Crypto

Jan Mazur

Hej, ale nie opowiedziałeś dokładnie o X!

Not so crypto - szyfry podstawieniowe

Kawałki tekstu zastępowane kawałkami szyfrogramu według danej, stałej reguły.

- monoalfabetyczny, podstawienie 1-1
  - ROTx - skończona, mała liczba kluczy
  - w ogólności dla alfabetu dł. 26 - 26! ~ 4e26 ~ dużo.
  - analiza częstotliwościowa, ataki słownikowe
  - guballa.de/substitution-solver
- polialfabetyczny, podstawienie 1-1
  - szyfr Vigenere'a, xor - github.com/hellman/xortool
  - typowanie długości klucza
    - Kasiski Examination
    - **abcde**abcdeabcde**abcde**abcdeabcdeabc
    - **crypto** is short for **crypto**graphy.
    - Index of Coincidence
    - odległość Hamminga
  - analiza częstotliwościowa, ataki słownikowe
  - guballa.de/vigenere-solver
  - guballa.de/implementierung-eines-vigenere-solvers

Demo #1

xortool vs XorXor, XorXorXor

Kryptograficzne dziury

W zdecydowanej większości przypadków wynikają z błędnego użycia kryptosystemu oraz jego nieznajomości.

Here’s Johnny!

Narzędzia automatyzujące łamanie haszy i haseł wielu możliwych standardów.

- John the Ripper - openwall.com/john
- hashcat - hashcat.net

wyliczają kolizje?

Generatory liczb pseudolosowych

Kryptograficznie bezpieczne pseudolosowe RNG:

- /dev/urandom
- CryptGenRandom

Pseudolosowe RNG:

- Mersenne Twister
- Knuth Subtractive
- Wichmann-Hill
- Linear Congruential Generator (LCG)

PRNG są deterministyczne. Znając stan przewidujemy kolejne losowania.

github.com/altf4/untwister - wielowątkowe narzędzie do odzyskiwania ziarna z różnych PRNG

CSPRNG i PRNG w językach programowania

Language	Method	PRNG
.NET	`System.Random()`	Knuth Subtractive
Java	`java.util.Random()`	LCG
PHP	`mt_rand()`	Mersenne Twister
Python	`random.random()`	Mersenne Twister

Language	CSPRNG
.NET	`RNGCryptoServerProvider()`
Java	`java.security.SecureRandom()`
JavaScript (Node.js)	`crypto.RandomBytes()`
PHP	`random_bytes()`
Python	`random.SystemRandom()` `os.urandom()`

rand() z libc używa LCG

CS

active

wister

wister

er()

dom()

)

()

Zadanka

```
#include <iostream>

char *flag = "flg{*****}";

int main(){

        srand(truly_random_seed_from_urandom);
        printf("%d\n", rand());

        for(int i=0; i<5 i++){
                int s;
                scanf("%d", &s);
                if(s != rand()){
                        printf("you failed!\n");
                        exit();
                }
        }

        printf("Flag is: %s\n", flag);

        return 0;
}
```

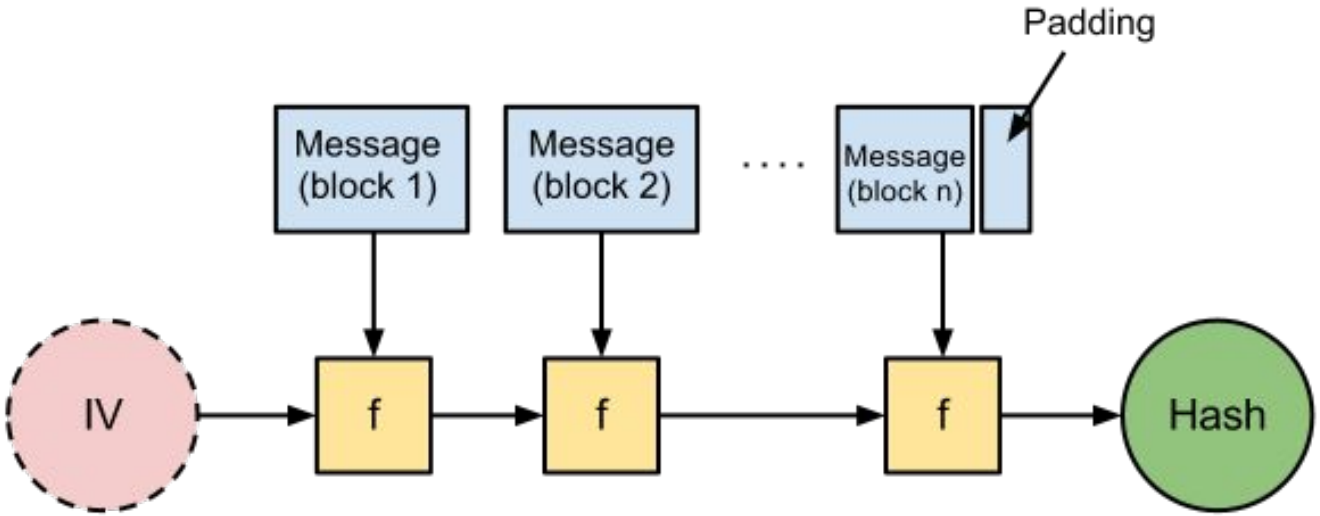
angstrom ctf 2018 ofb crypto120 - LCG

meepwn-quals-2018-Still OldSchool - Mersenne Twister

0ctf - rand2 - C rand()

hash length extension

“100% of the state needed to continue a hash is in the output of most hashing algorithms!” - nie dla HMAC



http://joncave.co.uk/2012/08/i-captured-the-flag/

H(secret ‖ message) => H(secret || message || append) bez znajomości secret

github.com/iagox86/hash_extender

hash length extension - przykład

Original Data: count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo
Original Signature: 6d5f807e23db210bc254a28be2d6759a0f5f5d99

Desired New Data: count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo**&waffle=liege**

New Data: count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo**\x80\x00\x00**

\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00

\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00

\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x02\x28&waffle=liege

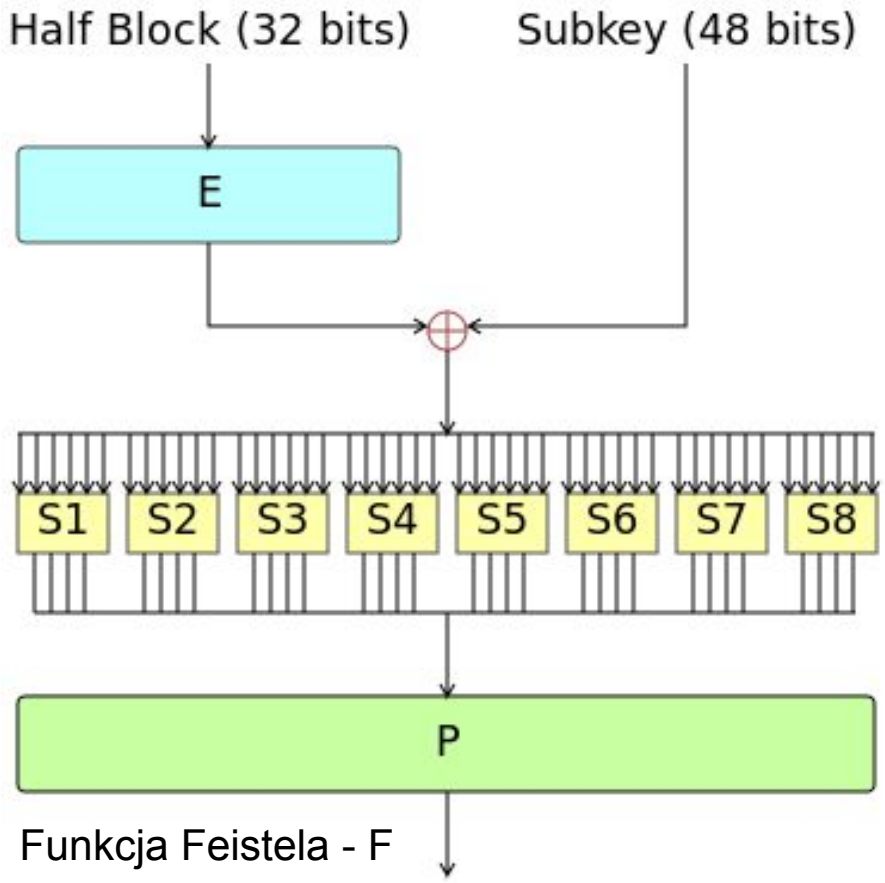
New Signature: 0e41270260895979317fff3898ab85668953aaa2

Trzeba znać długość tego co jest hashowane, w szegółności długość sekretu aby dobrze dobrać padding.

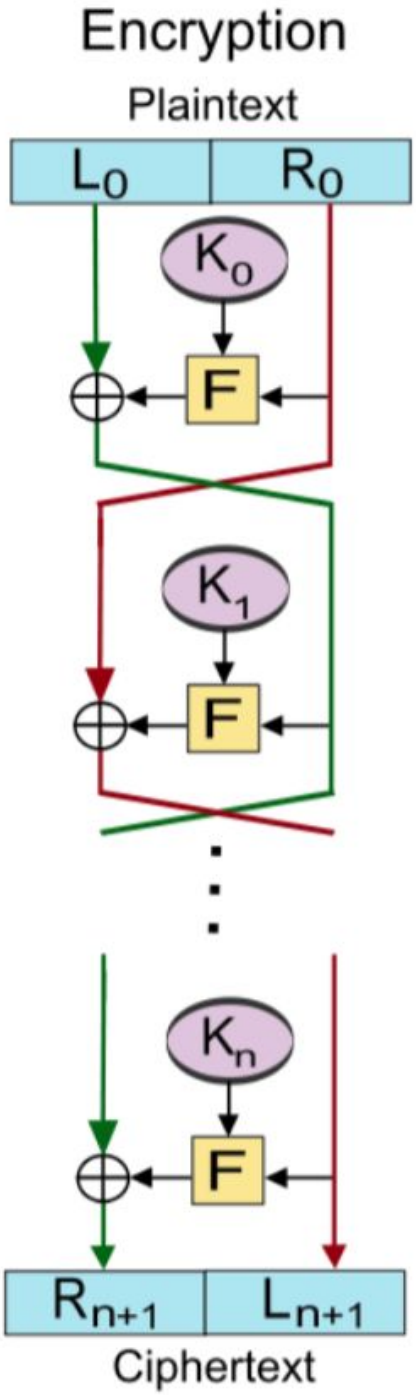
Padding to bajt 0x80, po nim bajty 0x00 i bajty kodujące długość paddingu. Ilość bajtów zerowych i ilość bajtów zarezerwowanych na kodowanie długości zależy od algorytmu.

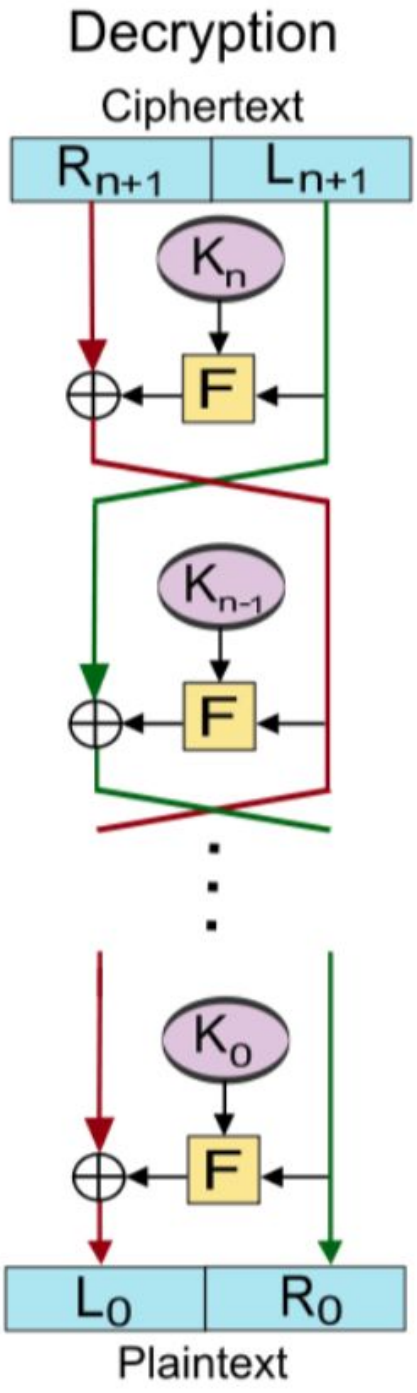
Sieć Feistela - DES

SIAM Journal on Computing, 1988, Vol. 17, No. 2 : pp. 373-386



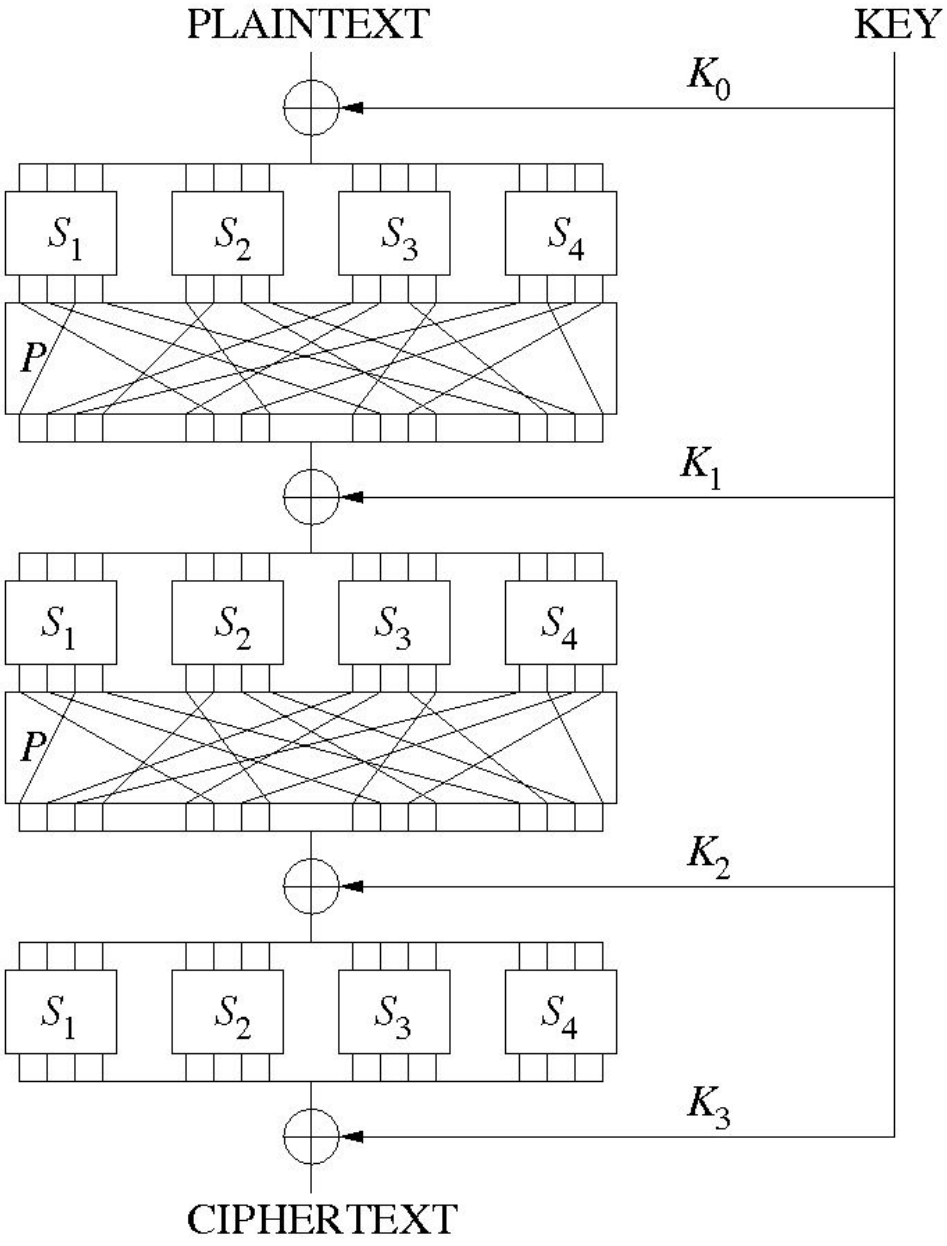
Funkcja Feistela - F





Substitution-permutation network

3-rundowa sieć, szyfrująca
16-bitowy blok tekstu do 16-bitów szyfrogramu.



S-Box (DES 6 do 4)

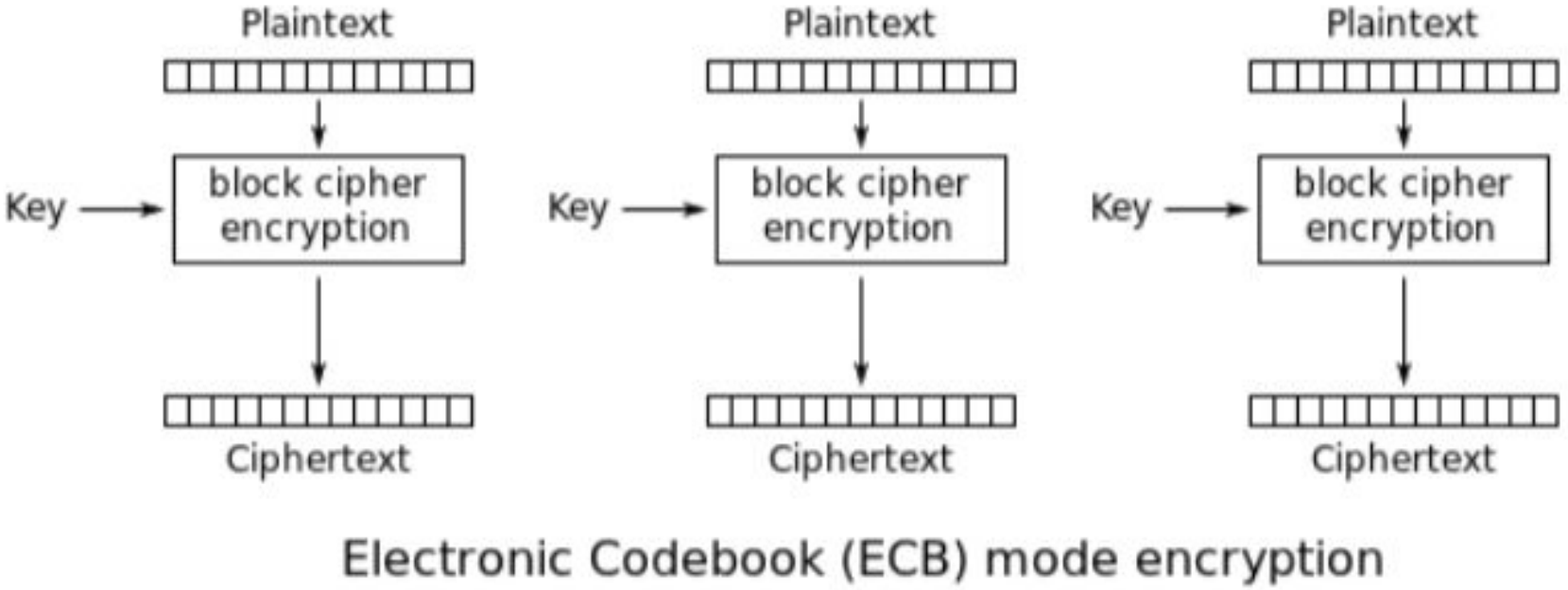
S-boxy DESa to jedne z najlepiej przebadanych “tablic”.

Zastanawiano się:

- czy mają backdoora?
- jak powinny wyglądać?
- czy drobne zmiany znacznie osłabiają algorytm?

S_5		Middle 4 bits of input															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Outer bits	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

Wiązanie bloków



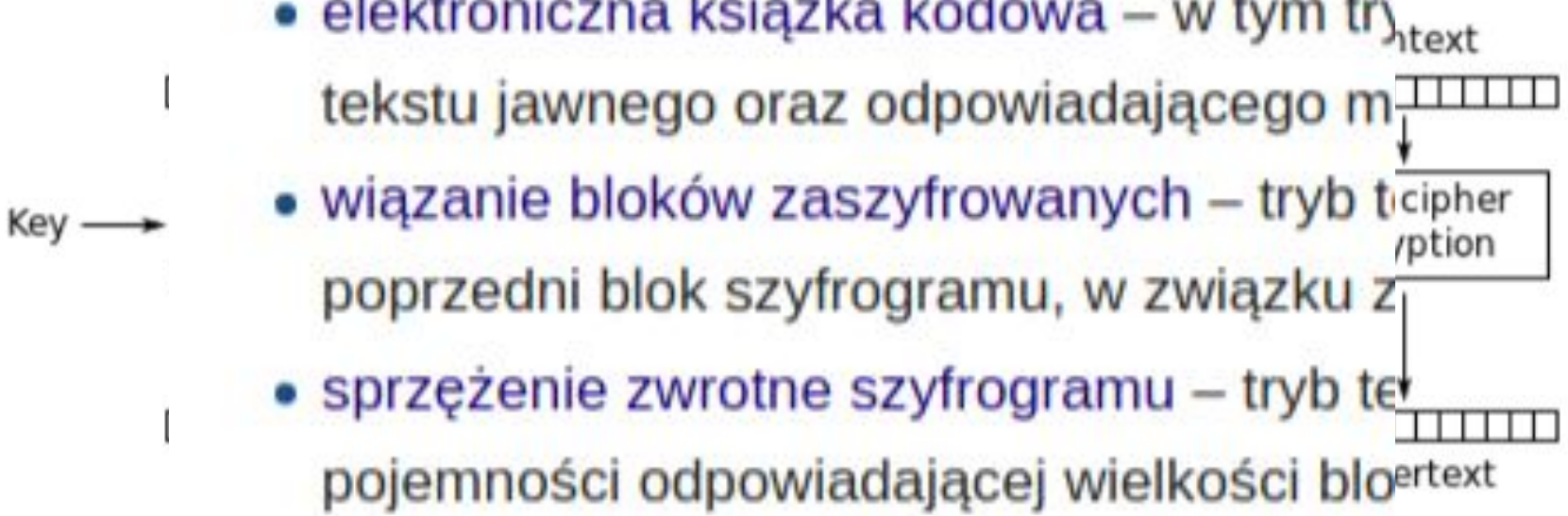
Electronic Codebook (ECB) mode encryption

Demo ECB

[picoCTF2018 SpyFi - Points: 300]

Wiązanie bloków

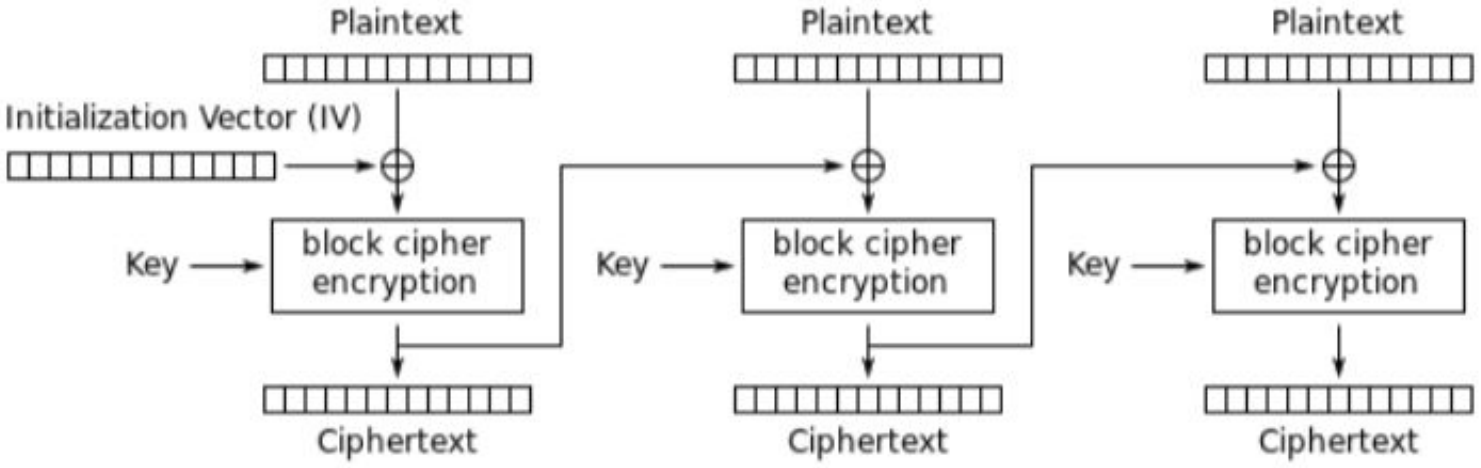
- elektroniczna książka kodowa – w tym tr
  tekstu jawnego oraz odpowiadającego m
- wiązanie bloków zaszyfrowanych – tryb t
  poprzedni blok szyfrogramu, w związku z
- sprzężenie zwrotne szyfrogramu – tryb te
  pojemności odpowiadającej wielkości blo
  bitów z rejestru sumowanych jest modulo
- sprzężenie zwrotne wyjściowe – tryb ten
  bloku wyjściowego, a nie szyfrogramu



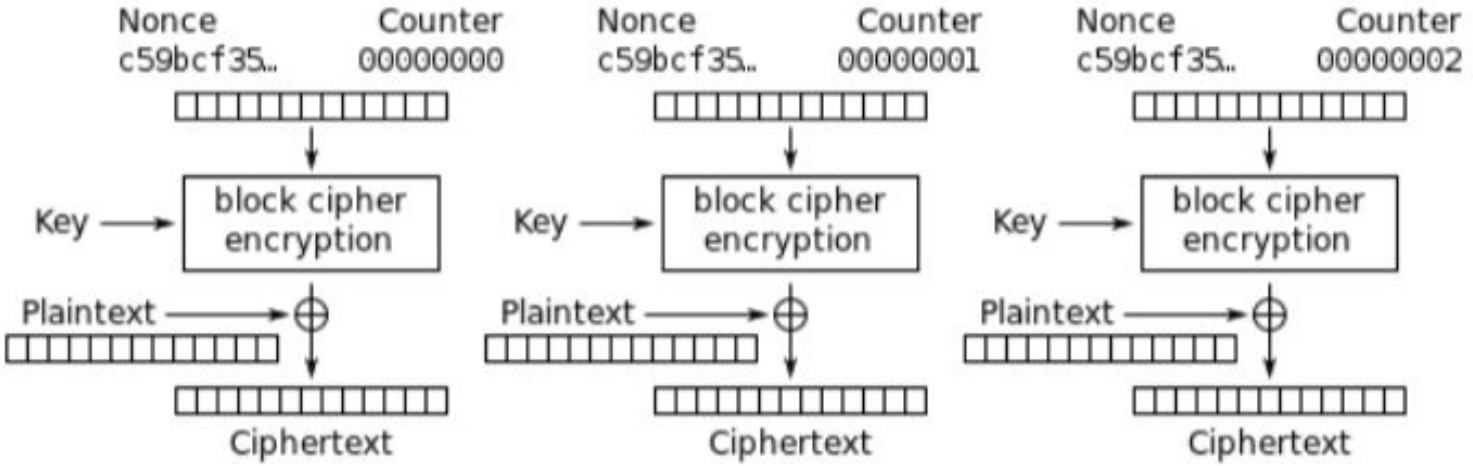
Demo E

picoCTF

CBC, CTR



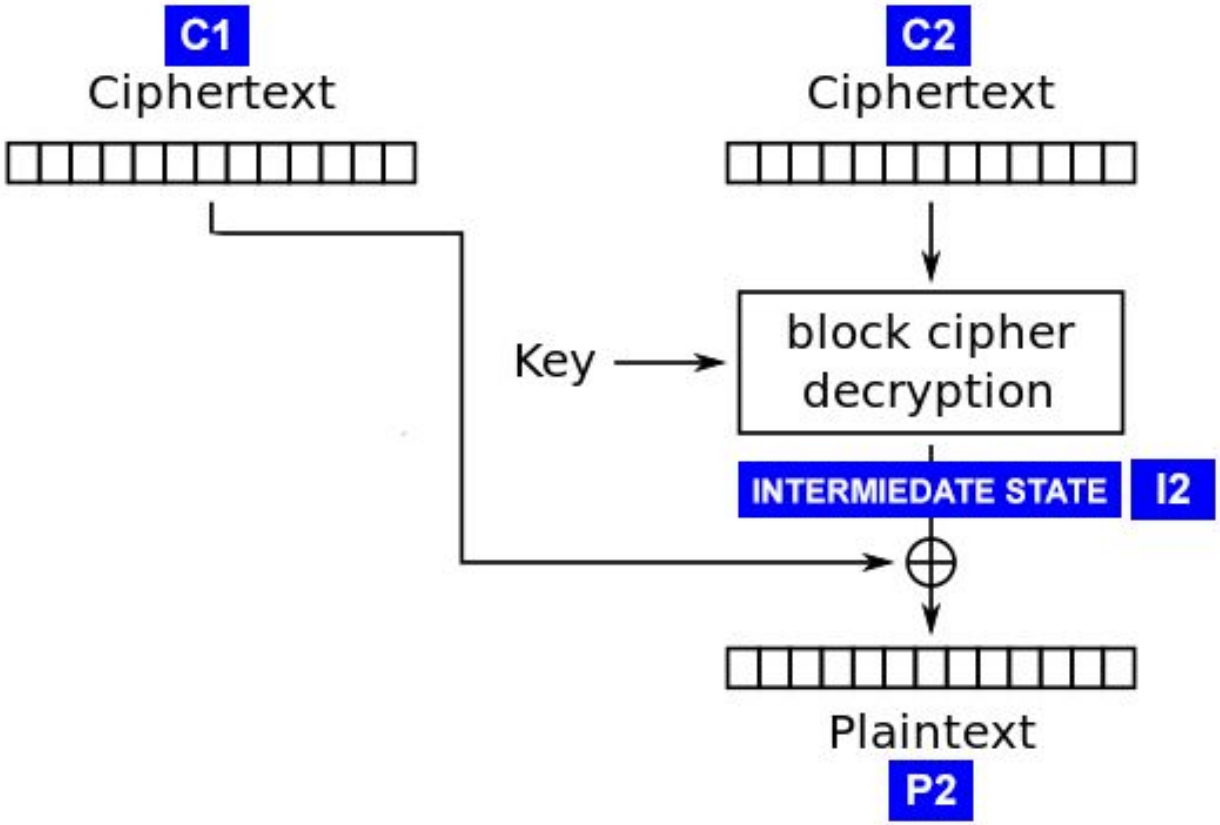
Cipher Block Chaining (CBC) mode encryption



Counter (CTR) mode encryption

oracle padding - deszyfrowanie

- padding PKCS#7
  - pad_m = m + (16 - len(m) % 16) * chr(16 - len(m) % 16)
- Wyrocznia mówi czy dany ciphertext ma poprawny padding



Stan pośredni

P2 = C1 ⊕ I2

Mamy kontrolę nad Ci.

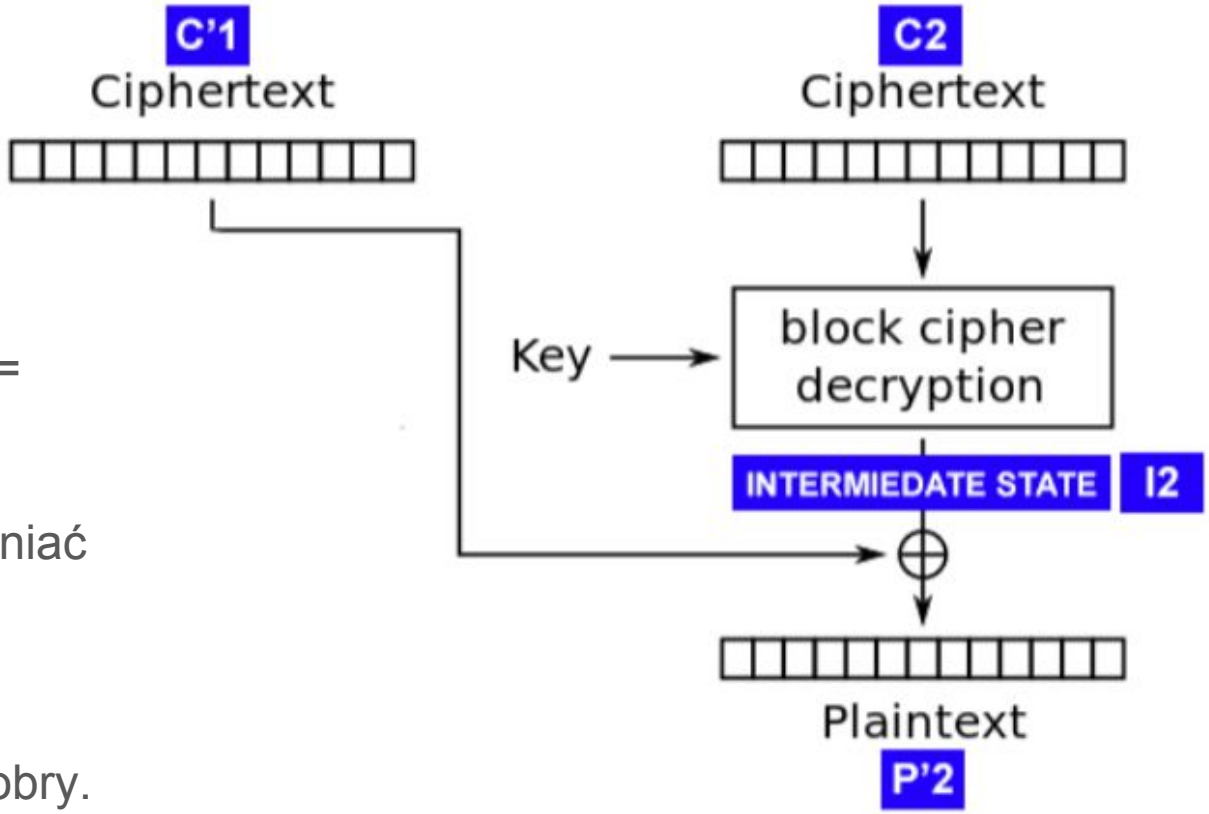
Chcąc odgadnąć C2[16]:

Wysyłamy C1’ t. że C1[16] = [0,255].

Będziemy dzięki temu zmieniać I2[16], czyli P2[16].

Serwer (co najmniej) raz odpowie, że padding jest dobry.

Wtedy P2[16] == 01!



oracle padding attack - szyfrowanie

- Atak znajduje $I_i = D_k(C_i)$ dla i-tego bloku w danym ct.
- Ustawiamy odpowiednio C_i (jeżeli i = n to na dowolną wartość)
- Za pomocą wyroczni dostajemy I_i
- $P_i = I_i \oplus C_{i-1}$ => wyliczamy C_{i-1}
- Powtarzamy powyższe kroki.

Demo:

I am the oracle - eliminacje do European Cyber Security Challenge - Seniorzy

picoCTF2018 - Magic Padding Oracle - Points: 450

CTR zadanka

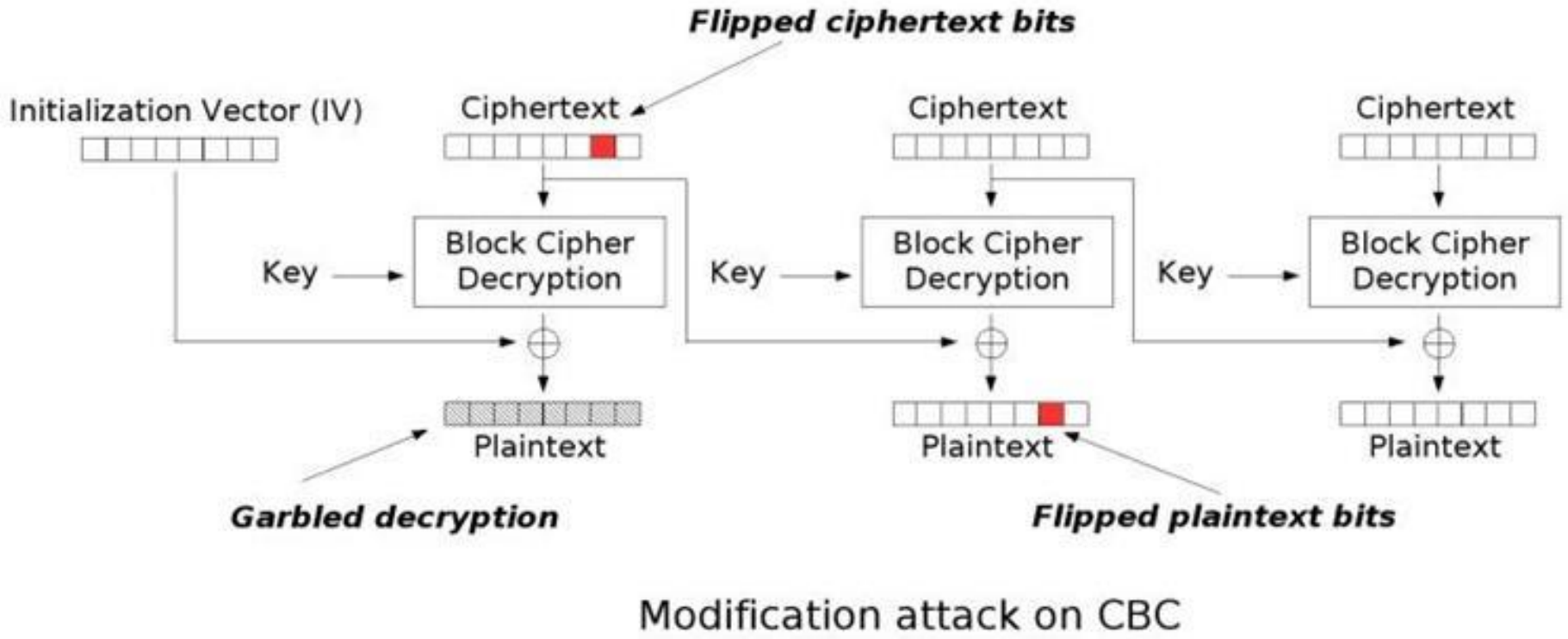
cryptopals - ten sam nonce

cryptopals - ten sam nonce, podejście statystyczne

Demo:

picoCTF - eleCTRic - Points: 400

CBC byte flipping



Modification attack on CBC

[CBC byte flipping 101]

w tym linku są też zadanka!

CBC byte flipping

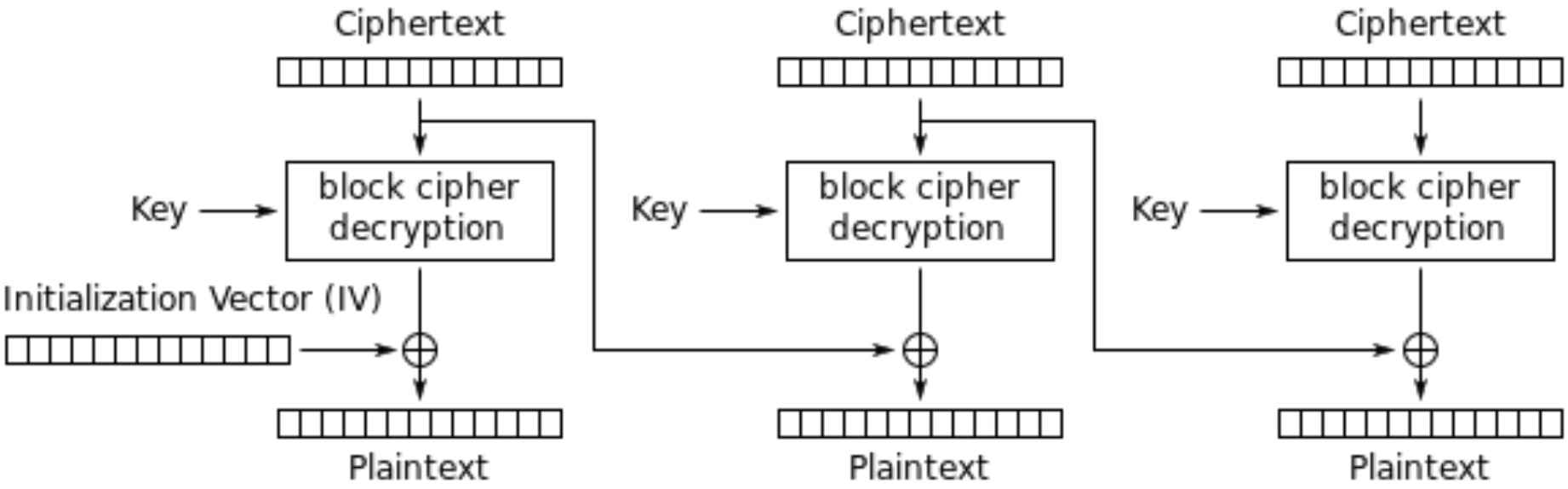
cryptopals.com/sets/2/challenges/16

prefix = "comment1=cooking%20MCs;userdata="

infix = “admin=[False/True]”

sufix = ";comment2=%20like%20a%20pound%20of%20bacon"

POODLE attack - SSLv3 killer



Cipher Block Chaining (CBC) mode decryption

POODLE na przykładzie

https://l4w.io/2015/01/tetcon-ctf-2015-crypto200-the-poodle-attack/

```
    def get(self):
        p1 = str(self.request.get('p1').strip())
        p2 = str(self.request.get('p2').strip())
        if p1 and p2:
            self.response.clear()
            self.response.set_status(200)
            self.response.out.write(Oracle(KEY).encrypt(p1 + FLAG + p2).encode('hex
        else:
            self.post()
```

```
    def post(self):
        c = self.request.get('c').strip().decode('hex')
        po = Oracle(KEY)
        try:
            m = po.decrypt(c)
            if FLAG in m:
                self.response.clear()
                self.response.set_status(200)
                return
            else:
                self.error(404)
                return
        except:
            self.error(403)
            return
```

```python
def random_pad(data, bs=16):
    last = bs - (len(data) % bs) - 1
    data += os.urandom(last)
    data += chr(last)
    return data
def random_unpad(data, bs=16):
    pad_length = ord(data[-1]) + 1
    return data[0:-pad_length]
...
    def encrypt(self, msg):
        iv = random_block(self.bs)
        aes = AES.new(self.key, 2, iv) # MODE_CBC
        mac = hmac.new(self.key)
        mac.update(iv)
        mac.update(msg)
        p = random_pad(msg + mac.digest())
        return iv + aes.encrypt(p)

    def decrypt(self, ciphertext):
        iv = ciphertext[0:self.bs]
        ciphertext = ciphertext[self.bs:]
        aes = AES.new(self.key, 2, iv) # MODE_CBC
        mac = hmac.new(self.key)
        p = random_unpad(aes.decrypt(ciphertext))
        msg = p[:-16]
        digest = p[-16:]
        mac.update(iv)
        mac.update(msg)
        if mac.digest() != digest:
            raise Exception
        return msg
```

Modyfikując p1 i p2 jesteśmy w stanie wyznaczyć długość flagi.

len(flag) == 32

Wyślijmy p1 = 16*’A’ i

p2 = 16*’A’.

Co dostajemy?

```
0acbad9b05fbc03551afdf147446a49b -> IV
88754146ca526818f5d61a14b4b3fc5d -> AAAAAAAAAAAAAAAA (ENCRYPTED)
73e12492fbf42c7d326d21a5329dc673 -> FLAG (ENCRYPTED)
ddae3c3ca20d8509dc899f3e3a50a43d -> FLAG (ENCRYPTED)
2983f99e0b6b0e28ee3270f42b9ce770 -> AAAAAAAAAAAAAAAA (ENCRYPTED)
c74bf206eb25e67371b2dab91af52fc3 -> HMAC (ENCRYPTED)
4fc8f00f557a19299e203e49618e3a90 -> PADDING BLOCK (random bytes + last byte specifies padding size)
```

Ostatni bajt wyznacza długość paddingu, i tyle bajtów od końca jest obcinanych z plaintextu.

Co się stanie jeśli sprytnie zamienimy kolejność bloków?

```
0acbad9b05fbc03551afdf147446a49b -> IV
88754146ca526818f5d61a14b4b3fc5d -> AAAAAAAAAAAAAAAA (ENCRYPTED)
73e12492fbf42c7d326d21a5329dc673 -> FLAG (ENCRYPTED)
ddae3c3ca20d8509dc899f3e3a50a43d -> FLAG (ENCRYPTED)
2983f99e0b6b0e28ee3270f42b9ce770 -> AAAAAAAAAAAAAAAA (ENCRYPTED)
c74bf206eb25e67371b2dab91af52fc3 -> HMAC (ENCRYPTED)
ffffffffffffffffffffffffffffffXX -> APPEND THIS BLOCK
88754146ca526818f5d61a14b4b3fc5d -> AAAAAAAAAAAAAAAA (ENCRYPTED)
```

88754146ca526818f5d61a14b4b3fc5d zostanie zinterpretowany przez serwer jako blok będący paddingiem.

Zmieniając bajt XX w przedziale [0,255] serwer w końcu odpowie HTTP 200.

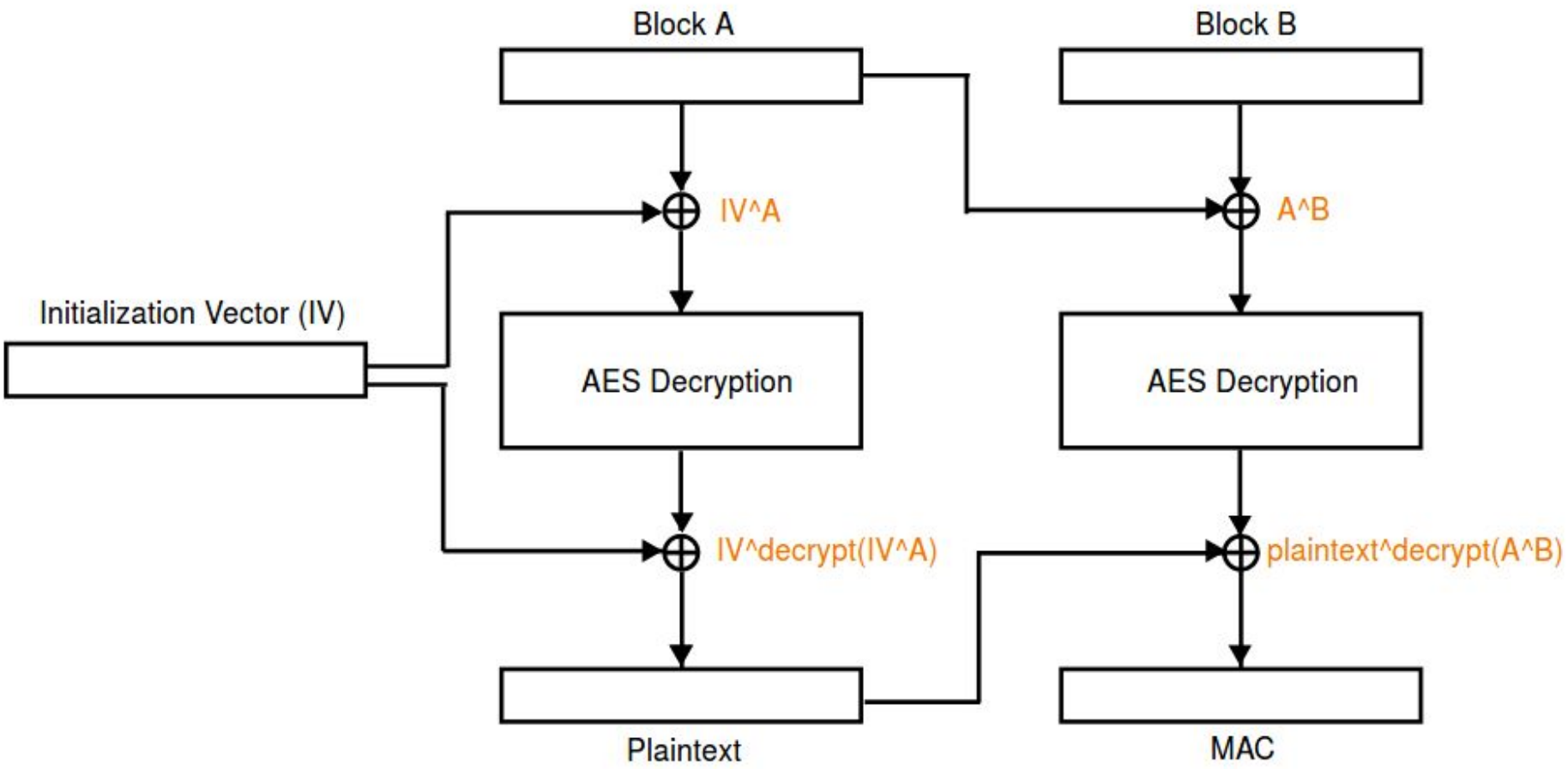
Serwer odpowie 200 jeśli odkoduje ostatni bajt paddingu do 31 i pominie blok ffffffffffffffffffffffffffffXX.

Więc ostatni bajt odszyfrowanej wiadomości to będzie 31 $\oplus$ XX, czyli ‘A’.

Wystarczy teraz zamiast bloku 16*’A’ podać na koniec blok z flagą.

Długość flagi nie musi być wielokrotnością dł. bloku bo możemy ustawiać prefiks i sufiks.

Teaser Dragon CTF 2018 - AES-128-TSB



plaintext = IV^decrypt(IV^A)

MAC = plaintext^decrypt(A^B) = IV^decrypt(IV^A)^decrypt(A^B)

Jaki feedback dostajemy od serwera?

Jak ominąć sprawdzenie MAC?

Czy padding jest dobrze zrobiony?

Czy można kontrolować jego długość? Jeśli tak to jak?

Czy możemy znaleźć pt0 dla dowolnego ct0?

Jak już dostaniemy zaszyfrowaną flagę to jak ją odszyfrować?

https://balsn.tw/ctf_writeup/20180929-teaserdragonctf/#aes-128-tsb

Think before you code

“When you are trying to solve a Crypto Challenge for a CTF, first of all, you need to detect which Cipher is used.

For example if it’s a Symmetric or Asymmetric cipher, a Classic cipher or if it’s just an Hash.”

3524047076446693726637644444779273976469522843493998667864643747675644 72 ?

Think before you code

“When you are trying to solve a Crypto Challenge for a CTF, first of all, you need to detect which Cipher is used.

For example if it’s a Symmetric or Asymmetric cipher, a Classic cipher or if it’s just an Hash.”

3524047076446693724767564472 ?

Zadanka

- 2018game.picoctf.com
- hack.cert.pl
- cryptopals.com

Źródła

- https://www.insomniasec.com/downloads/publications/Not%20So%20Random%20-%20Exploiting%20Unsafe%20Random%20Number%20Generator%20Use.pdf
- https://blog.skullsecurity.org/2012/everything-you-need-to-know-about-hash-length-extension-attacks
- https://robertheaton.com/2013/07/29/padding-oracle-attack/
- https://resources.infosecinstitute.com/cbc-byte-flipping-attack-101-approach/#gref

- http://pequalsnp-team.github.io/cheatsheet/crypto-101
- https://cryptopals.com/
- https://crypto.stackexchange.com/questions/40800/is-the-padding-oracle-attack-deterministic

- https://l4w.io/2015/01/tetcon-ctf-2015-crypto200-the-poodle-attack//