



NTNU

Norwegian University of
Science and Technology

SIDE-CHANNEL ATTACKS 2: SYMMETRIC KEY CRYPTO

TTM4205 – Lecture 13

Tjerand Silde

29.09.2025

Contents

Previous Lecture

SCA on Symmetric Ciphers

Contents

Previous Lecture

SCA on Symmetric Ciphers

Black Box Crypto

We design the security of a cryptographic scheme to follow Kerckhoff's principle: if everything about the scheme, except for the key, is known, then the scheme should be secure.

We analyze the scheme mathematically as black-box algorithms that take some (public or secret) input and give some (public or secret) output, and prove it secure concerning the algorithm description and the public data.

However, security depends on your model. In practice, it matters how these algorithms are implemented and what kind of information the *physical* system leaks about the inner workings of the algorithm computing on secret data.

Leakage

- ▶ The time it takes to compute...
- ▶ The power usage while computing...
- ▶ The electromagnetic radiation...
- ▶ The temperature variation...
- ▶ The memory pattern accessed...
- ▶ The sounds your laptop makes...

Attack Categories

- ▶ Remote vs physical attacks
- ▶ Software and hardware attacks
- ▶ Passive vs active attacks
- ▶ Invasive vs non-invasive attacks

Contents

Previous Lecture

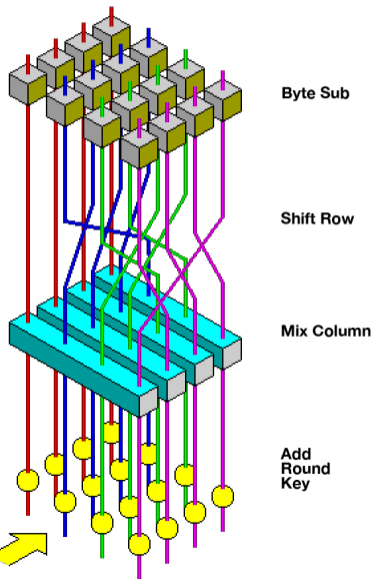
SCA on Symmetric Ciphers

Recall: AES

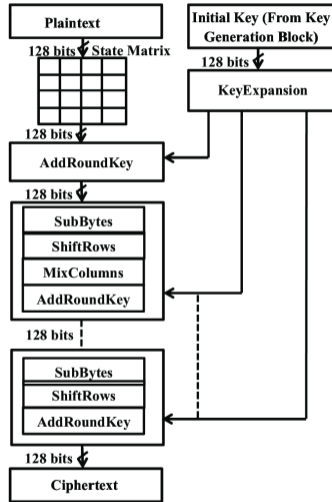
- ▶ AES is a symmetric key encryption scheme
- ▶ AES is a substitution-permutation network
- ▶ AES-128: uses 10 rounds and 128-bit keys
- ▶ Works on 4×4 column order array of 16 bytes
- ▶ Long messages are divided into 16 byte blocks
- ▶ Some modes of operations: ECB, CTR, GCM, etc.

Check out chapter 4 in Serious Cryptography by JPA.

Recall: AES



Recall: AES



Weaknesses and Defenses

In the following slides we will look at a common way to implement AES and its subroutines. For each algorithm, try to point out potential information leakage and protection.

Example Code

```
1   def encrypt(key, plaintext):
2
3       # AddRoundKey for initial round
4       ciphertext = AddRoundKey(plaintext, key[0])
5
6       for i in range(1, rounds):
7           ciphertext = SubBytes(ciphertext)
8           ciphertext = ShiftRows(ciphertext)
9           ciphertext = MixColumns(ciphertext)
10          ciphertext = AddRoundKey(ciphertext, key[i])
11
12         # Final round (no MixColumns)
13         ciphertext = SubBytes(ciphertext)
14         ciphertext = ShiftRows(ciphertext)
15         ciphertext = AddRoundKey(ciphertext, key[rounds])
16
17     return ciphertext
```

Differential Power Analysis

Differential Power Analysis

Paul Kocher, Joshua Jaffe, and Benjamin Jun

Cryptography Research, Inc.

~~-607 Market Street, 5th Floor~~

~~-San Francisco, CA 94105, USA-~~

<http://www.cryptography.com>

~~E-mail: {paul,josh,ben}@cryptography.com-~~

Abstract. Cryptosystem designers frequently assume that secrets will be manipulated in closed, reliable computing environments. Unfortunately, actual computers and microchips leak information about the operations they process. This paper examines specific methods for analyzing power consumption measurements to find secret keys from tamper resistant devices. We also discuss approaches for building cryptosystems that can operate securely in existing hardware that leaks information.

Keywords: differential power analysis, DPA, SPA, cryptanalysis, DES

Figure: <https://paulkocher.com/doc/DifferentialPowerAnalysis.pdf2>



Simple Power Analysis (on DES)

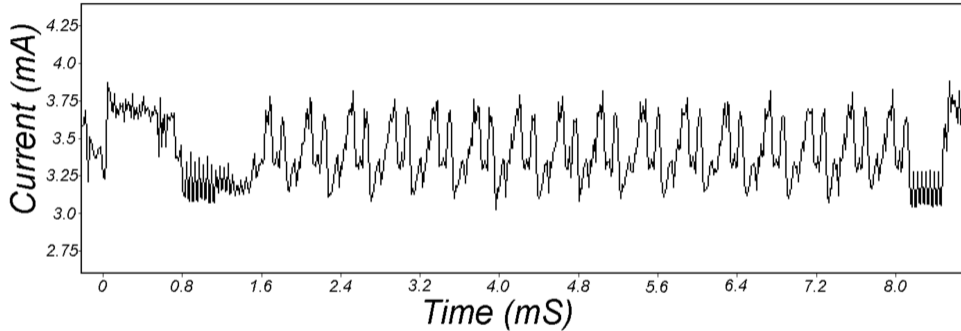


Figure 1: SPA trace showing an entire DES operation.

Detailed SPA (on DES)

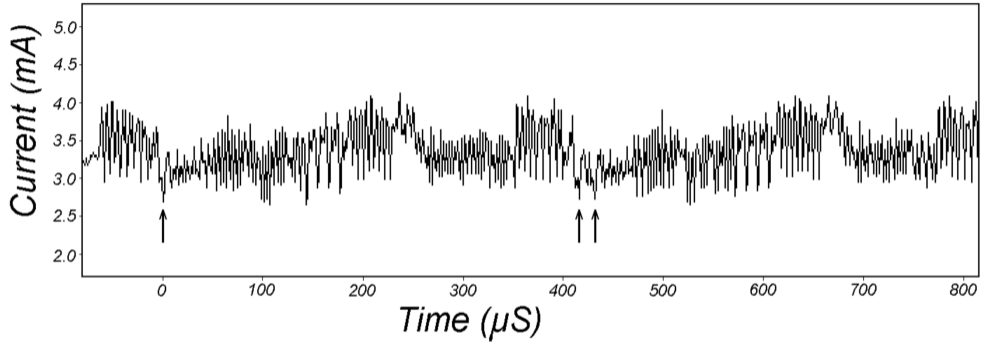


Figure 2: SPA trace showing DES rounds 2 and 3.

Potential Weaknesses

Some information leak directly:

- ▶ We can easily see how many rounds are computed
- ▶ We can easily see which operation is computed
- ▶ We can compare known traces with the first round

Based on this, we can compute the correlation between different keys and power consumption.

Correlation

Statistical Analysis via Pearson Correlation Coefficient ρ

- Linear relationship between 2 random variables
(how much do they change together)
- X : predictions corresponding to one key hypothesis
- Y : measured samples corresponding to one point in time

$$\rho = \frac{\text{Cov}(X, Y)}{\sqrt{\text{Var}(X) \cdot \text{Var}(Y)}} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$$

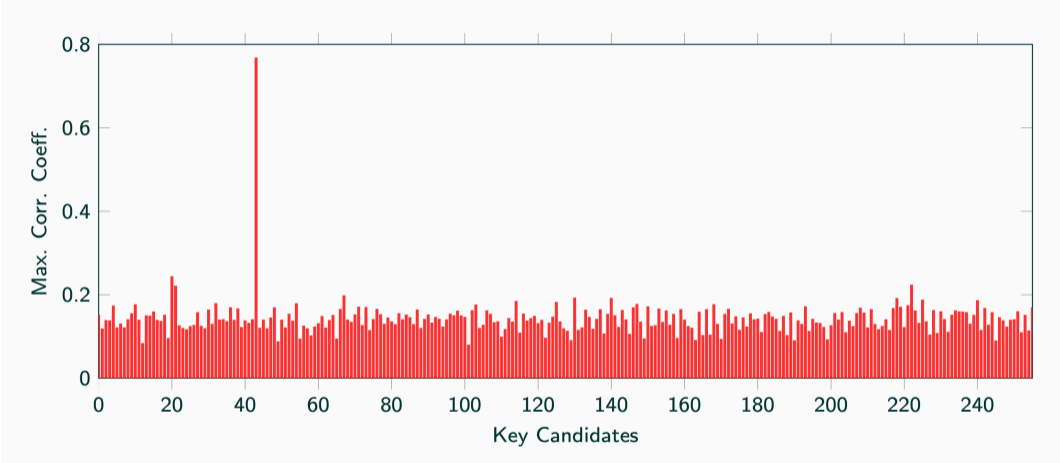
Cov = Covariance,
Var = Variance,
E = Expected value,
 σ = Standard deviation,
 μ = Mean

Estimate:

$$r = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2} \sqrt{\sum_i (y_i - \bar{y})^2}}$$

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

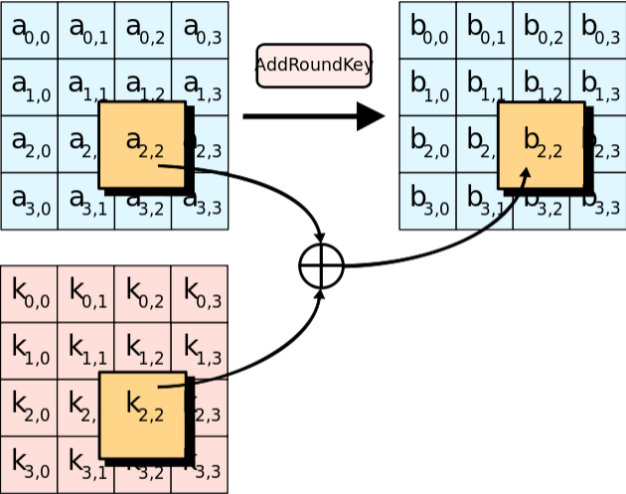
Key Candidates



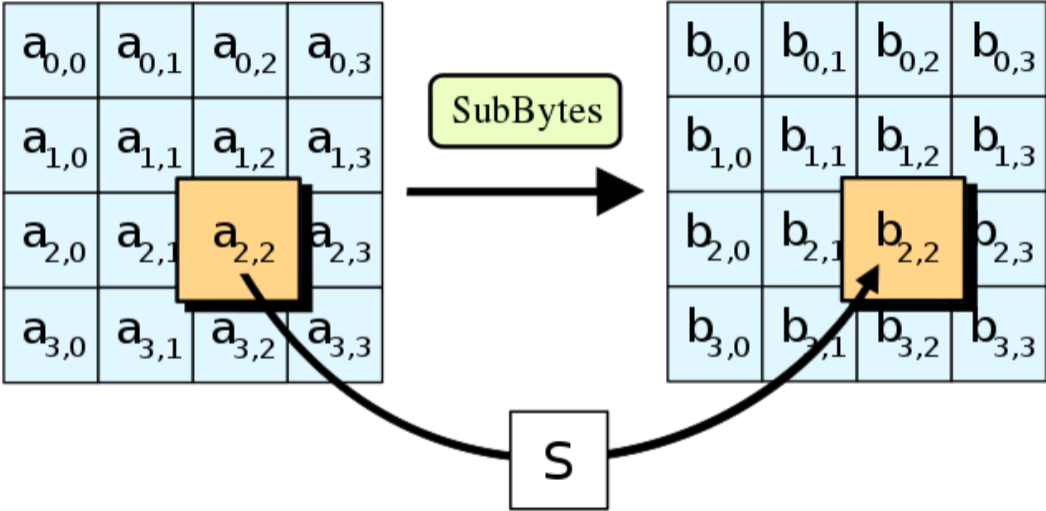
Potential Weaknesses

Usually, the correlation is not this strong, and we need to analyze each of the underlying subroutines in more detail. AddRoundKey, SubBytes, ShiftRows, and MixColumns works as follows:

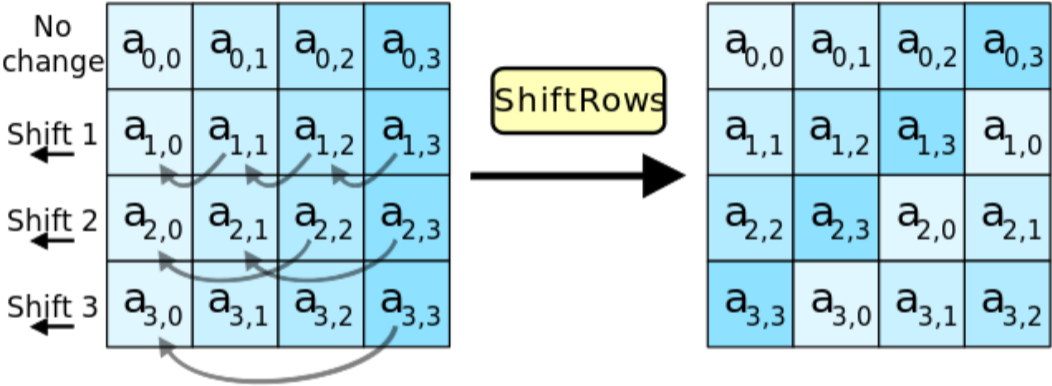
AddRoundKey



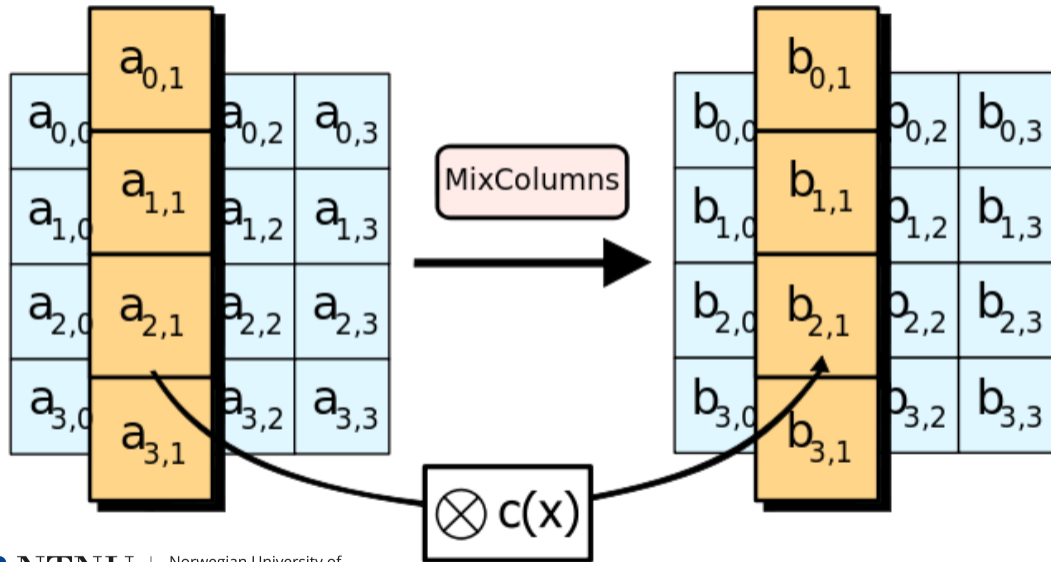
SubBytes (S-Box)



ShiftRows



MixColumns



Potential Weaknesses

- ▶ Computation after AddRoundKey might leak HW
- ▶ SubBytes is a non-linear operation (inverses)
- ▶ MixColumns is a polynomial/matrix multiplication
- ▶ Algebraic operations are computed over $GF(2^8)$

MixColumns

```
1  def MixColumns(state):
2
3      def single_col(col):
4
5          b = (col << 1) ^ (0x11B & -(col >> 7))
6
7          col_mixed = [
8              b[0] ^ col[3] ^ col[2] ^ b[1] ^ col[1],
9              b[1] ^ col[0] ^ col[3] ^ b[2] ^ col[2],
10             b[2] ^ col[1] ^ col[0] ^ b[3] ^ col[3],
11             b[3] ^ col[2] ^ col[1] ^ b[0] ^ col[0],
12         ]
13         return col_mixed
14
15     state[:, 0] = single_col(state[:, 0])
16     state[:, 1] = single_col(state[:, 1])
17     state[:, 2] = single_col(state[:, 2])
18     state[:, 3] = single_col(state[:, 3])
19     return state
```

Sub-Algorithm

```
1   def AddRoundKey(self, state, key):  
2       return np.bitwise_xor(state, key)  
3  
4   def SubBytes(self, state):  
5       return self.S_box[state]  
6  
7   def ShiftRows(self, state):  
8       return state.take(  
9           (0, 1, 2, 3, 5, 6, 7, 4, 10, 11, 8, 9, 15, 12, 13, 14)  
10          ).reshape(4, 4)
```

SubBytes (S-Box)

```
self.S_box = np.array(  
    [0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,  
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,  
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,  
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,  
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,  
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,  
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,  
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,  
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,  
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,  
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,  
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,  
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,  
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,  
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,  
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16], np.uint8)
```

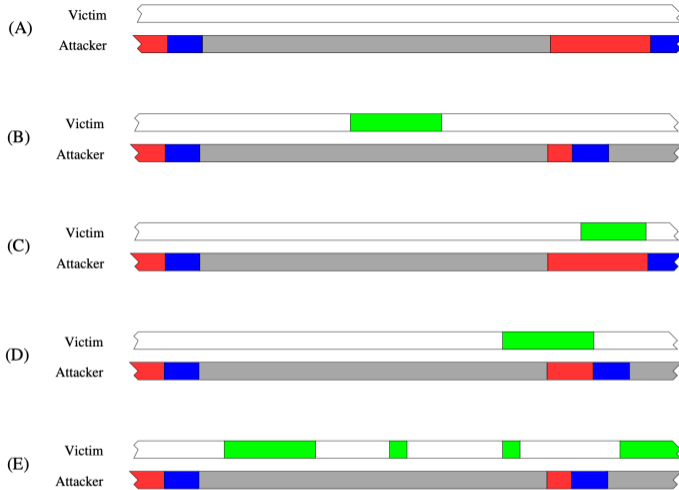
Cache-timing attacks on AES

Daniel J. Bernstein *

Department of Mathematics, Statistics, and Computer Science (M/C 249)
The University of Illinois at Chicago
Chicago, IL 60607-7045
`djb@cr.yp.to`

Potential Weaknesses

- ▶ NIST when standardizing the SubBytes in AES:
“Table lookup: not vulnerable to timing attacks”
- ▶ Several finalists in the competition were secure, but Rijndael was fastest and this was important
- ▶ Flush+Reload attacks on cache leaks the secret indices of the SubBytes lookup table



Potential Defenses

We must ensure one of the following:

- ▶ Avoid memory access, or
- ▶ Always read all entries, or
- ▶ Disable cache-sharing

The latter is impractical and affects general performance.

A Fast New DES Implementation in Software

Eli Biham

Computer Science Department
Technion – Israel Institute of Technology
Haifa 32000, Israel
Email: biham@cs.technion.ac.il
WWW: <http://www.cs.technion.ac.il/~biham/>

Bitslicing

Technique to avoid side-channel analysis:

- ▶ Work over bits not bytes in $GF(2^8)$
- ▶ Only use OR, AND, XOR, NAND, etc.
- ▶ Execute operations on vectors
- ▶ Is slower, but constant time
- ▶ Need a circuit for table lookup
- ▶ Integrated in hardware AES

We can combine this with randomized masking.

Provably Secure Higher-Order Masking of AES*

Matthieu Rivain¹ and Emmanuel Prouff²

¹ CryptoExperts

`matthieu.rivain@cryptoexperts.com`

² Oberthur Technologies

`e.prouff@oberthur.com`

Figure: <https://eprint.iacr.org/2010/441.pdf>

AES Masking

- ▶ d -order masking: split secret in d parts
- ▶ linear operations are easy, non-linear not
- ▶ AddKey, ShiftRows and MixColumns are linear
- ▶ SubBytes is not linear: requires extra work
- ▶ statistical analysis is exponential in d
- ▶ added work scales with $d \log_2 d$ operations

Masking AND

Secure logical AND. Let a and b be two bits and let c denote $\text{AND}(a, b) = ab$. Let us assume that a and b have been respectively split into $d + 1$ shares $(a_i)_{0 \leq i \leq d}$ and $(b_i)_{0 \leq i \leq d}$ such that $\bigoplus_i a_i = a$ and $\bigoplus_i b_i = b$. To securely compute a $(d + 1)$ -tuple $(c_i)_{0 \leq i \leq d}$ s.t. $\bigoplus_i c_i = c$, Ishai *et al.* perform the following steps:

1. For every $0 \leq i < j \leq d$, pick up a random bit $r_{i,j}$.
2. For every $0 \leq i < j \leq d$, compute $r_{j,i} = (r_{i,j} \oplus a_i b_j) \oplus a_j b_i$.
3. For every $0 \leq i \leq d$, compute $c_i = a_i b_i \oplus \bigoplus_{j \neq i} r_{i,j}$.

Masking AND

The completeness of the solution follows from:

$$\begin{aligned}\bigoplus_i c_i &= \bigoplus_i (a_i b_i \oplus \bigoplus_{j \neq i} r_{i,j}) = \bigoplus_i (a_i b_i \oplus \bigoplus_{j > i} r_{i,j} \oplus \bigoplus_{j < i} (r_{j,i} \oplus a_i b_j \oplus a_j b_i)) \\ &= \bigoplus_i (a_i b_i \oplus \bigoplus_{j < i} (a_i b_j \oplus a_j b_i)) = (\bigoplus_i a_i) (\bigoplus_i b_i) .\end{aligned}$$

Table 2. Comparison of secure AES implementations

Method	Reference	cycles	RAM (bytes)	ROM (bytes)
Unprotected Implementation				
No Masking	Na.	3×10^3	32	1150
First Order Masking				
Re-computation	[23]	10×10^3	256 + 35	1553
Tower Field in \mathbb{F}_4	[28, 29]	77×10^3	42	3195
Our scheme for $d = 1$	This paper	129×10^3	73	3153
Second Order Masking				
Double Re-computations	[38]	594×10^3	512 + 90	2336
Single Re-computation	[34]	672×10^3	256 + 86	2215
Our scheme for $d = 2$	This paper	271×10^3	79	3845
Third Order Masking				
Our scheme for $d = 3$	This paper	470×10^3	103	4648

Summary

Protecting secret key computations are difficult. We need to:

- ▶ avoid lookup tables
- ▶ constant time operations
- ▶ vectorize operations
- ▶ use randomness/masking



The banner features a stylized brown bear on the left and a modern building with a green roof on the right. A small green padlock icon is positioned between the bear and the building. Below the building, a sign reads "BEARSSL a smaller SSL/TLS library by Thomas Pornin".

Why Constant-Time Crypto?

MAIN
API DOCUMENTATION
BROWSE SOURCE CODE
CHANGE LOG
PROJECT GOALS
ON NAMING THINGS
SUPPORTED CRYPTO
ROADMAP AND STATUS
OOP IN C
API OVERVIEW
X.509 CERTIFICATES
CONSTANT-TIME CRYPTO

In 1996, Paul Kocher published a novel attack on RSA, specifically on RSA *implementations*, that extracted information on the private key by simply measuring the time taken by the private key operation on various inputs. It took a few years for people to accept the idea that such attacks were practical and could be enacted remotely on, for instance, an SSL server; see [this article](#) from Boneh and Brumley in 2003, who conclude that:

Our results demonstrate that timing attacks against network servers are practical and therefore all security systems should defend against them.

Since then, many timing attacks have been demonstrated in lab conditions, against both symmetric and asymmetric cryptographic systems.

Figure: <https://www.bearssl.org/constanttime.html#aes>

Questions?