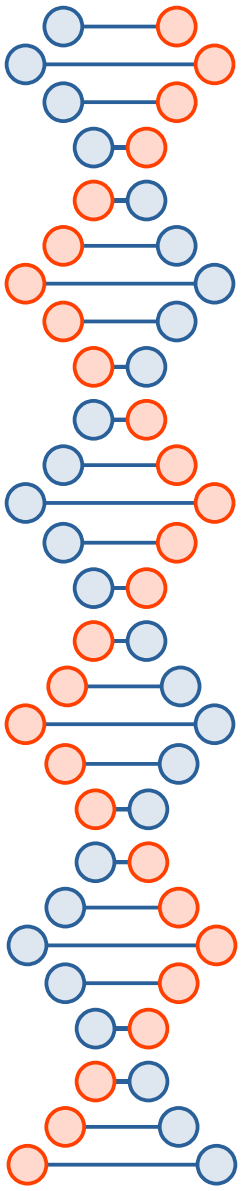


AES and cipherblock chaining modes

CSE 468 Fall 2026

jedimaestro@asu.edu

Finite fields...

https://en.wikipedia.org/wiki/%C3%89variste_Galois

https://en.wikipedia.org/wiki/Quadratic_equation

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

https://en.wikipedia.org/wiki/Cubic_equation

$$\frac{a}{q}x^2 + \frac{bq + ap}{q^2}x + \frac{cq^2 + bpq + ap^2}{q^3}$$

What is a field?

- “In mathematics, a field is a set on which addition, subtraction, multiplication, and division are defined and behave as the corresponding operations on rational and real numbers do.”

  --Wikipedia

- In cryptography, we often want to “undo things” or get the same result two different ways
  - Zmap will also use this trick
- On digital computers the math you learned in grade school is not good enough
  - Suppose we want to multiply by a plaintext, and the plaintext is 3. Great!
  - Now the decryption needs the inverse operation. Crap!
  - 1/3 is not easy to deal with (not even in floating point or fixed point)

Field

- Commutative
  - $a + b = b + a$
  - $a * b = b * a$
- Associative
  - $(a + b) + c = a + (b + c)$
  - $(a * b) * c = a * (b * c)$
- Identity
  - $0 \neq 1,\ a + 0 = a,\ a * 1 = a$
- Inverse
  - $a + -a = 0$
  - $a * a^{-1} = 1$
- Distributive
  - $a * (b + c) = (a * b) + (a * c)$

Arithmetic modulo a prime is a finite field

$$6 + 4 = 3 \pmod 7$$
$$3 - 6 = 4 \pmod 7$$
$$5 * 2 = 3 \pmod 7$$
$$5 * 3 = 1 \pmod 7$$
$$3 * 5^{-1} = 3 * 3 = 2 \pmod 7$$

This is called GF(7)

GF(2)

$0 + 0 = 0 \pmod 2$
$0 + 1 = 1 \pmod 2$
$1 + 0 = 1 \pmod 2$
$1 + 1 = 0 \pmod 2$

How to subtract?
Where have you seen this before?

GF(2)

$0 * 0 = 0 \pmod 2$
$0 * 1 = 0 \pmod 2$
$1 * 0 = 0 \pmod 2$
$1 * 1 = 1 \pmod 2$

Where have you seen this before?

GF(2)

- $K + K = 0$
- $(P + K) + K = P$
- $(A + K) + (B + K) = A + B$
- $0 + K = K$

XOR

- $K \oplus K = 0$
- $(P \oplus K) \oplus K = P$
- $(A \oplus K) \oplus (B \oplus K) = A \oplus B$
- $0 \oplus K = K$

How to use GF(2) to achieve what we want?

- Want to define a field over 2^k possibilities for a k-bit number
- 2 is prime, all other powers of 2 are not
  - Need to use irreducible polynomials

https://jedcrandall.github.io/courses/cse548spring2024/miniaesspec.pdf

Published in Cryptologia, XXVI (4), 2002.

Mini Advanced Encryption Standard (Mini-AES): A Testbed for Cryptanalysis Students

Raphael Chung-Wei **Phan**

2.1 The Finite Field GF(2^4)

The nibbles of Mini-AES can be thought of as elements in the finite field GF(2^4). Finite fields have the special property that operations (+,–, × and ÷) on the field elements always cause the result to be also in the field. Consider a nibble $n = (n_3, n_2, n_1, n_0)$ where $n_i \in \{0,1\}$. Then, this nibble can be represented as a polynomial with binary coefficients i.e having values in the set {0,1}:

$$n = n_3 x^3 + n_2 x^2 + n_1 x + n_0$$

Example 1

Given a nibble, n = 1011, then this can be represented as

$$n = 1 x^3 + 0 x^2 + 1 x + 1 = x^3 + x + 1$$

Note that when an element of GF(2^4) is represented in polynomial form, the resulting polynomial would have a degree of at most 3.

2.2 Addition in GF(2^4)

When we represent elements of GF(2^4) as polynomials with coefficients in {0,1}, then addition of two such elements is simply addition of the coefficients of the two polynomials. Since the coefficients have values in {0,1}, then the addition of the coefficients is just modulo 2 addition or exclusive-OR denoted by the symbol ⊕. Hence, for the rest of this paper, the symbols + and ⊕ are used interchangeably to denote addition of two elements in GF(2^4).

Example 2

Given two nibbles, n = 1011 and m = 0111, then the sum, n + m = 1011 + 0111 = 1100 or in polynomial notation:

$$n + m = (x^3 + x + 1) + (x^2 + x + 1) = x^3 + x^2$$

2.3 Multiplication in GF(2^4)

Multiplication of two elements of GF(2^4) can be done by simply multiplying the two polynomials. However, the product would be a polynomial with a degree possibly higher than 3.

Example 3

Given two nibbles, n = 1011 and m = 0111, then the product is:

$$(x^3 + x + 1)(x^2 + x + 1) = x^5 + x^4 + x^3 + x^3 + x^2 + x + x^2 + x + 1$$
$$= x^5 + x^4 + 1$$

In order to ensure that the result of the multiplication is still within the field GF(2^4), it must be reduced by division with an irreducible polynomial of degree 4, the remainder of which will be taken as the final result. An irreducible polynomial is analogous to a prime number in arithmetic, and as such a polynomial is irreducible if it has no divisors other than 1 and itself. There are many such irreducible polynomials, but for Mini-AES, it is chosen to be:

$$m(x) = x^4 + x + 1$$

Example 4

Given two nibbles, n = 1011 and m = 0111, then the final result after multiplication in GF(2^4), called the 'product of n × m modulo m(x)' and denoted as ⊗, is:

$$(x^3 + x + 1) \otimes (x^2 + x + 1) = x^5 + x^4 + 1 \text{ modulo } x^4 + x + 1$$
$$= x^2$$

This is because:

$$\begin{array}{rl} & x + 1 \qquad \text{(quotient)} \\ x^4 + x + 1 \;\overline{)}\; & x^5 + x^4 + 1 \\ & + \; x^5 + x^2 + x \\ \hline & \qquad x^4 + x^2 + x + 1 \\ & + \quad x^4 + \quad x + 1 \\ \hline & \qquad\quad x^2 \qquad \text{(remainder)} \end{array}$$

Note that since the coefficients of the polynomials are in {0,1}, then addition is simply exclusive-OR and hence subtraction is also exclusive-OR since exclusive-OR is its own inverse.

Example 4

Given two nibbles, n = 1011 and m = 0111, then the final result after multiplication in GF(2^4), called the 'product of n × m modulo m(x)' and denoted as ⊗, is:

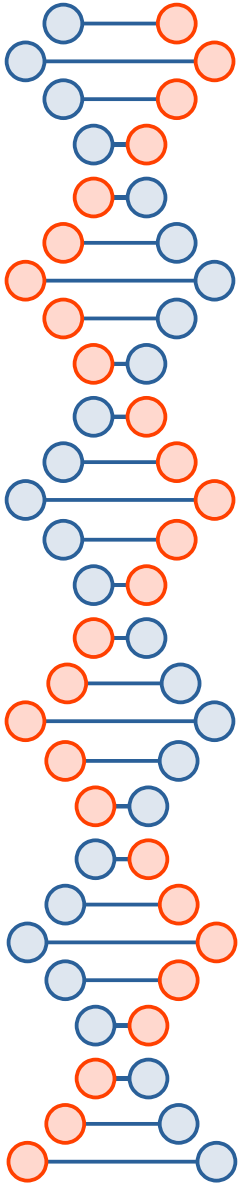
$$(x^3 + x + 1) \otimes (x^2 + x + 1) \quad = x^5 + x^4 + 1 \text{ modulo } x^4 + x + 1$$

This is because:

This is how to show your work on Exam 1

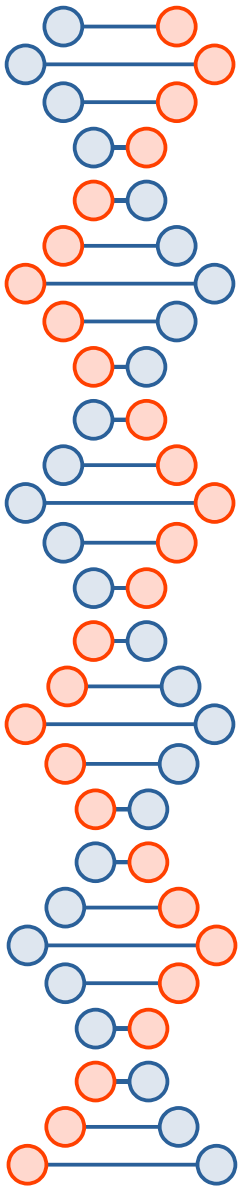
$$\begin{array}{r} x + 1 \qquad\qquad\qquad\qquad \text{(quotient)} \\ x^4 + x + 1 \,\overline{)\, x^5 + x^4 + 1} \\ \underline{+\ x^5 + x^2 + x} \\ x^4 + x^2 + x + 1 \\ \underline{+\quad x^4 + \quad x + 1} \\ x^2 \qquad\qquad\qquad\qquad \text{(remainder)} \end{array}$$

Note that since the coefficients of the polynomials are in {0,1}, then addition is simply exclusive-OR and hence subtraction is also exclusive-OR since exclusive-OR is its own inverse.



Why study AES?

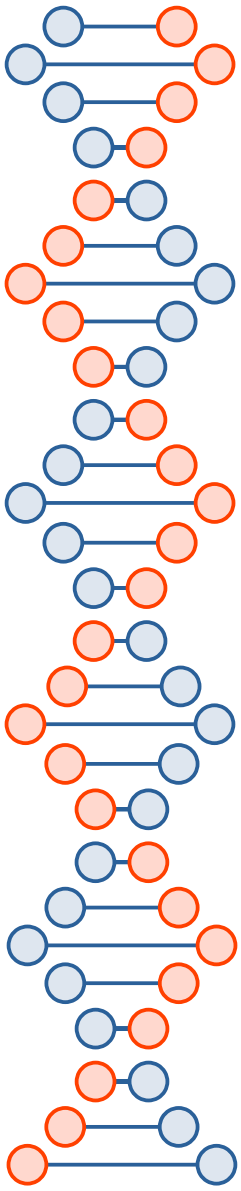
- It's a good demonstration of principles we've been talking about (*e.g.*, confusion and diffusion)
- It's the workhorse of the majority of network crypto, *e.g.*, many SSH and TLS connections
- It's easy to see that AES is just substitution, permutation, and XOR



Substitution

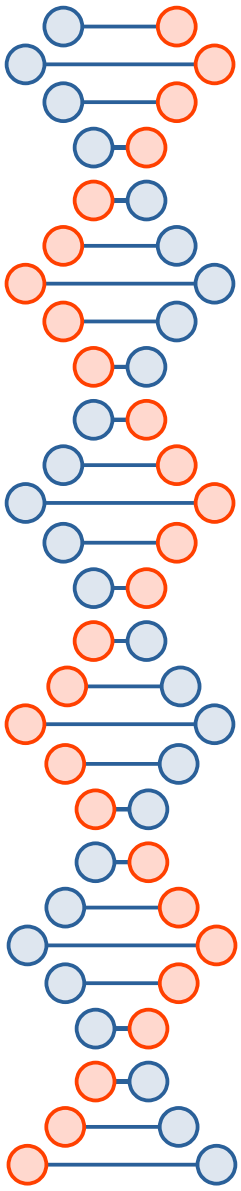
HELLO WORLD

TNWWX DXPWE



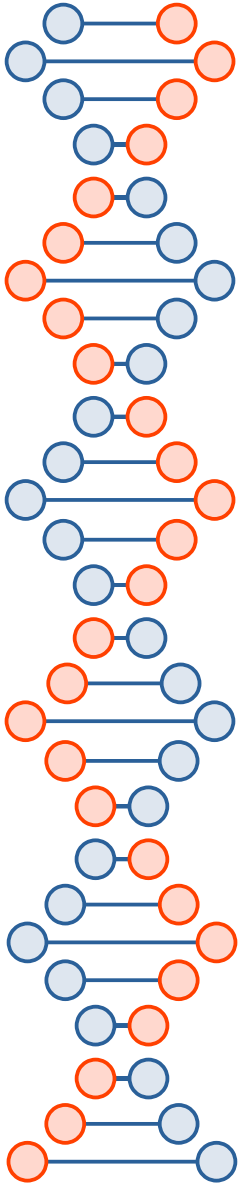
Permutation

ABCD	ABDC	ACBD	ACDB	ADBC	ADCB
BACD	BADC	BCAD	BCDA	BDAC	BDCA
CABD	CADB	CBAD	CBDA	CDAB	CDBA
DABC	DACB	DBAC	DBCA	DCAB	DCBA



Bitwise XOR

$$
\begin{aligned}
&00101010_{b} \\
\oplus\ &10000110_{b} \\
=\ &10101100_{b}
\end{aligned}
$$



Symmetric encryption over time...

- Handwritten notes, *etc.* for centuries
  - Typically the algorithm was secret
- 1883 … Kerckhoff's rules
  - Now we know the key should be the only secret
- 1975 … DES
  - Efficient in hardware, not in software
- 2001 … AES
  - Efficient in software, and lots of different kinds of hardware

AES S-box requirements

- Can’t pull it out of our &%# like the NSA did for DES
- Should have good nonlinear properties
  - Better nonlinearity means fewer rounds
- Should be reversible
  - Don’t want to use a Feistel structure for performance reasons

An alternative to AES: Tiny Encryption Algorithm (TEA), Feistel structure with 32 rounds

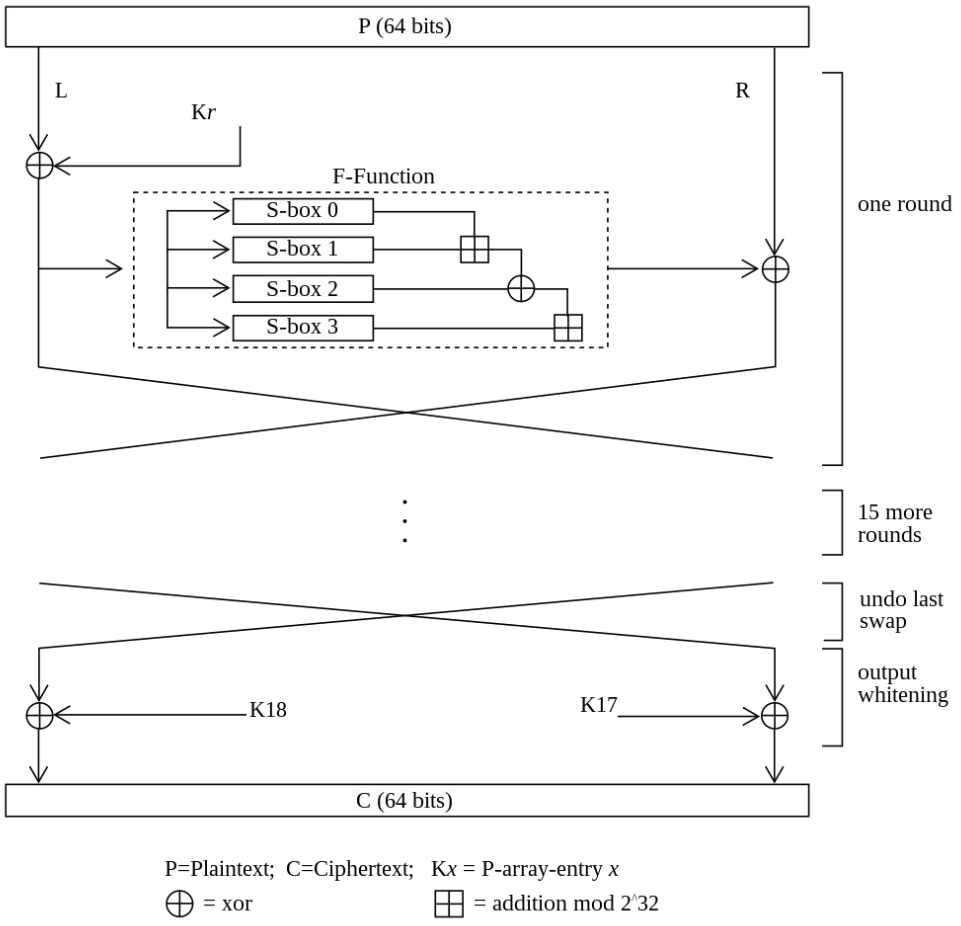
```c
#include <stdint.h>

void encrypt (uint32_t v[2], const uint32_t k[4]) {
    uint32_t v0=v[0], v1=v[1], sum=0, i;           /* set up */
    uint32_t delta=0x9E3779B9;                     /* a key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];   /* cache key */
    for (i=0; i<32; i++) {                         /* basic cycle start */
        sum += delta;
        v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
    }                                              /* end cycle */
    v[0]=v0; v[1]=v1;
}

void decrypt (uint32_t v[2], const uint32_t k[4]) {
    uint32_t v0=v[0], v1=v[1], sum=0xC6EF3720, i;  /* set up; sum is (delta << 5) & 0xFFFFFFFF */
    uint32_t delta=0x9E3779B9;                     /* a key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];   /* cache key */
    for (i=0; i<32; i++) {                         /* basic cycle start */
        v1 -= ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
        v0 -= ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        sum -= delta;
    }                                              /* end cycle */
    v[0]=v0; v[1]=v1;
}
```

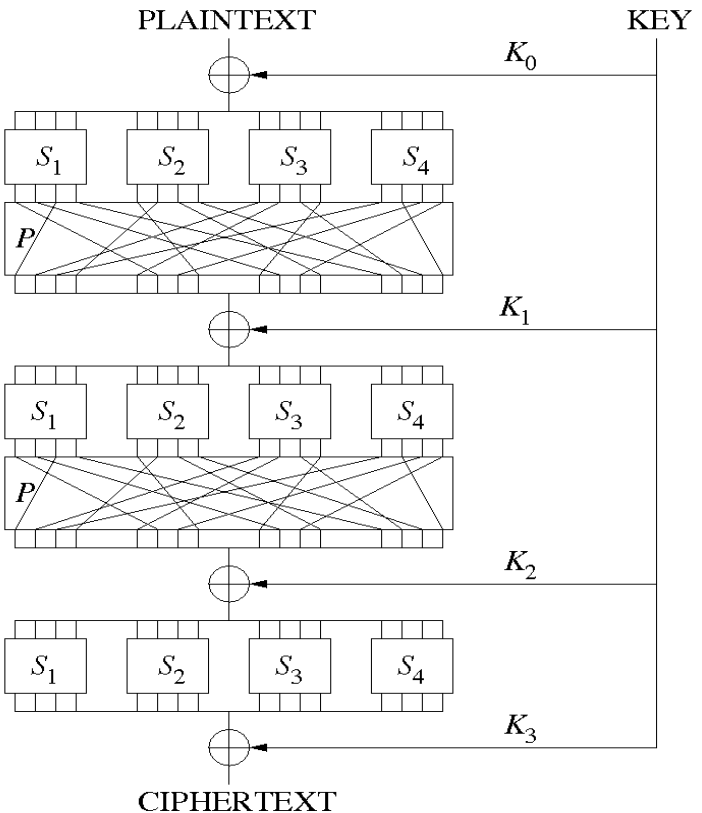
Another alternative to AES: Blowfish (Twofish was in the AES competition)

https://en.wikipedia.org/wiki/Blowfish_(cipher)



Substitution Permutation Network

e.g., AES 128-bit blocks, (128-, 192-, 256-)bit key, (10, 12, 14) rounds



DEMONSTRATION OF THE AES S-BOX: NON-LINEARITY AND CONFUSION

Part 1: The Rijndael S-Box Array

16x16 Substitution Look-up Table (Hexadecimal)

Row Index:	Column Index: 0	1	2	...	4	5	7	8	9	A	...	C	...	E	F
0	00	01	02	03	04	05	07	00	00	0A	BB	CC	DD	EE	FF
1	01	15	19	15	26	EB	FF	1F	1F	A6	8F	EA	B8	FF	FF
2	00	03	36	25	35	G9	FA	04	B4	3C	88	CF	B3	FB	FB
3	00	06	AC	0D	06	82	F8	02	92	5C	89	DD	EB	B8	EF
4	00	08	23	0F	0C	99	99	1C	97	E6	08	DD	DC	BB	FB
5	00	96	31	33	99	3	1F	15	99	EB	B9	A7	AG	E6	P6
6	00	02	15	0C	05	85	0F	08	99	5D	DF	1F	EF	EF	FF
7	00	01	0F	0D	04	82	0F	08	89	0A	B7	C1	B8	FF	FF
8	00	03	0C	0D	05	5B	0A	00	F8	F6	07	CB	04	0B	BB
9	00	08	2B	0E	05	8E	FB	01	F2	6D	03	F7	EB	FE	FE
A	00	05	01	0C	06	87	07	03	B5	BC	D6	DD	DC	DD	A7
B	00	06	04	0B	06	88	0F	07	05	C6	D7	A7	BB	BE	A7
C	99	98	99	06	08	85	99	57			DD	A7	A7	BB	A7
D	00	01	02	03	04	95	05	0A	B8	DE	BD	AB	C5	DG	BF
E	00	01	02	04	04	8F	FB	0F	F3	EB	EB	FC	ED	EE	EF
F	00	01	02	03	04	95	07	00	BD	FE	CB	FC	FD	FF	FF

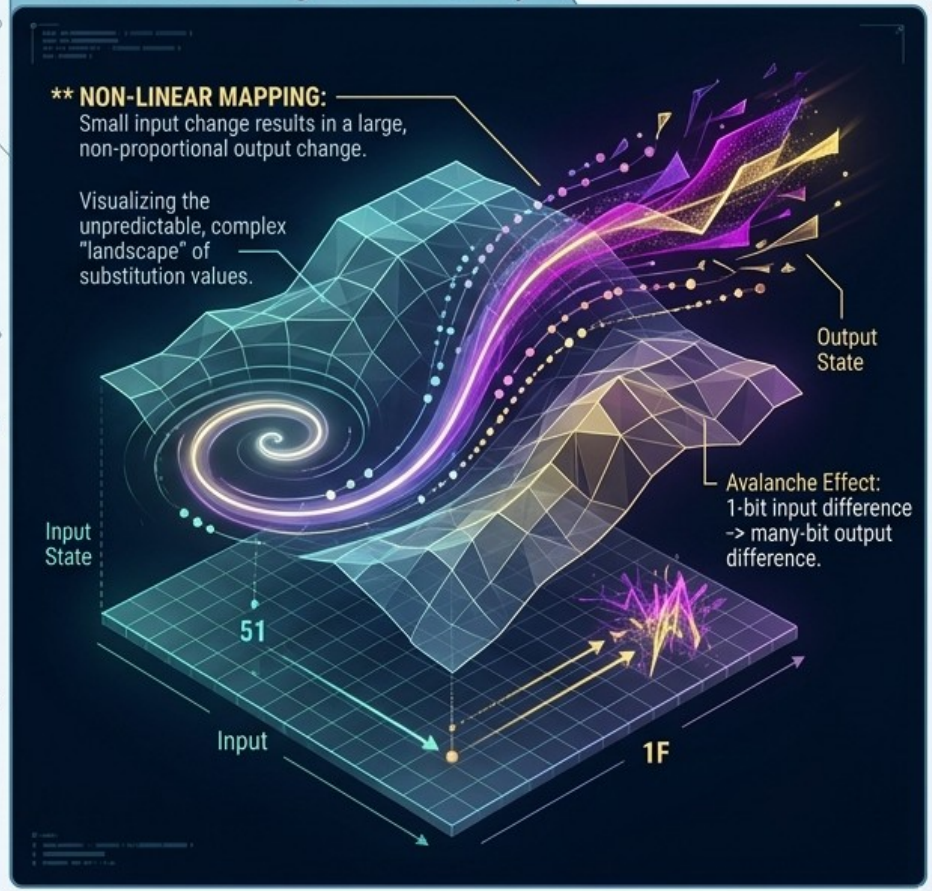
Input

51

Input (Row: C, Col: 9) -> Output: DD (A7)

Conceptual Flow

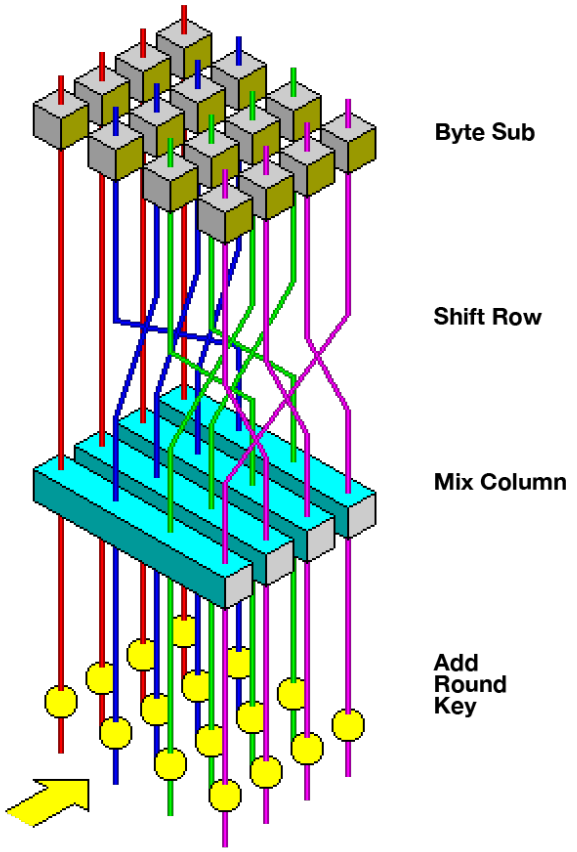
Part 2: Visualizing Non-Linearity



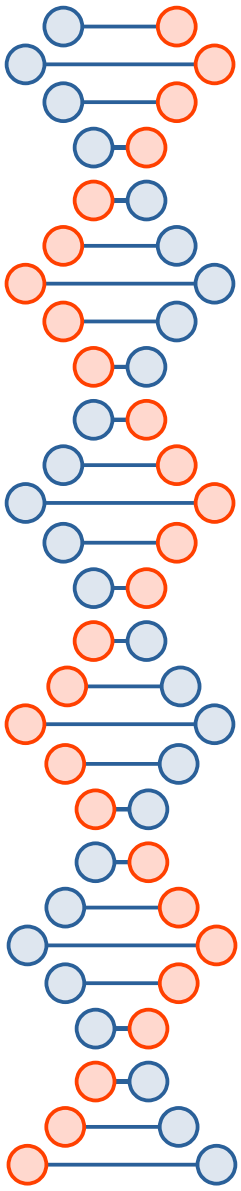
By introducing high non-linearity, the S-Box is the core mechanism that destroys statistical patterns, making cryptanalysis impossible.

AES

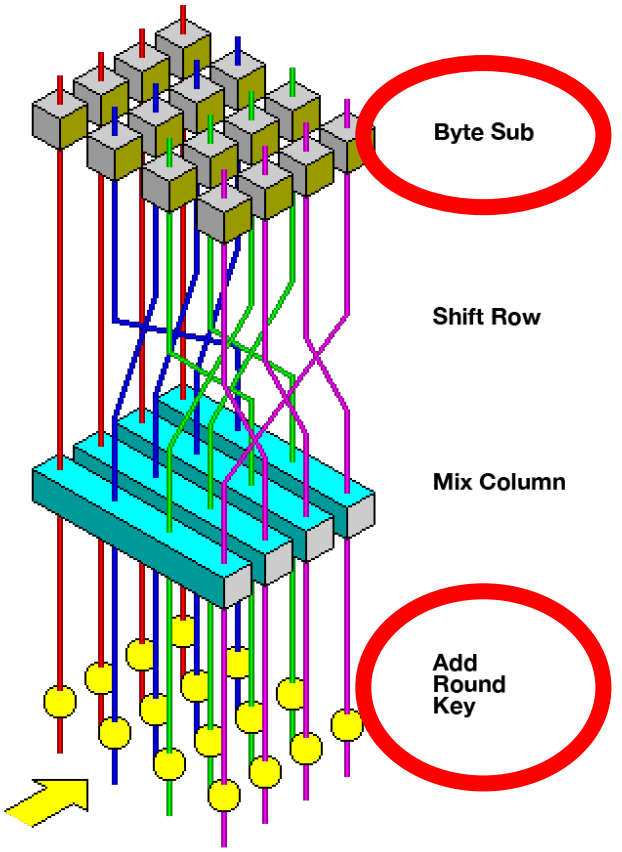
- Rijndael
  - Joan Daemen and Vincent Rijmen
- Very clever S-box design that comes from Kaisa Nyberg
  - Based on finite fields (*a.k.a.* Gallois fields)



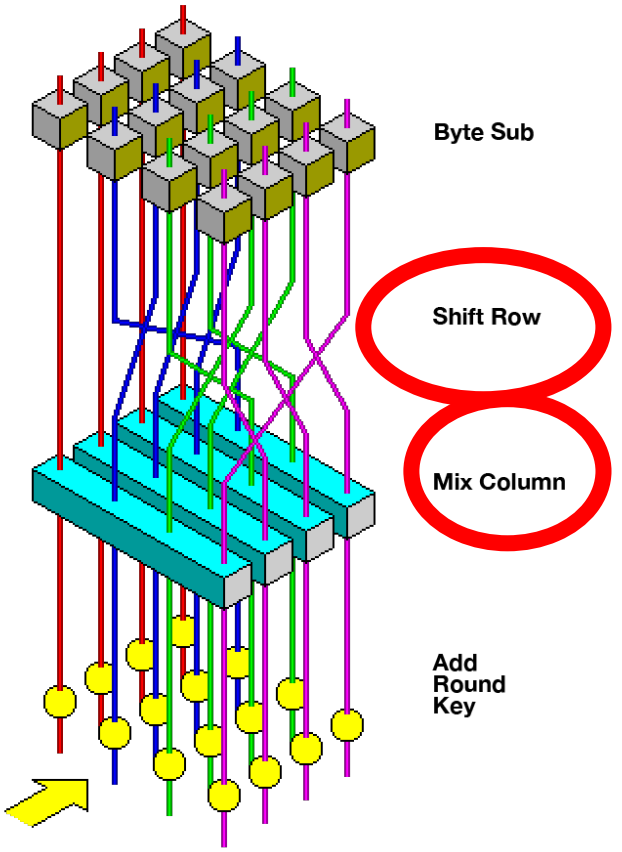
https://en.wikipedia.org/wiki/Kaisa_Nyberg



Confusion



Diffusion



The MixColumns operation also uses Galois fields (but is a linear function)...

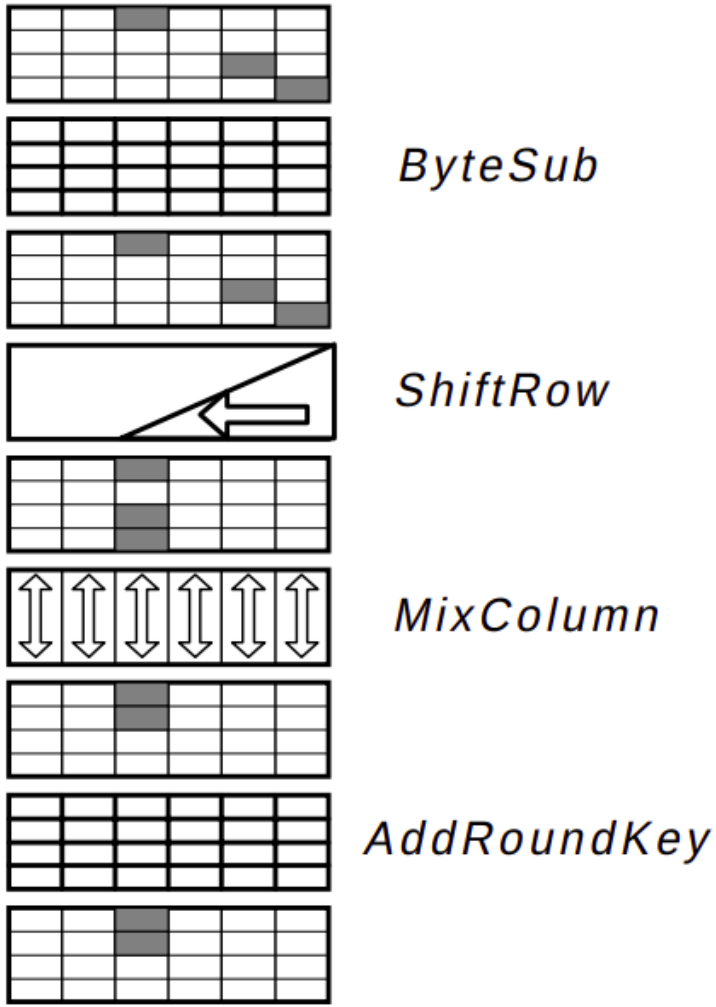
```
1  private byte GMul(byte a, byte b) { // Galois Field (256) Multiplication of two Bytes
2      byte p = 0;
3
4      for (int counter = 0; counter < 8; counter++) {
5          if ((b & 1) != 0) {
6              p ^= a;
7          }
8
9          bool hi_bit_set = (a & 0x80) != 0;
10         a <<= 1;
11         if (hi_bit_set) {
12             a ^= 0x1B; /* x^8 + x^4 + x^3 + x + 1 */
13         }
14         b >>= 1;
15     }
16
17     return p;
18 }
19
20 private void MixColumns() { // 's' is the main State matrix, 'ss' is a temp matrix of the same dimensions
   as 's'.
21     Array.Clear(ss, 0, ss.Length);
22
23     for (int c = 0; c < 4; c++) {
24         ss[0, c] = (byte)(GMul(0x02, s[0, c]) ^ GMul(0x03, s[1, c]) ^ s[2, c] ^ s[3, c]);
25         ss[1, c] = (byte)(s[0, c] ^ GMul(0x02, s[1, c]) ^ GMul(0x03, s[2, c]) ^ s[3,c]);
26         ss[2, c] = (byte)(s[0, c] ^ s[1, c] ^ GMul(0x02, s[2, c]) ^ GMul(0x03, s[3, c]));
27         ss[3, c] = (byte)(GMul(0x03, s[0,c]) ^ s[1, c] ^ s[2, c] ^ GMul(0x02, s[3, c]));
28     }
29
30     ss.CopyTo(s, 0);
31 }
```

https://en.wikipedia.org/wiki/Rijndael_MixColumns

AES

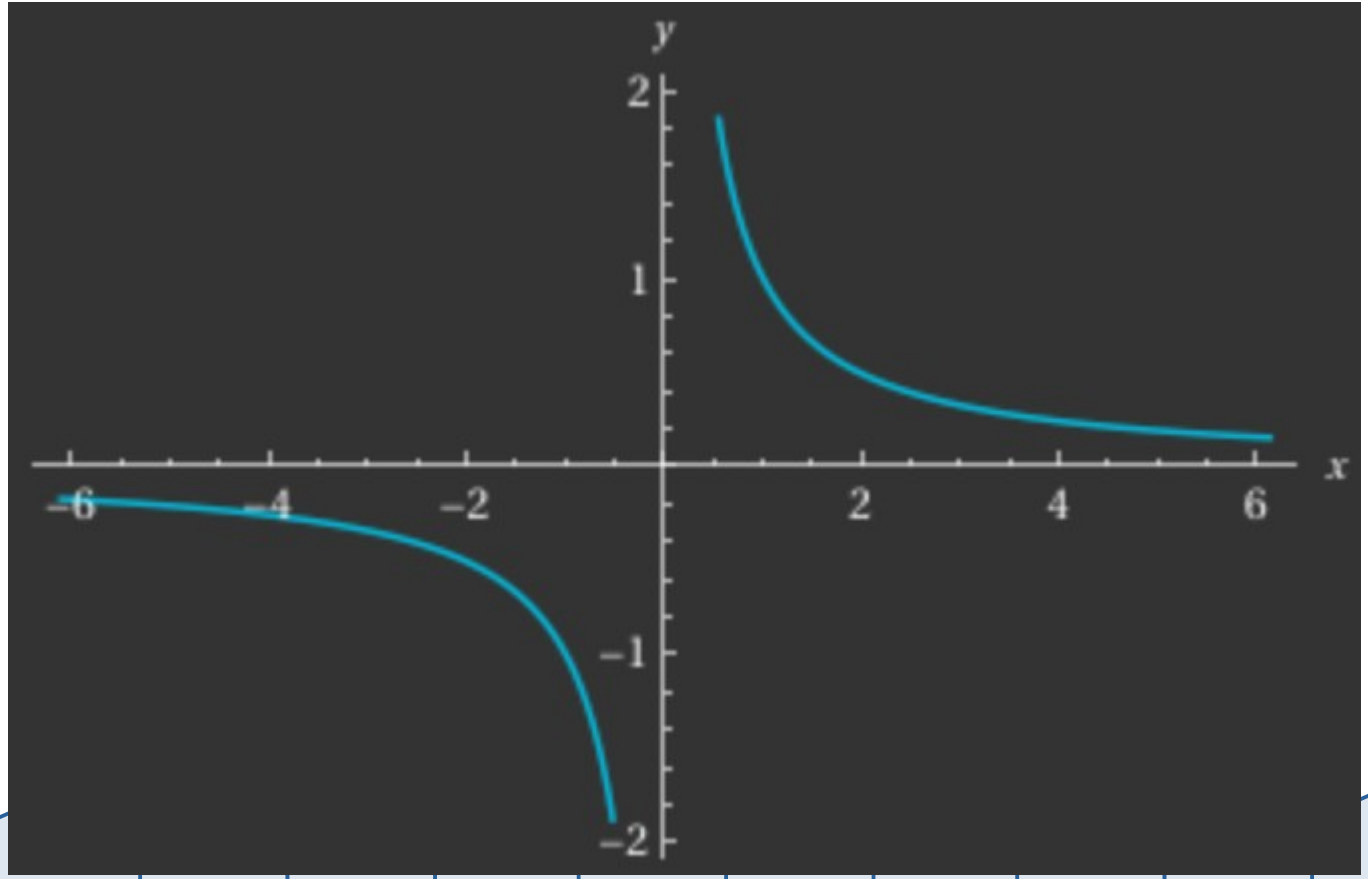
- 128-bit blocks, 128-, 192-, or 256-bit keys
  - 10, 12, or 14 rounds respectively
- No less secure than the other candidates, but better performance…
  - In hardware *and* software
    - Different word sizes (8, 16, 32, 64)
  - With or without specialized hardware support
    - *E.g.*, Gallois Fields on Blackfin DSPs
    - *E.g.*, AES special instruction set on Intel chips

https://www.cs.miami.edu/home/burt/learning/Csc688.012/rijndael/rijndael_doc_V2.pdf


ByteSub
ShiftRow
MixColumn
AddRoundKey

ByteSub gives non-linearity...

f(x) = 1/x



Affine transformation

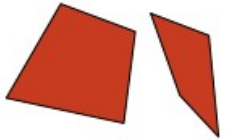
https://math.stackexchange.com/questions/2717306/math-background-for-approximating-a-projective-transformation-by-an-affine-trans

2D transformation hierarchy

A square transforms to:

Projective 8dof

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

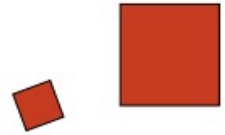


Affine 6dof

$$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

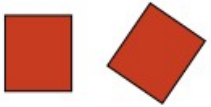
Similarity 4dof

$$\begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

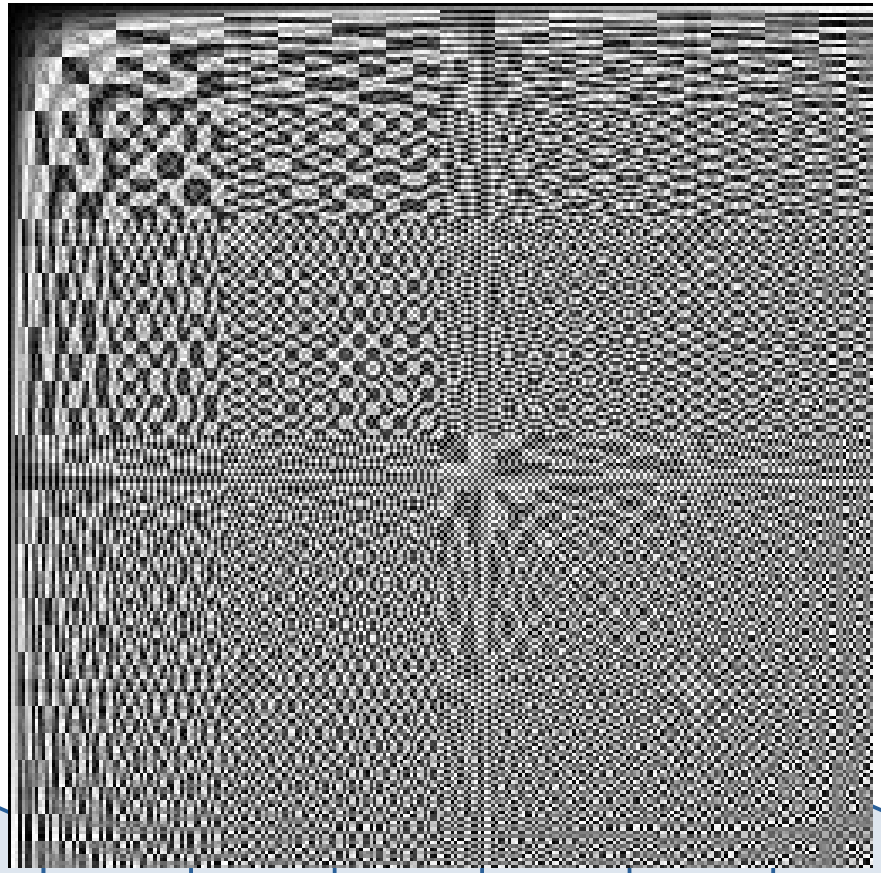


Euclidean 3dof

$$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

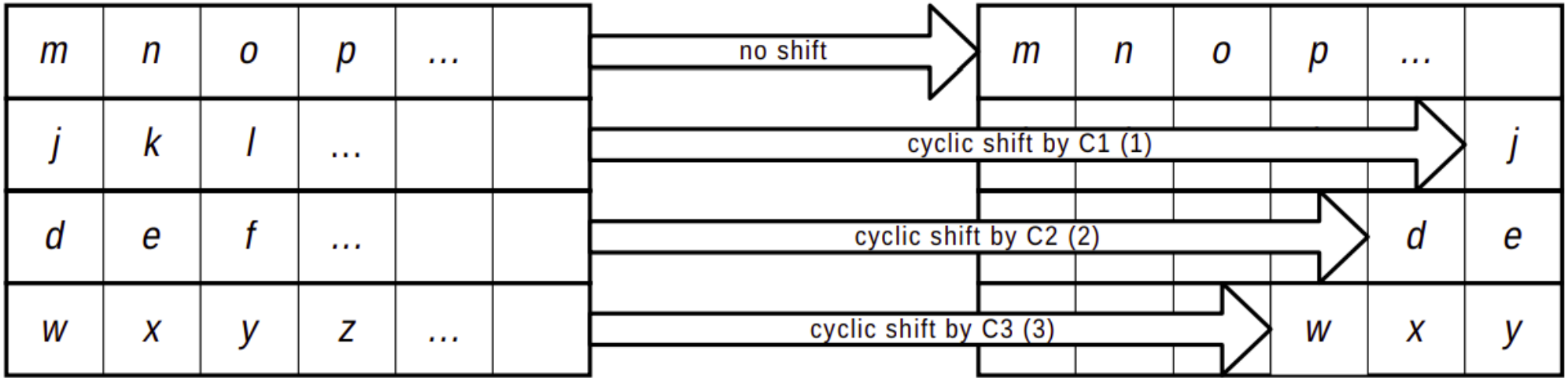


https://www.cs.uaf.edu/2015/spring/cs463/lecture/03_23_AES.html



ShiftRow and MixColumn give diffusion...

ShiftRow


m n o p ...
j k l ...
d e f ...
w x y z ...
no shift
cyclic shift by C1 (1)
cyclic shift by C2 (2)
cyclic shift by C3 (3)
m n o p ...
j
d e
w x y

MixColumn

4.2.3 The MixColumn transformation

In MixColumn, the columns of the State are considered as polynomials over GF(2^8) and multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x)$, given by

$$c(x) = \text{'03'}\ x^3 + \text{'01'}\ x^2 + \text{'01'}\ x + \text{'02'}\ .$$

This polynomial is coprime to $x^4 + 1$ and therefore invertible. As described in Section 2.2, this can be written as a matrix multiplication. Let b(x) = c(x) ⊗ a(x),

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

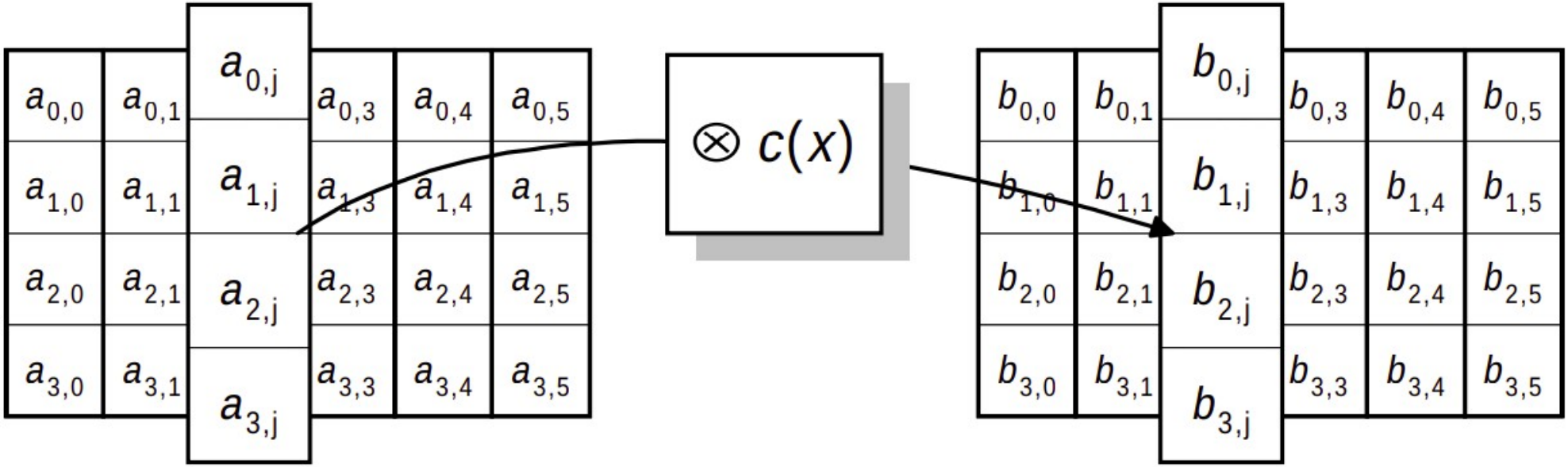


Figure 4: MixColumn operates on the columns of the State.

The inverse of MixColumn is similar to MixColumn. Every column is transformed by multiplying it with a specific multiplication polynomial $d(x)$, defined by

$$(\text{`03'}\ x^3 + \text{`01'}\ x^2 + \text{`01'}\ x + \text{`02'}) \otimes d(x) = \text{`01'}\ .$$

It is given by:

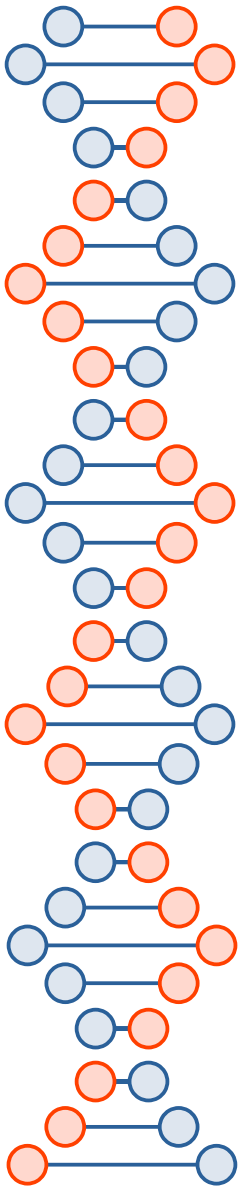
$$d(x) = \text{`0B'}\ x^3 + \text{`0D'}\ x^2 + \text{`09'}\ x + \text{`0E'}\ .$$

AddRoundKey is a simple XOR on purpose...

8.6 Weak keys as in IDEA

The weak keys discussed in this subsection are keys that result in a block cipher mapping with detectable weaknesses. The best known case of weak keys are those of IDEA [Da95]. Typically, this weakness occurs for ciphers in which the non-linear operations depends on the actual key value. This is not the case for Rijndael, where keys are applied using the EXOR and all non-linearity is in the fixed S-box. In Rijndael, there is no restriction on key selection.

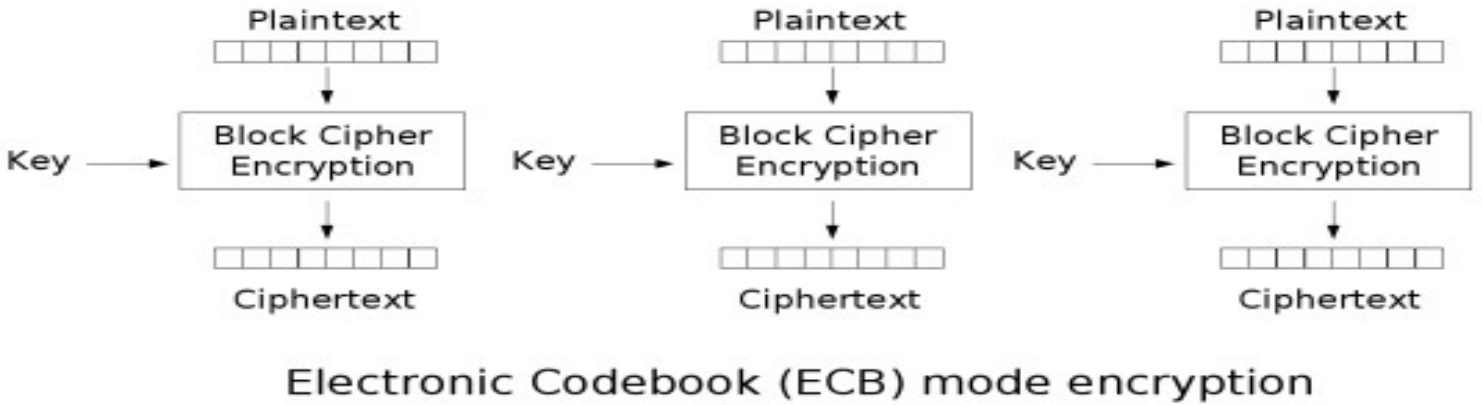
Okay to just encrypt block by block?



Cipher modes

- ECB, CBC discussed in the next slides
- Also Counter Mode, Galois Counter Mode, Cipher Feedback, Output Feedback, more...
  - Parallelization, message authentication, and other features
  - Can make stream ciphers out of block ciphers

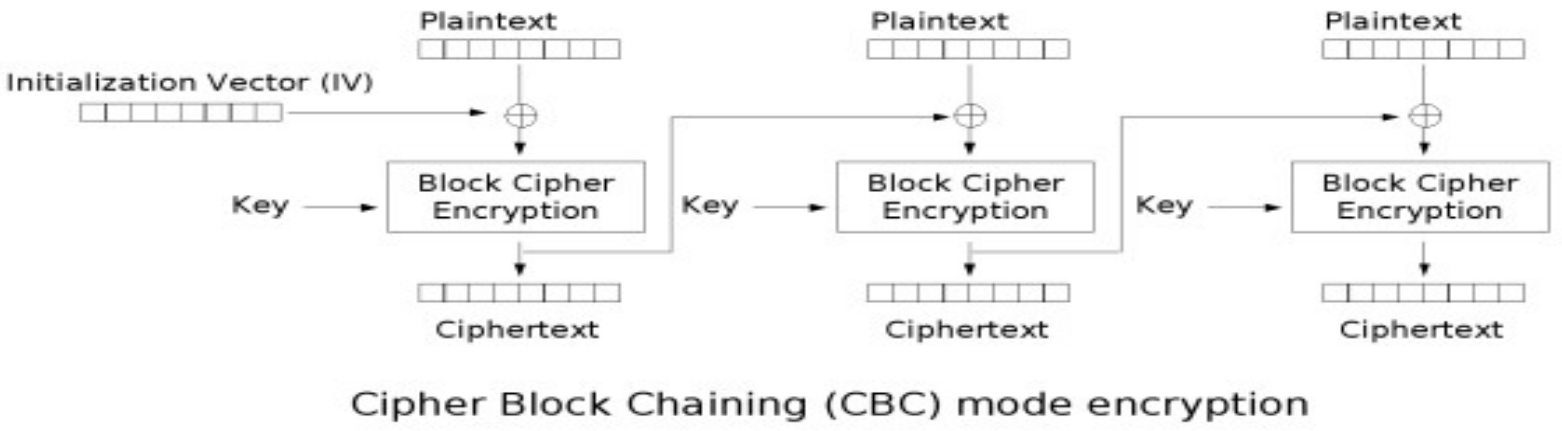
Electronic Codebook (ECB)



Electronic Codebook (ECB) mode encryption

Image stolen from Wikipedia

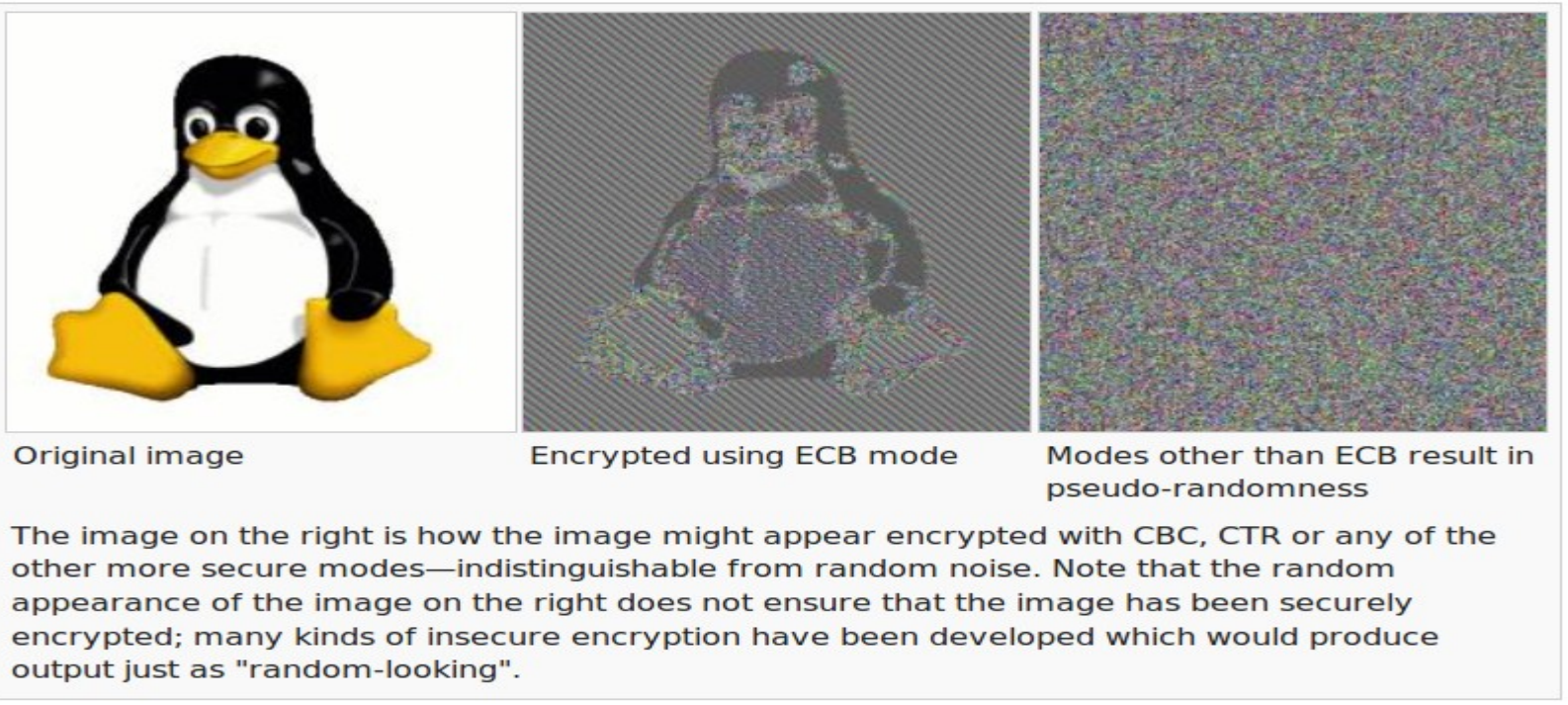
Cipher Block Chaining (CBC)



Cipher Block Chaining (CBC) mode encryption

Image stolen from Wikipedia

ECB is generally bad



Original image

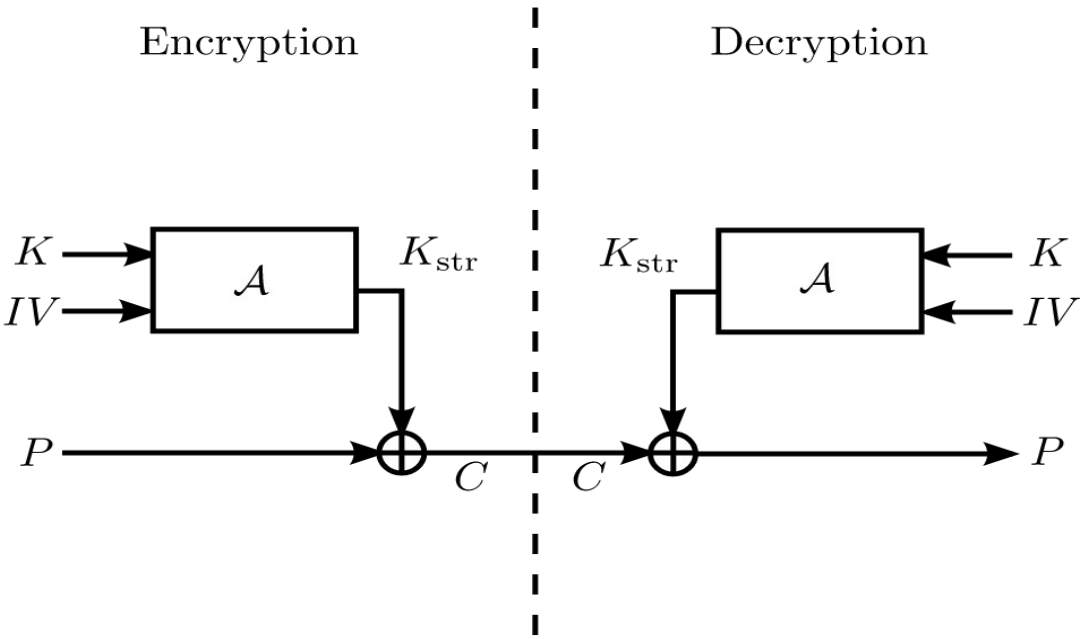
Encrypted using ECB mode

Modes other than ECB result in pseudo-randomness

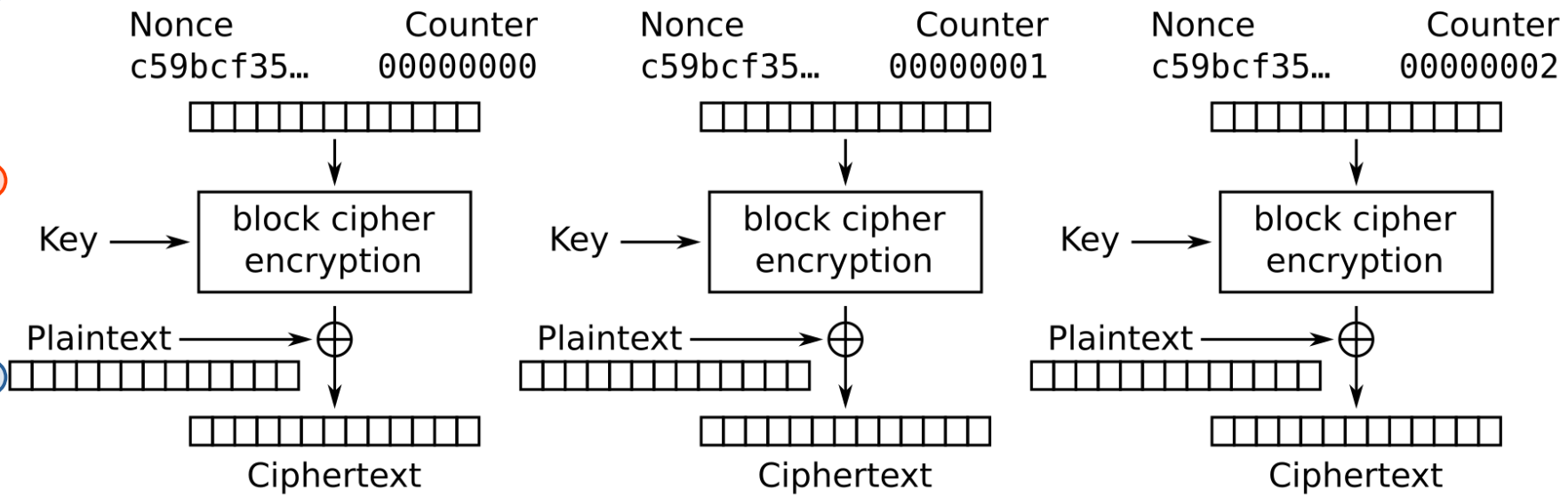
The image on the right is how the image might appear encrypted with CBC, CTR or any of the other more secure modes—indistinguishable from random noise. Note that the random appearance of the image on the right does not ensure that the image has been securely encrypted; many kinds of insecure encryption have been developed which would produce output just as "random-looking".

Image stolen from Wikipedia

Stream ciphers can be built out of block ciphers

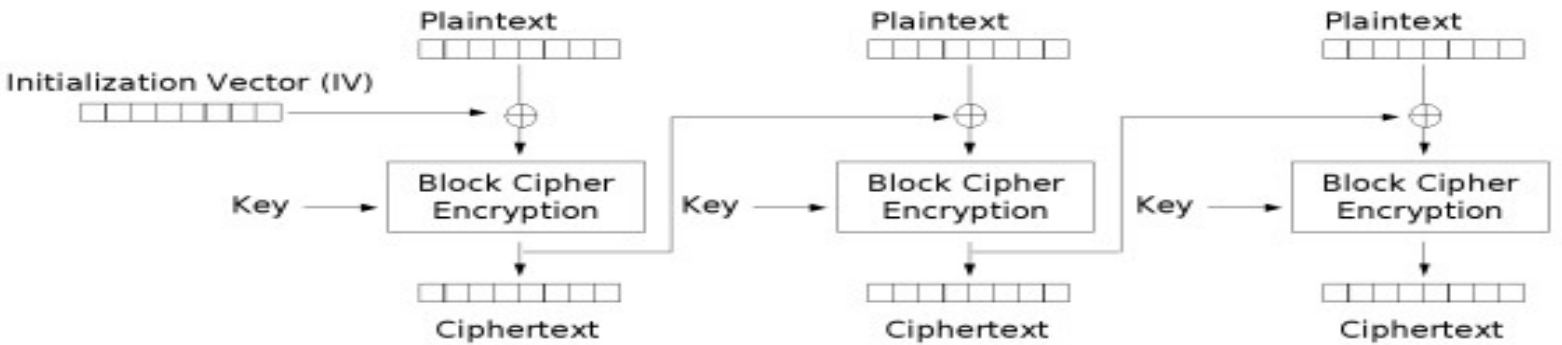


Stream ciphers can be built out of block ciphers



Counter (CTR) mode encryption

CBC padding oracle attacks...

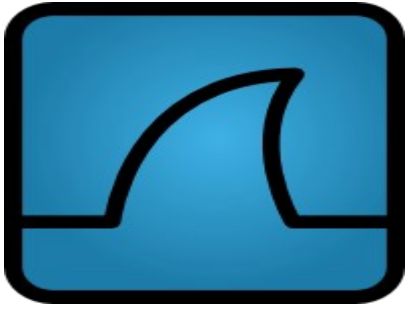


Cipher Block Chaining (CBC) mode encryption

CBC padding oracle attack examples

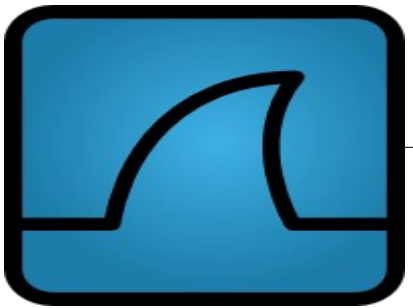
- Serge Vaudenay published the original attack in 2002
  - Applied to web frameworks like Ruby on Rails, ASP.NET, and JavaServer Faces
  - https://www.iacr.org/cryptodb/archive/2002/EUROCRYPT/2850/2850.pdf
- POODLE (published by Google in 2014) exploited SSLv3 that is still widely used by web servers and browsers
  - https://security.googleblog.com/2014/10/this-poodle-bites-exploiting-ssl-30.html

Alice and Bob have a shared secret key



Eve makes a copy of the ciphertext as it is transmitted from Alice to Bob.

Alice and Bob have a shared secret key



Eve re-plays modified copies of the encrypted message and learns information about the plaintext from Bob's behavior (*e.g.*, Bob throws an exception for padding error)

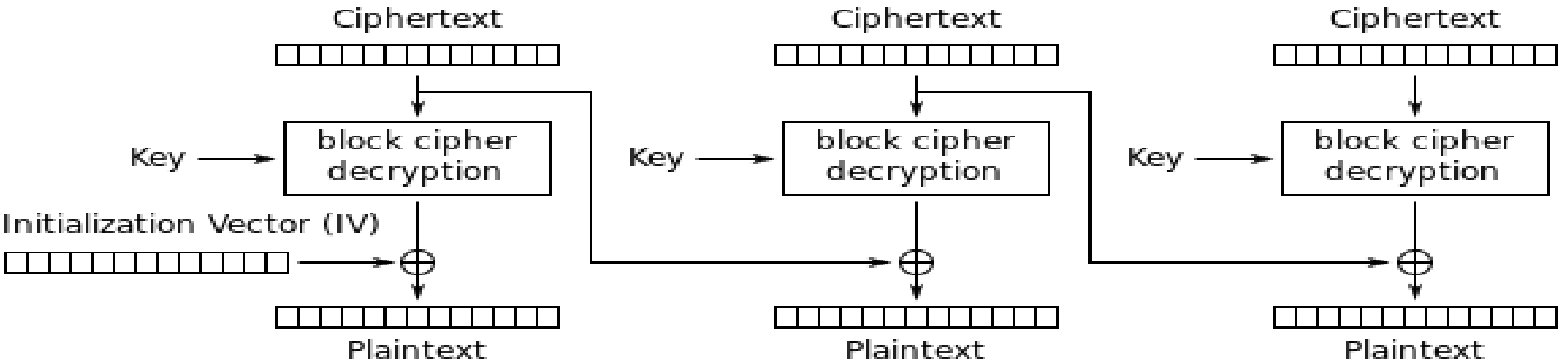
PKCS#7 padding

- AES always encrypts in 128-bit blocks
  - 128 bits == 16 bytes
- If you fill up blocks, that's great
  - But, the last block might not be full
- Need an “unambiguous” way to pad the last block so the decrypting party knows the padding to throw out
  - *E.g.*, PKCS#7 (PKCS == Public Key Cryptography Standards)

															01
														02	02
													03	03	03
												04	04	04	04
											05	05	05	05	05
										06	06	06	06	06	06
									07	07	07	07	07	07	07
								08	08	08	08	08	08	08	08
							09	09	09	09	09	09	09	09	09
						0A	0A	0A	0A	0A	0A	0A	0A	0A	0A
					0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B
				0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C
			0D	0D	0D	0D	0D	0D	0D	0D	0D	0D	0D	0D	0D
		0E	0E	0E	0E	0E	0E	0E	0E	0E	0E	0E	0E	0E	0E
	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10

When last block is decrypted

- Check last byte of the last block, that's the number of bytes of padding
  - Call it N
- There should be N N's on the end
  - If not, throw a padding error
  - If so, remove them, they're padding
    - Might remove the whole last block if N = 16 (or 10 in hex)



Cipher Block Chaining (CBC) mode decryption

Requirements for attack

- Ability to modify ciphertexts and replay them
  - Chosen ciphertext attack
- A padding oracle
  - *I.e.*, something that tells you whether the corresponding plaintext (for any ciphertext you send) has valid padding or not

Example plaintext (we don't know the plaintext yet before the attack)

H	e	l	l	o	20	W	o	r	l	d	!	\n	03	03	03

Example protocol for a client to send an encrypted message to a server

N	u	m	b	l	k	s	:	1	K	e	y	l	D	:	A3
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
98	CC	BE	01	FF	26	39	97	85	A1	02	1E	BC	A5	7E	98

Example protocol for a client to send an encrypted message to a server

Number of blocks

Which key?

N	u	m	b	l	k	s	:	1	K	e	y	l	D	:	A3
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
98	CC	BE	01	FF	26	39	97	85	A1	02	1E	BC	A5	7E	98

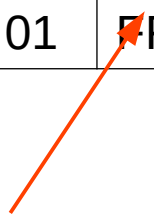
Example protocol for a client to send an encrypted message to a server

N	u	m	B	l	k	s	:	1	K	e	y	I	D	:	A3
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
98	CC	BE	01	FF	26	39	97	85	A1	02	1E	BC	A5	7E	98

IV is randomly chosen but visible on the wire and known to you, won't be 0 like in this illustration

Example protocol for a client to send an encrypted message to a server

N	u	m	B	l	k	s	:	1	K	e	y	I	D	:	A3
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
98	CC	BE	01	FF	26	39	97	85	A1	02	1E	BC	A5	7E	98



Ciphertext is what you want to decrypt, you will recover the plaintext without needing to know the key!

Server response is visible to you

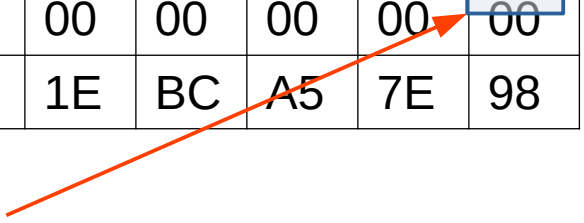
- “Message decrypted successfully”

---or---

- “Padding error during decryption”

You can record a client message and replay it to the server

N	u	m	b	l	k	s	:	1	K	e	y	l	D	:	A3
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
98	CC	BE	01	FF	26	39	97	85	A1	02	1E	BC	A5	7E	98



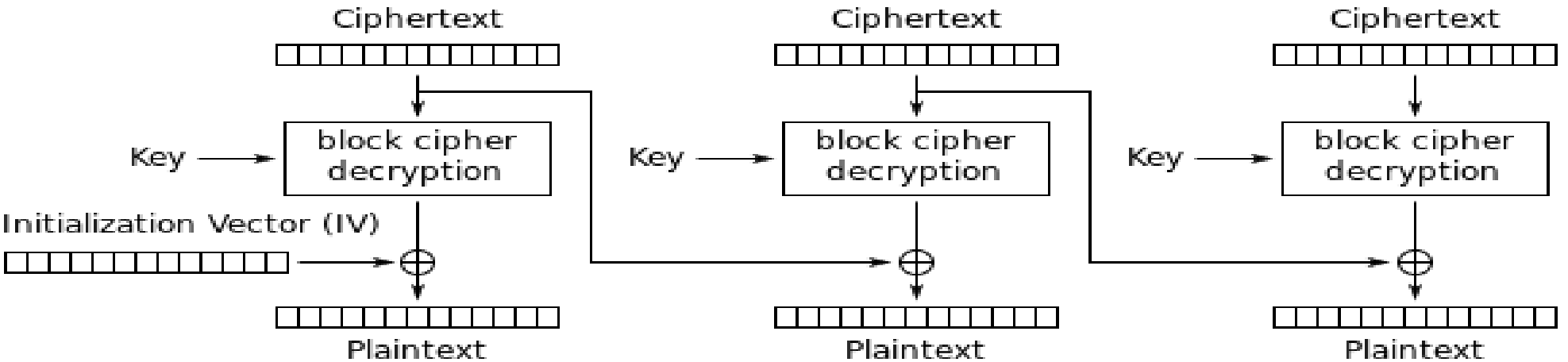
Try every value of this byte from 00 to FF

You can record a client message and replay it to the server

N	u	m	b	l	k	s	:	1	K	e	y	l	D	:	A3
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
98	CC	BE	01	FF	26	39	97	85	A1	02	1E	BC	A5	7E	98

Try every value of this byte from 00 to FF,
will flip bits here...

H	e	l	l	o	20	W	o	r	l	d	!	\n	03	03	03



Cipher Block Chaining (CBC) mode decryption

Suppose two values give valid padding

- 00 gives valid padding, this is just confirmation that the original plaintext has valid padding
- 02 also gives valid padding
  - Can recover one byte of plaintext:

    Q XOR 02 == 01, so... Q == 01 XOR 02 == 03

    Q is the byte of plaintext we're trying to guess

WTF?

N	u	m	b	l	k	s	:	1	K	e	y	I	D	:	A3
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	02
98	CC	BE	01	FF	26	39	97	[illegible]	A1	02	1E	BC	A5	7E	98

H	e	l	l	o	20	W	o	r	l	d	!	\n	03	03	01

“Information only has meaning in that it is subject to interpretation”

01 XOR 02 = 03

N	u	m	b	l	k	s	:	1	K	e	y	l	D	:	A3
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01
98	CC	BE	01	FF	26	39	97	[illegible]	A1	02	1E	BC	A5	7E	98

H	e	l	l	o	20	W	o	r	l	d	!	\n	03	03	02

Now attack here

01 XOR 02 = 03

Hold this at 01

N	u	m	b	l	k	s	:	1	K	e	y	I	D	:	A3
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01
98	CC	BE	01	FF	26	39	97	[illegible]	A1	02	1E	BC	A5	7E	98

H	e	l	l	o	20	W	o	r	l	d	!	\n	03	03	02

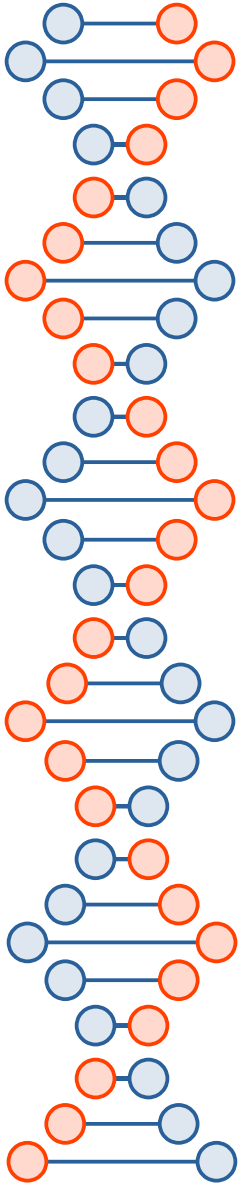
Now attack here

Discussion

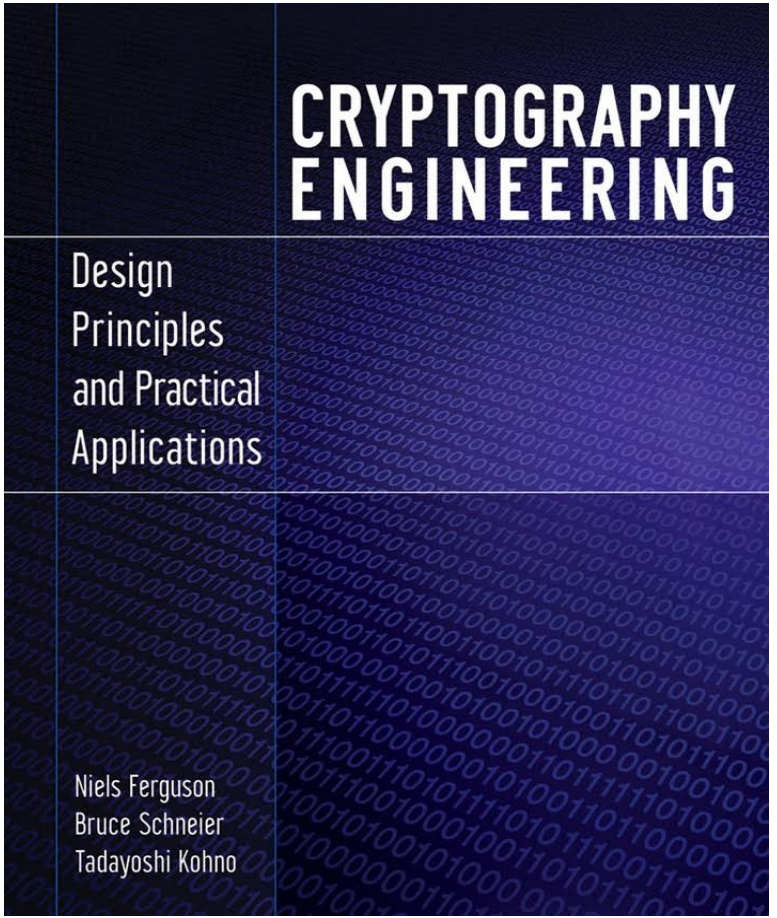
- You still don't know the key, probably never will
- It doesn't matter how secure AES is or the size of the key
- CBC is probably the most commonly used mode for some application types
- What if a byte is already what it needs to be?
- What if there is more than one block?

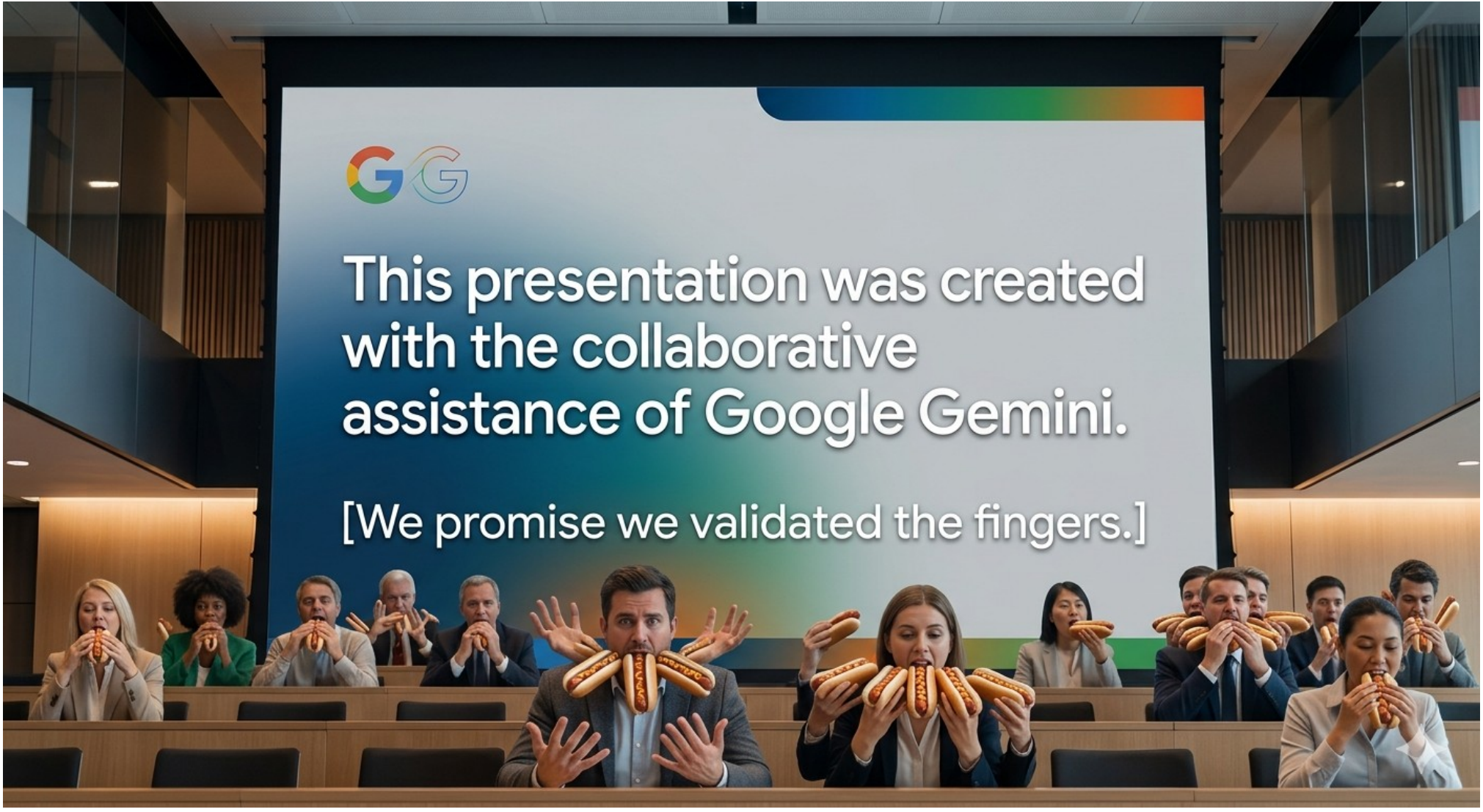
References

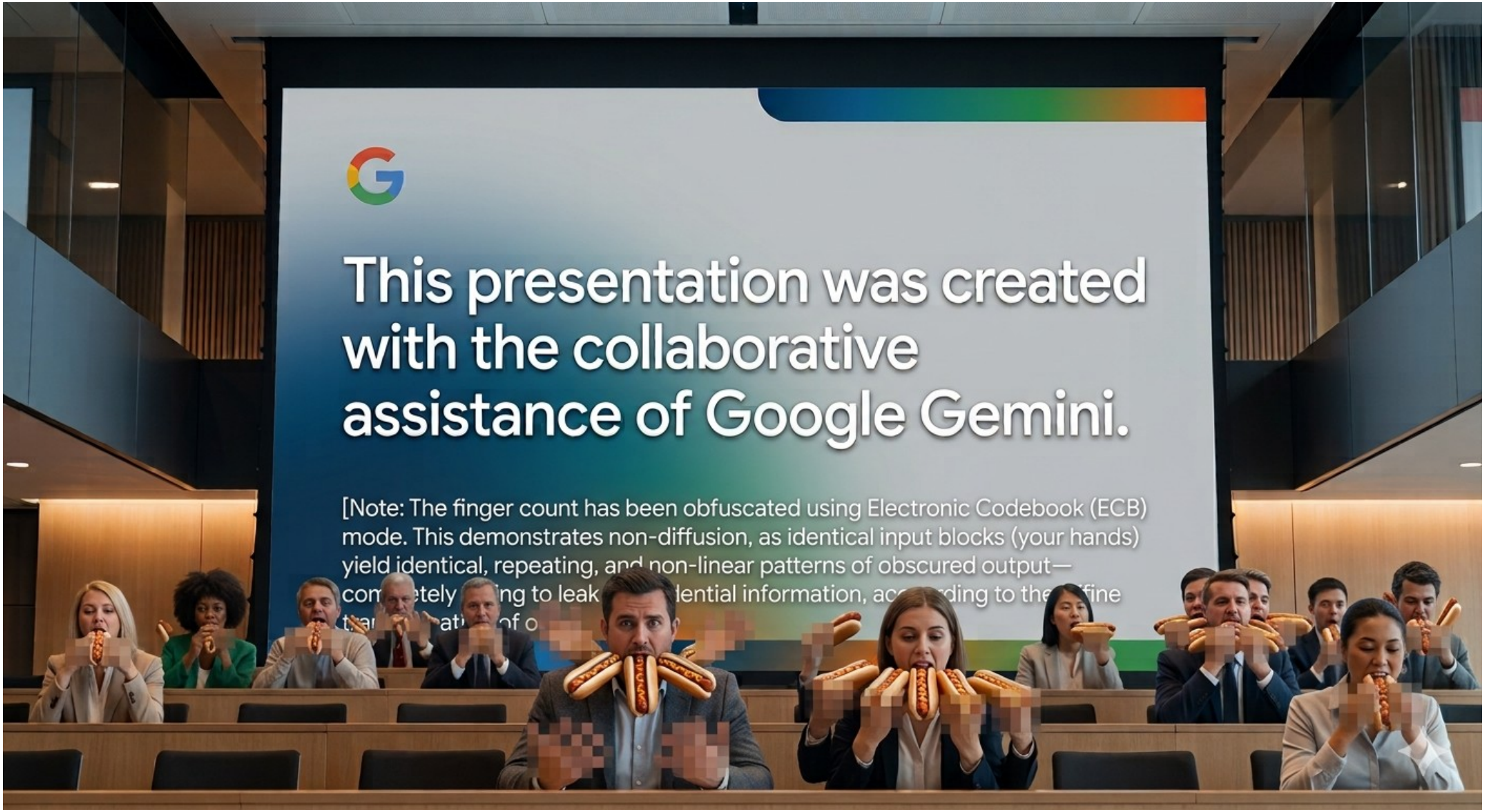
- https://grymoire.wordpress.com/2014/12/05/cbc-padding-oracle-attacks-simplified-key-concepts-and-pitfalls/



Cryptography Engineering by Ferguson *et al.*




This presentation was created with the collaborative assistance of Google Gemini.
[We promise we validated the fingers.]


This presentation was created with the collaborative assistance of Google Gemini.
[Note: The finger count has been obfuscated using Electronic Codebook (ECB) mode. This demonstrates non-diffusion, as identical input blocks (your hands) yield identical, repeating, and non-linear patterns of obscured output—