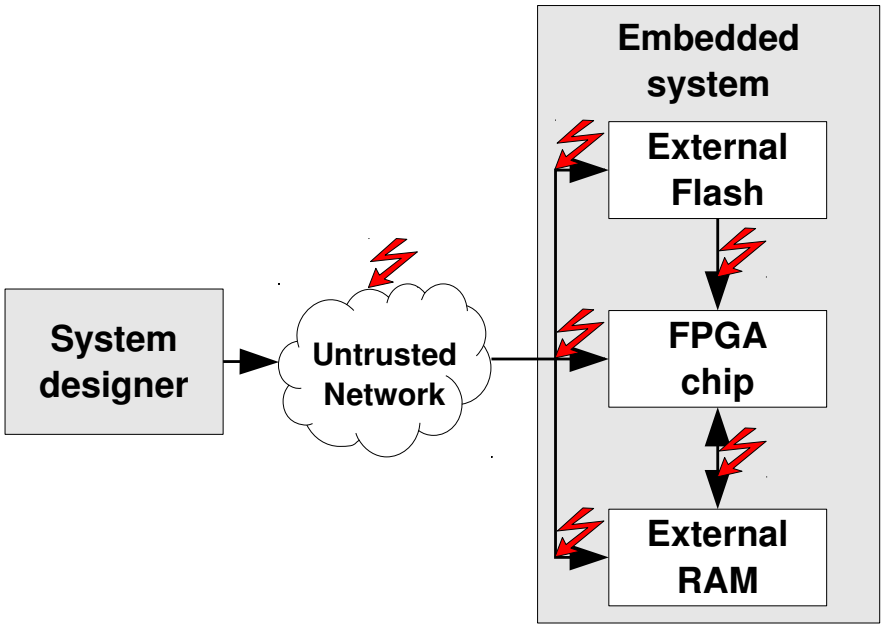
Embedded OS for FPGA platform: a Hardware-to-Software Security Overview

Florian Devic[1,2], Lionel Torres[1] and Benoit Badrignans[2]

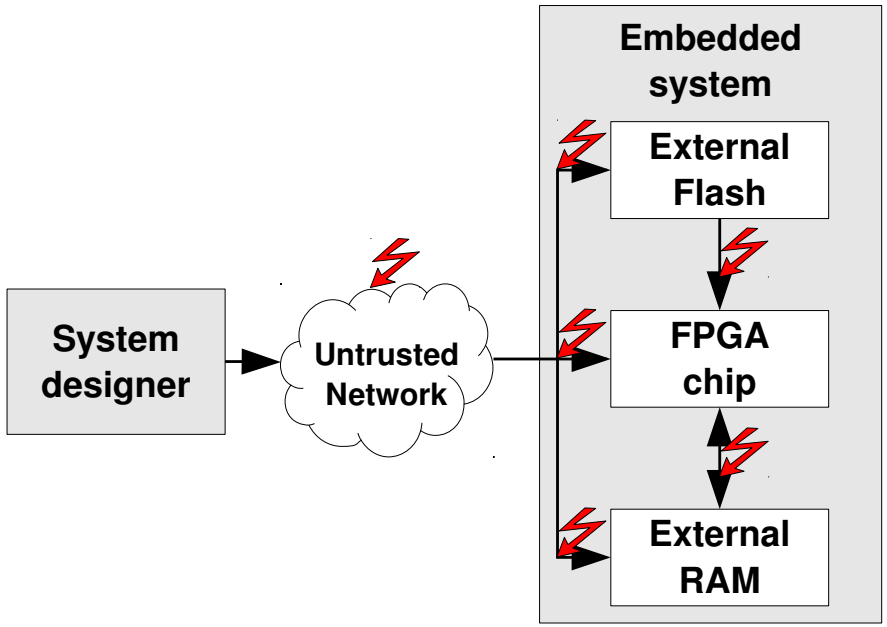
[1] LIRMM UMR -CNRS 5506, University of Montpellier 2, Montpellier, France
[2] SAS NETHEOS, Montpellier, France

June 16, 2011



Goals

- Secure the **bitstream**, the **boot** and finally the **execution** of an embedded OS (stored in an external memory) on FPGA.
- **Upgraded** the system through an insecure network.
- Prevents an attacker to execute his own potentially **malicious program** or to **replay** an old bitstream to downgrade the system.



Goals

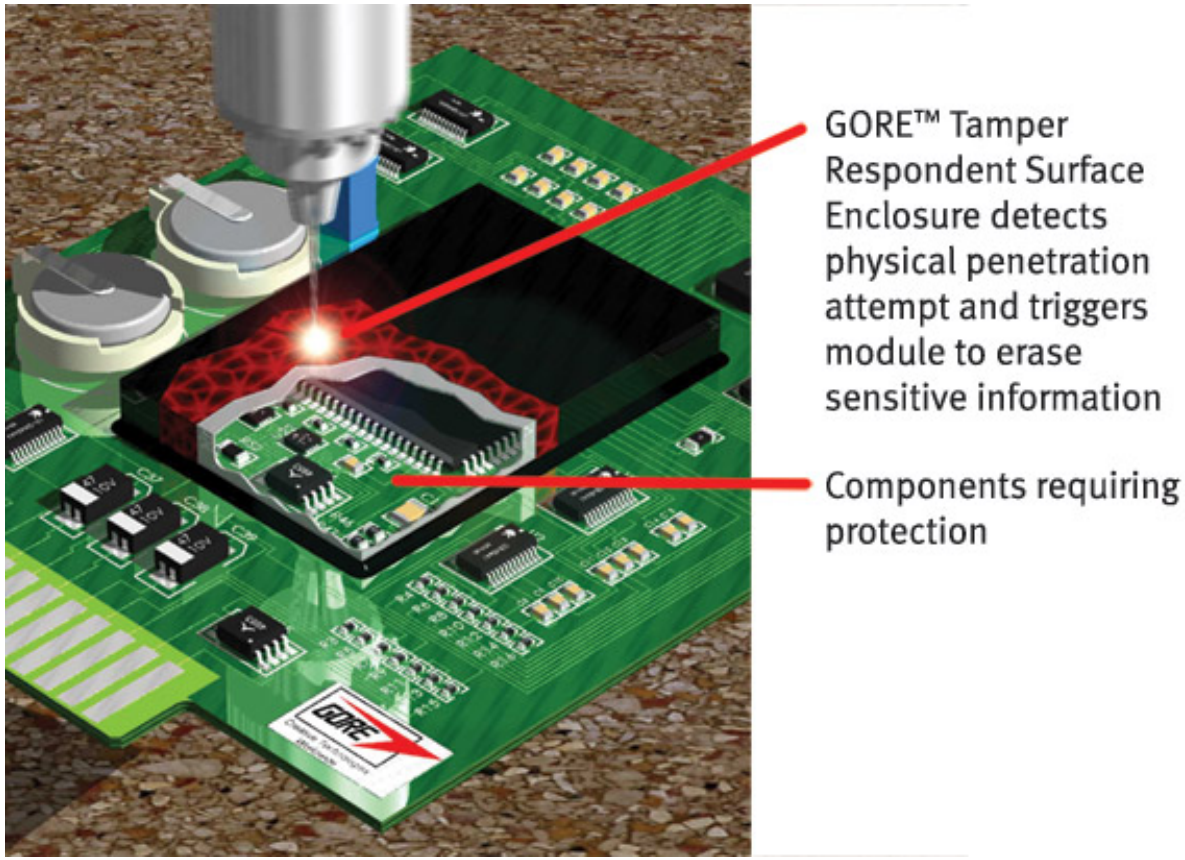
- Secure the **bitstream**, the **boot** and finally the **execution** of an embedded OS (stored in an external memory) on FPGA.
- **Upgraded** the system through an insecure network.
- Prevents an attacker to execute his own potentially **malicious program** or to **replay** an old bitstream to downgrade the system.

→ **Man in the middle, off-chip probing and injections**

Outline

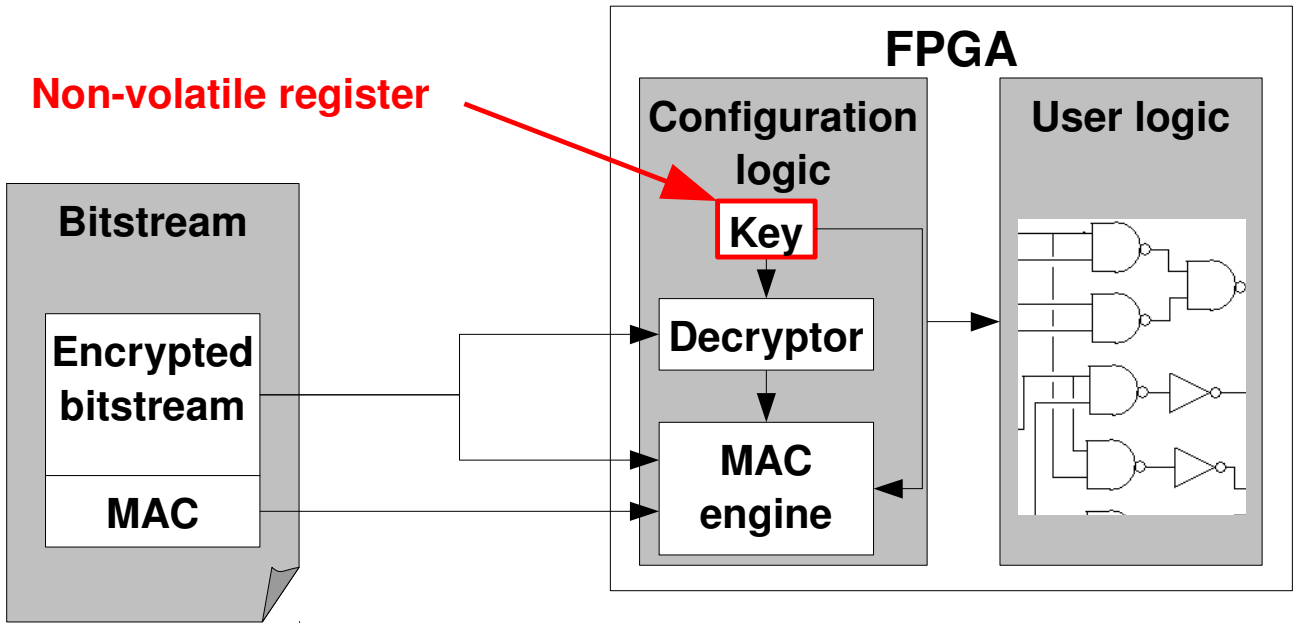
Outline

FPGA chip



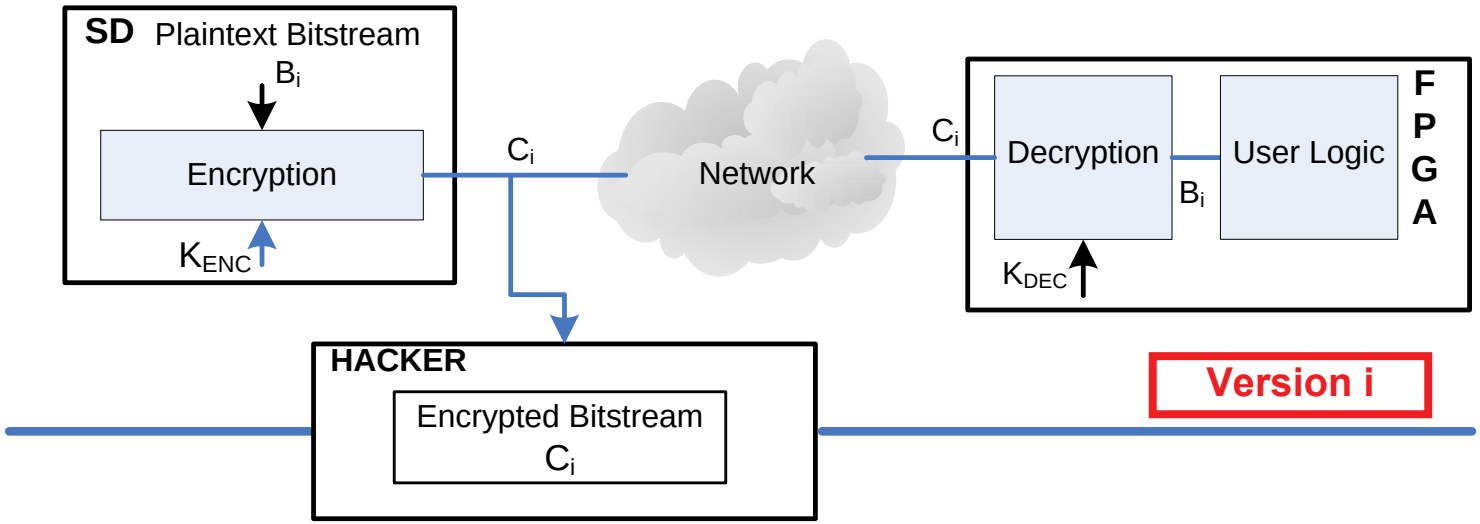
more information on http ://www.gore.com/en_xx/products/electronic/anti-tamper/tamper-surface-enclosure.html

Bitstream confidentiality and integrity

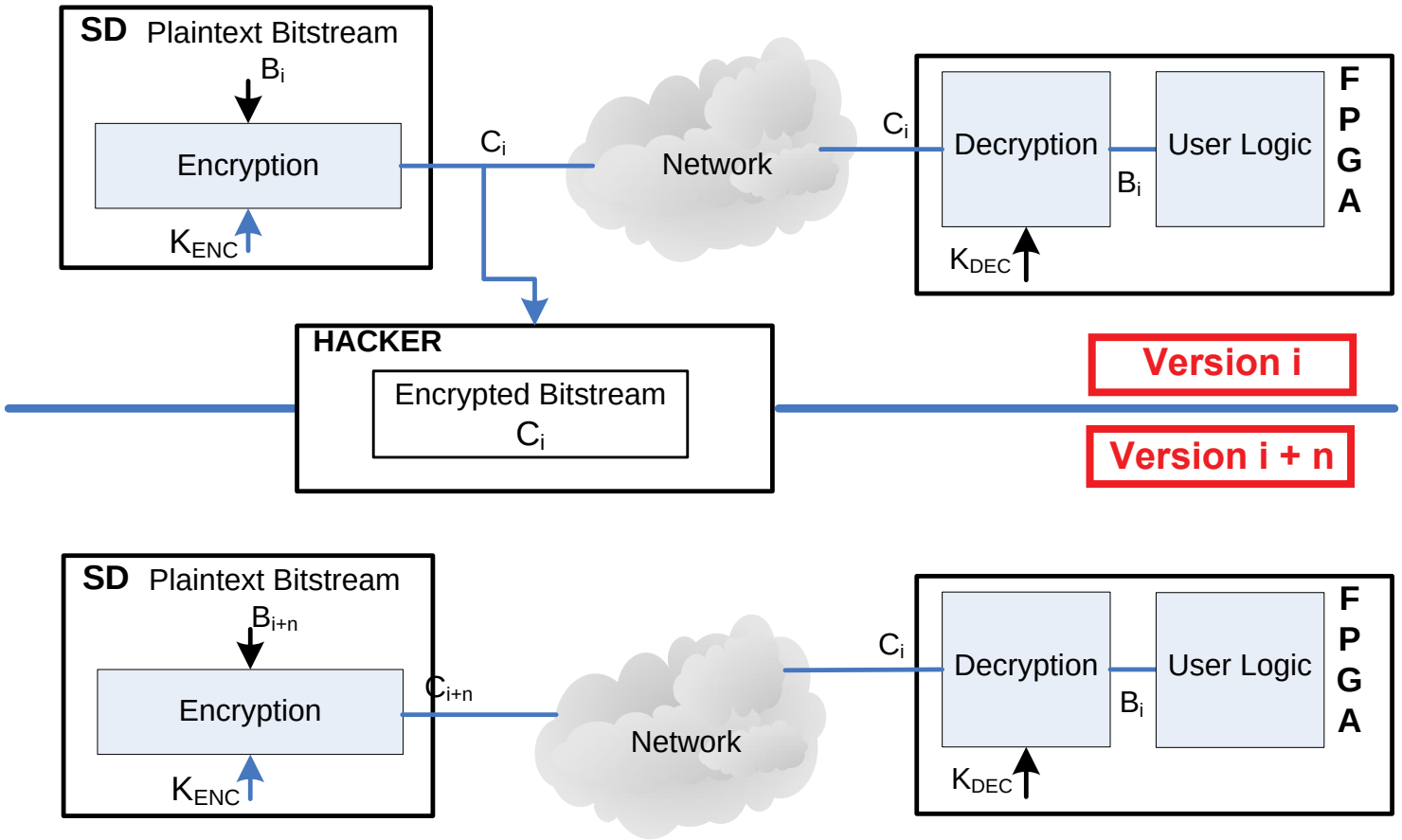


- Prevent cloning
- Prevent reverse engineering
- Prevent modifications

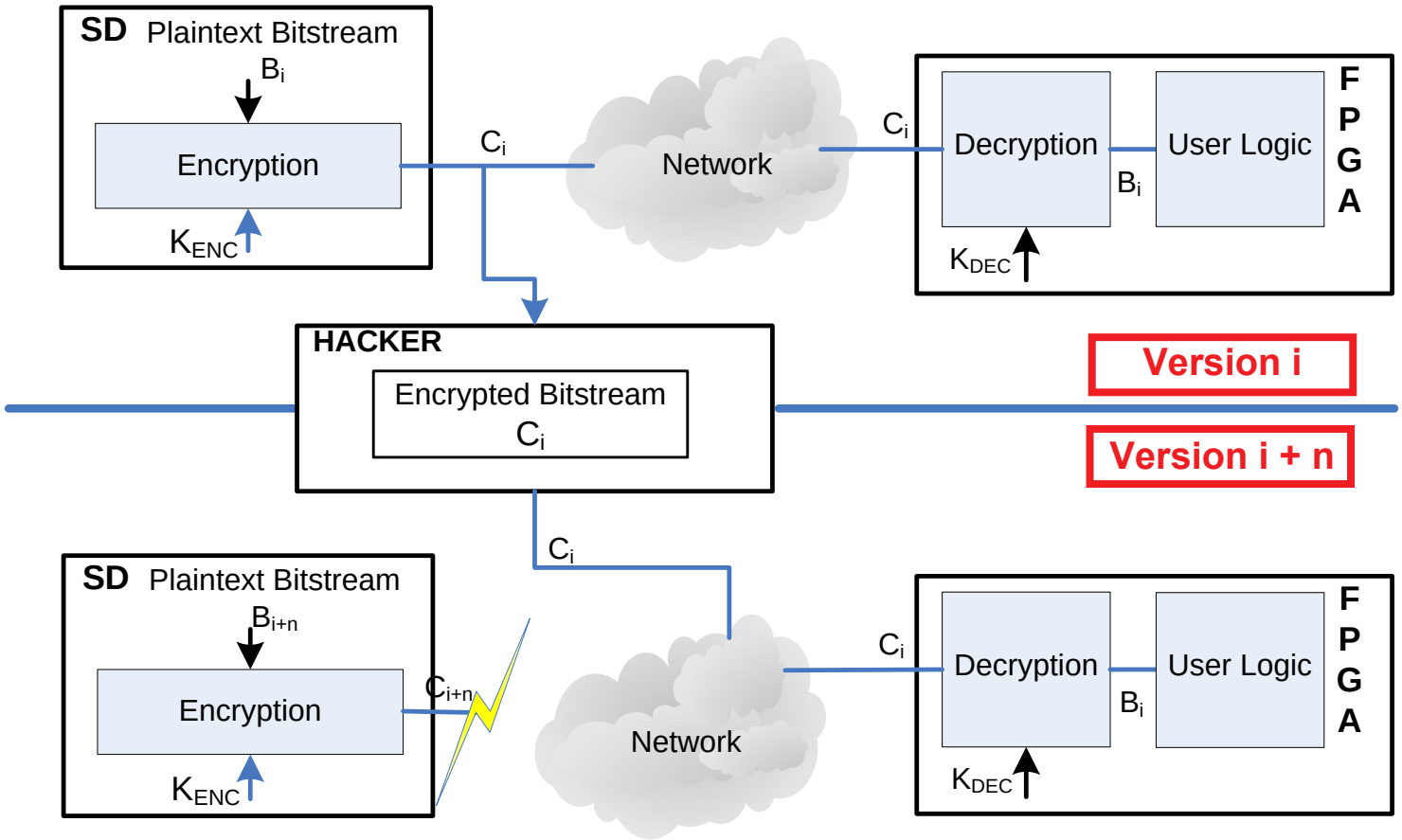
Replay attack



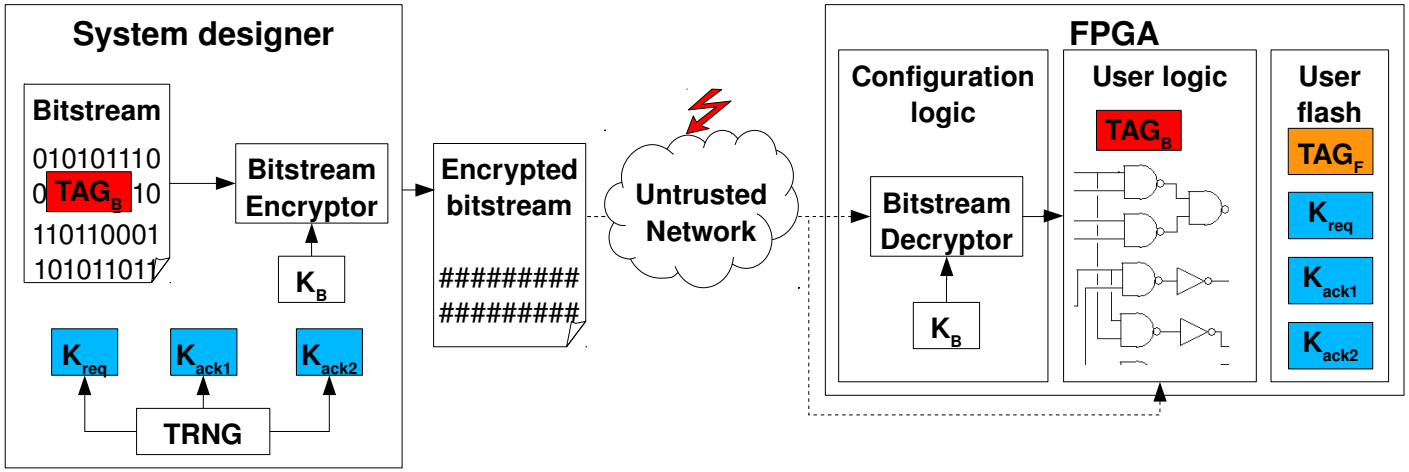
Replay attack



Replay attack



Principle with an embedded Flash



Goal : Lock the FPGA to a dedicated version

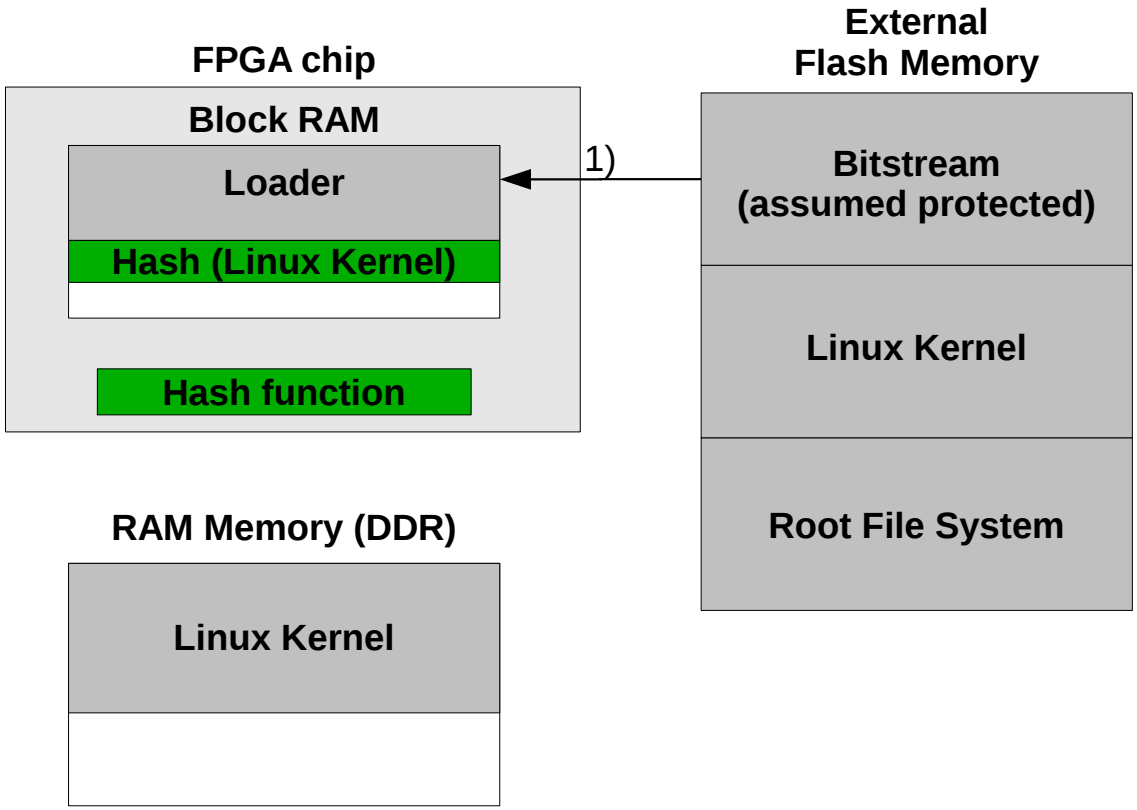
TAG_B and TAG_F are the current bitstream version

- K_{req} : for the Update command
- K_{ack1} : for the Update command acknowledgement
- K_{ack2} : for the new bitstream version startup acknowledgement

F. Devic, B. Bradrignans, and L. Torres, "Secure FPGA Configuration Architecture Preventing System Downgrade" in FPL'10

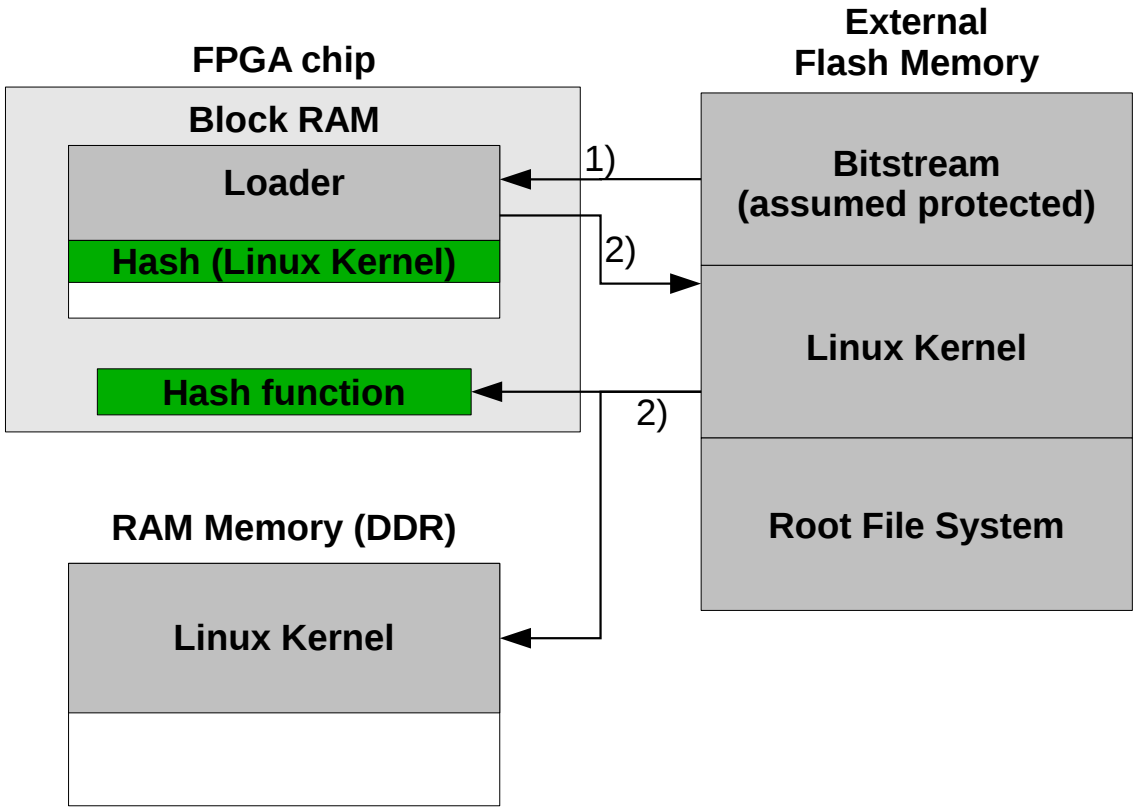
Outline

Integrity verification



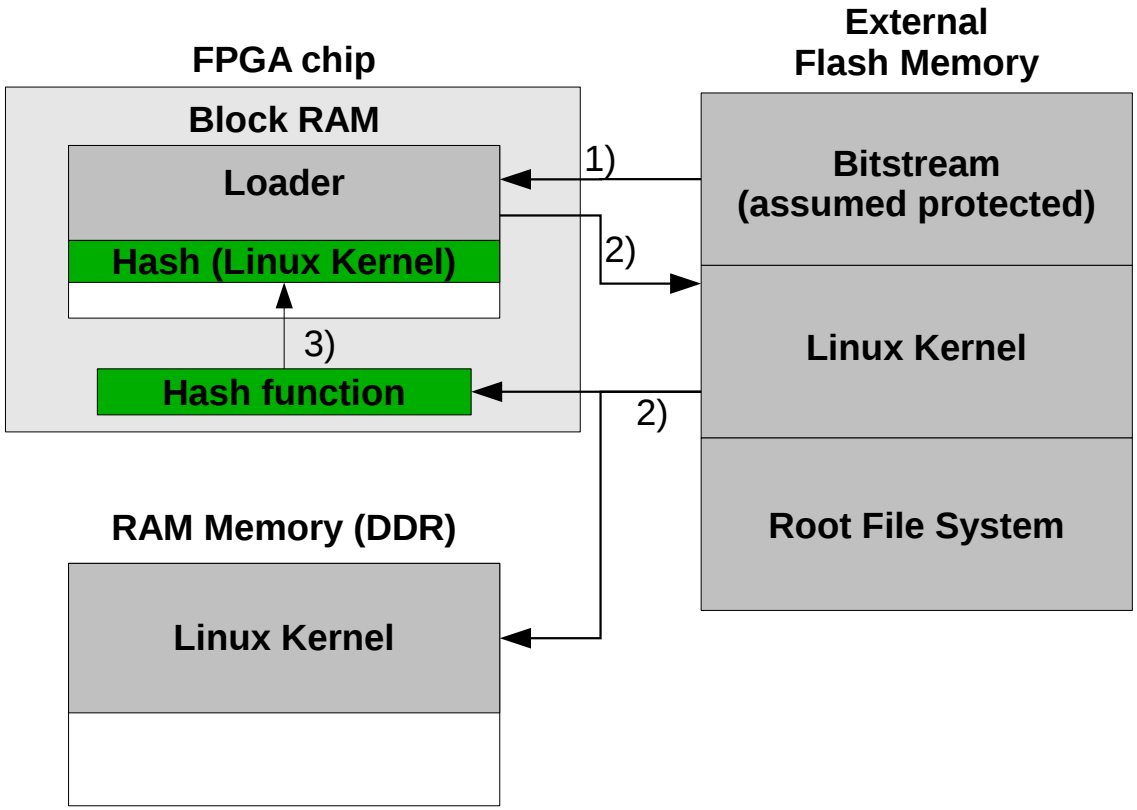
Boot steps :
1) The loader is stored in block RAM at power-up from bitstream
2) The loader copies Kernel from Flash to RAM and compute its hash
3) The loader verifies the Kernel integrity thanks to the hash
4) The loader branches to the Kernel and Linux boots

Integrity verification



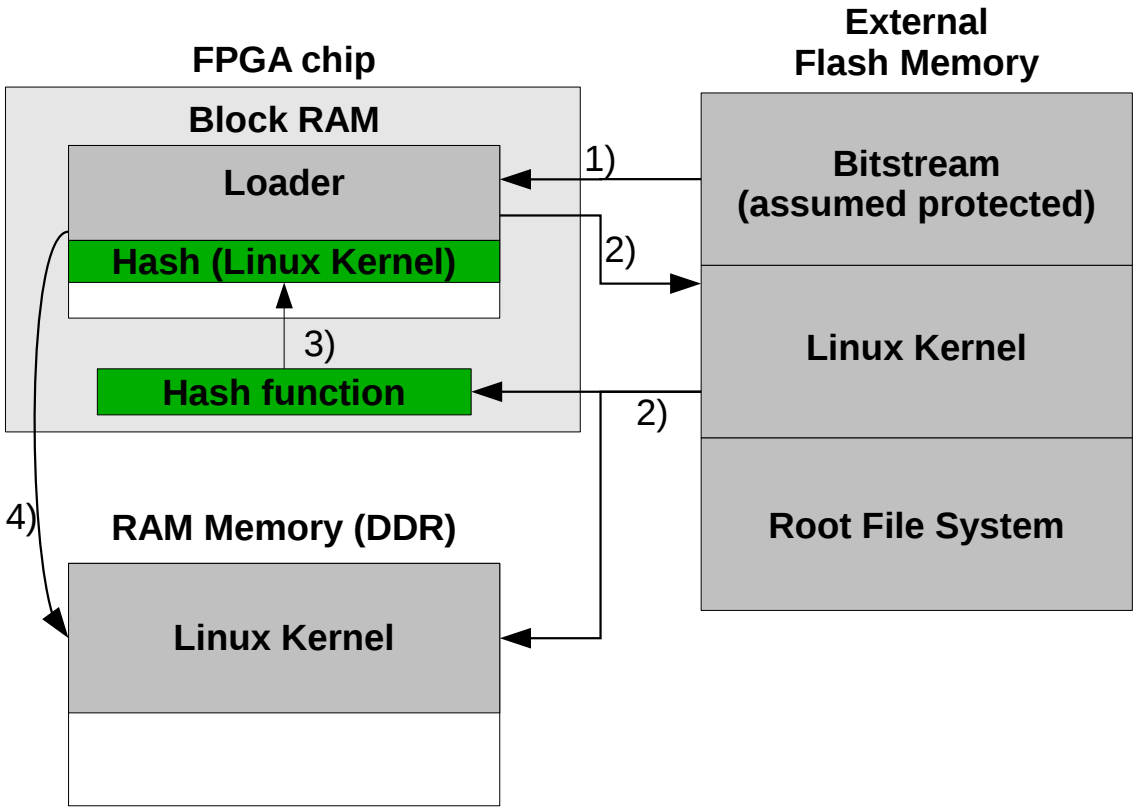
Boot steps :
1) The loader is stored in block RAM at power-up from bitstream
2) The loader copies Kernel from Flash to RAM and compute its hash
3) The loader verifies the Kernel integrity thanks to the hash
4) The loader branches to the Kernel and Linux boots

Integrity verification



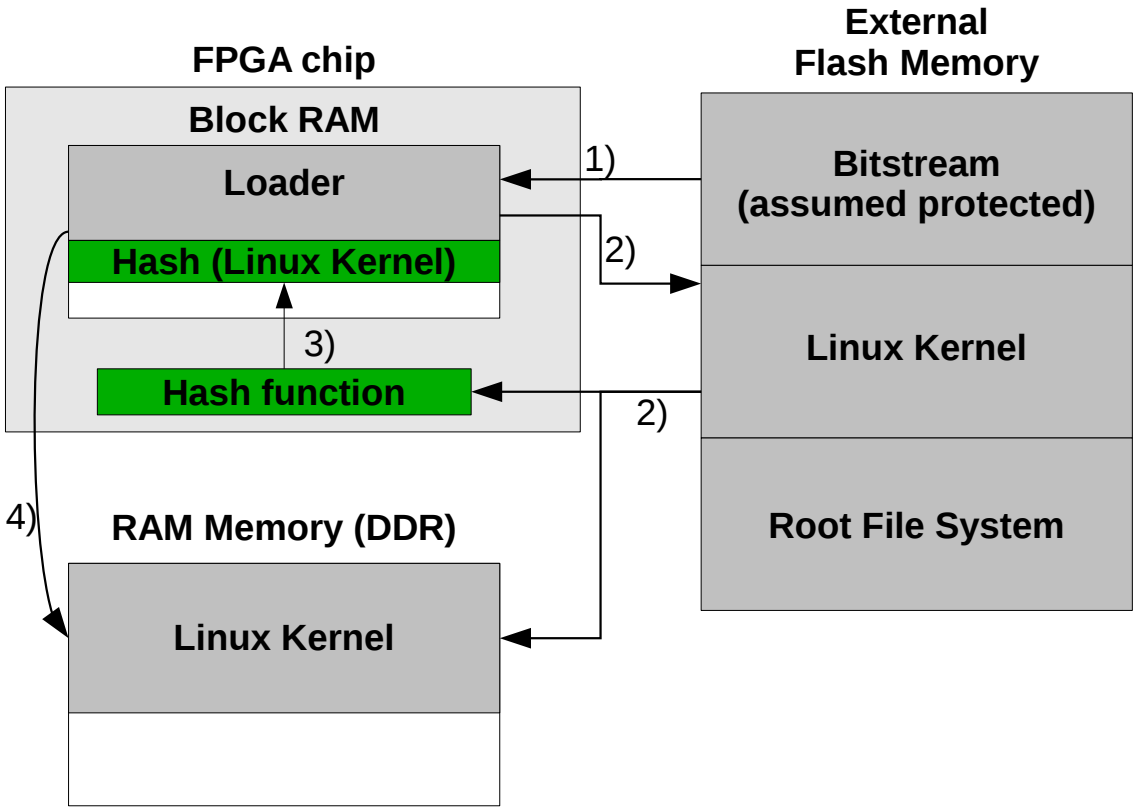
Boot steps :
1) The loader is stored in block RAM at power-up from bitstream
2) The loader copies Kernel from Flash to RAM and compute its hash
3) The loader verifies the Kernel integrity thanks to the hash
4) The loader branches to the Kernel and Linux boots

Integrity verification



Boot steps :
1) The loader is stored in block RAM at power-up from bitstream
2) The loader copies Kernel from Flash to RAM and compute its hash
3) The loader verifies the Kernel integrity thanks to the hash
4) The loader branches to the Kernel and Linux boots

Integrity verification



Boot steps :
1) The loader is stored in block RAM at power-up from bitstream
2) The loader copies Kernel from Flash to RAM and compute its hash
3) The loader verifies the Kernel integrity thanks to the hash
4) The loader branches to the Kernel and Linux boots

→ **Changing the kernel requires to change the bitstream**

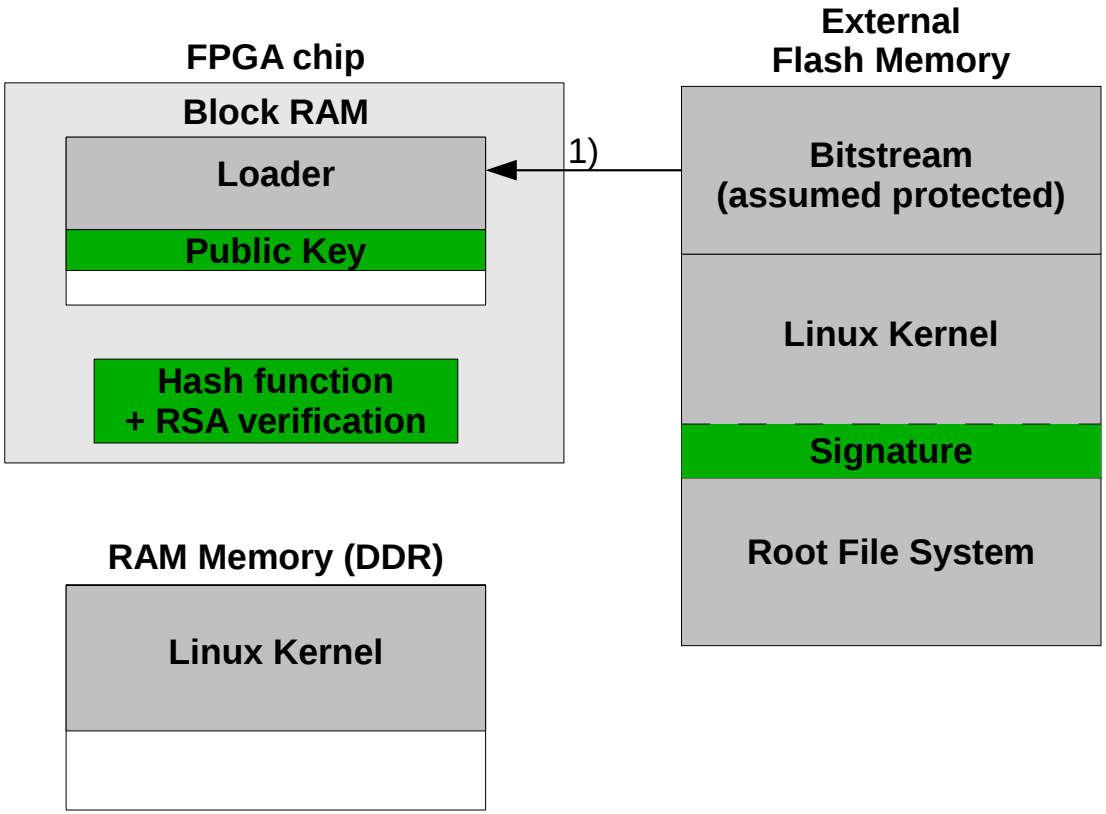
Performance overhead

Results :

Linux kernel = 2.8 MB
Virtex 6 : Processor frequency = 100Mhz
Hash algorithm = Sha-256

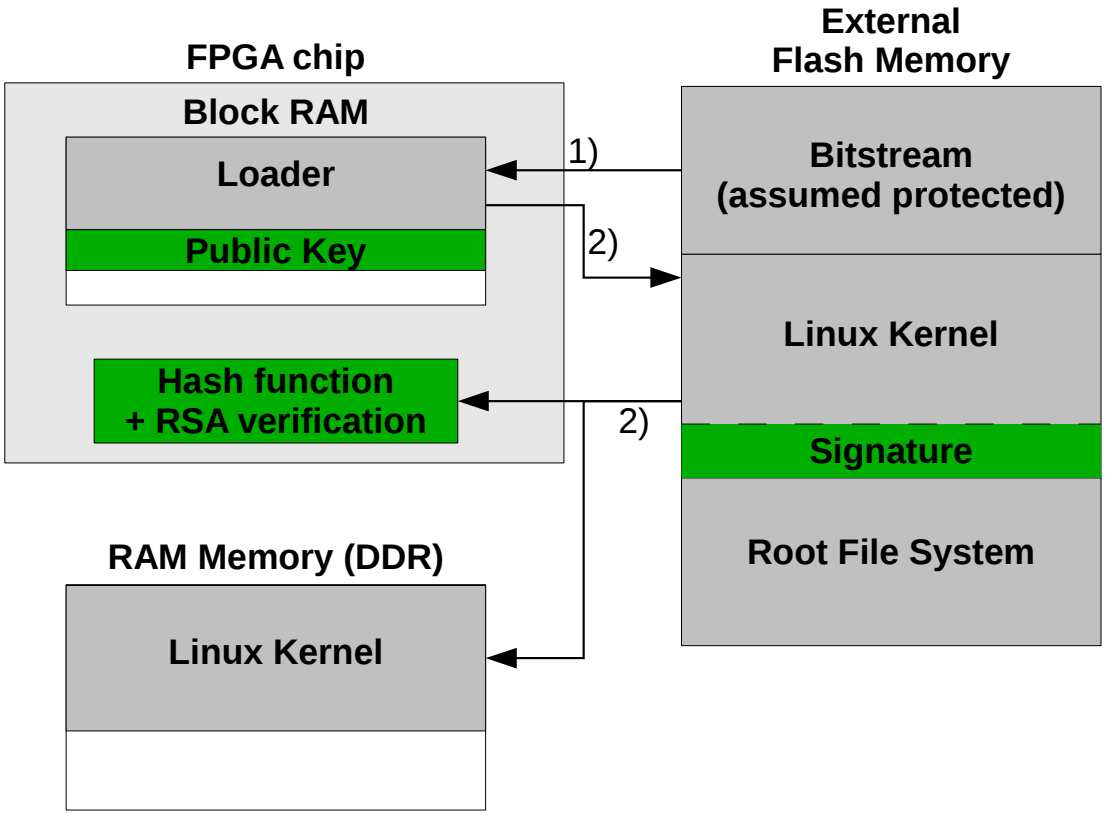
IP	# Cycles	Boot time Overhead	Throughput	Gain
Soft SHA-256	295 860 775	2,959 s	0,95 MB/s	ref.
Hard SHA-256	38 376 545	0,384 s	7,29 MB/s	x7,7
+ DMA transfer	4 221 304	0,042 s	66,67 MB/s	x70

Adding flexibility



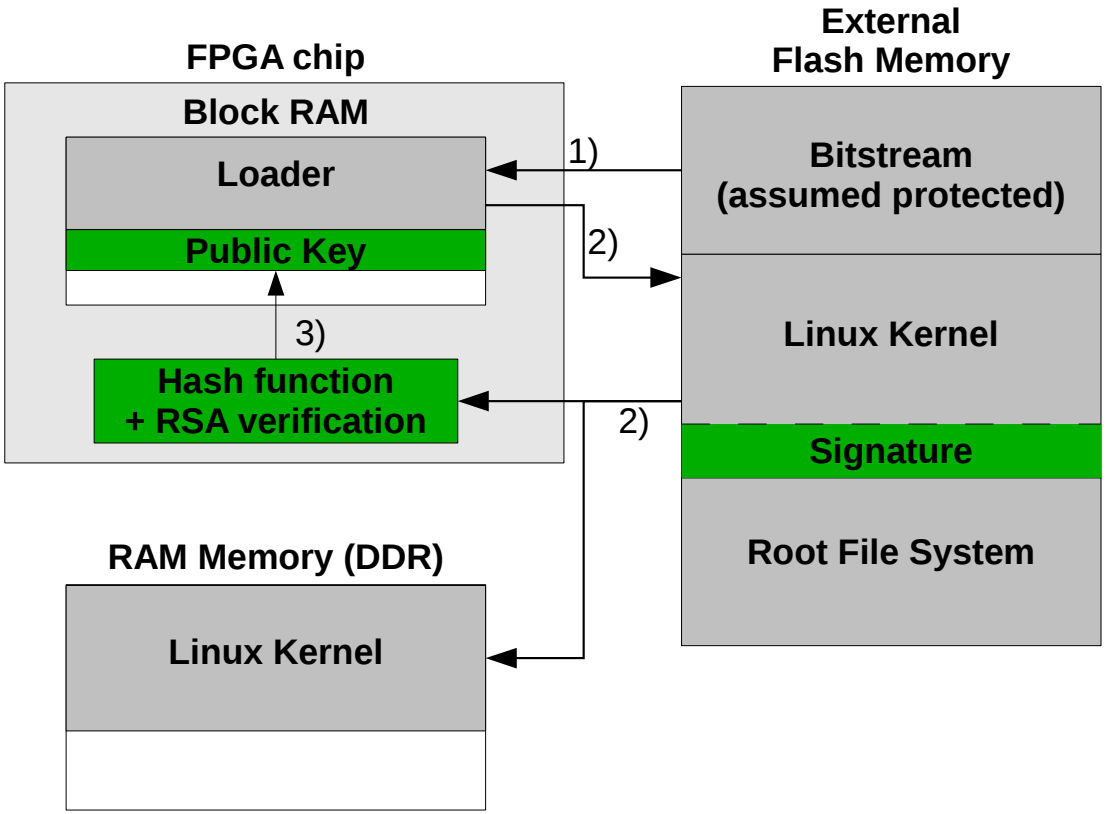
Boot steps :
1) The loader is stored in block RAM at power-up from bitstream
2) The loader copies Kernel from Flash to RAM and compute its hash
3) The loader verifies the Kernel integrity by verifying the signature
4) The loader branches to the Kernel and Linux boots

Adding flexibility



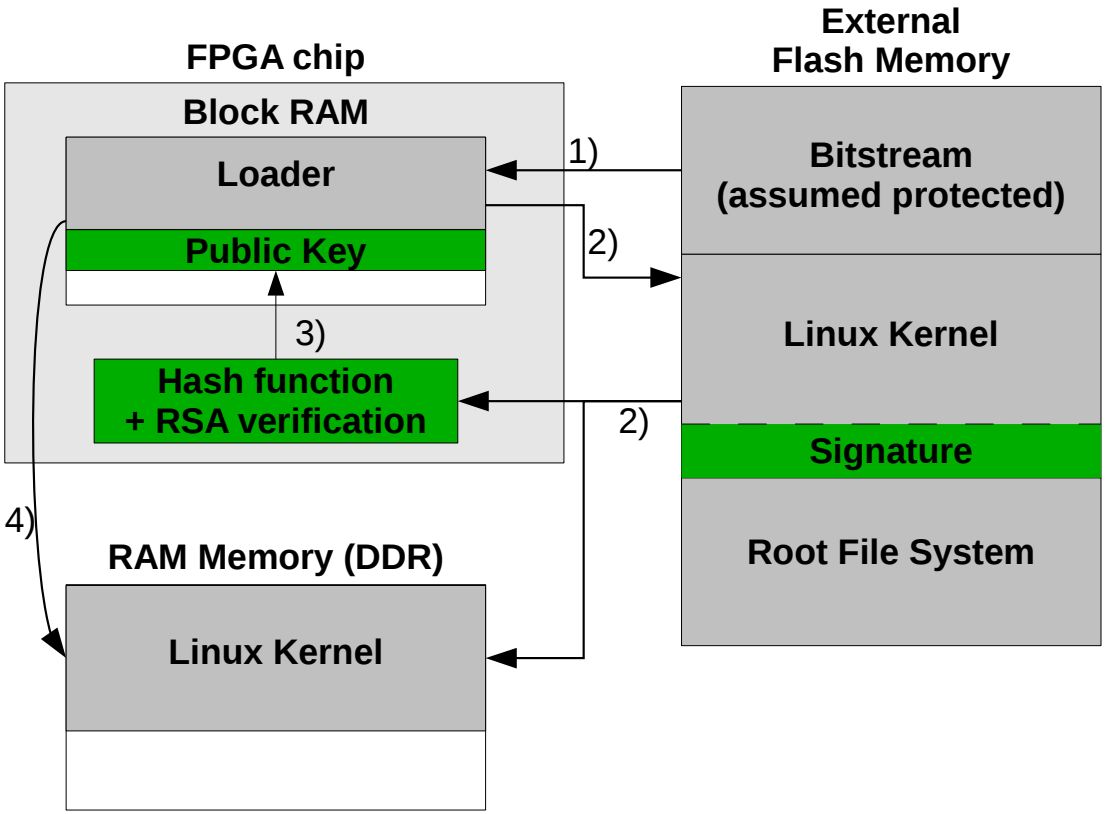
Boot steps :
1) The loader is stored in block RAM at power-up from bitstream
2) The loader copies Kernel from Flash to RAM and compute its hash
3) The loader verifies the Kernel integrity by verifying the signature
4) The loader branches to the Kernel and Linux boots

Adding flexibility



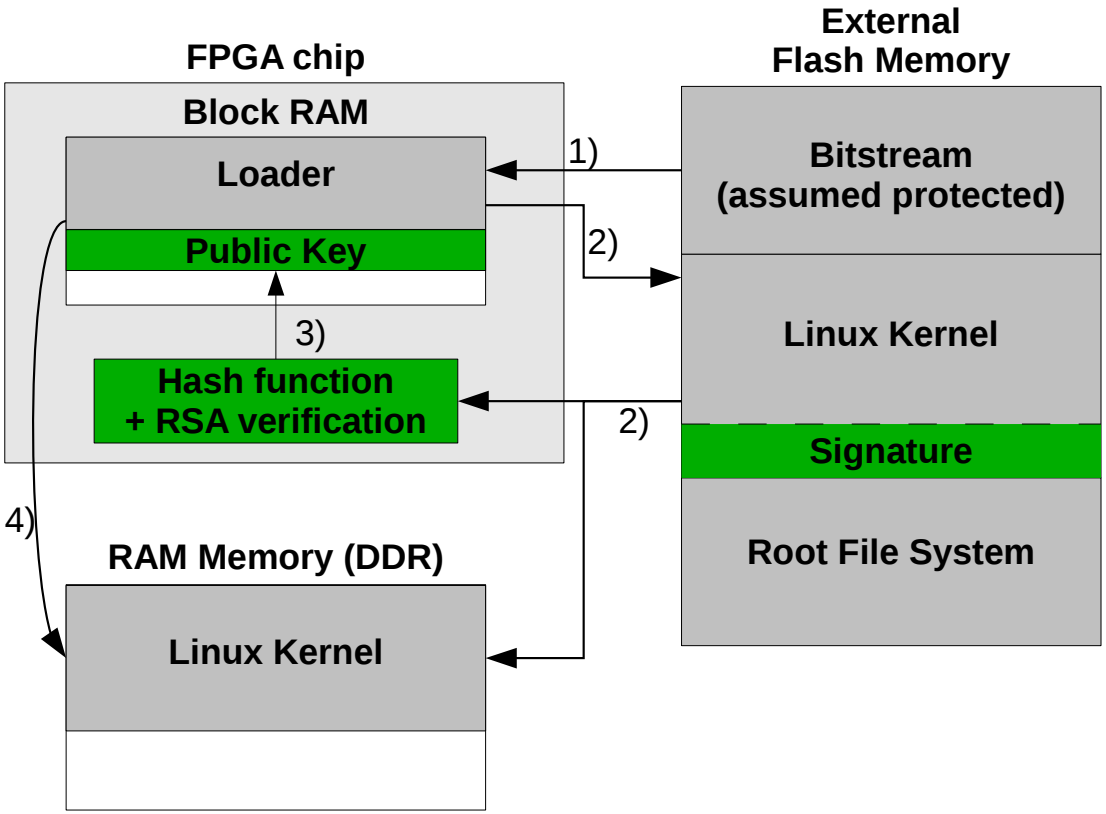
Boot steps :
1) The loader is stored in block RAM at power-up from bitstream
2) The loader copies Kernel from Flash to RAM and compute its hash
3) The loader verifies the Kernel integrity by verifying the signature
4) The loader branches to the Kernel and Linux boots

Adding flexibility



Boot steps :
1) The loader is stored in block RAM at power-up from bitstream
2) The loader copies Kernel from Flash to RAM and compute its hash
3) The loader verifies the Kernel integrity by verifying the signature
4) The loader branches to the Kernel and Linux boots

Adding flexibility



Boot steps :
1) The loader is stored in block RAM at power-up from bitstream
2) The loader copies Kernel from Flash to RAM and compute its hash
3) The loader verifies the Kernel integrity by verifying the signature
4) The loader branches to the Kernel and Linux boots

→ **This flexibility makes the kernel vulnerable to replay attacks**

Security concerns

IP	# Cycles	Boot time Overhead	Throughput	Gain
RSA-1024	92 867	0,001 s	N/A	N/A

Discussion

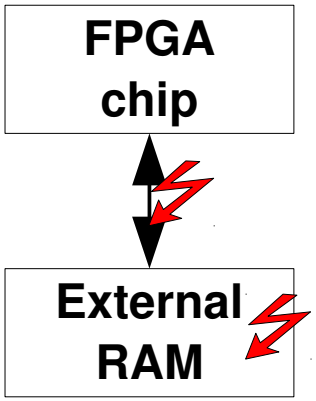
- This flexibility makes the kernel **vulnerable to replay** attacks.
- In case of **critical security update** of the kernel, it is possible to **regenerate an asymmetric key-pair** and update the bitstream.
- *Possibility to store the key-pair in the user non-volatile memory : kernel protected against replay attacks without change the bitstream.*

Area overhead

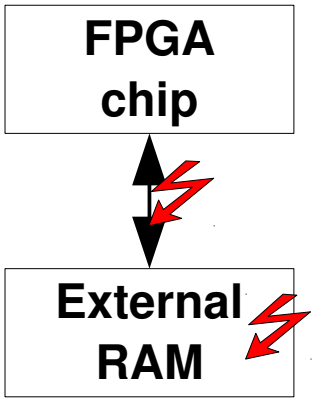
	Details				Total				
Strategies	**Components**	**Slice FF**	**Slice LUT**	**BRAM**	**Slice FF**	**Slice LUT**	**BRAM**	***Fraction of V6 VLX240T***	
Base system (or with soft SHA-256)	Microblaze	3 196	3 874	19					
	Cache	6	14	16					
	DDR3	5 091	4 245	11					
	Flash	479	389						
	PLB	178	657		***8 950***	***9 179***	***46***	***6% + 11% de BRAM***	
+ Hard SHA-256	SHA (+wrapper)	1 509	1 897	1					
	Interrupt ctrl.	190	180		***10 649***	***11 256***	***47***	***7% + 11% de BRAM***	**+1%**
+ DMA	Central DMA	561	799		***11 210***	***12 055***	***47***	***8% + 11% de BRAM***	**+1%**
+ RSA-1024	RSA (+wrapper)	684	989	4	***11 894***	***13 044***	***51***	***9% + 12% de BRAM***	**+1%**

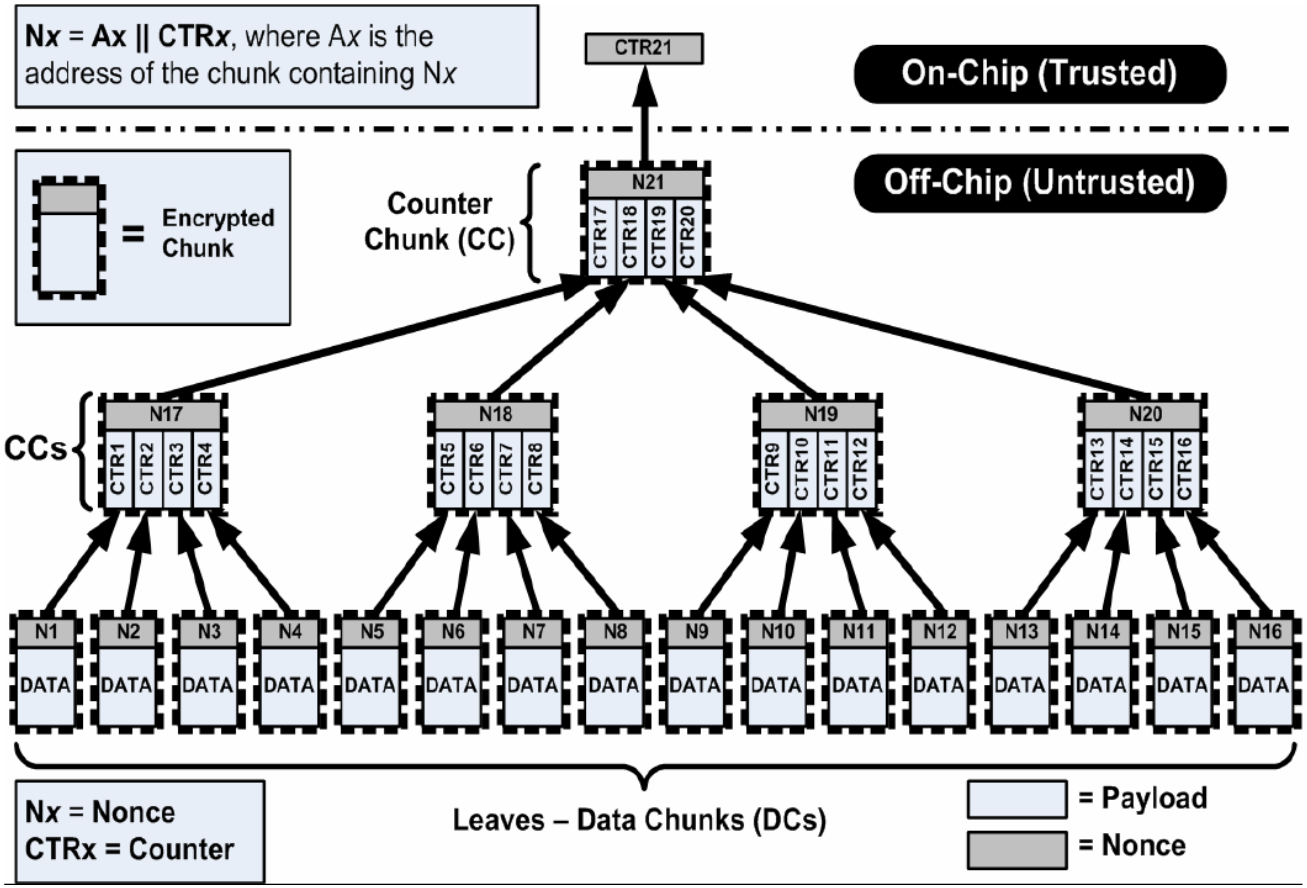
Outline

RAM protection : Merkle-tree



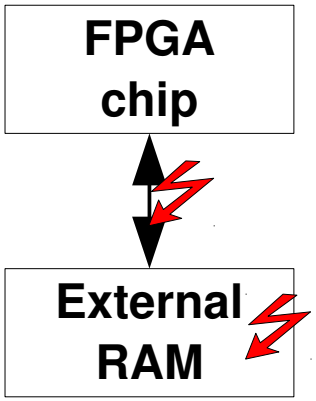
RAM protection : Merkle-tree





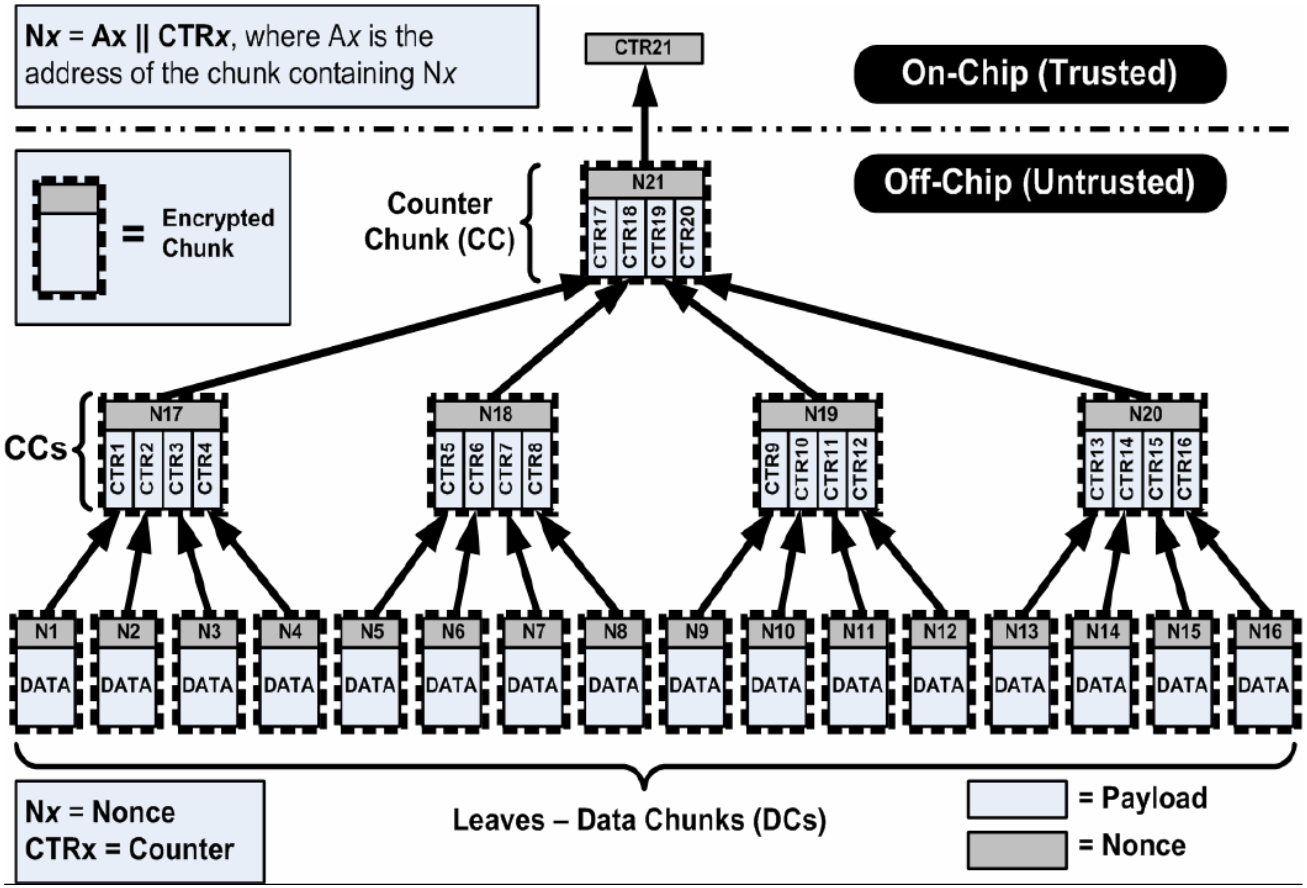
R. Elbaz, D. Champagne, R. Lee and L. Torres, "TEC-Tree : A Low-Cost, Parallelizable Tree for Efficient Defense Against Memory Replay Attacks" in CHES'07

RAM protection : Merkle-tree



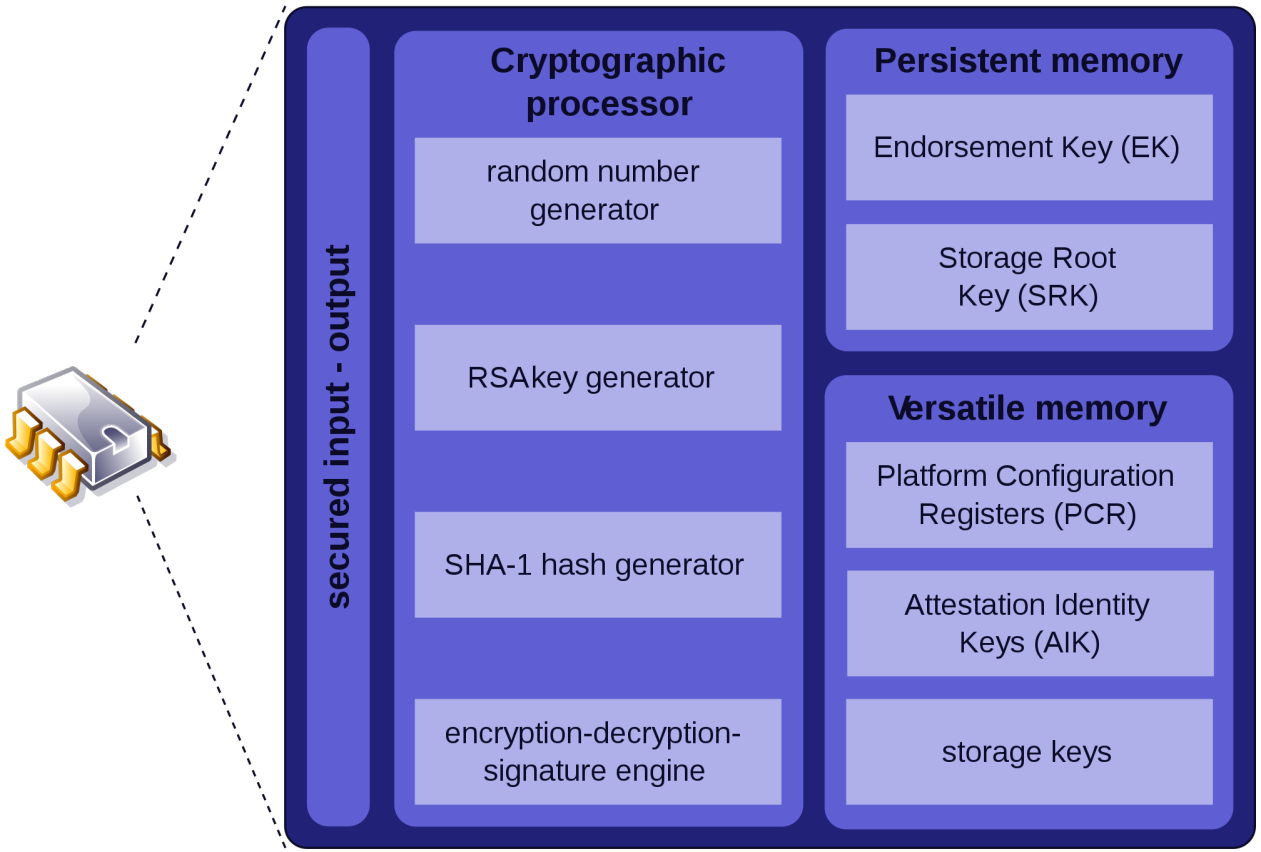
Preventing :

- replaces
- replays
- modifications
- data decryptions

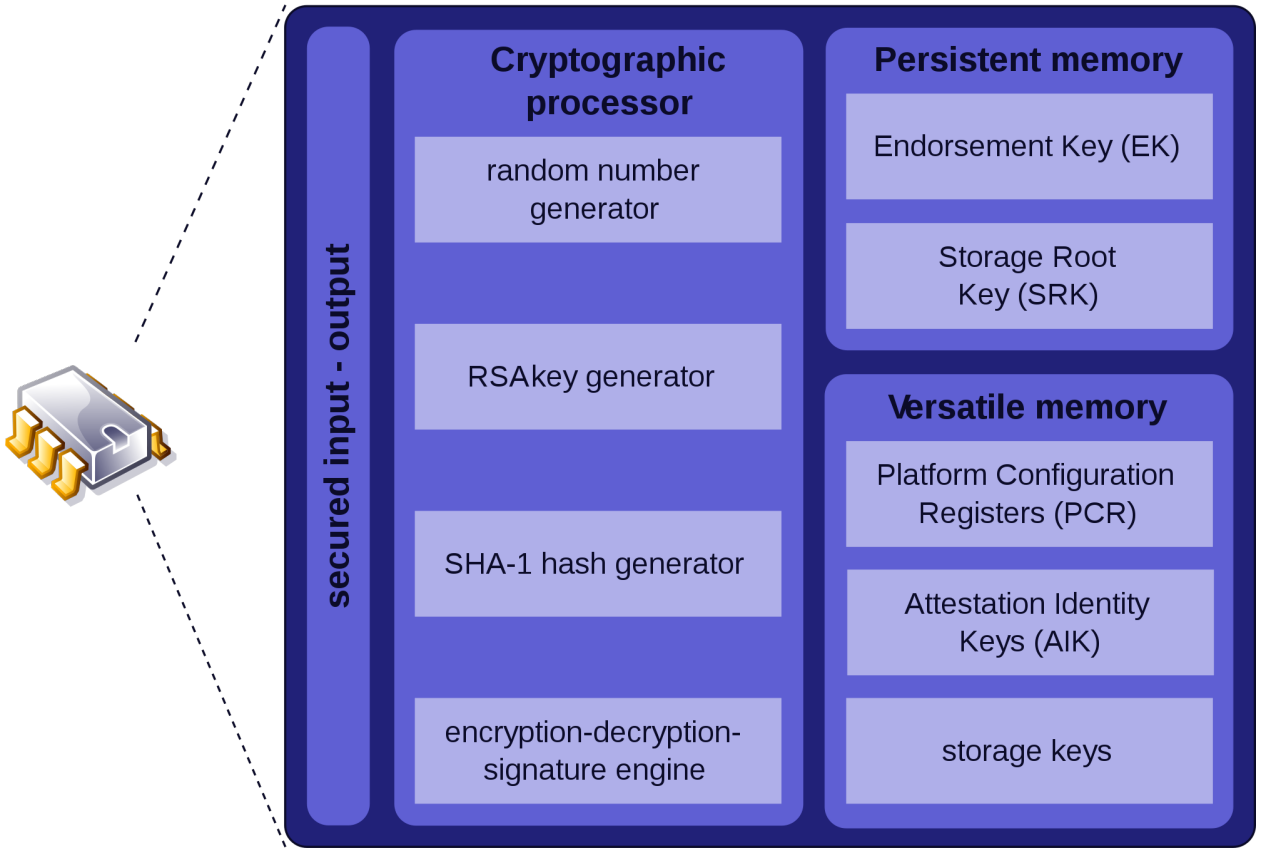


R. Elbaz, D. Champagne, R. Lee and L. Torres, "TEC-Tree : A Low-Cost, Parallelizable Tree for Efficient Defense Against Memory Replay Attacks" in CHES'07

TPM : Trusted Platform Module

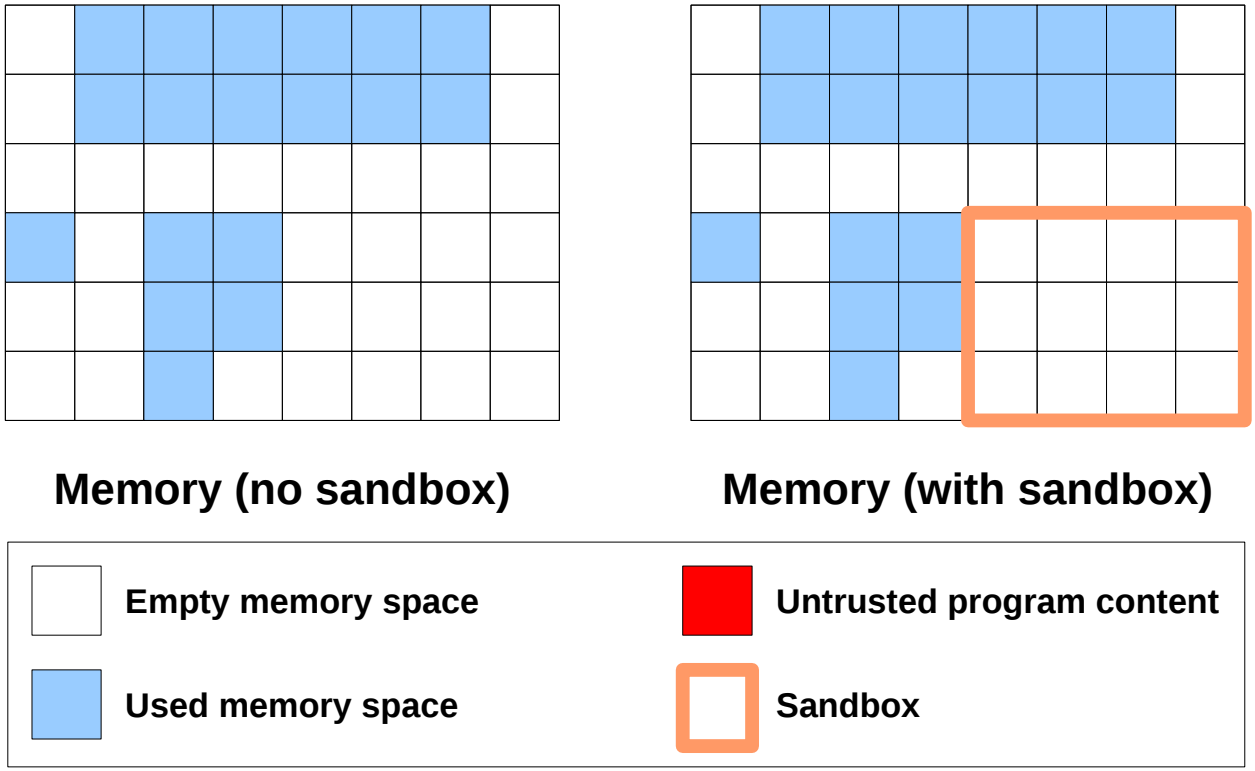


TPM : Trusted Platform Module

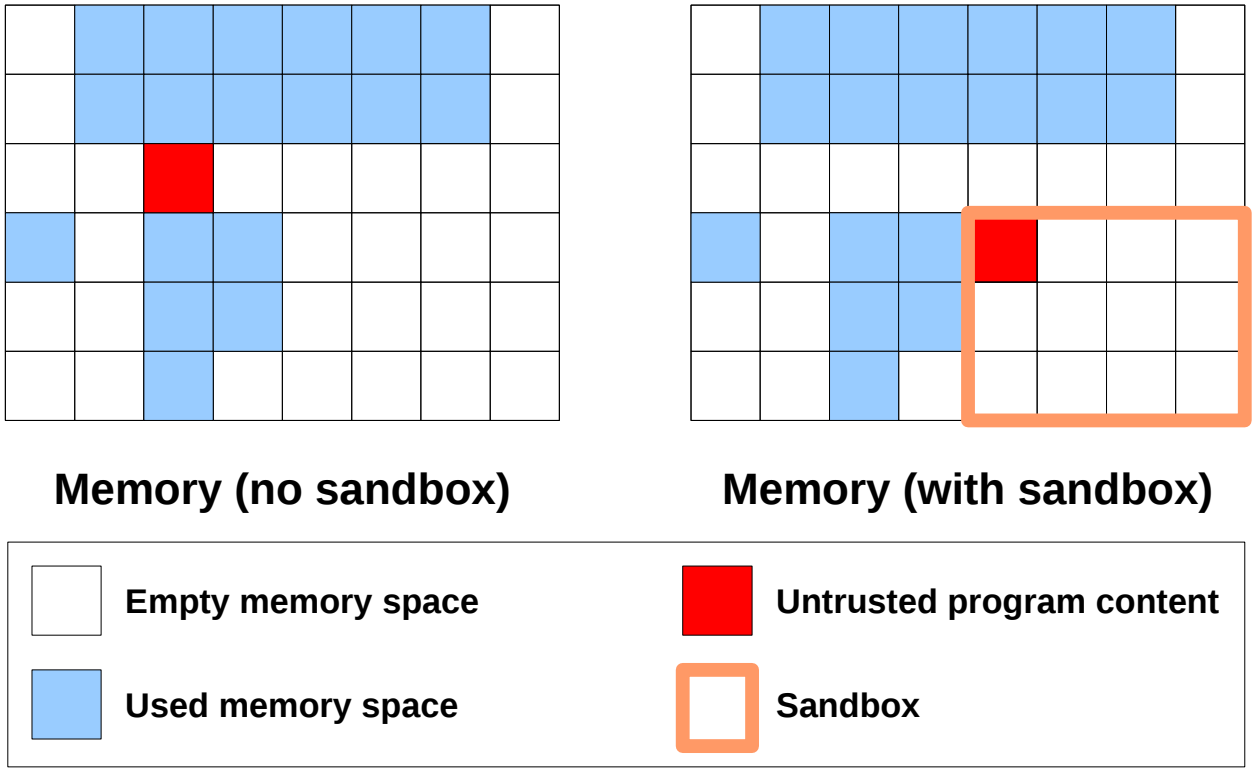


→ **Can be implemented in the FPGA**

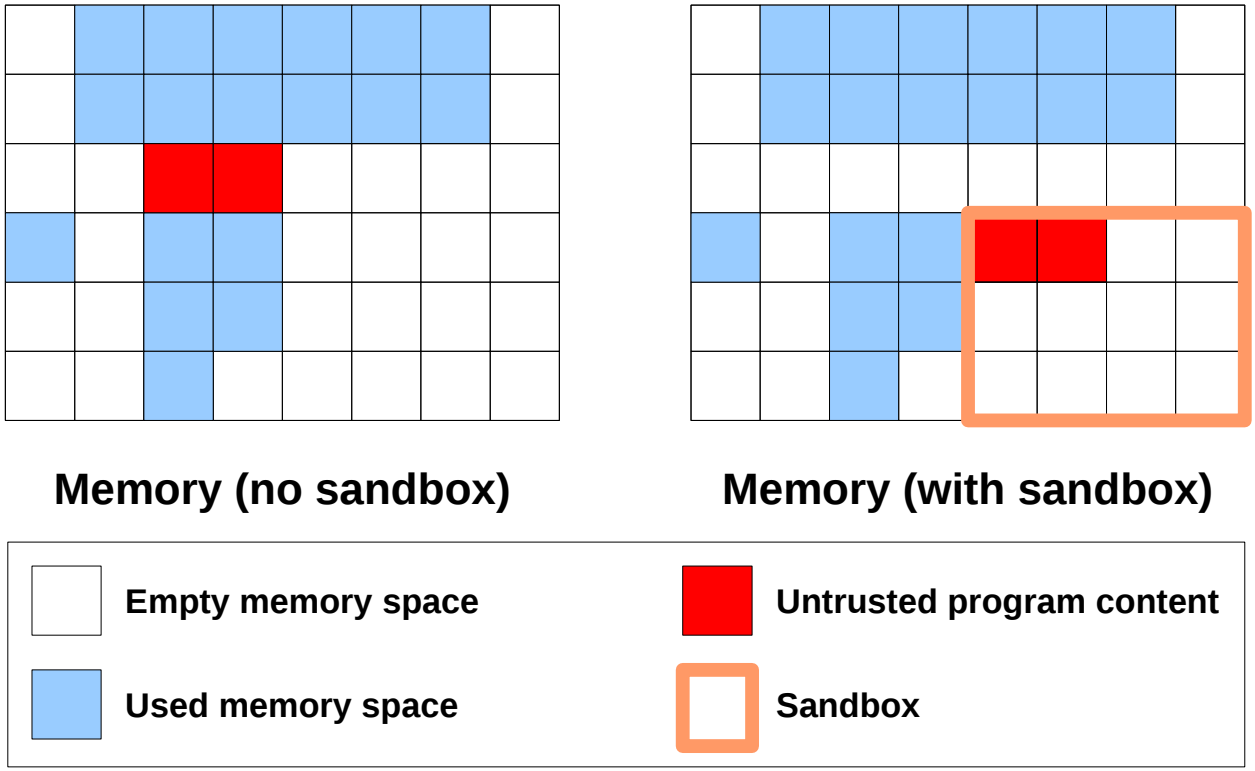
Sandboxing


Memory (no sandbox)
Memory (with sandbox)
Empty memory space
Used memory space
Untrusted program content
Sandbox

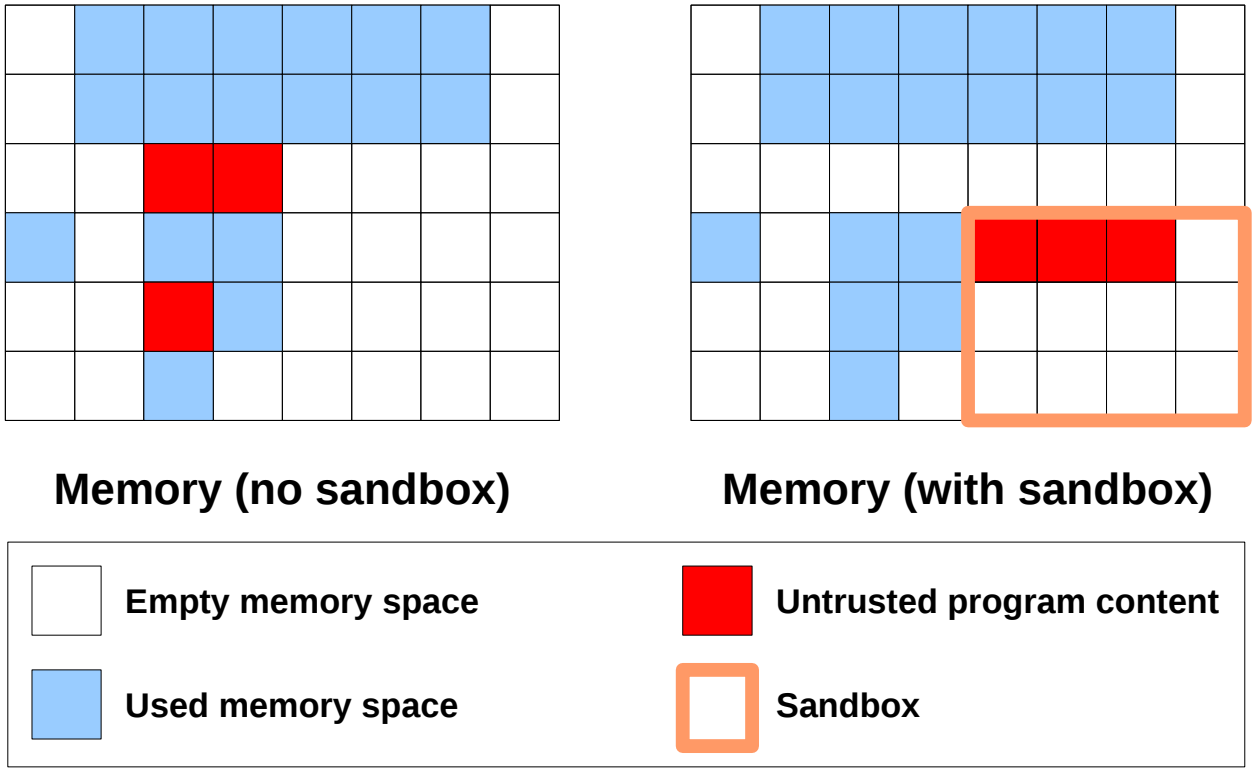
Sandboxing


Memory (no sandbox)
Memory (with sandbox)
Empty memory space
Untrusted program content
Used memory space
Sandbox

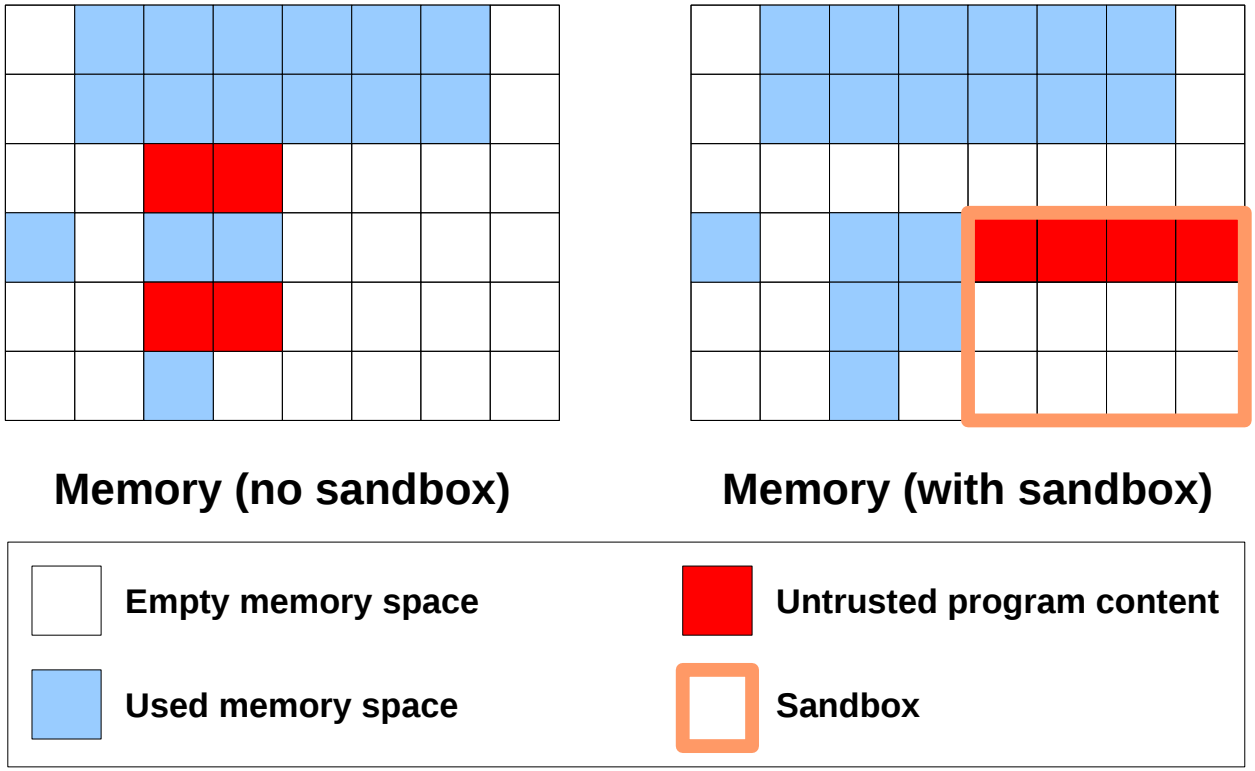
Sandboxing


Memory (no sandbox)
Memory (with sandbox)
Empty memory space
Untrusted program content
Used memory space
Sandbox

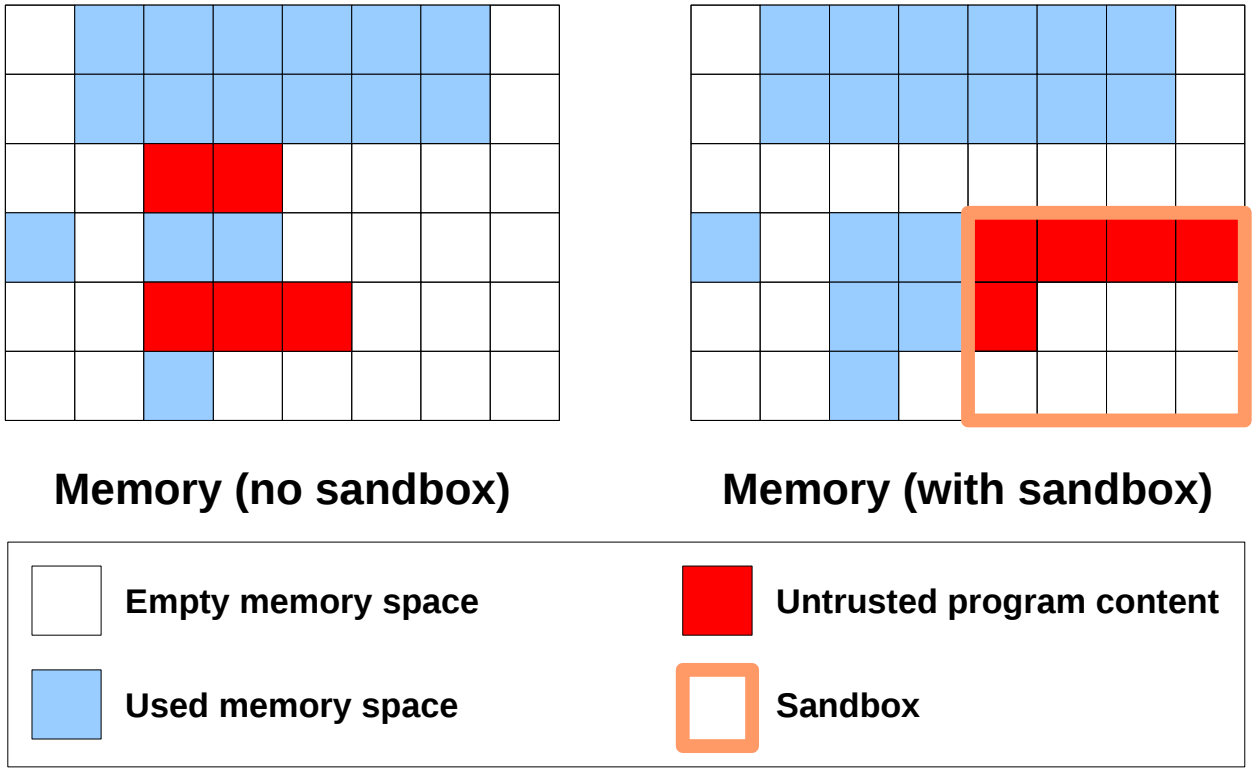
Sandboxing


Memory (no sandbox)
Memory (with sandbox)
Empty memory space
Untrusted program content
Used memory space
Sandbox

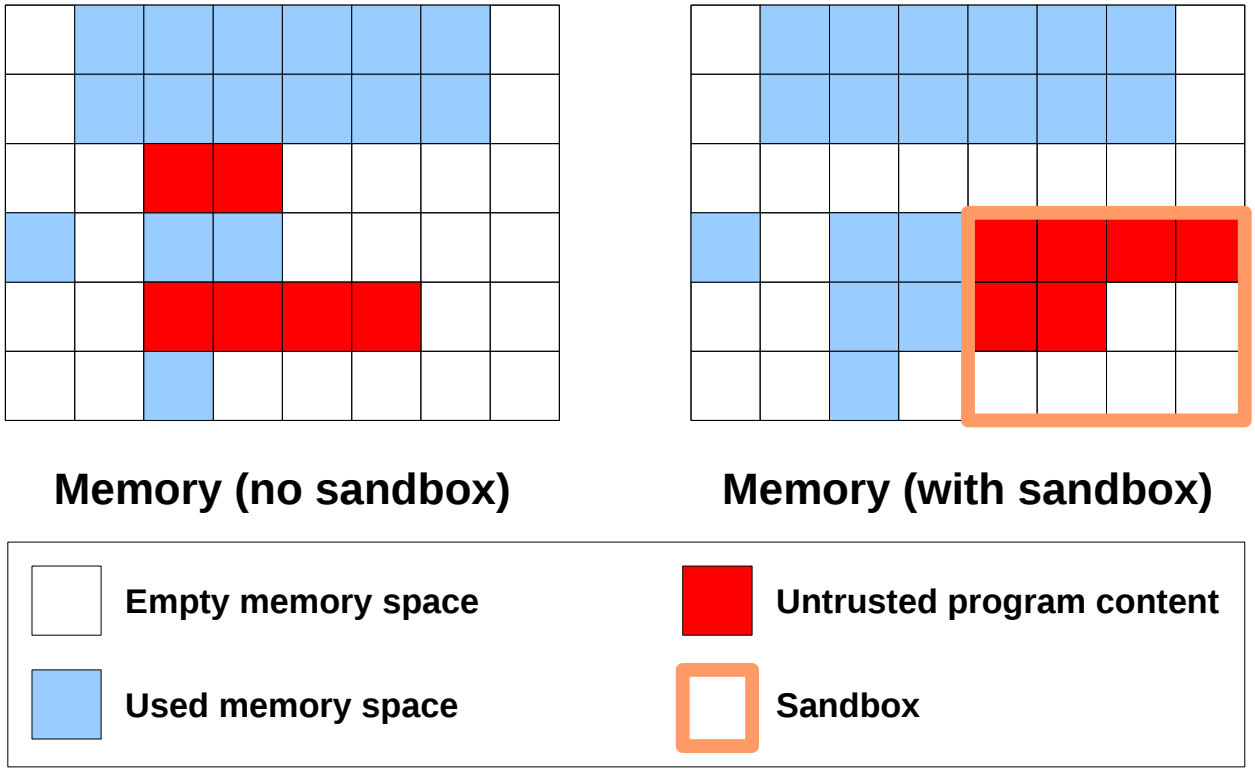
Sandboxing


Memory (no sandbox)
Memory (with sandbox)
Empty memory space
Untrusted program content
Used memory space
Sandbox

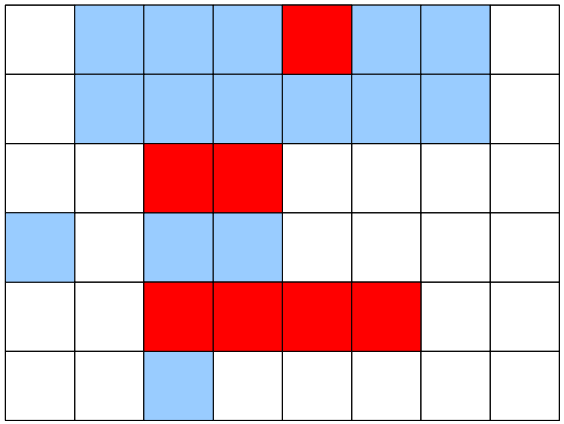
Sandboxing


Memory (no sandbox)
Memory (with sandbox)
Empty memory space
Untrusted program content
Used memory space
Sandbox

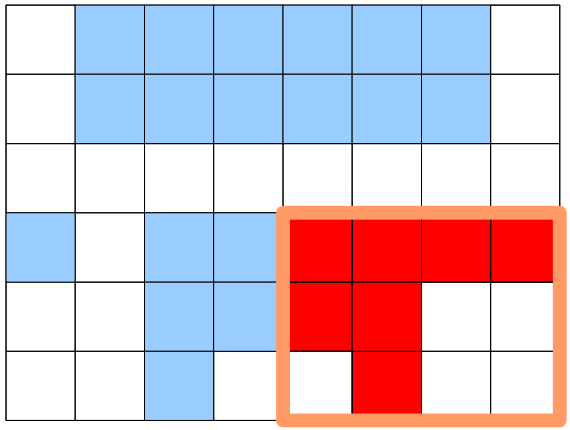
Sandboxing


Memory (no sandbox)
Memory (with sandbox)
Empty memory space
Untrusted program content
Used memory space
Sandbox

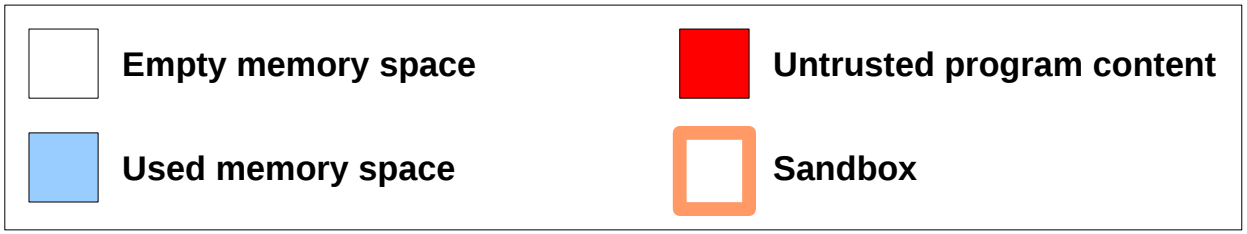
Sandboxing



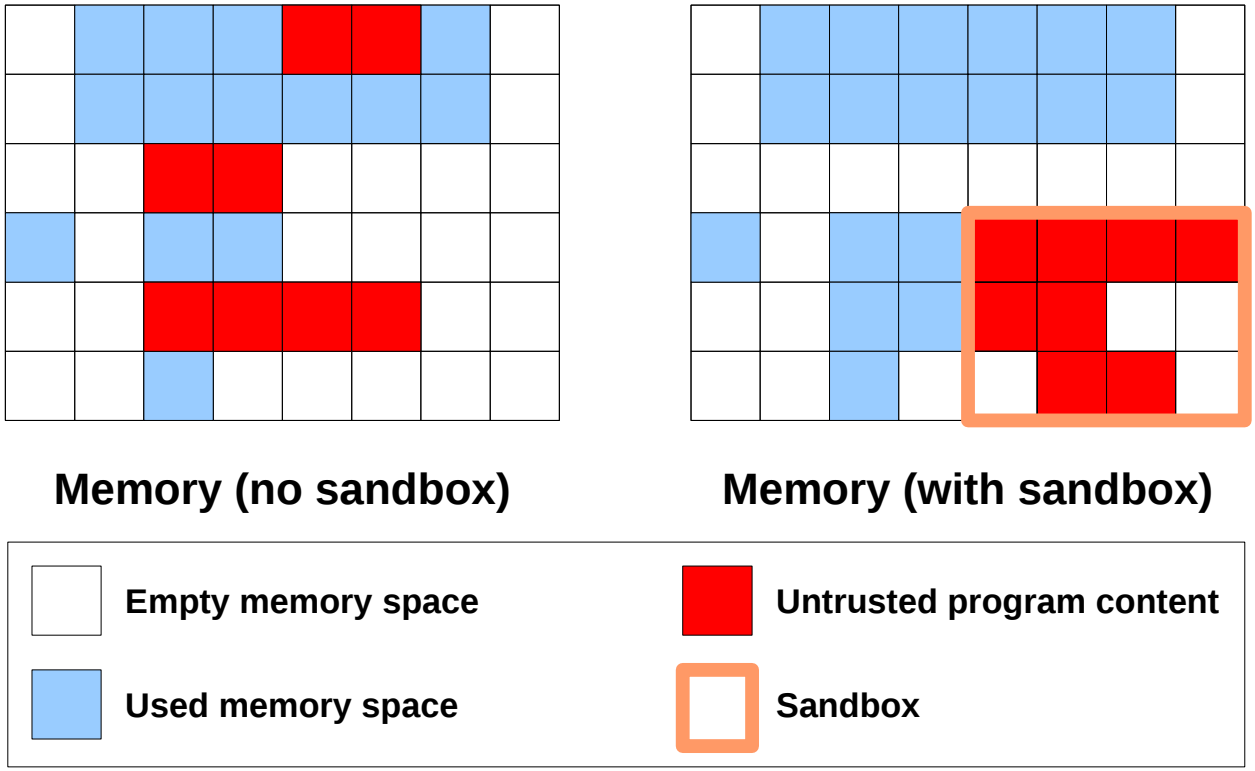
Memory (no sandbox)



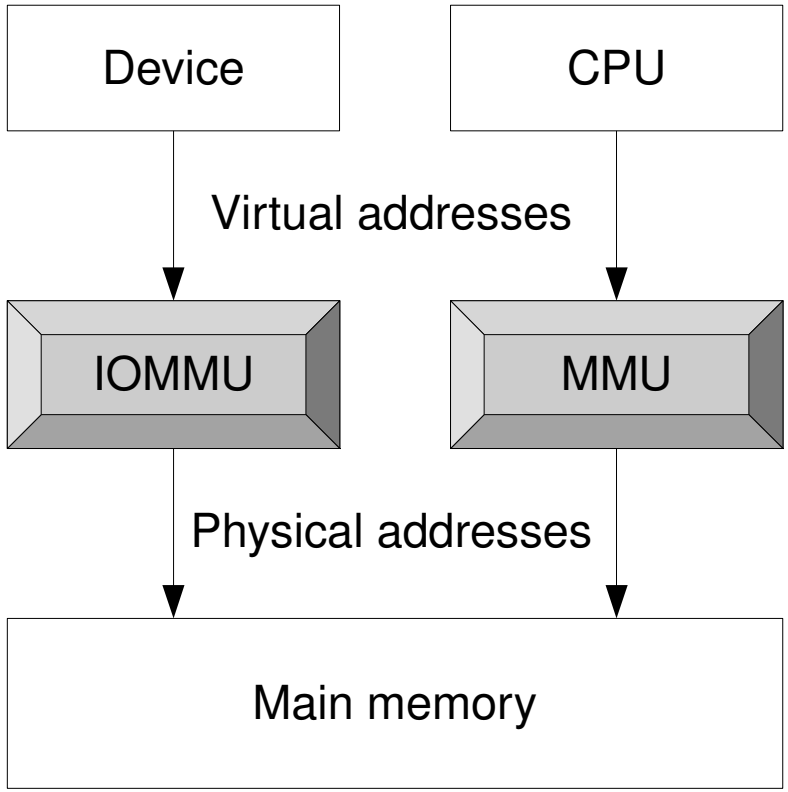
Memory (with sandbox)



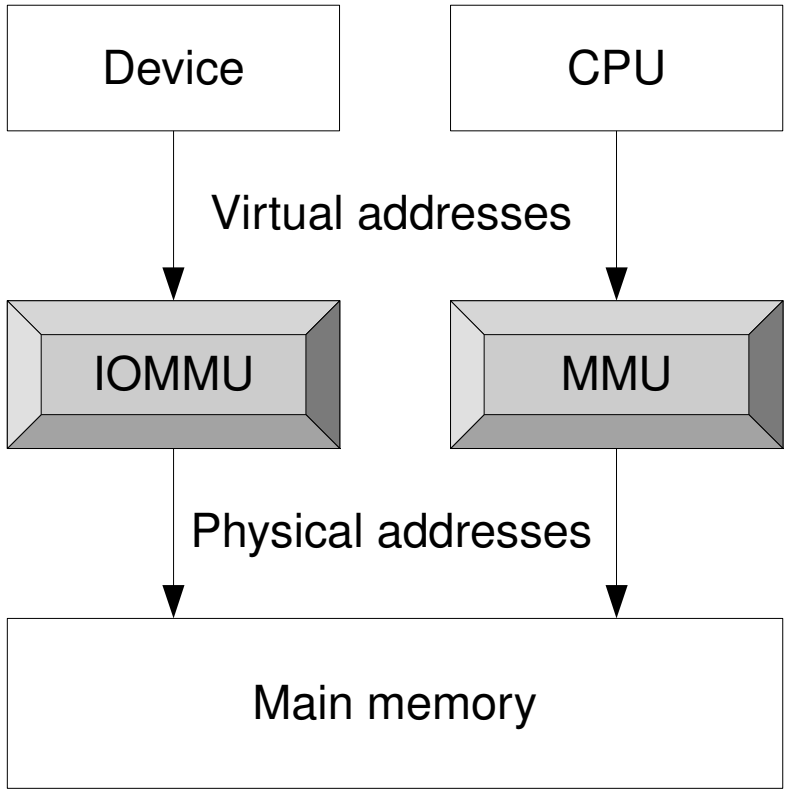
Sandboxing


Memory (no sandbox)
Memory (with sandbox)
Empty memory space
Untrusted program content
Used memory space
Sandbox

Memory rights management



Memory rights management

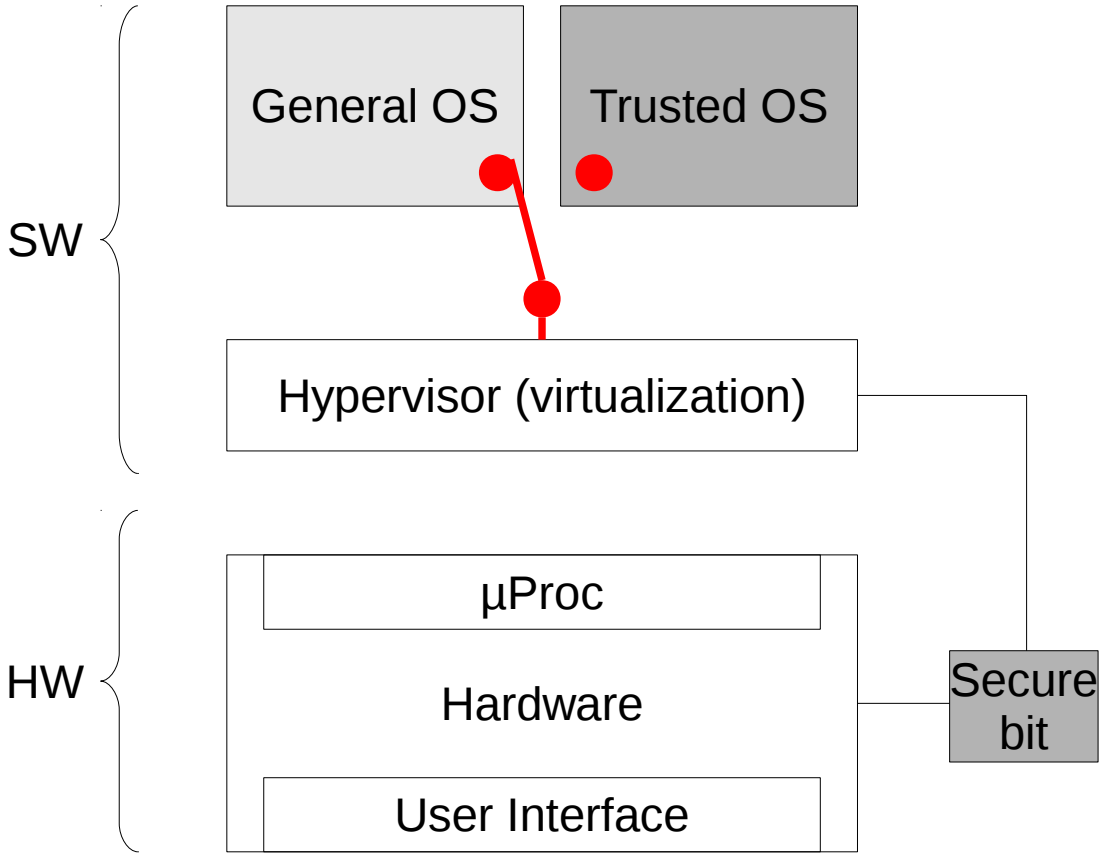


→ **Vulnerabilities**

F. L. Sang, E. Lacombe, V. Nicomette, and Y. Deswarte, "Analyse de l'efficacité du service fourni par une IOMMU" in SSTIC'10

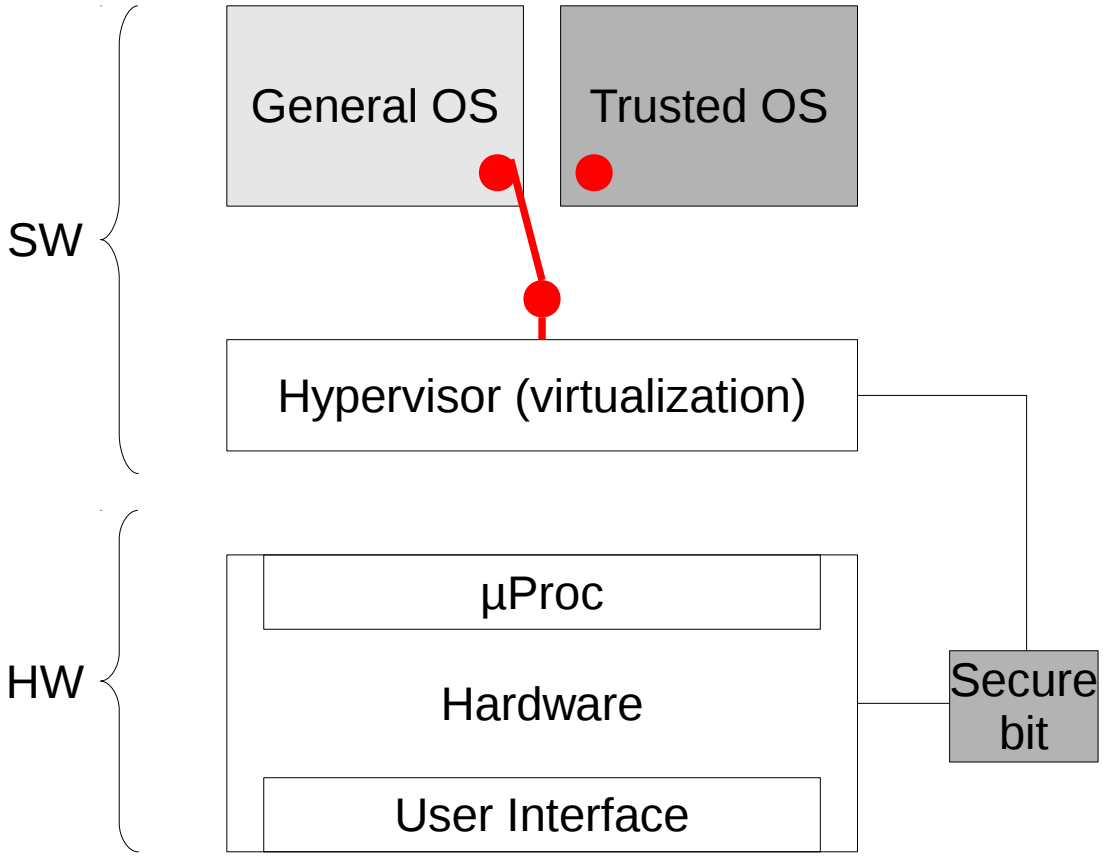
Virtual machines

example : TrustZone (ARM)



Virtual machines

example : TrustZone (ARM)



→ **Vulnerabilities**

Outline

Conclusion and future works

- Complete protection : very difficult
  - Large attack surface : Bitstream $\rightarrow$ OS
  - Multidisciplinary skills : Avoid security holes
- RAM protection : high performance overhead
- OS : difficult to trust and certify

Conclusion and future works

Thank you for your attention ! ! !