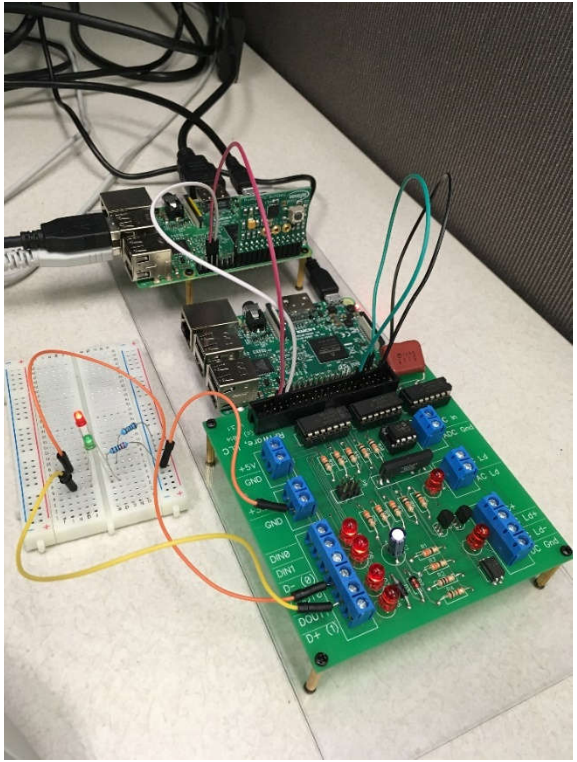
Successful Verification



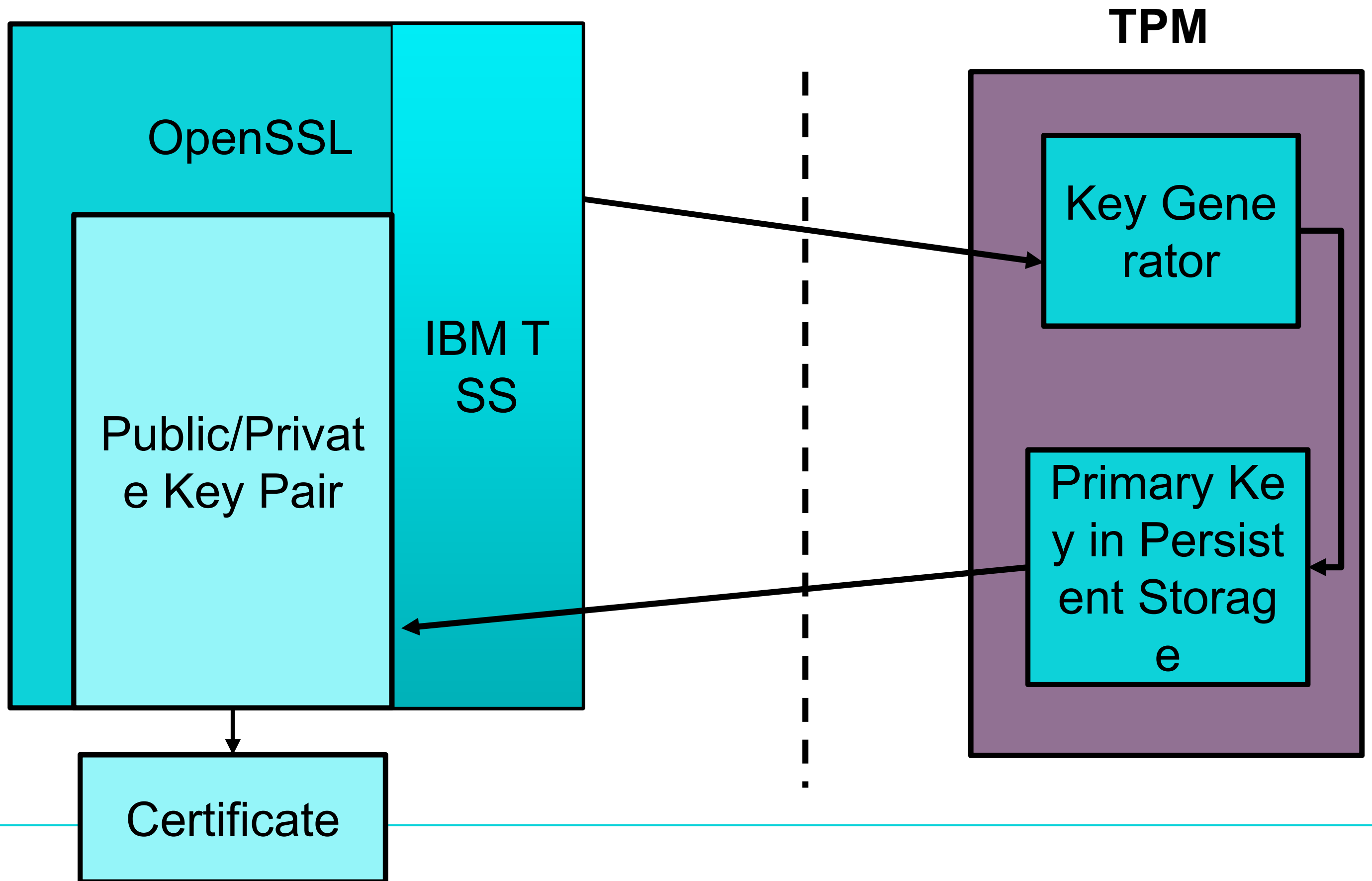
```
Removing existing keys and files.
persistentHandle: 0x81010000
persistentHandle: 0x81010010
Generating Primary and Attestation Keys
Name of loaded key: 00 0b 14 3a 39 5b 40 06 31 86 85 af 5a 7d e2 83
49 55 21 c4 b8 9e 74 da 5b 63 de 5a 14 64 54 bf
96 43

Loaded key handle:  800000ff
256+0 records in
256+0 records out
256 bytes copied, 0.00425398 s, 60.2 kB/s
Generating DER key
Generating PEM key
Taking hash and signing message.txt using the generated attestation key

hash value(hex type): a7 a3 d0 06 d0 b3 78 72 52 6f 57 52 90 14 86 4b 1d a5 14 e9 e0 07 99 eb 4f 8b 71 d0 80 c5 a9

validation value(hex type): ee 5f d6 d1 ef 32 ee 64 6e 5b e5 c6 54 9f 0b 2e d7 7d 68 a6 13 95 f9 28 df f8 47 e6 1a
Extracting signature from the TPMU_SIGNATURE structure in sign.bin
256+0 records in
256+0 records out
256 bytes copied, 0.00239082 s, 107 kB/s
Verifying signature with openssl
Verified OK
```

Failed Verification

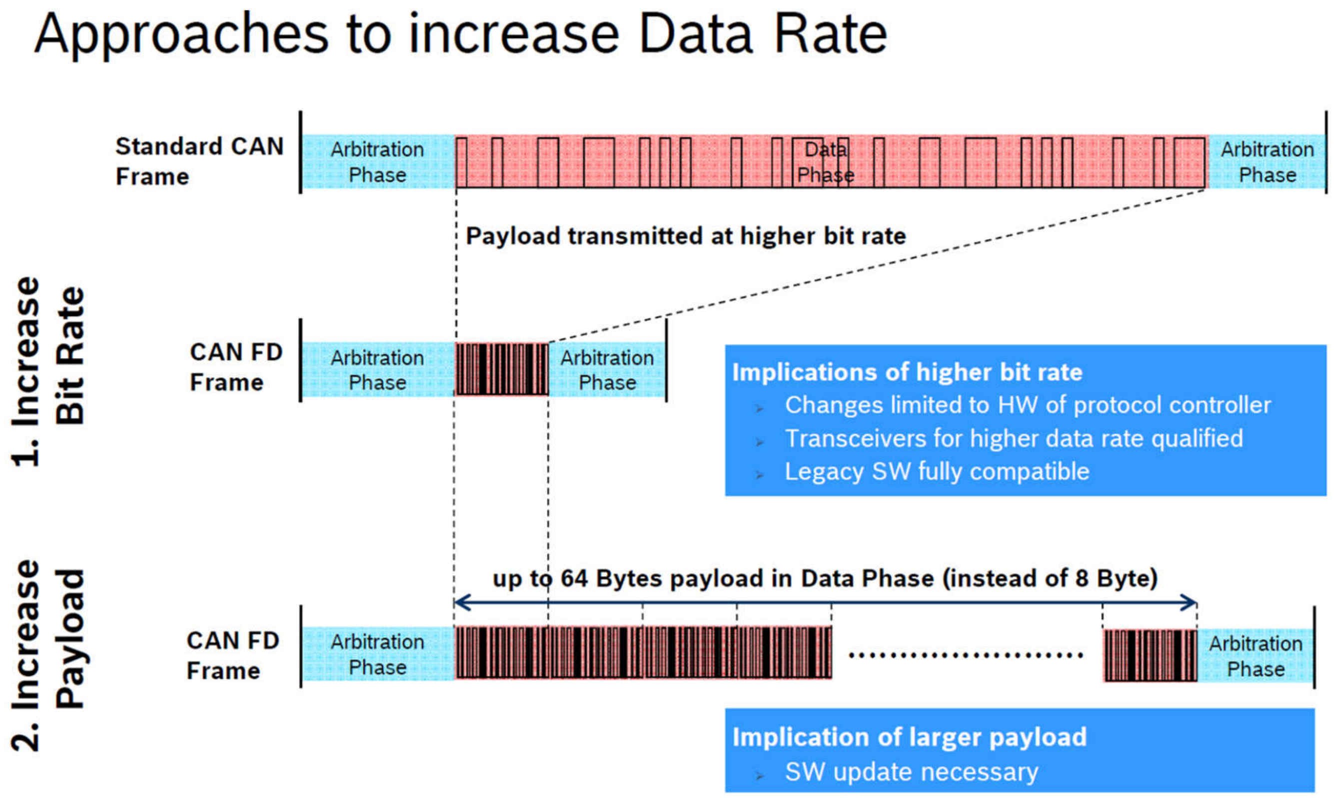




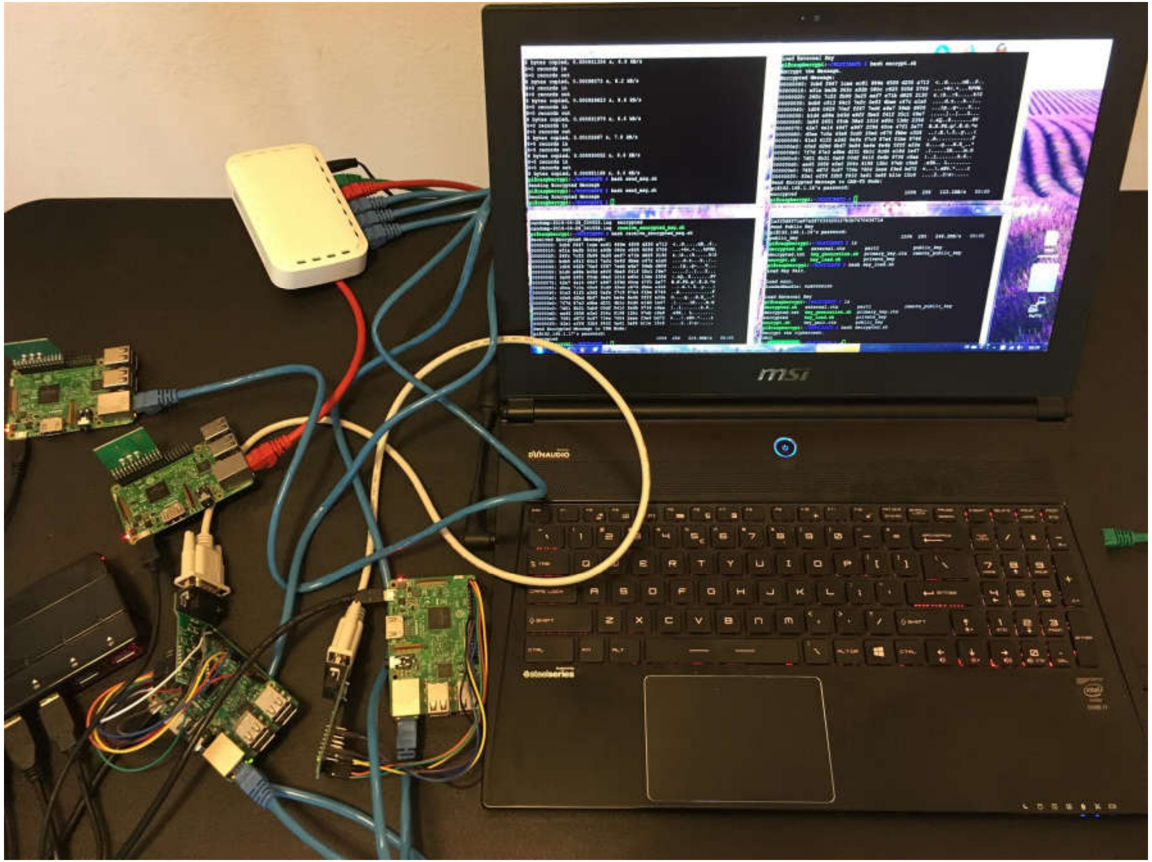
Certification Authority



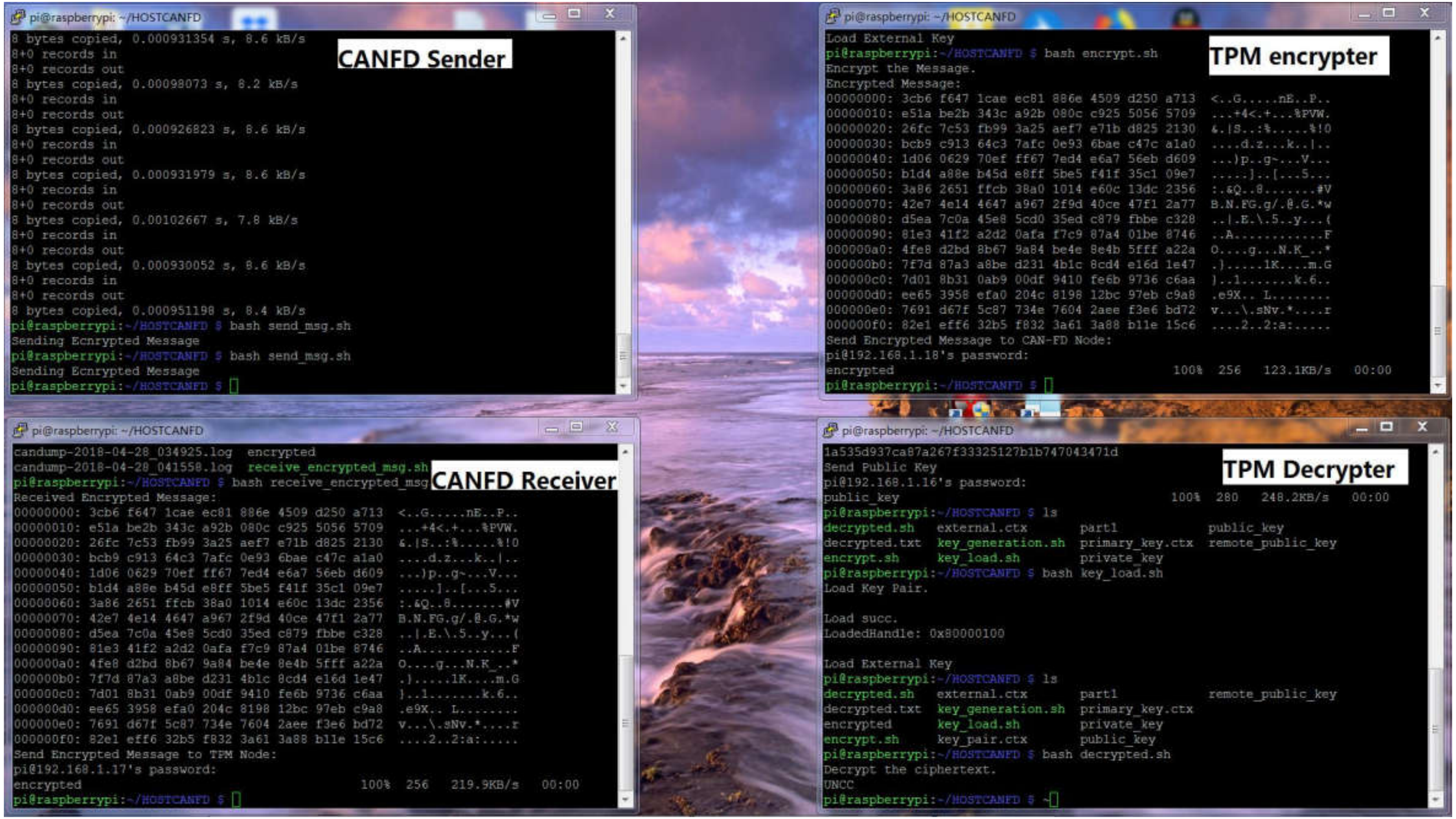
Intra vehicle communication over CAN-FD



Intra vehicle communication over CAN-FD



Intra vehicle communication over CAN-FD

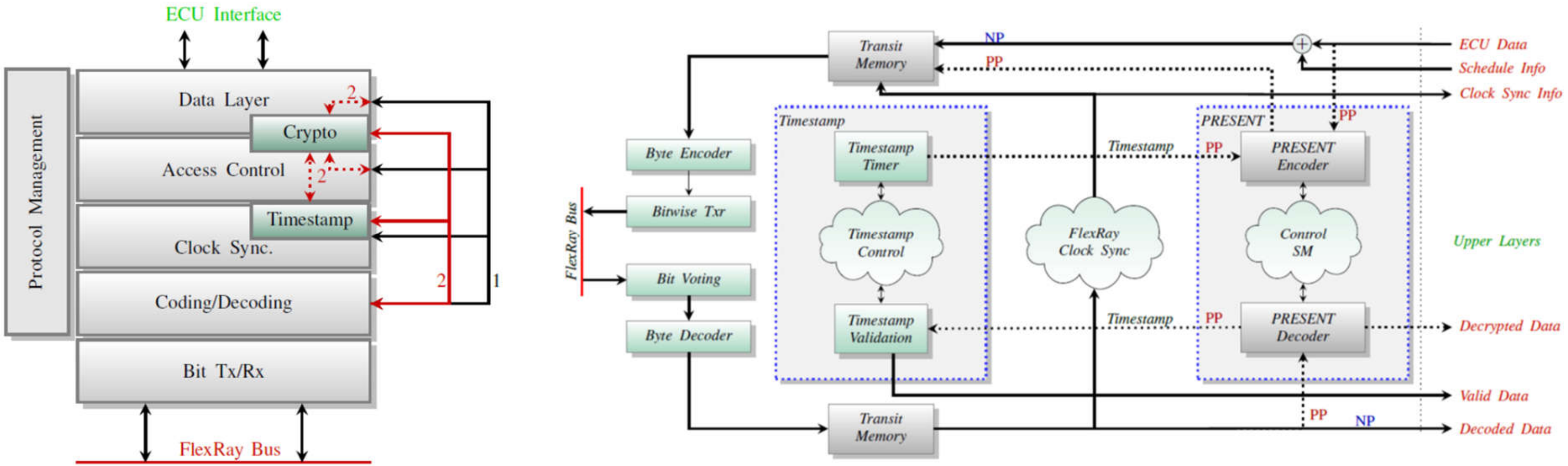


Intra vehicle communication over CAN-FD

DEMO

Automotive: FPGA accelerators

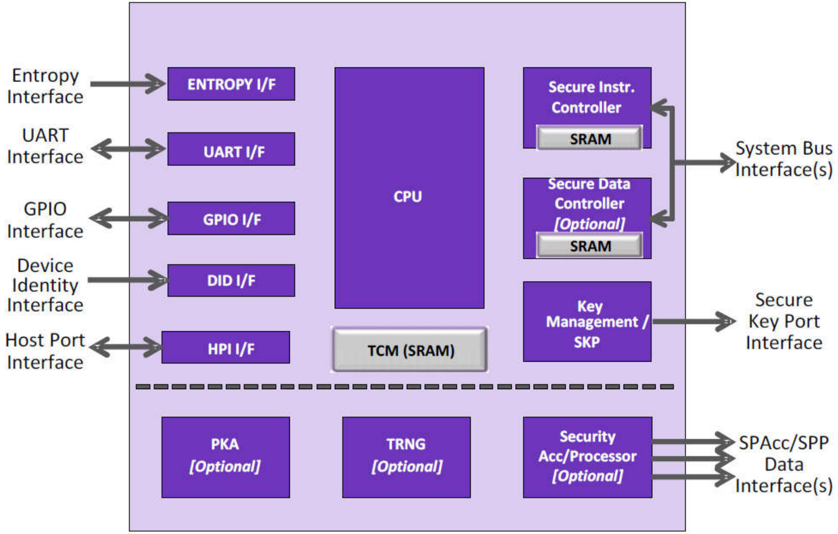
- FPGA based ECUs can integrate security such as data and secure boot transparently at the network and physical layer.
- The encrypted communication can meet real-time guarantees.



[source: Shreejith et.al, FPT 2014]

Automotive: Secure Boot

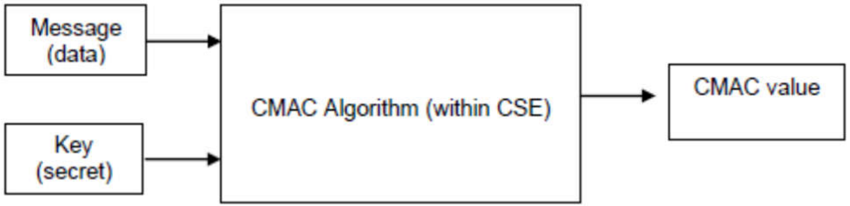
- **An HSM provides SoC ICs with unique identity and secure tamperproof environ ment.**
- **Create, store and use secrets critical to the system.**
  - Secure bootstrap
  - Secure access control
  - Secure authentication
  - Firmware integrity assurance
  - Secure storage
  - Secure debug and test access control



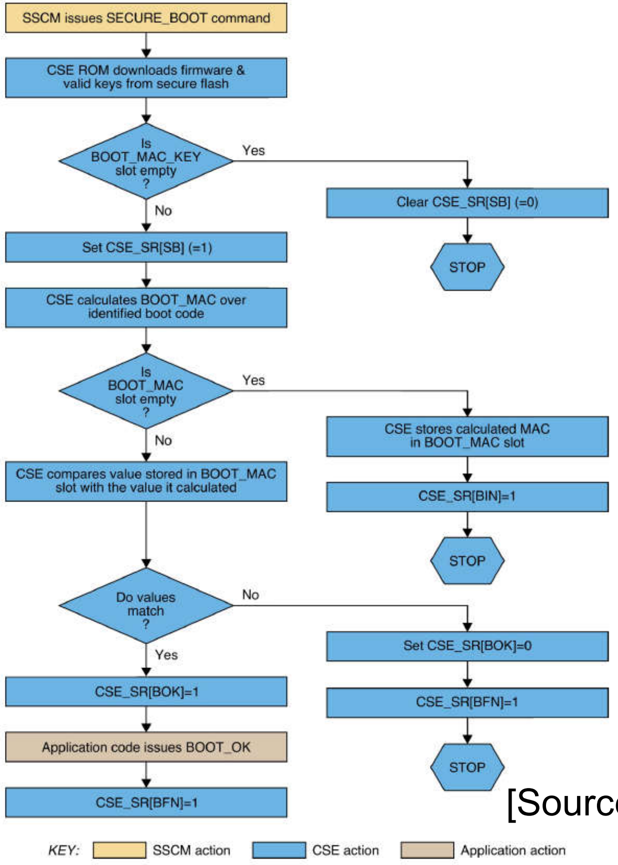
Secure reconfiguration of programmable logic

DEMO

Automotive: Cryptographic Service Engine



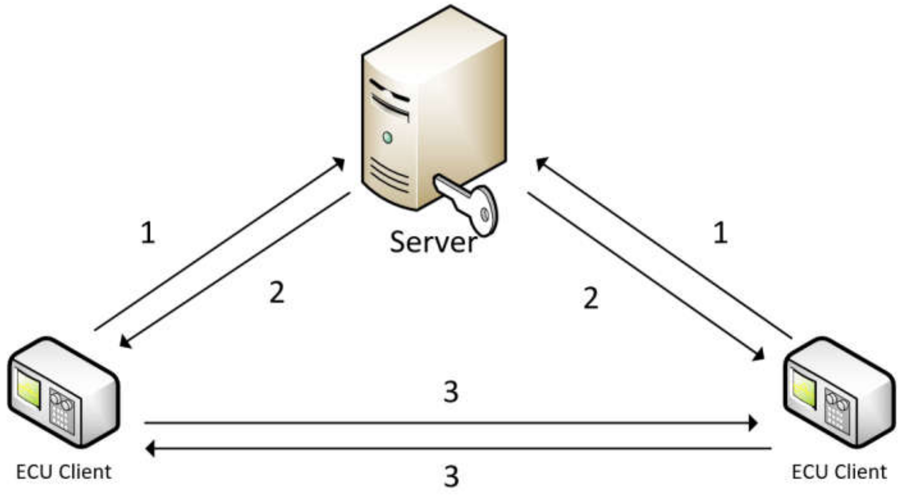
- Check bootloader for integrity and authenticity.
- Check flash memory for integrity and authenticity.
- Secure communication and data acquisition between central ECUs to Sensor ECUs.
    - Random number generator
    - Encryption



[Source NXP]

Automotive: Cryptographic Service Engine

- Using the server's stored ECC public key, each client generates ECDH symmetric key and sends its public key encrypted to the server.
- The server verifies the public key of each node and sends each node a list of verified public keys of nodes.
- The clients generate ECDH symmetric keys for each other and are able to communicate.



[source: Saqib et.al, Asian HOST 2017]

Thank You!!

PUF-Based Authentication and Secure Boot for IoT

Professor Jim Plusquellic

ECE, UNM

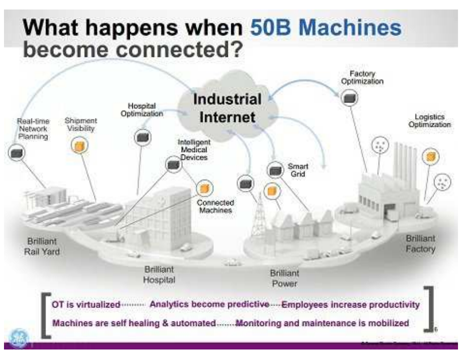
jimp@ece.unm.edu

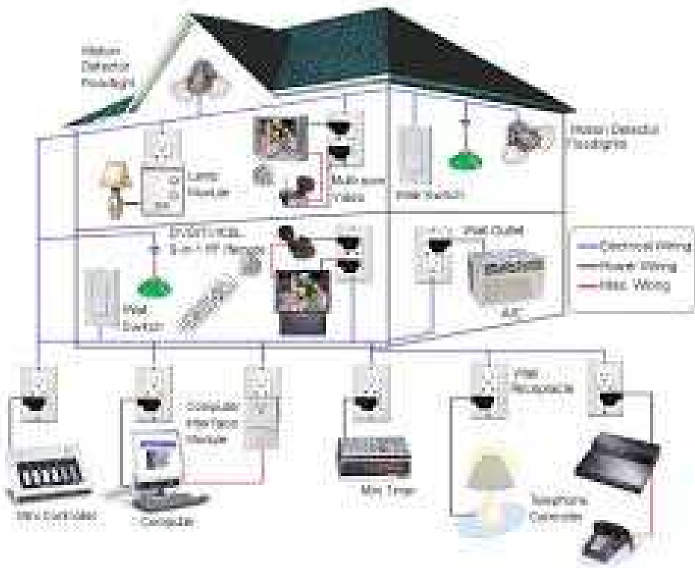
IoT Security and Trust Challenges

IoT defined (source wikipedia)

- A network of physical devices, vehicles, home appliances and other items embedded with electronics, software, sensors, actuators and connectivity, which enables these objects to connect and exchange data

RFID, Home automation, Industrial control (SCADA), vehicle V2V and V2X, smart buildings and cities, EMS, embedded medical, etc.





IoT Security and Trust Challenges

IoT Threats:

- Spoofing, mascarading, impersonation
- Malicious behavior and back-doors introduced by Hardware Trojans
- Information theft through the network or physical-layer side-channels
- Counterfeits, IC overbuilding and other forms of supply chain subversion
- Sabbatoge to the root-of-trust and illegal firmware updates

Countermeasures:

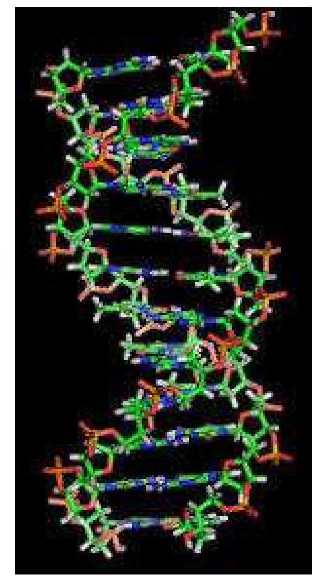
- Secure authentication
- Hardware Trojan screening methods, design obfuscation and tamper-evident verification methods
- Secure firewalls and side-channel-attack resistant logic styles
- Immutable, intrinsic identifiers and hardware metering protocols
- Non-NVM-based key generation and storage, and secure boot protocols

Physical unclonable functions (PUFs) can be used in many of these countermeasures

Physical Unclonable Functions

An inherent and unclonable instance-specific feature of a physical object

Akin to biometric features in humans, such as fingerprints, iris characteristics and DNA



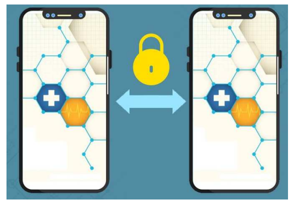
PUFs take advantage of *technical limitations* that exist in the physical process of fabricating integrated circuits

Even with *extreme* control over a fabrication process, no two physically identical instances of a chip can be created b/c of random and uncontrollable effects

PUFs Role in Information Security

PUFs are designed to generate **bitstrings** and **secret keys** for protocols that implement the basic tenets of information security:

- **Confidentiality**: Keeping information secret (Encryption)




- **Data Integrity**: Ensuring information has not been altered (Secure hashing)

- **Authentication**: Two forms: entity and message: Establishing identity through corroborative evidence (protocols)

- **Non-Repudiation**: Preventing the denial of previous commitments or actions (digital signatures)

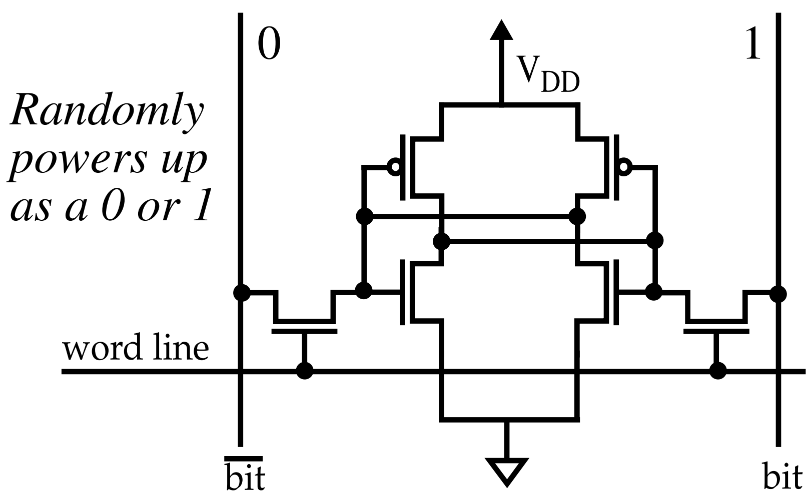
PUFs Defined

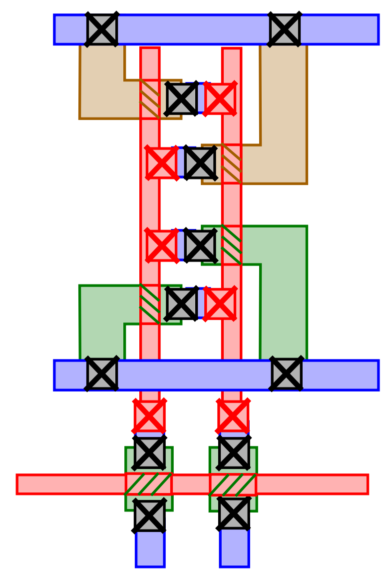
PUF Constructions: What do they look like and what do they leverage?

An **intrinsic PUF** is defined as a combination of

- A *physical source of randomness* (**Entropy**), i.e., an integrated circuit component that exhibits *within-die* variations
- A *measurement technique* that can convert small analog signal differences introduced by chip-to-chip/within-die variations into unique digital bitstrings

The **SRAM PUF** is the simpliest and requires no design changes





Symmetric and identical as drawn

PUF Statistical Metrics

Note that the instance-specific response of a PUF is affected by 1) chip-to-chip and within-die variations, 2) environmental conditions and3) wear-out effects

PUF responses are subjected to statistical testing to evaluate their:

- **Uniqueness:** Responses from different different chips are compared

```
1 0 1 0 0 1 0 1 1 0
1 1 0 0 0 1 1 1 0 1
---------------------------
0 1 1 0 0 0 1 0 1 1 = 5/10 = 50%
```

(Line 1: $Chip_0$ bitstring during enrollment under conditions α; line 2: $Chip_1$ bitstring during enrollment under conditions α; result: Inter-chip hamming distance, HD_{inter}, ideal is 50%)

- **Randomness:** Responses from the same PUF instance using different challenges

  NIST statistical tests are typically used

- **Reproducibility**: Responses from the same PUF instance using the *same* challenges but under different *environmental conditions*

```
1 0 1 0 0 1 0 1 1 0
1 0 1 0 1 1 0 1 1 0
---------------------------
0 0 0 0 1 0 0 0 0 0 = 1/10 = 10%
```

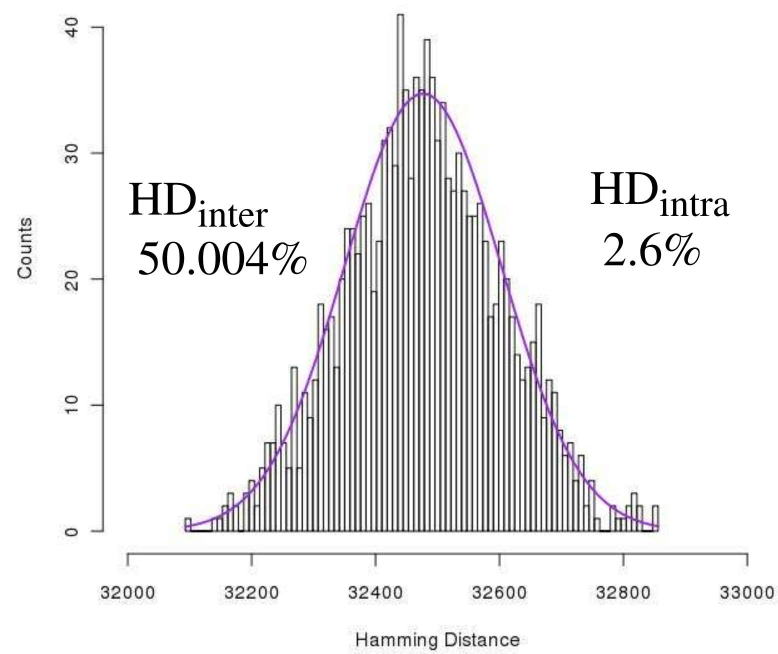
(Line 1: $Chip_0$ bitstring during enrollment under conditions α; line 2: $Chip_0$ bitstring during regeneration under conditions β; result: Intra-chip hamming distance, HD_{intra}, ideal is 0%)

PUF Inter-chip HD Example

With $N_{puf} = 50$ chips, the histogram is created from 50*49/2 = 1225 HD_{inter} values:

Ideal Ave. HD
32,474 bits

Actual Ave. HD
Mean: 32,477 bits
Std. Dev.: 126 bits



F. Saqib, M. Areno, J. Aarestad and J. Plusquellic, "An ASIC Implementation of a Hardware-Embedded Physical Unclonable Function", IET Computers & Digital Techniques, Vol. 8, Issue 6, Nov. 2014, pp. 288-299

Note that the distribution is actually characterized as **binomial** and not Gaussian

The expected standard deviation *std* of a binomial is given by

$$std_{binomial} = \sqrt{np(1-p)} = \sqrt{64948 \bullet 0.5 \bullet 0.5} = 127.4$$

Entropy and MinEntropy

Randomness is more difficult to evaluate than reliability and uniqueness, and requires a suite of tests

Entropy and **MinEntropy** are measures of the disorder or randomness of a random variable *X* with probabilities p_i, ..., p_n, (also measures information content):

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i \qquad \textbf{Entropy}$$

$$H_\infty(X) = \min_{i=1}^{n}(-\log_2 p_i) = -\log_2(\max_i(p_i)) \qquad \textbf{MinEntropy}$$

For example, assume you analyze a set of 20 binary bits (01110111101010011101) produced by a random variable and obtain the following 'occurrence' results:

- 8 0's (or 8/20 = 0.40)
- 12 1's (or 12/20 = 0.60)

We compute Entropy and MinEntropy using the above formula as:

Entropy = 0.60*$\log_2$(0.60) + 0.40*$\log_2$(0.40) = 0.4422 + 0.5288 = 0.971

MinEntropy = -$\log_2$(0.60) = 0.7370

PUF Statistical Metrics for Randomness

There are MANY ways to compute Entropy w.r.t. PUFs, and you will see different methods used in the literature

chip/bit #	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	*H(x)*
C1	0	1	0	0	1	0	1	0	0	0	1	1	1	1	0	0	0	1	1	1	1.000
C2	1	1	0	1	1	1	1	1	0	1	0	0	1	1	0	0	0	1	0	0	0.993
C3	1	1	0	0	1	0	1	0	0	0	0	1	0	1	1	0	0	1	0	0	0.971
C4	1	1	1	1	0	1	0	0	1	1	1	0	0	0	0	0	0	1	1	1	0.993
C5	1	0	0	1	1	0	0	0	1	0	0	1	1	1	0	1	0	1	1	0	1.000
C6	1	1	0	0	0	1	0	1	1	0	1	1	0	1	1	0	1	0	0	0	1.000
C7	0	1	1	0	1	0	1	1	1	1	0	1	1	0	1	0	1	1	0	0	0.971
C8	0	1	1	1	0	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0.971
C9	0	0	0	0	1	1	0	1	1	1	0	0	1	0	0	0	0	0	0	0	0.881
C10	1	0	1	0	1	0	1	0	1	1	1	1	0	0	1	0	0	0	0	1	1.000
H(x)	0.97	0.88	0.97	0.97	0.88	1.00	0.97	1.00	0.97	1.00	1.00	0.88	1.00	1.00	0.97	0.47	0.72	0.97	0.88	0.88	

Ideal is for PUF-generated bitstrings to have Entropy of 1 across bitstrings **and** chips

PUF Statistical Metrics for Randomness

The NIST Test Suite has 15 tests, several of which are described as follows:

- Frequency Test:

  Counts the number of '1' in a bitstring and assesses the closeness of the fraction of '1's to 0.5 (failing frequency usually means failure of most other tests)

- Block Frequency Test:

  Same except bitstring is partitioned into *M* blocks. Ensures bitstring is 'locally' random

- Fourier Transform Test:

  Analyzes the peak heights in the frequency spectrum of the bitstring, and tests if there are *periodic* features, i.e., repeating patterns close to each other

- Linear Complexity Test:

  Analyzes the bitstring to determine the length of the smallest set of LFSRs needed to reproduce the sequence

NIST Test Suite for Randomness

NIST ’finalAnalysisReport’ using HELP ASIC

50 chips

64,948 bits/chip

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-value	P/F	Proportion	P/F	Statistical test
2	4	5	6	7	5	5	5	5	6	0.956		50/50		Frequency
5	6	8	7	3	7	6	2	4	2	0.494		49/50		Block Frequency
4	2	5	6	5	4	8	7	4	5	0.817		50/50		CumulativeSums
4	1	6	7	8	4	3	4	7	6	0.494		50/50		CumulativeSums
12	3	10	7	2	2	4	5	2	3	0.007		47/50		Runs
5	6	5	6	5	6	4	7	5	1	0.851		49/50		LongestRun
9	8	3	4	4	8	4	3	2	5	0.290		50/50		Rank
8	3	4	5	6	4	5	5	7	3	0.851		50/50		FFT
6	1	5	5	8	2	6	6	6	5	0.575		50/50		NonOverlapping Template
...	...	...	...	...	...	...	...	...	...	...		...	*	...
2	6	5	7	5	4	6	4	6	5	0.936		50/50		ApproximateEntropy
5	6	5	7	6	3	7	4	6	1	0.699		49/50		Serial
7	6	7	2	2	9	7	4	4	2	0.237		50/50		Serial

The minimum pass rate for each statistical test with the exception of the random excursion (variant) test is approximately = 47 for a sample size = 50 binary sequences

Weak PUF vs Strong PUF

The distinction is rooted in the security properties of their challenge-response pairs

One definition of a **Strong PUF**:

Even after giving a adversary access to the PUF instance for a *prolonged period of time*, it is still possible to come up with a challenge that with high probability, the adversary **does not know the response**

This implies that

- The PUF has a **very large challenge space**, otherwise the adversary can simply query the PUF with all challenges to learn its complete CRP behavior
- It is **infeasible to build an accurate model** of the PUF using only a subset of CRPs to 'train' the model, as a means of learning its complete CRP behavior

PUFs which do not meet these requirements are called **Weak PUFs**

In the limit, some PUFs have only a single challenge and are called physically obfuscated key or POK

We discussed the SRAM PUF earlier that has only one challenge

PUF Usage Scenarios

- **Identification**

The PUF can be used to generate a 'serial number' to identify and/or track parts through manufacturing (the original proposed use by Keith Loftstrom in 1999!)

For manufacturing, ***uniqueness*** is the most important metric

A ***weak PUF*** is sufficient for this type of *low security* application

Reliability is not a concern as long as

- Bit flip errors are infrequent, i.e., HD_{intra} is relatively small, otherwise the probability of 'aliasing' gets unacceptably large
- It is possible to use a 'fuzzy match' criteria after the identifier is generated

- **Authentication**

The PUF is used to securely identify the chip in which it is embedded to an authority through corroborative evidence

As we will see when we discuss authentication scenarios, a ***strong PUF*** is best because the PUF inputs and outputs are **exposed** to the adversary

PUF Usage Scenarios

All three statistical metrics, i.e., uniqueness, randomness and reliability, are important for authentication

Some simple schemes relax the reliability metric as we will see

- **Encryption**

The PUF is used to generate a secret key, e.g., for symmetric encryption algorithms

In typical encryption applications, the key is not revealed outside the chip and therefore, a ***weak PUF*** can be used (although a strong PUF is better here too)

The ***inaccessability*** of the PUF responses makes **model-building** impossible

However, recent work shows that power analysis attacks can be used to enable model-building, which argues in favor of using strong PUFs for encryption too

Unfortunately, in contrast to authentication schemes, **tolerance to bit flip errors is 0**

Even a difference of 1 bit in a 256-bit key completely wrecks communication between parties because of the avalanche effect

PUF Implementations

There are MANY PUF implementations that have been proposed

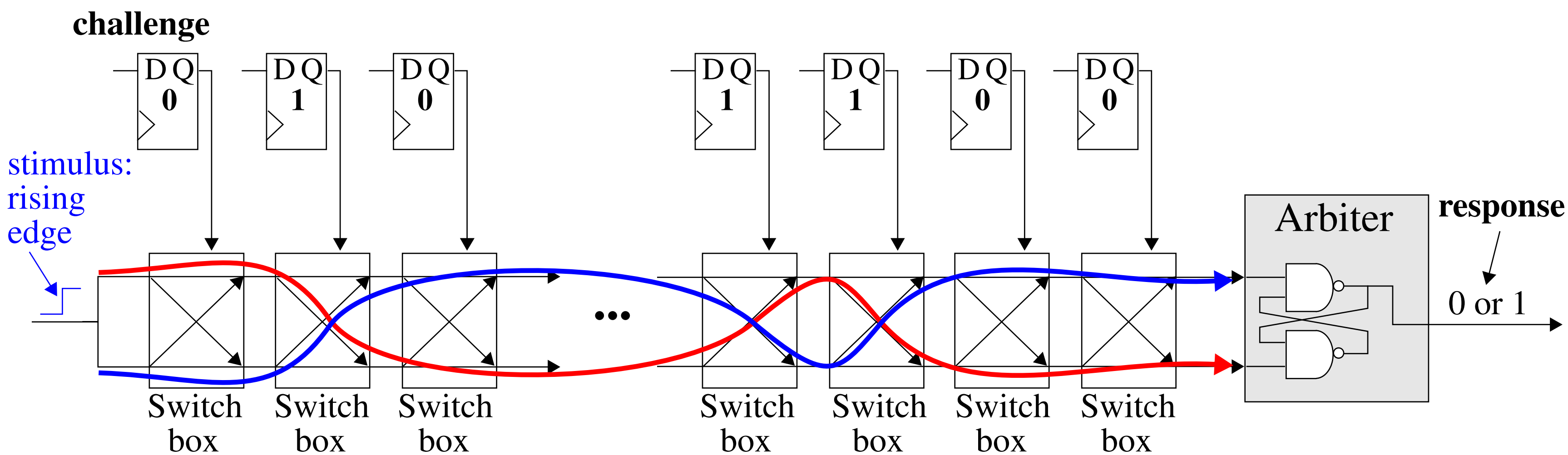
A rough characterization is as follows:

- *Delay-based PUFs*:
  - Delays along 'matched' paths (Arbiter)
  - Ring Oscillator frequencies
  - Glitches produced along paths within a functional unit
  - Delays along glitch-free paths within a functional unit (HELP)

- *Bi-stable PUFs*:
  - SRAM
  - Butterfly, Buskeepers
  - FFs and Latches

- *Mixed-Signal PUFs*: (These require a specialized analog-to-digital converter: ADC)
  - Transistor threshold voltage/transconductance
  - Dynamic/leakage current
  - Resistance/Capacitance

Arbiter PUF



A specialized structure implements **two paths**, each of which can be individually configured using a set of *challenge bits*

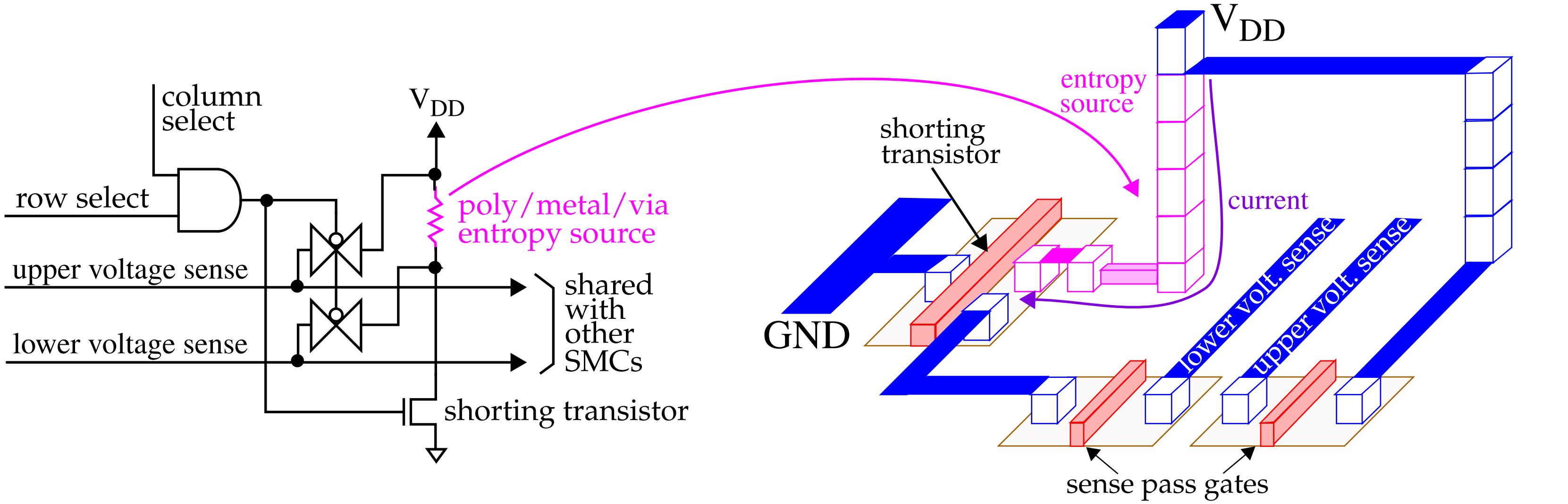
Each of the challenge bits controls a ‘Switch box’ in **pass mode** or **switch mode**

The faster path *controls the value stored* in the **Arbiter**

The arbiter PUF has an 2^n input challenges but the total amount of Entropy is relatively small with 128 switch-boxes, and therefore it is subject to model-building

Metal Resistance PUF

The metal PUF measures voltage drops across polysilicon wires, metal wires and vias as the source of entropy



Stimulus-Measure-Circuit (SMC)

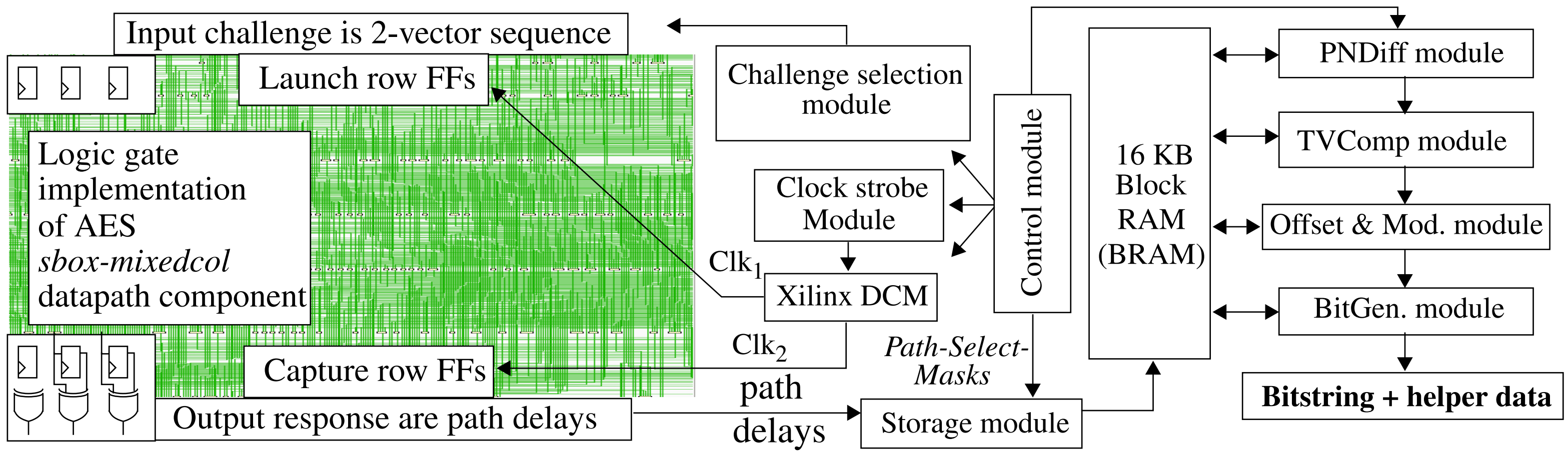
An SMC cell from a larger array is selected using *column* and *row* select signals

Once selected, a Stimulus-Measure-Circuit (SMC) enables a *shorting transistor* (stimulus) which creates a voltage drop across the poly-metal-via stack

Two ’pass gates’ are also enabled that allow voltages to be sensed and measured

Hardware Embedded Delay PUF (HELP)

HELP measures path delays in an on-chip functional unit, e.g., AES, and leverages random **within-die** variations in **propagation delay** as a source of entropy



HELP can be described entirely in an HDL, and therefore can be implemented on FPGAs

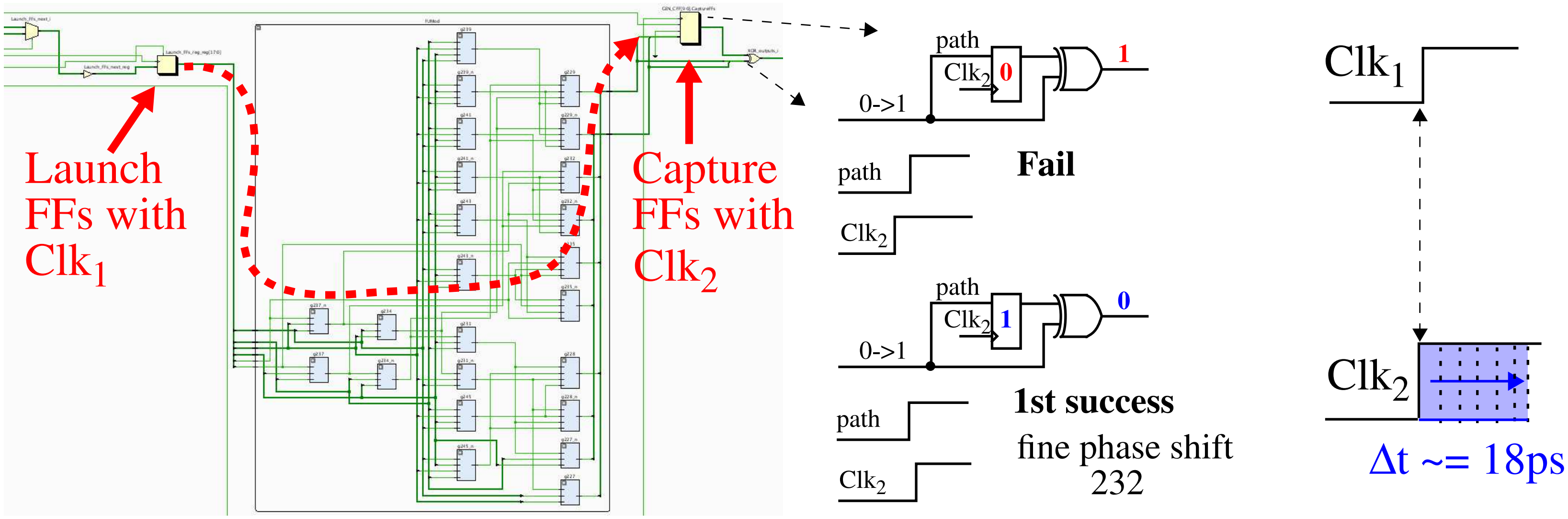
The functional unit (entropy source) is implemented using a specialized logic style that is **hazard-free**

This ensures paths remain *stable*, and can be timed accurately, as TV conditions vary

HELP is a STRONG PUF and is capable of generating a large # of random bitstrings

Hardware Embedded Delay PUF (HELP)

HELP uses a *launch-capture* timing mechanism to obtain high-resolution path delay values for combinational logic paths



Path delays can be measured using a **clock strobing** method

Or using an alternative *flash ADC* method that also works well

The *fine phase shift* feature within modern *digital clock managers* (DCMs) can be used to incrementally tune a capture clock, Clk_2, in a series of launch-capture tests

The integer-based *fine phase shift* value is used as the digitized path delay

Authentication Overview

Authentication refers to the process of '*verifying the identity of the communicating principals to one another*'

Authentication is typically carried out between
- A **prover** *A*, e.g., a hardware token such as a smart card, and
- A **verifier** *B*, e.g., a secure server operated by your bank

The verifier *B* either
- Confirms or *accepts* the prover's identity as authentic or
- Terminates without acceptance, i.e., *rejects*

Authentication protocols can be:
- **Unilateral**, i.e., from prover to verifier, or it may be **mutual**
- **Privacy preserving** to prevent malicious adversaries from tracking instances of authentications between the prover and verifier over time
- **Symmetric** in nature, requiring the use of a shared secret
- **Asymmetric** with the prover and verifier maintaining their own private secrets

PUF-Based Authentication

With the Internet-of-things (IoT), there are a growing number of applications in which the hardware token is **resource-constrained**

Therefore, novel authentication techniques are required that are *low in cost*, *energy* and *area overhead*

PUFs are attractive for authentication in **resource-constrained tokens** b/c:

- They *eliminate* (in many proposed authentication protocols) the need for NVM

- A special class of ***strong PUFs*** can also reduce area and energy overheads by reducing the number and type of hardware-instantiated cryptographic primitives

- The application controls the precise generation time of the secret bitstring

- They are *tamper-evident*, i.e., the entropy source of the PUF is sensitive to invasive probing attacks

Basic Protocol: Strong PUF with Unprotected Interface

The simplest mechanisms called ***challenge-response entity authentication*** exchange cleartext bitstrings directly, i.e., no cryptographic primitives are used

A PUF whose inputs and outputs can be accessed directly is said to have ***unprotected interfaces***

Prover (token ht_i with $\mathrm{ID_i}$)		**Verifier (server)**	
$r_j = PUF(c_j)$	⟷	(c_j, r_j) with $j \in [1 \ldots n]$ and $c_j \leftarrow TRNG()$	**Enrollment**
		$(c_j, r_j) \rightarrow \mathrm{DB}[\mathrm{ID}_i]$	
		(Server gens. challenges c_j and stores CRPs in $\mathrm{DB[ID_i]}$)	
	ID_i →	$\mathrm{DB}[ID_i] \rightarrow (c_n, r_n)$ (Server selects c_n)	**Authentication**
$r'_n = PUF(c_n)$ (PUF generates response r'_n with errors)	← c_n	$n = n - 1$ (CRP is deleted from DB)	
	r'_n →	$HD_{\mathrm{intra}}(r_n, r'_n) \overset{?}{<} \varepsilon$ Accept if match has $\mathrm{HD_{intra}}$ less than noise margin ε	

Basic Protocol: Strong PUF with Unprotected Interface

Benefits:

It is simple to implement and is very lightweight for the token

The **inability** of the PUF to precisely reproduce the response r_i makes it necessary to implement a *error-tolerant matching scheme* with $HD_{intra} > 0$

Drawbacks:

Large values of HD_{intra} increase the chance of impersonation, and act to reduce the strength of the authentication scheme

A large number of *CRPs* must be recorded during enrollment

This increases the storage requirements for the verifier, since the *worst-case usage scenario* must be accommodated

Or requires periodic *re-enrollment* at the secure facility

Basic Protocol: Strong PUF with Unprotected Interface

Drawbacks:

The protocol lacks resistance to *denial of service* attacks, whereby adversaries purposely deplete the server database

It lacks mutual authentication

It is susceptible to model-building attacks, and therefore is secure only when a *truely strong PUF* is used

A growing list of proposed protocols address these short-coming by incorporating **cryptographic primitives** on the prover and verifier side

The inclusion of cryptographic primitives enable significant improvements to the security properties of the protocols

And additionally enable *mutual authentication* and more efficient methods to *preserve privacy*

HELP Authentication Protocol Overview

VHDL description of Entropy source

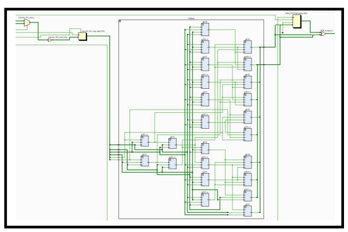
```
entity sbox_mixedcol is
  port (
    clk_in1: in std_logic;
    clk_in2: in std_logic;
    FCLK_CLK0: in std_logic;
     . . .
```

Glitch-free std. cell library

Cadence synthesis

Hazard-free conversion

Netlist



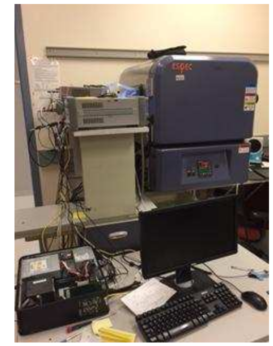
Automatic Test Pattern Generation

Challenges

001010100101...
110001010001...
001001010001...

Characterization

Subset of 30 chips



	PNR_0	•••	PNR_n	PNF_0	•••	PNF_m
C_1		•••			•••	
C_2						
C_3						
⋮						
C_{30}						

25^oC, 1.00V

•••

	PNR_0	•••	PNR_n	PNF_0	•••	PNF_m
C_1		•••			•••	
C_2						
C_3						
⋮						
C_{30}						

100^oC, 1.05V

Analysis of TVN and WID, challenge set selection

Enrollment of all chips at 25^oC, 1.00V

	PNR_0	PNR_1	PNR_2		PNR_x	PNF_0	PNF_1	PNF_2		PNF_x
C_1	380.1	294.8	366.9	•••	288.0	364.0	328.0	328.0	•••	276.2
C_2	366.6	282.8	352.7		278.6	374.3	334.6	337.1		286.1
C_3	366.3	288.4	355.7		280.8	372.7	336.4	338.0		282.3
	⋮									
C_n	387.5	301.2	373.5		292.3	362.9	325.18	323.7		272.1

Secure Server Database

HELP Authentication Protocol

Prover (token ht_i with ID_i)		Verifier (server)	
$\{PN_j\} = PUF(\{c_k\})$	← $\{c_k\}$	$\{c_k\} \leftarrow$ Server	**ID Phase** (Enrollment)
	$\{PN_j\}$ →	$ID_i \leftarrow$ ServerGenID()	
		$DB[ID_i] \leftarrow (\{PN_j\})$	
$\{PN_y\} = PUF(\{c_x\})$	← $\{c_x\}$	SelectATPG$(ID_i) \rightarrow \{c_x\}$	**Authen Phase** (Enrollment)
	$\{PN_y\}$ →	$DB[ID_i] \leftarrow (\{c_x, PN_y\})$	

Authentication

ID Phase

Prover		Verifier
$n_1 \leftarrow$ TRNG()	← n_2	$n_2 \leftarrow$ TRNG()
$m \leftarrow n_1 \oplus n_2$	n_1 →	$\{c_k\} \leftarrow$ Server
$(\text{Mod}, S, \mu_{ref}, \text{Rng}_{ref}, \text{Mar.}) \leftarrow \text{SelParam}(m)$	← $\{c_k\}, \{O_k\}$	$\{O_k\} \leftarrow$ Server
$\{mPNDco'_j\} \leftarrow AP(PUF(\{c_k\}), S, \mu_{ref}, \text{Rng}_{ref}, \text{Mod}, O_k)$		$m \leftarrow n_1 \oplus n_2$
$(bss', h') \leftarrow \text{SHBG}(\{mPNDco'_j\}, \text{Mar.})$	bss', h' →	$(\text{Mod}, S, \mu_{ref}, \text{Rng}_{ref}, \text{Mar.}) \leftarrow \text{SelParam}(m)$

For i in $DB[ID_i]$ **(Search for match)**

$\{mPNDco_j\}_i \leftarrow AP(\{PN_j\}_i, S, \mu_{ref}, \text{Rng}_{ref}, \text{Mod}, O_k)$

$(bss, bss'') \leftarrow \text{DHBG}(\{mPNDco_j\}_i, \text{Mar.}, bss', h')$

$bss'' \stackrel{?}{=} bss$

← ID_i

If match is found, proceed to *verifier authentication*

Secure Boot

Methods that guarantee that the system boots with an authorized FPGA bitstream and/or BootROM code establish the 'root of trust' in the system

The focus of our discussion will be on secure boot of FPGAs

In Xilinx FPGAs, the root of trust is the stored key

Keys can be stored in Battery Backed RAMs (**BBRAM**) or using **eFUSE**

The drawbacks of these on-chip digital storage mechanisms include

- BBRAM require a battery to be installed on the system board and therefore increase system cost
- The batteries for BBRAM also have a limited lifetime and therefore complicate system maintenance
- eFUSE is one-time-programmable (OTP) and therefore reduce flexibility in key management
- eFUSE keys can be read-out using, e.g., scanning electron microscopes (SEM)

Xilinx Secure Boot Process

The BBRAM or eFUSE keys are used as the root of trust in the Xilinx secure boot process

- In a secure facility, the Xilinx CAD tools can be used to encrypt the bitstream using a randomly generated or user-specified key

- The decryption key is loaded via JTAG at a secure facility into the eFUSE or BBRAM

- The in-field secure boot process first determines if the external bitstream includes an encrypted-bitstream indicator

  If so, the on-chip 256-bit AES engine decrypts the bitstream using cipher block chaining (CBC) mode of AES along with the eFUSE or BBRAM key

  CBC mode XORs the previous block ciphertext with the next block plaintext before encrypting the current block (decryption reverses this process)

  This forces different ciphertexts for replicated components in the plaintext

Xilinx Secure Boot Process

- Authentication is used to ensure data integrity of the bitstream using SHA-256 where a 256-bit *keyed MAC* (**HMAC**) is computed for the bitstream

  The HMAC is designed to prevent bit-flip attacks and other types of fault injection attacks

  Therefore, the HMAC authenticates the origin of the bitstream and detects any type of tamper

  The HMAC of the unencrypted bitstream is computed in a secure facility and embedded with the key in the bitstream, which is then encrypted by AES

  During in-field boot, a second HMAC is computed as the bitstream is decrypted and compared with the HMAC embedded in the decrypted bitstream

  If the comparison fails, the FPGA does not become active

The secure boot process provides confidentiality, data integrity and authentication

  It detects tamper and attempts to program FPGA with a non-authentic bitstream

Xilinx SoC Secure Boot Process

Xilinx FPGA SoCs, e.g., Zynq series, use an asymmetric (public-private) authentication (digital signature) scheme in the secure boot process

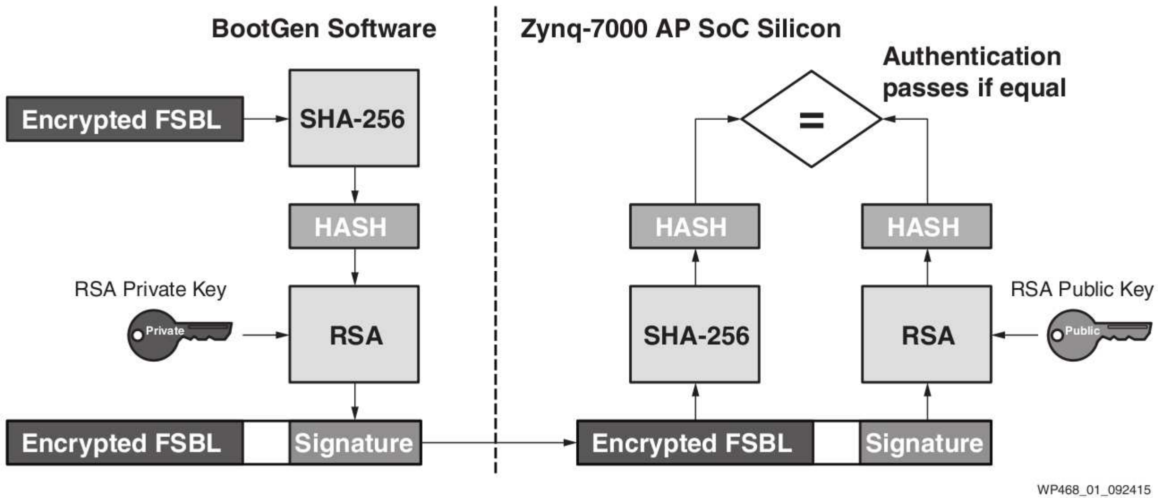


Figure 1: **Asymmetric Authentication Process**

Leveraging Asymmetric Authenticationto Enhance Security-Critical Applications Using Zynq-7000 All Programmable SoCs,WP468 (v1.0) October 20, 2015

Here, we see **bootgen** computes a SHA-256 hash of the encrypted first stage boot loader (**FSBL**) and a *digital signature* is then computed using the RSA private key

Signature verification is carried out by the Zynq chip using the public key to recover the hash, which is compared with a locally computed hash of the encrypted FSBL

Xilinx SoC Secure Boot Process

The first stage boot loader (FSBL) is authenticated as shown BEFORE it is decrypted and executed by the PS-side

If authentication succeeds, the FSBL is decrypted by a PL-side AES engine using a key stored in the BBRAM or eFUSE

RSA-2048 signature verification algorithm resides in the PS-side BootROM, which is a mask-programmed, hardwired, immutable memory

- Neither the private or public keys are stored on the FPGA

- Instead, a 256-bit hash of the public key is programmed into the eFUSE array

The **FSBL** then becomes the **root of trust** in the boot process

- PS-side images and PL configurations can then be loaded by the FSBL

- The user must include decryption and authentication functions in the FSBL to ensure these subsequent components of the boot process are secure

Xilinx Secure Boot Process

Secure boot requires the boot process to begin with a root of trust, and then carry out authentication in each of the subsequent stages

As indicated above, Xilinx FPGA SoCs use public key cryptography, i.e., RSA, for authentication and attestation of FSBL and other configuration files

And a hardwired 256-bit AES engine and HMAC to securely decrypt and authenticate boot images on chip using a BBRAM or eFUSE embedded key

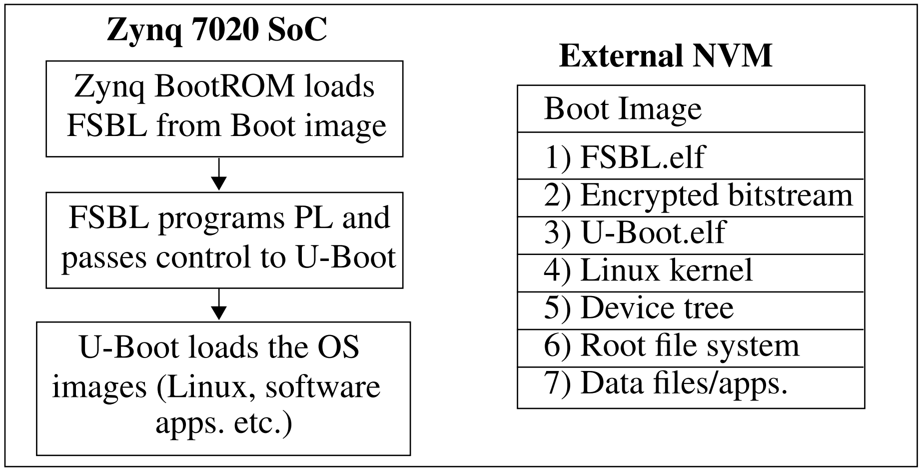
Although the Xilinx FPGA SoC root of trust begins with the RSA authenticated FSBL, which does not use an embedded key, decryption of the FSBL does

Moreover, the Xilinx non-SoC PL-side boots, as discussed earlier, use eFUSE and BBRAM for bitstream decryption

In either case, the **root of trust cannot be expanded** to include PS-side images and/or PL configuration data without keeping the embedded key confidential

Xilinx Boot Process

Let's examine the underlying steps of the Xilinx boot process and then look at an alternative self-authenticating PUF-based solution



The Xilinx BootROM loads the FSBL from an external NVM to DDR (DRAM)

The FSBL programs the PL side and then reads the second stage boot loader (U-Boot), which is copied to DDR, and passes control to U-Boot

U-Boot loads the OS images, which includes a bare-metal application, or the Linux OS, embedded software applications and data files

Bullet-Proof Boot for FPGAs (BulletProoF) Process

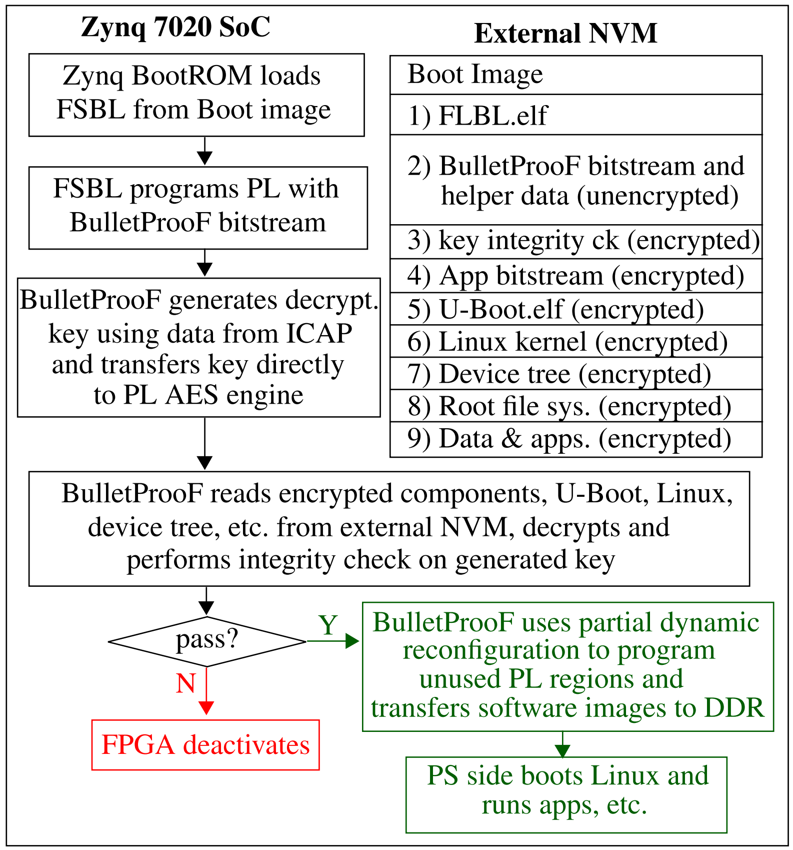
The BulletProoF boot process **does not** use any of the security features provided by Xilinx, i.e., it is self-contained and self-authenticaing

The first step is identical to the existing boot process

The PL component that is programmed into the PL side by the FSBL is the unencrypted BulletProoF bitstream

The FSBL then passes control to Bullet-ProoF and blocks

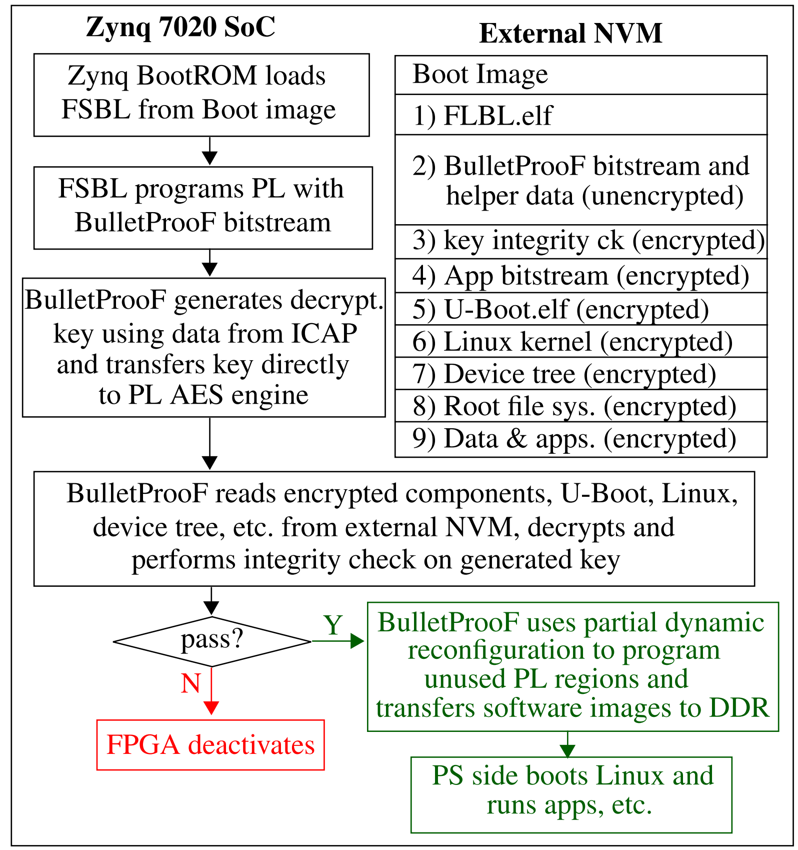
BulletProoF reads configuration data using ICAP and helper data from an NVM and carries out key regeneration



The key is transferred to an embedded PL-side AES engine

Bullet-Proof Boot for FPGAs (BulletProoF) Process

BulletProoF reads the encrypted second stage boot image components labeled as components 3 through 9 from external NVM and transfers them to the AES engine



An integrity check is performed at the beginning of the decryption process as a mechanism to determine if the proper key was regenerated

The first component decrypted is the key integrity check component (labeled 3)

This component can be an arbitrary string or a secure hash of, e.g., U-Boot.elf, that is encrypted during enrollment and stored in the external NVM

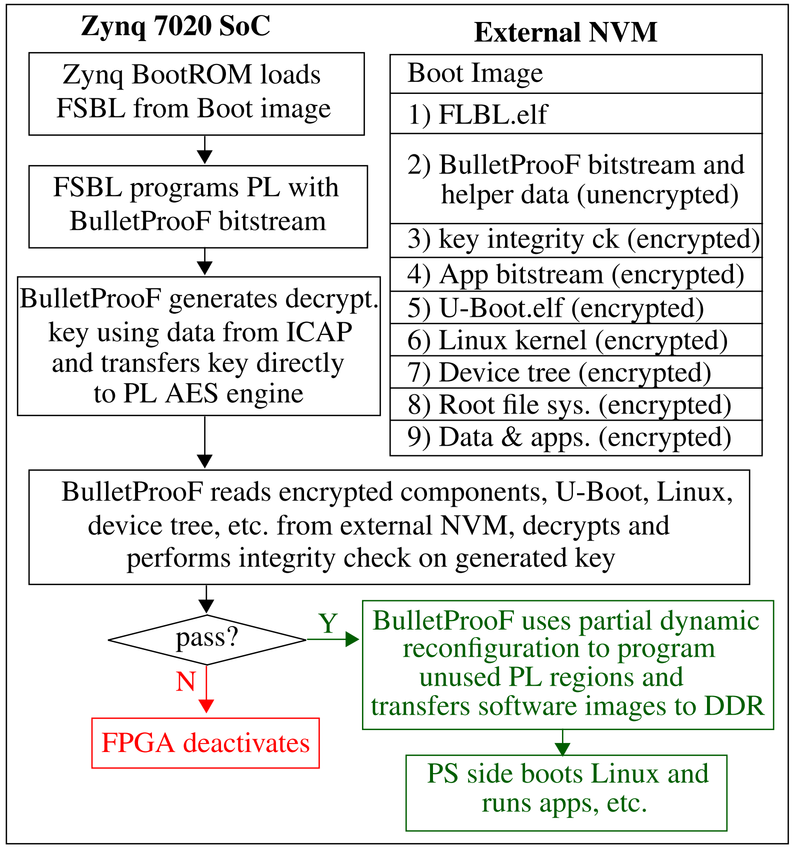
Bullet-Proof Boot for FPGAs (BulletProoF) Process

An unencrypted version of the key integrity check component is also stored as a constant in the BulletProoF bitstream

The integrity of the decryption key is checked by comparing the decrypted version with the BulletProoF version

If they match, then the integrity check passes and the boot process continues

Otherwise, the FPGA is deactivated and secure boot fails



If the integrity check passes, BulletProoF then decrypts components 4 through 9, starting with the application (App) bitstream

Bullet-Proof Boot for FPGAs (BulletProoF) Process

BulletProoF uses the HELP PUF to generate the decryption key as a mechanism to eliminate the vulnerabilities associated with on-chip key storage

Key generation using PUFs starts with an enrollment phase carried out in a secure environment

- The encryption key is generated using configuration data read from ICAP, which is then used to encrypt the 2nd stage boot images

- A special enrollment version of BulletProoF generates the key internally and transfers helper data off of the FPGA
  - Which is stored unencrypted in the external NVM

The internally generated key is then used to encrypt the other components of the NVM by configuring AES in encryption mode

The enrollment version performs encryption while the in-field version performs decryption, but the two versions are otherwise identical

Security Properties of BulletProof

The proposed system has the following security properties

- The enrollment and regeneration processes **never reveal the key** outside the FPGA, requiring the adversary to use physical, side-channel-based attacks to steal the key

- Any type of tamper with the unencrypted helper data by an adversary will only prevent the key from being regenerated and a subsequent failure of boot process

  Note that it is always possible to tamper with the contents stored in the external NVM, independent of whether it is encrypted or not

- The HELP PUF discussed earlier implements a helper data scheme that does not leak information about the key

- The HELP PUF to designed to self-authenticate itself, thereby detecting any type of tamper with unencrypted version of the BulletProoF bitstream

BulletProof Architecture

BulletProof derives challenges for the HELP PUF using the FPGA configuration data read directly from the ICAP interface

Since the FPGA is programmed with the unencrypted BulletProof bitstream, this represents a form of self-authentication

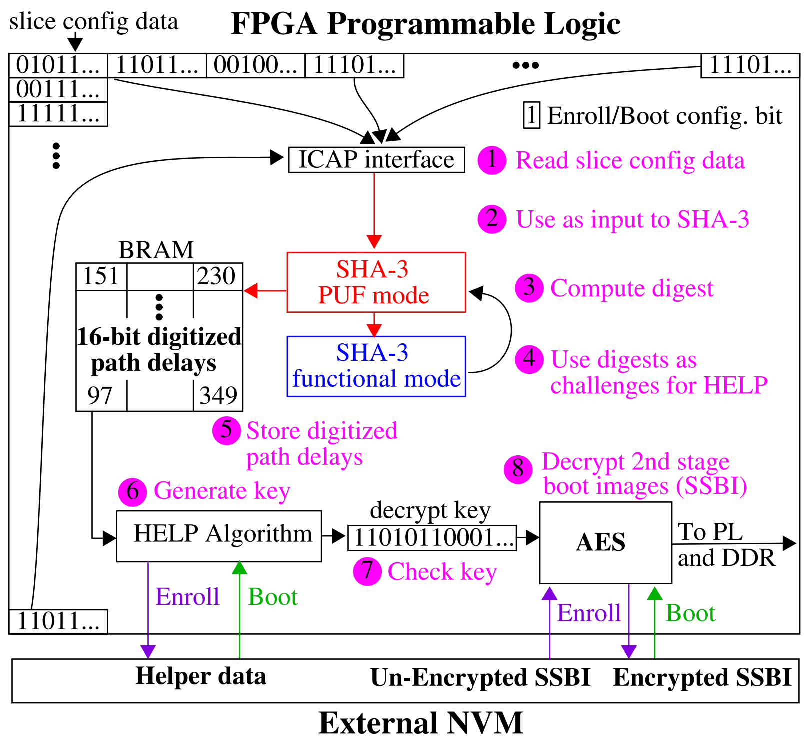
The source of entropy of the HELP PUF is an implementation of the SHA-3 algorithm

The bitstream configuration data is hashed using SHA-3 configured in Mode 1 (functional mode)

Periodically, the current state of the SHA-3 hash is used as a challenge to SHA-3 configured in Mode 2 (PUF mode) to generate timing data for key generation

BulletProof Architecture

The configuration data within the PL-side of the FPGA is shown overlaid on top of the BulletProof flow diagram

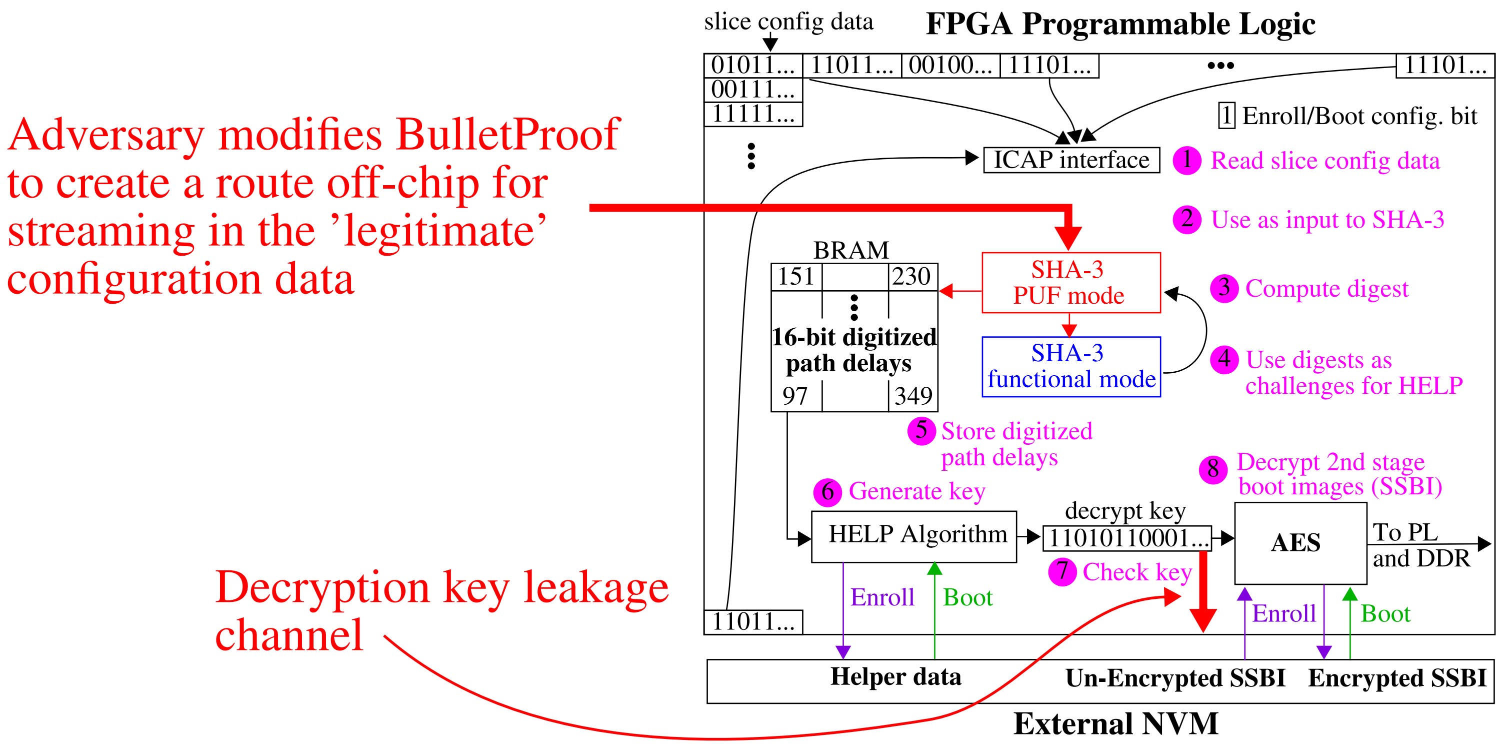


The SHA-3 blocks are shown as two separate blocks but are in fact one block

The BulletProof architecture is designed such that challenges are launched directly from the ICAP interface register to prevent a specific type of RE attack

BulletProof Architecture

The paths between the ICAP and SHA-3 are timed because of the following reverse engineering attack scenario

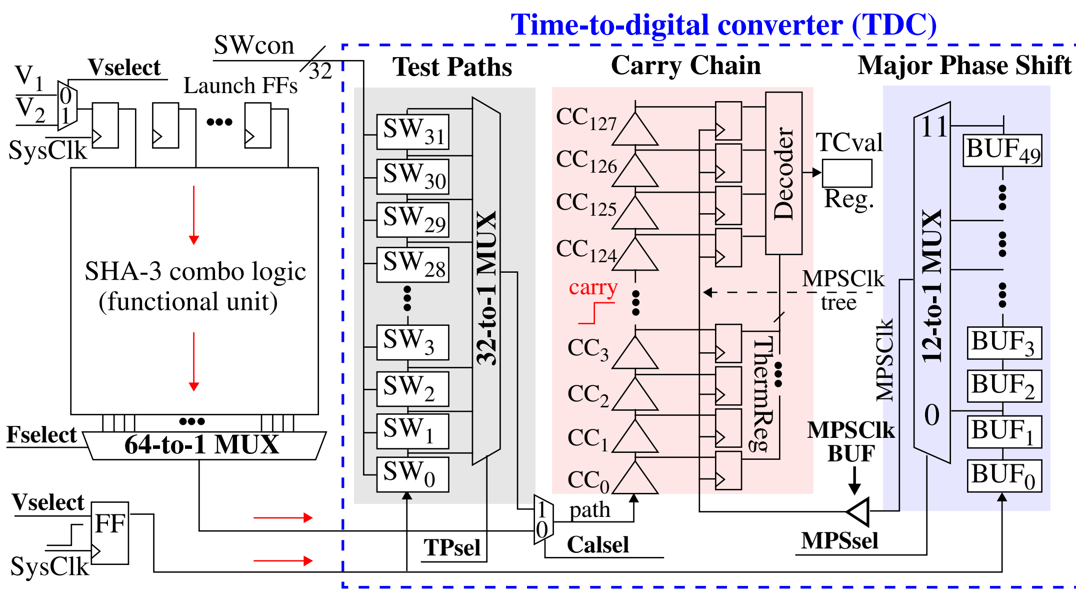


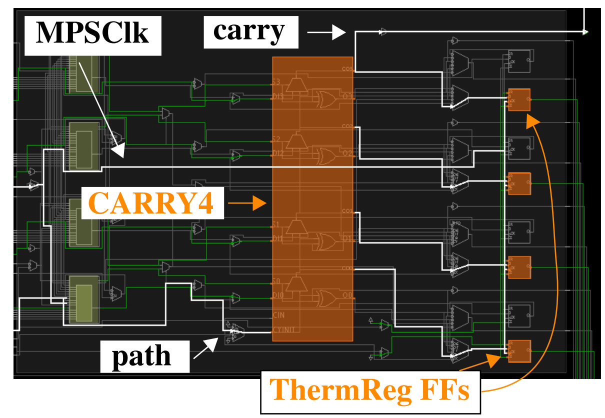
We must guarantee that the configuration data used as input to the SHA-3 originates from the ICAP interface

Otherwise, the adversary can create a route as shown and then change the on-chip version of BulletProof to leak the key off-chip

Time-to-Digital Converter Alternative to Xilinx MMCM

The original *clock strobing* method for timing paths can be replaced with a time-to-digital converter (TDC) that leverages high-speed carry chains on the FPGA





The TDC timing engine replaces the Xilinx MMCM, and when used with a ring oscillator as the clock source, prevents attacks that attempt to stop the clock