

Design and Implementation of the AEGIS Single-Chip Secure Processor Using Physical Random Functions

G. Edward Suh, Charles W. O'Donnell,
Ishan Sachdev, and Srinivas Devadas

Massachusetts Institute of Technology



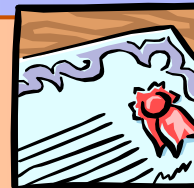


New Security Challenges

- Computing devices are becoming **distributed**, **unsupervised**, and **physically exposed**
 - Computers on the Internet (with untrusted owners)
 - Embedded devices (cars, home appliances)
 - Mobile devices (cell phones, PDAs, laptops)
- Attackers can **physically tamper** with devices
 - Invasive probing
 - Non-invasive measurement
 - Install malicious software
- Software-only protections are not enough

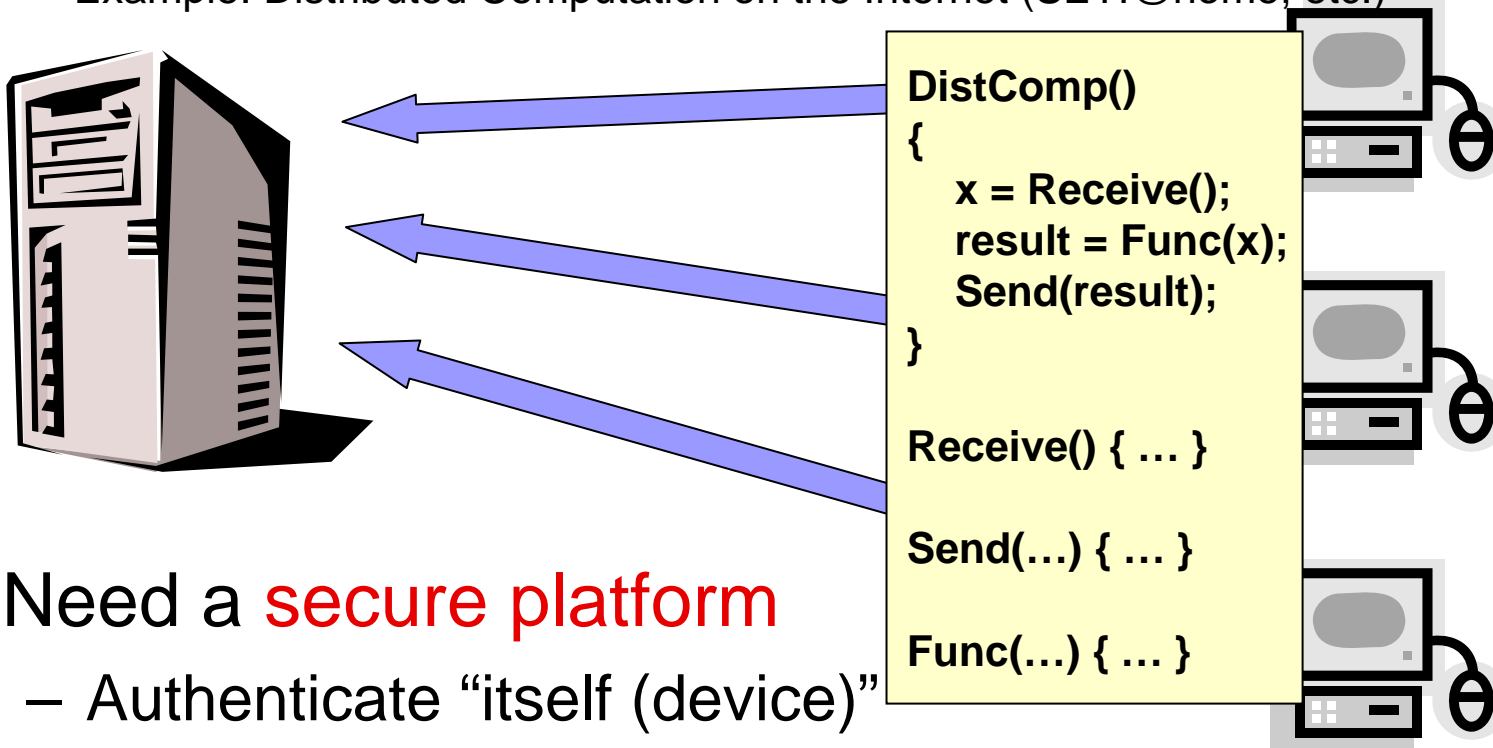


Distributed Computation



- How can we “**trust**” remote computation?

Example: Distributed Computation on the Internet (SETI@home, etc.)



- Need a **secure platform**
 - Authenticate “itself (device)”
 - Authenticate “software”
 - Guarantee the integrity and privacy of “execution”



Existing Approaches

Tamper-Proof Package: IBM 4758

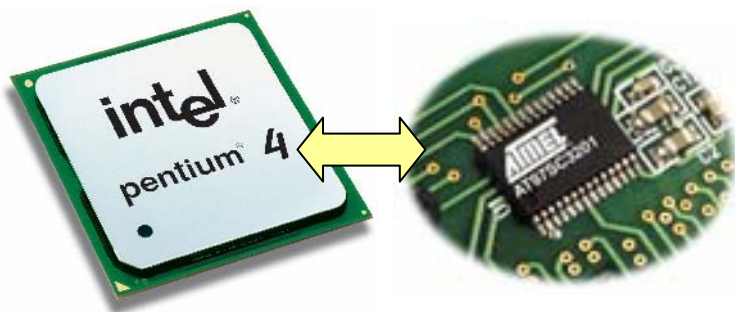


Sensors to detect attacks

Expensive

Continually battery-powered

Trusted Platform Module (TPM)



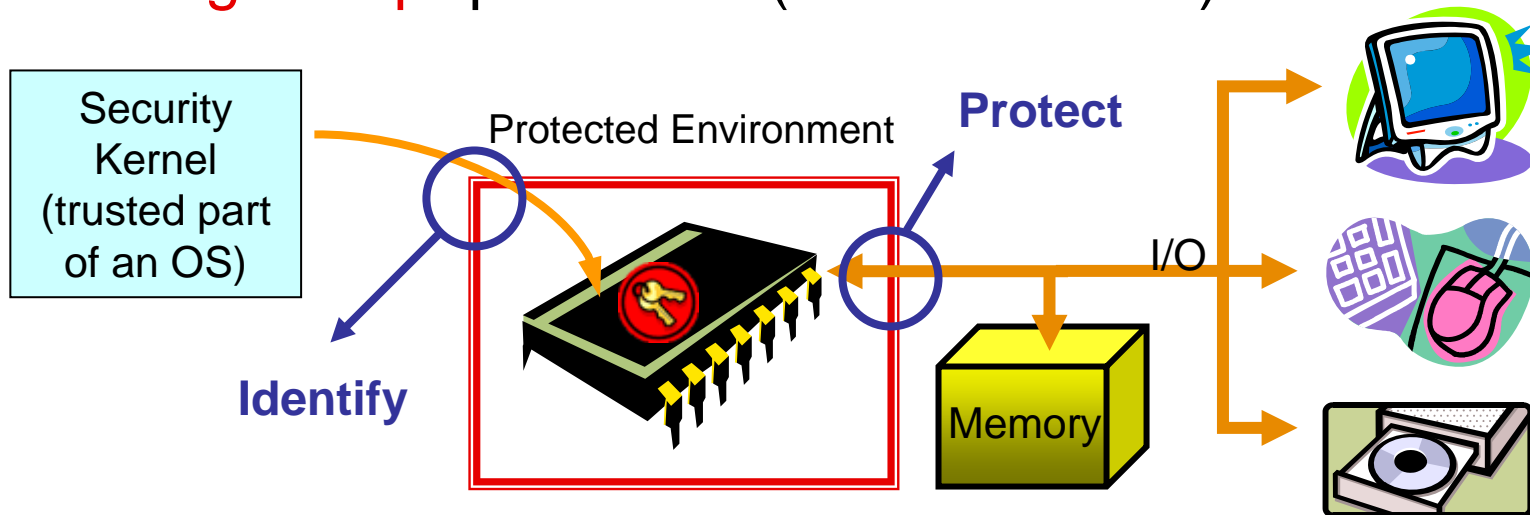
A separate chip (TPM) for security functions

Decrypted “secondary” keys can be read out from the bus

Our Approach



- Build a secure computing platform with only trusting a “**single-chip**” processor (named AEGIS)



- A **single chip** is easier and cheaper to protect
- The processor **authenticates** itself, **identifies** the security kernel, and **protects** off-chip memory



Contributions

- Physical Random Functions (PUFs)
 - Cheap and secure way to authenticate the processor
- Architecture to minimize the trusted code base
 - Efficient use of protection mechanisms
 - Reduce the code to be verified
- Integration of protection mechanisms
 - Additional checks in MMU
 - Off-chip memory encryption and integrity verification (IV)
- Evaluation of a fully-functional RTL implementation
 - Area Estimate
 - Performance Measurement

Physical Random Function

(PUF – Physical Unclonable Function)



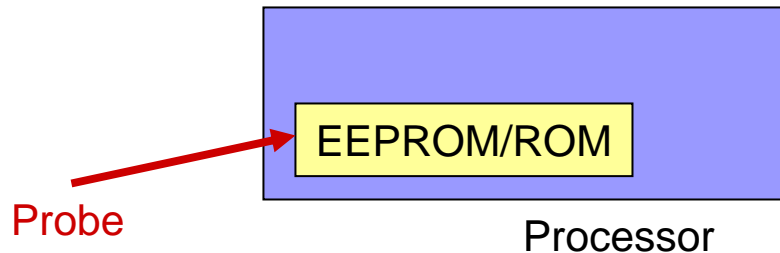
CSAIL

MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY



Problem

Storing **digital** information in a device in a way that is resistant to **physical attacks** is difficult and expensive.

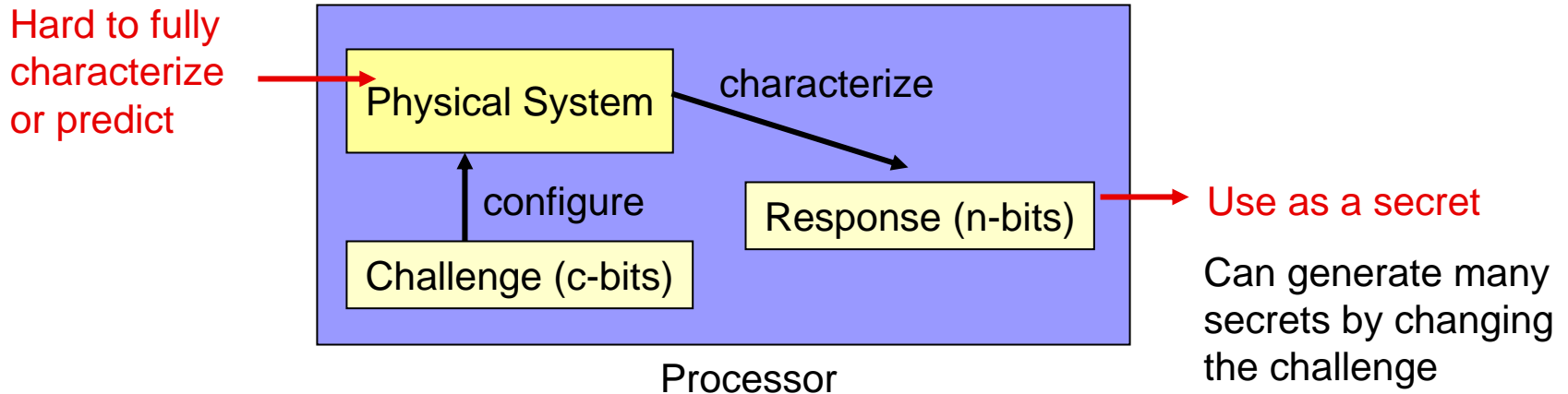


- Adversaries can physically **extract secret** keys from EEPROM while **processor is off**
- Trusted party must **embed and test secret** keys in a secure location
- EEPROM adds additional complexity to manufacturing



Our Solution: Physical Random Functions (PUFs)

- Generate keys from a **complex physical system**

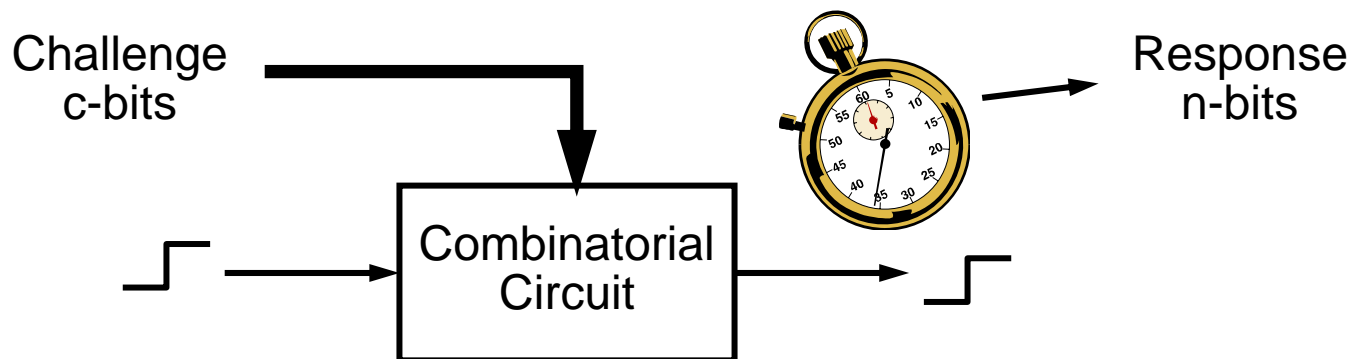


- Security Advantage
 - Keys are generated on demand → No non-volatile secrets
 - No need to program the secret
 - Can generate multiple master keys
- What can be **hard to predict**, but **easy to measure**?



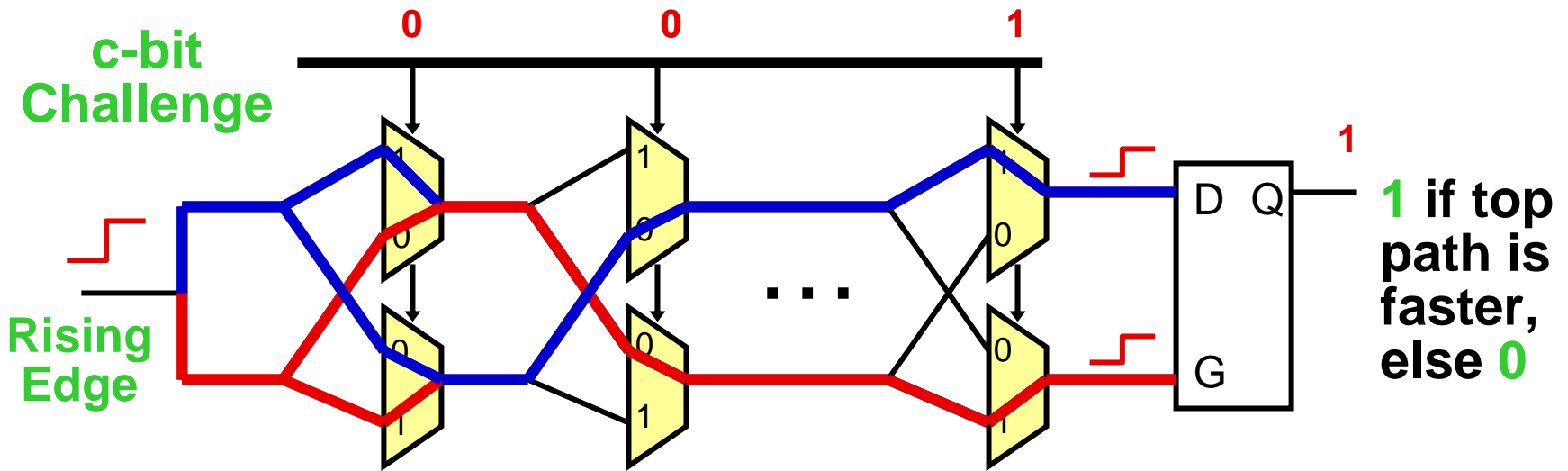
Silicon PUF – Concept

- Because of random process variations, **no two Integrated Circuits even with the same layouts are identical**
 - Variation is **inherent** in fabrication process
 - **Hard** to remove or predict
 - Relative variation **increases** as the fabrication process advances
- Experiments in which **identical circuits with identical layouts** were placed on different ICs show that path delays vary enough across ICs to use them for identification.



A (Simple) Silicon PUF

[VLSI'04]



Each challenge creates two paths through the circuit that are excited simultaneously. The digital response of 0 or 1 is based on a **comparison of the path delays** by the arbiter

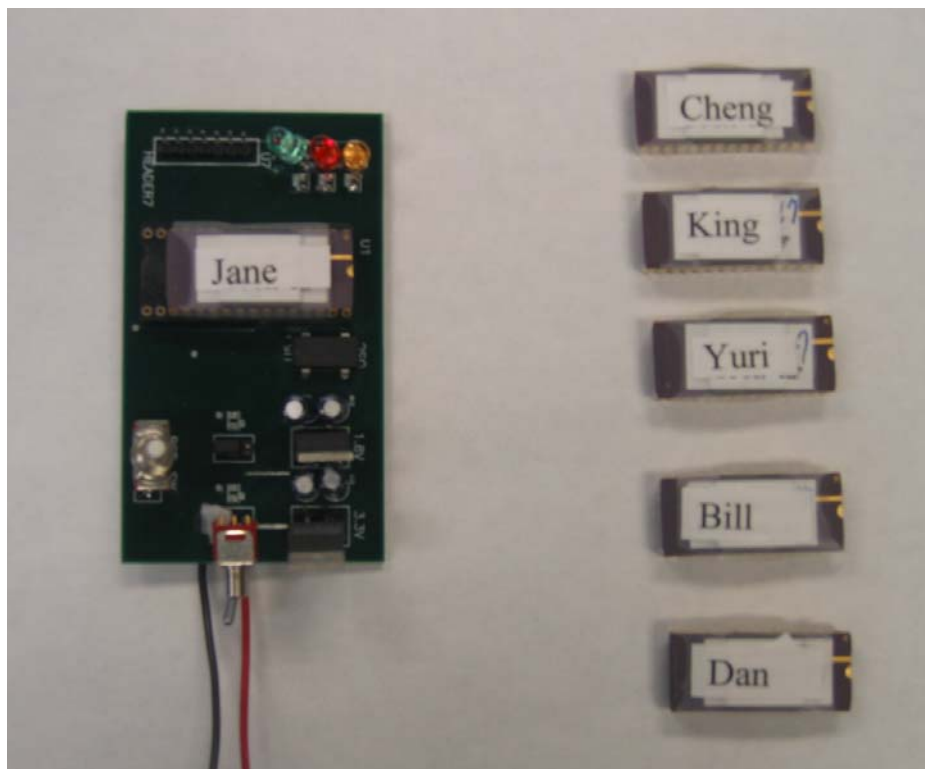
We can obtain n-bit responses from this circuit by either duplicate the circuit n times, or use n different challenges

Only use standard digital logic → **No special fabrication**



PUF Experiments

- Fabricated 200 “identical” chips with PUFs in TSMC 0.18 μ on 5 different wafer runs



Security

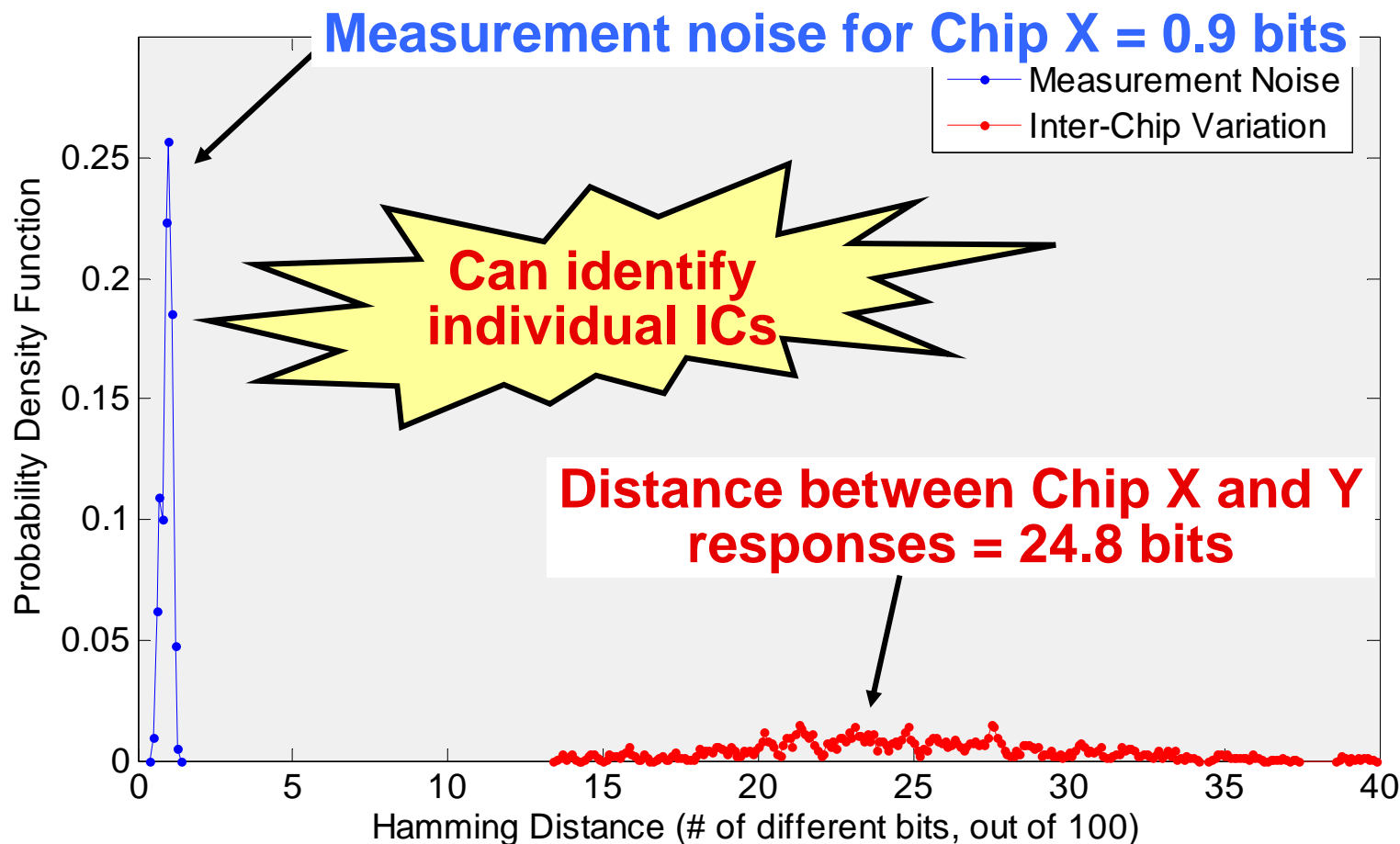
- What is the probability that a challenge produces different responses on two different PUFs?

Reliability

- What is the probability that a PUF output for a challenge changes with temperature?
- With voltage variation?

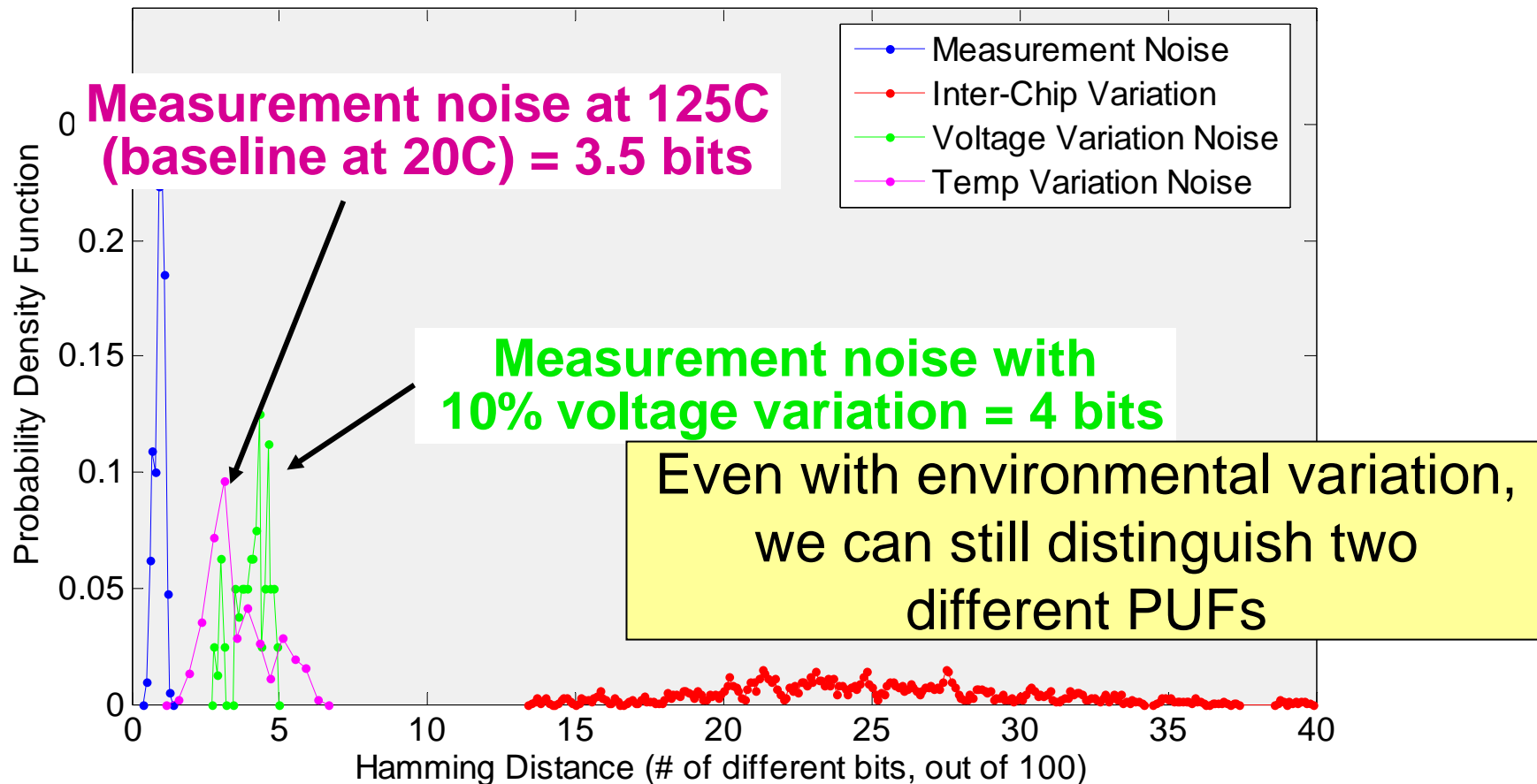
Inter-Chip Variation

- Apply random challenges and observe 100 response bits



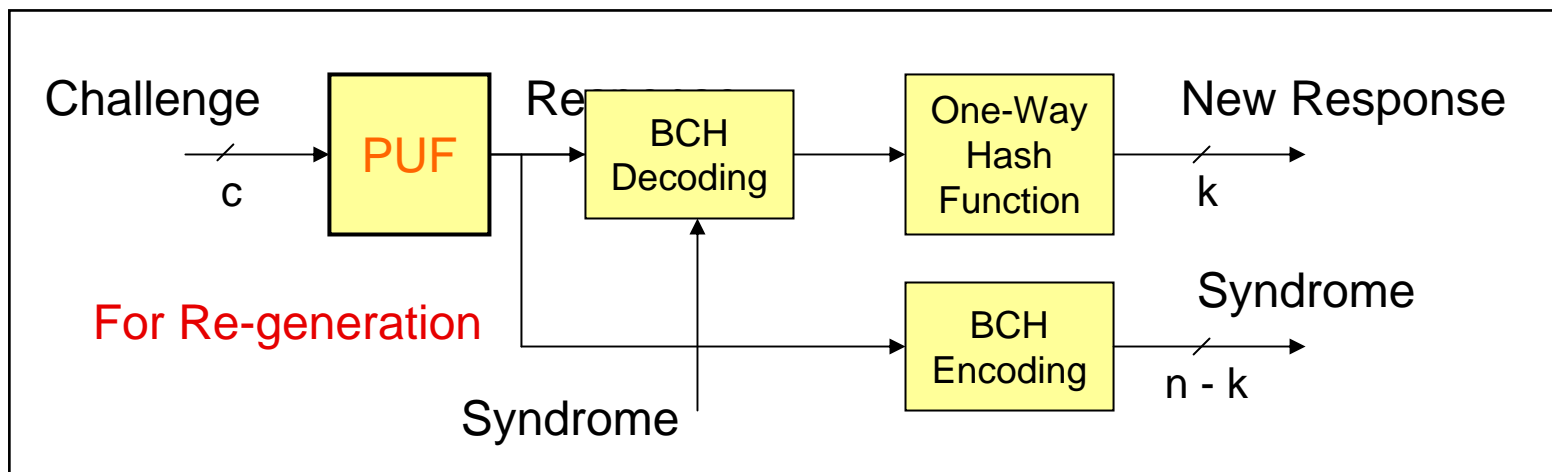
Environmental Variations

- What happens if we change voltage and temperature?



Reliable PUFs

PUFs can be made more **secure** and **reliable** by adding extra control logic



- Hash function (SHA-1, MD5) **precludes** PUF “model-building” attacks since, to obtain PUF output, adversary has to invert a one-way function
- Error Correcting Code (ECC) can **eliminate the measurement noise** without compromising security

Architecture Overview



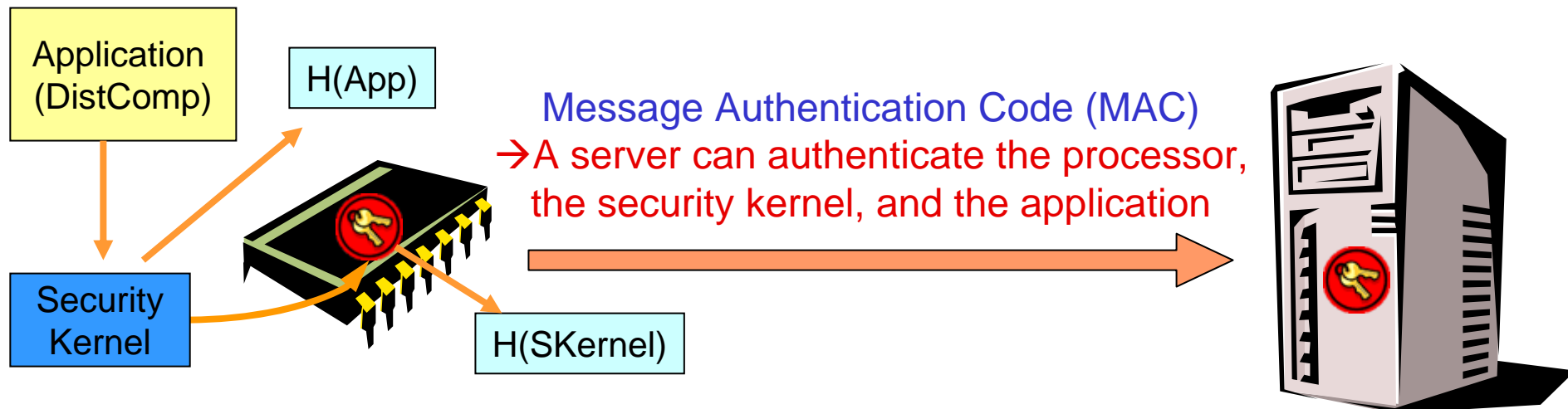
CSAIL

MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY



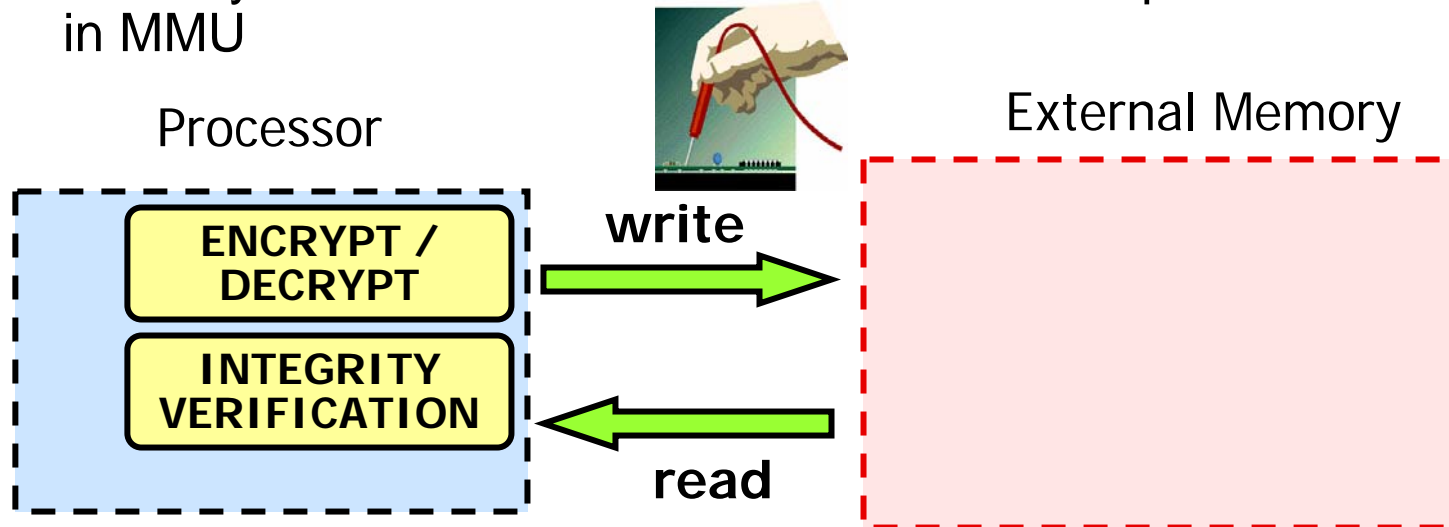
Authentication

- The processor **identifies security kernel** by computing the kernel's **hash** (on the `I.enter.aegis` instruction)
 - Similar to ideas in TCG TPM and Microsoft NGSCB
 - Security kernel identifies application programs
- $H(\text{SKernel})$ is used to produce a **unique key** for security kernel from a PUF response (`I.puf.secret` instruction)
 - Security kernel provides a unique key for each application



Protecting Program State

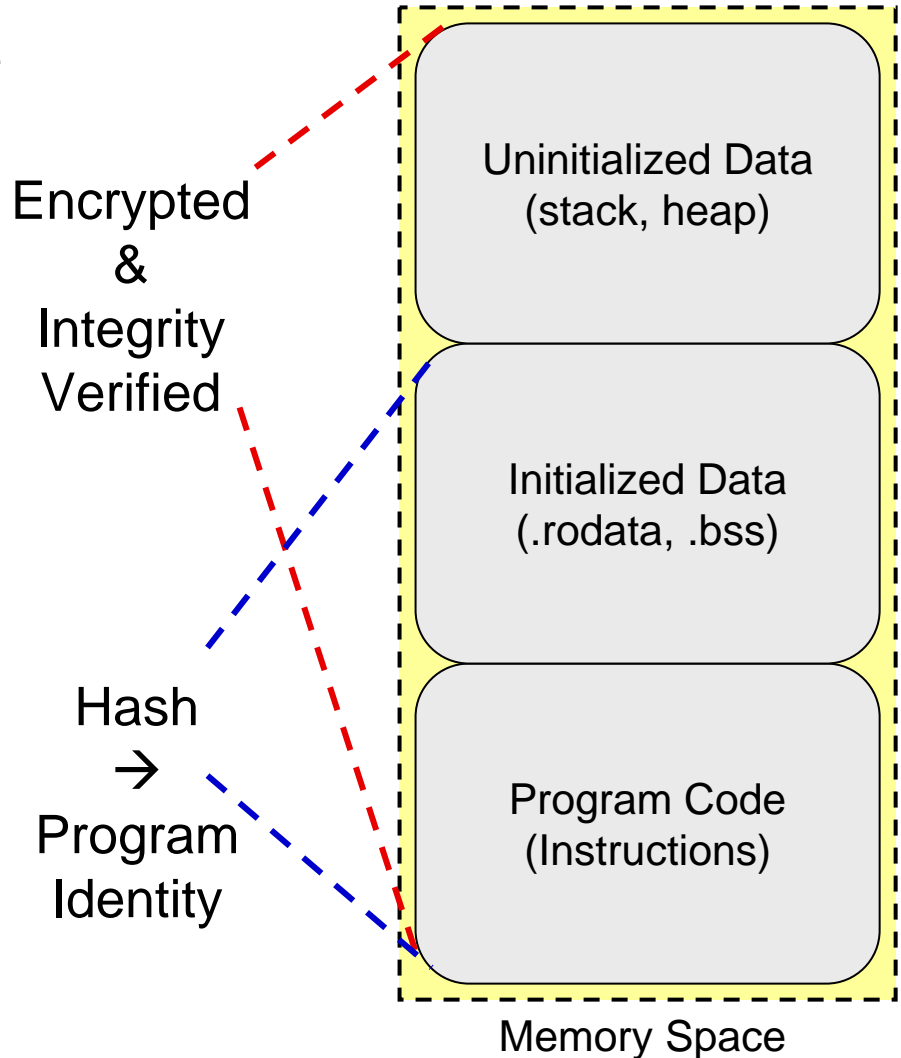
- On-chip registers and caches
 - Security kernel handles context switches and permission checks in MMU



- Memory Encryption [MICRO36][Yang 03]
 - Counter-mode encryption
- Integrity Verification [HPCA'03,MICRO36,IEEE S&P '05]
 - Hash trees

A Simple Protection Model

- How should we apply the authentication and protection mechanisms?
- What to protect?
 - All instructions and data
 - Both integrity and privacy
- What to trust?
 - The entire program code
 - Any part of the code can read/write protected data





What Is Wrong?

- Large Trusted Code Base
 - Difficult to verify to be bug-free
 - How can we trust shared libraries?
- Applications/functions have varying security requirements
 - Do all code and data need privacy?
 - Do I/O functions need to be protected?
 - Unnecessary performance and power overheads
- Architecture should provide **flexibility** so that software can choose the **minimum** required trust and protection





Distributed Computation Example

DistComp()

{

x = Receive();


result = Func(x);

key = get_puf_secret();
mac = MAC(x,result,key);

Send(result,mac);

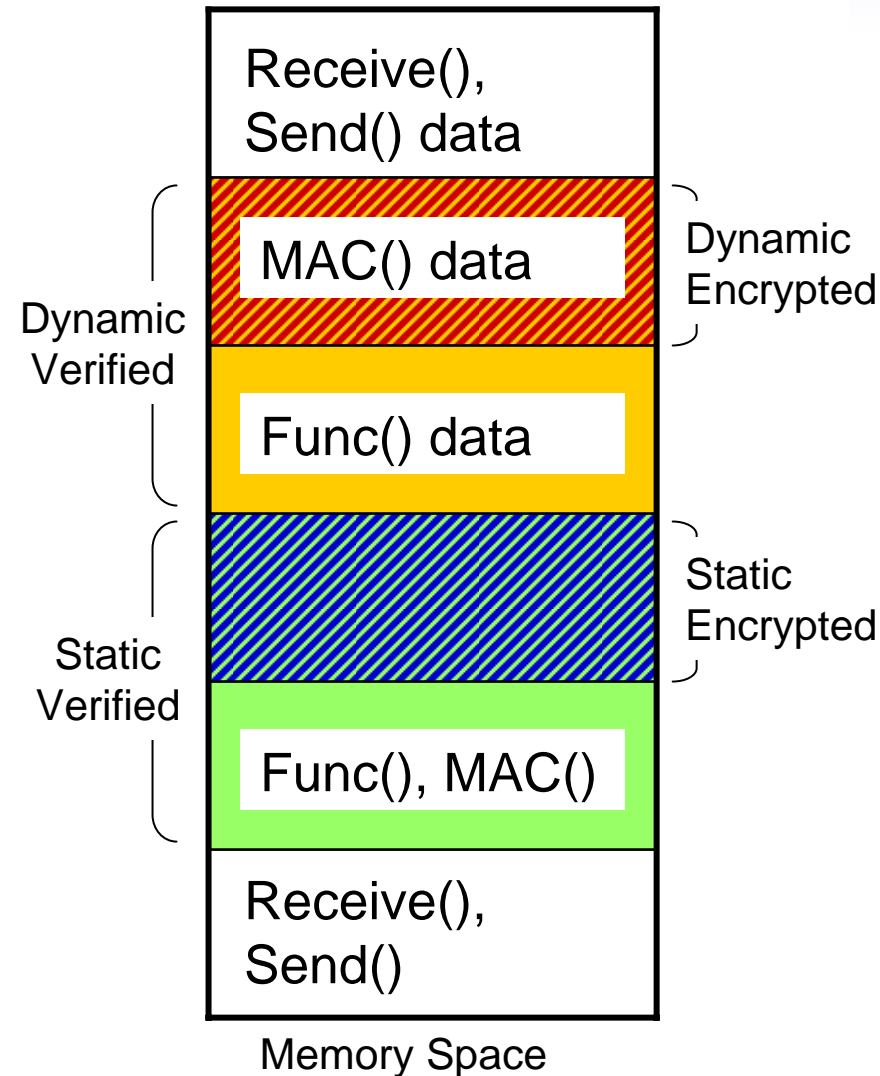
}

- Obtaining a secret key and computing a MAC
 - Need **both privacy and integrity**
- Computing the result
 - Only need **integrity**
- Receiving the input and sending the result (I/O)
 - No need for protection
 - No need to be trusted



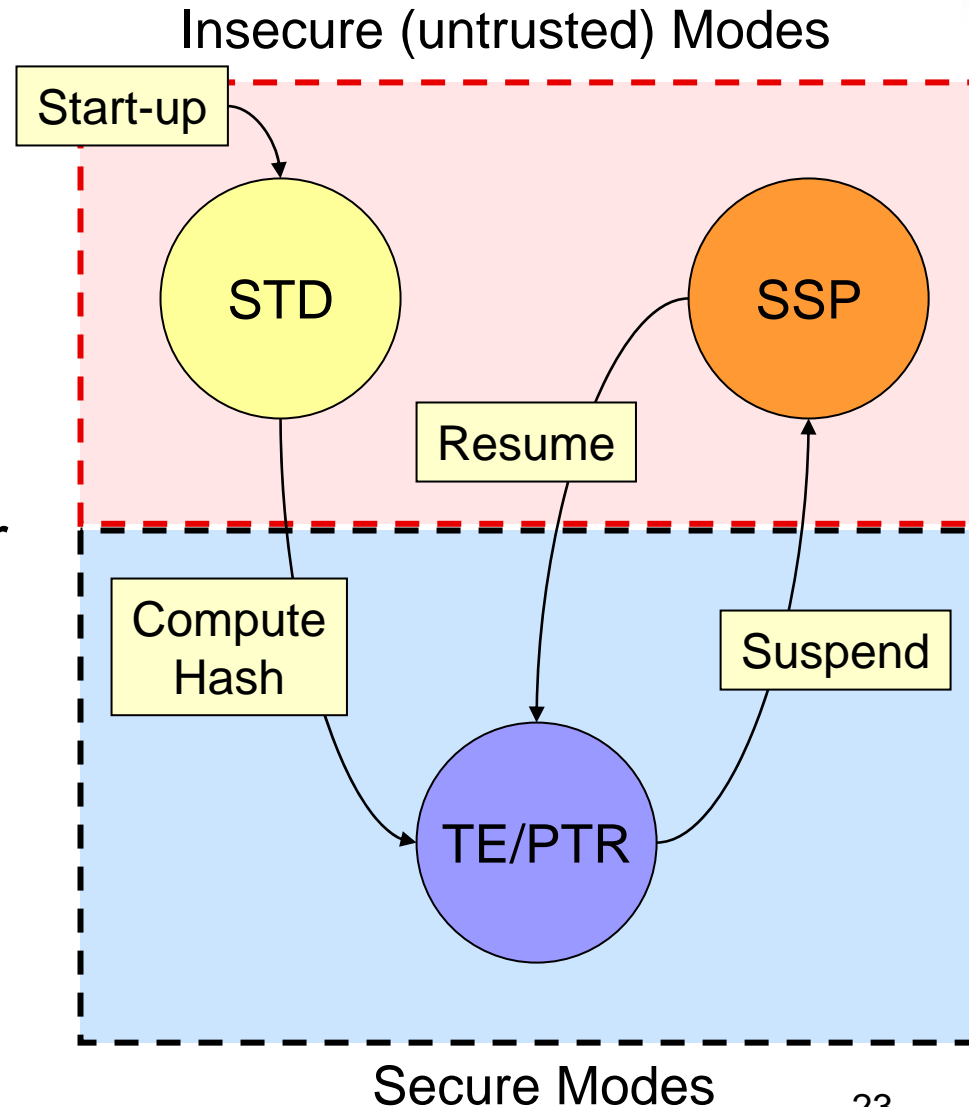
AEGIS Memory Protection

- Architecture provides five different memory regions
 - Applications choose how to use
- Static (read-only)
 - Integrity verified
 - Integrity verified & encrypted
- Dynamic (read-write)
 - Integrity verified
 - Integrity verified & encrypted
- Unprotected
- Only authenticate code in the verified regions



Suspended Secure Processing (SSP)

- Two security levels within a process
 - Untrusted code such as Receive() and Send() should have less privilege
- Architecture ensures that SSP mode cannot tamper with secure processing
 - No permission for protected memory
 - Only resume secure processing at a specific point



Secure Modes

Implementation & Evaluation

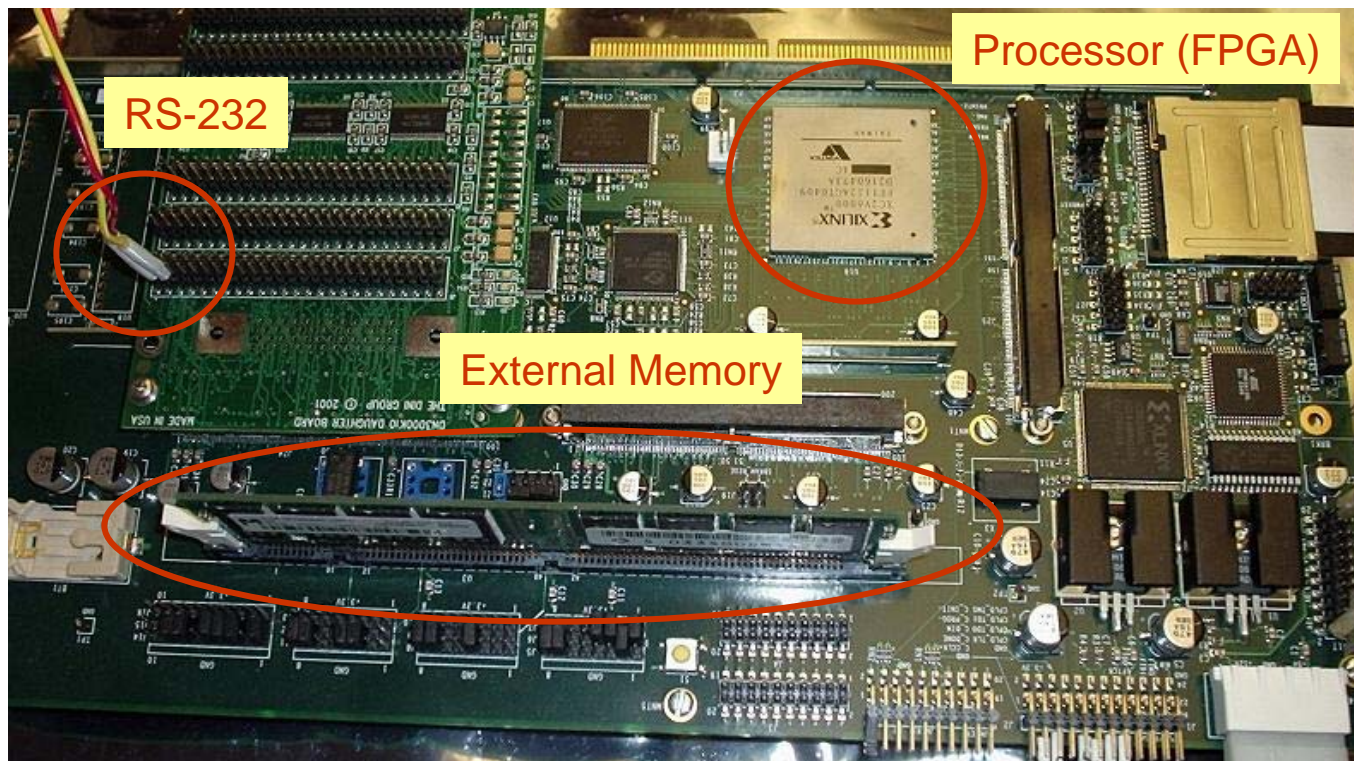


CSAIL

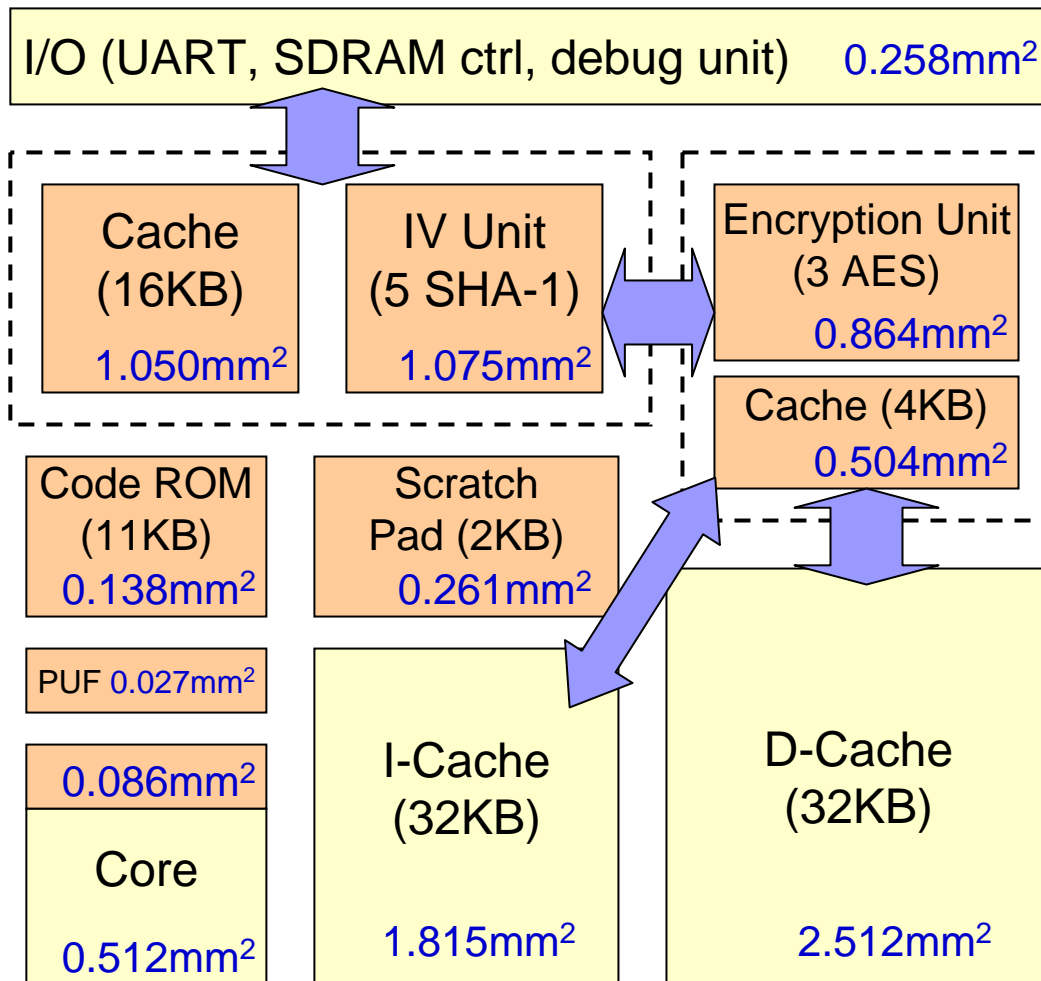
MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

Implementation

- Fully-functional system on an FPGA board
 - AEGIS (Virtex2 FPGA), Memory (256MB SDRAM), I/O (RS-232)
 - Based on openRISC 1200 (a simple 4-stage pipelined RISC)
 - AEGIS instructions are implemented as special traps



Area Estimate



- Synopsys DC with TSMC 0.18u lib
- New instructions and PUF add 30K gates, 2KB mem (1.12x larger)
- Off-chip protection adds 200K gates, 20KB memory (1.9x larger total)
- The area can be further optimized



Performance Slowdown

- Performance overhead comes from **off-chip protections**
- Synthetic benchmark
 - Reads 4MB array with a varying stride
 - Measures the slowdown for a varying cache miss-rate
- Slowdown is reasonable for realistic miss-rates
 - Less than 20% for integrity
 - 5-10% additional for encryption

D-Cache miss-rate	Slowdown (%)	
	Integrity	Integrity + Privacy
6.25%	3.8	8.3
12.5%	18.9	25.6
25%	31.5	40.5
50%	62.1	80.3
100%	130.0	162.0



EEMBC/SPEC Performance

- 5 EEMBC kernels and 1 SPEC benchmark
- EEMBC kernels have negligible slowdown
 - Low cache miss-rate
 - Only ran 1 iteration
- SPEC twolf also has reasonable slowdown

Benchmark	Slowdown (%)	
	Integrity	Integrity + Privacy
routelookup	0.0	0.3
ospf	0.2	3.3
autocor	0.1	1.6
conven	0.1	1.3
fbital	0.0	0.1
twolf (SPEC)	7.1	15.5



Related Projects

- XOM (eXecution Only Memory)
 - Stated goal: Protect integrity and privacy of code and data
 - Operating system is completely untrusted
 - Memory integrity checking does not prevent replay attacks
 - Privacy enforced for all code and data
- TCG TPM / Microsoft NGSCB / ARM TrustZone
 - Protects from software attacks
 - Off-chip memory integrity and privacy are assumed
- AEGIS provides “higher security” with “smaller Trusted Computing Base (TCB)”



Summary

- Physical attacks are becoming more prevalent
 - Untrusted owners, physically exposed devices
 - Requires secure hardware platform to trust remote computation
- The trusted computing base should be small to be secure and cheap
 - Hardware: **single-chip** secure processor
 - Physical random functions
 - Memory protection mechanisms
 - Software: suspended secure processing
- Initial overheads of the AEGIS single-chip secure processor is promising

Questions?

More information on www.csg.csail.mit.edu



CSAIL

MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY