A Code Compression Method with Encryption and Integrity Checking

CryptArchi 2007

Eduardo WANDERLEY

CEFET-RN, Brazil

LESTER Lab. UBS, France

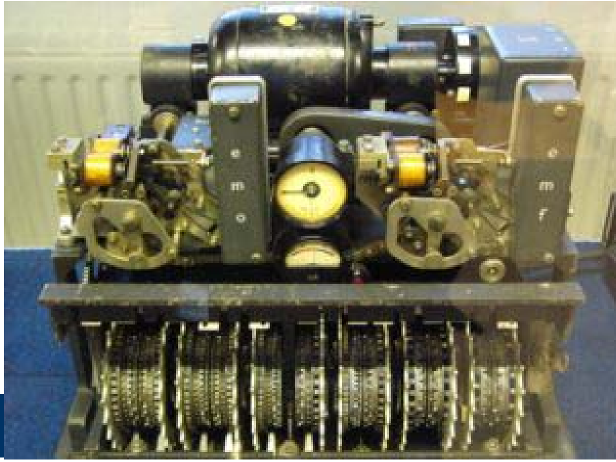
IcTER Project

Plan

- Introduction
- Code Compression + Cryptography
- The IBCEI
- Results
- Conclusions

Introduction

- Security in Embedded Systems
  - Becoming more and more important
    - Mobile Terminal users need to supply personal information in order to access data sensitive services
    - Software companies need to protect their products
    - Threats in many data exchange interfaces
      - Memory – Processor interface => Probing



Introduction

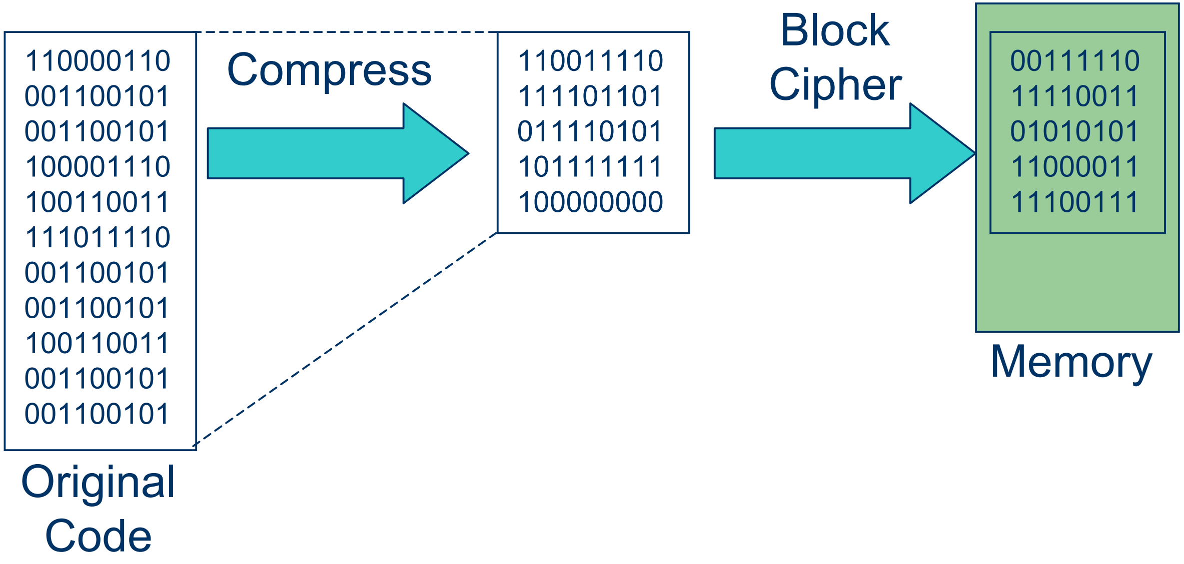
- Cryptography
  - Confidentiality
    - only the entities involved in the execution or the communication can have access to the data
  - Integrity
    - The message/program must not be altered during the transfer or execution.
  - Implications
    - **Deciphering overhead**
    - **Area Overhead**
      (e.g. keep signatures)

Introduction

- Code Compression
  - Compress the code and store it into the memory
  - On-the-fly decompression during execution
  - Memory-Processor traffic
    - More Instructions/Memory access
  - Implications:
    - Decompression overhead
  - Advantages
    - Performance + Energy Consumption + Memory Area

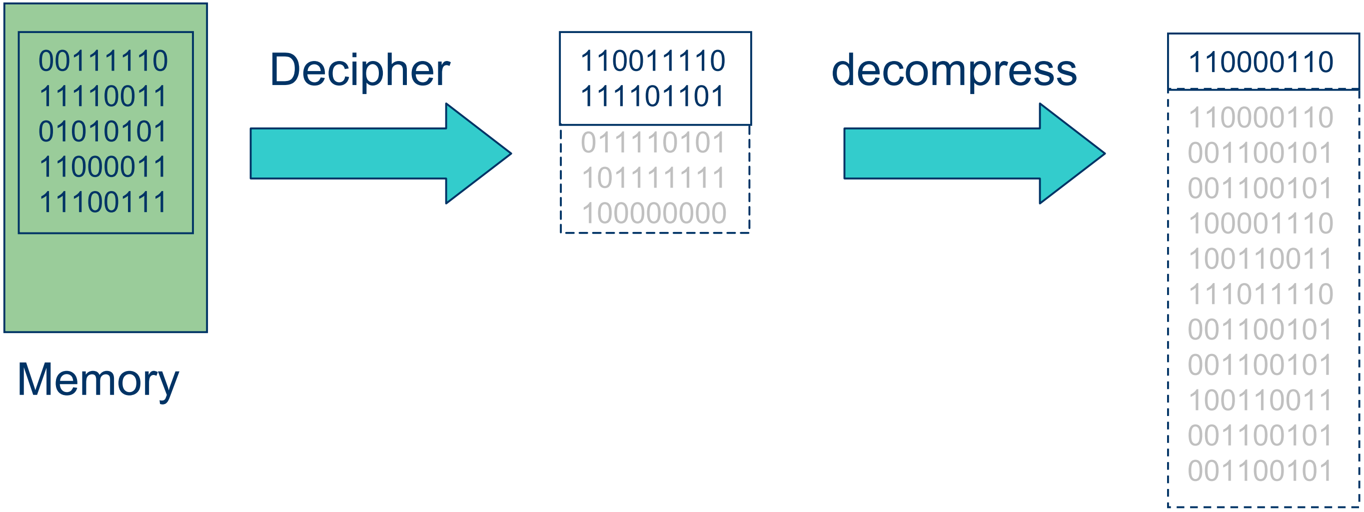
Code Compression + Cryptography

- General Idea
  - Minimizing the deciphering overhead by utilizing Code Compression



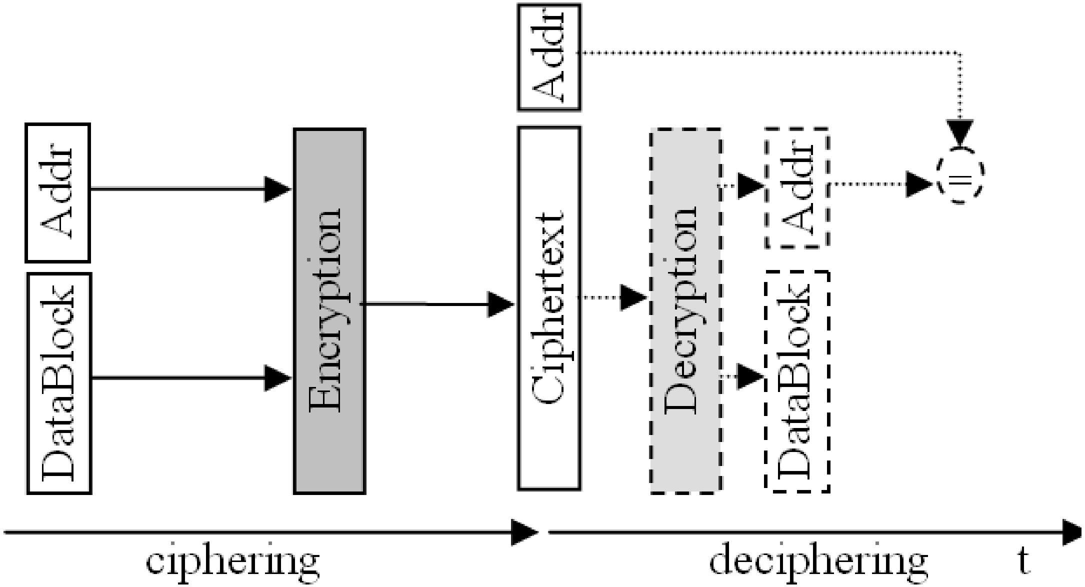
Code Compression + Cryptography

- General Idea
  - Minimizing the deciphering overhead by utilizing Code Compression



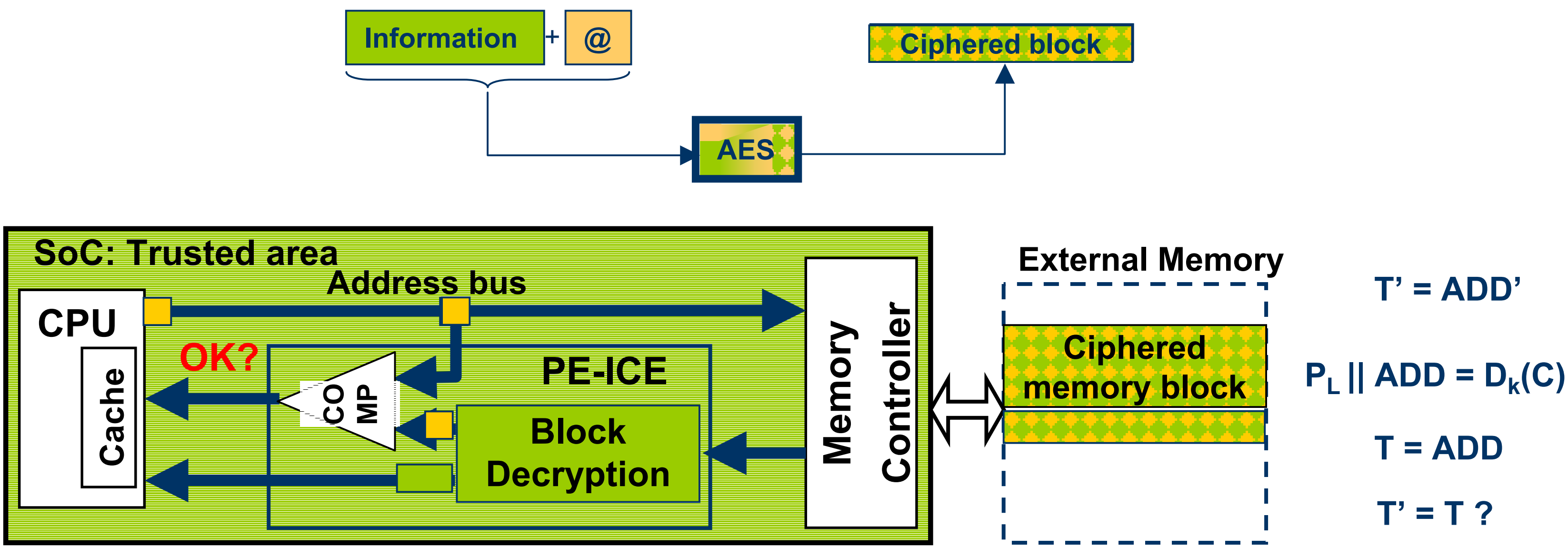
Code Compression + Cryptography

- Confidentiality + Integrity
  - What about Integrity?
  - PE-ICE solution



Code Compression + Cryptography

- IBC + Confidentiality + Integrity
  - Compression IBC
  - PE-ICE (AES based) or other...

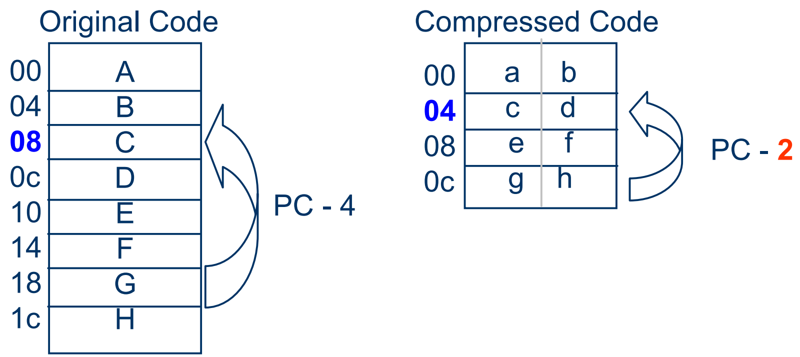


Code Compression + Cryptography

- Problems with PE-ICE:
  - Memory footprint overhead (to keep the TAG)
  - More memory accesses to retrieve the Information + tag (address)
- Code Compression:
  - Reduce the memory footprint
  - Reduce memory accesses to retrieve the Information

The IBCEI

- Dictionary Based Code Compression
  - The original instruction will be substituted for indexes into a dictionary
- The compressed addresses will not be the same original addresses (branch targets)
  - An Address Translation Table – ATT – will be used.

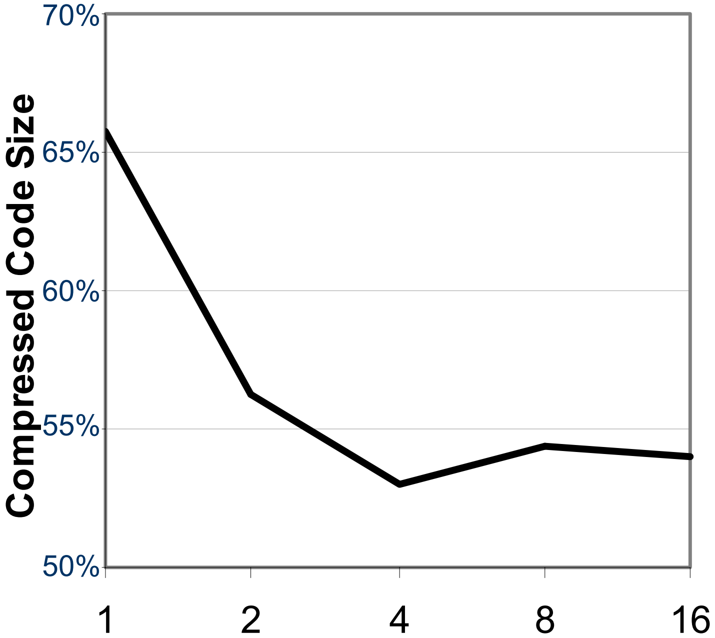


The IBCEI

- Utilize ***n*** dictionaries of different sizes
  - The smaller is the dictionary, the smaller is the index.
    - Smaller dictionary: holds the instructions that appear the most in the code
    - Bigger dictionary: holds the instructions that rarely appear in the code
  - Drawback: how can I guess which index belong to which dictionary?
    - Solution: using a prefix to identify the dictionary
      - The bigger is ***n***, the bigger is the prefix

The IBCEI

- Utilize ***n*** dictionaries of different sizes
  - How many Dictionaries should I use?



The IBCEI

- Example

instruction	occur
nop	32
add $1,$5,$6	20
sub $3,$4,$7	12
call 1032	10
add $1,$1,$6	8
beq $2,$3,10	8
sub $3,$2,$7	6
beq $5,$3,1032	6
add $1,$0,$6	4
jal $32	4
ret	4
mul $1,$2,$3	3
div $1,$2,$3	3
sll $1,$2,$3	3
sethi $7	1
ld $8,4($5)	1
addi $1,$1,1	1
jr $1	1
st $8,4($5)	1
total	**128**

Prefix = 00_2

	nop

Prefix = 01_2

0	add $1,$5,$6
1	sub $3,$4,$7

Prefix = 10_2

00	call 1032
01	add $1,$1,$6
10	beq $2,$3,10
11	sub $3,$2,$7

Prefix = 11_2

0000	beq $5,$3,1032
0001	add $1,$0,$6
0010	jal $32
0011	ret
0100	mul $1,$2,$3
0101	div $1,$2,$3
0110	sll $1,$2,$3
0111	sethi $7
1000	ld $8,4($5)
1001	addi $1,$1,1
1010	jr $1
1011	st $8,4($5)
1100	

Compressed Code Size = 32x2 + 32x(2+1) + 32x(2+2) + 32x(2+4)
= **480** bits

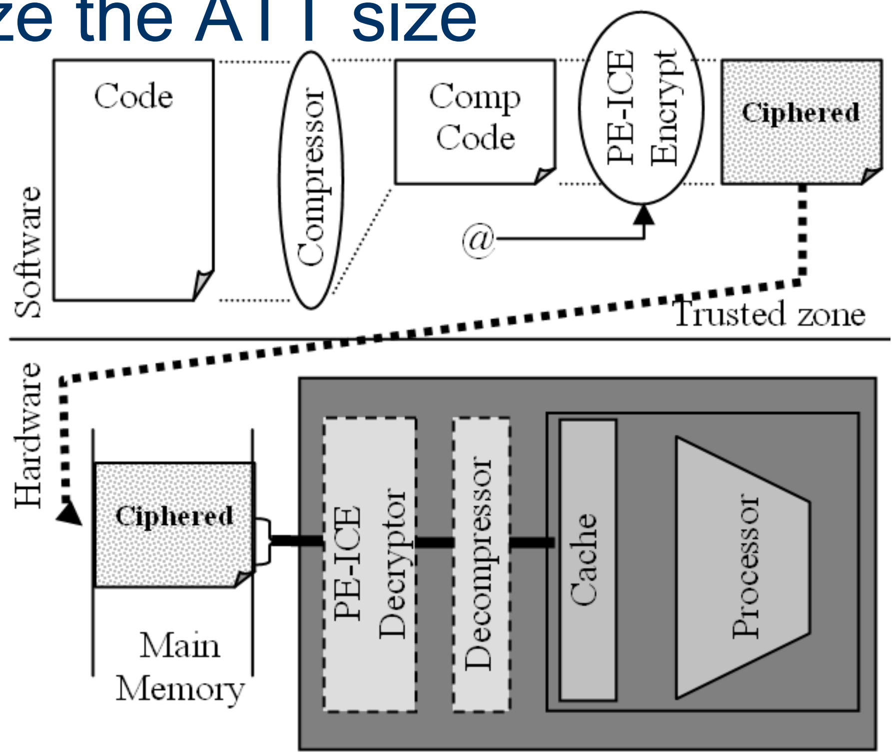
Compressed Code Size with 1 dict = 608
Compressed Code Size with 2 dict = 512
$\boxed{\text{Compressed Code Size with 4 dict = 480}}$
Compressed Code Size with 8 dict = 494

The IBCEI

- Hiding the decompression latency and minimize the ATT size
  - Decompressor between the Main Memory and the Cache
  - Each cache miss will invoke the decompressor. The ATT will keep only the cache line addresses correspondences.

The IBCEI

- Hiding the decompression latency and minimize the ATT size

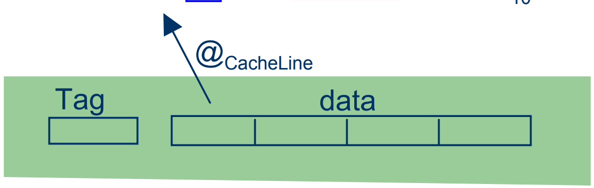


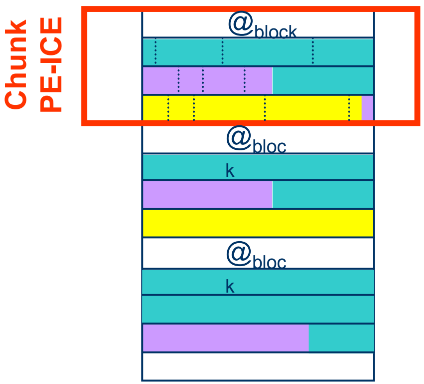
The IBCEI

- PE-ICE + Code Compression

Address Translation Table

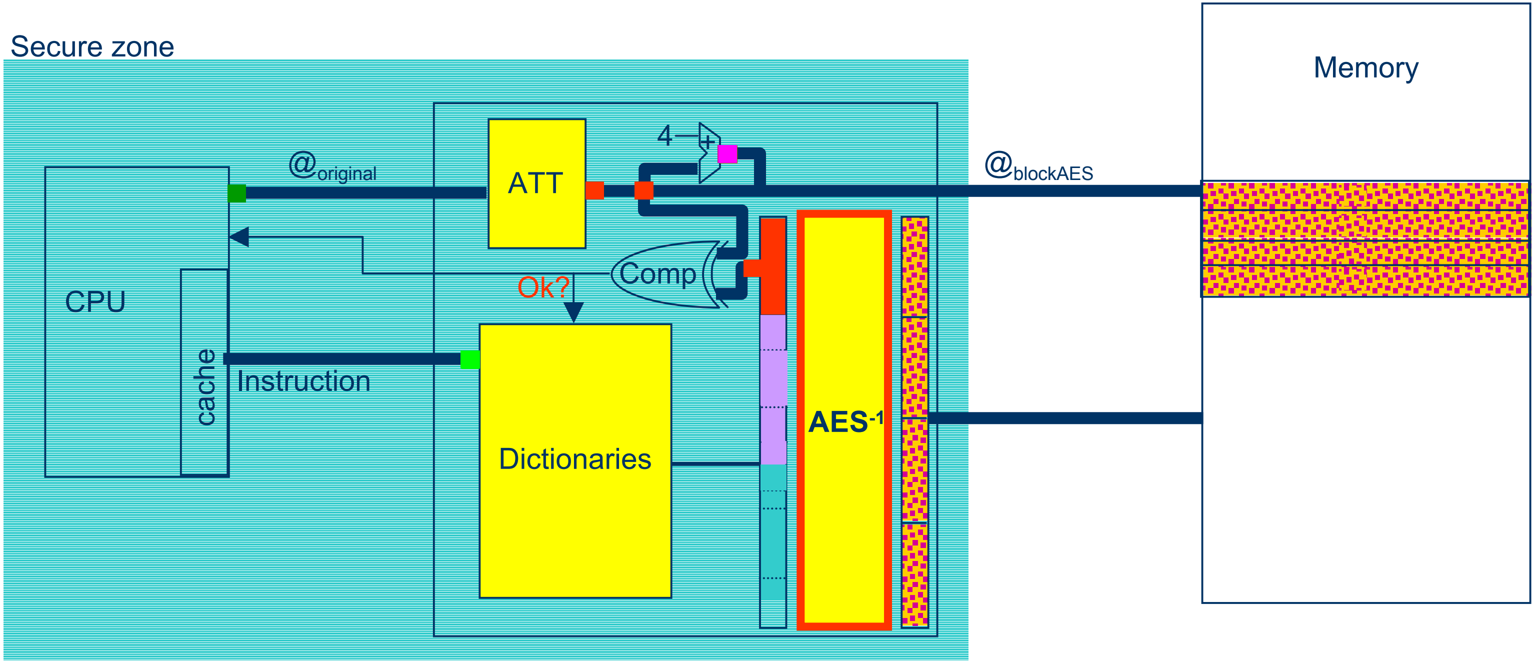
$@_{original}$	$@_{ChunkPE\text{-}ICE}$	offset
00000000	00000000	32_{10}
00000010	00000000	89_{10}
00000020	00000000	99_{10}
00000030	00000010	32_{10}
00000040	00000010	89_{10}
00000050	00000010	96_{10}
00000060	00000020	32_{10}
00000070	00000020	125_{10}
...	...	...
000001f0	00000048	19_{10}





The IBCEI

- The execution view

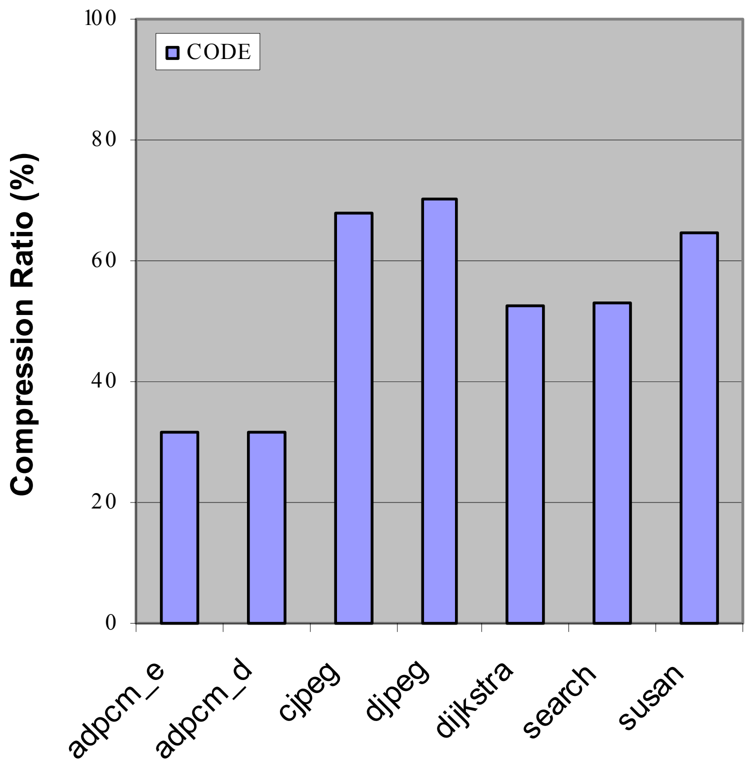


Results

- Leon (SPARC v8) processor
- Mediabench and MiBench benchmarks suites
- LECCS – GCC Leon cross compiler with –O2
- Simulator based

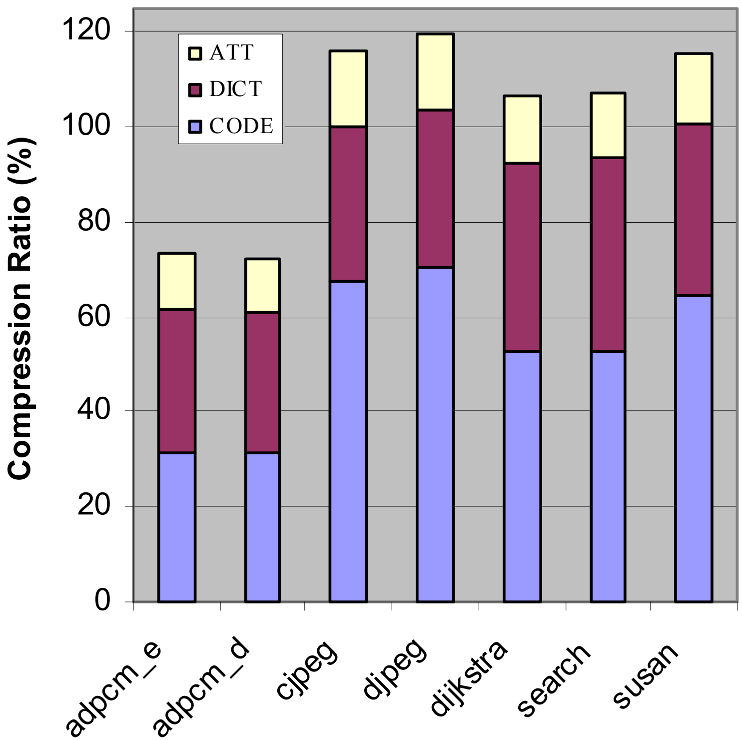
Results

- External Memory footprint



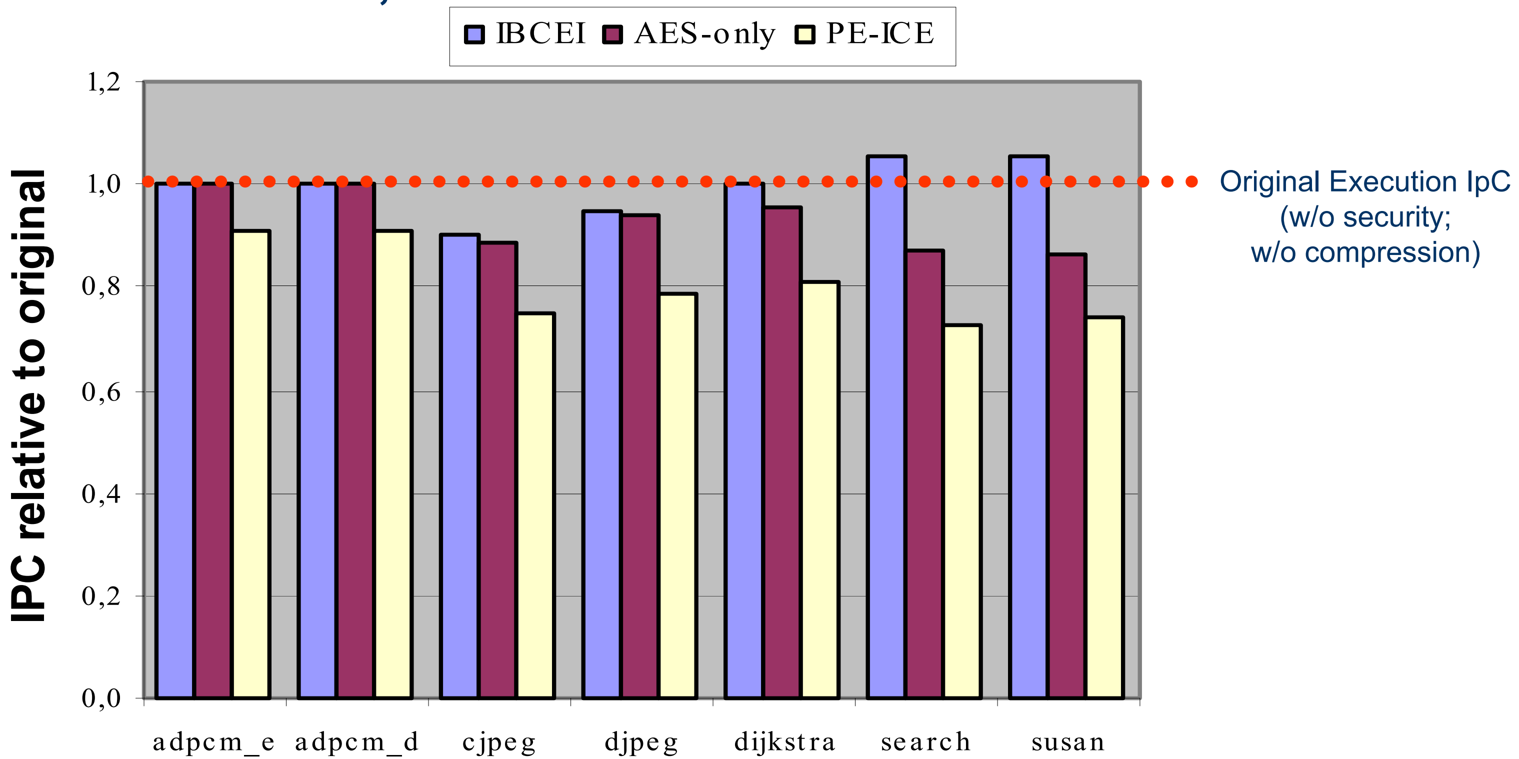
Results

- Area



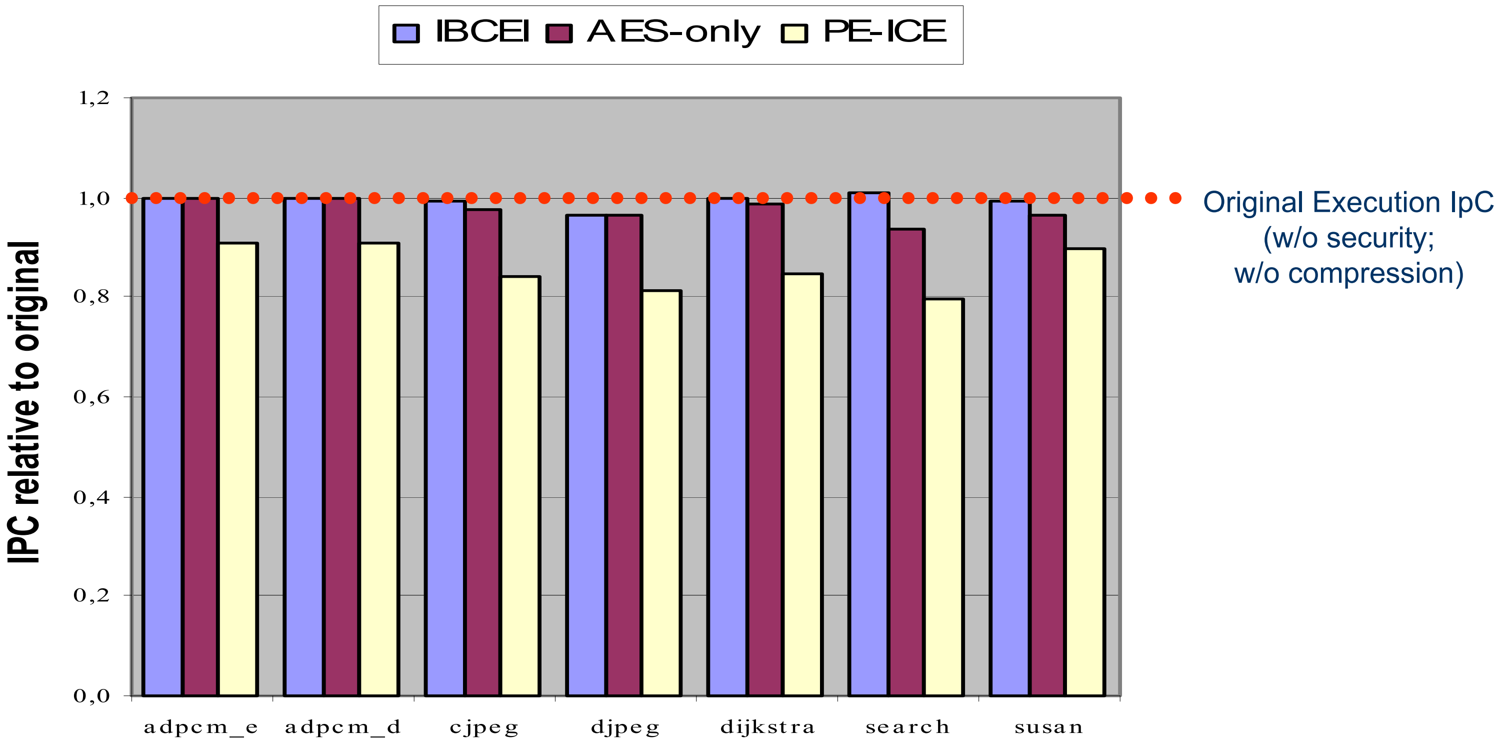
Results

- Performance, cache 1k



Results

- Performance, cache 8k



Conclusions

- IBCEI
  - Performance:
    - Less memory accesses than PE-ICE
    - AES block is less utilized too
    - Decompression Overhead
  - Area:
    - Smaller memory footprint than PE-ICE
    - Decompressor Overhead (minimum)
  - Security:
    - Same level of PE-ICE security
    - Compression minimizes the redundancy before ciphering
    - Compression introduces a novel statistic component to the eavesdropper

Conclusions

- Future Work
  - Dynamic Dictionaries
  - Parameters impact
  - RW Data compression

Thank you