Securing the Shakti Processors

Arjun Menon

SHAKTI Group | CSE Dept | RISE Lab | IIT Madras

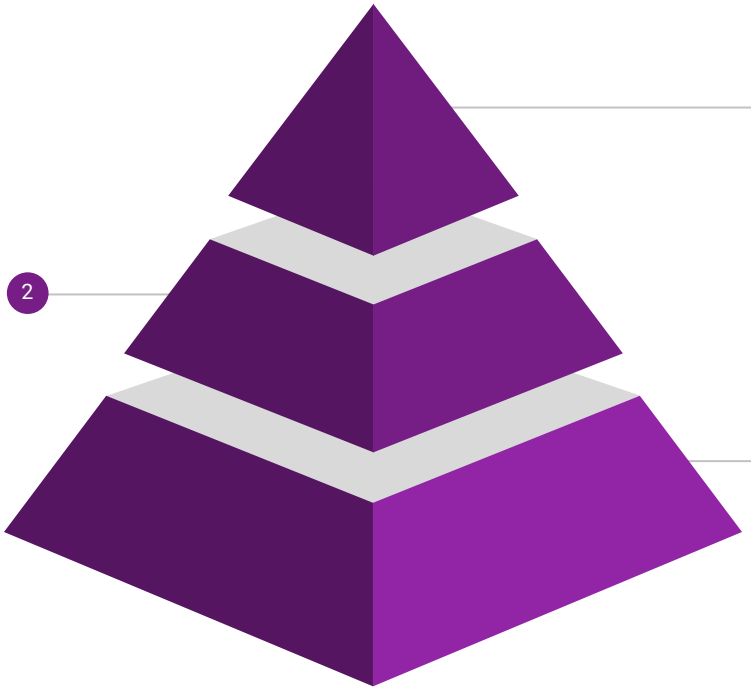
Microprocessor Attack Surface

Vulnerabilities in Software

1 Memory Vulnerabilities, Control Flow Attacks, Integer Overflows, Format String; Privilege Escalation

Vulnerabilities due to Micro-architecture

2 Cache Timing Attacks; TLB attacks; Speculative attacks (Meltdown, Spectre, Foreshadow, ZombieLoad)



Vulnerabilities in Hardware

3

Non-invasive:	Power Analysis; EM Attacks;
Semi-invasive:	Fault injection
Invasive:	Bus probing

SHAKTI

Security Research at IIT Madras

Vulnerabilities in Software

Fat pointers; Hardware Enabled Compartments; Safe-languages (OCaml)

Vulnerabilities due to Micro-architecture

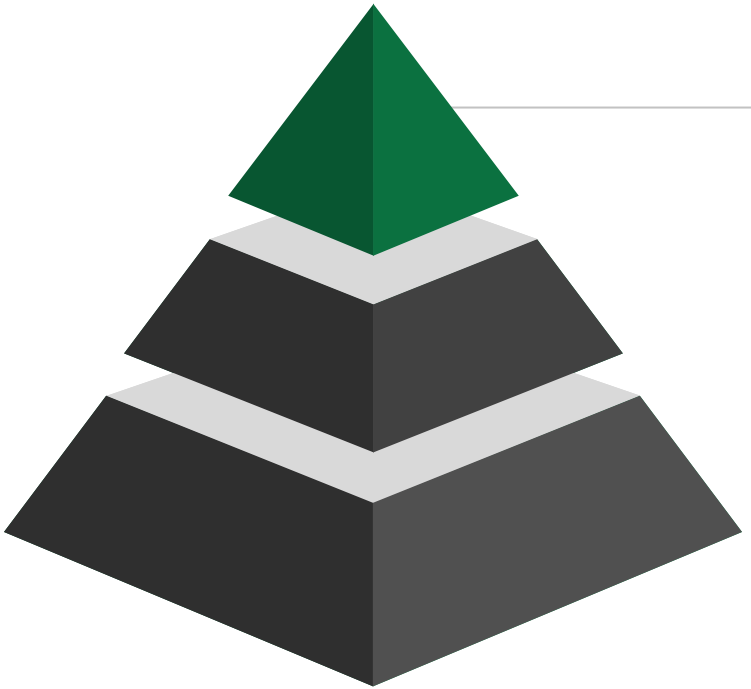
Speculation Safe, Timing Attack Protection, Security Counters

Vulnerabilities in Hardware

Power Attack Secure Microprocessor, EDA tools for Power Attack Security

SHAKTI

Security Research at IIT Madras



Vulnerabilities in Software

1. Fat pointers; Hardware Enabled Compartments; Safe-languages (OCaml)

SHAKTI

Memory attacks

Spatial

```
char dest[10], source[20];
strcpy(dest,source);
```

- Accessing data beyond the valid range of addresses
- Two types:
  - Write overflow
  - Read overflow

Temporal

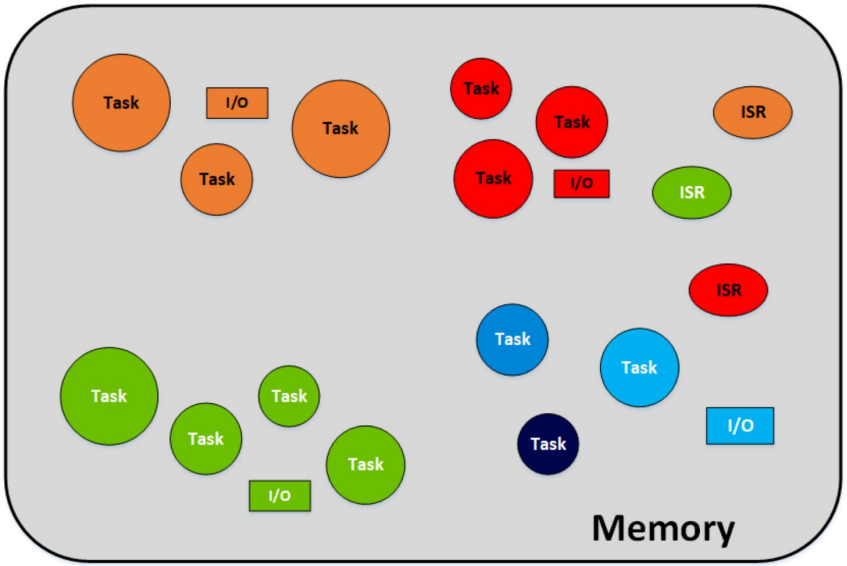
```
int *ptr= malloc(20);
...
free(ptr)
...
int b=*ptr;
```

- Accessing stale data
  - Use-after-free
- Freeing an already freed memory
  - Double-free

What does RISC-V have?

- Physical Memory Protection (PMP)
  - Upto 16 physical memory regions which can be specified at a granularity of 4 bytes
  - Every access is checked if it is within one of the these regions
  - Attributes:
    - Read
    - Write
    - Execute
    - Granularity
    - Locked

Real Time Operating System (RTOS)



- Collection of tasks
- Scheduler that schedules these tasks
- No virtualisation
  - Each task has access to complete physical memory

Image src: ttps://content.riscv.org/wp-content/uploads/2018/12/Using-the-RISC-V-PMP-with-an-Embedded-RTOS-to-Achieve-Process-Separation-and-Isolation-Labrosse.pdf

SHAKTI

How is PMP useful?

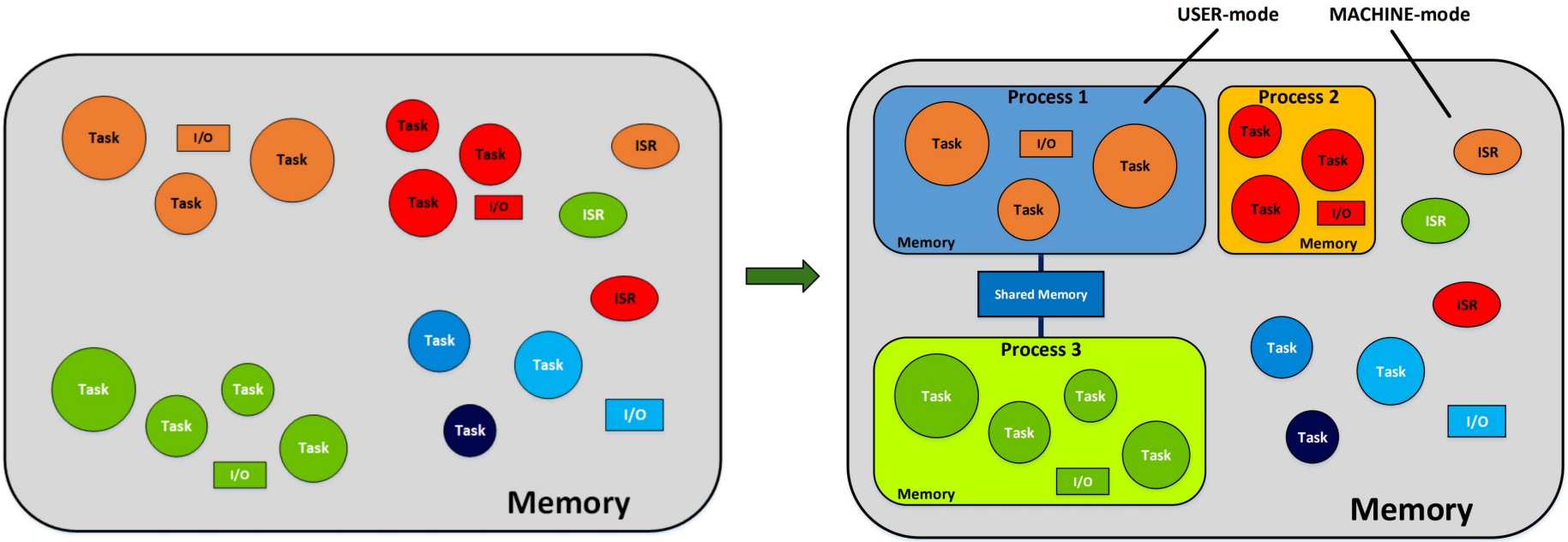


Image src: ttps://content.riscv.org/wp-content/uploads/2018/12/Using-the-RISC-V-PMP-with-an-Embedded-RTOS-to-Achieve-Process-Separation-and-Isolation-Labrosse.pdf

SHAKTI

How is PMP useful?

- Red zone
  - Small area at the bottom of each stack
  - CPU exception if data is pushed into that area
  - Not guaranteed to catch every overflow

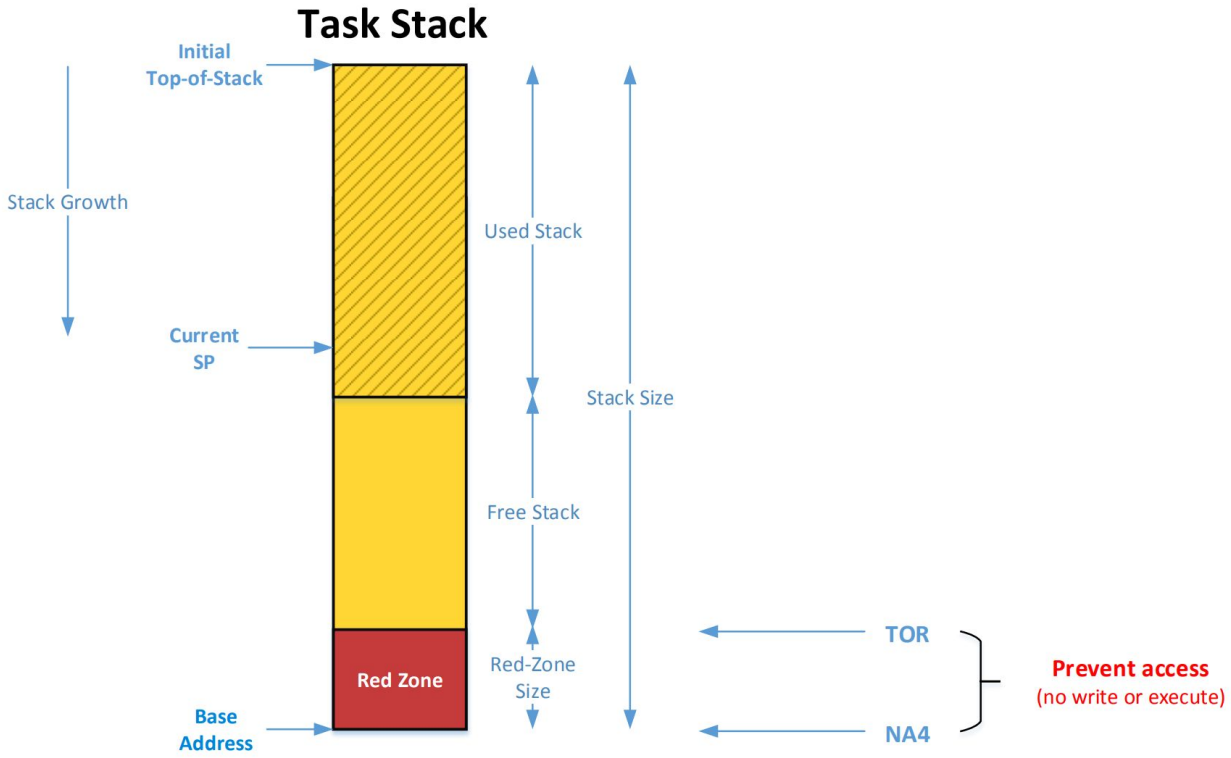


Image src: ttps://content.riscv.org/wp-content/uploads/2018/12/Using-the-RISC-V-PMP-with-an-Embedded-RTOS-to-Achieve-Process-Separation-and-Isolation-Labrosse.pdf

Shakti-T: A RISC-V Processor with Light Weight Security Extensions

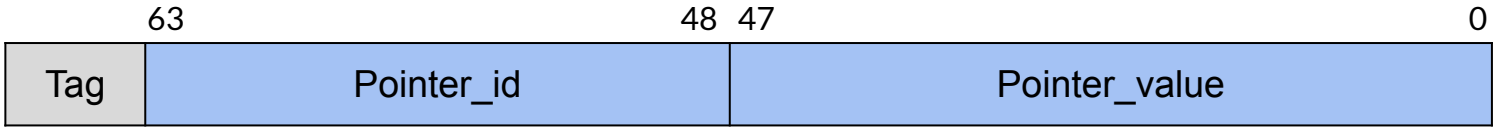
Arjun Menon, Subadra Murugan, Chester Rebeiro, Neel Gala and Kamakoti Veezhinathan

@HASP'17

SHAKTI

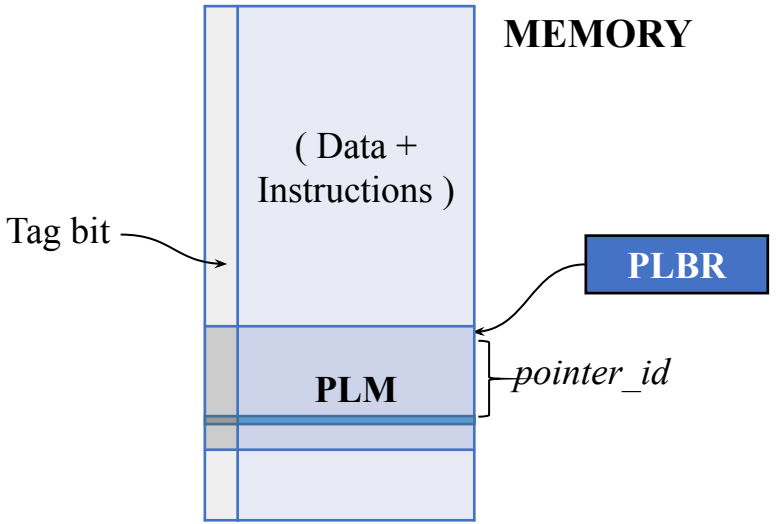
The Idea

- Tagged architecture with fat-pointers to prevent spatial and temporal buffer overflows
- 1-bit tag that indicates if a processor word is pointer or data
- Fat-pointer structure:



The Solution

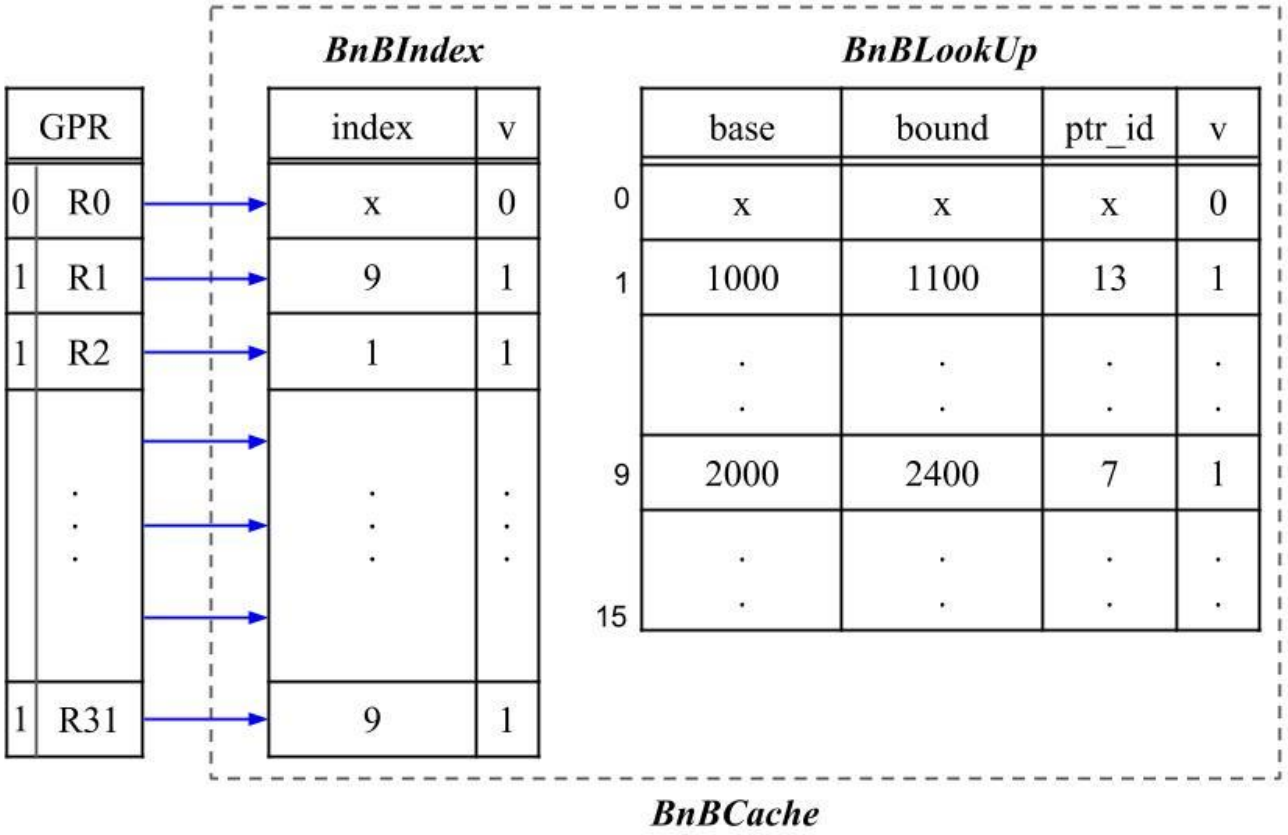
1. Have a common memory region called Pointer Limits Memory (PLM) to store the values of base and bounds
   - Declare a new CSR register which points to the base address of PLM
   - Base and bounds are associated with a pointer by the value of the offset (*pointer_id*)
2. Add a 1-bit tag to every memory word
   - 0: Data/Instruction
   - 1: Pointer



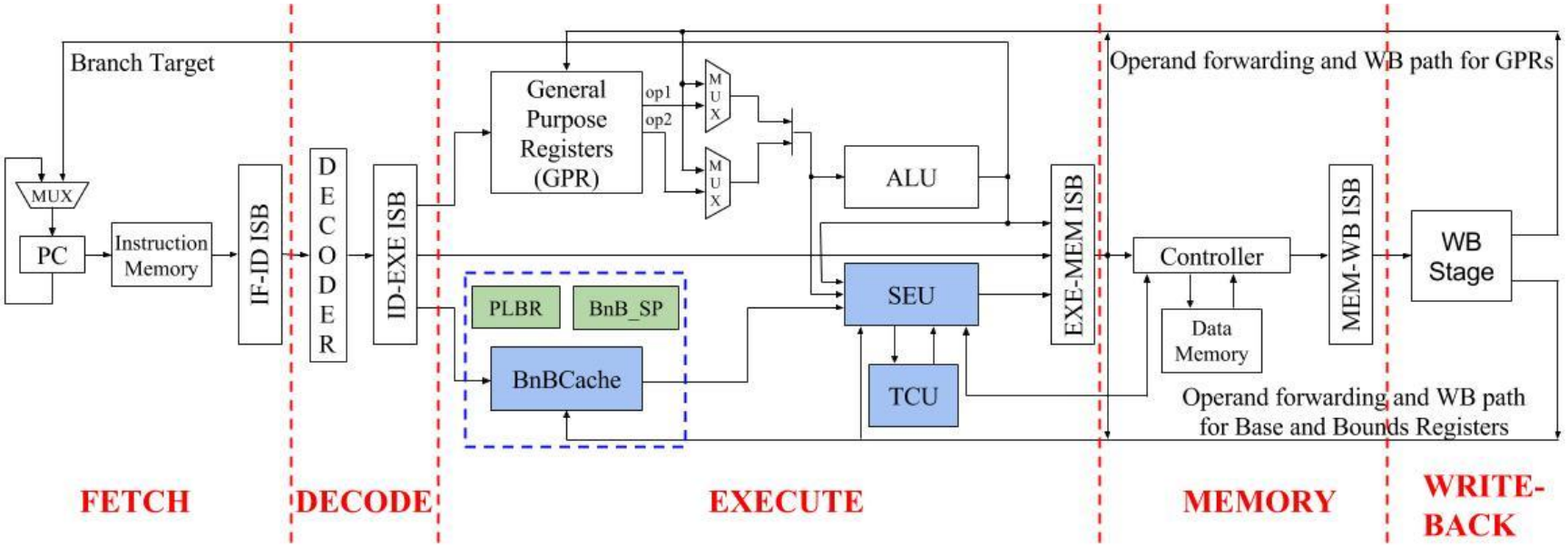
The Solution

3. Maintain a separate table alongside the register file that stores the values of base and bounds (and also, the *pointer_id*)
    - One-level indexing is used to associate a GPR holding a pointer with its corresponding values of base and bounds

The Solution



Microarchitecture



Tag Computation Unit

- Compute the tag of the result based on the operand tags and the operation
- Examples:

Operation	Operand 1	Operand 2	Result
Add immediate	Pointer	Immediate value	Pointer
Add	Pointer	Data	Pointer
Mul	Pointer	X	Exception
Subtract	Pointer	Pointer	Data

- Future implementations with multiple tag bits that help enforce stronger security properties

Comparison with Existing Work

	Safety checking	Instrumentation methodology	Metadata size for *n* aliased pointers	Memory fragmentation	Performance overhead (delay)
Intel MPX [1]	Spatial	Compiler	128 x *n*	No	N/A
HardBound [2]	Spatial	Hardware	128 x *n*	No	HW: N/A SW: 10%
Low-fat Pointer [3]	Spatial	Hardware	0	Yes	HW: 5%
Watchdog [4]	Spatial & Temporal	Compiler + Hardware	(256 x *n*) + 64	No	HW: N/A SW: 25%
WatchdogLite [5]	Spatial & Temporal	Compiler	(256 x *n*) + 64	No	SW: 29%
Shakti-T [6]	Spatial & Temporal	Hardware	(64 x *n*) + 128	No	HW: 1.5%

Shakti-MS: A RISC-V Processor for Memory Safety in C

Sourav Das, R. Harikrishnan Unnithan, Arjun Menon, Chester Rebeiro and Kamakoti Veezhinathan

@LCTES'19

SHAKTI

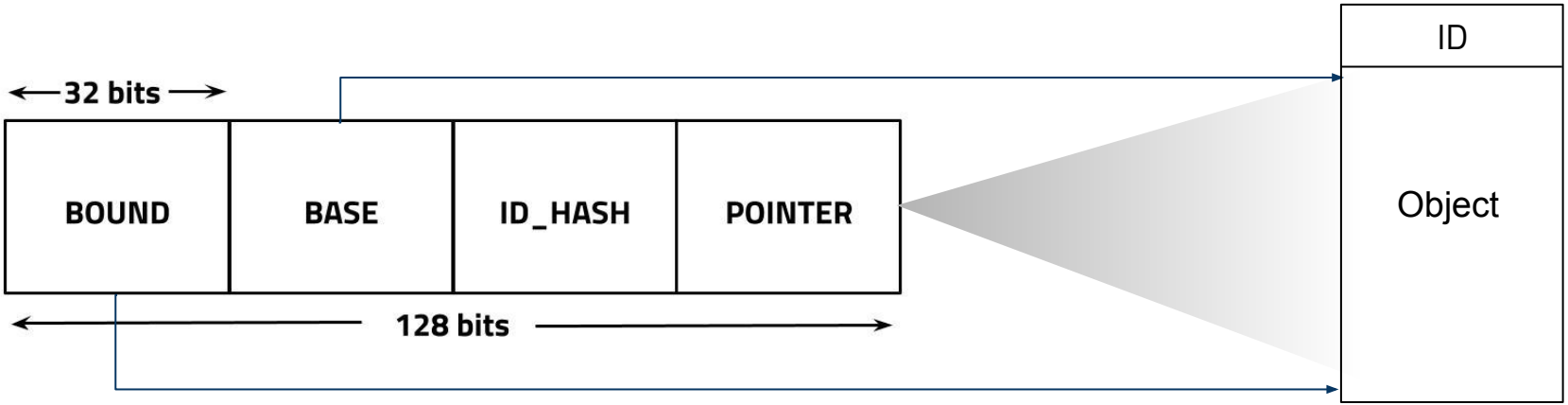
Contributions

1. Low overhead fat pointers ensuring spatial and temporal safety for embedded systems
2. LLVM based open source compiler framework
3. Hardware extensions on open-sourced Shakti ecosystem

An end to end open-sourced hardware-software solution for memory safety in C

Compiler framework: https://github.com/illustris/riscv-llvm-toolchain
Custom hardware: https://bitbucket.org/arjunmenon/sec-c
Docker image is available at *illustris/shakti-ms-artefacts* in dockerhub.

Fat-pointer



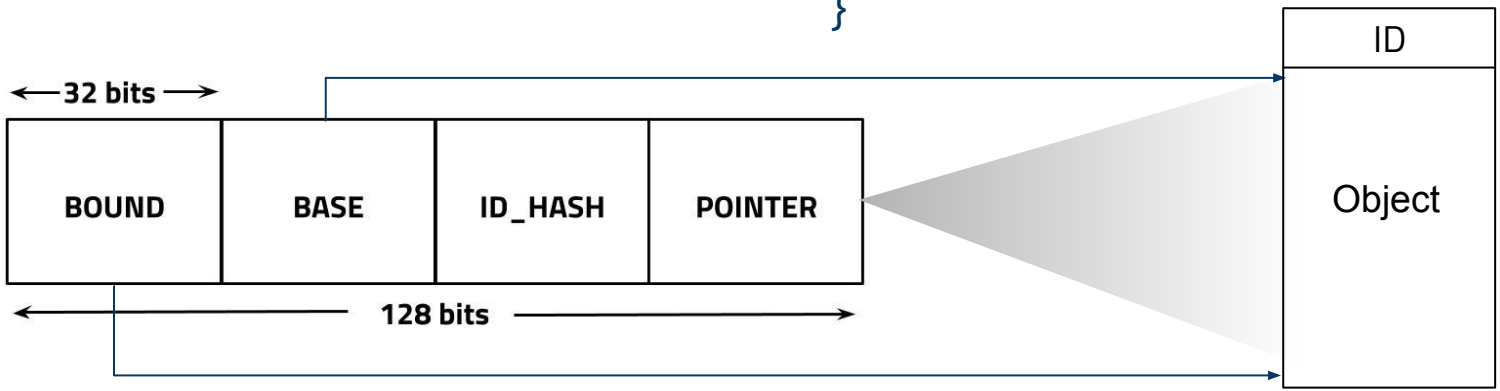
Fat-pointer

Spatial Safety

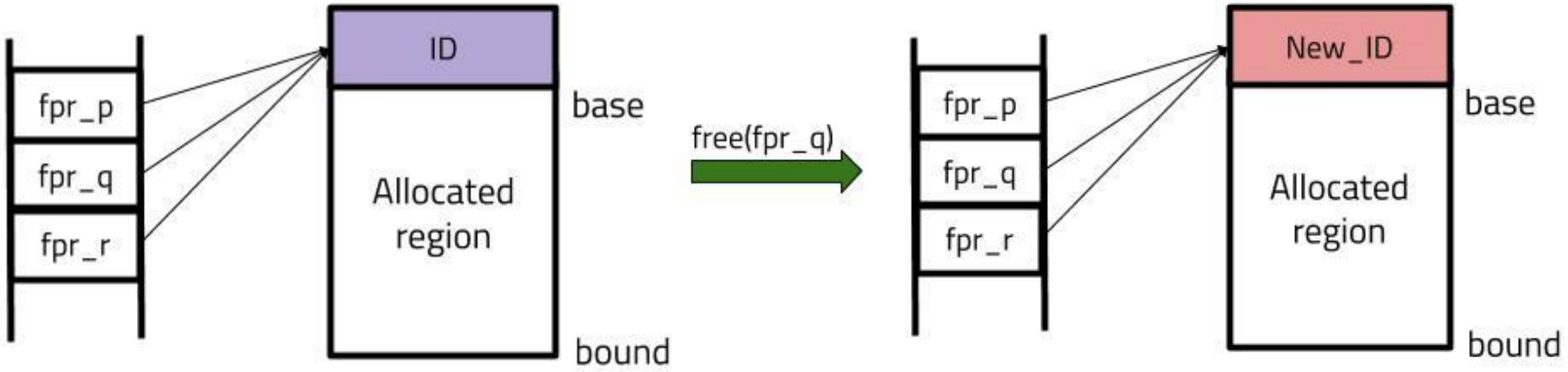
```
if( (pointer < base)  OR
(pointer + access_size) >= bound ) {
        trap;
}
```

Temporal Safety

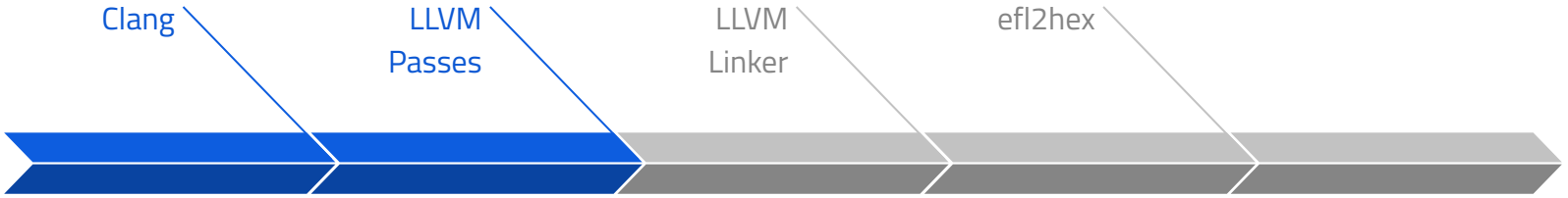
```
if( (base == NULL)  OR
( id_hash != hash (Memory[base-8]) ) {
        trap;
}
```



Preventing Temporal Attacks on Heap



Methodology



C-program

```
char buf[10] = "" ;
...
buf[10] = 'A' ;
```

Generated IR

```
%2 = alloca [10 x i8], align 1
%3 = getelementptr
        inbounds [10 x i8],
        [10 x i8]* %2, i64 0, i64 5

...
store i8 65, i8* %3, align 1
```

Modified IR

```
%2 = alloca [10 x i8], align 1
 ...
%fpr = call i128 @craft(i32
        %pti, i32 %stack_cookie_32,
        i32 %absolute_bnd,
        i32 %stack_hash)
...
call void @llvm.RISCV.validate( i64
        %fpr_hi, i64 %fpr_low)
...
store i8 65, i8* %ptrs, align 1
```

Executable

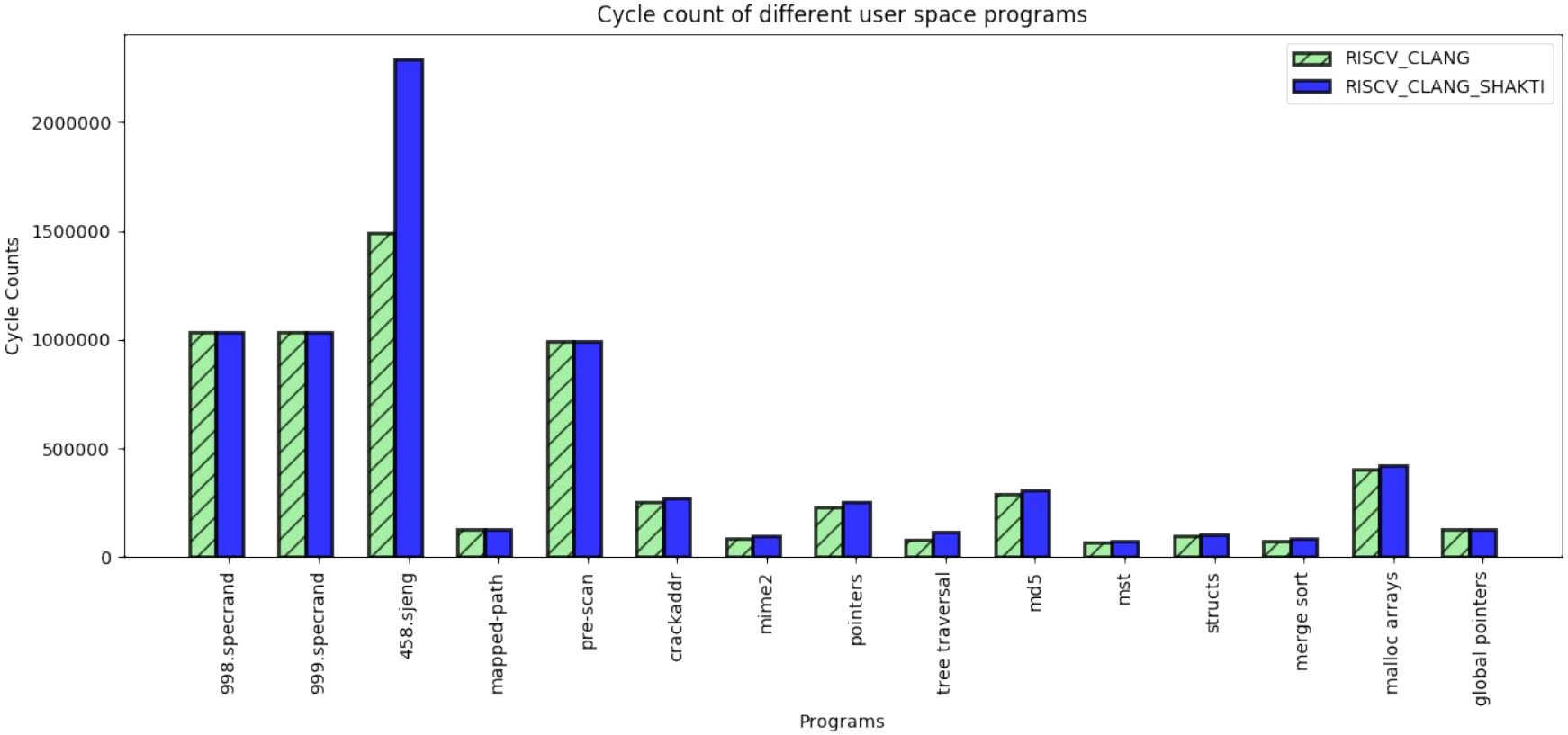
```
addi    sp,sp,-80
sd      ra,72(sp)
...
addiw   a1,a1,10
lui     a2,0x1002
addi    a2,a2,1360 #
        1002550
<craft>
...
srli    a0,a0,0x20
add     a0,a1,a0
val     a0,a2
...
sd      a0,-40(s0)
```

Hardware

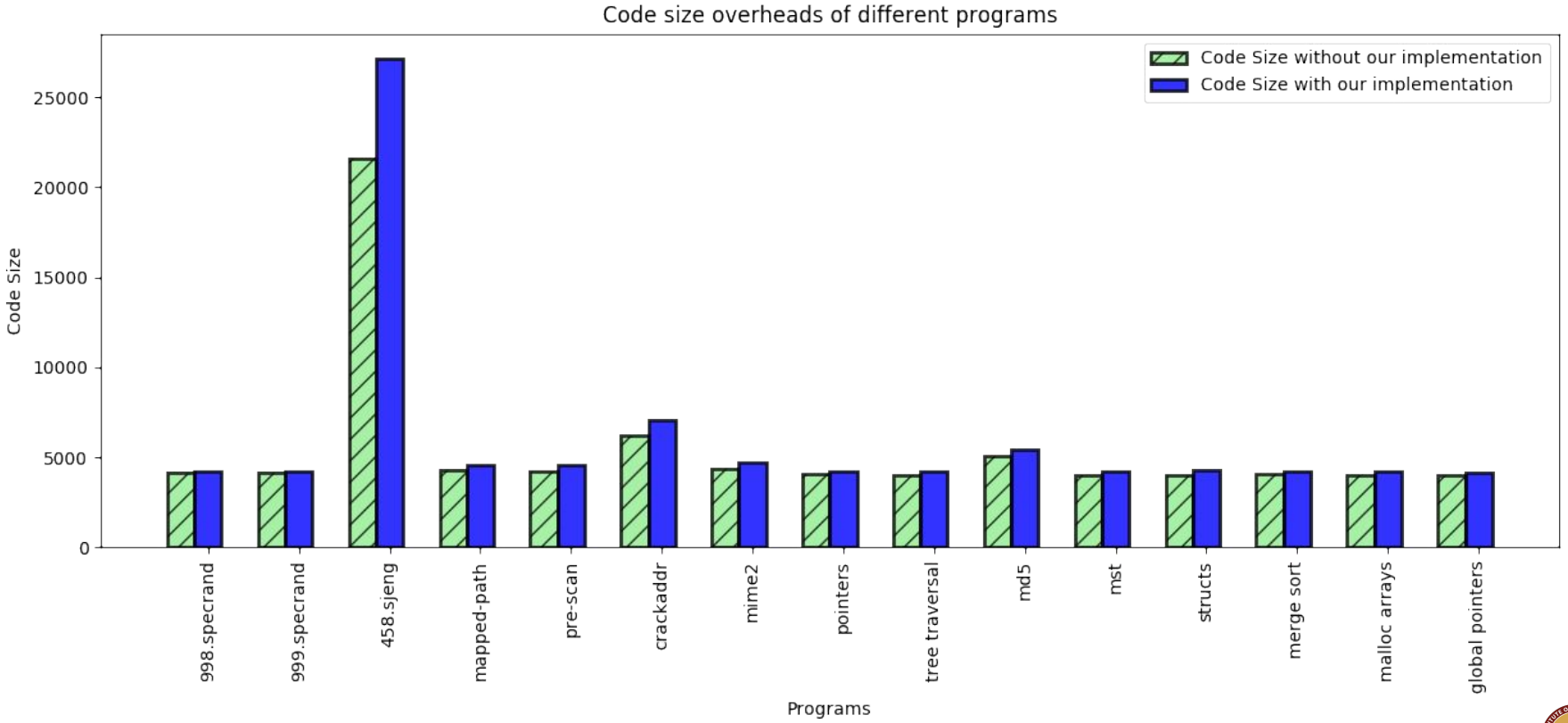
```
fb010113
04113423
...
00a5859b
01002637
55060613

...
02055513
00a58533
0006052b
...
fca43c23
```


SHAKTI

Results


Cycle count of different user space programs
RISCV_CLANG
RISCV_CLANG_SHAKTI
Cycle Counts
2000000
1500000
1000000
500000
0
998.specrand
999.specrand
458.sjeng
mapped-path
pre-scan
crackaddr
mime2
pointers
tree traversal
md5
mst
structs
merge sort
malloc arrays
global pointers
Programs

Results


Code size overheads of different programs
Code Size without our implementation
Code Size with our implementation
Code Size
0
5000
10000
15000
20000
25000
998.specrand
999.specrand
458.sjeng
mapped-path
pre-scan
crackaddr
mime2
pointers
tree traversal
md5
mst
structs
merge sort
malloc arrays
global pointers
Programs

Do you want to use an unsafe language
for your critical applications?

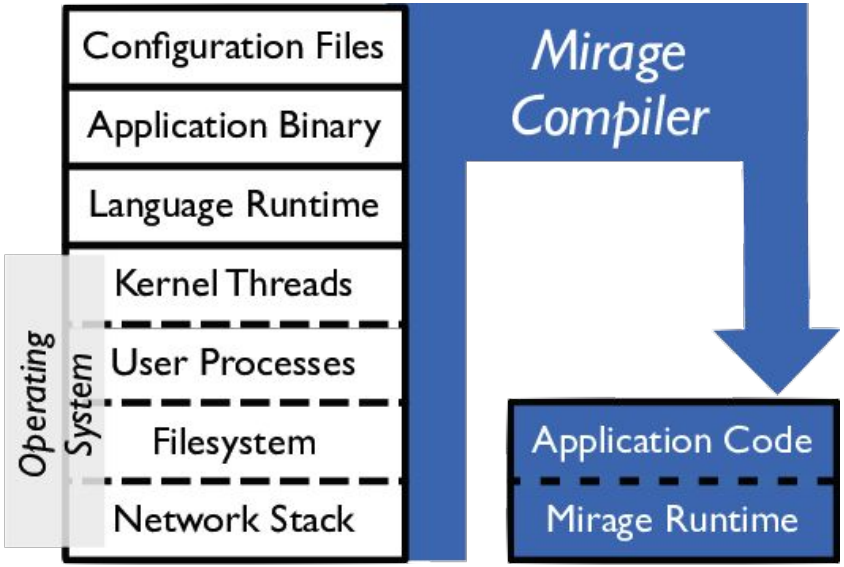
OCaml

Features of OCaml

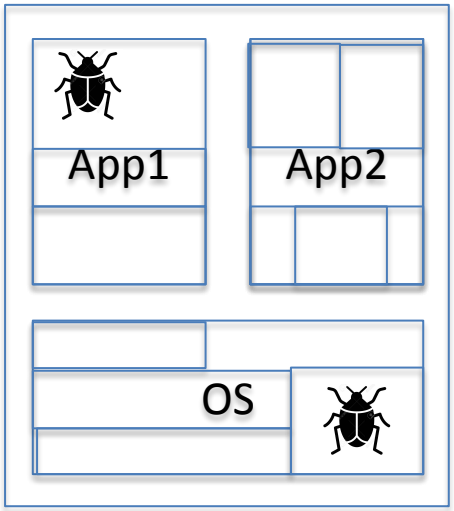
- Powerful type system
  - Parametric polymorphism
  - **Static type inference**
- User-defined algebraic data types and pattern-matching
  - Combination of records and sums
- Foreign function interface
  - To interoperate with C code when necessary
- Fast native backend suitable for embedded systems
- **Automatic memory management**
  - Incremental garbage collection
- **Aptness to symbolic computation**
  - Missing branches are detected and reported
- Performance is ~1.2X to ~1.5X slower than C

Mirage OS

- It takes only parts of OS and links it to the final executable
  - Lesser attack surface
  - Networking libraries
- Docker for MAC and Windows uses mirage libraries internally
- **Ported majority of Mirage OS to RISC-V and is open sourced**
  - https://github.com/mirage-shakti-iitm/

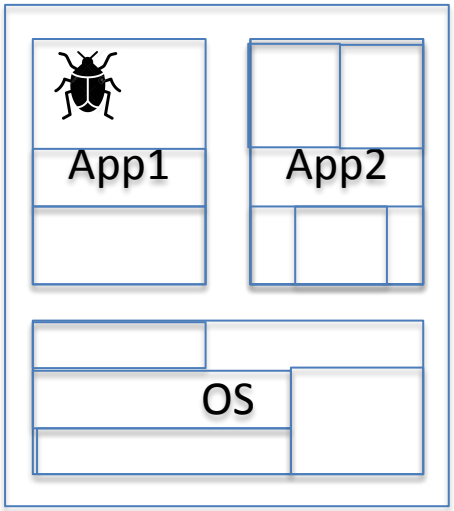


Threat Surface

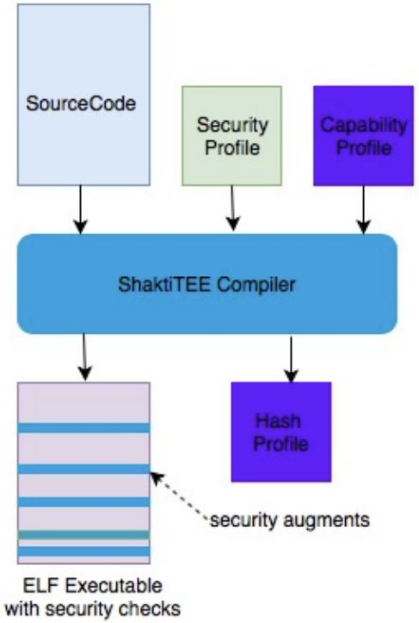


Shakti Trusted Execution Environment

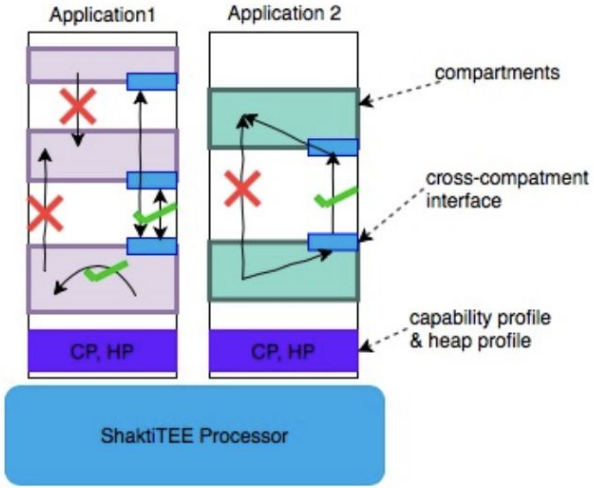
Threat Surface with Compartments



Compartmentalization

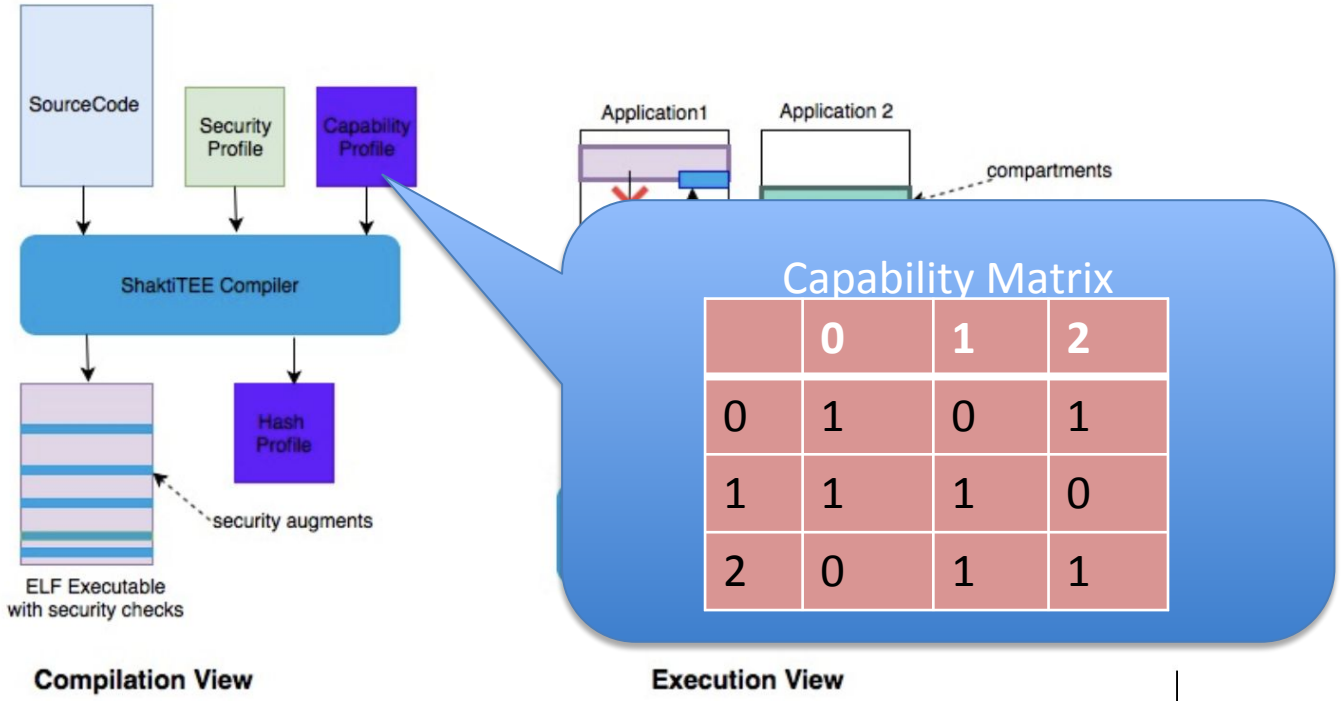


Compilation View

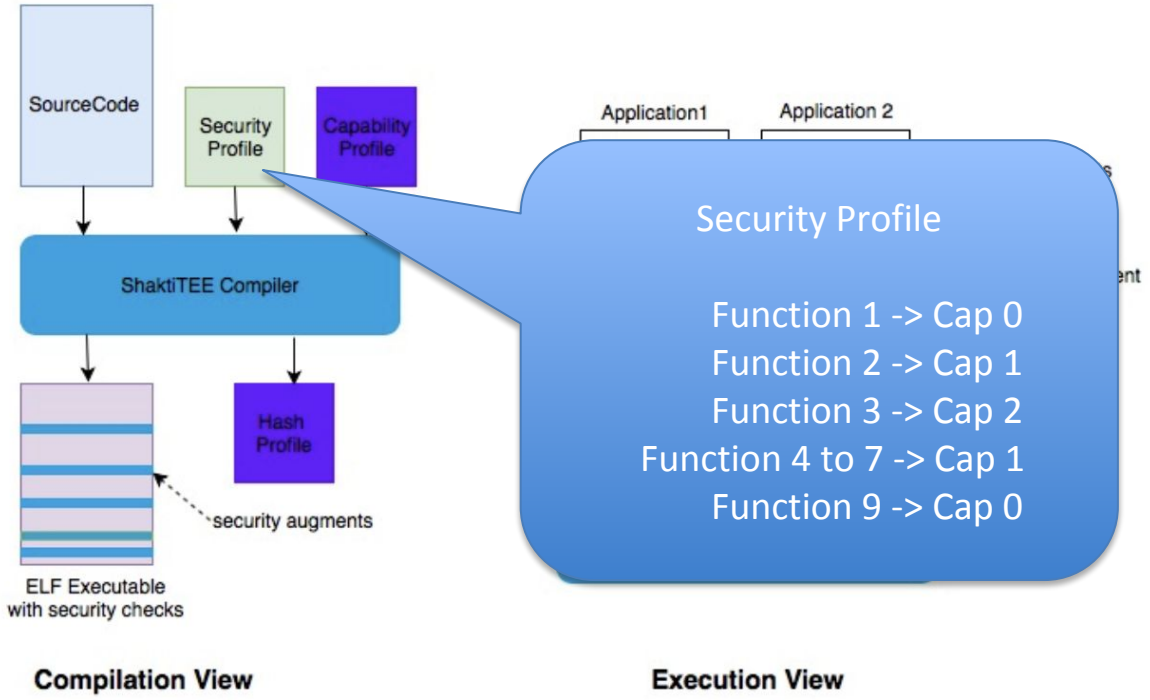


Execution View

Compartmentalization



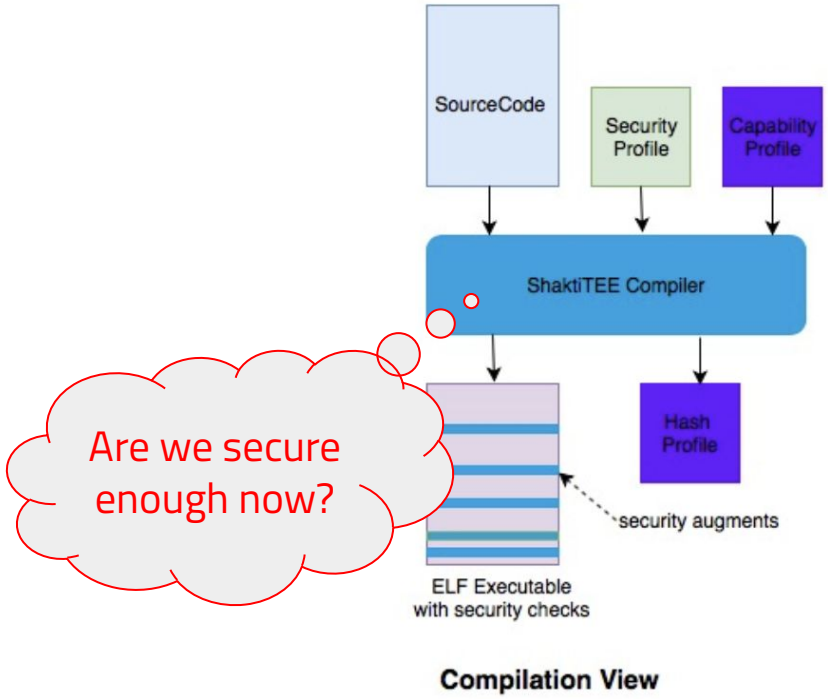
Compartmentalization



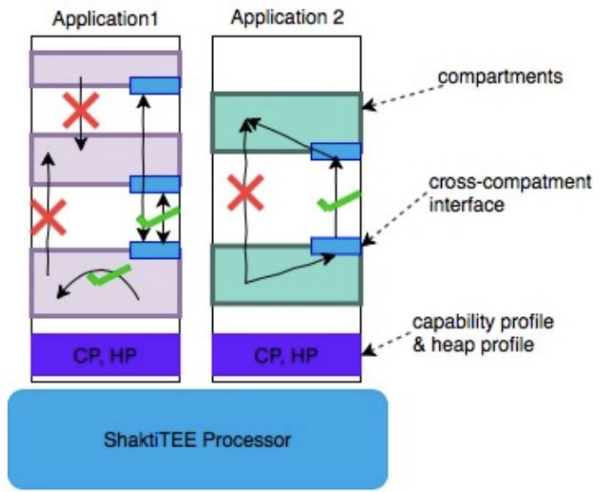
New instruction

- New instruction for assigning and checking capability
- Assembly: **checkcap** imm12
  - Checks if the current compartment_id == imm12, and if not equal raises an exception
  - Uses the custom opcode space in RISC-V
- Inserted at the beginning of every function

Compartmentalization



Compilation View



Execution View

Vulnerabilities due to Micro-architecture

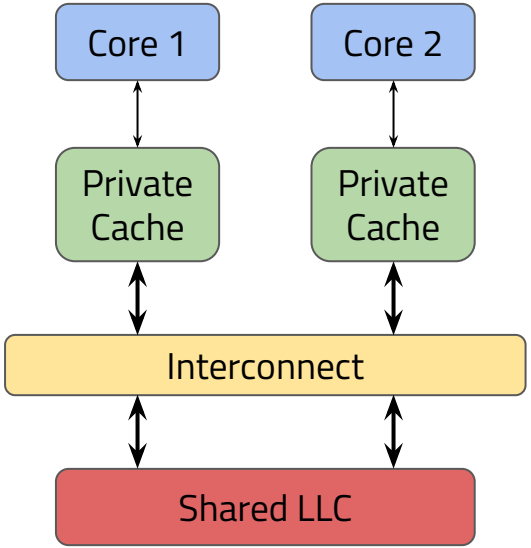
2

Meltdown Safe Memory Protection

SHAKTI

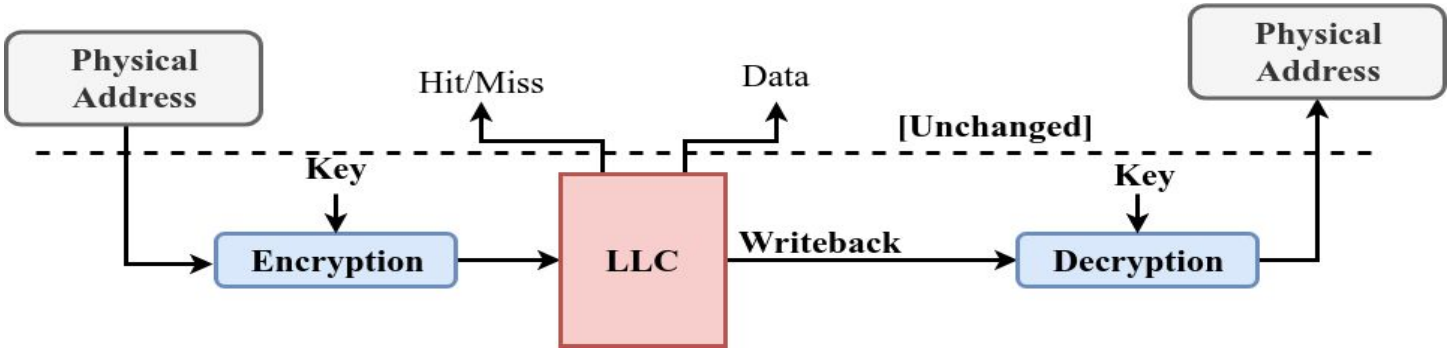
Cache Timing Side Channel Attacks

- Last Level Caches(LLCs) have been target for most of these attacks which is shared across all the cores.
- A Attacker Programs(AP) learns details about Victim Program(VP) through it's interference in LLCs
- The Interference in LLCs causes timing difference in accessing attacker program's cache data.
- Broadly the interference can be categorised as two types:
  - AP accesses VP's cached data - lesser access time
  - AP accesses VP evicted data - higher access time
- Solutions:
  - Interference can be blocked: Partitioning the cache
  - Interference can be minimized: Randomizing the cache mappings.



Solution

- Addresses to LLCs are encrypted
- Advantageous to use low-latency block ciphers like PRINCE and Feistel
- Keep changing the encryption key periodically
  - Borrow ideas like gradual remapping (CEASER @MICRO'18)



Vulnerabilities in Hardware

3 Power Attack Secure Microprocessor

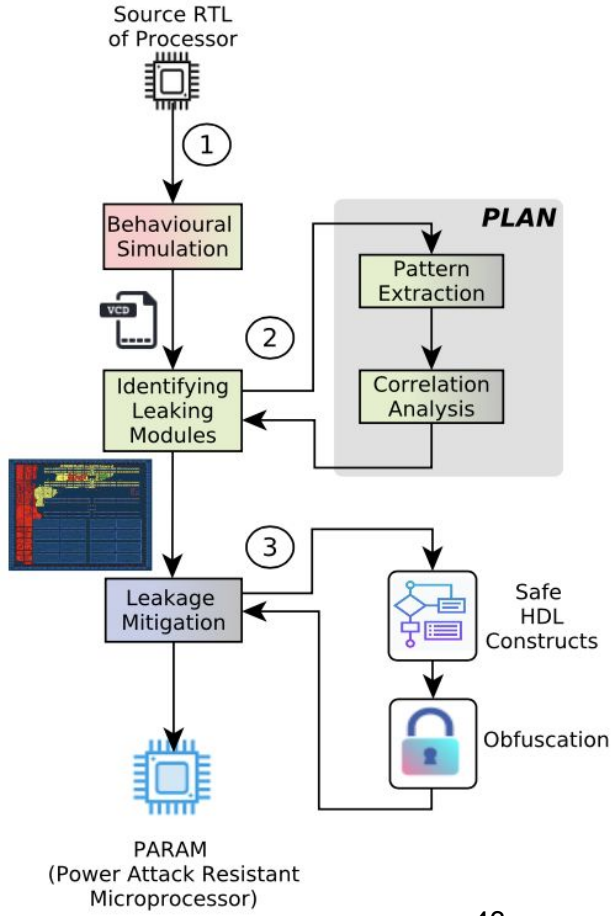
PARAM: Power-Attack Resistant Microprocessor

Muhammad Arsath, Vinod Ganesan, Rahul Bodduna and Chester Rebeiro

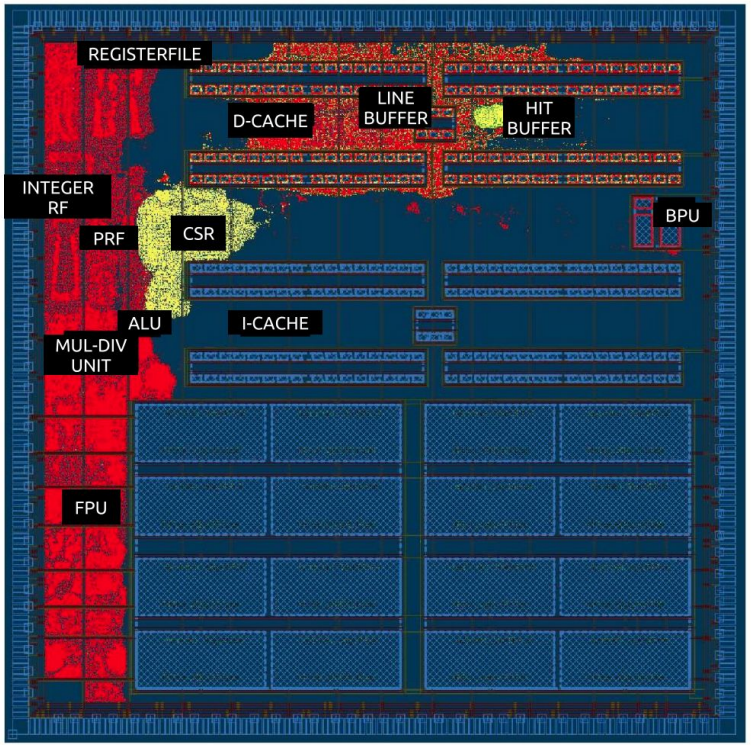
SHAKTI

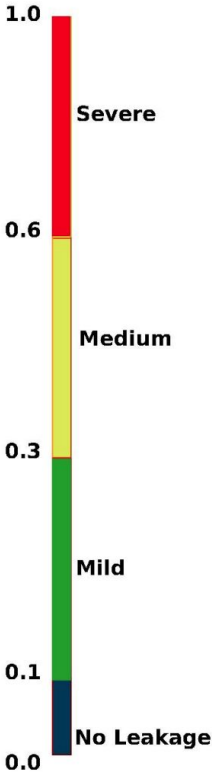
Steps Involved

1. Simulate RTL of the processor
2. Identify modules that leak the most
3. Apply appropriate countermeasures



Leakage Analysis on Shakti C-Class





For an implementation of AES in software

	Module	Leak Count	SVF	#Leaking Signals
Memory Hierarchy	Data Cache	165	0.99	85
	Hit Buffer	35	0.35	
	Line Buffer	2	1.0	
Registers	Register File	3	1.0	24
	PRF	25	1.0	
	CSR	13	0.64	
Functional Units	ALU	38	0.99	21
	FPU	3	1.0	
	Mul-Div Unit	2	1.0	
Interstage Buffers	IF-ID	0	0	5
	ID-EXE	11	1.0	
	EXE-MEM	6	1.0	
	MEM-WB	4	1.0	
Instruction Fetch	BPU	2	0.95	1
Instruction Memory	Instruction Cache	0	0.0	0
	Line Buffer	0	0.0	
TLB	Instruction TLB	0	0.0	0
	Data TLB	0	0.10	

SHAKTI

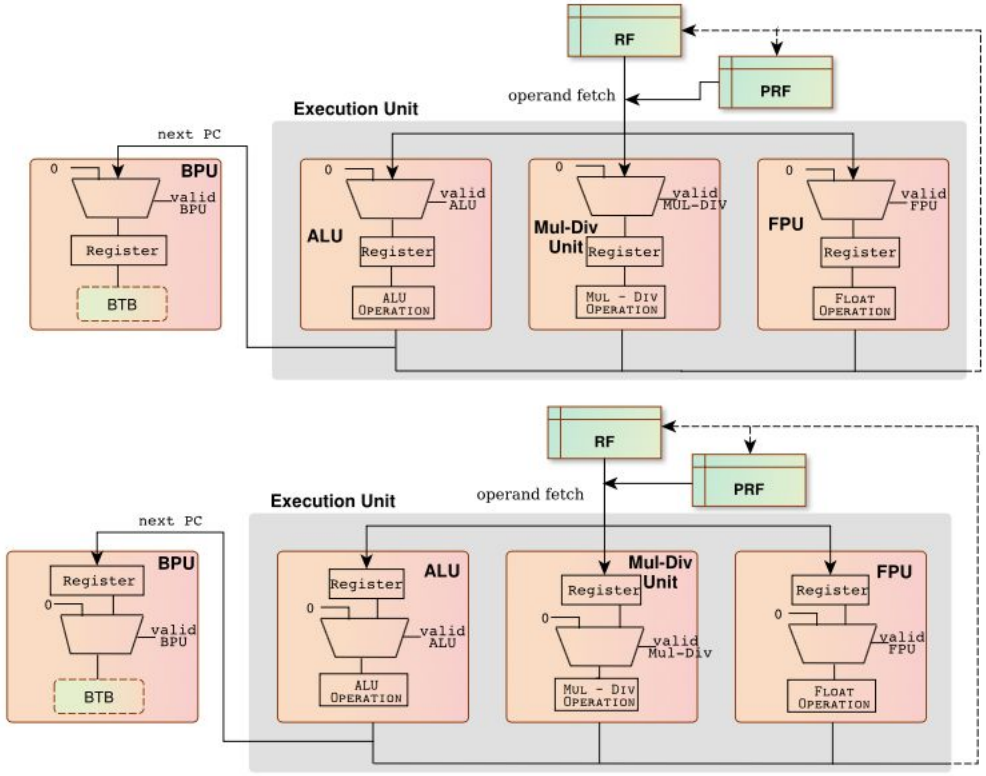
Leakage Analysis on Shakti C-Class

Expected Design:

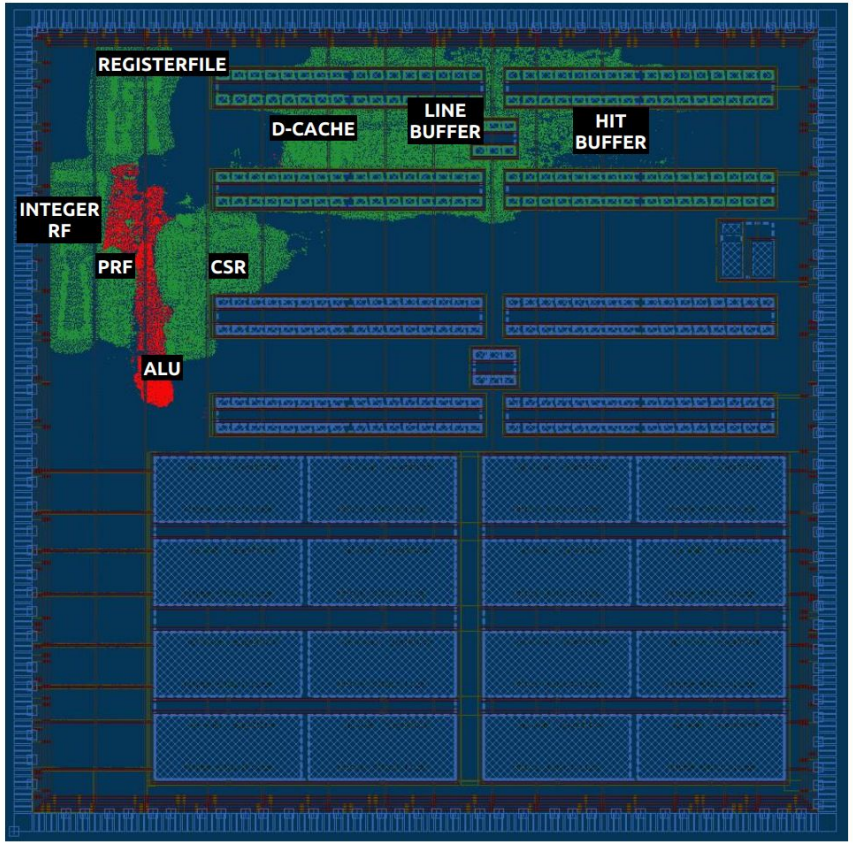
- Operands are sent only to relevant execution units

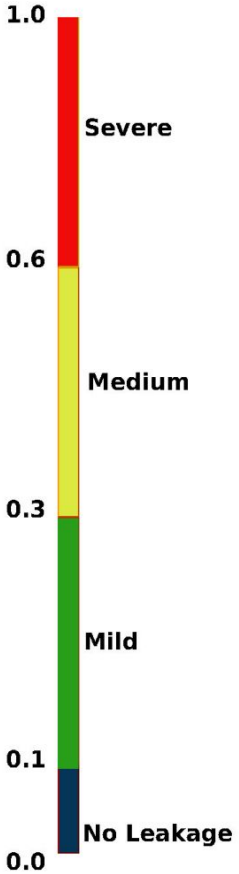
Actual Design:

- Operands are sent to all execution units



Leakage Mitigation





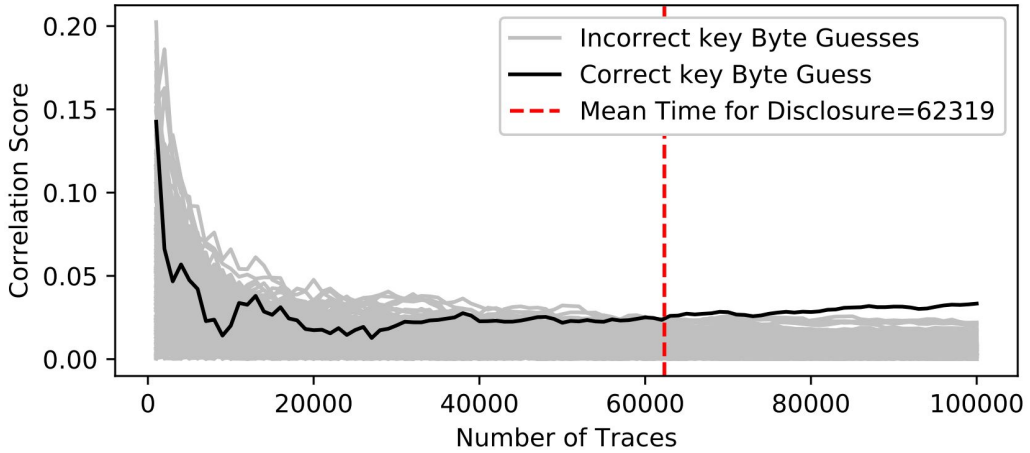
- Lightweight encryption on all registers and cache
  - 4-round Feistel structure
- Cache tag and set indices are remapped using a key
- Architectural changes to prevent flow of unused data to execution units
- Area overhead of 43%
- Frequency reduction of 33%
  - Can be reduced to < 5%

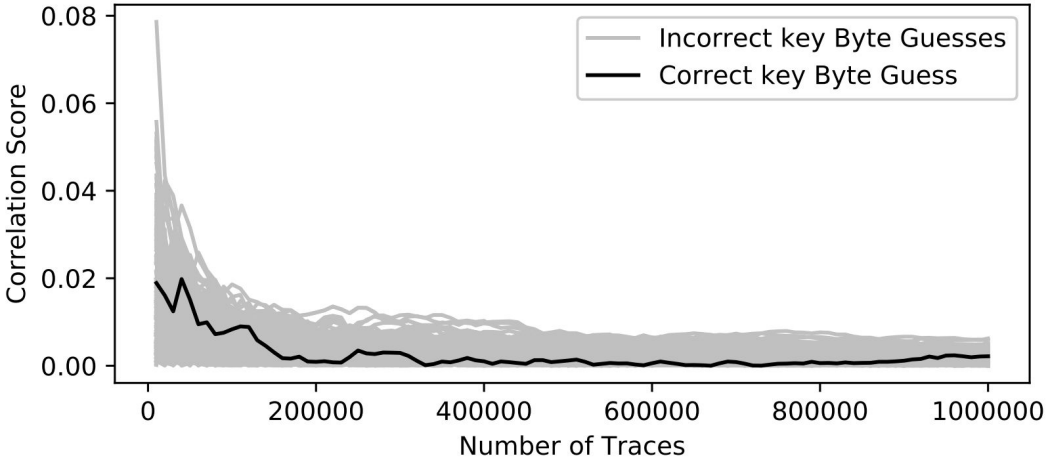
SHAKTI

Security Analysis

Shakti-C

- Leaks information after 62319 power traces





PARAM

- No leakage even after 1 million traces

The path ahead

- Integrate Countermeasures
- Formal Verification
  - Hardware
    - COQ/Kami
  - Software
    - Use COQ or F* for specifying highly sensitive code
    - Use existing libraries from HACL*
- SoC Security
- Single Address Space OS
- AI based vulnerability assessment

THANK YOU

REFERENCES

[1] Intel Corporation, “Intel MPX Explained.” https://intel-mpx.github.io/design/

[2] Devietti, Joe, et al. "Hardbound: architectural support for spatial safety of the C programming language." *ACM SIGARCH Computer Architecture News*. Vol. 36. No. 1. ACM, 2008.

[3] Kwon, Albert, et al. "Low-fat pointers: compact encoding and efficient gate-level implementation of fat pointers for spatial safety and capability-based security." *Proceedings of the 2013 ACM SIGSAC conference on Computer & Communications Security (CCS)*. ACM, 2013.

[4] Nagarakatte, Santosh, et al. “Watchdog: Hardware for safe and secure manual memory management and full memory safety.”, *Proceedings of the 2012 International Symposium on Computer Architecture (ISCA). ACM, 2012.*

[5] Nagarakatte, Santosh, et al. "WatchdogLite: Hardware-accelerated compiler-based pointer checking." *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. ACM, 2014.

[6] Menon, Arjun, et al. "Shakti-T: A RISC-V processor with light weight security extensions." In *Proceedings of the Hardware and Architectural Support for Security and Privacy (HASP)*, p. 2. ACM, 2017.

SHAKTI