



The Feasibility of Memory Encryption and Authentication

Donald Owen, Jr.

Laboratory for Computer Architecture
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78705, USA

FastPath 2013: April 21, 2013



Outline

Introduction

Motivation

Background

Solution Characterization

Software

Hardware

Results

Conclusion



Motivation

Digital data is increasingly put on mobile devices and remote servers. This data needs to be protected.

Security Problems

Health records

SSN

Private communication (emails)

Corporate documents

Crypto keys

... more

Security Solutions?

Disk encryption

Strong passwords

Hardened software

A major component left unprotected: DRAM.

Existing Protections Not Enough



Attacks abound

- ▶ Passive
 - ▶ Bus Sniffing*
- ▶ Active
 - ▶ Spoofing
 - ▶ Splicing
 - ▶ Replay
 - ▶ Cold boot*



Existing Protections Not Enough

Xbox™ hacked by Andrew “bunnie” Huang (MIT) using a bus sniffer to read a secret key / decryption code.



Image Huang [1]

Existing Protections Not Enough



The “cold boot” attack, and variants thereof, exploit DRAM remanence to extract data from RAM.

1. Interrupt power of a running system
2. Reboot into custom OS
3. Dump contents of DRAM to permanent storage
4. Mine dumped data for keys, files, fragments
5. Use recovered data to exploit, for example, encrypted disk



Outline

Introduction

Motivation

Background

Solution Characterization

Software

Hardware

Results

Conclusion



Memory Encryption & Authentication



What do we want?

C: Confidentiality

I: Integrity

A: Authentication

How do we get it?

Encryption

Hash/Tag/Signature

Tag/Signature

In short, encrypt and tag data to RAM; decrypt and authenticate data from RAM.



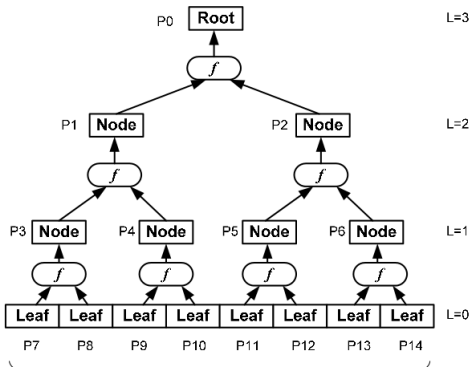
Memory Encryption & Authentication



- ▶ Approach 1: Encrypt and tag each cache line. Store tags on the chip. Verify on reading back from RAM.
 - ▶ Problem: Storage space.
- ▶ Approach 2: Use a tree structure! Store tags in DRAM with the root stored on the chip. Verify up the tree on reading back.
 - ▶ Problem: Speed.
 - ▶ Variants: Use a dedicated tag cache on the chip. Verify until you hit in the cache.



Memory Encryption & Authentication



Memory Space ($M = 8$)

f : authentication primitive
Leaf: data node

Px: Position x
L : Tree level

Image Elbaz et al. [2]

Galois Counter Mode

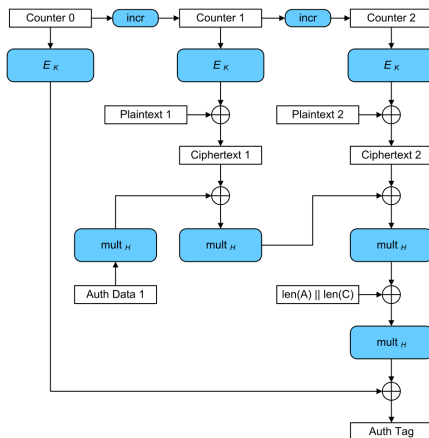


Image Wikipedia [3]

Outline

Introduction

Motivation

Background

Solution Characterization

Software

Hardware

Results

Conclusion

Solution Characterization



Previous work assumed the existence of hardware adequate to the task of encryption, decryption, tagging, and verifying fast enough to meet performance demands.

We evaluated how feasible those assumptions are under different implementation characteristics.


- ▶ Software
 - ▶ Pure C on x86
 - ▶ C + x86 Assembly
 - ▶ C + x86 Assembly + ISA Extensions
- ▶ RTL on FPGA
- ▶ RTL on synthesized ASIC



Experimental Setup



Table : Experimental Setup

Processor	Intel Core i7 2620M
OS	Fedora Linux 17 GNU/Linux 3.7 x86_64 
Compiler	GCC 4.7.2
FPGA Synthesis	Xilinx ISE v. 14.3 Kintex 7-325T
ASIC Synthesis	Synopsys Design Vision v. E2010-12 FreePDK 45 nm Library



Pure C Implementation



MiBench has an AES (Rijndael) benchmark. We modified this benchmark to suit the implementation requirements.

Modifications

- ▶ Convert AES-CBC to AES-GCM.
- ▶ Convert File I/O to in-memory operations.
- ▶ Profile at cache line sizes



Pure C Implementation



Table : Cycles per Byte Measurements for Pure C Implementation of AES-GCM

Buffer Size	Encrypt	Decrypt
32B	52.2	74.2
64B	37.8	50.4
128B	35.6	39.8
256B	28.3	35.8
512B	25.4	33.0



C + Assembly



We can do better!

- ▶ The same code in the pure C implementation has optional Assembly routines.
- ▶ OpenSSL uses Assembly optimizations.
- ▶ Modern x86 processors have ISA extensions for AES and GCM.



C + Assembly



Table : Cycles per Byte Measurements for Assembly-Optimized AES-GCM (64B Buffer)

Method	Encrypt	Decrypt
Pure C	37.8	50.4
C + Opt.	22	30
OpenSSL	~25	~25
ISA Extensions [4]	3.5	~3.5



RTL Module



Most previous work assumes the existence of hardware modules. We adapted an open-source AES-GCM module to be suitable for both FPGA and ASIC synthesis.

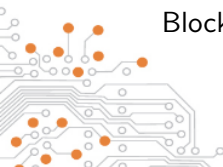


RTL Module Characteristics - FPGA

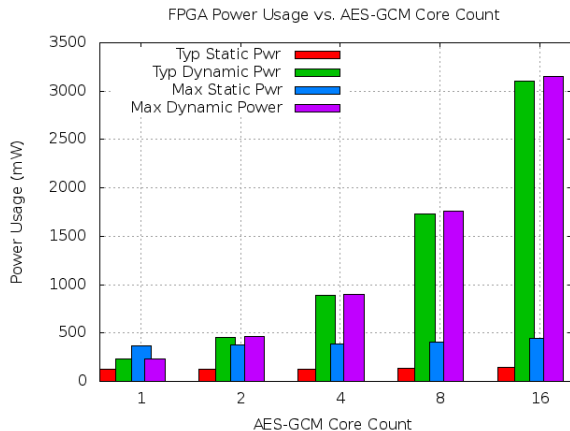


Table : Open-Source vs. Representative Commercial AES-GCM RTL Core on Kintex 7 FPGA

Metric	Open Source	Commercial
Startup	19 clocks	0 clocks
16B Enc/Dec	22 clocks	12 clocks
16B Tag(Hash)	17 clocks	12 clocks
64B Cache Line + Tag	123 clocks	60 clocks
Freq. Max	212 MHz	256 MHz
Logic Slices	~800	~1000
Block RAMs	8	12



FPGA RTL Synthesis Results



Linear Fit: $193\text{mW}/\text{instance}$.

RTL Module Characteristics - ASIC



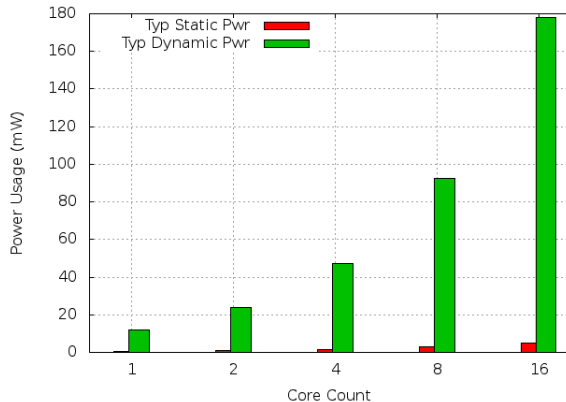
FreePDK 45 Implementation

- ▶ Freq Max: 250 MHz
- ▶ Area: 89k μm^2
- ▶ Power: 12 mW



ASIC RTL Synthesis Results

ASIC Typical Power Usage vs. AES-GCM Core Count



Linear Fit: $11.05\text{mW}/\text{instance}$.

Outline



Introduction

Motivation

Background

Solution Characterization

Software

Hardware

Results

Conclusion



Results - Summary



Table : Summary of Different Implementation Methods

	ASIC	FPGA	x86 C	x86 Assembly	x86 ISA Ext.
Clock (Hz)	225 M	200 M	2.7 G	2.7 G	2.7 G
$\frac{\text{Cycles}}{\text{Byte}}$	1.9	1.9	44	22	3.5
Throughput	936.6 Mbps	882.5 Mbps	490.9 Mbps	981.8 Mbps	6.17 Gbps
Typ. Power	11.05 mW	192.9 mW	~35 W	~35 W	~35 W
Typ. Area	74.1k μm^2	-	-	-	-
Mbps/mW	84.7	4.57	$1.40 * 10^{-2}$	$2.81 * 10^{-2}$	$1.76 * 10^{-1}$



Implementation Feasibility



Table : Peak Memory Bandwidth of Several Modern Systems

	Nexus 7	Nexus 10	iPhone 5	iPad 3	Intel i7	AMD FX
BW ($\frac{GB}{s}$)	5.3	12.8	8.5	12.8	25.6	21



Implementation Feasibility



Table : Number of Instances to Meet Peak BW

	C	C + Opt.	C + ISA	FPGA	ASIC
Intel i7	590	210	34	230	220

220 ASIC Modules \approx 16 mm² at 45 nm.

220 ASIC Modules \approx 2.4 W at 45 nm.



Outline

Introduction

Motivation

Background

Solution Characterization

Software

Hardware

Results

Conclusion



Summary



What Have We Shown?

- ▶ Software solutions require too much power.
- ▶ Software solutions require too much area.
- ▶ Software solutions are too slow.
- ▶ FPGA solution may be useful for existing designs.
- ▶ ASIC solution may be feasible for implementation in a real system.





Thank you!

Questions?



Backup Slides



Backup Slides



References



[Online]. Available: <http://www.xenatera.com/bunnie/proj/anatak/xboxmod.html#ldt>



R. Elbaz, D. Champagne, C. Gebotys, R. B. Lee, N. Potlapally, and L. Torres, "Hardware mechanisms for memory authentication: A survey of existing techniques and engines," in *Transactions on Computational Science IV*, M. L. Gavrilova, C. J. Tan, and E. D. Moreno, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. Hardware Mechanisms for Memory Authentication: A Survey of Existing Techniques and Engines, pp. 1–22. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01004-0_1



[Online]. Available: https://en.wikipedia.org/wiki/Galois/Counter_Mode



S. Gueron and M. E. Kounavis, "Intel® carry-less multiplication instruction and its usage for computing the gcm mode," *White Paper*, 2010.

