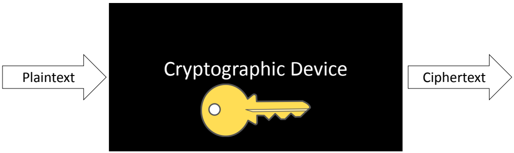
Timing Attacks

Kostas Papagiannopoulos
University of Amsterdam
k.papagiannopoulos@uva.nl

Contents

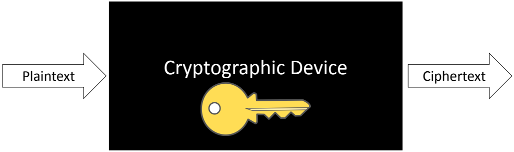
Introduction

Introduction



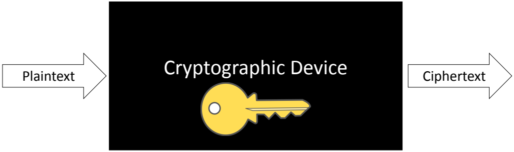
- Cryptographic algorithms like AES, PRESENT and RSA are a black box, parameterized with key K, that turns plaintext P to ciphertext C

Introduction



- Cryptographic algorithms like AES, PRESENT and RSA are a black box, parameterized with key K, that turns plaintext P to ciphertext C
- Analyzing the cipher's security in the **blackbox scenario** relates to classical cryptanalysis techniques like linear and differential cryptanalysis

Introduction



- Cryptographic algorithms like AES, PRESENT and RSA are a black box, parameterized with key K, that turns plaintext P to ciphertext C
- Analyzing the cipher's security in the **blackbox scenario** relates to classical cryptanalysis techniques like linear and differential cryptanalysis
- "Can you recover the secret key by observing plaintext/ciphertext pairs?"

Introduction

- The cryptographic algorithm is typically implemented on a device such as a desktop/laptop/cellphone processor, microcontroller, FPGA, ASIC etc.

Introduction

- The cryptographic algorithm is typically implemented on a device such as a desktop/laptop/cellphone processor, microcontroller, FPGA, ASIC etc.
- We can observe certain **physical quantities** in the device's vicinity such as the cipher's execution time, the power consumption of the device and others

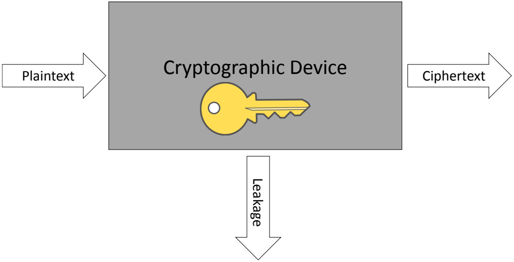
Introduction





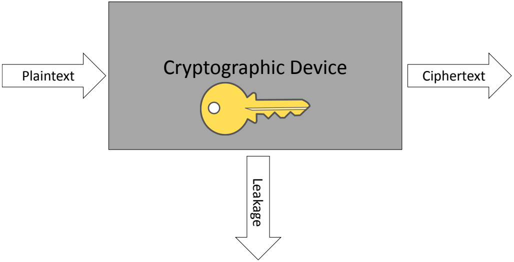

- The cryptographic algorithm is typically implemented on a device such as a desktop/laptop/cellphone processor, microcontroller, FPGA, ASIC etc.
- We can observe certain **physical quantities** in the device's vicinity such as the cipher's execution time, the power consumption of the device and others
- Although they seem harmless, these quantities can provide substantial information during cryptanalysis
- In fact, they grant us limited access to the internal computations carried out by the cipher

Introduction



- We refer to this case as the **greybox scenario** and to the relevant physical quantities as **side-channels**

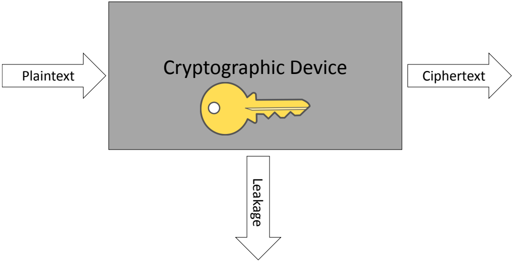
Introduction



- We refer to this case as the **greybox scenario** and to the relevant physical quantities as **side-channels**
- The **side-channel leakage** is any unintentional signal that offers us a blurry view of the cipher's internal computations

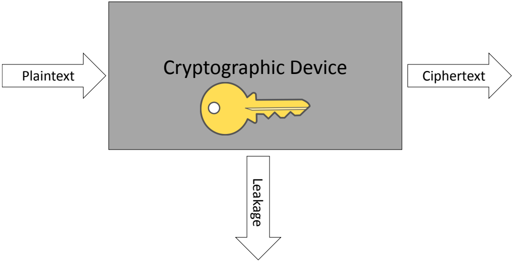
  e.g. the padding error message during server communication can be considered a 1-bit side-channel leakage

Introduction



- "Can you derive the secret key by observing plaintext/ciphertext pairs **and a side-channel**?"

Introduction



- "Can you derive the secret key by observing plaintext/ciphertext pairs **and a side-channel**?"
- Secure algorithms under the blackbox scenario may not be secure under the greybox scenario

  e.g. AES and DES are resistant to differential cryptanalysis but can be broken easily with a side-channel attack

Introduction

- We describe a simple timing attack on PIN code verification
- Assume that the following code was written (hastily) to check if a PIN matches the correct value 5902

Input: A 4-digit PIN code
Output: PIN accept or reject

```
Process CheckPIN(pin[4])
int pin_ok = 0;
if pin[0]==5 then
    if pin[1]==9 then
        if pin[2]==0 then
            if pin[3]==2 then
                pin_ok = 1 ;
            end
        end
    end
end
return pin_ok ;
EndProcess
```

Introduction

- We describe a simple timing attack on PIN code verification
- Assume that the following code was written (hastily) to check if a PIN matches the correct value 5902

```
Input: A 4-digit PIN code
Output: PIN accept or reject
Process CheckPIN(pin[4])
int pin_ok = 0;
if pin[0]==5 then
    if pin[1]==9 then
        if pin[2]==0 then
            if pin[3]==2 then
                pin_ok = 1 ;
            end
        end
    end
end
return pin_ok ;
EndProcess
```

- What are the execution times of the process for PIN inputs `[0,1,2,3]`, `[5,3,0,2]`, `[5,9,0,0]`?

Introduction

- We describe a simple timing attack on PIN code verification
- Assume that the following code was written (hastily) to check if a PIN matches the correct value 5902

```
Input: A 4-digit PIN code
Output: PIN accept or reject
Process CheckPIN(pin[4])
int pin_ok = 0;
if pin[0]==5 then
    if pin[1]==9 then
        if pin[2]==0 then
            if pin[3]==2 then
                pin_ok = 1 ;
            end
        end
    end
end
return pin_ok ;
EndProcess
```

- What are the execution times of the process for PIN inputs `[0,1,2,3]`, `[5,3,0,2]`, `[5,9,0,0]`?
- Notice that the execution time increases as we get closer to the correct PIN value `[5,9,0,2]`

Introduction

- We describe a simple timing attack on PIN code verification
- Assume that the following code was written (hastily) to check if a PIN matches the correct value 5902

```
Input: A 4-digit PIN code
Output: PIN accept or reject
Process CheckPIN(pin[4])
int pin_ok = 0;
if pin[0]==5 then
    if pin[1]==9 then
        if pin[2]==0 then
            if pin[3]==2 then
                pin_ok = 1 ;
            end
        end
    end
end
return pin_ok ;
EndProcess
```

- What are the execution times of the process for PIN inputs `[0,1,2,3]`, `[5,3,0,2]`, `[5,9,0,0]`?
- Notice that the execution time increases as we get closer to the correct PIN value `[5,9,0,2]`
- Timing attacks exploit the relationship between the execution time and a secret (e.g. a PIN or a cipher key)

AES Tbox Implementation

AES Tbox

AES T-box implementation

- The standard AES code implements every operation independently

  i.e. we code and optimize the key addition, sbox, shift rows and mix columns separately

AES Tbox

AES T-box implementation

- The standard AES code implements every operation independently

  i.e. we code and optimize the key addition, sbox, shift rows and mix columns separately
- To speed up software implementations, Rijndael proposed **merging** the sbox, shift rows and mix columns operations in lookup tables (LUTs) called **T-boxes**

AES Tbox

AES T-box implementation

- The standard AES code implements every operation independently

  i.e. we code and optimize the key addition, sbox, shift rows and mix columns separately
- To speed up software implementations, Rijndael proposed **merging** the sbox, shift rows and mix columns operations in lookup tables (LUTs) called **T-boxes**
- The merging led to very fast implementations, especially in 32-bit architectures
- The downside was increased memory requirements, since the implementation needed 4 lookup tables of 1024 bytes each

AES Tbox

AES T-box implementation

- The standard AES code implements every operation independently

  i.e. we code and optimize the key addition, sbox, shift rows and mix columns separately
- To speed up software implementations, Rijndael proposed **merging** the sbox, shift rows and mix columns operations in lookup tables (LUTs) called **T-boxes**
- The merging led to very fast implementations, especially in 32-bit architectures
- The downside was increased memory requirements, since the implementation needed 4 lookup tables of 1024 bytes each
- Software libraries often use the merged AES implementation while resource-constrained hardware keep standard implementations

AES Tbox

AES 16-byte state

$$\text{state} \quad A = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}$$

AES Tbox

AES 16-byte state

$$\text{state} \quad A = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}$$

Sbox operation

$$b_{i,j} = S(a_{i,j}), \quad \text{for all } i,j \in \{0,1,2,3\}$$

In column-vector notation:

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} S(a_{0,j}) \\ S(a_{1,j}) \\ S(a_{2,j}) \\ S(a_{3,j}) \end{bmatrix}, \quad \text{for } j = 0,1,2,3$$

AES Tbox

Shift Rows operation

$$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j} \\ b_{1,j-C_1} \\ b_{2,j-C_2} \\ b_{3,j-C_3} \end{bmatrix}, \quad \text{for } j = 0, 1, 2, 3$$

For AES-128: $C_1 = 1, C_2 = 2, C_3 = 3$ and $j - C = \mathrm{mod}(j - C, 4)$

AES Tbox

Shift Rows operation

$$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j} \\ b_{1,j-C_1} \\ b_{2,j-C_2} \\ b_{3,j-C_3} \end{bmatrix}, \quad \text{for } j = 0, 1, 2, 3$$

For AES-128: $\quad C_1 = 1, C_2 = 2, C_3 = 3 \quad \text{and} \quad j - C = \text{mod}(j - C, 4)$

Mix Columns operation

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}, \quad \text{for } j = 0, 1, 2, 3$$

AES Tbox

Key addition operation

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}, \quad \text{for } j = 0, 1, 2, 3$$

AES Tbox

- We now combine the previous expressions backwards (for $j = 0, 1, 2, 3$)

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} =$$

AES Tbox

- We now combine the previous expressions backwards (for $j = 0, 1, 2, 3$)

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} =$$

AES Tbox

- We now combine the previous expressions backwards (for $j = 0, 1, 2, 3$)

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} =$$

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} b_{0,j} \\ b_{1,j-C_1} \\ b_{2,j-C_2} \\ b_{3,j-C_3} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} =$$

AES Tbox

- We now combine the previous expressions backwards (for $j = 0, 1, 2, 3$)

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} =$$

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} b_{0,j} \\ b_{1,j-C_1} \\ b_{2,j-C_2} \\ b_{3,j-C_3} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} =$$

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S(a_{0,j}) \\ S(a_{1,j-C_1}) \\ S(a_{2,j-C_2}) \\ S(a_{3,j-C_3}) \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

AES Tbox

- Using the matrix multiplication rule we can re-write the expression as follows

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = S(a_{0,j}) \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \oplus S(a_{1,j-C_1}) \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus S(a_{2,j-C_2}) \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \oplus S(a_{3,j-C_3}) \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

AES Tbox

- Using the matrix multiplication rule we can re-write the expression as follows

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = S(a_{0,j}) \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \oplus S(a_{1,j-C_1}) \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus S(a_{2,j-C_2}) \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \oplus S(a_{3,j-C_3}) \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

- Observe that we can now view the column j of the AES round output $[e_{0,j} \ \ e_{1,j} \ \ e_{2,j} \ \ e_{3,j}]^\mathsf{T}$ as the sum of 4 vectors with the key vector

AES Tbox

- We now define 4 lookup tables aka the **T-boxes** $T_0(\cdot), T_1(\cdot), T_2(\cdot), T_3(\cdot)$

$$T_0(a) = \begin{bmatrix} S(a)\cdot 02 \\ S(a) \\ S(a) \\ S(a)\cdot 03 \end{bmatrix}, \quad T_1(a) = \begin{bmatrix} S(a)\cdot 03 \\ S(a)\cdot 02 \\ S(a) \\ S(a) \end{bmatrix}$$

$$T_2(a) = \begin{bmatrix} S(a) \\ S(a)\cdot 03 \\ S(a)\cdot 02 \\ S(a) \end{bmatrix}, \quad T_3(a) = \begin{bmatrix} S(a) \\ S(a) \\ S(a)\cdot 03 \\ S(a)\cdot 02 \end{bmatrix}$$

AES Tbox

- We now define 4 lookup tables aka the **T-boxes** $T_0(\cdot), T_1(\cdot), T_2(\cdot), T_3(\cdot)$

$$T_0(a) = \begin{bmatrix} S(a)\cdot 02 \\ S(a) \\ S(a) \\ S(a)\cdot 03 \end{bmatrix}, \quad T_1(a) = \begin{bmatrix} S(a)\cdot 03 \\ S(a)\cdot 02 \\ S(a) \\ S(a) \end{bmatrix}$$

$$T_2(a) = \begin{bmatrix} S(a) \\ S(a)\cdot 03 \\ S(a)\cdot 02 \\ S(a) \end{bmatrix}, \quad T_3(a) = \begin{bmatrix} S(a) \\ S(a) \\ S(a)\cdot 03 \\ S(a)\cdot 02 \end{bmatrix}$$

- Each T-box has 256 entries since the byte-sized input a takes values in $\{0, 1, \ldots, 255\}$
- Every T-box entry contains 4 rows with values thus every entry contains 4 bytes

AES Tbox

- We now define 4 lookup tables aka the **T-boxes** $T_0(\cdot), T_1(\cdot), T_2(\cdot), T_3(\cdot)$

$$T_0(a) = \begin{bmatrix} S(a)\cdot 02 \\ S(a) \\ S(a) \\ S(a)\cdot 03 \end{bmatrix}, \quad T_1(a) = \begin{bmatrix} S(a)\cdot 03 \\ S(a)\cdot 02 \\ S(a) \\ S(a) \end{bmatrix}$$

$$T_2(a) = \begin{bmatrix} S(a) \\ S(a)\cdot 03 \\ S(a)\cdot 02 \\ S(a) \end{bmatrix}, \quad T_3(a) = \begin{bmatrix} S(a) \\ S(a) \\ S(a)\cdot 03 \\ S(a)\cdot 02 \end{bmatrix}$$

- Each T-box has 256 entries since the byte-sized input a takes values in $\{0, 1, \ldots, 255\}$
- Every T-box entry contains 4 rows with values thus every entry contains 4 bytes
- In total, every T-box has a size of $256 \times 4 = 1024$ bytes. Combined, all 4 T-boxes require 4Kbytes of memory space.

AES Tbox

- Using the 4 T-boxes we can now express all the AES round operations as follows

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = T_0(a_{0,j}) \oplus T_1(a_{1,j-C_1}) \oplus T_2(a_{2,j-C_2}) \oplus T_3(a_{3,j-C_3}) \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

AES Tbox

- Using the 4 T-boxes we can now express all the AES round operations as follows

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = T_0(a_{0,j}) \oplus T_1(a_{1,j-C_1}) \oplus T_2(a_{2,j-C_2}) \oplus T_3(a_{3,j-C_3}) \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

- Repeating the expression above for $j = 0, 1, 2, 3$ will yield the full 16-byte cipher state
- The T-boxes have merged the sbox, shift rows and mix columns, improving speed but increasing memory requirements

Cache Timing Attacks on AES

Cache Timing Attacks on AES

Attack Idea:

- An AES implementation that uses T-boxes will perform several memory lookups

Cache Timing Attacks on AES

Attack Idea:

- An AES implementation that uses T-boxes will perform several memory lookups
- AES starts with the 16-byte plaintext P getting XORed with the round key K_{round0}

$$A = P \oplus K_{round0}$$

Cache Timing Attacks on AES

Attack Idea:

- An AES implementation that uses T-boxes will perform several memory lookups
- AES starts with the 16-byte plaintext P getting XORed with the round key K_{round0}

$$A = P \oplus K_{round0}$$

- Then the state A will be used to perform T-box lookups in memory

$$T_0(a_{0,j}), \quad T_1(a_{1,j-C_1}), \quad T_2(a_{2,j-C_2}), \quad T_3(a_{3,j-C_3}), \quad j = 0, 1, 2, 3$$

Cache Timing Attacks on AES

Attack Idea:

- An AES implementation that uses T-boxes will perform several memory lookups
- AES starts with the 16-byte plaintext P getting XORed with the round key K_{round0}

$$A = P \oplus K_{round0}$$

- Then the state A will be used to perform T-box lookups in memory

$$T_0(a_{0,j}), \quad T_1(a_{1,j-C_1}), \quad T_2(a_{2,j-C_2}), \quad T_3(a_{3,j-C_3}), \quad j = 0,1,2,3$$

- Let's focus on the 1st T-box lookup for $j = 0$

$$T_0(a_{0,0}) = T_0(p_{0,0} \oplus k_{0,0})$$

Cache Timing Attacks on AES

Attack Idea:

- An AES implementation that uses T-boxes will perform several memory lookups
- AES starts with the 16-byte plaintext P getting XORed with the round key K_{round0}

$$A = P \oplus K_{round0}$$

- Then the state A will be used to perform T-box lookups in memory

$$T_0(a_{0,j}), \quad T_1(a_{1,j-C_1}), \quad T_2(a_{2,j-C_2}), \quad T_3(a_{3,j-C_3}), \quad j = 0, 1, 2, 3$$

- Let's focus on the 1st T-box lookup for $j = 0$

$$T_0(a_{0,0}) = T_0(p_{0,0} \oplus k_{0,0})$$

- Notice that if there is any relationship between the execution time of this operation and the lookup index $a_{0,0}$ then measuring time reveals the value of index $a_{0,0}$

Cache Timing Attacks on AES

Attack Idea:

- An AES implementation that uses T-boxes will perform several memory lookups
- AES starts with the 16-byte plaintext P getting XORed with the round key K_{round0}

$$A = P \oplus K_{round0}$$

- Then the state A will be used to perform T-box lookups in memory

$$T_0(a_{0,j}), \quad T_1(a_{1,j-C_1}), \quad T_2(a_{2,j-C_2}), \quad T_3(a_{3,j-C_3}), \quad j = 0, 1, 2, 3$$

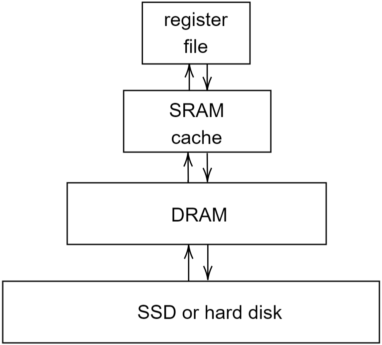
- Let's focus on the 1st T-box lookup for $j = 0$

$$T_0(a_{0,0}) = T_0(p_{0,0} \oplus k_{0,0})$$

- Notice that if there is any relationship between the execution time of this operation and the lookup index $a_{0,0}$ then measuring time reveals the value of index $a_{0,0}$
- Any information about $a_{0,0}$ leads to information about $k_{0,0} = a_{0,0} \oplus p_{0,0}$ since $p_{0,0}$ is publicly known

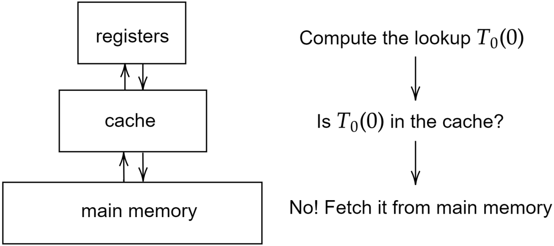
Cache Timing Attacks on AES

- Processors utilize data and instruction caches to speedup memory access
- Most systems have a memory hierarchy to ensure spatial and temporal locality



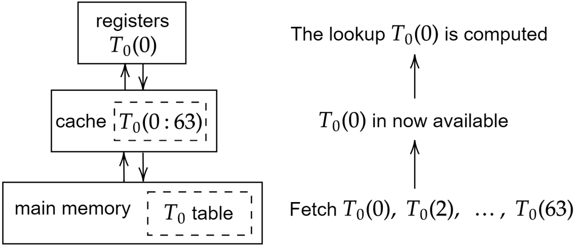
Cache Timing Attacks on AES

- To perform e.g. the AES T-box lookup $T_0(0)$ we need to transfer a specific memory element of T_0 based on index $a_{0,0} = 0$ to the registers
- If $T_0(0)$ is not stored in the cache then we need to fetch it from the main memory



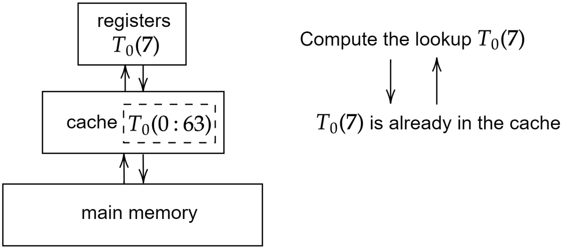
Cache Timing Attacks on AES

- To achieve good spatial locality we will (probably) transfer also several continuous values of table T_0 to the cache
- This type of data access is a **cache miss** and it takes a large amount of time



Cache Timing Attacks on AES

- Some subsequent memory accesses will be faster
- For example accessing $T_0(7)$ is a **cache hit** and it takes a small amount of time
- Thus there is a dependency between the index $a_{0,0}$ and the execution time
- Remember that $a_{0,0}$ relates to the key $k_{0,0}$



Cache Attacks on AES

This timing attack works in two phases: profile phase and attack phase

Profile phase:

1. Setup a server that can encrypt with AES and uses a known key k

Cache Attacks on AES

This timing attack works in two phases: profile phase and attack phase

Profile phase:

1. Setup a server that can encrypt with AES and uses a known key k
2. Send a large number (n) of random plaintexts to the server, encrypt them and measure the AES execution times $t_1, t_2, \ldots, t_n$

Cache Attacks on AES

This timing attack works in two phases: profile phase and attack phase

Profile phase:

1. Setup a server that can encrypt with AES and uses a known key k
2. Send a large number (n) of random plaintexts to the server, encrypt them and measure the AES execution times $t_1, t_2, \ldots, t_n$
3. Estimate the average execution time μ

$$\mu = \frac{1}{n} \sum_{i=1}^{n} t_i$$

Cache Attacks on AES

This timing attack works in two phases: profile phase and attack phase

Profile phase:

1. Setup a server that can encrypt with AES and uses a known key k
2. Send a large number (n) of random plaintexts to the server, encrypt them and measure the AES execution times $t_1, t_2, \ldots, t_n$
3. Estimate the average execution time μ

$$\mu = \frac{1}{n} \sum_{i=1}^{n} t_i$$

4. Focus on a specific plaintext byte, say $p_{0,0}$ and group the time measurements according to the value of $p_{0,0}$

$$G_0 = \{t_i \text{ s.t. } p_{0,0} = 0\}, \quad G_1 = \{t_i \text{ s.t. } p_{0,0} = 1\}, \quad \ldots, \quad G_{255} = \{t_i \text{ s.t. } p_{0,0} = 255\}$$

Cache Timing Attacks on AES

5. Compute the average of every group

$$\mu_0 = \frac{1}{|G_0|} \sum_{t_i \in G_0} t_i, \quad \mu_1 = \frac{1}{|G_1|} \sum_{t_i \in G_1} t_i \;, \quad \ldots \;, \quad \mu_{255} = \frac{1}{|G_{255}|} \sum_{t_i \in G_{255}} t_i$$

Cache Timing Attacks on AES

5. Compute the average of every group

$$\mu_0 = \frac{1}{|G_0|} \sum_{t_i \in G_0} t_i, \quad \mu_1 = \frac{1}{|G_1|} \sum_{t_i \in G_1} t_i \ , \quad \dots \ , \quad \mu_{255} = \frac{1}{|G_{255}|} \sum_{t_i \in G_{255}} t_i$$

6. Compute the difference between group average $\mu_{p_{0,0}}$ and the total average μ

$$\delta_0 = \mu_0 - \mu, \quad \delta_1 = \mu_1 - \mu \ , \quad \dots \ , \quad \delta_{255} = \mu_{255} - \mu$$

Cache Timing Attacks on AES

5. Compute the average of every group

$$\mu_0 = \frac{1}{|G_0|} \sum_{t_i \in G_0} t_i, \quad \mu_1 = \frac{1}{|G_1|} \sum_{t_i \in G_1} t_i \ , \quad \ldots \ , \quad \mu_{255} = \frac{1}{|G_{255}|} \sum_{t_i \in G_{255}} t_i$$

6. Compute the difference between group average $\mu_{p_{0,0}}$ and the total average μ

$$\delta_0 = \mu_0 - \mu, \quad \delta_1 = \mu_1 - \mu \ , \quad \ldots \ , \quad \delta_{255} = \mu_{255} - \mu$$

7. Find the maximum difference δ_{max} and the value p_{max} that caused it

$$p_{max} = \text{argmax}(\delta_0, \delta_1, \ldots, \delta_{255})$$

Thus when the server receives plaintext input $p_{0,0} = p_{max}$, the lookup performed is (on average) the slowest

Cache Timing Attacks on AES

5. Compute the average of every group

$$\mu_0 = \frac{1}{|G_0|} \sum_{t_i \in G_0} t_i, \quad \mu_1 = \frac{1}{|G_1|} \sum_{t_i \in G_1} t_i \ , \quad \dots \ , \quad \mu_{255} = \frac{1}{|G_{255}|} \sum_{t_i \in G_{255}} t_i$$

6. Compute the difference between group average $\mu_{p_{0,0}}$ and the total average μ

$$\delta_0 = \mu_0 - \mu, \quad \delta_1 = \mu_1 - \mu \ , \quad \dots \ , \quad \delta_{255} = \mu_{255} - \mu$$

7. Find the maximum difference δ_{max} and the value p_{max} that caused it

$$p_{max} = \text{argmax}(\delta_0, \delta_1, \dots, \delta_{255})$$

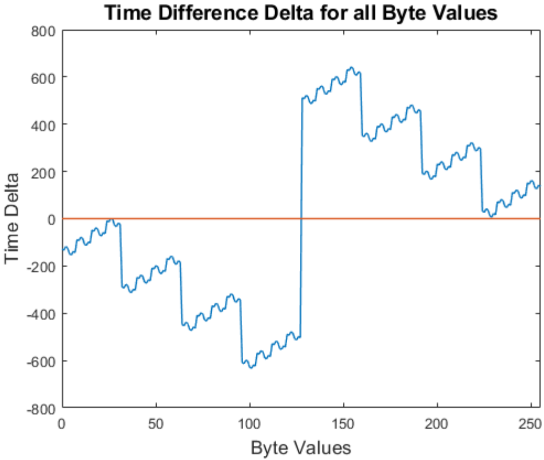
   Thus when the server receives plaintext input $p_{0,0} = p_{max}$, the lookup performed is (on average) the slowest

8. During the profiling phase we know the server key $k_{0,0}$ and we can use it and find which is the exact lookup index a_{max} that caused the slowest lookup

$$a_{0,0} = k_{0,0} \oplus p_{0,0} \iff a_{max} = k_{0,0} \oplus p_{max}$$

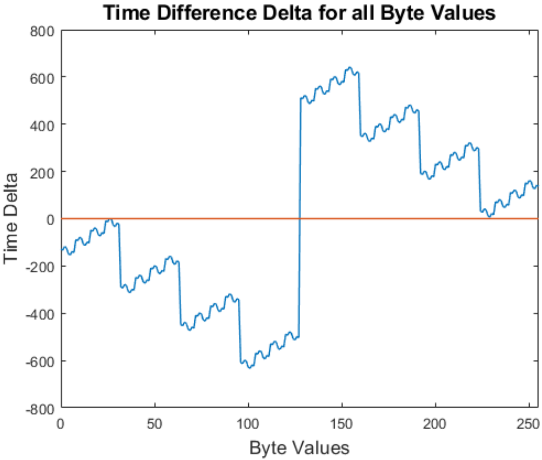
Cache Timing Attacks on AES

- We plot the differences δ for every possible value of the plaintext input byte $p_{0,0}$



Cache Timing Attacks on AES

- We plot the differences δ for every possible value of the plaintext input byte $p_{0,0}$



- The timing attack assumes that the lookup index a_{max} that caused the slowest lookup will be the same in other servers with the same hardware and AES implementation

Cache Attacks on AES

Attack phase:

1. Attack a server that uses the **exact same** hardware and AES implementation

Cache Attacks on AES

Attack phase:

1. Attack a server that uses the **exact same** hardware and AES implementation
2. Naturally, the server key k' is unknown

Cache Attacks on AES

Attack phase:

1. Attack a server that uses the **exact same** hardware and AES implementation
2. Naturally, the server key k' is unknown
3. Send a large number (n') of random plaintexts to the server, encrypt them and measure the AES execution times $t'_1, t'_2, \ldots, t'_{n'}$

Cache Attacks on AES

Attack phase:

1. Attack a server that uses the **exact same** hardware and AES implementation
2. Naturally, the server key k' is unknown
3. Send a large number (n') of random plaintexts to the server, encrypt them and measure the AES execution times $t'_1, t'_2, \ldots, t'_{n'}$
4. Estimate the average execution time μ'

$$\mu' = \frac{1}{n'} \sum_{i=1}^{n'} t'_i$$

Cache Attacks on AES

Attack phase:

1. Attack a server that uses the **exact same** hardware and AES implementation
2. Naturally, the server key k' is unknown
3. Send a large number (n') of random plaintexts to the server, encrypt them and measure the AES execution times $t'_1, t'_2, \ldots, t'_{n'}$
4. Estimate the average execution time μ'

$$\mu' = \frac{1}{n'} \sum_{i=1}^{n'} t'_i$$

5. Focus on a specific plaintext byte, say $p'_{0,0}$ and group the time measurements according to the value of $p'_{0,0}$

$$G'_0 = \{t'_i \text{ s.t. } p'_{0,0} = 0\}, \quad G'_1 = \{t'_i \text{ s.t. } p'_{0,0} = 1\}, \ \ldots, \ G'_{255} = \{t'_i \text{ s.t. } p'_{0,0} = 255\}$$

Cache Timing Attacks on AES

5. Compute the average of every group

$$\mu'_0 = \frac{1}{|G'_0|} \sum_{t'_i \in G_0} t'_i, \quad \mu'_1 = \frac{1}{|G'_1|} \sum_{t'_i \in G'_1} t'_i \;, \quad \ldots\;, \quad \mu'_{255} = \frac{1}{|G'_{255}|} \sum_{t'_i \in G'_{255}} t'_i$$

Cache Timing Attacks on AES

5. Compute the average of every group

$$\mu'_0 = \frac{1}{|G'_0|} \sum_{t'_i \in G_0} t'_i, \quad \mu'_1 = \frac{1}{|G'_1|} \sum_{t'_i \in G'_1} t'_i \ , \quad \ldots \ , \quad \mu'_{255} = \frac{1}{|G'_{255}|} \sum_{t'_i \in G'_{255}} t'_i$$

6. Compute the difference between group average $\mu'_{p'_{0,0}}$ and the total average μ'

$$\delta'_0 = \mu'_0 - \mu', \quad \delta'_1 = \mu'_1 - \mu' \ , \quad \ldots \ , \quad \delta'_{255} = \mu'_{255} - \mu'$$

Cache Timing Attacks on AES

5. Compute the average of every group

$$\mu'_0 = \frac{1}{|G'_0|} \sum_{t'_i \in G_0} t'_i, \quad \mu'_1 = \frac{1}{|G'_1|} \sum_{t'_i \in G'_1} t'_i \ , \quad \ldots \ , \quad \mu'_{255} = \frac{1}{|G'_{255}|} \sum_{t'_i \in G'_{255}} t'_i$$

6. Compute the difference between group average $\mu'_{p'_{0,0}}$ and the total average μ'

$$\delta'_0 = \mu'_0 - \mu', \quad \delta'_1 = \mu'_1 - \mu' \ , \quad \ldots \ , \quad \delta'_{255} = \mu'_{255} - \mu'$$

7. Find the maximum difference δ'_{max} and the value p_{max} that caused it

$$p_{max} = \text{argmax}(\delta'_0, \delta'_1, \ldots, \delta'_{255})$$

Thus when the server receives plaintext input $p'_{0,0} = p'_{max}$, the lookup performed is (on average) the slowest

Cache Timing Attacks on AES

5. Compute the average of every group

$$\mu'_0 = \frac{1}{|G'_0|} \sum_{t'_i \in G_0} t'_i, \quad \mu'_1 = \frac{1}{|G'_1|} \sum_{t'_i \in G'_1} t'_i \;, \quad \ldots \;, \quad \mu'_{255} = \frac{1}{|G'_{255}|} \sum_{t'_i \in G'_{255}} t'_i$$

6. Compute the difference between group average $\mu'_{p'_{0,0}}$ and the total average μ'

$$\delta'_0 = \mu'_0 - \mu', \quad \delta'_1 = \mu'_1 - \mu' \;, \quad \ldots \;, \quad \delta'_{255} = \mu'_{255} - \mu'$$

7. Find the maximum difference δ'_{max} and the value p_{max} that caused it

$$p_{max} = \text{argmax}(\delta'_0, \delta'_1, \ldots, \delta'_{255})$$

   Thus when the server receives plaintext input $p'_{0,0} = p'_{max}$, the lookup performed is (on average) the slowest

8. During the profiling phase we have found the lookup index a_{max} that causes the slowest lookup

Cache Timing Attacks on AES

5. Compute the average of every group

$$\mu'_0 = \frac{1}{|G'_0|} \sum_{t'_i \in G_0} t'_i, \quad \mu'_1 = \frac{1}{|G'_1|} \sum_{t'_i \in G'_1} t'_i \ , \quad \ldots \ , \quad \mu'_{255} = \frac{1}{|G'_{255}|} \sum_{t'_i \in G'_{255}} t'_i$$

6. Compute the difference between group average $\mu'_{p'_{0,0}}$ and the total average μ'

$$\delta'_0 = \mu'_0 - \mu', \quad \delta'_1 = \mu'_1 - \mu' \ , \quad \ldots \ , \quad \delta'_{255} = \mu'_{255} - \mu'$$

7. Find the maximum difference δ'_{max} and the value p_{max} that caused it

$$p_{max} = \text{argmax}(\delta'_0, \delta'_1, \ldots, \delta'_{255})$$

   Thus when the server receives plaintext input $p'_{0,0} = p'_{max}$, the lookup performed is (on average) the slowest

8. During the profiling phase we have found the lookup index a_{max} that causes the slowest lookup
9. Since the AES implementation and server hardware are the same:

$$a_{max} = a'_{max} \iff a_{max} = k'_{0,0} \oplus p'_{max} \iff k'_{0,0} = a_{max} \oplus p'_{max}$$

Cache Timing Attacks on AES

Notes on timing attacks

- Timing attacks can be executed remotely over the network
- The majority of LUT-based AES implementations can be broken. Many cryptographic libraries, including OpenSSL, were found vulnerable, regardless of the server hardware.
- Even if the server does not provide timestamps in the packets, the attacker can measure the packet round-trip-time over the network

Cache Timing Attacks on AES

Countermeasures

- Inserting random delays during the AES computations can make the attack harder but not impossible because of the Central Limit theorem
  - Let time measurements $T_1, T_2, \ldots, T_n$, all with mean μ and standard deviation σ (i.e. the measurements are noisy)
  - Let also their average $\overline{T} = \frac{1}{n}\sum_{i=1}^{n} T_i$
  - Then, for large n, $\overline{T}$ follows a normal distribution $\mathcal{N}\left(\mu, \frac{\sigma}{\sqrt{n}}\right)$

Cache Timing Attacks on AES

Countermeasures

- Inserting random delays during the AES computations can make the attack harder but not impossible because of the Central Limit theorem
  - Let time measurements $T_1, T_2, \ldots, T_n$, all with mean μ and standard deviation σ (i.e. the measurements are noisy)
  - Let also their average $\overline{T} = \frac{1}{n}\sum_{i=1}^{n} T_i$
  - Then, for large n, $\overline{T}$ follows a normal distribution $\mathcal{N}\left(\mu, \frac{\sigma}{\sqrt{n}}\right)$
- Thus, as we increase the number n of time measurements, the noise of the average measurement decreases by factor $\sqrt{n}$.

Cache Timing Attacks on AES

Countermeasures

- Ensure constant time execution of the encryption/decryption algorithm
  - Use custom hardware that enforces constant-time memory access – yet this hardware will be slower than average
  - Implement the whole of AES without lookup tables – yet this may damage performance, especially in software implementations
  - Fully understand cache behavior and avoid time-dependent hits and misses during memory lookups – yet this can be very challenging due to the proprietary CPU design