Security for the Internet of Things

Srini Devadas

Webster Professor of Electrical Engineering and Computer Science

Computer Science and Artificial Intelligence Laboratory (CSAIL)
Massachusetts Institute of Technology

Agenda

Why is security for IoT hard?

Threat models

Defensive strategies and examples

- Prevention
- Resilience under attack
- Detection and Recovery

Attacks on Individuals: Ransomware

- Worm enters system through downloaded file.

- Payload encrypts user’s hard drive and deletes the original files – user cannot decipher his/her own files

- Pay $500 in Bitcoin to get your files back!

Attacks on Services

Target Store in 2013

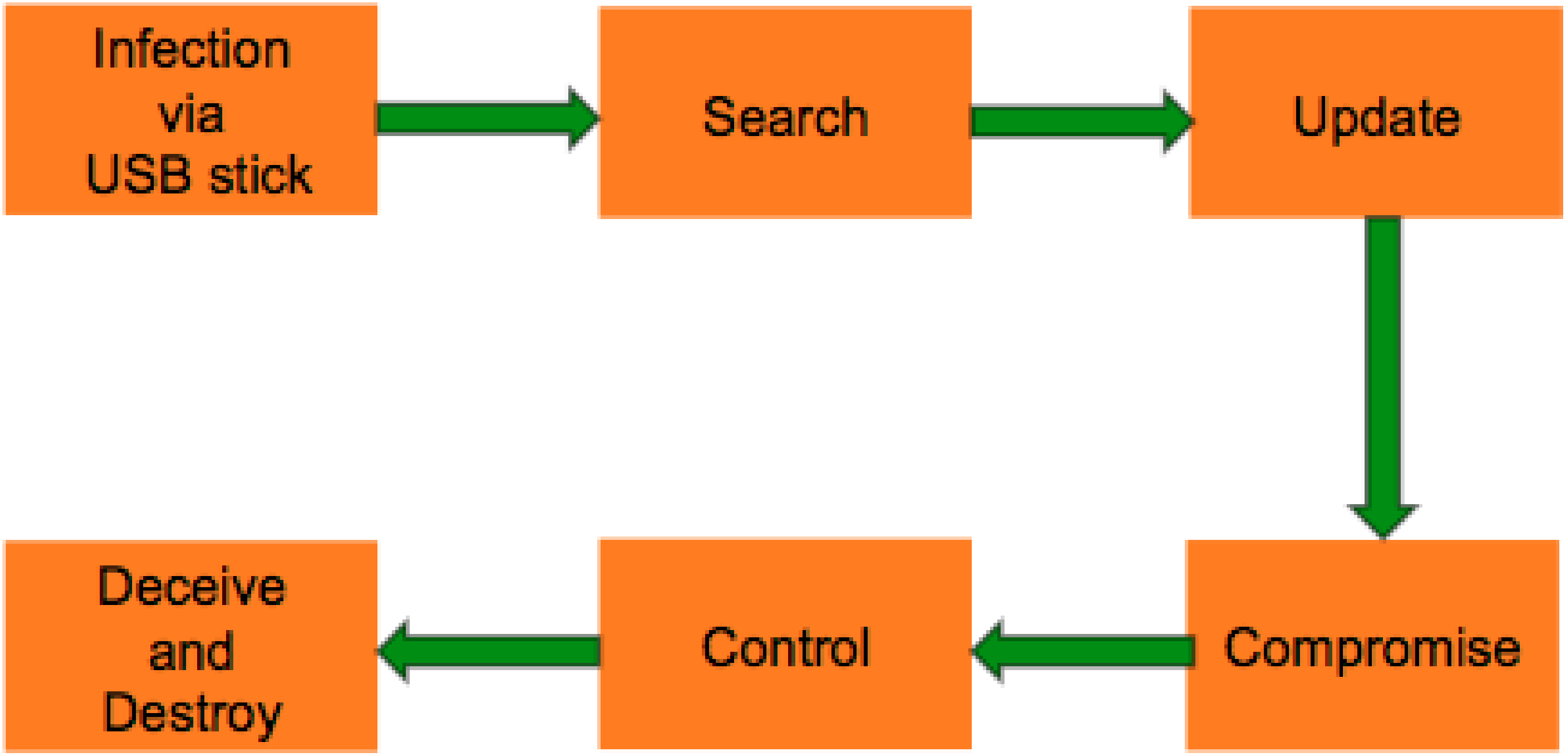
40 million: Number of credit and debit cards thieves stole.
70 million: The number of records stolen that included names, and addresses
46: % drop in profits in the 4th quarter of 2013, compared to 2012.
200 million: Estimated cost for reissuing 21.8 million cards.
53.7 million: The income that hackers likely generated from the sale of 2 million cards

0: Number of customer cards with AVAILABLE hardware security technology that would have stopped the bad guys from stealing

Attacks on Infrastructure
The Stuxnet Cyberphysical Attack

- A 500 Kbyte computer worm that infected the software of at least 14 industrial sites in Iran including a nuclear facility
- Goal was to cause fast-spinning centrifuges to tear themselves apart
- Stuxnet was tracked down by Kaspersky Labs but not before it did some damage

How Stuxnet Worked



Why Do These Attacks Happen? *or* What Makes Security Hard?

Security is a negative goal.

- Want to achieve something despite whatever adversary might do.

- Positive goal: "Frans can read grades.txt".
  - Ask Frans to check if our system meets this positive goal.
- Negative goal: "Nick cannot read grades.txt".
  - Ask Nick if he can read grades.txt?
  - Must reason about all possible ways in which Nick might get the data.

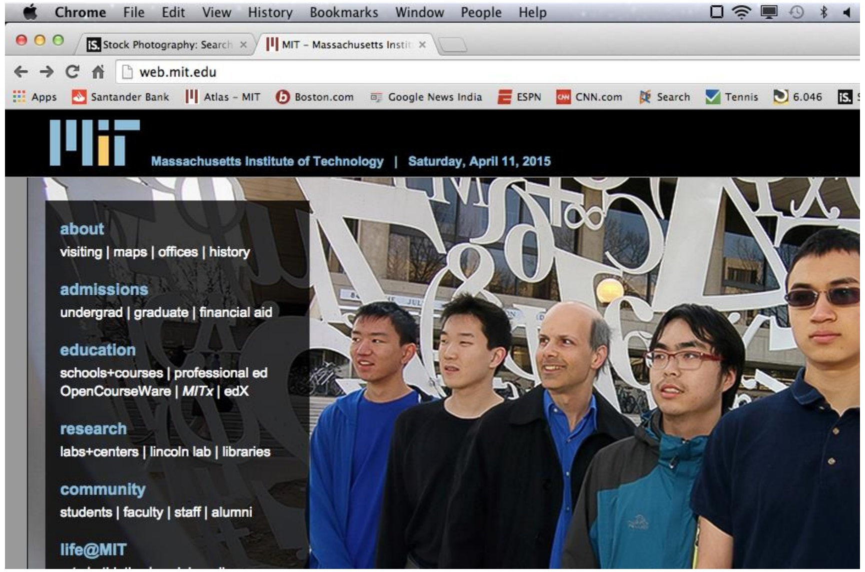
Ways to Access Grades.txt

- Change permissions on grades.txt to get access

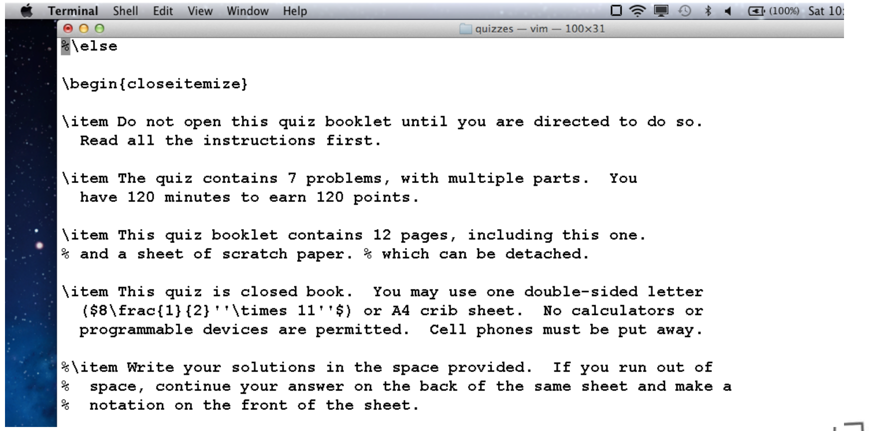
- Access disk blocks directly

Ways to Access Grades.txt

- Access grades.txt via web.mit.edu



- Reuse memory after Frans's text editor exits, read data



Ways to Access Grades.txt

- Read backup copy of grades.txt from Frans's text editor

- Intercept network packets to file server storing grades.txt

Ways to Access Grades.txt

- Send Frans a trojaned text editor that emails out the file

- Steal disk from file server storing grades.txt

Ways to Access Grades.txt

- Get discarded printout of grades.txt from the trash

- Call sysadmin, pretend to be Frans, reset his password

Agenda

Why is security for IoT hard?

Threat models

Defensive strategies and examples
- Prevention
- Resilience under attack
- Detection and Recovery

Why Threat Models?

Often don't know in advance who might attack, or what they might do.

- Adversaries may have different goals, techniques, resources, expertise.
- Adversary might be your hardware vendor, software vendor, administrator, ..

Cannot be secure against arbitrary adversaries, as we saw with Nick vs. Frans.

Need to make some plausible assumptions to make progress.

What Does a Threat Model Look Like?

- Adversary controls some computers, networks (but not all).
- Adversary controls some software on computers he doesn't fully control.
- Adversary knows some information, such as passwords or keys (but not all).
- Adversary knows about bugs in your software

What Does a Threat Model Look Like? – 2

Physical attacks?

Social engineering attacks?

Many systems compromised due to unrealistic / incomplete threat models.

- Adversary is outside of the company firewall
- Adversary doesn't know legitimate users' passwords.
- Adversary won't figure out how the system works.

Cybersecurity and Threats

Cybersecurity is a property of computer systems similar to performance and energy

Attackers take a holistic view by attacking any component or interface of system

Diverse threat models dictate different desirable security properties

One Philosophy

- Computer systems are so complex that it is impossible to design them without vulnerabilities.
- Best strategy is therefore to:
  - Focus on existing computing systems and their attacks to discover flaws
  - Design mechanisms into systems to protect against these attacks
  - Manage risk and administer systems well

One Philosophy

Unfortunately, new flaws are always discovered…

Can we do better than a "Patch & Pray, Perimeter Protection" mindset?

A Holistic Philosophy

Security property cannot be articulated well when *isolated* to a component or layer

→ need a systems-wide, architectural viewpoint

New theoretical and practical foundations of secure computing that integrate security in the design process

→ security "by default"

→Remove program error as a source of vulnerability

Three Defenses

Prevention: Increasing the difficulty of attacks

Resilience: Allowing a system to remain functional despite attacks

Detection and Recovery: Allowing systems to more quickly detect and recover from attacks to fully functional state.

Agenda

Why is security hard?

Threat models

Defensive strategies and examples

- Prevention
- Resilience under attack
- Detection and Recovery

Prevention

Protection Against Physical Attack

Traditional Device Authentication

Each IC needs to be unique

- Embed a unique secret key SK in on-chip non-volatile memory

Use cryptography to authenticate an IC

Cryptographic operations can address other problems such as protecting IP or secure communication

Traditional Device Authentication

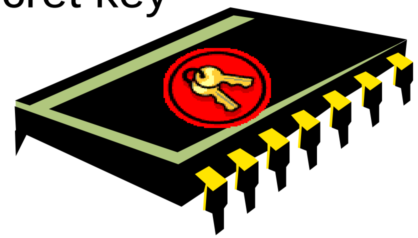
Each IC needs to be unique

- Embed a unique secret key SK in on-chip non-volatile memory

Use cryptography to authenticate an IC

Cryptographic operations can address other problems such as protecting IP or secure communication

IC with
a secret key



Sends a random number

Sign the number with a secret key
→Only the IC's key can generate
a valid signature

IC's Public Key

BUT…

How to generate and store secret keys on ICs in a secure and inexpensive way?

- Adversaries may physically extract secret keys from non-volatile memory
- Trusted party must embed and test secret keys in a secure location

Invasive probing

Non-invasive measurement

What if cryptography is NOT available?

- Extremely resource (power) constrained systems such as passive RFIDs and sensor nodes in IoT

Physical Unclonable Functions (PUFs)

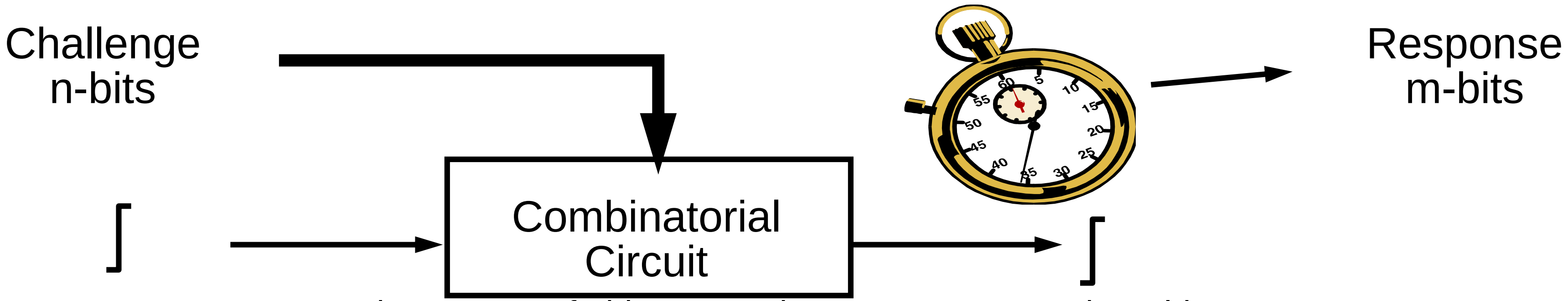
Extract secrets from a complex physical system

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

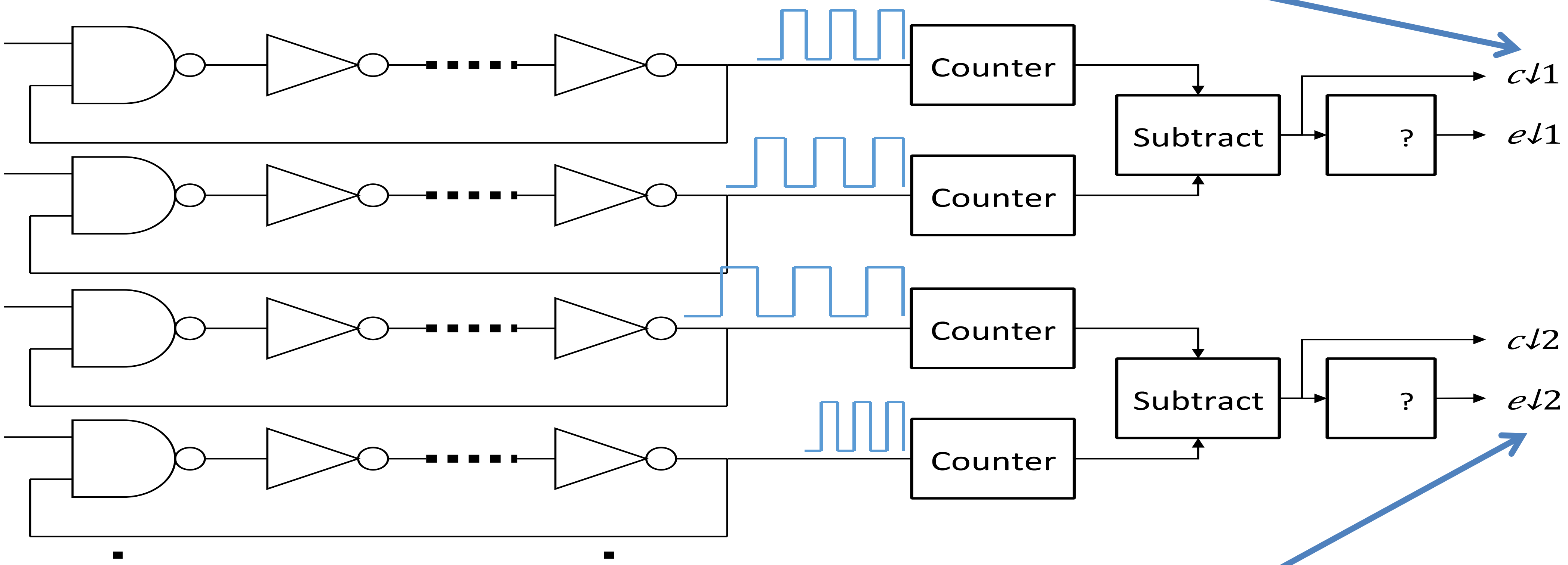
Delay-Based Silicon PUF concept (2002)

- Generate keys from unique delay features of chips



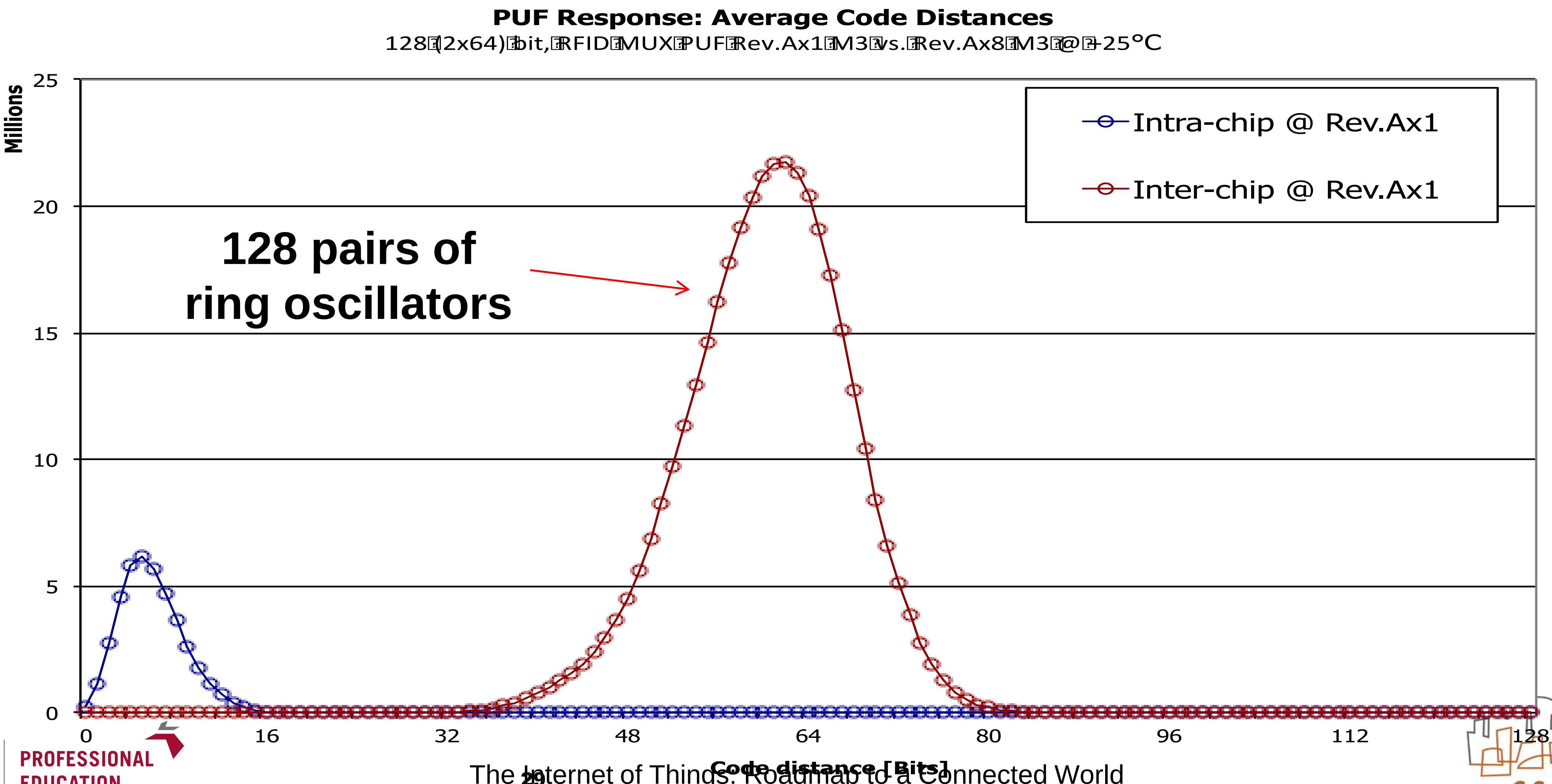
Ring Oscillators

Confidence information: stability of the bit

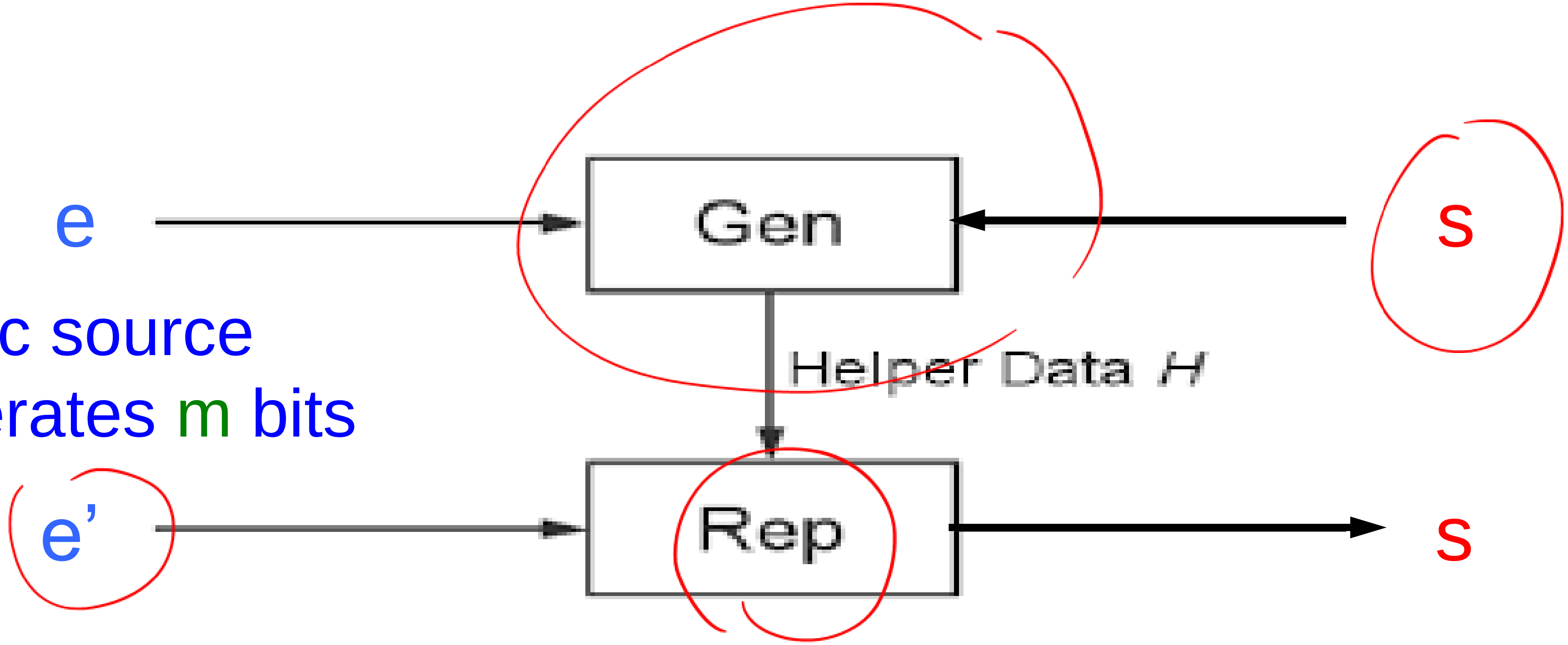


Response bit 0 or 1

Ring Oscillator Experiments



Fuzzy Extractors



Biometric source (re)generates m bits

- Helper Data H will leak information about e, s
- Entropy of secret key s should be large enough even with knowledge of H

Learning Parity with Noise

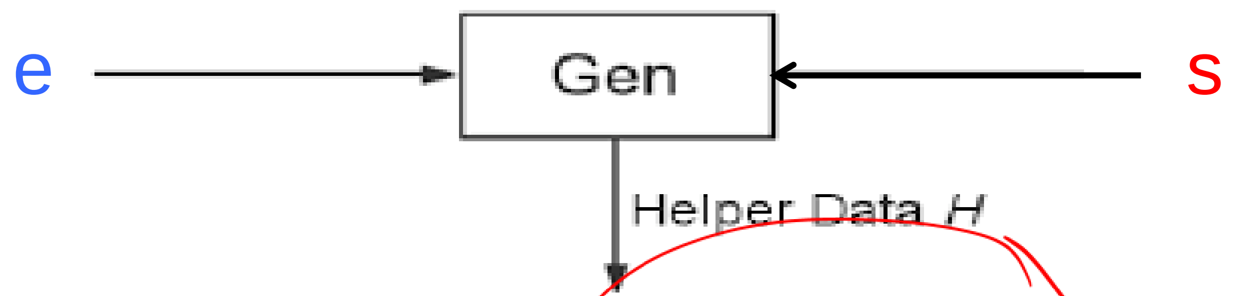
$$b_1 = a_1 . s + e_1$$
$$b_2 = a_2 . s + e_2$$
$$\ldots$$
$$b_m = a_m . s + e_m$$

s secret, a_i and b_i public, e_i is hidden noise

a_i, s are n-bit vectors, b_i, e_i are bits

Hard to discover s given a_i and b_i
for any $m > n$ for any non-zero noise level

Gen Step



$$b_1 = a_1 \,.\, s + e_1$$

$$b_2 = a_2 \,.\, s + e_2$$

$$\ldots$$

$$b_m = a_m \,.\, s + e_m$$

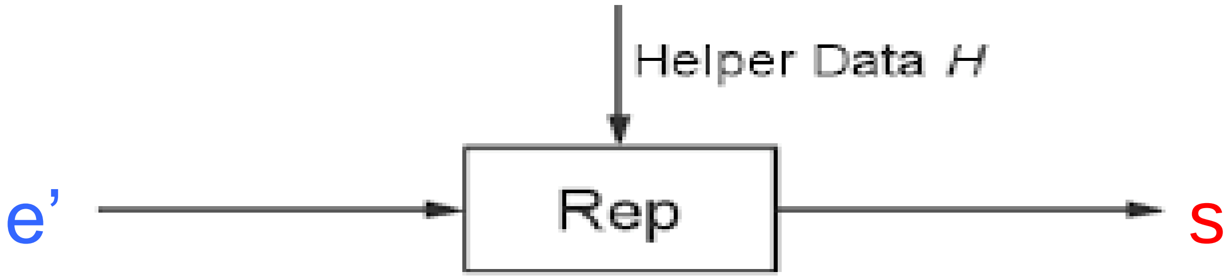
Computed and is public helper data

Choose randomly. Same and public for all instances

PUF generates these values

Choose randomly and is secret

Rep Step



$$b_1 = a_1 \,.\, s + e'_1$$
$$b_2 = a_2 \,.\, s + e'_2$$
$$\ldots$$
$$b_m = a_m \,.\, s + e'_m$$

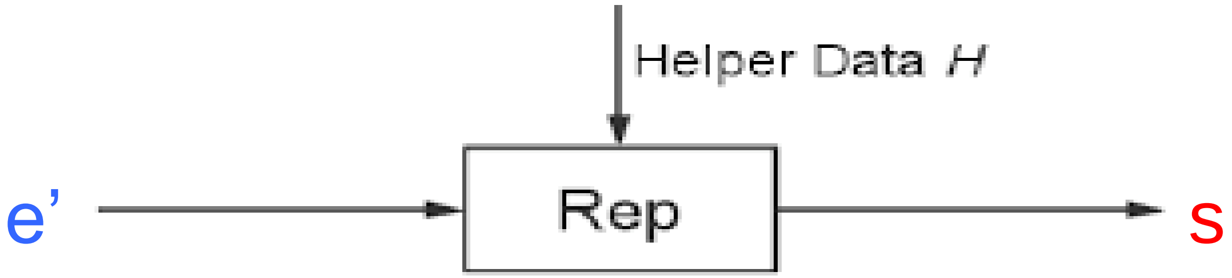
known

known

PUF regenerates these values

unknown

Rep Step



$$b_1 = a_1 \,.\, s + e'_1$$

$$b_2 = a_2 \,.\, s + e'_2$$

$$\ldots$$

$$b_m = a_m \,.\, s + e'_m$$

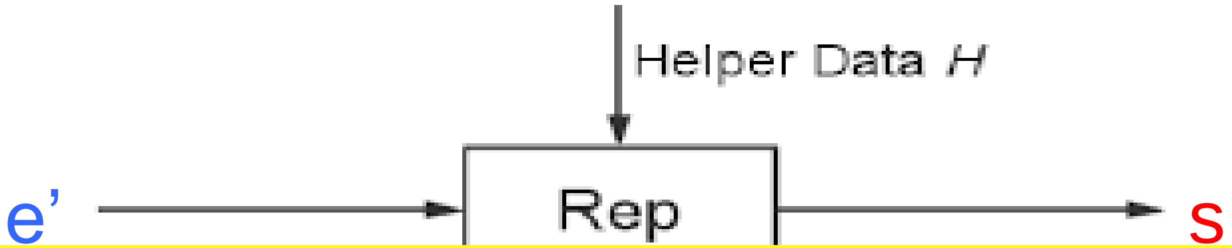
known

known

PUF regenerates these values

unknown

Rep Step



- Problem: e'_i values not the same as e_i values
- LPN is hard even for small amount of noise

☹

known

unknown

Confidence Information is a “Trapdoor”

$$b_1 = a_1 \,.\, s + e'_1$$

$$\ldots$$

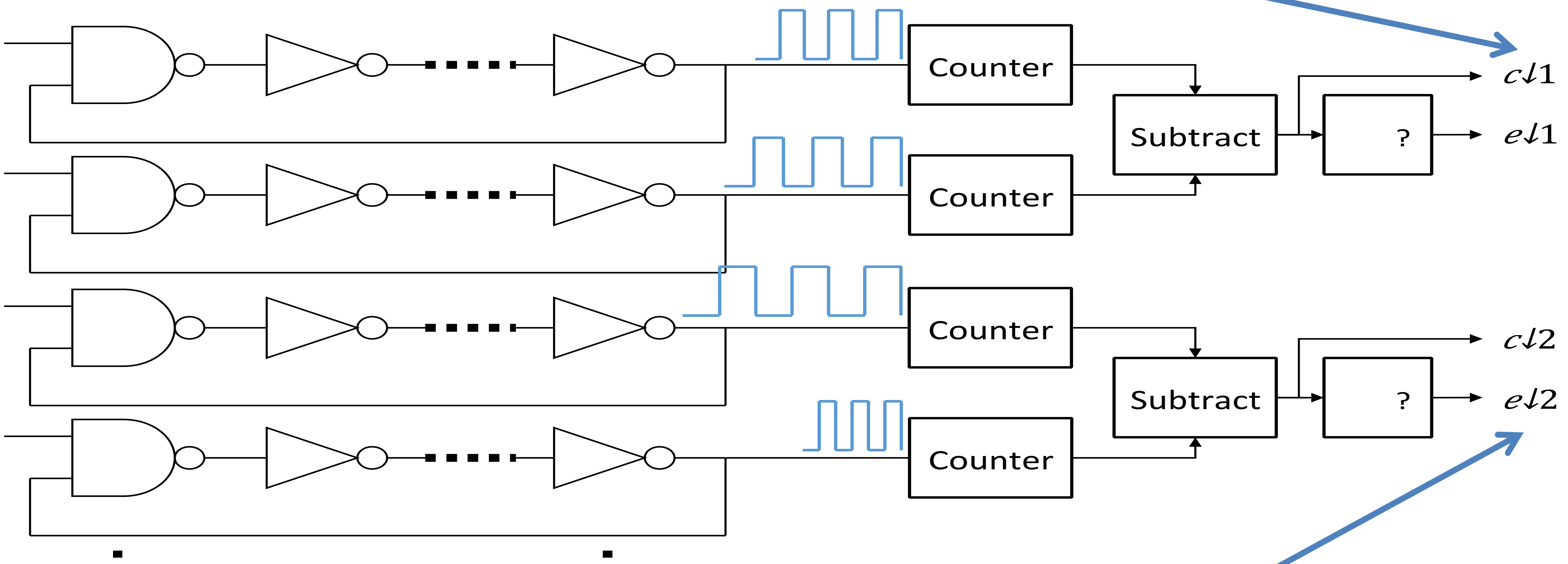
$$b_m = a_m \,.\, s + e'_m$$

Need only n out of m e'_i values to be correct to solve for s

Can use confidence information associated with each e'_i value to find n correct values!

Ring Oscillators

Confidence information: stability of the bit



Response bit 0 or 1

Result

$$b_1 = a_1 \,.\, s + e_1$$

$$b_3 = a_3 \,.\, s + e_3$$

$$\ldots$$

$$b_m = a_m \,.\, s + e_m$$

Theoretical result: If $m = \Theta(n^2)$ can correct $\Theta(m)$ errors with *1* $O(n^3)$ Gaussian elimination (GE) step

Adversary: solve LPN (exponential work) to discover key

PUF Applications

PUF can enable secure, low-cost authentication

- Use PUF as a function: challenge ⌫ response
- Only authentic IC can produce a correct response for a challenge
- Inexpensive: no special fabrication technique

PUF can generate a unique secret key / ID

- Physically secure: volatile secrets, no need for trusted programming
- Can integrate key generation into a secure processor

Resilience Under Attack

Encrypted Computation

Trusted Computing Base

The trusted computing base (TCB) is the set of software and hardware components that need to be trusted by a user

In the cloud, the TCB is > 20M lines of code from tens of software vendors

- No wonder we have so many security breaks!

In the Internet of Things the TCB could be even larger!

Trusted Computing Base

The trusted computing base (TCB) i f software and hardware components that ed by a user

In TPM-based system e veracity of several millions of code

- The TPM ecurity

In the clou M lines of code from tens of software ve

- No wond ave so many security breaks!

CAN WE SHRINK THE TCB?

Computing with Untrusted Software

I want to delegate processing of my data, without giving away access to it.

Separating processing from access via encryption:

- I will encrypt my data before sending it to the cloud
- They will apply their processing on the encrypted data, send me processed (still encrypted) result
- I will decrypt the result and get my answer

Computation Under Encryption

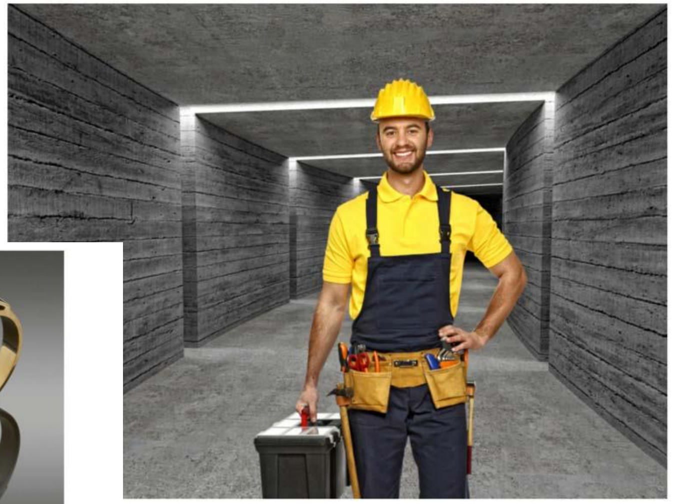
An Analogy: Alice's Jewelry Store

- Alice's workers need to assemble raw materials into jewelry




An Analogy: Alice’s Jewelry Store

- Alice’s workers need to assemble raw materials into jewelry



An Analogy: Alice’s Jewelry Store

- Alice’s workers need to assemble raw materials into jewelry
- But Alice is worried about theft

An Analogy: Alice’s Jewelry Store

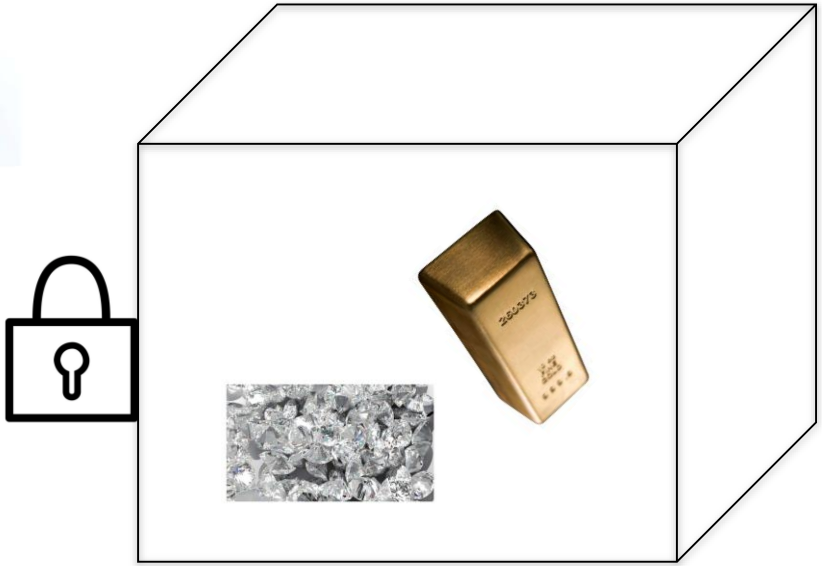
- Alice’s workers need to assemble raw materials into jewelry
- But Alice is worried about theft
  - How can the workers process the raw materials without having access to them?




An Analogy: Alice's Jewelry Store

- Alice puts materials in locked glove box
  - For which only she has the key

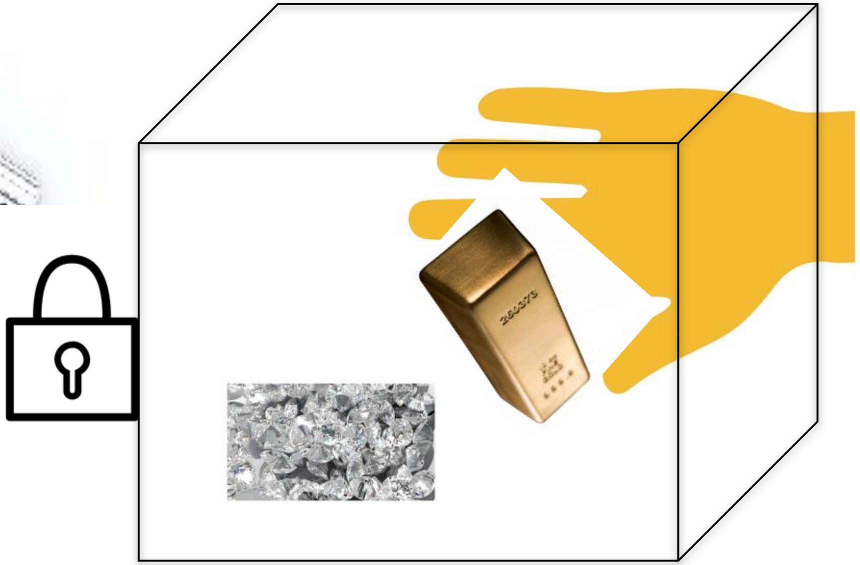





An Analogy: Alice's Jewelry Store

- Alice puts materials in locked glove box
  - For which only she has the key

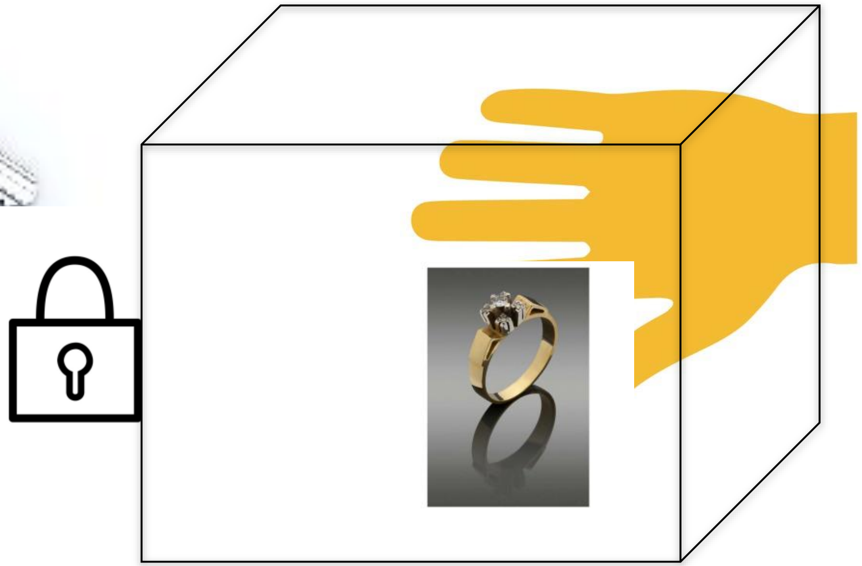
Workers assemble jewelry in the box



An Analogy: Alice's Jewelry Store

- Alice puts materials in locked glove box
  - For which only she has the key

Workers assemble jewelry in the box



An Analogy: Alice’s Jewelry Store

- Alice puts materials in locked glove box
  - For which only she has the key

Workers assemble jewelry in the box

Alice unlocks box to get “results”

The Analogy

Encrypt: putting things inside the box

- Alice does this using her key
- $c_i \leftarrow \mathrm{Enc}(m_i)$

Decrypt: Taking things out of the box

- Only Alice can do it, requires the key
- $m^* \leftarrow \mathrm{Dec}(c^*)$

Process: Assembling the jewelry

- Anyone can do it, computing on ciphertext
- $c^* \leftarrow \mathrm{Process}(c1,\ldots, cn)$

$m^* = \mathrm{Dec}(c^*)$ “the ring”, made from “raw material” m_i

Encrypted Computation

Encrypted computation can thus be achieved using Fully Homomorphic Encryption (FHE) **without trusting anything on the server side**

- Server does not need to store a secret key

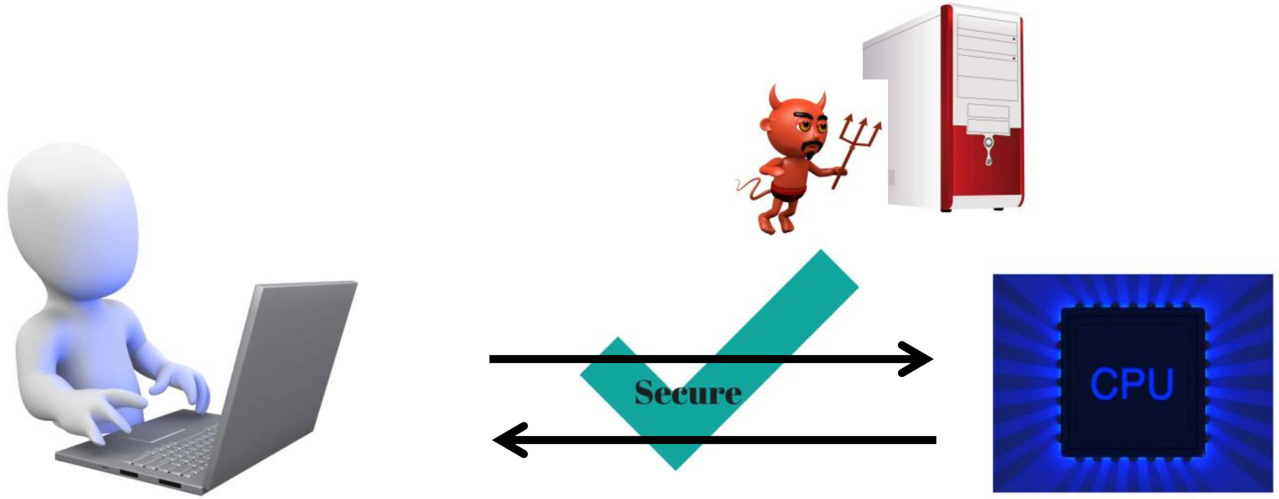
Unfortunately, FHE overheads are about 10^8 to 10^9 for straight-line code and overheads grow if there is complex control in the program

- Only usable for simple computations

What About Hardware Approaches?

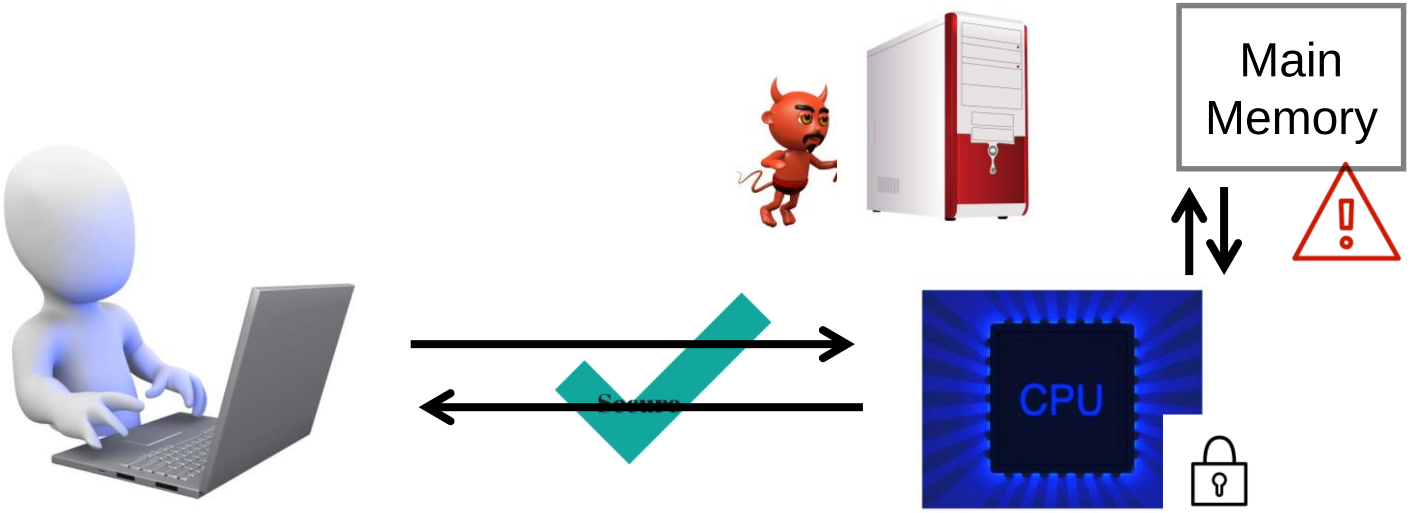
Tamper Resistant Hardware

- The secure processor is trusted, shares secret key with client.
- Private information stored in the hardware is not accessible through external means.
- examples: XOM, Aegis, TPM, TPM + TXT, etc.



Tamper Resistant Hardware Limitations

- Just trusting the tamper resistance of the chip not enough!
- I/O channels of the secure processor can be monitored by software and leak information
- Examples: address channel, I/O timing channel
- Malicious OS software can monitor these channels



Leakage through Address Channel

```
for i = 1 to N
    if (x == 0)
        sum += A[ i ]
    else
        sum += A[0]
```

Address sequence: 0x00, 0x01, 0x02

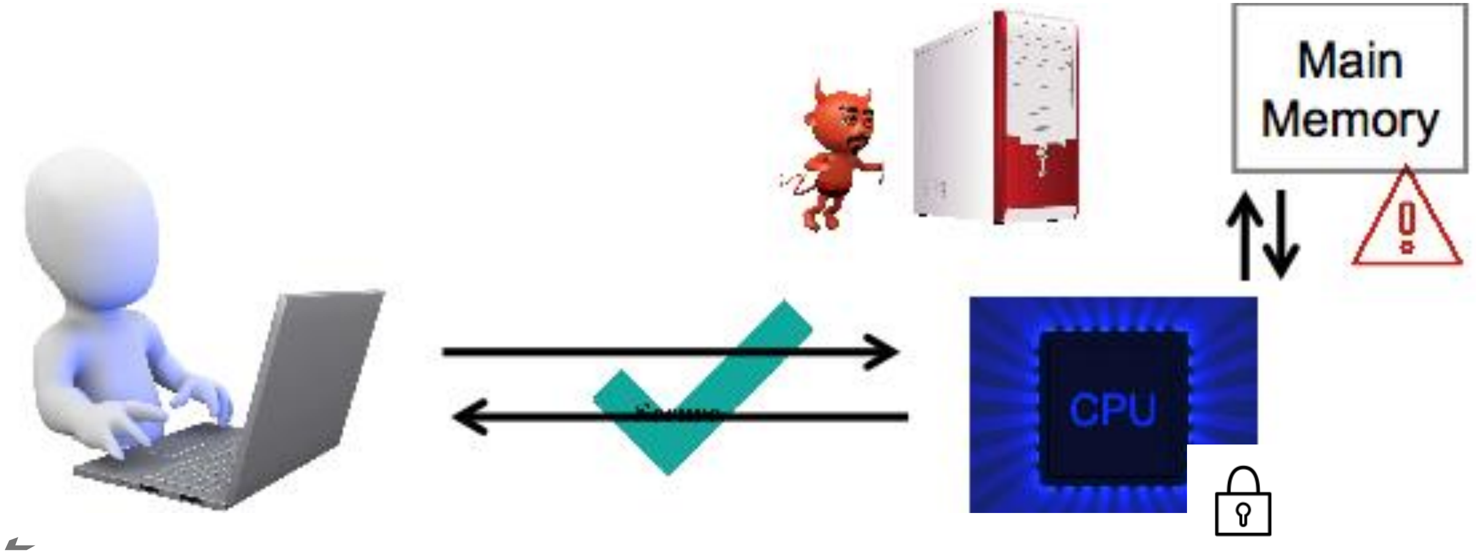
Address sequence: 0x00, 0x00, 0x00

- The value of x is leaked through the access pattern
- Sensitive data exposed by observing the addresses!

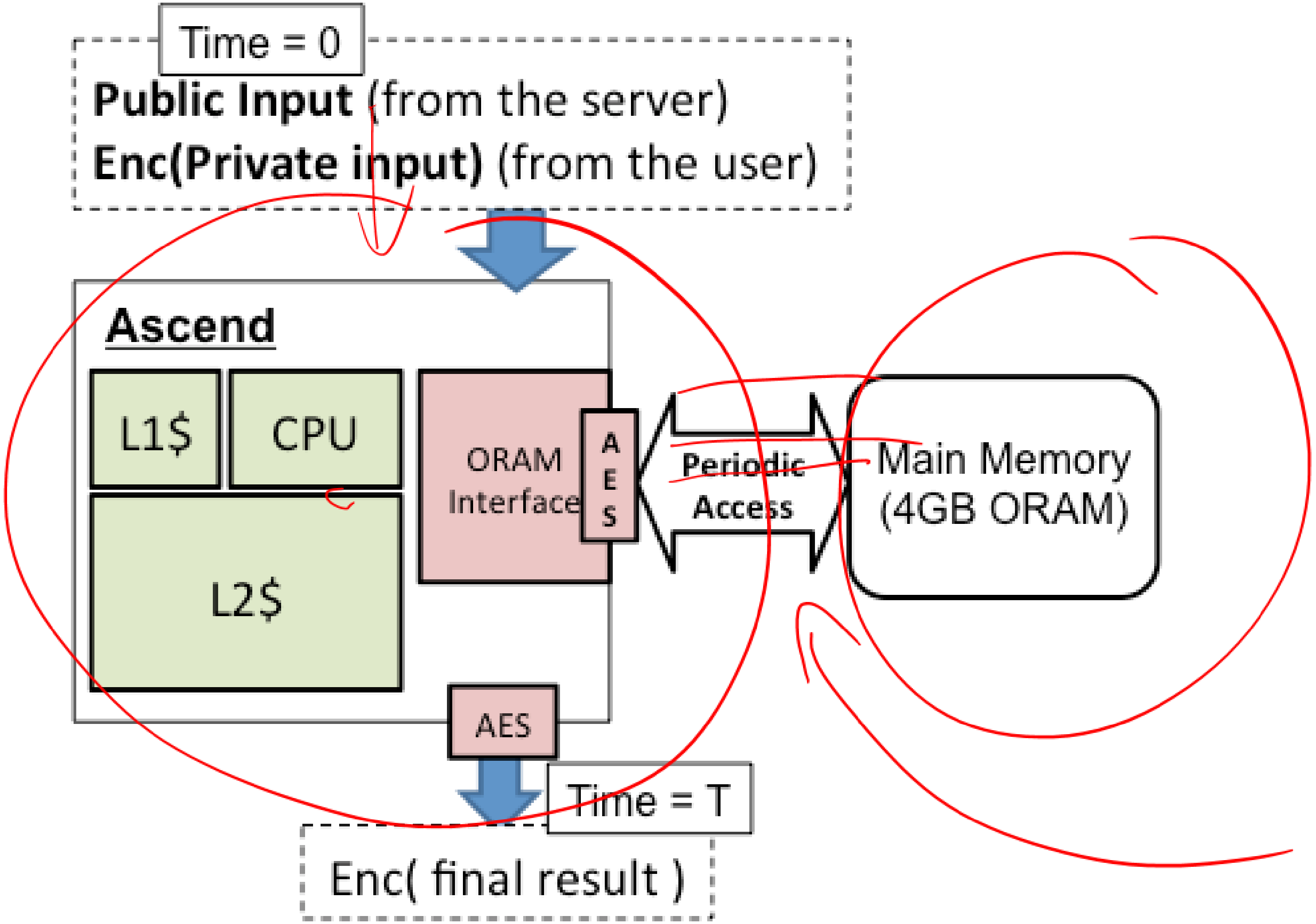
Ascend Processor Security Goal

Protect against all software-based and some hardware-based attacks when running untrusted software

An adversary cannot learn a user's private information by observing the pin traffic of Ascend.



Ascend Processor for IoT: Eliminate leakage over chip pins



Detection and Recovery

Integrity of Computation

Attacks on Integrity

Sometimes one is only concerned with obtaining correct results, not privacy leakage

Integrity of storage (malicious errors) implies reliability of storage (random errors)

- Solution: Cryptographic hash functions

Reliability and integrity of computation is a harder problem

- Errors can have catastrophic effects
- Many possible attacks on computation

Redundant Computations

Redundancy in the form of retries or parallel computations is key to recovery

Challenge is to keep overheads manageable → hardware can help

Key idea: Hardware computes confidence information for each computation

- Confidence low on data from an external source, high on data from trusted sources

Information Flow Tracking
Tracking Confidence

- Architect a processor to track the flow of information through the code
  - This can be done in software albeit with greater overhead
  - Worked well for buffer overflow attacks
  - Tracking “calculus” becomes more complicated under more sophisticated attacks

- Abort computation or retry when confidence falls below threshold

Summary

- Given just *one* example of each defensive strategy
- Mechanisms corresponding to different strategies have been developed for different layers, e.g., hardware, compiler, operating system
- A secure system may require mechanisms corresponding to *all* three defensive strategies at different layers

THANK YOU!

Srini Devadas

Webster Professor of Electrical Engineering and Computer Science

Computer Science and Artificial Intelligence Laboratory (CSAIL)
Massachusetts Institute of Technology