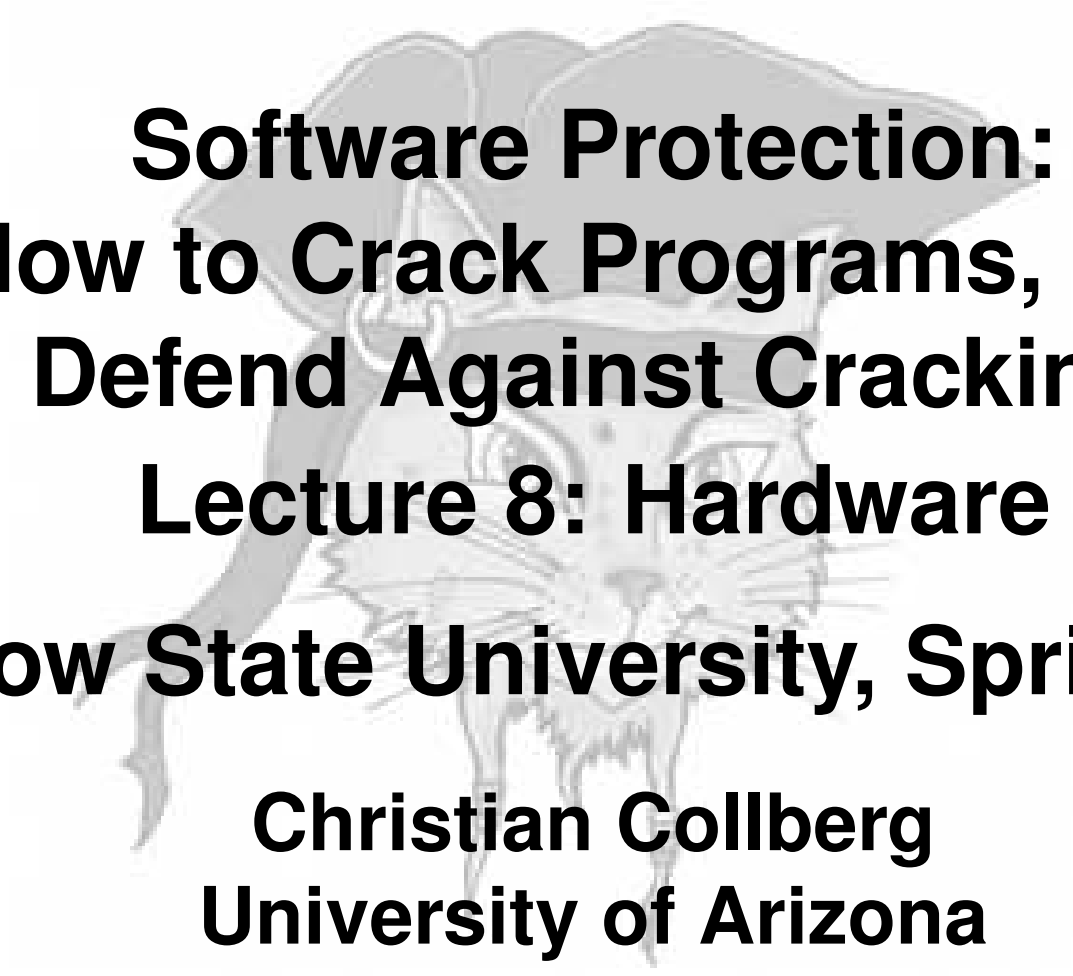


Software Protection: How to Crack Programs, and Defend Against Cracking

Lecture 8: Hardware

Moscow State University, Spring 2014

Christian Collberg
University of Arizona

www.cs.arizona.edu/~collberg

Introduction

This lecture

1. Disk and dongle protection against software and media piracy.
   - Disk protection — distribute the program on a physical medium, such as a CD and make the CD hard to copy.

This lecture

1. Disk and dongle protection against software and media piracy.
   - Disk protection — distribute the program on a physical medium, such as a CD and make the CD hard to copy.
   - Dongle protection — the program refuses to run without a hardware token distributed with the program.

This lecture...

2. Trusted Platform Module (TPM):
   - Small chip soldered to the motherboard of a PC
   - Used to boot a trusted environment
   - IBM/Lenovo ThinkPad ships with a TPM chips
   - not used in practice

This lecture...

3. Crypto-processor:
   - Can execute encrypted programs.
   - Contains encryption key.
   - Program is encrypted with processor's public key.
   - Only one computer can run the program!

This lecture. . .

4. Attacks on tamperproof hardware:
   - Physical attacks (shave off the top!)
   - Side channel attacks (“listen” to the chip!)

Dongles and Tokens

Dongles

- Dongles are hardware devices distributed with a program.

Dongles

- Dongles are hardware devices distributed with a program.
- Prevent piracy.

Dongles

- Dongles are hardware devices distributed with a program.
- Prevent piracy.
- The dongle must be connected to the user's computer,

Dongles

- Dongles are hardware devices distributed with a program.
- Prevent piracy.
- The dongle must be connected to the user's computer,
- The program periodically queries the dongle.

Dongles

- Dongles are hardware devices distributed with a program.
- Prevent piracy.
- The dongle must be connected to the user's computer,
- The program periodically queries the dongle.
- Is the dongle present? Genuine?

HASP

- Today dongles are active devices with built-in processors.

HASP

- Today dongles are active devices with built-in processors.
- Sometimes backed up by a battery.

```
class Dongle {
    private static long count = 3;
    private static long memory[]=new long[127];
    private static String password = "heyahaya";
    private static final long ID = 2376423;
    private static final long KEY = 0xab0ab012;
    private static java.util.Random rnd;

    public static final byte LOGIN = 0;
    public static final byte ISPRESENT = 1;
    public static final byte ENCODE = 2;
    public static final byte DECODE = 3;
    public static final byte READ = 4;
    public static final byte WRITE = 5;
    public static final byte GETID = 6;
    public static final byte GETTIME = 7;
    public static final long PRESENT = 0xca75ca75;
```

```
    static long call(byte operation,
                     String pw,
                     long arg1, long arg2) {
        if (!pw.equals(password) || (count<0)) return -1;
        switch (operation) {
            case LOGIN     : count--;
                             rnd=new java.util.Random(arg1)
                             return 0;
            case ISPRESENT : return PRESENT;
            case ENCODE    : return arg1^KEY;
            case DECODE    : return arg1^KEY;
            case READ      : return memory[(int)arg1];
            case WRITE     : memory[(int)arg1]=arg2;
                             return 0;
            case GETID     : return ID;
            case GETTIME   : return System.currentTimeMilli
            default        : return -1;
        }
    }
}
```

Aladin HASP

- The dongle has a small amount of internal memory

Aladin HASP

- The dongle has a small amount of internal memory
- a battery-backed internal realtime clock,

Aladin HASP

- The dongle has a small amount of internal memory
- a battery-backed internal realtime clock,
- an identifier unique to every dongle

Aladin HASP

- The dongle has a small amount of internal memory
- a battery-backed internal realtime clock,
- an identifier unique to every dongle
- a counter,

Aladin HASP

- The dongle has a small amount of internal memory
- a battery-backed internal realtime clock,
- an identifier unique to every dongle
- a counter,
- a password,

Aladin HASP

- The dongle has a small amount of internal memory
- a battery-backed internal realtime clock,
- an identifier unique to every dongle
- a counter,
- a password,
- a pseudo-random number generator

Aladin HASP

- The dongle has a small amount of internal memory
- a battery-backed internal realtime clock,
- an identifier unique to every dongle
- a counter,
- a password,
- a pseudo-random number generator
- an encryption engine with a hardcoded key.

```
public class Main {
   static String password = "heyahaya";
   static final long timeout = 1281964454908L;
   static final long seed = 260124545;
   static java.util.Random rnd;

   public static void main (String args[]) {
      Dongle.call(Dongle.LOGIN,password,seed,0);
      rnd = new java.util.Random(seed);
      if (Dongle.call(Dongle.GETTIME,
                      password,0,0)>timeout)
         System.exit(-1);
      ...
   }
}
```

Calling the dongle

```
if (Dongle.call(Dongle.ISPRESENT,password,0,0)!=
    Dongle.PRESENT) {
   System.err.println("No dongle present");
   System.exit(-1);
}
...
long here = Dongle.call(Dongle.ISPRESENT,password,0,0);
...
boolean OK = here == Dongle.PRESENT;
...
if (!OK) System.exit(-1);
```

- Obfuscate the challenge calls such that automatic removal becomes difficult.

Add bogus calls!

```
Dongle.call((byte)42,password,secret1,secret2);
```

- Sprinkle bogus calls to the dongle all over your code!

Problem with Dongles

- Contribute to the cost of distribution.
- If they contain a battery they will eventually have to be replaced.
- If they are lost, the owner will have to request a new copy.
- $\Rightarrow$ Today are only used to protect very expensive programs.

http://www.nodongle.com

We can make Emulators for any protection type, Hasp4, HaspHL, Sentinel Super Pro, Aladdin Hardlock, . . . , any protection.

Our emulators are 100% perfect, 100% guaranteed. 100% private,. . .

Dongle Emulator is a software to allow your program to run without any key attached

We dont have fixed prices, our prices depends [sic] on protection, not program price dependent, this will be very cheap if compared with program price...

Send a email with some infos about your program, like name and protection, so we can give you a personalized answer.

Authenticated boot using a trusted platform module

Algorithm: Authenticated boot

- We've assumed the program executes in an untrusted environment. The adversary can
  1. examine (reverse engineer),
  2. modify (tamper),
  3. copy (pirate),

  our program.

Algorithm: Authenticated boot

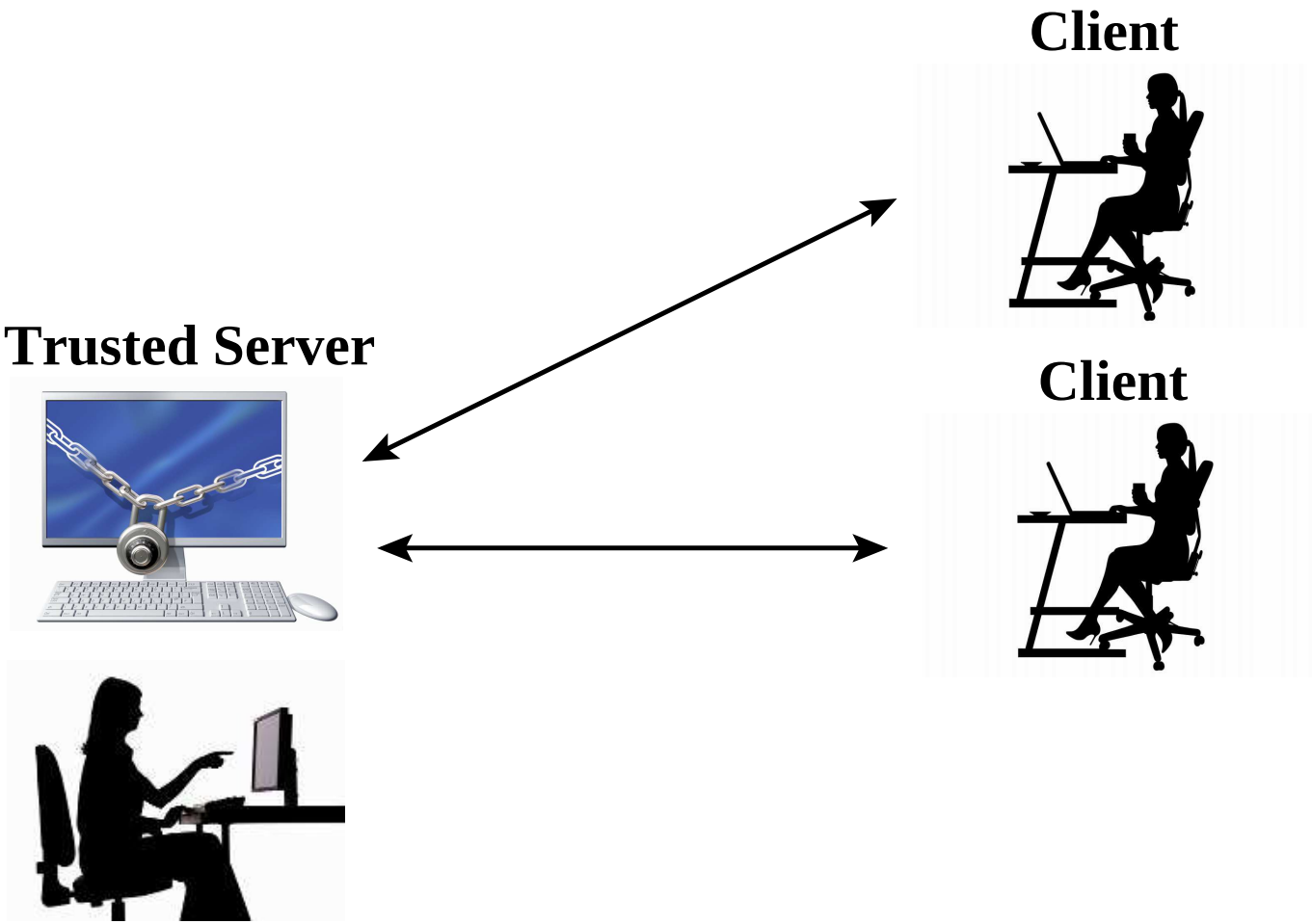
- We've assumed the program executes in an untrusted environment. The adversary can
  1. examine (reverse engineer),
  2. modify (tamper),
  3. copy (pirate),

  our program.
- Our techniques have been software based:
  1. obfuscation (to make a program harder to examine),
  2. tamperproofing (to make modification harder),
  3. watermarking (to trace copying).

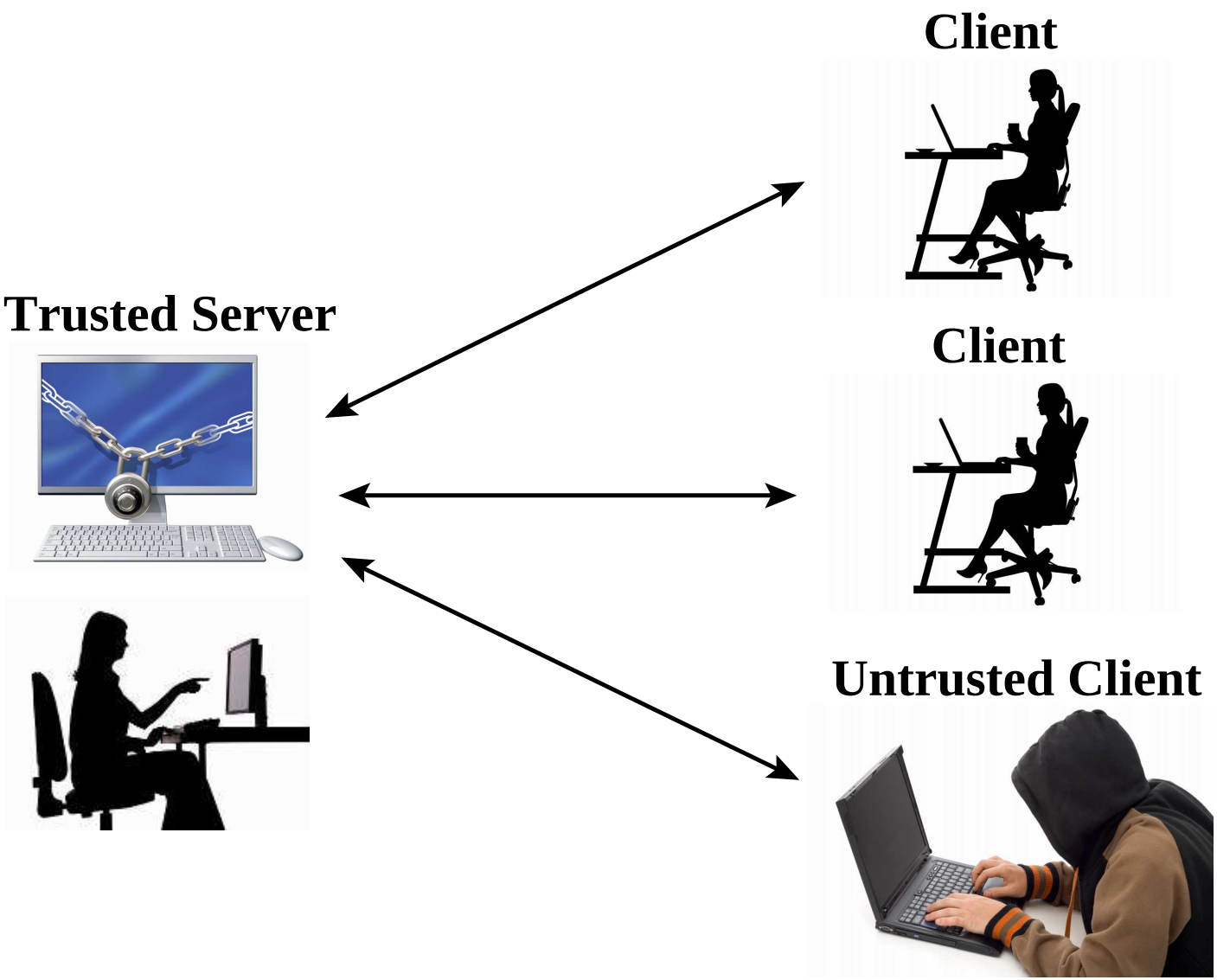
Algorithm: Authenticated boot

- What if we could trust the client to run trusted
  1. hardware,
  2. operating system,
  3. applications?

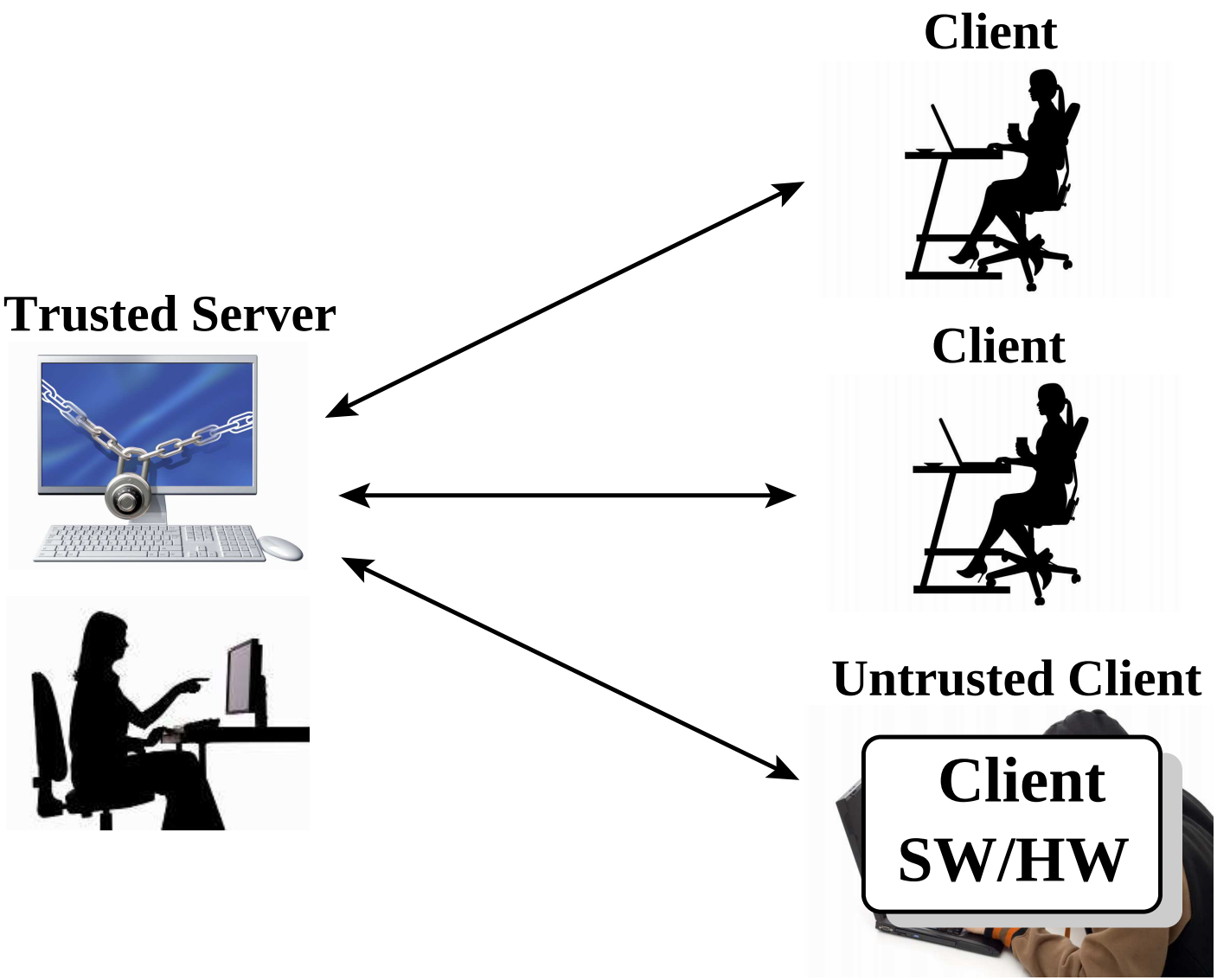
Trusted Platform Boot



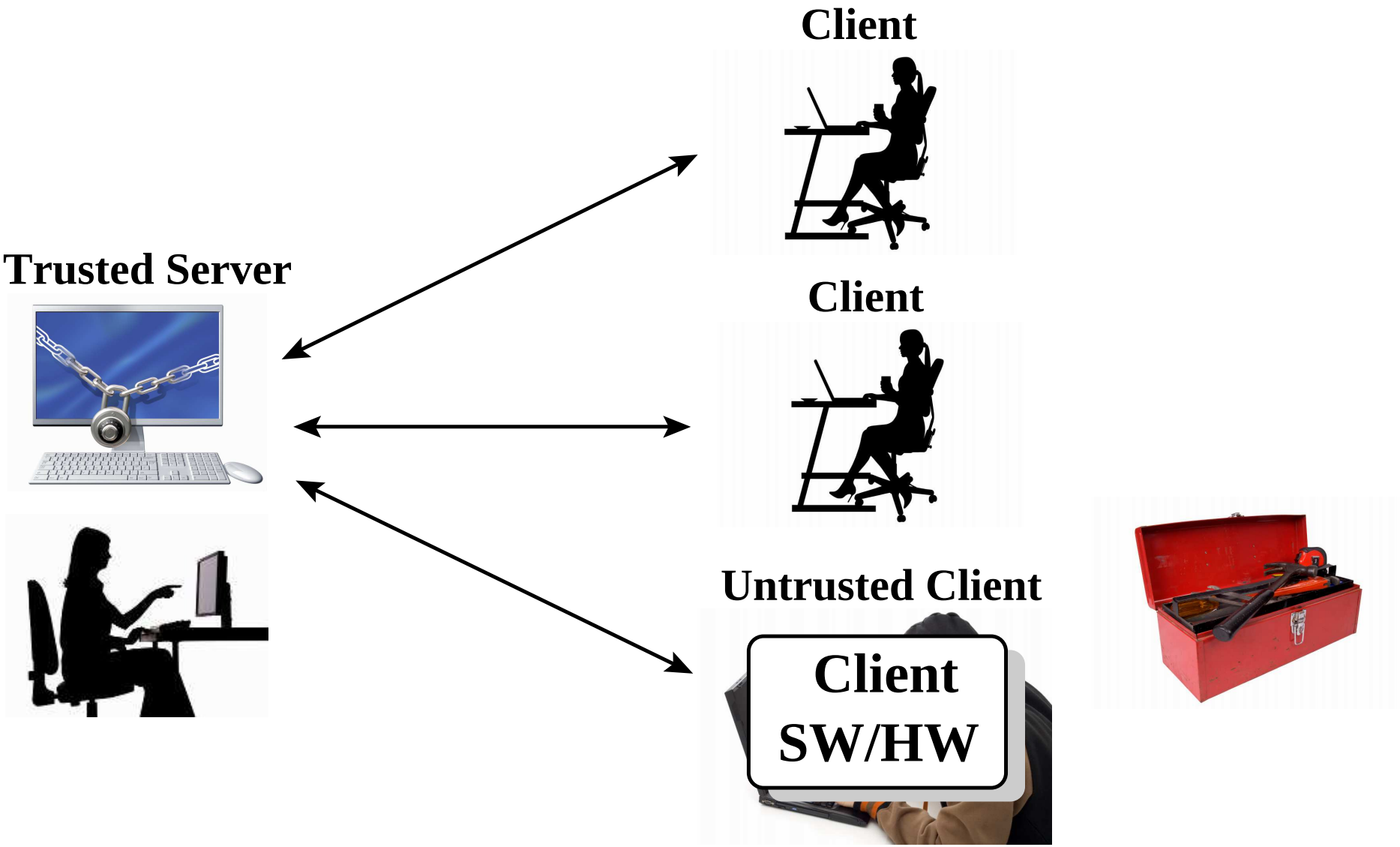
Trusted Platform Boot



Trusted Platform Boot



Trusted Platform Boot



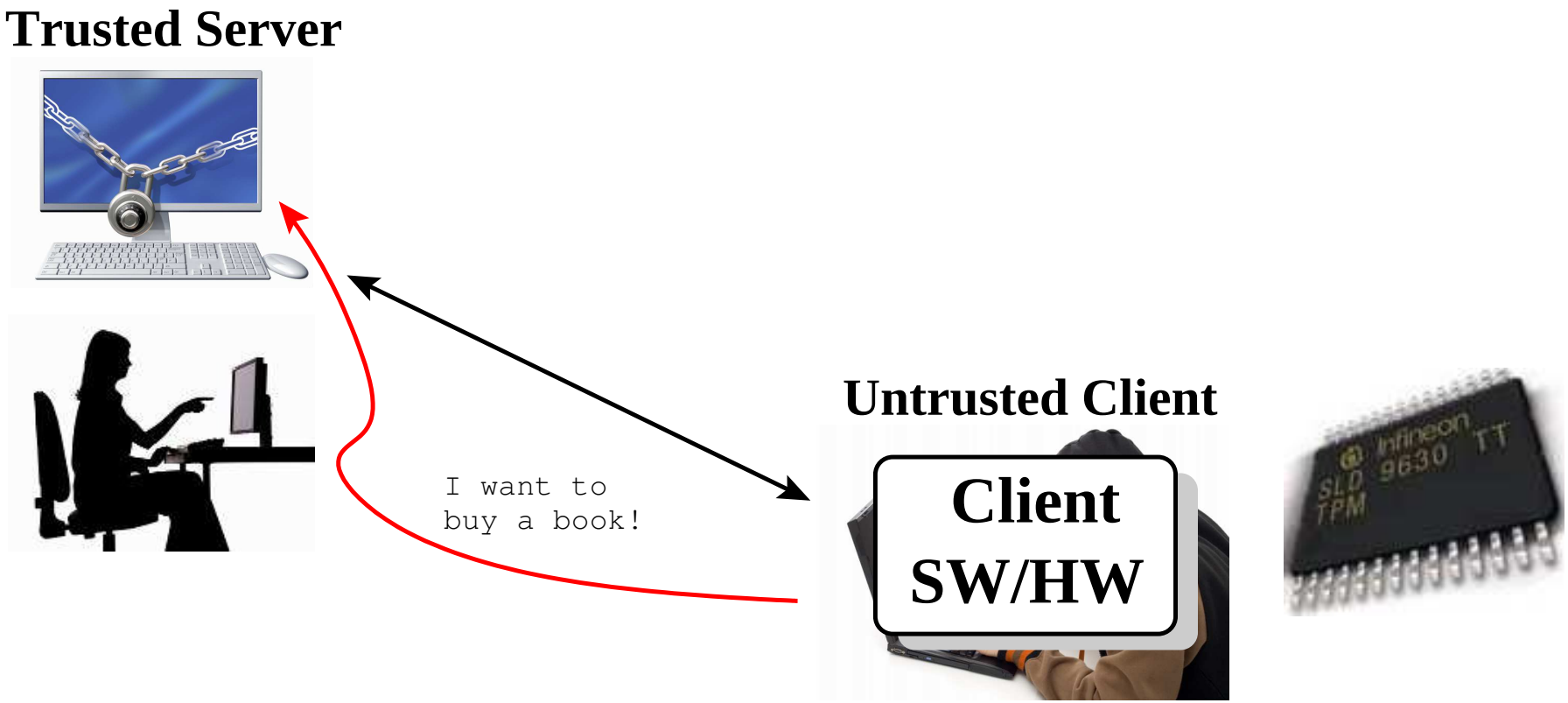
Trusted Platform Boot

Trusted Server

Untrusted Client

SHA 1

Trusted Platform Boot



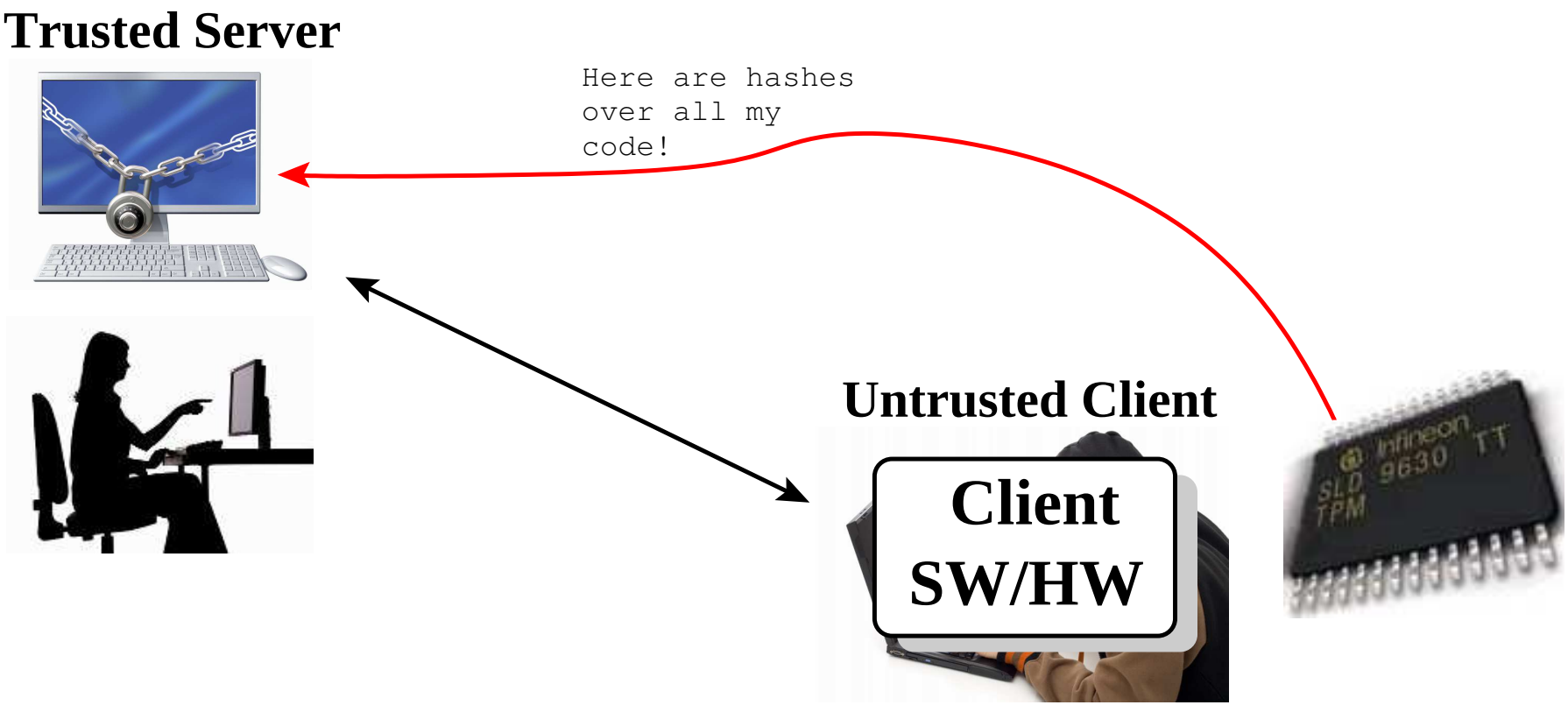
Trusted Platform Boot

Prove you are safe!

Trusted Server

Untrusted Client

Trusted Platform Boot



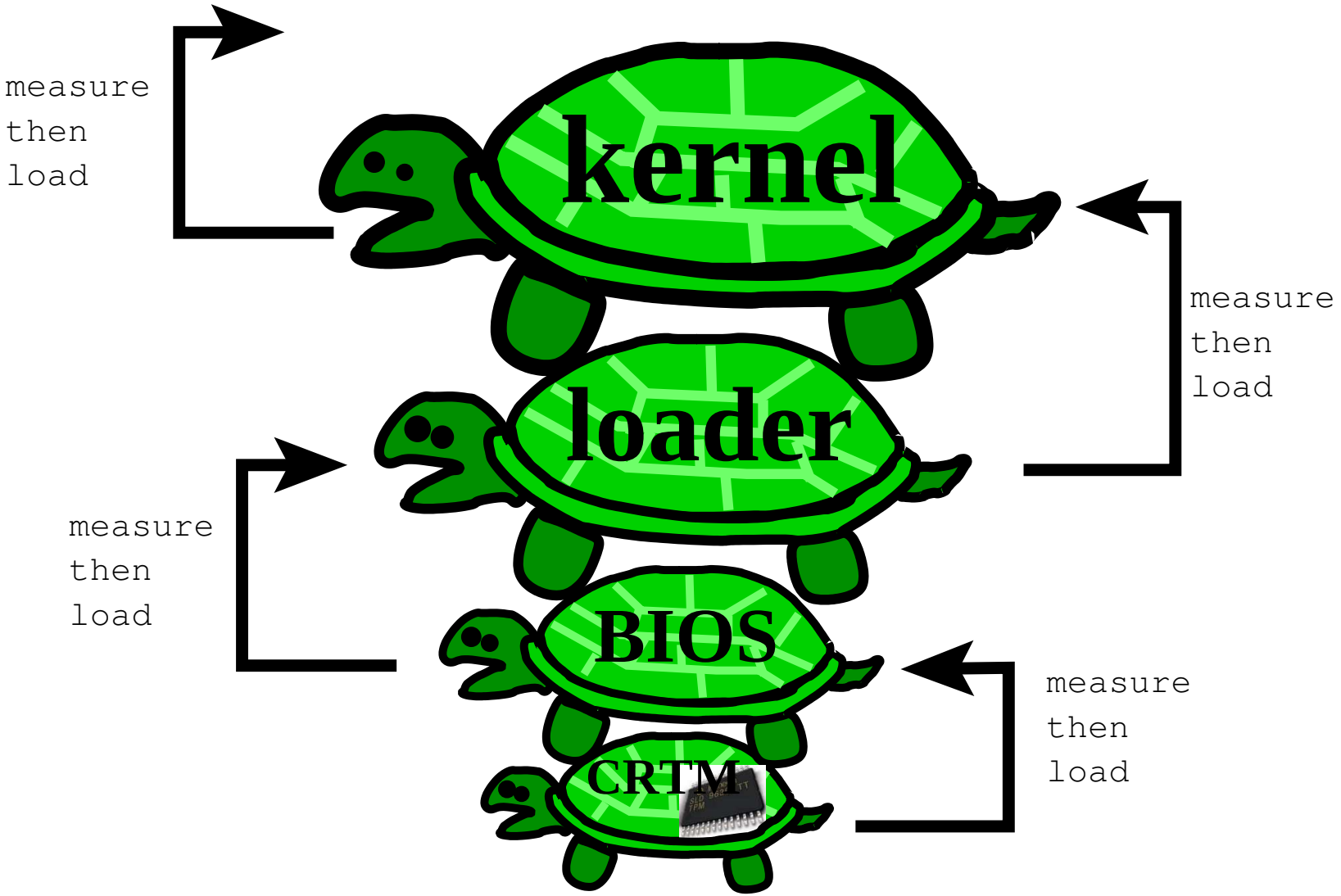
Algorithm: Authenticated boot

- Before you agree to communicate with a system you ask it to prove to you that it won't do anything bad.

Algorithm: Authenticated boot

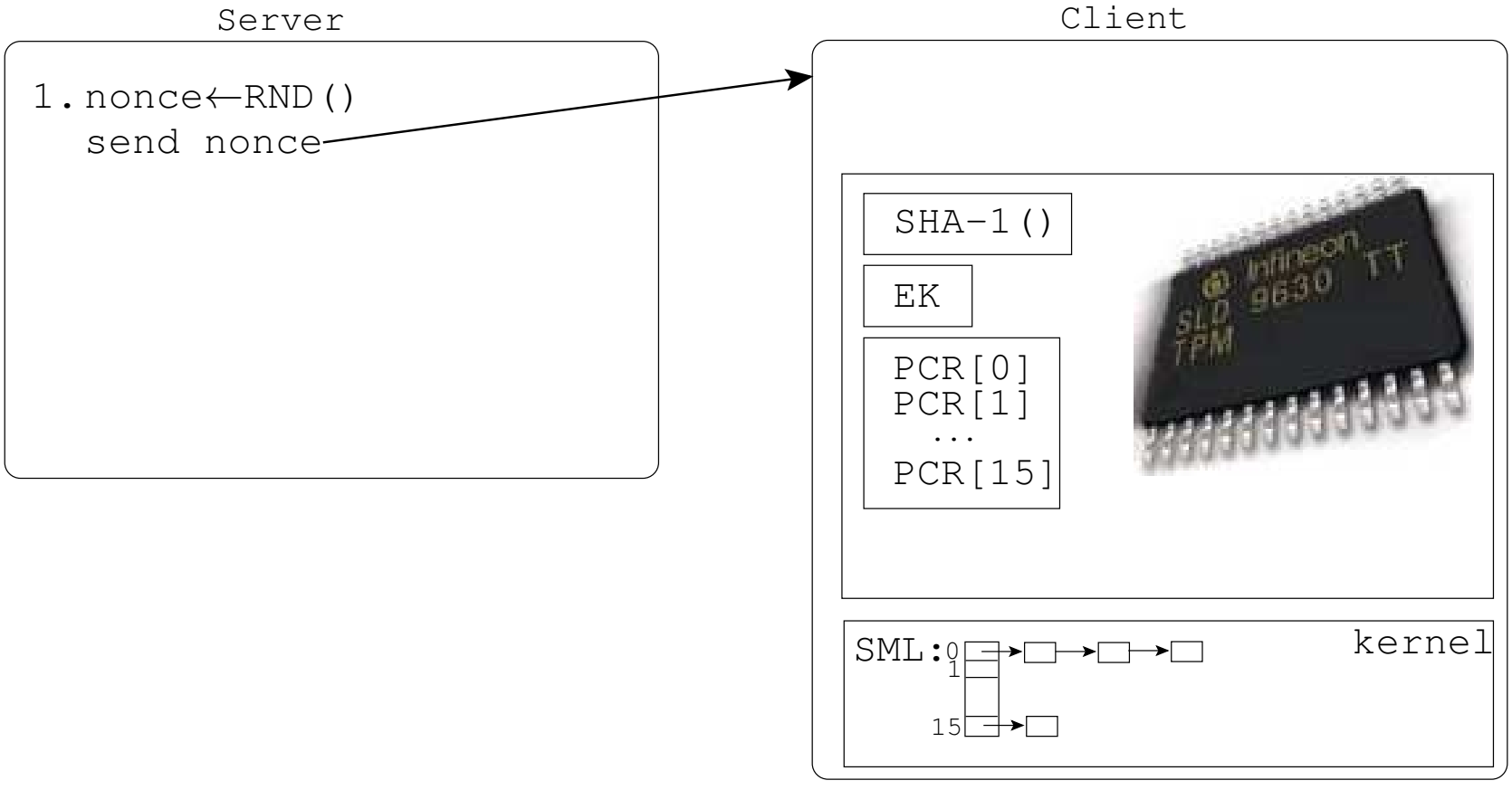
- Before you agree to communicate with a system you ask it to prove to you that it won't do anything bad.
- *Anything* that's running on Bob's computer could, potentially, affect whether you should trust it:
  1. OS,
  2. BIOS,
  3. bootloader,
  4. application programs,
  5. firmware, . . .

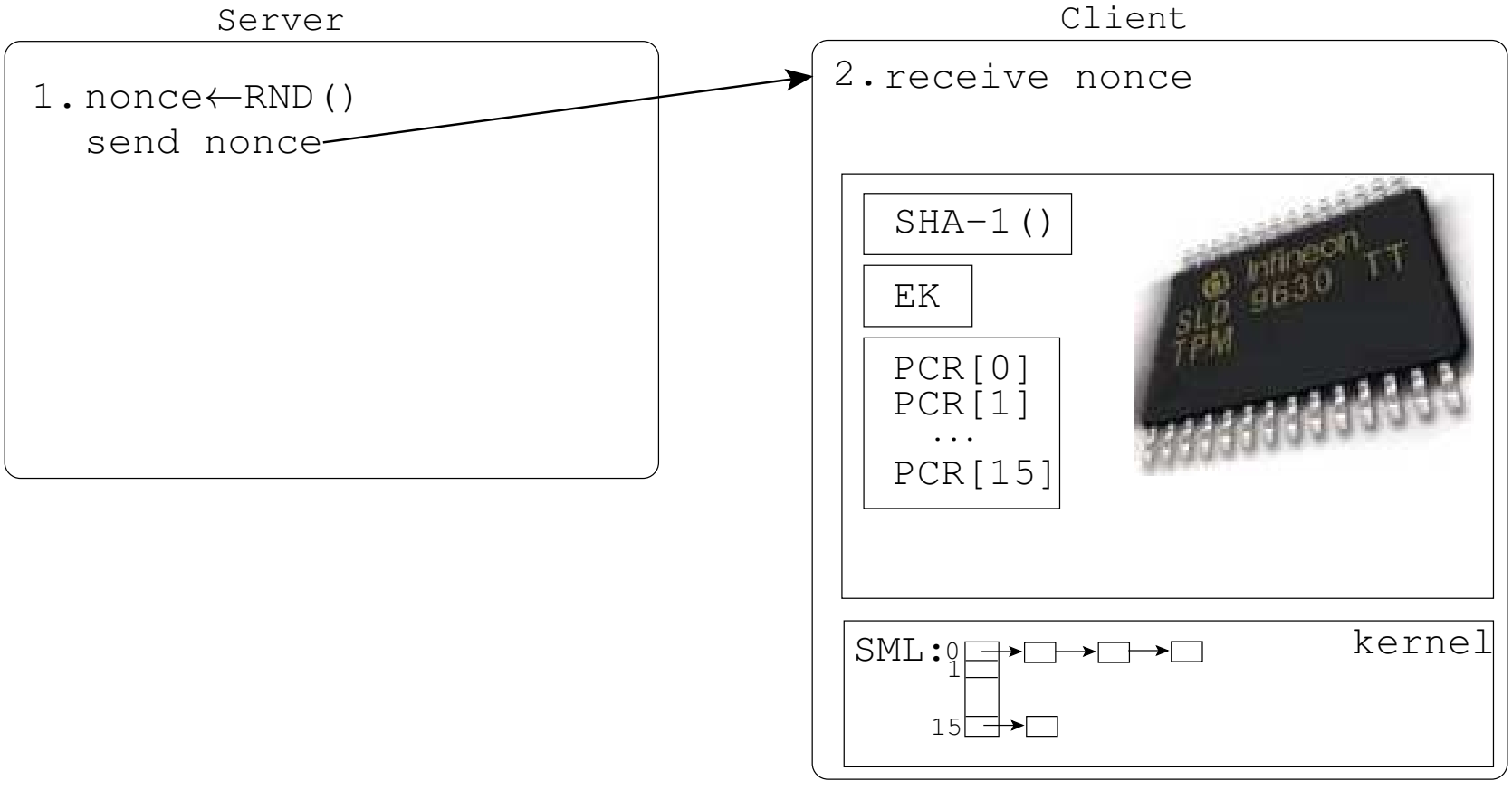
Algorithm: Authenticated boot



Secure boot vs. Authenticated boot

- Secure boot: only ever boot a system consisting of code that you trust.
- Authenticated boot:
  - Bob can boot whatever system he wants!
  - But, he cannot lie about what system he's booted!


Server
1. nonce←RND()
send nonce
Client
SHA-1()
EK
PCR[0]
PCR[1]
...
PCR[15]
Infineon
SLD 9630 TT
TPM
SML:
0
1
15
kernel


Server
1. nonce←RND()
send nonce
Client
2. receive nonce
SHA-1()
EK
PCR[0]
PCR[1]
...
PCR[15]
Infineon
SLD 9630 TT
TPM
SML:
0
1
15
kernel

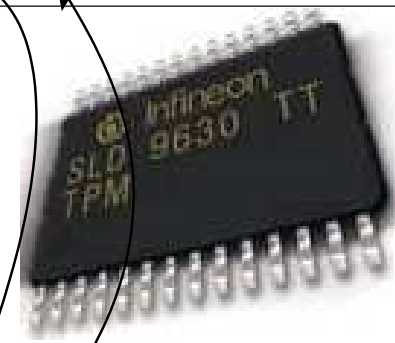
Server

1. nonce←RND()
   send nonce

Client

2. receive nonce
3. send (quote, SML)

SHA-1()



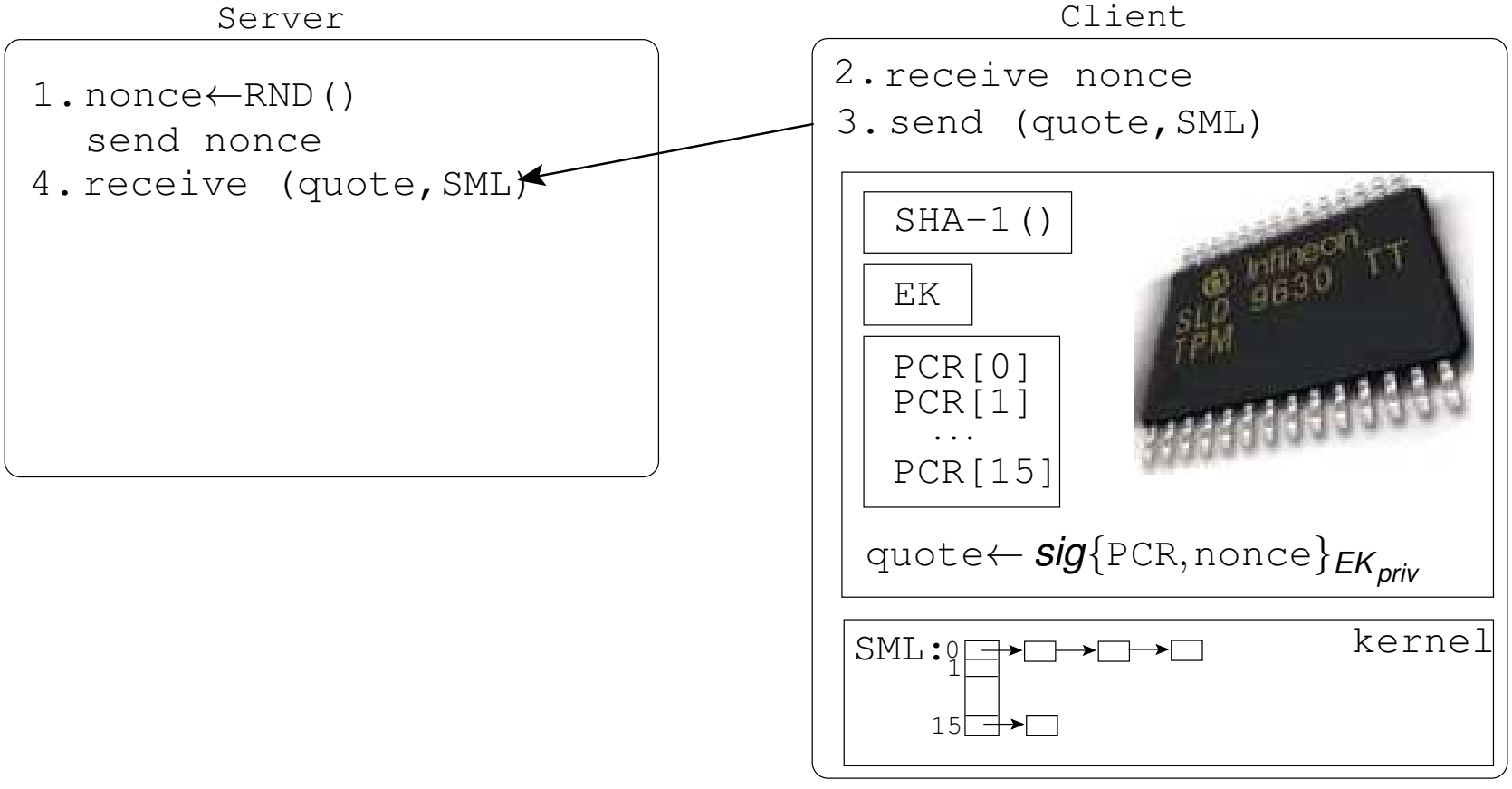
PCR[0]
PCR[1]

PCR[15]

quote← ***sig***{PCR, nonce}$_{EK_{priv}}$

SML: 0 1 15

kernel


Server
1. nonce←RND()
send nonce
4. receive (quote,SML)
Client
2. receive nonce
3. send (quote,SML)
SHA-1()
EK
PCR[0]
PCR[1]
...
PCR[15]
Infineon
SLD 9630 TT
TPM
quote← sig{PCR,nonce}EKpriv
SML:
0
1
15
kernel

Server

1. nonce←RND()
   send nonce
4. receive (quote,SML)
5. *cert*(EK_{pub}) OK?

Client

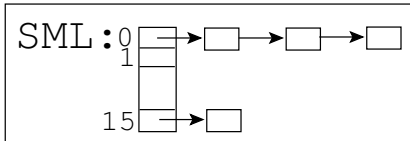
2. receive nonce
3. send (quote,SML)

SHA-1()

EK

PCR[0]
PCR[1]
...
PCR[15]

quote← *sig*{PCR,nonce}$_{EK_{priv}}$



Server

1. nonce←RND()
   send nonce
4. receive (quote,SML)
5. *cert*(EK_{pub}) OK?
6. *sig*{PCR,nonce2}$_{EK_{priv}}$ valid?

Client

2. receive nonce
3. send (quote,SML)

SHA-1()

EK

PCR[0]
PCR[1]
...
PCR[15]

quote← *sig*{PCR,nonce}$_{EK_{priv}}$

SML: 0 1 15

kernel

Server

1. nonce←RND()
   send nonce
4. receive (quote,SML)
5. *cert*(EK_{pub}) OK?
6. *sig*{PCR,nonce2}$_{EK_{priv}}$ valid?
7. nonce2 fresh?
   SML not tampered?
   measurements OK?

Client

2. receive nonce
3. send (quote,SML)

SHA-1()

EK

PCR[0]
PCR[1]
...
PCR[15]

quote← *sig*{PCR,nonce}$_{EK_{priv}}$

SML: 0 1 15

kernel

IBM's implementation

- Database of measurements for Redhat Fedora: 25000 measurements.
- A typical `SML`: 700-1000 measurements.

IBM's implementation

- Database of measurements for Redhat Fedora: 25000 measurements.
- A typical `SML`: 700-1000 measurements.
- How to collect "good" measurements:
  1. boot a "trusted system"
  2. measure all modules, config files, scripts $\Rightarrow$ whitelist of hashes

IBM's implementation

- Database of measurements for Redhat Fedora: 25000 measurements.
- A typical `SML`: 700-1000 measurements.
- How to collect "good" measurements:
  1. boot a "trusted system"
  2. measure all modules, config files, scripts $\Rightarrow$ whitelist of hashes
- How to collect "bad" measurements:
  1. boot a compromised system (root kits, trojans, ...)
  2. measure infected files $\Rightarrow$ blacklist of hashes
- How do we keep these lists up-to-date?!?!

Sealing

- Disney encrypts Nemo with a special sealing key *Seal*.

Sealing

- Disney encrypts Nemo with a special sealing key *Seal*.
- *Seal* depends on
  - the values in the `PCR`s
  - the TPM itself.

  $\Rightarrow$ your friend with an identical computer can't watch your copy of Nemo!

Sealing

- Disney encrypts Nemo with a special sealing key *Seal*.
- *Seal* depends on
  - the values in the `PCR`s
  - the TPM itself.

  $\Rightarrow$ your friend with an identical computer can't watch your copy of Nemo!
- If you reboot with slightly hacked OS
  - the `PCR`s will have changed
  - $\Rightarrow$ the *Seal* will be different
  - $\Rightarrow$ you can't decrypt Nemo!

Sealing

- Disney encrypts Nemo with a special sealing key *Seal*.
- *Seal* depends on
  - the values in the `PCR`s
  - the TPM itself.

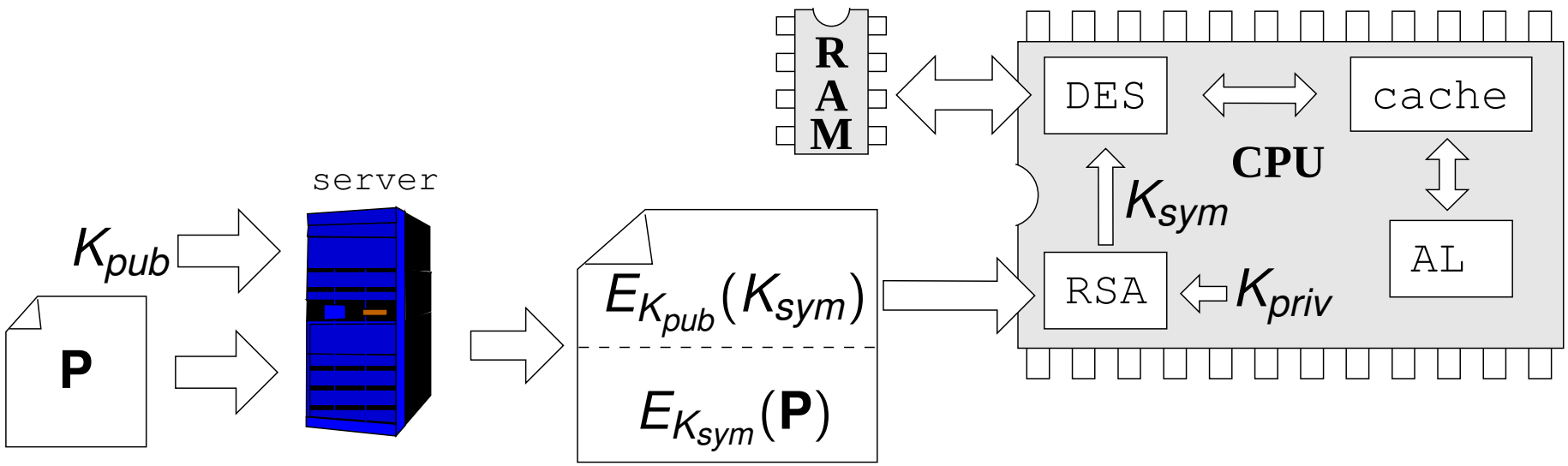
  $\Rightarrow$ your friend with an identical computer can't watch your copy of Nemo!
- If you reboot with slightly hacked OS
  - the `PCR`s will have changed
  - $\Rightarrow$ the *Seal* will be different
  - $\Rightarrow$ you can't decrypt Nemo!
- Microsoft could do the same to protect Office.

Encrypted execution

Encrypted execution

- Idea:
  - Encrypt the program.
  - Keep a (unique) secret key in the CPU.
  - Decrypt inside the CPU.
- Protect algorithms (privacy)!
- Protect from tampering (integrity)!
- Protect from cloning (piracy)!
- Assume the CPU cannot be tampered with.

XOM Overview



The XOM architecture

- Stanford design.
- Never implemented in silicon.
- Simulated in software.
- Operating system, *XOMOS*, runs on top of it.

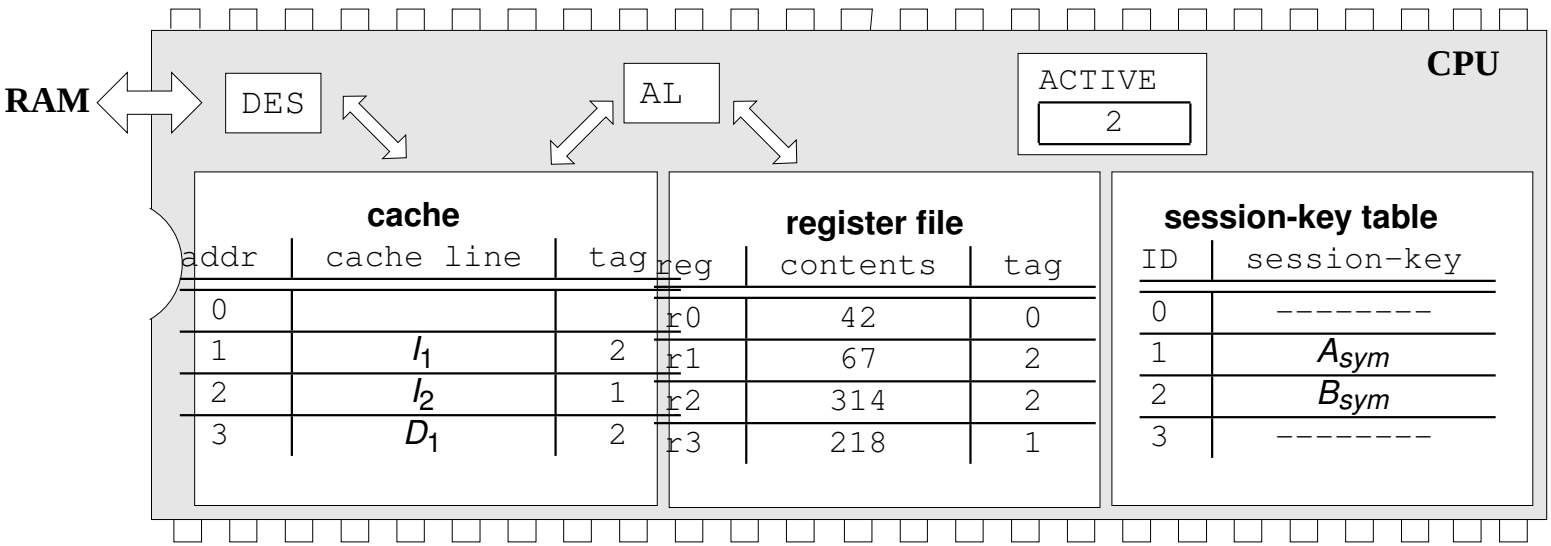
The XOM architecture — Compartments

- Each process may run in different security mode.
- Encrypted programs are slow!
- Programs may switch between encrypted and cleartext execution.
- CPU has 4 Compartments:
  - logical containers
  - protect one process from being observed or modified by another process.
  - the OS is untrusted: runs in its own compartment!

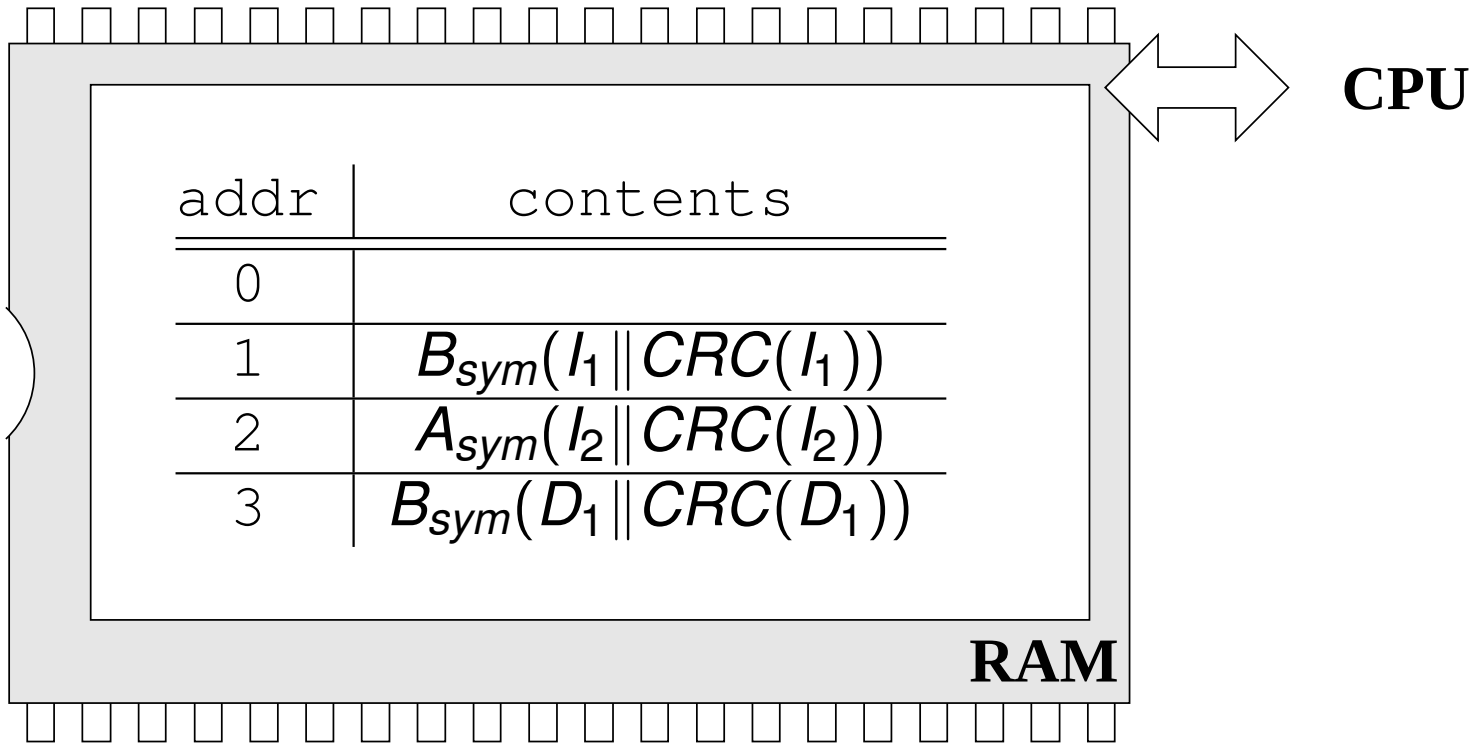
The XOM architecture — Compartments

- Compartment 0: code runs unencrypted
- ACTIVE register: current executing compartment
- Session key table: Maps compartment ID to key.
- Each register is tagged with compartment key.
- Each cache line is tagged with compartment key.
- On-chip data is in cleartext.
- On cache flush: encrypt!

The XOM architecture — Example



The XOM architecture — Example



The XOM architecture — Example

The CPU tries to load data value D_1 at address 3 into register r0:

1. Look in the cache line: empty!

The XOM architecture — Example

The CPU tries to load data value D_1 at address 3 into register r0:

1. Look in the cache line: empty!
2. Cache miss, read $B_{sym}(D_1 \| CRC(D_1))$ from address 3.

The XOM architecture — Example

The CPU tries to load data value D_1 at address 3 into register r0:

1. Look in the cache line: empty!
2. Cache miss, read $B_{sym}(D_1 \| CRC(D_1))$ from address 3.
3. Look up key for the active compartment: B_{sym}.

The XOM architecture — Example

The CPU tries to load data value D_1 at address 3 into register r0:

1. Look in the cache line: empty!
2. Cache miss, read $B_{sym}(D_1 \| CRC(D_1))$ from address 3.
3. Look up key for the active compartment: B_{sym}.
4. Decrypt the cache-line!

The XOM architecture — Example

5. Adversary could have swapped D_1 it for another encrypted value from some other part of the code!
   - Store CRC hash of each cache line.
   - If CRC doesn't match $\Rightarrow$ exception!
   - Otherwise, load D_1 into register `r0`
   - Set `r0`'s tag to 2.

Attacks on Tamperproof Devices

Attacks on Tamperproof Devices

- Assumption: Physical barrier isn't broken!
- This section:
  1. Attacking the XBOX
  2. Invasive attacks on smartcards
  3. Non-invasive attacks on smartcards

The Microsoft XBOX hack

- MIT Ph.D. student Andrew “bunnie” Huang got an XBOX game console from his fiance’ for Christmas.

The Microsoft XBOX hack

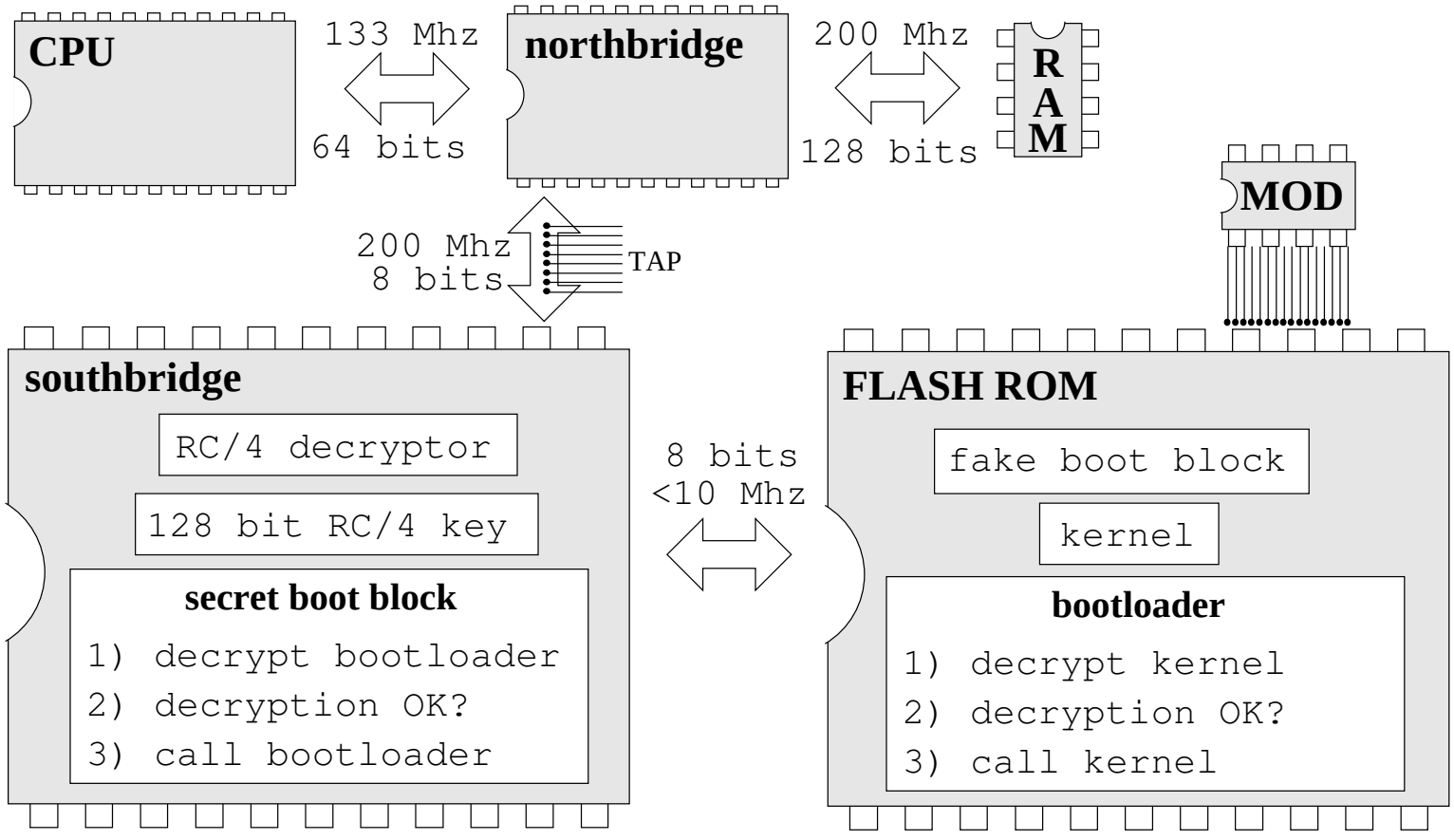
- MIT Ph.D. student Andrew “bunnie” Huang got an XBOX game console from his fiance’ for Christmas.

- He took it apart.

The Microsoft XBOX hack

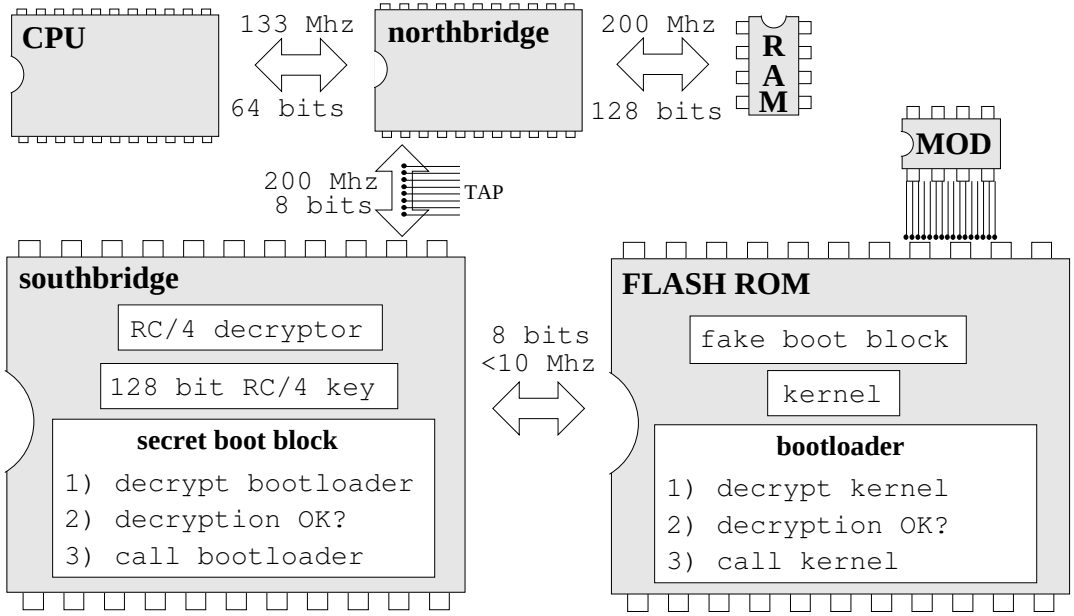
- MIT Ph.D. student Andrew “bunnie” Huang got an XBOX game console from his fiance’ for Christmas.

- He took it apart.
- He cracked the security.



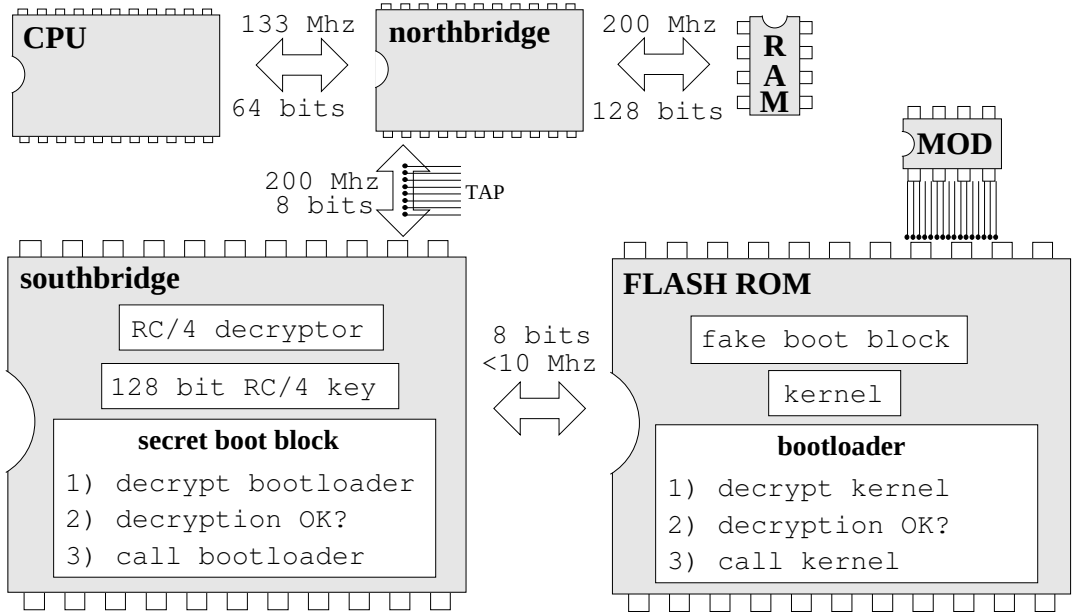
- X86 CPU, 64 MB of cheap RAM, northbridge and southbridge chips, IDE controllers, etc.

The Microsoft XBOX hack



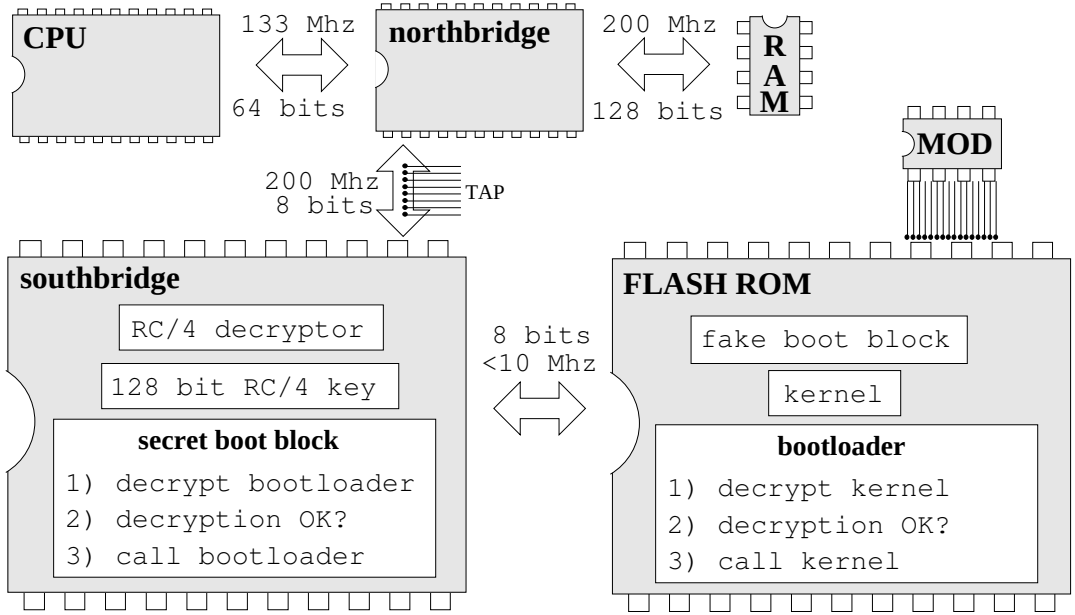
- System startup:
  1. boot block executes

The Microsoft XBOX hack



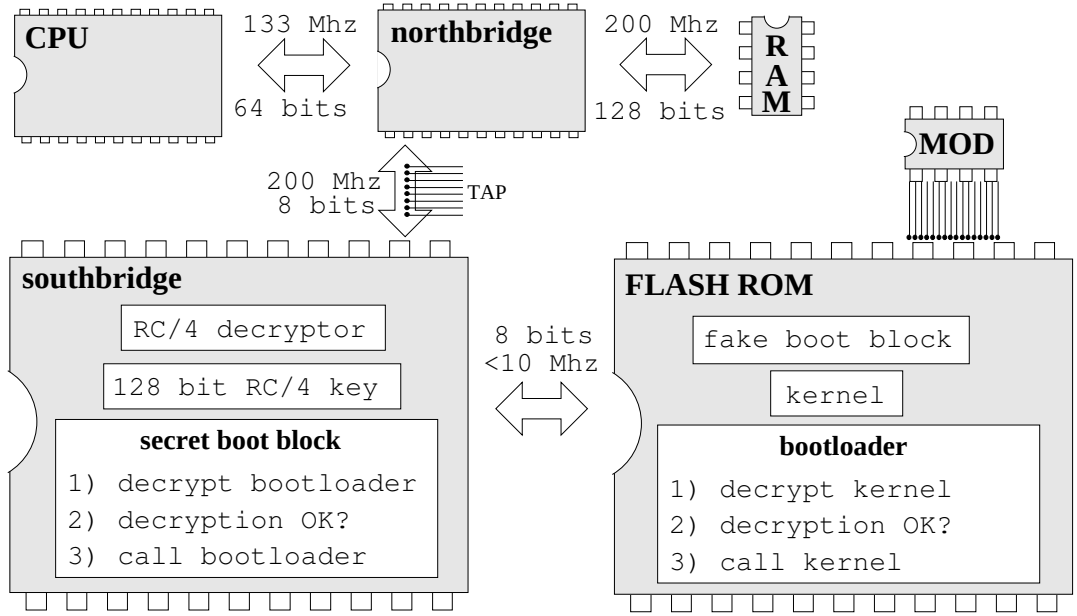
- System startup:
  1. boot block executes
  2. boot block decrypts, verifies, and jumps to the bootloader

The Microsoft XBOX hack



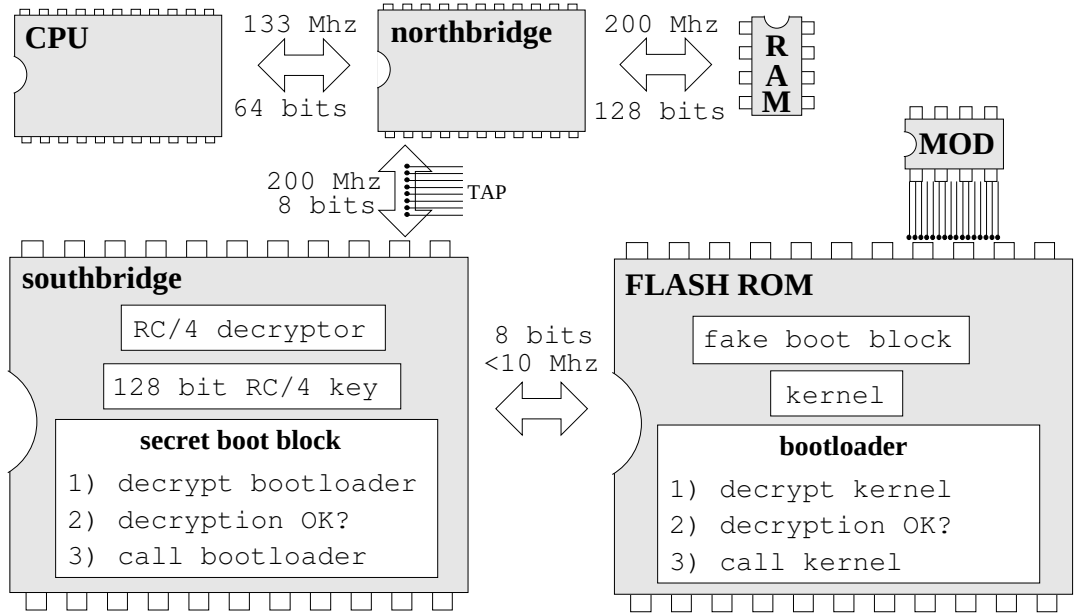
- System startup:
  1. boot block executes
  2. boot block decrypts, verifies, and jumps to the bootloader
  3. bootloader decrypts, verifies, and jumps to the OS kernel

The Microsoft XBOX hack



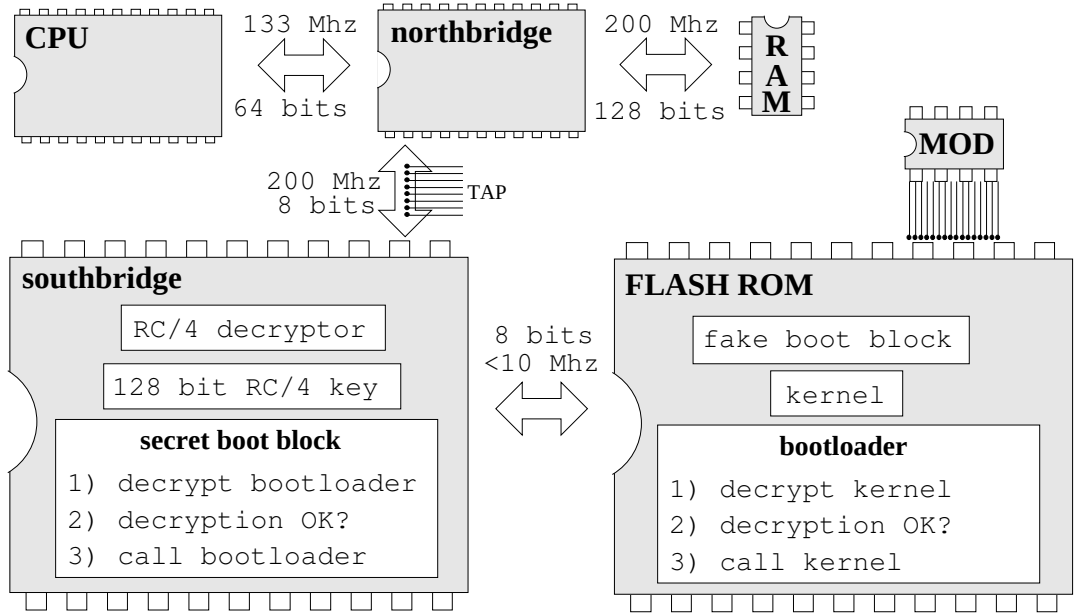
- Observations:
  1. key and decryptor in the southbridge.

The Microsoft XBOX hack



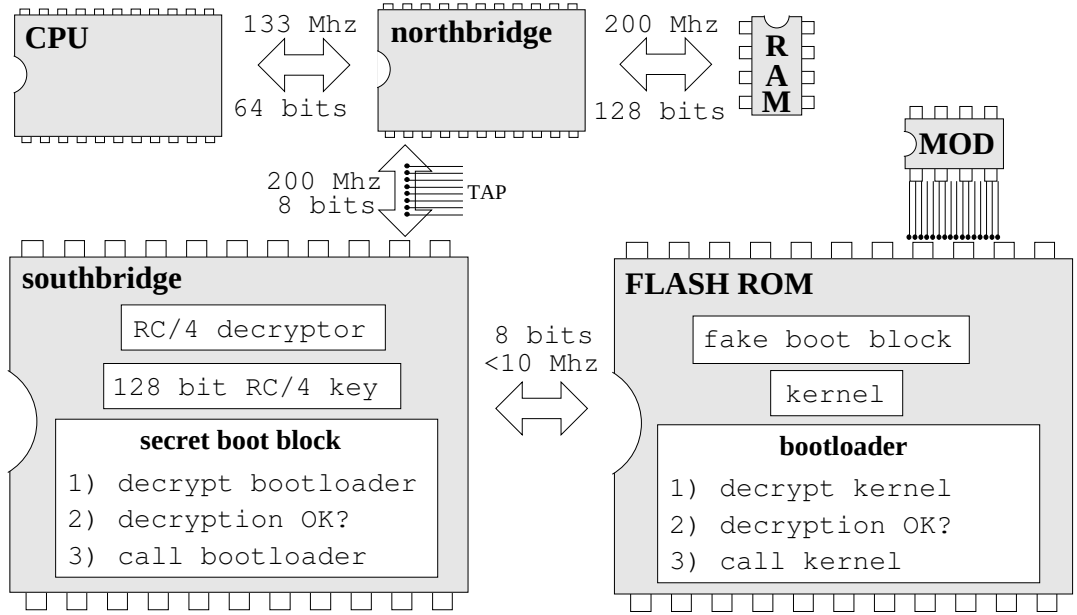
- Observations:
  1. key and decryptor in the southbridge.
  2. the CPU decrypts

The Microsoft XBOX hack



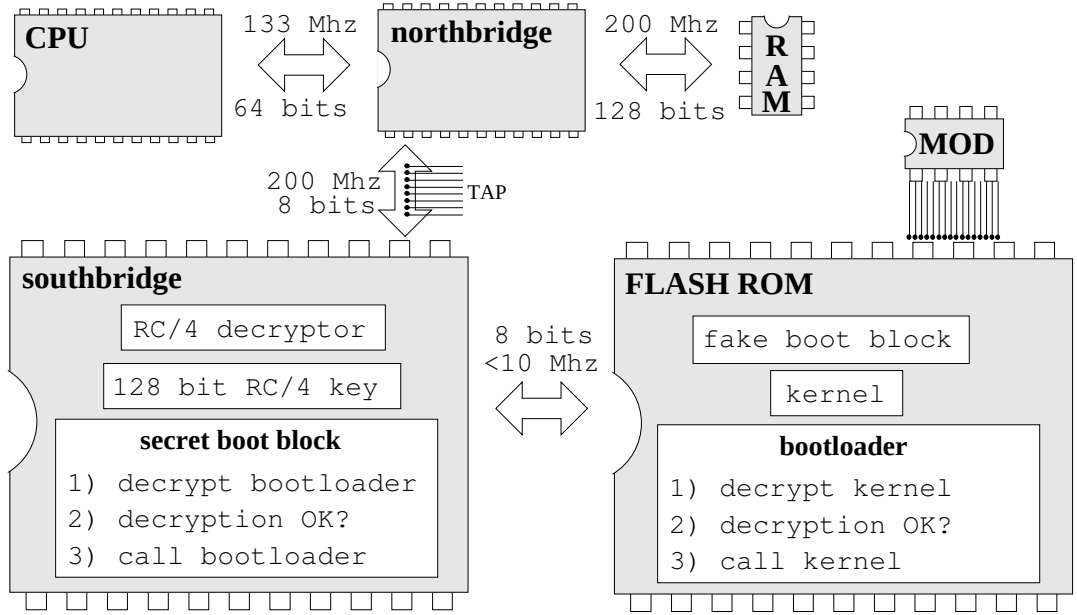
- Observations:
  1. key and decryptor in the southbridge.
  2. the CPU decrypts
  3. $\Rightarrow$ key/decryptor will travel in cleartext over two busses!

The Microsoft XBOX hack



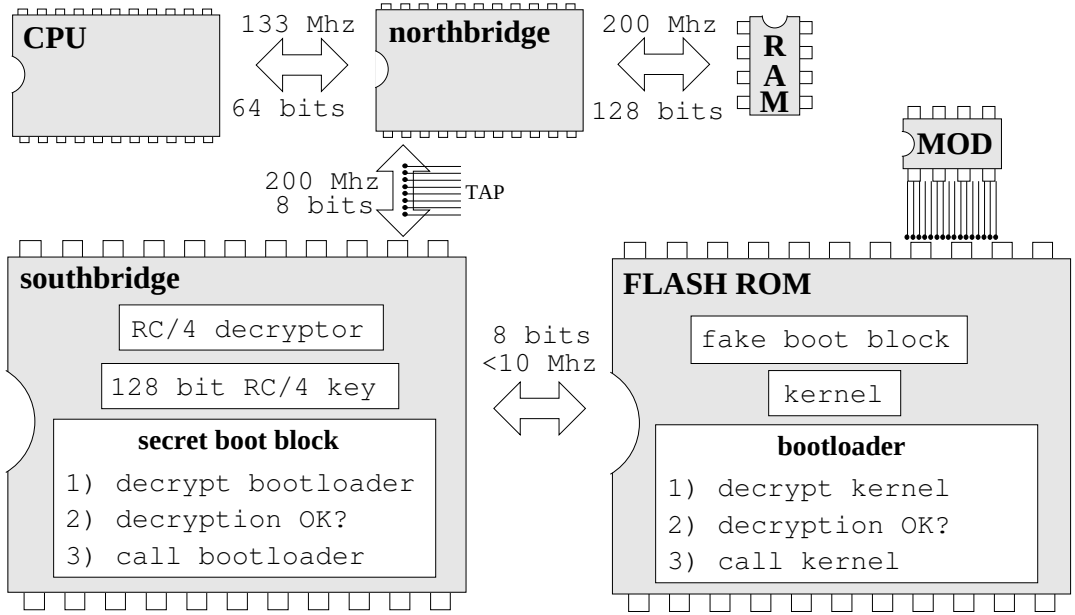
- Which bus to tap?
  1. north-southbridge bus: 0nly 8 bits wide!

The Microsoft XBOX hack



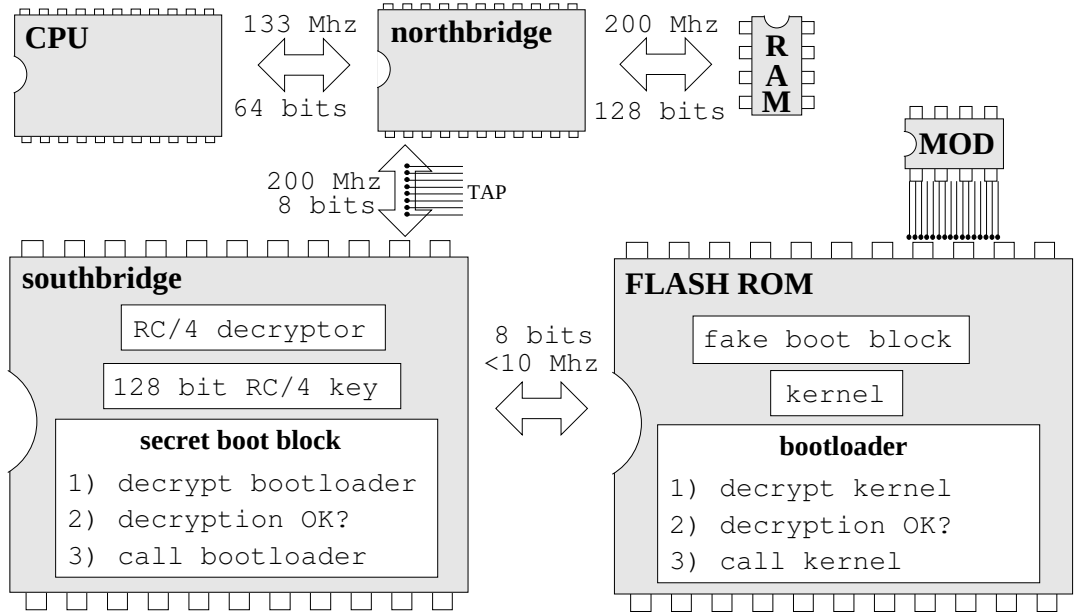
- Which bus to tap?
  1. north-southbridge bus: 0nly 8 bits wide!
  2. solder a tap board onto the bus

The Microsoft XBOX hack



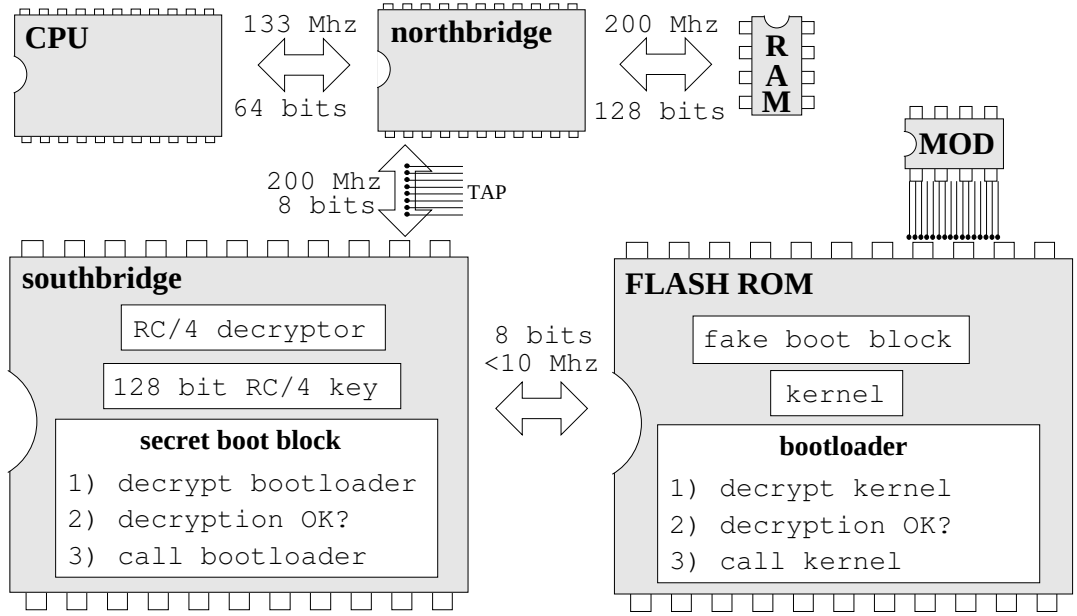
- Which bus to tap?
  1. north-southbridge bus: 0nly 8 bits wide!
  2. solder a tap board onto the bus
  3. sniff the decryptor + key

The Microsoft XBOX hack



- Which bus to tap?
  1. north-southbridge bus: 0nly 8 bits wide!
  2. solder a tap board onto the bus
  3. sniff the decryptor + key
  4. pattern match $\Rightarrow$ RC/4 decryptor!

The Microsoft XBOX hack



- Which bus to tap?
  1. north-southbridge bus: 0nly 8 bits wide!
  2. solder a tap board onto the bus
  3. sniff the decryptor + key
  4. pattern match $\Rightarrow$ RC/4 decryptor!
  5. try every 16 bytes from bootblock as key!

Hacking smartcards

Smartcards

- Mass transit, prepaid phone cards, identification cards, *SIM cards*, pay-TV set-top boxes, credit cards.
- Protected memory in which a secret can be stored.

Smartcards

- Gets power and clock from Card Acceptance Device (CAD).
- CAD and card communicate over 1-bit serial ylink.

Invasive vs. non-invasive attacks

- Invasive attack:
  1. expose the bare chip,

Invasive vs. non-invasive attacks

- Invasive attack:
  1. expose the bare chip,
  2. probe the surface to extract information

Invasive vs. non-invasive attacks

- Invasive attack:
  1. expose the bare chip,
  2. probe the surface to extract information
  3. poke the surface to modify the chip

Invasive vs. non-invasive attacks

- Invasive attack:
  1. expose the bare chip,
  2. probe the surface to extract information
  3. poke the surface to modify the chip
- Non-invasive attack:
  - monitor execution characteristics (power, radiation, execution time) etc.

Invasive vs. non-invasive attacks

- Invasive attack:
  1. expose the bare chip,
  2. probe the surface to extract information
  3. poke the surface to modify the chip
- Non-invasive attack:
  - monitor execution characteristics (power, radiation, execution time) etc.
  - watch normal operations or induce faults

Smartcards — Invasive attacks

Chipworks can extract analog or digital circuits from semiconductor devices and deliver detailed easy-to-understand schematics that document a single functional block or all the circuits. . . . We decapsulate the chip and analyze the die to locate the circuit blocks of interest. Then, using our Image Capture and Imaging System (ICIS) we generate mosiacs for each level of interconnect. Finally, advanced software and expertise is used to extract the circuits for analysis.

Smartcards — Invasive attacks — Depackaging

1. Remove the chip from the card itself by heating and bending it.
2. Remove the epoxy resin around the chip by dipping it in 60°C fuming nitric acid.
3. Clean the chip by washing it with acetone in an ultrasonic bath.
4. Mount the exposed chip in a test package and connect its pads to the pins of the package.

Smartcards — Invasive attacks — Deprocessing

5. Use an optical microscope to take large high-resolution pictures of the chip surface.
6. Identify major architectural features (ROM, ALU, EEPROM, etc.) and/or lower-level features such as busses and gates.
7. Remove the top metal track layer by dipping the chip in hydrofluoric acid in an ultrasonic bath.
8. Repeat from 5, for each layer.

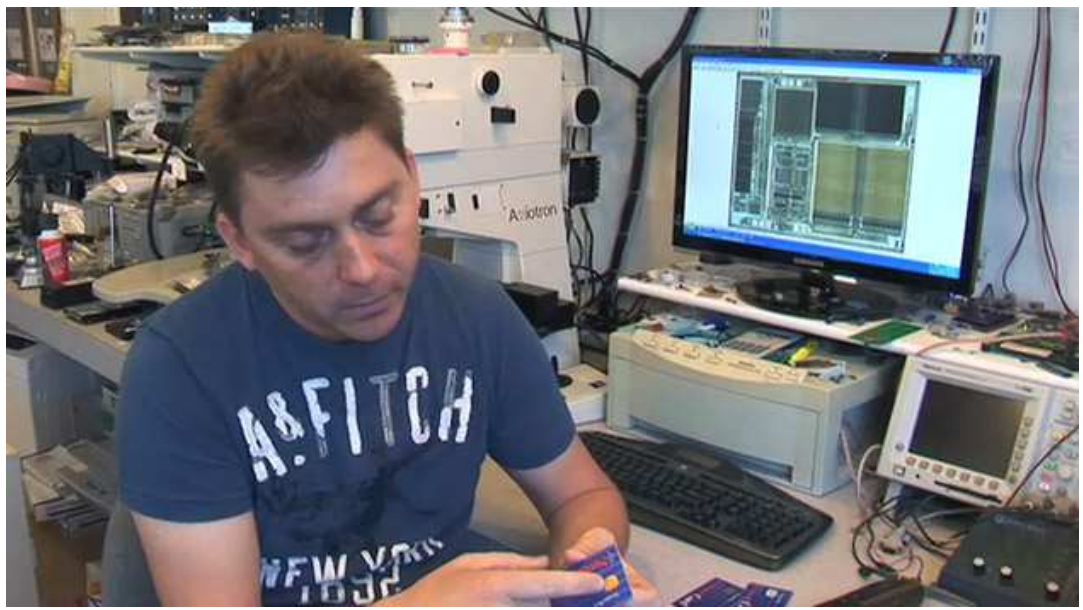
Smartcards — Invasive attacks — Reverse Engineering

- Reverse engineer the chip
- Analyze the information collected
- Understand the functional units of the chip

Smartcards — Invasive attacks — Microprobing

9. To allow the probe contact with the chip, use a laser cutter mounted on the microscope to remove (patches of) the *passivation layer* that covers the top-layer aluminum interconnect lines.
10. Record the activity on a few of the bus lines (as many as you have probes) as you go through a transaction with the card.
11. Repeat from 10 until you've collected the bus activity trace from all of the bus lines.

Smartcards — Christopher Tarnovsky



Smartcards — Christopher Tarnovsky

> *Dish Network is accusing News Corp . . . of hiring hacker Christopher Tarnovsky to break into Dish's network, steal the security codes, and use them to make pirated cards to flood the black market.*
>
> *Tarnovsky admitted in court he was paid James Bond villain style, with $20,000 cash payments mailed from Canada hidden inside "electronic devices."*

`http://gizmodo.com/383753/news-corp-hir`

Smartcards — Invasive attacks — Summary

- Attacks get harder as features get smaller
- Rent a lab!
- User your university lab!

Smartcards — Non-invasive attacks

- Passive attack:
  - Watch what comes out of the chip
  - . . . , electromagnetic radiation, power consumption, execution time, . . .
- Active attack:
  - Feed carefully constructed data/power/clock/. . . to the chip,
  - *then* measure the chip's behavior.

Smartcards — Timing attacks

```
s[0] = 1;
for(k=0; k<w; k++) {
    if (x[k] == 1)
        R[k] = (s[k]*y) mod
n;
    else
        R[k] = s[k];
    s[k+1] = R[k]*R[k] mod n
}
return R[w-1];
```

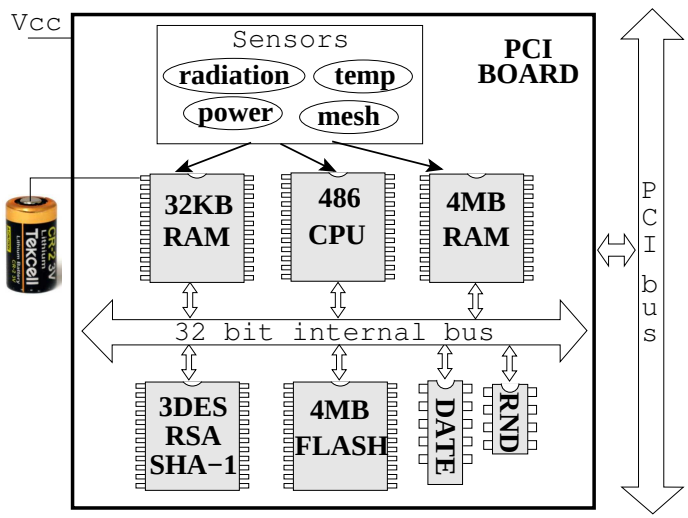
- Ask card: “please encrypt this file with your secret key”

Boardlevel protection

IBM 4758 cryptographic coprocessor

- Physical protection hasn't been broken!
- Costs about $4000.
- Vending machines that "sell money": topping up mobile phones, adding money to smartcards, or printing postage stamps.
- Used by banks to protect electronic transfers from insider attacks.

IBM 4758 cryptographic coprocessor



IBM 4758 cryptographic coprocessor

- PCI card — plug in to a host computer.
- 486-type processor, 4MB of RAM, 4MB of FLASH, 32KB of battery backed RAM.
- Cryptographic facilities: DES, 3DES, RSA, true random number generation, SHA-1.
- Date and time unit.

IBM 4758 — Tamper-detection

- When tamper-detection sensor triggers:
  1. Turn off power to the battery backed RAM $\Rightarrow$ zero secret code and data.
  2. Reset the CPU $\Rightarrow$ destroy RAM contents.

IBM 4758 — Attacks and Defenses

- Penetration attacks:
  - $\Rightarrow$ “polyurethane mixture and a film with an imprinted circuit pattern to detect minute penetration and erosion attacks”

IBM 4758 — Attacks and Defenses

- Penetration attacks:
  - $\Rightarrow$ “polyurethane mixture and a film with an imprinted circuit pattern to detect minute penetration and erosion attacks”
- Memory remanence attacks:
  - Freeze/radiate, remove from computer, drill down to RAM
  - $\Rightarrow$ temperature and radiation sensors

IBM 4758 — Attacks and Defenses

- Penetration attacks:
  - $\Rightarrow$ “polyurethane mixture and a film with an imprinted circuit pattern to detect minute penetration and erosion attacks”
- Memory remanence attacks:
  - Freeze/radiate, remove from computer, drill down to RAM
  - $\Rightarrow$ temperature and radiation sensors
- Fault-induction attacks:
  - Bring voltage abnormally high or low.
  - $\Rightarrow$ low and high voltage sensors.

IBM 4758 — Attacks and Defenses

- Power analysis attacks:
  - $\Rightarrow$ filter the power supply

IBM 4758 — Attacks and Defenses

- Power analysis attacks:
  - $\Rightarrow$ filter the power supply
- Electromagnetic analysis attacks:
  - $\Rightarrow$ shield the board in a metal enclosure (*Faraday Cage*)

IBM 4758 — Summary

- No known attacks against the physical protection of the 4758.

IBM 4758 — Summary

- No known attacks against the physical protection of the 4758.
- Lessons: Must protect against
  - *any* kind of leakage of information
  - *any* injection of faulty code/data
  - *any* adverse environmental conditions

IBM 4758 — Summary

- No known attacks against the physical protection of the 4758.
- Lessons: Must protect against
  - *any* kind of leakage of information
  - *any* injection of faulty code/data
  - *any* adverse environmental conditions
- Can we build a device that is
  - efficient,
  - secure, and
  - cheap?

Discussion

Disadvantages

- Deployment — long time before new technology is on every PC.
- Cost — extra hardware is expensive.
- Usability —
  - Lose a dongle?
  - Upgrade to a faster CPU, with different key?
  - Software vendor goes out of business?
- Security — invasive attacks, side-channel attacks.
- Performance — crypto-processors are slower.
- Engineering — design complexity, ease of testing, . . .

Thank You!