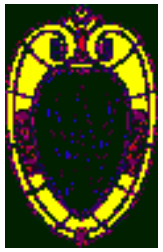


CS/ECE/Math 435
Introduction to Cryptography

Professor Chris DeMarco
Department of Electrical & Computer Engineering
University of Wisconsin-Madison

Spring Semester 2021

Background:

In 2000, U.S. National Institute for Standards and Technology (NIST) put out a public call for proposals for an encryption standard to replace DES. Selection process included a competition.

Winning design submitted by a team of Belgian Cryptographers: Joan Daemen & Vincent Rijmen.

General structure of their design has much in common with DES

AES improvements over DES:

- Overall block size 128 bits (DES was 64)
  (128 = 16 × 8)
- Selectable, larger key lengths
  (128, 192, or 256 bits; DES was 56)
- Nonlinear algorithm to select subkeys from master key
  (DES used simple linear step to create subkeys).
- Critical nonlinearity in encryption functions based on inverse function in $BF(2^8)$ (no need for Feistel construct, no look-up tables).

Stucture and Notation for AES
for a representative Block encryption
operation:

Overall block size is 128 bits;
a block is decomposed/partitioned
into sixteen 8-bit binary vectors.
Later steps organize as 4 x 4 x [8-bit].

Each 8-bit binary vector is
interpreted as a "field element"
of $GF(2^8)$; Bach notes use shorthand
of $F = GF(2^8)$.

Recall simple association:

$GF(2^8)$ polynomial to 8-bit words:

$0 \cdot X^7 + 0 \cdot X^6 + 0 \cdot X^5 + 0 \cdot X^4 + 0 \cdot X^3 + 0 \cdot X^2 + 0 \cdot X^1 + 0 \cdot X^0$

$\Leftrightarrow$ 0 0 0 0 0 0 0 0 as 8-bit word.

$0 \cdot X^7 + 0 \cdot X^6 + 1 \cdot X^5 + 0 \cdot X^4 + 0 \cdot X^3 + 0 \cdot X^2 + 1 \cdot X^1 + 0 \cdot X^0$

$\Leftrightarrow$ 0 0 1 0 0 0 1 0 as 8-bit word

in general: $c_7\ c_6\ c_5\ c_4\ c_3\ c_2\ c_1\ c_0$

BUT ALL ALGEBRAIC OPERATIONS ON THESE 8-bit WORDS FOLLOW RULES OF $GF(8)$! (with one modification ...)

FOR CONVENIENCE: we "extend" the inverse operation on $GF(8)$.

In particular, for $\underline{u} \in GF(8)$

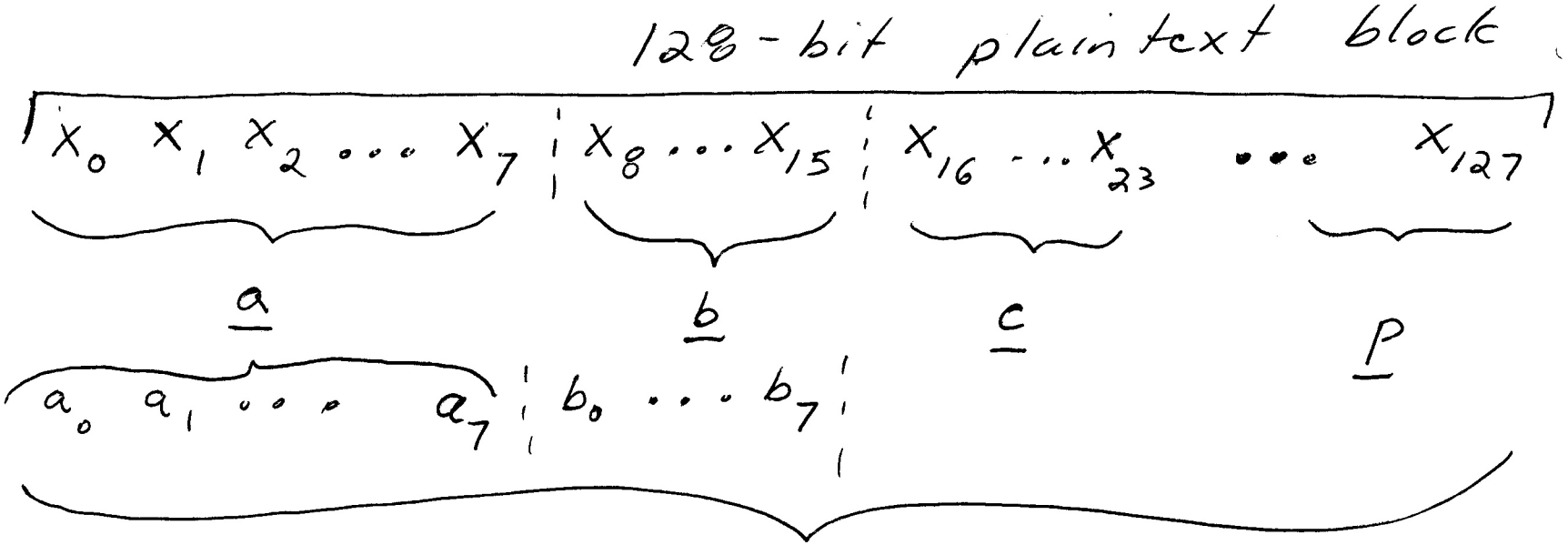
↑ we use the data structure of 8-dimensional vector with binary elements, but interpret as polynomial.

$$\underline{u}^{-1} = \begin{cases} \underline{0} & \text{if } \underline{u} = \underline{0}. \\ \text{multiplicative inverse of } \underline{u} \text{ in } GF(8) & \text{if } \underline{u} \neq 0. \end{cases}$$

Recall that the critical nonlinear encryption steps in DES ultimately reduced to 6-bit input, 4-bit output nonlinear Boolean functions defined by look-up tables. Termed "S-boxes."

Editorial comment: Weird an inelegant. (Daemen & Rijmen seemed to feel same).

For discussion to follow, let us define how 128-bit block is decomposed into 8-bit sub-blocks, and the simple array data structure used for these sub-blocks



16 x 8-bit sub-blocks.
each sub-block $\in GF(2^8)$

For later, "mixing" steps of AES algorithm, we'll organize these into 4×4 arrays, with each element of the array <u>represented</u> by a vector in $\mathbb{Z}_2^8$, but <u>interpreted</u> as an element in $GF(2^8)$.

AES's Improvement over DES's S-boxes.

Core nonlinear step combines extended inversion operation, $GF(2^8) \to GF(2^8)$, with an affine Hill encryption (i.e., matrix operation with additive offset).

Denote overall function $BS: GF(2^8) \to GF(2^8)$.

CAUTION: We again "mix" two interpretations of $\underline{a} \in GF(2^8)$ below. For inverse operation, we genuinely treat $\underline{a}$ as a polynomial in $GF(2^8)$. But for

the matrix operation, we simply treat $\underline{u}^{-1}$ as an 8-element vector. Choose $[A] \in \mathbb{Z}_2^{8\times 8}$, $\underline{b} \in \mathbb{Z}_2^8$

$$BS(\underline{u}) = [A]\cdot \underline{u}^{-1} + \underline{b} \mod 2$$

Again: after computing $\underline{u}^{-1}$, interpreting $\underline{u} \in GF(2^8)$, we then treat $\underline{u}^{-1}$ as an "ordinary" vector in $\mathbb{Z}_2^8$, and do "ordinary" matrix·vector multiply (mod 2, of course)

For AES, allowable $[A]^{-1}$'s must, of course, be invertible in $\mathbb{Z}_2^{8\times 8}$.

In fact AES restricts $[A]^{-1}$'s to "circulant" matrices of form:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ \vdots & & & & & & & \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \leftarrow \begin{cases} \text{row } k+1 \\ \text{"circulates"} \\ \text{row } k \text{ one} \\ \text{position right} \end{cases}$$

With this structure, straightforward to show $\det(A) \bmod 2 = 1 \Rightarrow$ invertible.

$\underline{b}$ is just chosen as

$$\underline{b} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Therefore; given A, b as described:

$$\underline{v} = BS(\underline{u}) = A \cdot \underline{u}^{-1} + \underline{b} \mod 2$$

$$\Rightarrow A^{-1}\underline{v} = \underline{u}^{-1} + \underline{b} \mod 2$$

$$\Rightarrow A^{-1}\underline{v} + \underline{b} \mod 2 = \underline{u}^{-1}$$

$$\Rightarrow \underline{u} = \underbrace{\left[A^{-1}\underline{v} + \underline{b} \mod 2\right]}^{-1} = BS^{-1}(\underline{v})$$

again, interpret these as "ordinary" mod 2 operations on vectors with elements in $\mathbb{Z}_2$

But the inverse operation is the extended multiplicative inverse in $GF(2^8)$

So core nonlinear "ingredient" of AES encryption is this function $BS(\cdot)$, with a well defined $BS^{-1}(\cdot)$ for decryption. Like DES, AES will use multiple "rounds," that will also involve other linear steps.

For shorthand, we'll speak of a "BS step."

Add Round Key ("ARK step") Ingredient:

At any step of algorithm, we have 128-bit state: the state is the plaintext block at start, it is intermediate ciphertext at any intermediate step.

Let $\underline{s}$ denote state, $\underline{k} \in \mathbb{Z}_2^{128}$ a 128-bit key. ARK step is simply:

$$ARK_{k_0 \cdots 1}(\underline{s}) = \underline{s} + \underline{k} \mod 2.$$

Recall that the AES has a 128-bit key as given data; denote this as $\underline{k}_0 \in \mathbb{Z}_2^{128}$.

Q: How does AES generate additional $\underline{k}_i$'s, with i^{th} key used in the "i^{th} round" of AES.

(Aside: AES will use 11 rounds, so we'll define $k_1, k_2 \cdots k_{10}$, though we use them later in reverse order).

Ans: First decompose / organize $\underline{k}_0$ (and subsequent $\underline{k}_i$'s) as 4×4 array, with each entry $\in \mathbb{Z}_2^8$.

Label columns

$$\underline{c}_0 \quad \underline{c}_1 \quad \underline{c}_2 \quad \underline{c}_3$$

$$\left.\begin{array}{l} \\ \\ \\ \\ 4 \\ \text{rows} \end{array}\right\} \left[\begin{array}{cccc} (8\text{-bits}) & (8\text{-bits}) & \cdots & (8\text{-bits}) \\ (8\text{-bits}) & & & \vdots \\ (8\text{-bits}) & & & \vdots \\ (8\text{-bits}) & \cdots & & (8\text{-bits}) \end{array}\right]$$

} so $\underline{c}_0$ to $\underline{c}_3$ are specified from original 128-bit key $\underline{k}_0$

(note $4 \times 4 \times 8 = 128$)

Same data structure used for k_1 to k_{10}, i.e.

$\underline{k}_1$ of 128-bits $\Leftrightarrow$ $\begin{bmatrix} \underline{c}_4 & \underline{c}_5 & \underline{c}_6 & \underline{c}_7 \end{bmatrix}$

each $\underline{c}_i$ 4-dimensional vector composed of 8-bit elements

From given $\underline{c}_0$ to $\underline{c}_3$, we generate $\underline{c}_i^\circ$ for $i^\circ \geq 4$ as follows

$$\underline{c}_i^\circ = \begin{cases} \underline{c}_{i-1}^\circ + \underline{c}_{i-4}^\circ & \text{if } \mathrm{mod}(i^\circ, 4) \neq 0 \\ N_i^\circ(\underline{c}_{i-1}^\circ) + \underline{c}_{i-4}^\circ & \text{if } \mathrm{mod}(i^\circ, 4) = 0 \end{cases}$$

where

$$N_i^\circ(\underline{c}) = \begin{bmatrix} BS(\text{row 1 element of } \underline{c}) \\ BS(\text{row 2 element of } \underline{c}) \\ BS(\text{row 3 element of } \underline{c}) \\ BS(\text{row 4 element of } \underline{c}) \end{bmatrix} + \begin{bmatrix} X^{(i^\circ-4)/4} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Observe: The N_i function returns to interpreting terms as $\in BF(2^8)$; i.e. as polynomials.

So ...

$$\underbrace{BS(\text{row 1 element of } \underline{c})}_{\text{this result is interpreted as a degree 7 polynomial in } GF(2^8)} + \underbrace{X^{p}}_{p = \frac{i-4}{4}}$$

suppose $i = 16$, $p = 3$, we'd be adding the polynomial

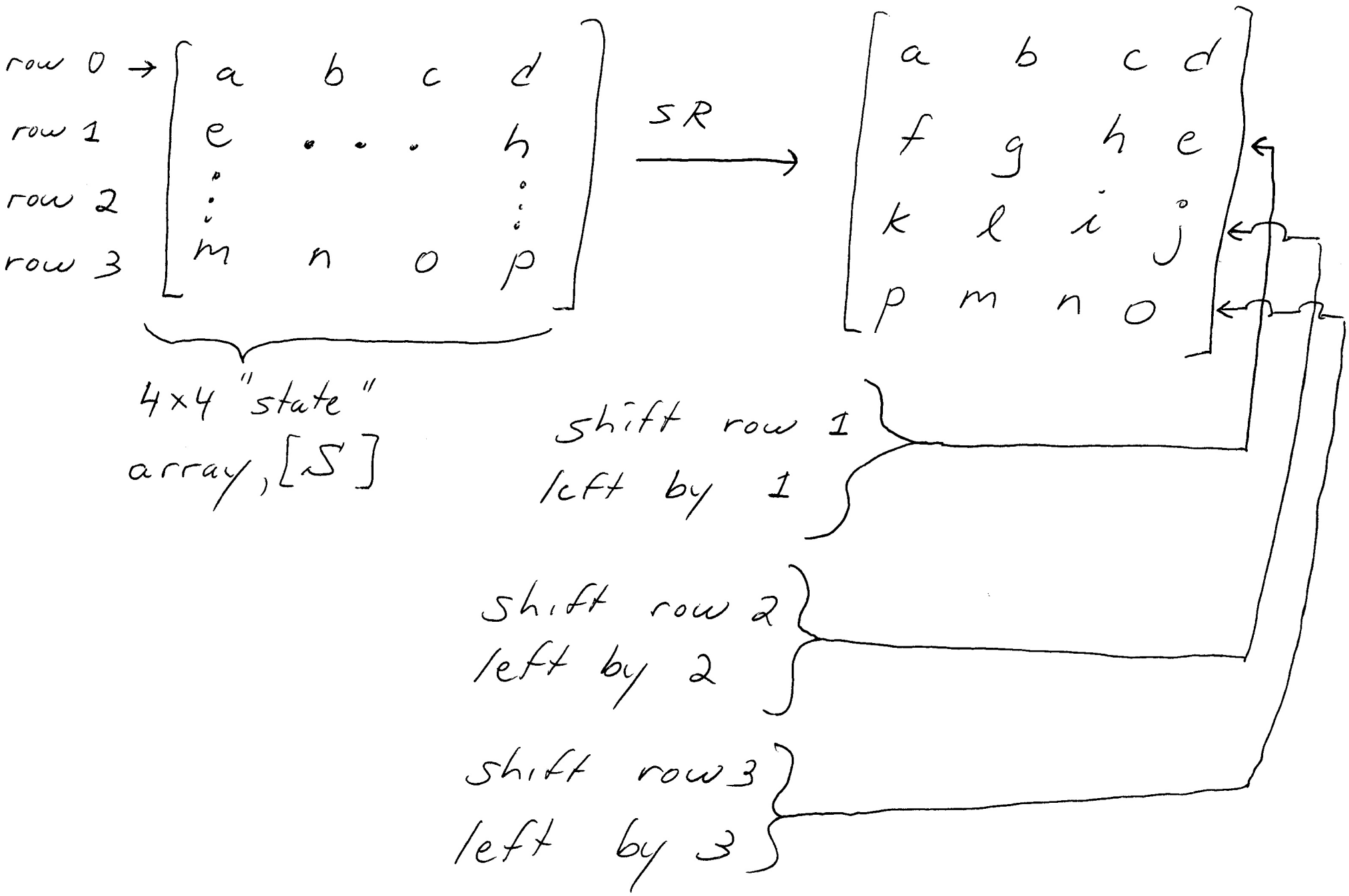
$$[0 \cdot X^7 + 0 \cdot X^6 + \ldots + 1 \cdot X^3 + 0 \cdot X^2 \ldots + 0]$$

Clever recycling of BS(·) functions to generate intermediate subkeys.

Shift Rows Ingredient

Recall that original input plaintext block, 128-bits, is organized as 4×4 array of 8-bit elements. Each intermediate ciphertext (the "state") is similarly organized.

The SR operator is



The Mix Columns Ingredient

Here we perform a polynomial (i.e. treat elements as $\in GF(2^8)$) matrix multiply:

$$MC([S]) = \begin{bmatrix} X & (X+1) & 1 & 1 \\ 1 & X & (X+1) & 1 \\ 1 & 1 & X & (X+1) \\ (X+1) & 1 & 1 & X \end{bmatrix} [S]$$

↑

again, 4×4 matrix/array with elements in $GF(2^8)$

Overall Block Encryption Operation

for AES: $\underline{X}$ = 128-bit input plaintext

$\underline{Y}$ = 128-bit output ciphertext

$\underline{X}$, and intermediate ciphertext results are organized in data structures composed of 4×4 arrays with 8-bit elements, denoted $[S]$

Eleven "rounds" of operations employing BS, ARK, SR, and MC steps. Each round uses a distinct $\underline{k}_i$ (128 bits, then decomposed into four $\underline{c}$ columns)

Caution on key indexing notation:
we "use" keys in reverse order,
relative to their index "i."

Overall flow diagram for AES:

$\underline{X}$ (organized as $[S]_0$) $\xrightarrow{ARK[k_{10}]}$ $[S]_1$

} Think of $[S]_1, [S]_2 \ldots$ as intermediate ciphertext

$\xrightarrow{BS, SR, MC, ARK[k_9]}$ $[S]_2$

$\xrightarrow{BS, SR, MC, ARK[k_8]}$ $[S]_3$

• • •

NOTE: MC step skipped for last round

$\xrightarrow{BS, SR, ARK[k_0]}$ $[S]_{10} = \underline{y}$