CRYPTOGRAPHY LECTURE, 2023

Master level

Mirosław Kutyłowski

Encryption -symmetric case

Encryption function:

- $\mathrm{Enc}\colon \mathrm{keyspace} \times \{0,1\}^* \to \{0,1\}^*$
- or: $\mathrm{Enc}\colon \mathrm{keyspace} \times \{0,1\}^m \to \{0,1\}^k$ if Enc defined for blocks of size m

notation: $C := \mathrm{Enc}_K(M)$ means that C is a ciphertext of plaintext M

$$\mathrm{Enc}\colon \mathrm{keyspace} \times \{0,1\}^* \to \{0,1\}^*$$

Decryption function:

- $\mathrm{Dec}\colon \mathrm{keyspace} \times \{0,1\}^* \to \{0,1\}^*$ (or $\mathrm{Dec}\colon \mathrm{keyspace} \times \{0,1\}^k \to \{0,1\}^m$

notation: $M := \mathrm{Dec}_K(C)$

Of course:

$$M = \mathrm{Dec}_K(\mathrm{Enc}_K(M))$$

Encryption as a Random Oracle Permutation

ideal situation: Random Oracle

- on request $\mathrm{Enc}_K(M)$ check if there is a tuple (K, M, X) in the table T
  i. if yes, then return X
  ii. if no, choose Z at random that it is different from all X such that $(K, *, X)$ is already in T, insert (K, M, Z) in T and return Z
- on request $\mathrm{Dec}_K(C)$ check if there is a tuple (K, M, C) in T:
  - if yes , then return M
  - if no, choose U at random that is different from all M such that $(K, M, *)$ is already in T, insert (K, M, Z) in T and return Z

How secure is Encryption?

given ciphertexts $C_1,\ldots,C_n$.

Cryptanalysis: provide **any information** about the corresponding plaintexts $M_1, \ldots, M_n$

- if $\mathrm{Enc}\colon \mathrm{keyspace} \times \{0,1\}^m \to \{0,1\}^m$ then some information leaked: $M_i = M_j \Leftrightarrow C_i = C_j$
- if $\mathrm{Enc}\colon \mathrm{keyspace} \times \{0,1\}^* \to \{0,1\}^*$ then some information leaked via the length
- some pairs (M, C) (plaintext,ciphertext) might be already known
- a priori knowledge about probability distribution of plaintexts available

Full answer from information theoretic point of view:

assign probability to each tuple $(P_0, ..., P_n)$ for :

$$\mathrm{Dec}_K[C_1,..., C_n] = [P_0, ..., P_n]$$

(also taking into account a priori information)

Computational (in)feasibility: typically infeasible due to the number of possibilities for $(P_0, ..., P_n)$

Alternative: list the plaintexts for each key (infeasible, if keylength ≥ 128)

Task: concentrate on some function $F(P_0, ..., P_n)$ treating $P_0, ..., P_n$ as functions of the random variable K (key)

Ideal situation

for each tuple $(P_0, ..., P_n)$ the probability of

$$\mathrm{Dec}_K[C_1,..., C_n] = [P_0, ..., P_n]$$

is the same (or almost the same)

But may be we demand too much?

- some property F (computable and in some sense valuable for cryptanalyst - e.g., $P_0 < P_1$)
- estimate probability that

$$F(\mathrm{Dec}_K[P_0, ..., P_n)) = \mathsf{true}$$

Semantic security - Chosen Plaintext Attack (IND-CPA)

left-or-right game:

1. the adversary chooses messages M_0 and M_1
2. the challenger chooses bit b at random and sets $C := \mathrm{Enc}_K(M_b)$
3. the adversary analyses C and returns b'

the adversary wins if $b' = b$

advantage of adversary is ϵ if: probability to win is $0.5 + \epsilon$

(pbb to win $=0.5$, if the answer is random)

Relation to the former approach

if the adversary may derive a non-trivial property F of the plaintext, then Enc is **not** IND-CPA secure:

1. the adversary chooses messages M_0 and M_1 so that $F(M_0) = \text{false}$, $F(M_1) = \text{true}$
2. the challenger chooses bit b at random and sets $C := \mathrm{Enc}_K(M_b)$
3. the adversary attempts to compute $F(\mathrm{Dec}_K(C))$ (without K !)
   i. if true, then $b' := 1$
   ii. if false, then $b' := 0$

* **Corollary:**

Enc is IND-CPA secure $\Rightarrow$

no nontrivial property of the plaintext may be derived based on plaintexts only

Adaptive chosen-ciphertext attack -- Indistinguishability IND-CCA2

left-or-right game:

1. the adversary chooses arbitrarily $C_1, ..., C_n$, $N_1, ..., N_m$
2. the challenger decrypts $C_1, ..., C_n$ and encrypts $N_1, ..., N_m$
3. the challenger chooses messages M_0, M_1
4. the adversary chooses bit b at random and sets $C := \mathrm{Enc}_K(M_b)$
5. steps 1-2 repeated for new data, but questions about C, M_0, M_1 are ignored
6. the adversary returns b'

the adversary wins if $b' = b$

Converting IND-CPA into IND-CCA2

simple idea: ciphertexts are impossible to guess,

$\Rightarrow$ the adversary will (almost) always get the answer invalid from decryption oracle

Example of suitable encryption: of M

i. $T := M \| \mathrm{Hash}(M)$

ii. $C := \mathrm{Enc}_K(T)$

Probability that a chosen C' will be a valid ciphertext ≈ 0

Version 2: use HMAC instead of Hash

Consequence

$\text{Enc}: \text{keyspace} \times \{0,1\}^m \to \{0,1\}^k$ where $\boldsymbol{k \gg m}$

(reason: information theory)

Keylength - historical remarks

DES algorithm from 70's:

- keylength =64, but
- key = 8 bytes, but each byte contains a parity bit
- effective keylength = 56

countermeasure was: 3DES

$$3\mathrm{DES}_{K,K'}(M) = \mathrm{DES}_K(\mathrm{DES}_{K'}^{-1}(\mathrm{DES}_K(M)))$$

- backwards compatible with DES (set $K' = K$)
- effective keylength =112
- ... still many devices with 3DES used ...

Randomized versus deterministic encryption

problem: if Enc deterministic, then any repetition of plaintext immediately visible

solution: randomized encoding a a plaintext:

M of length $k - \delta$ where $\delta \gg \log(\text{potential number of encryptions of } M)$

i. choose $\mathrm{padding}$ of length δ

ii. calculate $C := \mathrm{Enc}_K(M \| \mathrm{padding})$

While solving one problem we have created another one:

a place for information leakage via **rejection sampling** by malicious encryption device

Rejection sampling:

key Γ shared by encryption device and the adversary

Malicious encryption:

1. choose $\mathrm{padding}$ of length δ
2. calculate $C := \mathrm{Enc}_K(M \| \mathrm{padding})$
3. calculate $Z := \mathrm{HMAC}(\Gamma, C)$
4. if (n most significant bits of Z) $\neq S$, then goto 1
5. output(C)

* **Adversarial decryption:**

1. $S :=$ n most significant bits of $\mathrm{HMAC}(\Gamma, C)$

weakness and strength of rejection sampling

- n must be small, but leaking a few bits per ciphertext is easy
- undetectable as long as Γ unavailable and HMAC is secure

So what to do?

- chaining
- counters
- ...

Some other issues (IND-CCA2 does not authomatically mean security in practice)

- **related keys:** what if similar keys provide similar ciphertexts?
  - **brute force easier** since it suffices to come into a vicinity of the sought key
  - so: **avalanche effect needed**: one bit changed in the key and the result is completely unrelated
- **physical attacks:**
  i. compute $C := \mathrm{Enc}_K(M)$
  ii. set an unknown keybit k_i of K to 1 with a high precision laser, new key $= K'$
  iii. compute $C' := \mathrm{Enc}_{K'}(M)$
  iv. if $C' = C$, then $k_i = 1$, else $k_i = 0$
  v. goto i to learn other unknown keybits

  countermeasure: key stored together with a MAC and correctness of MAC checked at each encryption

Block ciphers

- each plaintext is a block of a fixed length
- Enc: $\{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$

Notes:

- randomization , turning into IND-CCA2 will be handled on top of it
- to solve: how to encrypt shorter and longer messages?
- choice of m: big enough to prevent creating a codebook

  but not too big: otherwise problems with operating systems, complexity of mixing within a block

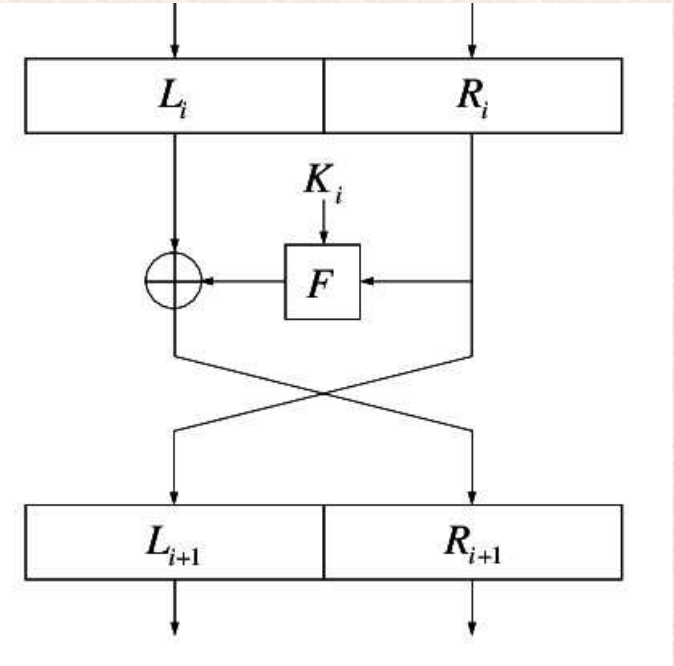
Round concept

- encryption (decryption) consists of a number of identical rounds (with few exceptions)
- each round uses ***round keys***
- total length of round keys $\gg$ keylength, so round keys derived by separate ***KeySchedule*** algorithm

Challenge

- round function should use “one-way’ functions to prevent decription by solving some equations ...
- ... but we need to invert encryption witin the decryption process

Solution: **Feistel architecture**



This figure was uploaded Julio Hernandez-Castro

round i: intermediate state (L_i, R_i)

encryption in round $i+1$:

$$L_{i+1} = R_{i,} \qquad R_{i+1} = L_i \oplus F(R_i, K_i),$$

decryption in the reverse order of rounds:

$$R_i = L_{i+1}, \qquad L_i = R_{i+1} \oplus F(L_{i+1}, K_i).$$

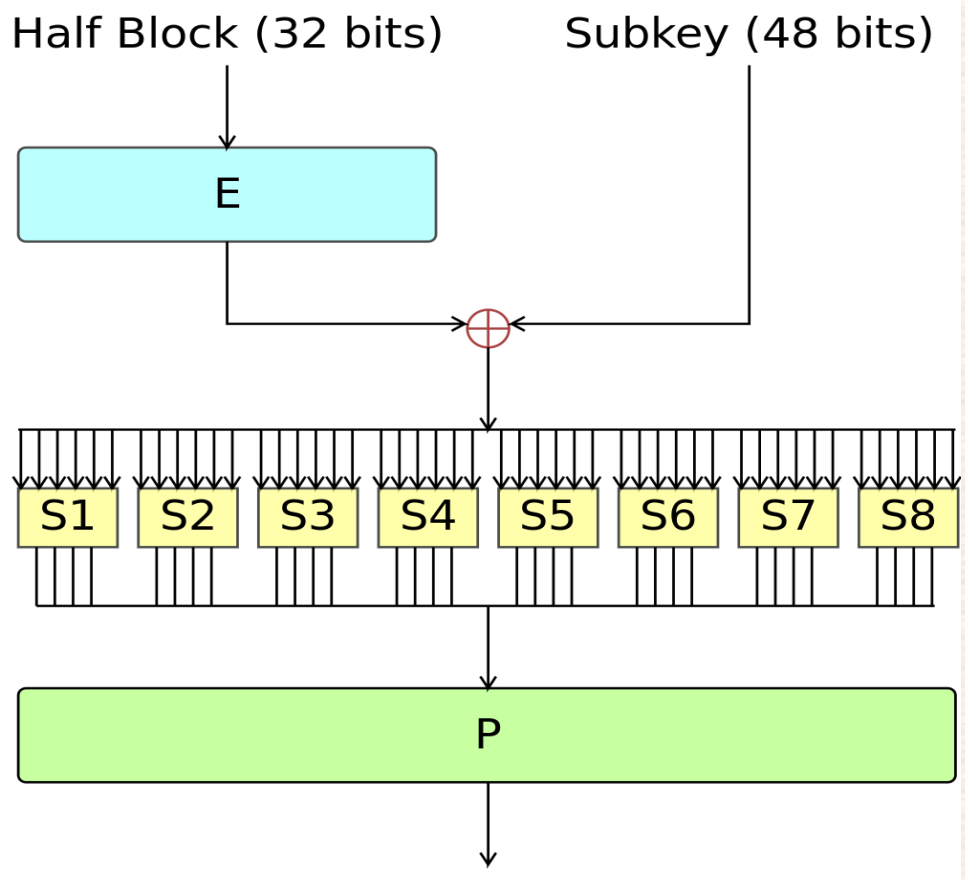
Why it works?

$L_i = R_{i+1} \oplus F(L_{i+1}, K_i)$ iff $R_{i+1} = L_i \oplus F(L_{i+1}, K_i)$

but $L_{i+1} = R_i$ so ... iff $R_{i+1} = L_i \oplus F(R_i, K_i)$

S-Boxes concept – example DES

- DES has Feistel structure
- the crucial point: round F function
- old design: short words
- S-Boxes – very carefully selected functions $\{0,1\}^6 \to \{0,1\}^4$ (so not invertible!)
- complicated interconnections that are easy to route with special purpose hardware but hard for general purpose processors



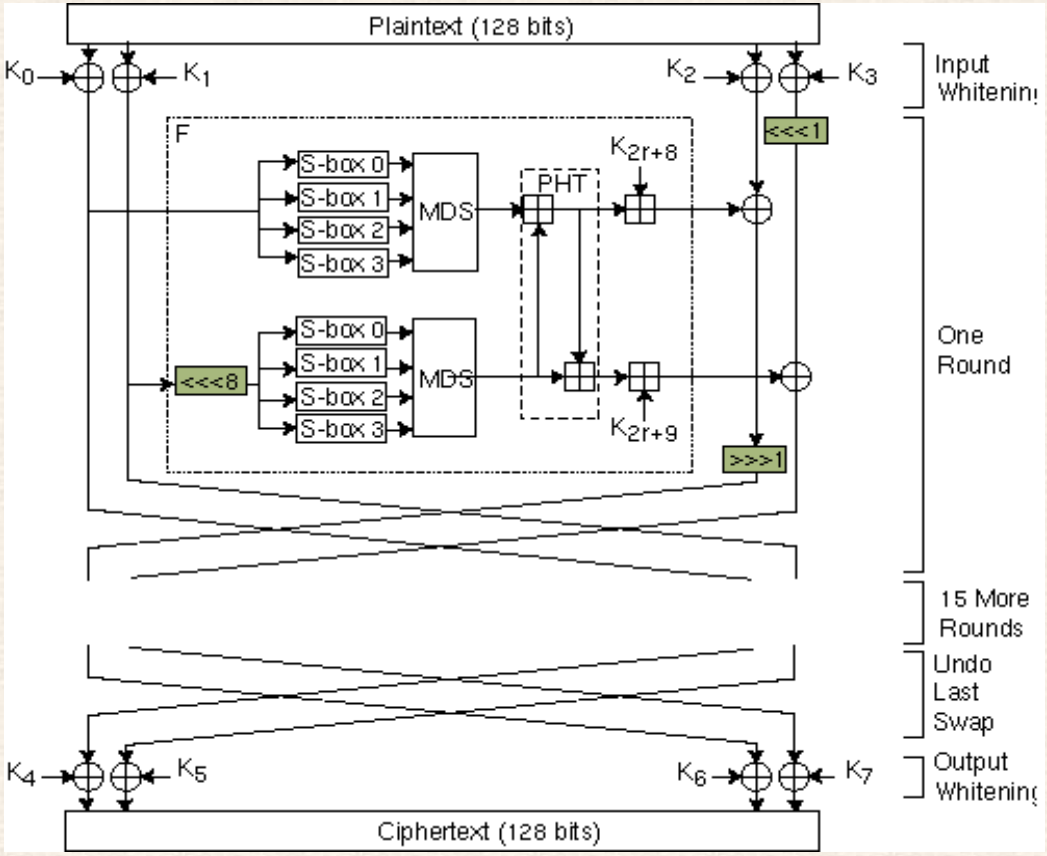
By User:Hellisp - File:Data Encryption Standard InfoBox Diagram.png, CC0, https://commons.wikimedia.org/w/index.php?curid=33316542

AES competition (Advanced Encryption Standard)

finalists:

- Rijndael (winner, as AES)
- MARS, RC6, Serpent, Twofish (based upon Blowfish)

some of them widely available in libraries and popular



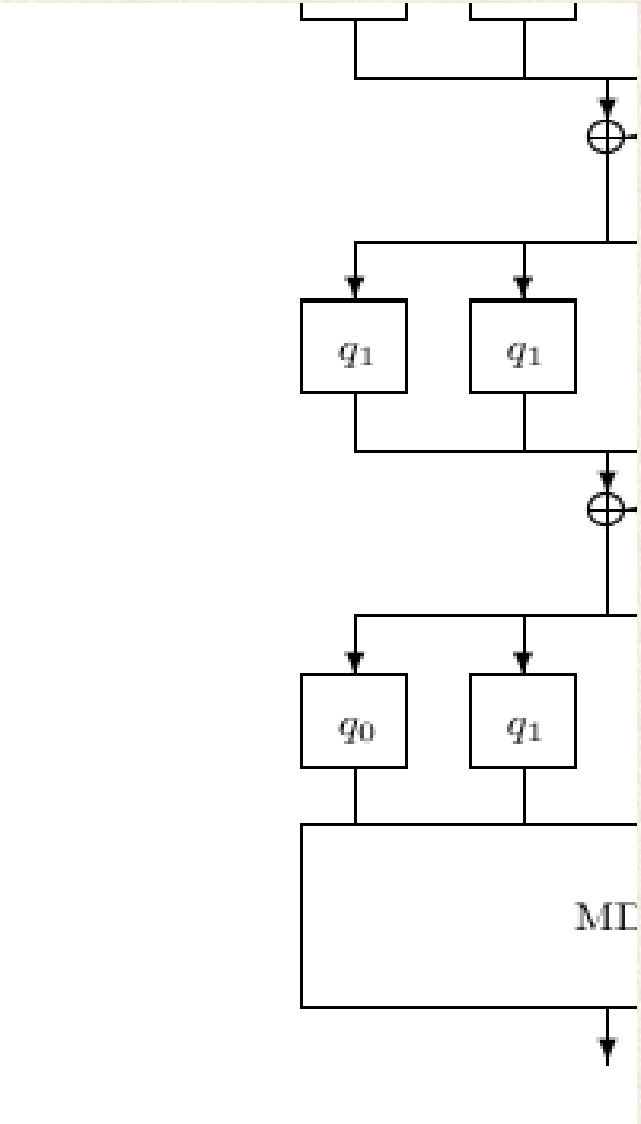
Twofish by Bruce Schneier, whitepaper

MDS - maximum distance separable matrix

PHT - pseudo Hadamard Transformation

four Keyed S -boxes

- bijective functions
- key-dependant (key material precomputed and added during computation)
- using fixed permutations q_0 and q_1 ("carefully chosen")



the Sbox (whitepaper by authors)

Encryption Modes

Electronic Codebook (ECB): $C_i = \mathrm{Enc}_K(P_i)$

not really useful unless each block different (e.g.: contains a block index) :

- if $P_i = P_j$ then $C_i = C_j$

Cipher Block Chaining (CBC)

encryption: $C_0 = \mathrm{IV}, \qquad C_{i+1} = \mathrm{Enc}_K(C_i \oplus P_{i+1})$

decryption: $P_{i+1} = \mathrm{Dec}_K(C_{i+1}) \oplus C_i$

advantages:

- $P_i = P_j$ does not imply that $C_i = C_j$
- C_i depends on $P_1, \ldots, P_i$

disadvantages:

- replacing a single plaintext block requires re-encryption from this block to the end

Malleability

CBC: if the plaintext is known, then one can change every second block to a desired value (every second block would be junk):

- recall that $C_i = E_K(C_{i-1} \oplus P_i)$
- **replace** C_{i-1} with $C_{i-1} \oplus P_i \oplus P'$
- **effect:** then the ith block decrypts to P' (while block P_{i-1} will become junk):

$$\mathrm{New}(P_i) = \mathrm{new}(C_{i-1}) \oplus \mathrm{Dec}_K(C_i) = (C_{i-1} \oplus P_i \oplus P') \oplus (C_{i-1} \oplus P_i) = P'$$

Padding attack (Serge Vaudenay)

Attacked scenario:

- the plaintext should consist of some number of blocks of length $=b$
- padding is always applied (even if unnecessary)
- if i positions have to be padded: the padding consists of i bytes, each of them is i.
- So: removing padding is obvious
- encrypt the resulting padded plaintext $x_1, \ldots, x_N$ in the CBC mode with IV (fixed or random):

$$y_1 = \mathrm{Enc}(\mathrm{IV} \oplus a_1), \quad y_i = \mathrm{Enc}(y_{i-1} \oplus x_i)$$

- properties:
  - efficient
  - warning: do not repeat IV. (if IV fixed, then one can check that two plaintexts have the same prefix)

Attack:

- manipulate cipertext
- destination node decrypts, padding might be incorrect
- **how to react to incorrect padding**? Each reaction will turn out to be wrong:
  - → reaction “reject”: creates padding oracle (attacker can see that some manipulations result in correct padding)
  - → reaction “proceed”: enables manipulation of the plaintext data

option “reject”, last word oracle:

- goal: compute $a = \mathrm{Dec}(y)$ for a block y
- create an input for the padding oracle:
  - create a 2 block ciphertext: $r = r_1 \ldots r_b$ chosen at random, $c := r \,|\, y$
  - oracle call: if $\mathrm{valid}\,(c)$, then $\mathrm{Dec}(y) \oplus r$ should yield a correct padding.
  - whp this happens if $a_b = r_b \oplus 1$ (that is, if the padding consists of a single “1”).
  - other options: suffix “22”, “333”, “4444”,.... are less probable

Fishing out the cases of a longer suffix

- it may happen that the oracle says `valid` because of other correct padding.
- Solution (idea: change consequtive words in the padding until invalid):
  1. pick $r_1, r_2 ..., r_b$ at random, take $i = 0$
  2. put $r = r_1 r_2 ... r_{b-1}(r_b \oplus i)$
  3. run padding oracle on $r|y$, if the result `invalid` then increment i and goto (2)
  4. /* now we have a correct padding of an unknown length
  5. for $j = b$ to 2:

     $r := r_1 ... r_{b-j}(r_{b-j+1} \oplus 1) r_{b-j} ... r_b$

     /* attempting to disturb padding, from left to right

     ask padding oracle for $r|y$, if `invalid` then output $(r_{b-j+1} \oplus j) ... (r_b \oplus j)$ and halt
  6. output $r_b \oplus 1$ /* manipulating all positions except the rightmost one create no error $\Rightarrow$ the padding has length 1, so $y_b \oplus r_b = 1$ or $y_b = r_b \oplus 1$

block decryption oracle

let $a_1 ... a_b$ be the plaintext of y

decryption:

- get a_b via the last word oracle
- proceed step by step learning a_{j-1} once $a_j, ..., a_b$ are already known

  1. set $r_k := a_k \oplus (b - j + 2)$ for $k = j, ..., b$ /* preparing the values so that the padding values $(b - j + 2)$ appear at the end)
  2. set $r_1, ..., r_{j-1}$ at random, $i := 0$ /* search for the value that makes a proper padding
  3. $r := r_1 ... r_{j-2}(r_{j-1} \oplus i) r_j ... r_b$
  4. if output on $r|y$ is `invalid`, then $i := i + 1$ and goto 3
  5. output $r_{j-1} \oplus i \oplus (b - j + 2)$

decryption oracle

- block by block, (after decryption we have to XOR with the previous ciphertext block due to CBC construction)
- the only problem is the first block if IV is secret

BEAST

attack, phase 0:

1. P to be recovered (e.g. a password, cookie, etc), requires ability to force Alice to put secret bits on certain positions
2. force Alice to send a ciphertext of $0...0P_0$ (requires malware on her computer), where P_0 the last byte of P
3. eavesdrop and get $C_p = \mathrm{Enc}(C_{p-1} \oplus 0...0P_0)$
4. guess a byte g
5. force Alice to send encrypted plaintext $C_{i-1} \oplus C_{p-1} \oplus 0...0g$:

   then Alice sends $C_i = \mathrm{Enc}(C_{i-1} \oplus C_{i-1} \oplus C_{p-1} \oplus 0...0g) = \mathrm{Enc}(C_{p-1} \oplus 0...0g)$
6. if $C_i = C_p$ then $P_0 = g$

attack phase 1:

1. P_0 already known
2. force Alice to send $0...0P_0P_1$ and proceed as in phase 0

phases 2-15 until the whole $P_0...P_{15}$ learned

protection: browser must be carefully designed, **injecting plaintexts must be prevented** (SOP- Same Origin Protection).

How to encrypt a disk?

CBC – problems:

- place for initial vector IV
- updates (necessity to re-encrypt from the updated block)
- attacks discussed (malleability etc)

LRW Liskov, Rivest, and Wagner

– $C = \mathrm{Enc}_K(P \oplus X) \oplus X$ ($\oplus$ denotes addition in the field)

where $X = F \otimes I$ ($\otimes$ denotes multiplication in the field)

F is the additional key, I is the index of the block

– **the issue of "red herrings":** encrypting the block $F||0^n$:

$$C_0 = \mathrm{Enc}_K(F \oplus F \otimes 0) \oplus (F \otimes 0) = \mathrm{Enc}_K(F)$$

$$C_1 = \mathrm{Enc}_K(0 \oplus F \otimes 1) \oplus F \otimes 1 = \mathrm{Enc}_K(F) \oplus F$$

so F will be revealed

Xor–encrypt–xor (XEX)

- $X_J = \mathrm{Enc}_K(I) \otimes \alpha^J$
- $C_J = \mathrm{Enc}_K(P \oplus X_J) \oplus X_J$
- I is the sector number, J is the block numer in the sector and α is a generator

XEX-based tweaked-codebook mode with ciphertext stealing (XTS)

- IEEE 1619 Standard Architecture for Encrypted Shared Storage Media
- different key for IV than for encryption ("through misunderstanding XEX specification")
- deals with the sector size not divisible by the block size
- **problem:** no MAC, one can manipulate blocks, something will be recovered!

for the last block (a problem due to fixed size – one cannot use paddings!)

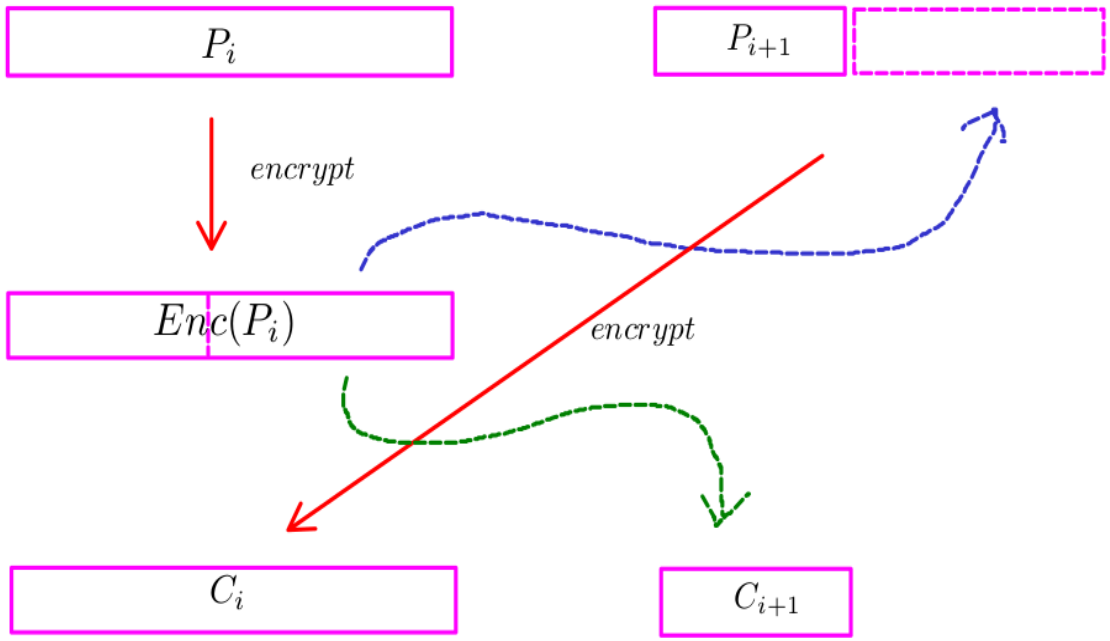
(picture on the next page)

i. expands the k byte plaintext with the last bytes of the ciphertext of the previous block,

ii. the resulting ciphertext stores in place of the ciphertext of the previous block

iii. the ciphertext from the previous block truncated to k bytes and stored as the last ciphertext

for decryption: the missing $n - k$ bytes are recovered from decryption of the ciphertext of the last (originally) block


P_i
P_{i+1}
encrypt
$Enc(P_i)$
encrypt
C_i
C_{i+1}

Format preserving encryption

disk encryption is one of the cases of Format Preserving Encryption:

the size of the output must be exactly the same as the size of the plaintext

example: encrypting credit card numbers in a database

challenge: redesign of block ciphers to small blocks is hardly possible

Generic methods

Random walks

- a sequence of simple transformations determined by the (long) key, each transformation is a permutation
- concept based on a random walk in a (relatively small) graph
- based on concept of rapid mixing of Markov chains and approaching the uniform distribution

Cycling

example: having a block encryption scheme Enc with blocks of length k create a FPE for block length $k-1$:

- append input x with a zero: $x' := x \| 0$
- $c' := \mathrm{Enc}_K(x')$
- if $c' = c\|0$, then output c
- else $c'' := \mathrm{Enc}_K(c')$
- if $c'' = c\|0$ then output c
- else continue in the same way until getting a ciphertext of the form $c\|0$

decryption:

- $c' := c\|0$
- decrypt c' repeatedly until you get a plaintext of the form $p\|0$. Then output p

Problem: this approach does not work as FPE for really short data

Feistel constructions - example

(pict. from Amon et al)

$(L, R) \leftarrow P;$
for $i \leftarrow 0$ **to** 7 **do**
  if $i \mod 2 = 0$ **then**
    $L \leftarrow L \boxplus AES_K(Encode96(R)||T_R \oplus i) \mod M;$
  else
    $R \leftarrow R \boxplus AES_K(Encode96(L)||T_L \oplus i) \mod N;$
return $C \leftarrow L||R;$

$\oplus$ is bitwise XOR, $\boxplus$ is modular addition

FF3 is one of two algorithms recommended by NIST as FPE

ATTACKS on FF3

the attacks are generally of high complexity but for small plaintext size they may be still dangerous

example: message recovery attack

- an unknown plaintext can be encrypted with chosen tweaks (important!)
- idea: characteristics and differential cryptanalysis:
  - → difference only in L: $X = (L, R)$, $X' = (L', R)$
  - → after the first round difference not changed (say $(\Delta, 0)$)
  - → in the second round the output of the round function =0 with probability $1/2^{\text{length of } L}$
  - → ... so with this probability the difference remains $(\Delta, 0)$
  - → final ciphertext difference $(\Delta, 0)$ with a fair pbb
- known L from (L, R), other input (L', R) where L', R are unknown, goal: learn L'
- collect ciphertexts with many different tweaks:
  - → outputs (C, D) and (C', D') with difference $(\Delta, 0)$ yield a candidate $L' = L \otimes \Delta$

Cipher Feedback mode (CFB)

Encryption: $C_0 = \mathrm{IV}$, $C_{i+1} = \mathrm{Enc}_K(C_i) \oplus P_{i+1}$

decryption: $P_{i+1} = C_{i+1} \oplus \mathrm{Dec}_K(C_i)$

Advantages:

- C_i depends on all $P_1, ..., P_i$
- some advantage with encryption rate if P_i's come irregularly

Counter mode (CTR)

encryption

$Y_i = \mathrm{Enc}_K(\mathrm{IV} + f(i))$, where f is some counter function

$C_i = Y_i \oplus P_i$

decryption

$Y_i = \mathrm{Enc}_K(\mathrm{IV} + f(i))$,

$P_i = Y_i \oplus C_i$

(dis)advantages:

- in order to replace P_i by P_i' it suffices to compute

  $C_i := C_i \oplus (P_i \oplus P_i')$

  so only to use together with MAC (authenticated CTR mode)

GCM (The Galois/Counter Mode)

background:

- popular as replacement for CBC mode (due to attacks presented!) and weaknesses of RC4 (now forbidden in TLS)
- fundamental critics already before standardization
- finally (April 2018) Google decided to remove it until April 2019
- operations over GF(2^{128}), addition in the field represented by $\boxplus$

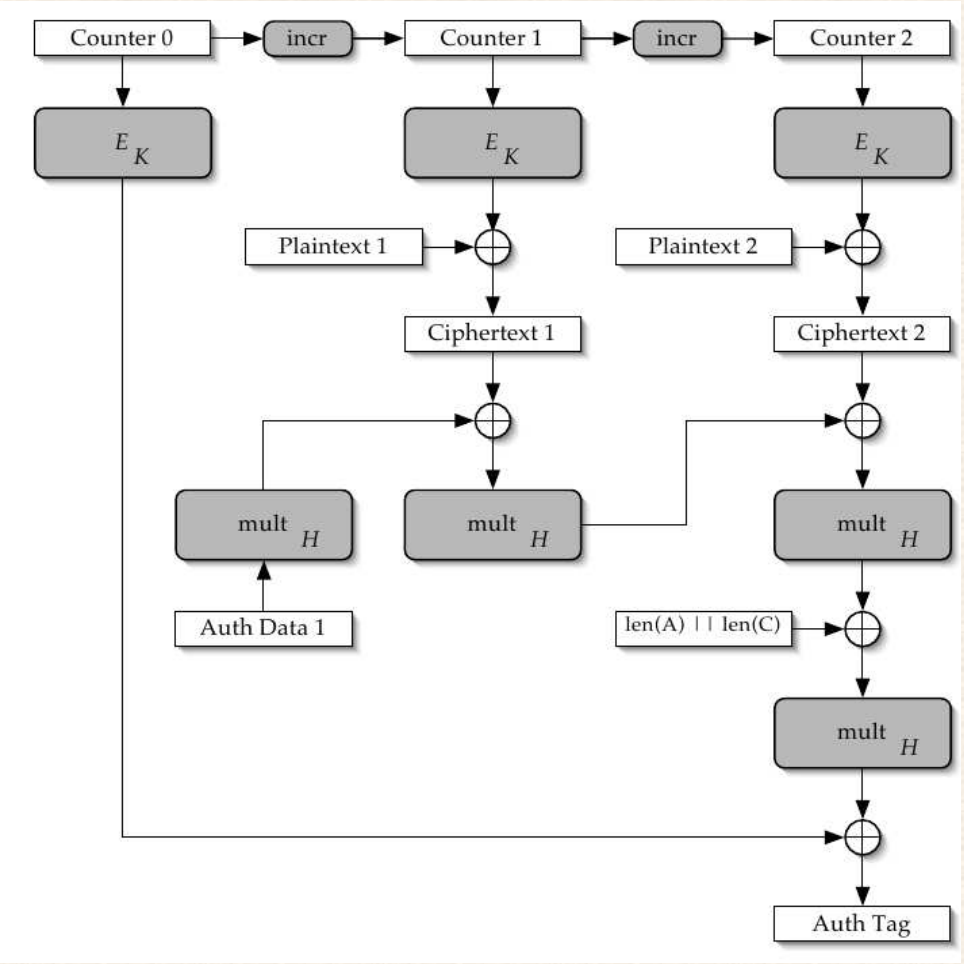
  representation as polynomials of degree 127, operations modulo

  $x^{128} + x^7 + x^2 + x + 1$

Computation:

1. $H := \mathrm{Enc}_K(0^{128})$
2. $Y_0 := \mathrm{IV}||0^{31}1$ if length of IV should be 96

   or $Y_0 := \mathrm{GHASH}(H, \{\}, \mathrm{IV})$
3. $Y_i := \mathrm{incr}(Y_{i-1})$ for $i = 1, ..., n$ (counter computation)
4. $C_i := P_i \oplus \mathrm{Enc}_K(Y_i)$ for $i = 1, ..., n-1$ (counter based encryption)
5. $C_n^* := P_n \oplus \mathrm{MSB}_u(\mathrm{Enc}_K(Y_n))$ (the last block need not to be full)
6. $T := \mathrm{MSB}_t(\mathrm{GHASH}(H, A, C)) \oplus \mathrm{Enc}_K(Y_0)$


Counter 0
incr
Counter 1
incr
Counter 2
E K
E K
E K
Plaintext 1
Plaintext 2
Ciphertext 1
Ciphertext 2
mult H
mult H
mult H
Auth Data 1
len(A) || len(C)
mult H
Auth Tag

(wikipedia)

Details of computation of the tag

$\mathrm{GHASH}(H, A, C) = X_{m+n+1}$ where m is the length of authenticating information A, and:

X_i equals:

0	for $i = 0$
$(X_{i-1} \boxplus A_i) \cdot H$	for $i = 1, \ldots, m-1$
$((X_{i-1} \boxplus (A_m^* \| 0^{128-v})) \cdot H$	for $i = m$
$(X_{i-1} \boxplus C_i) \cdot H$	for $i = m+1, \ldots, m+n-1$
$((X_{m+n-1} \boxplus (C_m^* \| 0^{128-u})) \cdot H$	for $i = m+n$
$((X_{m+n} \boxplus (\mathrm{len}(A) \| \mathrm{len}(C))) \cdot H$	for $i = m+n+1$

Decryption:

1. $H := \mathrm{Enc}_K(0^{128})$
2. $Y_0 := \mathrm{IV}||0^{31}1$ if length of IV should be 96

   or $Y_0 := \mathrm{GHASH}(H, \{\}, \mathrm{IV})$
3. $T' := \mathrm{MSB}_t(\mathrm{GHASH}(H, A, C)) \oplus \mathrm{Enc}_K(Y_0)$, is $T = T'$?
4. $Y_i := \mathrm{incr}(Y_{i-1})$ for $i = 1, ..., n$
5. $P_i := C_i \oplus \mathrm{Enc}_K(Y_i)$ for $i = 1, ..., n$
6. $P_n^* := C_n^* \oplus \mathrm{MSB}_u(\mathrm{Enc}_K(Y_n))$

Fundamental flaws (by Nils Ferguson)

- engineering disadvantages: message size up to $2^{36} - 64$ bytes, arbitrary bit length (instead of byte length)
- collisions of IV: the same pseudorandom string for encryptions
- collisions of Y_0 also possible. Due to birthday paradox 2^{64} executions might be enough for 128-bit values, for massive use in TLS the number of executions 2^{64} is maybe a threat

Ferguson attack via linear behavior

- authenticating tag computed as leading bits of $T = K_0 + \sum_{i=1}^{N} F_i \cdot H^i$ where each F_i is known but H is secret
- representing elements of $\mathrm{GF}(2^{128})$: X – as an abstract element of the field, $\mathrm{Poly}(X)$ – as a polynomial over $\mathrm{GF}(2)$ with coefficients X_0, X_1, ..., X_{127}, multiplication in the field=multiplication of polynomials modulo a polynomial of degree 128
- multiplication by a constant D : $X \to D \cdot X$ can be expressed by multiplication by a matrix:

$$(D \cdot X)^T = M_D \cdot X^T \qquad \text{where } M_D \quad \text{has size } 128 \times 128$$

- squaring is linear: $(A+B)^2 = A^2 + B^2$ (field of characteristic 2), so

$$(X^2)^T = M_S \cdot X^T$$

where M_S is a fixed 128×128 matrix **(important point for the weakness!)**

– the goal is to find a collision, i.e. C' such that

$$\sum_{i=1}^{N} C_i \cdot H^i = \sum_{i=1}^{N} C'_i \cdot H^i$$

or its leading bits (taken to MAC) are the same. Then authentication would fail – one could change the bits in a ciphertext C

– let $C_i - C'_i = E_i$, so we look for a nonzero solution to $\sum_{i=1}^{N} E_i \cdot H^i = 0$

– we confine ourselves to $E_i = 0$ except for i which is a power of 2. Let $D_i = E_{2^i}$. Let $2^n = N$

– we have to to find a solution for

$$E^T = \sum_{i=1}^{n} M_{D_i} \cdot (M_S)^i \cdot H^T$$

where E is an error vector that should become 0

- let

$$A_D = \sum_{i=1}^{n} M_{D_i} \cdot (M_S)^i$$

- then we have $E^T = A_D \cdot H^T$
- write equations to force a row of A_D to be a row of zeros (then in the result the bit of E corresponding to this row is 0), there is an equation for each bit, so 128 linear equations for the whole row
- there are $128 \cdot n$ free variables describing the values D_i (128 for each D_i)
- find a nonzero solution describing the values of D_i so that $n-1$ rows of A_D become rows of zeroes
- consider messages of length 2^{17}, $D_0 = 0$ due to issues like not changing the length
- $D_1, D_2, .., D_{17}$ can be chosen so that 16 rows of A_D are zero,
- GCM used with 32 bits MAC, so still 16 bits might be non-zero, so the chance of forgery is 2^{-16}
- **Bad news: after collision found one can also find H! Then the game is over**

Poly1035

- designed by D. Bernstein, no patent
- 16 byte MAC, variable message length, 16 byte nonce
- works with AES, but AES can be replaced
- the only way to break Poly is to break AES
- keys: k - for AES, r - little endian 128-bit number
- some limitations on r because of efficiency of implementation

  $r = r_0 + r_1 + r_2 + r_3$ where

  $r_0 \in \{0, 1, ..., 2^{28} - 1\}$, $r_1/2^{32} \in \{0, 4, 8, 12, ..., 2^{28} - 4\}$,...

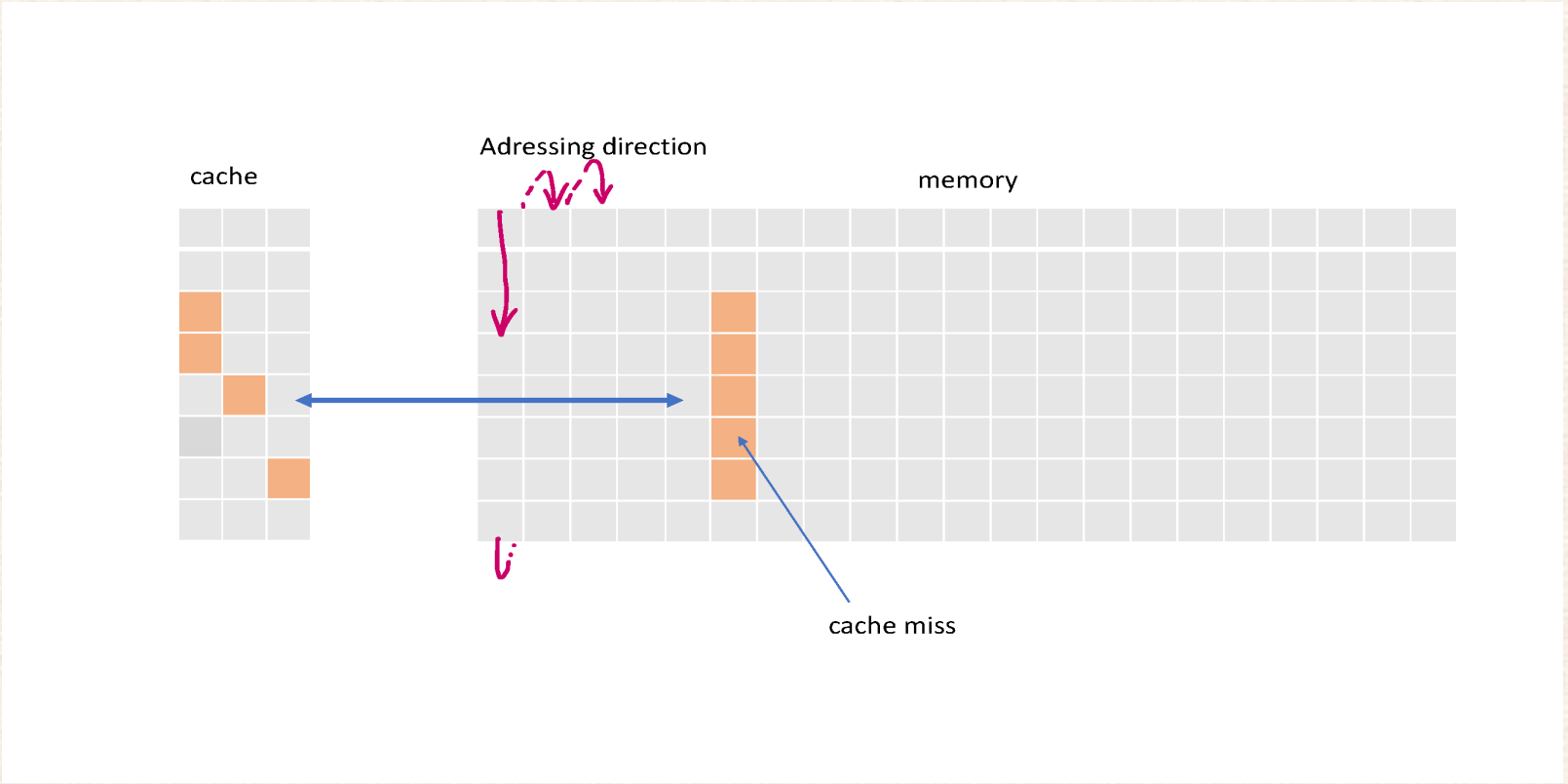
- message: divided into 16 byte chunks. Each chunk treated as a 17-byte number with little-endian, where the most significant byte is an added 1, the result for a message is: $c_1, \ldots, c_q$
- authenticator

$$(((c_1\, r^q + c_2\, r^{q-1} + \ldots + c_q\, r^1) \bmod 2^{130} - 5) + \mathrm{AES}_k\,(\mathrm{nonce})) \bmod 2^{128}$$

  denoted also $H_r(m) + \mathrm{AES}_k(\mathrm{nonce})$
- $2^{130} - 5$ is a prime,
- a nonce must be usely once
- security: for random messages m, m' of length L pbb that $H_r(m) = H_r(m') + g$ is at most $8\lceil L/16 \rceil / 2^{108}$ (all differentials have small probability)

Cache attacks

cache architecture: simple addressing, if data kept in the cache, then response is almost immediate



subsequent blocks from memory go to different cache lines

- think about implementations based on lookup tables
- the attack aims to learn which parts of the lookup table has been used
- $\Rightarrow$ this might say what are the intermediate values of some variables
- **target situation:** $c = k \oplus p$, where p is some plaintext byte(s), c learned through calls to lookup table . Then we learn k

cache measurement strategy: Evict+Time

i. procedure:

   1. trigger encryption of a plaintext p
   2. **evict:** access memory addresses so that **one cache set overwritten** completely
   3. trigger encryption of the plaintext p

ii. in the evicted cache set one cache line from T_l **is missing**

iii. measure time: if long, then cache miss and the encryption refers to evicted δ positions from the lookup table

iv. practical problem: triggering may invoke other activities and timing is not precise

measurement: Prime+Probe

i. procedure

1. **prime:** overwrite entire cache by reading A: a contiguous memory of the size of the cache
2. trigger an encryption of p – it results in **eviction** at places where lookup has occurred
3. **probe:** read memory addresses of A and **detect which locations have been evicted**

ii. easier: probe timing checked, not the time at encryption

- **plaintext random but known**, corresponds to the situation where one can trigger encryption (e.g. VPN with unknown key, dm-crypt of Linux)
- phase 1: measurements, phase 2: analysis
- from experiments: AES key recovered using 65 ms of measurements (800 writes) and 3 sec analysis

AES software implementation:

- particularly vulnerable because of its design
- AES defined in algebraic terms, but lookup table is typically faster
- there are arguments against algebraic implementations as the execution time may provide a side channel
- key expansion: round zero: simply the key bytes directly, other rounds: key expansion reversable (details irrelevant for the attack)
- fast implementation based on lookup tables T_0, T_1, T_2, T_3 and $T_0^{(10)}, T_1^{(10)}, T_2^{(10)}, T_3^{(10)}$ for the last round (with no `MixColumns`)

- round operation

$$\left(x_0^{(r+1)}, x_1^{(r+1)}, x_2^{(r+1)}, x_3^{(r+1)}\right) := T_0(x_0^r) \oplus T_1(x_5^r) \oplus T_2(x_{10}^r) \oplus T_3(x_{15}^r) \oplus K_0^{(r+1)}$$

$$\left(x_4^{(r+1)}, x_5^{(r+1)}, x_6^{(r+1)}, x_7^{(r+1)}\right) := T_0(x_4^r) \oplus T_1(x_9^r) \oplus T_2(x_{14}^r) \oplus T_3(x_3^r) \oplus K_1^{(r+1)}$$

$$\left(x_8^{(r+1)}, x_9^{(r+1)}, x_{10}^{(r+1)}, x_{11}^{(r+1)}\right) := T_0(x_8^r) \oplus T_1(x_{13}^r) \oplus T_2(x_2^r) \oplus T_3(x_7^r) \oplus K_2^{(r+1)}$$

$$\left(x_{12}^{(r+1)}, x_{13}^{(r+1)}, x_{14}^{(r+1)}, x_{15}^{(r+1)}\right) := T_0(x_{12}^r) \oplus T_1(x_1^r) \oplus T_2(x_6^r) \oplus T_3(x_{11}^r) \oplus K_3^{(r+1)}$$

attack notation:

- $\delta = B/\text{entrysize of lookup table}$, typically: entrysize=4bytes, $\delta = 16$, (so δ entries of a lookup table are within the same cache line – this is a complication for the attack!)
- for a byte y let $\langle y \rangle = \lfloor y/\delta \rfloor$, it indicates a memory block of y in T_l
- if $\langle y \rangle = \langle z \rangle$, then x and y correspond to requests to the same memory block of the lookup table and therefore to the same cache line

“synchronous attack”

- **plaintext random but known**, corresponds to the situation where one can trigger encryption (e.g. VPN with unknown key, dm-crypt of Linux)
- phase 1: measurements, phase 2: analysis
- from experiments: AES key recovered using 65 ms of measurements (800 writes) and 3 sec analysis

attack on round 1:

i. lookup tables queried for $x_i^{(0)} = p_i \oplus k_i$ for $i = 0, \ldots, 15$

ii. goal: find information $\langle k_i \rangle$ – one cannot derive lsb; candidates for k_i denoted by $\bar{k_i}$

iii. if $\langle k_i \rangle = \langle \bar{k_i} \rangle$ and $\langle y \rangle = \langle p_i \oplus \bar{k_i} \rangle$, then block y of T_l queried

iv. if $\langle k_i \rangle \neq \langle \bar{k_i} \rangle$, then there is no lookup in block y for T_l during the first round, **but**

- there are $4 \cdot 9 - 1 = 35$ other queries affected by other plaintext bits during the entire encryption - 4 per round, 9 rounds in total (the last round uses different look-up tables)
- probability that none of them accesses block y for T_l is

  $\left(1 - \frac{\delta}{256}\right)^{35} \approx 0.104$ for $\delta = 16$

v. few dozens of samples required to find a right candidate for $\langle k_i \rangle$

vi. together we determine $\log(256/\delta) = 4$ bits of each byte of the key

vii. no more possible for the first round, still 64 key bits to be found, so one cannot do the rest with a brute force

viii. in reality more samples needed due to noise in detection of cache misses

attack on round 2: the goal is to find the still unknown key bits

i. we exploit equations derived from the Rijndeal specification:

$$x_2^{(1)} = s(p_0 \oplus k_0) \oplus s(p_5 \oplus k_5) \oplus 2\bullet s(p_{10} \oplus k_{10}) \oplus 3\bullet s(p_{15} \oplus k_{15}) \oplus s(k_{15}) \oplus k_2$$

$$x_5^{(1)} = s(p_4 \oplus k_4) \oplus 2\bullet s(p_9 \oplus k_9) \oplus 3\bullet s(p_{14} \oplus k_{14}) \oplus s(p_3 \oplus k_3) \oplus s(k_{14}) \oplus k_1 \oplus k_5$$

$$x_8^{(1)} = \ldots.$$

$$x_{15}^{(1)} = \ldots.$$

where s stands for the Rijndael Sbox, and $\bullet$ means multiplication in the field with 256 elements

ii. lookup for $T_2\left(x_2^{(1)}\right)$:

- $\langle k_0 \rangle, \langle k_5 \rangle, \langle k_{10} \rangle, \langle k_{15} \rangle, \langle k_2 \rangle$ already known
- low level bits of $\langle k_2 \rangle$ influence only low bits of $x_2^{(1)}$ so not important for cache access pattern
- the upper bits of $x_2^{(1)}$ can be determined after guessing low bits of k_0, k_5, k_{10}, k_{15}: there are δ^4 possibilities ($=16^4$)
- a correct guess yields a lookup in the right place

- an incorrect guess: some $k_i \neq \bar{k_i}$ so

  $$x_2^{(1)} \oplus \bar{x}_2^{(1)} = c_i \bullet s(p_i \oplus k_i) \oplus c_i \bullet s(p_i \oplus \bar{k_i}) \oplus \ldots$$

  where ... depends on different random plaintext bits and therefore random
- differential properties of AES studied for AES competition:

  $$\Pr\left[\, c_i \bullet s(p_i \oplus k_i) \oplus c_i \bullet s(p_i \oplus \bar{k_i}) \neq z\right] > 1 - \left(1 - \frac{\delta}{256}\right)^3$$

  so probability for a wrong guess:
  - $\left(1 - \frac{\delta}{256}\right)^3$ for not computing $T_2\left(x_2^{(1)}\right)$
  - $\left(1 - \frac{\delta}{256}\right)$ for not referring to the same cache line as $T_2\left(x_2^{(1)}\right)$ for other 35 queries to T_2
  - together no access to this block of T_2 with pbb about $\left(1 - \frac{\delta}{256}\right)^{38}$
- this yields about 2056 samples necessary to eliminate all wrong candidates
- it has to repeated 3 more times to get other nibbles of key bytes

iii. optimization: guess $\Delta = k_i \oplus k_j$ and take $p_i \oplus p_j = \Delta$, then i.e. $s(p_0 \oplus k_0) \oplus s(p_5 \oplus k_5)$ cancels out and we have to guess less bits (4 instead of 8)

- **complications in practice:**

  i. **address of lookup tables in the memory** - offset uknown, has to be found by considering all offsets and then statistics for each offset (experiments show good results even in a noisy environment)

  ii. **hardware prefetcher** may disturb the effects. Solution: read and write the addresses of A according to a pseudorandom permutation

- **practical experiments:** e.g. Athlon 64, no knowledge of adresses mapping, 8000 encryptions with Prime & Probe

  Linux dm-crypt (disk, filesystem, file encryption): with knowledge of addressing, 800 encryptions (65 ms), 3 seconds analysis, full AES key

“asynchronous attrack” on round 1

- no knowledge of plaintext, no knowledge of ciphertext
- based on frequency F of bytes in e.g. English texts, frequency score for each of $\frac{256}{\delta}$ blocks of length δ
- F is nonuniform: most bytes have high nibble = 6 (lowercase characters “a” through “o”)
- find j such that j is particularly frequent indicates $j = 6 \oplus \langle k_i \rangle$ and shows $\langle k_i \rangle$
- complication: this frequency concerns at the same time k_0, k_5, k_{10}, k_{15} affecting T_0 so we learn 4 nibbles but not their actual allocation to k_0, k_5, k_{10}, k_{15}
- the number of bits learnt is roughly: $4 \cdot (4 \cdot 4 - \log 4!) \approx 4 \cdot (16 - 3.17) \approx 51$ bits
- experiment: OpenSSL, measurements 1 minute, 45.27 info bits o on the 128-bit key gathered

Countermeasures - all controversial and not satisfactory

- implementation based on **no-lookup** but algebraic algorithm (slow!!!) or bitslice implementation (sometimes possible and nearly as efficient as lookup)
- **alternative lookup tables:** if smaller then smaller leakage (but easier cryptanalysis for small Sboxes)
- **data-independent access to memory** blocks - every lookup causes a redundant read in all memory blocks, generally: oblivious computation possible theoretically, but overhead makes it rather useless
- **masking operations:** $\approx$"we are not aware of any method that helps to resist our attack"
- **cache state normalization:** load all lookup tables - equires deep changes in OS and reduces efficiency, even then LRU cache policy may leak information which part has been used!
- **process blocking:** again, deep changes in OS
- d**isable cache sharing**: deep degradation of performance

- **"no-fill" mode** during crypto operations:
  - preload lookup tables
  - activate "no-fill"
  - crypto operation
  - deactivate "no-fill"

  the first two steps are critical and no other process is allowed to run

  possible only in priviledged mode, cost of operation prohibitive

- **dynamic table storage:** e.g. many copies of each table, or permute tables

  details architecture dependent and might be costly

- **hiding timing information:** adding random values to timing makes the statistical analysis harder but still feasible

- **protect some rounds** (the first 2 and the last one) with any mean – but may be there are other attack techniques...

- **cryptographic services at system level**: good but unflexible

- **sensitive status for user processes:** erasing all data when interrupt

- **specialized hardware support:** crypto co-processor seems to be the best choice

  but the problem is not limited to AES or crypto – many sensitive data operations are not cryptographic and a coprocessor does not help