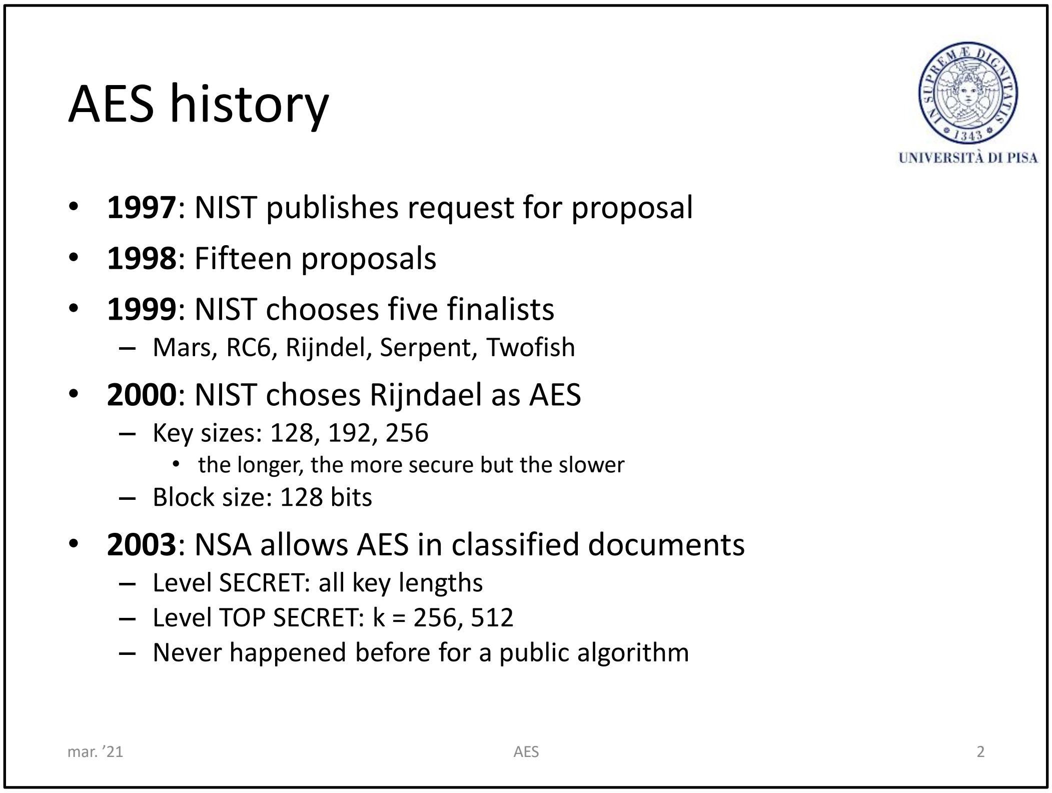
Advanced Encryption Standard (AES)

Gianluca Dini
Dept. of Ingegneria dell'Informazione
University of Pisa
gianluca.dini@unipi.it

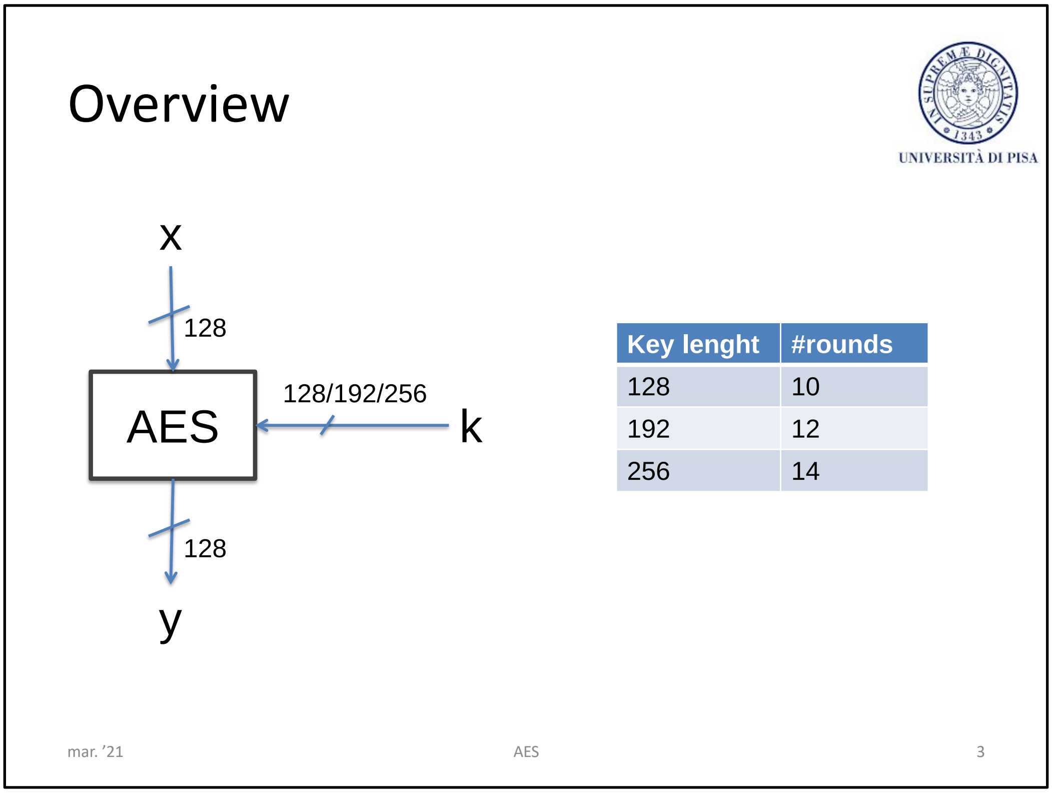
Version: 2021-03-16



AES history

- **1997**: NIST publishes request for proposal
- **1998**: Fifteen proposals
- **1999**: NIST chooses five finalists
  - Mars, RC6, Rijndel, Serpent, Twofish
- **2000**: NIST choses Rijndael as AES
  - Key sizes: 128, 192, 256
    - the longer, the more secure but the slower
  - Block size: 128 bits
- **2003**: NSA allows AES in classified documents
  - Level SECRET: all key lengths
  - Level TOP SECRET: k = 256, 512
  - Never happened before for a public algorithm

- **Time made evident that DES and 3DES were not designed for modern hw and thus are too slow.**



UNIVERSITÀ DI PISA

Overview

x

128

AES

128/192/256

k

128

y

Key lenght	#rounds
128	10
192	12
256	14

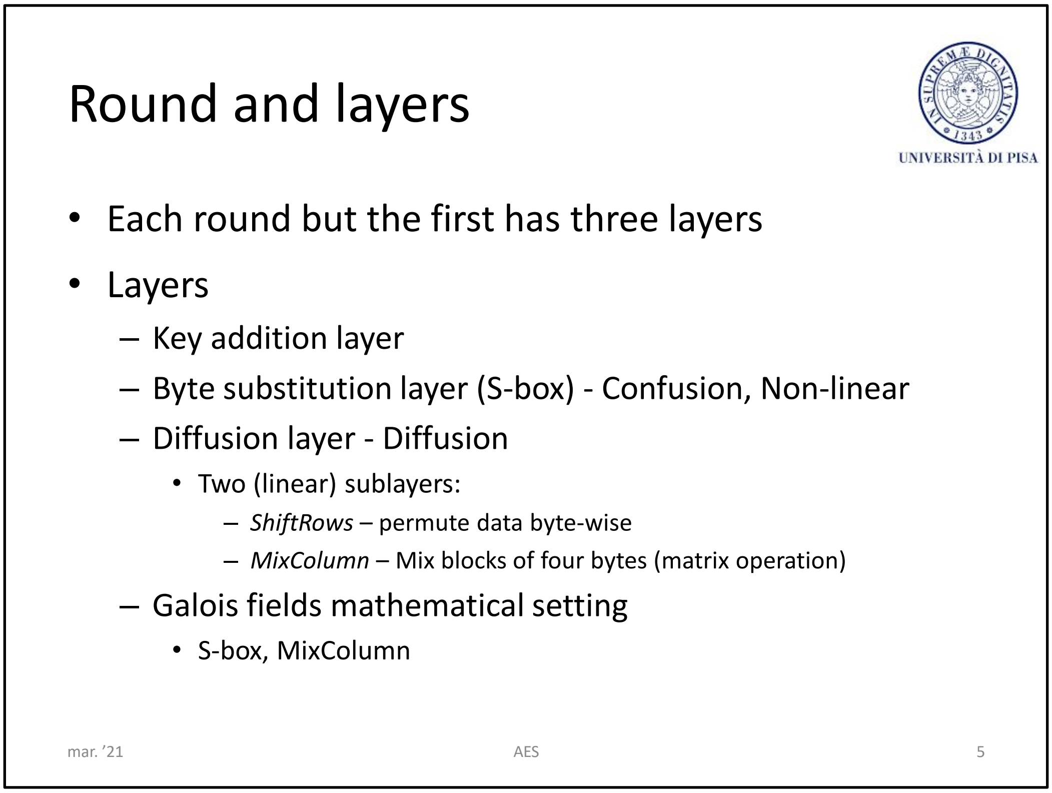
mar. '21 AES 3

AES has not a Feistel structure. Each round encrypts all bits and not just half of them. That is why AES has a comparably small number of rounds.

Introduction

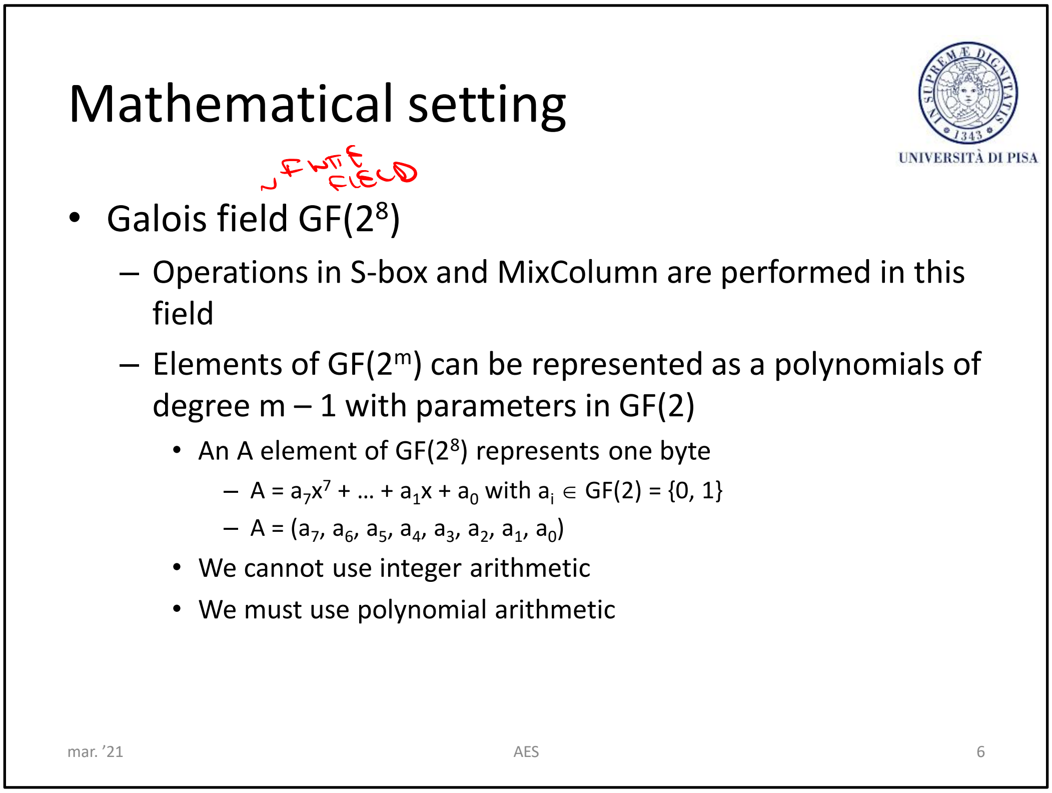
UNIVERSITÀ DI PISA

- AES
  - Has rounds
  - Does not have a Feistel network structure → DES CRIPTA SOLO METÀ DEI DATI
  - Encrypts an entire block in each round
    - DES encrypts half a block => $\#round_{AES} < \#round_{DES}$
  - Data path is called «state»



Round and layers

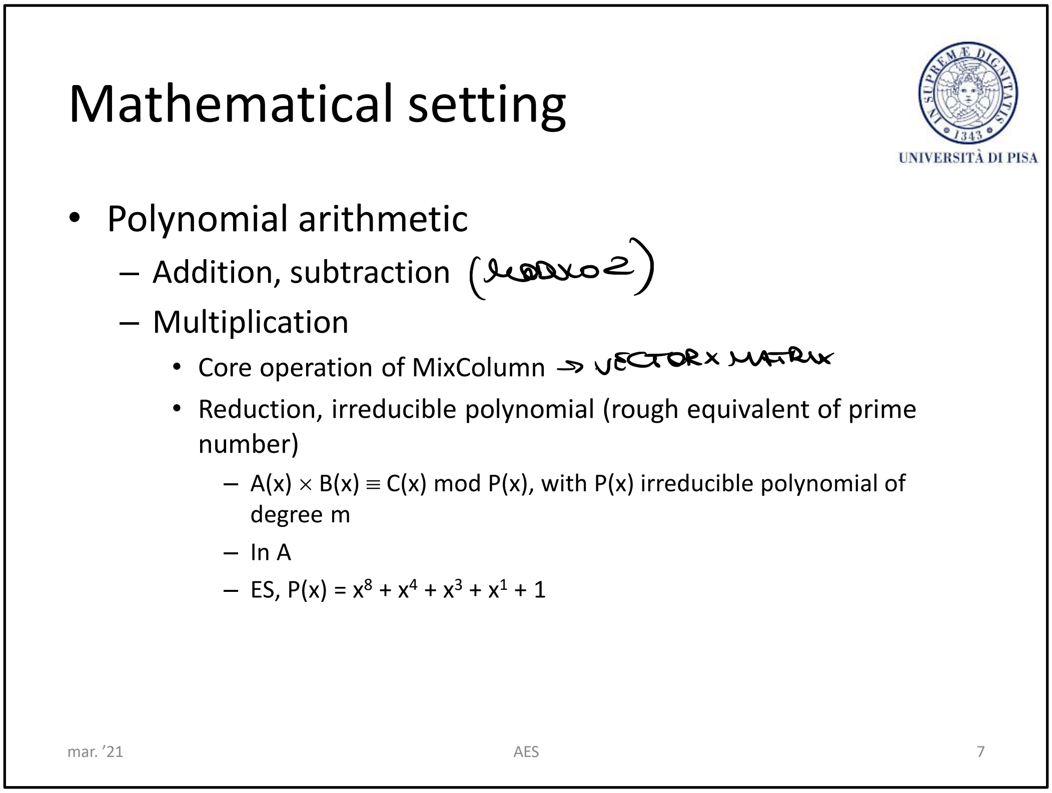
- Each round but the first has three layers
- Layers
  - Key addition layer
  - Byte substitution layer (S-box) - Confusion, Non-linear
  - Diffusion layer - Diffusion
    - Two (linear) sublayers:
      - *ShiftRows* – permute data byte-wise
      - *MixColumn* – Mix blocks of four bytes (matrix operation)
  - Galois fields mathematical setting
    - S-box, MixColumn



Mathematical setting

~ FINITE FIELD

- Galois field GF(2^8)
  - Operations in S-box and MixColumn are performed in this field
  - Elements of GF(2^m) can be represented as a polynomials of degree m – 1 with parameters in GF(2)
    - An A element of GF(2^8) represents one byte
      - $A = a_7x^7 + \ldots + a_1x + a_0$ with $a_i \in GF(2) = \{0, 1\}$
      - $A = (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$
    - We cannot use integer arithmetic
    - We must use polynomial arithmetic



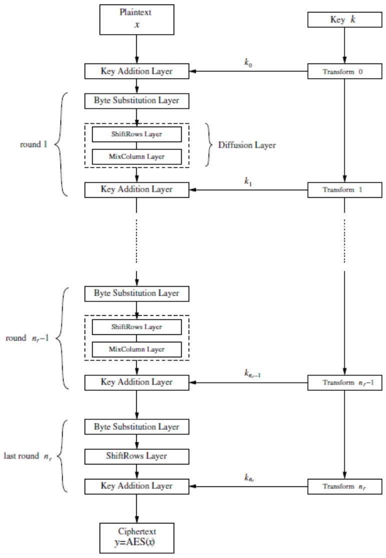
Mathematical setting

- Polynomial arithmetic
  - Addition, subtraction (MODULO 2)
  - Multiplication
    - Core operation of MixColumn → VECTOR x MATRIX
    - Reduction, irreducible polynomial (rough equivalent of prime number)
      - $A(x) \times B(x) \equiv C(x) \bmod P(x)$, with P(x) irreducible polynomial of degree m
      - In A
      - ES, $P(x) = x^8 + x^4 + x^3 + x^1 + 1$

Mathematical setting

- Polynomial arithmetic
  - Division
    - Core operation of Byte Substitution (S-boxes)
    - $A(x) \cdot A(x)^{-1} \equiv 1 \bmod P(x)$
    - In small fields (smaller than 2^{16} elements), inverse can be precomputed by lookup tables

AES encryption block diagram



Plaintext x

Key k

Key Addition Layer

k_0

Transform 0

round 1

Byte Substitution Layer

ShiftRows Layer

MixColumn Layer

Diffusion Layer

Key Addition Layer

k_1

Transform 1

round n_r-1

Byte Substitution Layer

ShiftRows Layer

MixColumn Layer

Key Addition Layer

k_{n_r-1}

Transform n_r-1

last round n_r

Byte Substitution Layer

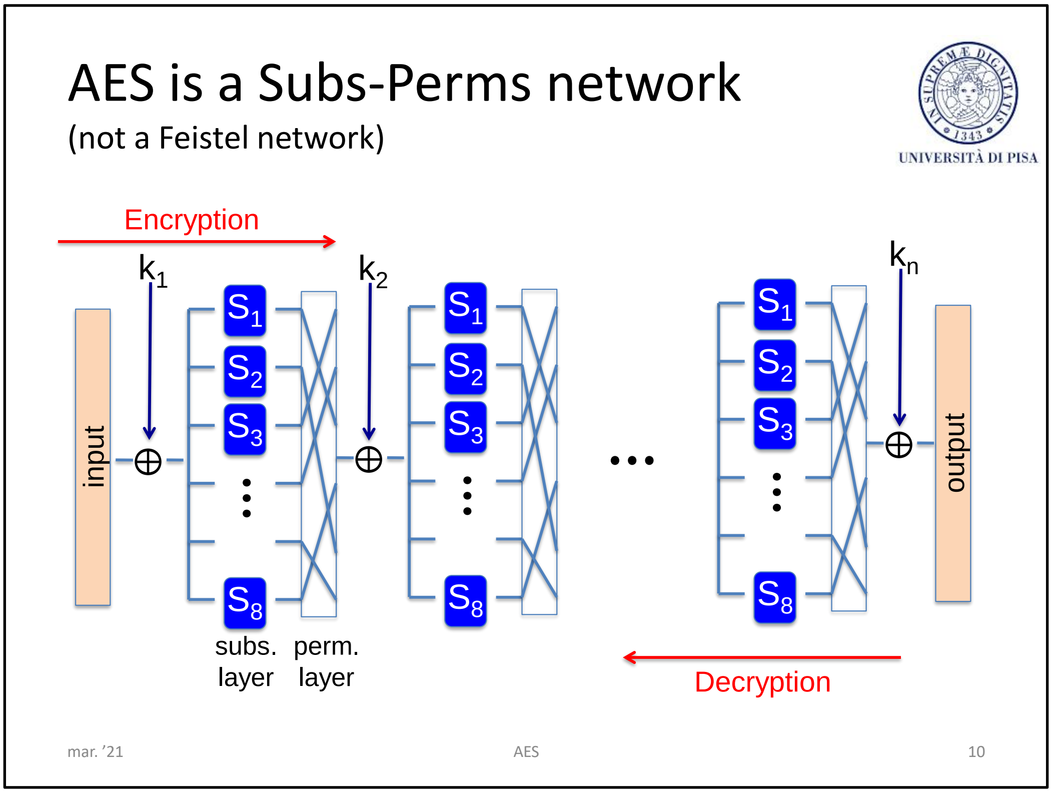
ShiftRows Layer

Key Addition Layer

k_{n_r}

Transform n_r

Ciphertext y=AES(x)



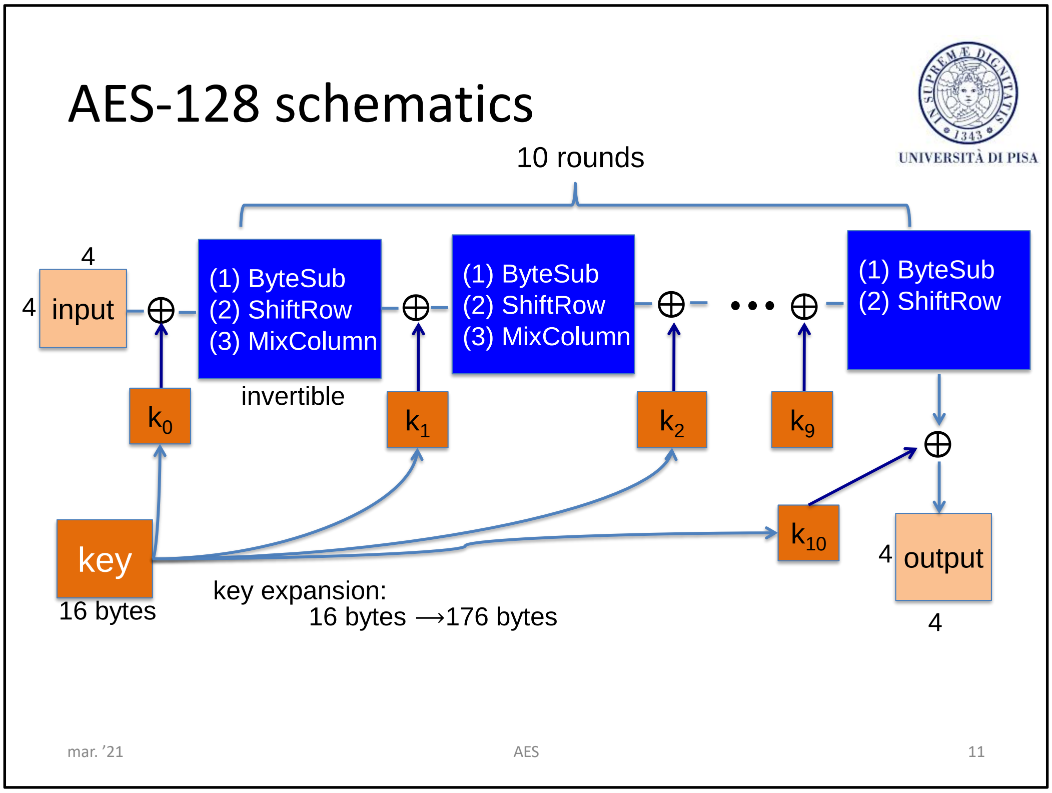
AES is a Subs-Perms network

(not a Feistel network)

UNIVERSITÀ DI PISA

mar. '21 AES 10

- AES is not based on a Feistel Network. In contrast, AES is based on a Substitutions-Permutations Network. In a Feistel network, half bits are not changed from round to round. In a S-P network, all bits are changed from round to round. For this reason, a S-P network requires a smaller number of rounds than a Feistel network
- Every layer is invertible, so the whole thing is invertible. In order to decrypt, rounds have to be applied in reverse order. Consider one round. When decrypting, we first apply permutation and then substitution. This implies that for a round to be invertible then, differently from DES, in AES the substitution layer is invertible although non-linear.



AES-128 schematics

mar. '21 AES 11

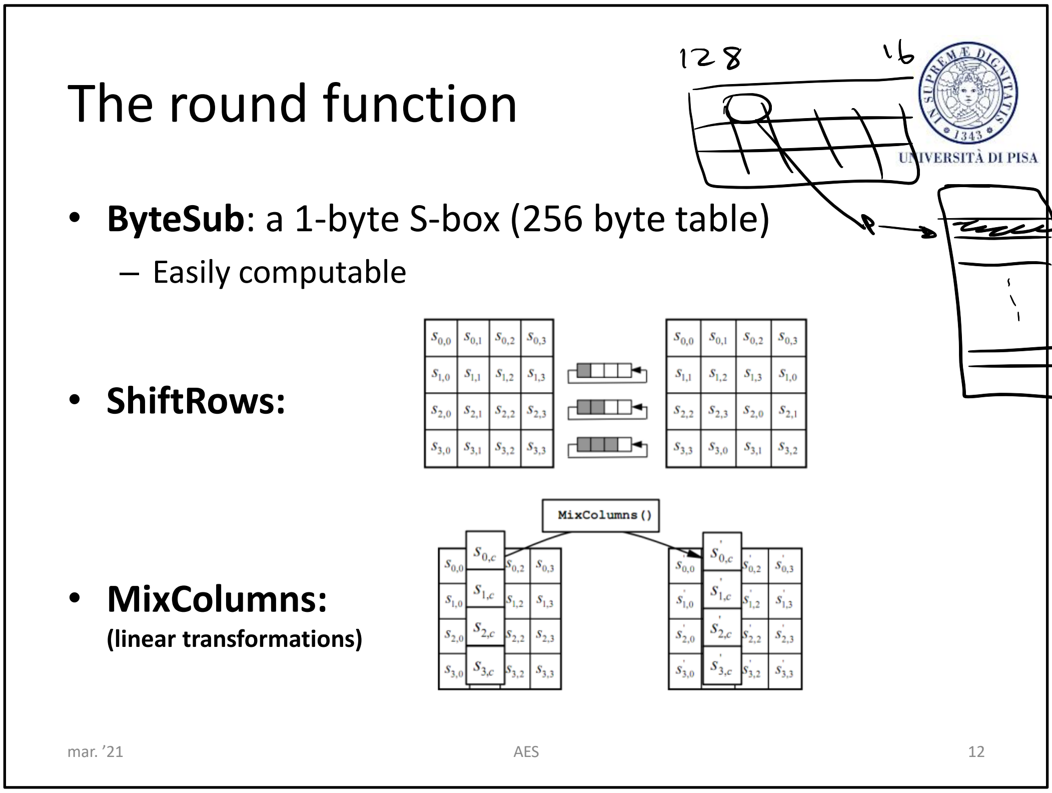
AES is a ten-rounds P-S network.

Input, state, and output state are structured as a 4 x 4 matrix. Each cell of the matrix contains one byte. (4 x 4 x 8bit = 128 bits)

Notice that in the last round, MixColumn operation is missing which makes the encryption and decryption scheme symmetric (remember that rounds in encryption and decryption are traversed in reverse order).

- Key Addition Layer. The subkey is xored to the state. The key schedule derives subkeys from the AES key.
- Byte Substitution (S box) layer introduces confusion. Changes in individual state bits propagate quickly across the data path.
- Diffusion layer provides diffusion. It consists of two sub-layers: ShiftRows and MixColumn.

All operations are byte-wise. This is in contrast to DES, which makes heavy use of bit permutation and can thus be considered to have a bit-oriented structure.



The round function

- **ByteSub**: a 1-byte S-box (256 byte table)
  - Easily computable
- **ShiftRows:**
- **MixColumns:**
  (linear transformations)

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

128
16
286
MixColumns()
$s_{0,0}$ $s_{0,c}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,c}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,c}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,c}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,c}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,c}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,c}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,c}$ $s'_{3,2}$ $s'_{3,3}$

mar. '21 AES 12

BYTE SUBSTITUTION LAYER.

It is implemented as an S-Box containing 256 bytes. AES applies the S-box to every byte in the current state. The current state is the four-by-four byte-element table. AES applies the S-box to each element in the table. Let A be the table and let B = S(A), then for all i, j, $B_{ij} = S(A_{ij})$.

ByteSubstitution is the only non-linear element in AES: S(A) + S(B) ≠ S(A+B).

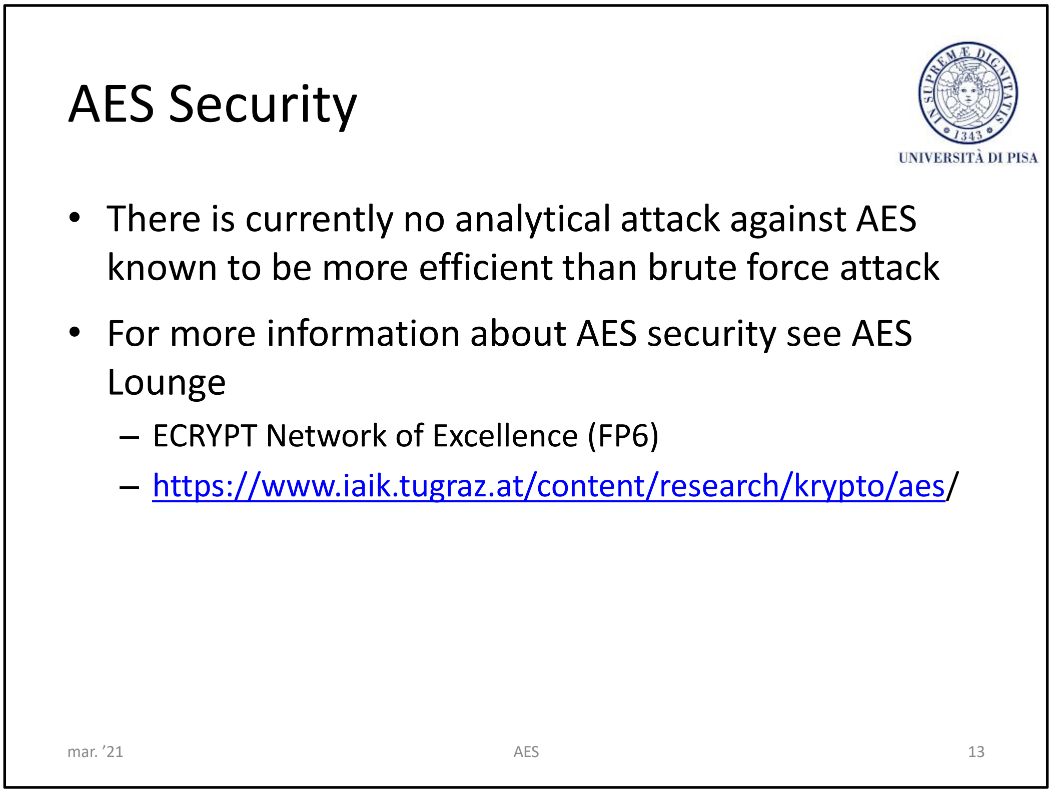
ByteSubstitution is a bijective mapping, although non linear, which maps one-to-one input to output. This makes ByteSubstitution invertible and thus makes decryption possible.

DIFFUSION LAYER.

Differently from the non-linear S-Box, the diffusion layer is linear: DIFF(A) + DIFF(B) = DIFF(A+B).

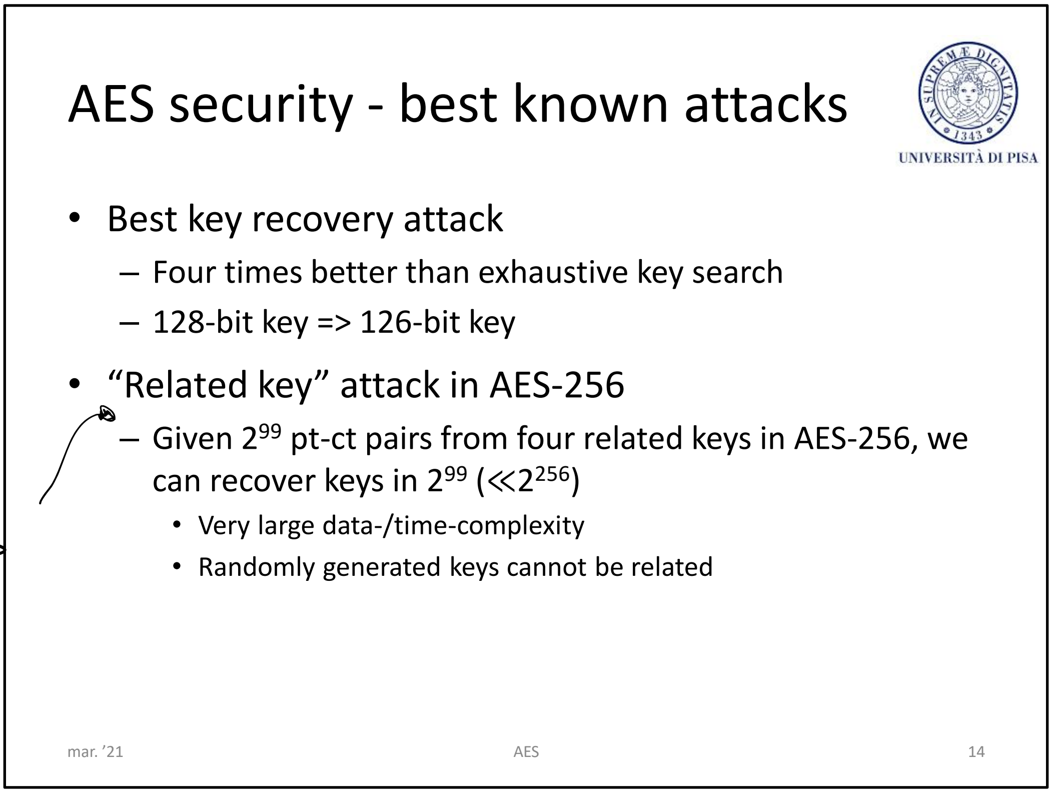
ShiftRow transformation cyclically shifts the second row of the state matrix by one byte to the left, the third row by two bytes to the right and the fourth row by three bytes to the right.

MixColumns is a linear transformation which mixes each column of the state matrix. We apply linear transformations independently to each one of the columns. The operation consists in multiplying the column by a matrix that represents the linear transformations. It follows that each input bytes influences four output bytes, then MixColumns is the major diffusion component of AES.



AES Security

- There is currently no analytical attack against AES known to be more efficient than brute force attack
- For more information about AES security see AES Lounge
  - ECRYPT Network of Excellence (FP6)
  - https://www.iaik.tugraz.at/content/research/krypto/aes/



AES security - best known attacks

- Best key recovery attack
  - Four times better than exhaustive key search
  - 128-bit key => 126-bit key
- “Related key” attack in AES-256
  - Given 2^{99} pt-ct pairs from four related keys in AES-256, we can recover keys in 2^{99} ($\ll 2^{256}$)
    - Very large data-/time-complexity
    - Randomly generated keys cannot be related

SOLO TEORICO

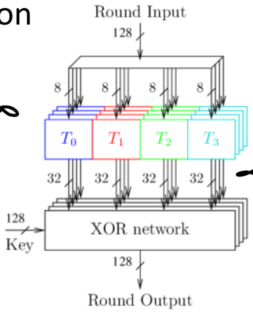
mar. ’21 AES 14

- **Key recovery attack**. The best key recovery attack is four times better than exhaustive key search. This means that it is “as if” AES had a 126-bit key. Notwithstanding a 2^{126}-time attack is impractical.
- **Related key attack**. It exploits weakness in the key expansion algorithm. It is assumed that pt-ct pairs come from related keys. This means that i) k2 is equal to k1 except a few bits are flipped; ii) key k3 is equal to k2 except a few bits are flipped; and so on and so forth. In a more theoretical way, we may say that the Humming distance between keys is very short. Related key attack has large time- and input-complexities which is however much smaller than 2^{256} time. Anyhow, related key attack relies on a very strong assumption, namely, inputs come from four related keys. In practice, this should never be the case because keys must be generated at random and thus, they cannot be related.

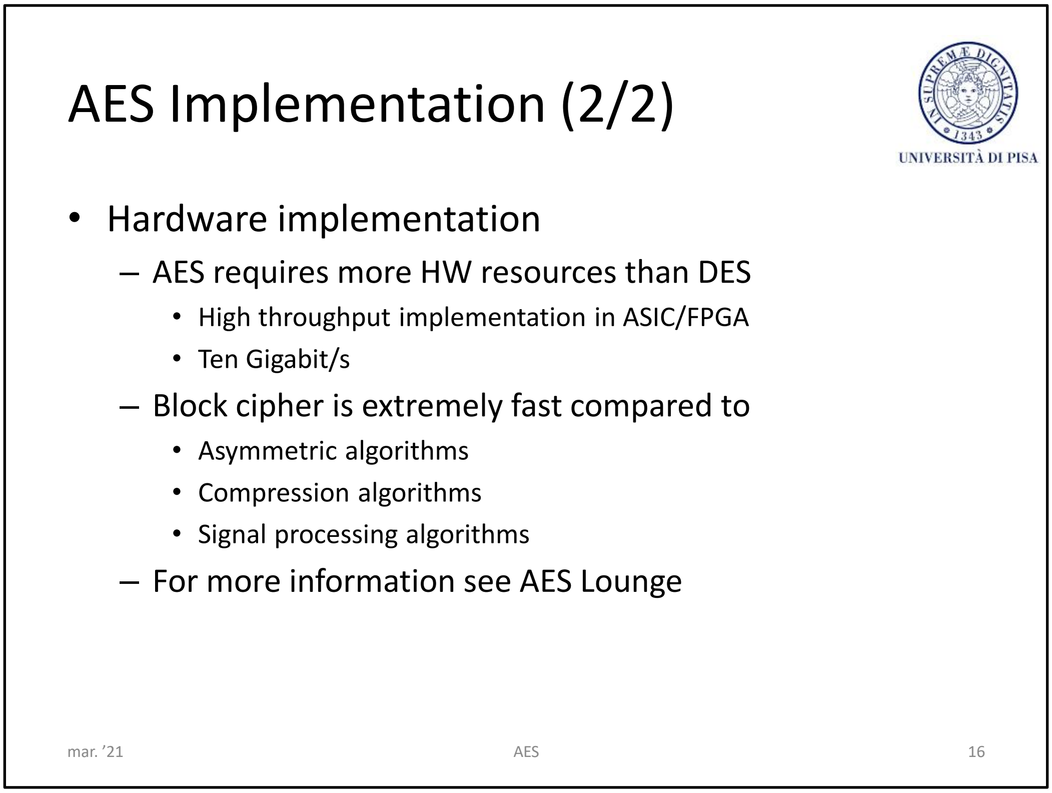
AES Performance (1/2)

- Software implementation
  - Direct implementation is well-suited for 8-bit processors (e.g., smartcard)
    - Processing 1-byte per instruction
  - For 32-/64-bit architecture, T-box optimization
    - Merge all the round functions into one look-up table (but key addition)
      - 4 tables (1 per byte) of 256 entries; each entry is 32 bit
      - 1 round, 16 lookups
  - Few hundreds Mbit/s

PRECOMPUTED OGNI LAYER!

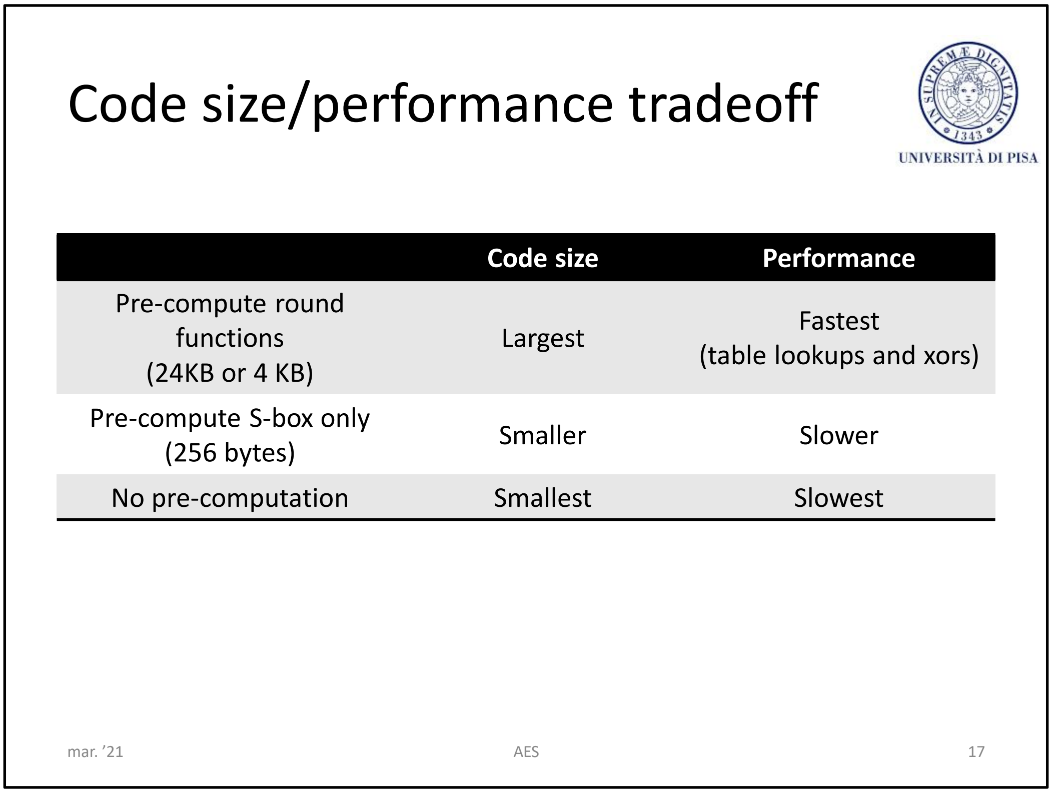


LOOK-UP TABLE



AES Implementation (2/2)

- Hardware implementation
  - AES requires more HW resources than DES
    - High throughput implementation in ASIC/FPGA
    - Ten Gigabit/s
  - Block cipher is extremely fast compared to
    - Asymmetric algorithms
    - Compression algorithms
    - Signal processing algorithms
  - For more information see AES Lounge



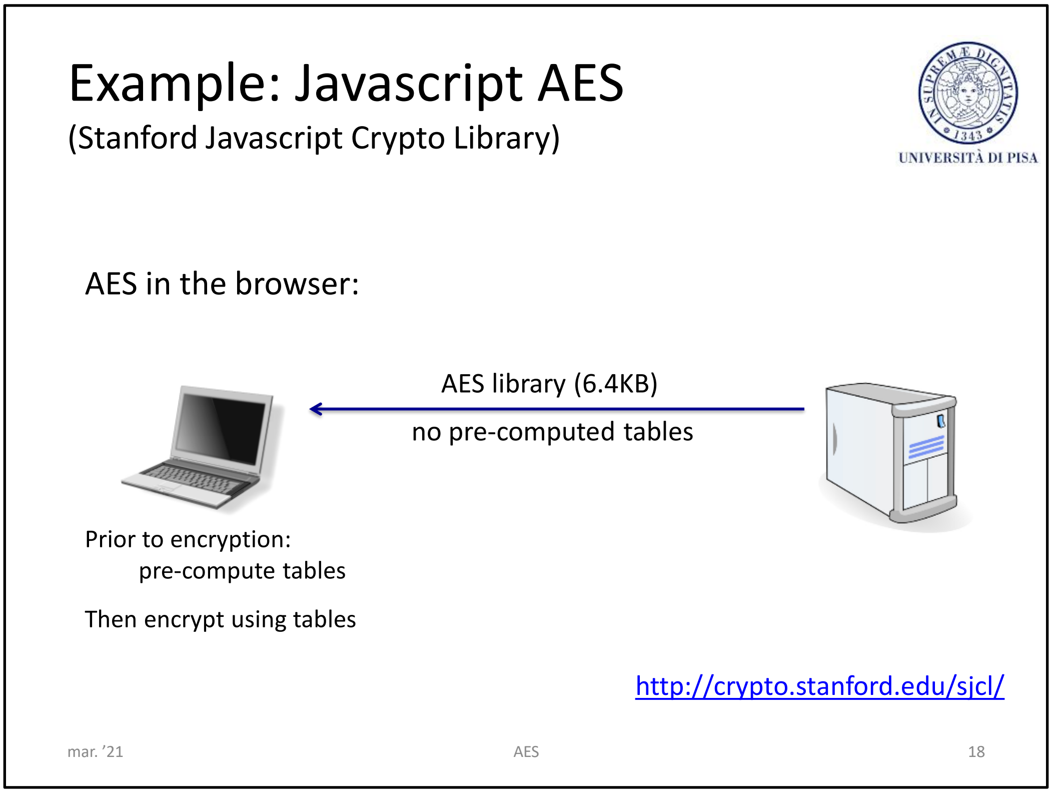
Code size/performance tradeoff

	Code size	Performance
Pre-compute round functions (24KB or 4 KB)	Largest	Fastest (table lookups and xors)
Pre-compute S-box only (256 bytes)	Smaller	Slower
No pre-computation	Smallest	Slowest

- Pre-compute all: for instance on servers
- No pre-computation: for instance 8-bits smart cards of wrist watch

Now, I should point out that, so far, shift rows and mixed columns are very easy to implement in code. And I should say that the substitution itself is also easily computable, so that you can actually write code that takes less than 256 bytes to write. ***And you can kind of shrink the description of AES by literally storing code that computes the table rather than hardwiring the table into your implementation***. And in fact, this is kind of a generic fact about AES, that if you can have allowed no pre computation at all, including computing the S box on the fly. Then in fact you get a fairly small implementation of AES, so it could fit on very constrained environments where there isn't enough room to hold, complicated code. But of course, this will be the slowest implementation, because everything is computed now on the fly, and as a result, the implementation, obviously, is gonna be, slower than things that were pre-computed. And then there is this trade off.

For example if you have a lot of space, and you can support large code, you can actually precompute quite a bit of the three steps that I just mentioned. In fact, there are multiple options of pre-computation, you can build a table that's only four kilobyte big. You can build a table that is even longer, maybe 24 kilobytes. So basically you will have these big tables in your implementation. But then your actual performance is going to be really good, because all your doing is just table look-ups and XORs. You're not doing any other complicated arithmetic. And as a result, if you can do a lot of pre-computation, these three steps here, [inaudible] should. [inaudible] and mixed columns can be converted just into a number, a small number of table lookups and some [inaudible]. All you can do is just compute the S-box, so now your implementation would just have 256 bytes. Hard coded The rest would just be code that's actually computing these three functions. The performance would be slower than in the previous step but the code footprint would also be smaller. So in overall, there's this nice tradeoff between code size and performance. So on high-end machines, on high-end servers, where you can afford to have a lot of code, you can precompute and store these big tables and get the best performance. Whereas on low-end machines like eight byte smart cards or think of like an eight byte wristwatch, you would actually have a relatively small implementation of AES. But as a result of course it won't be so fast.



Example: Javascript AES

(Stanford Javascript Crypto Library)

AES in the browser:

AES library (6.4KB)
no pre-computed tables

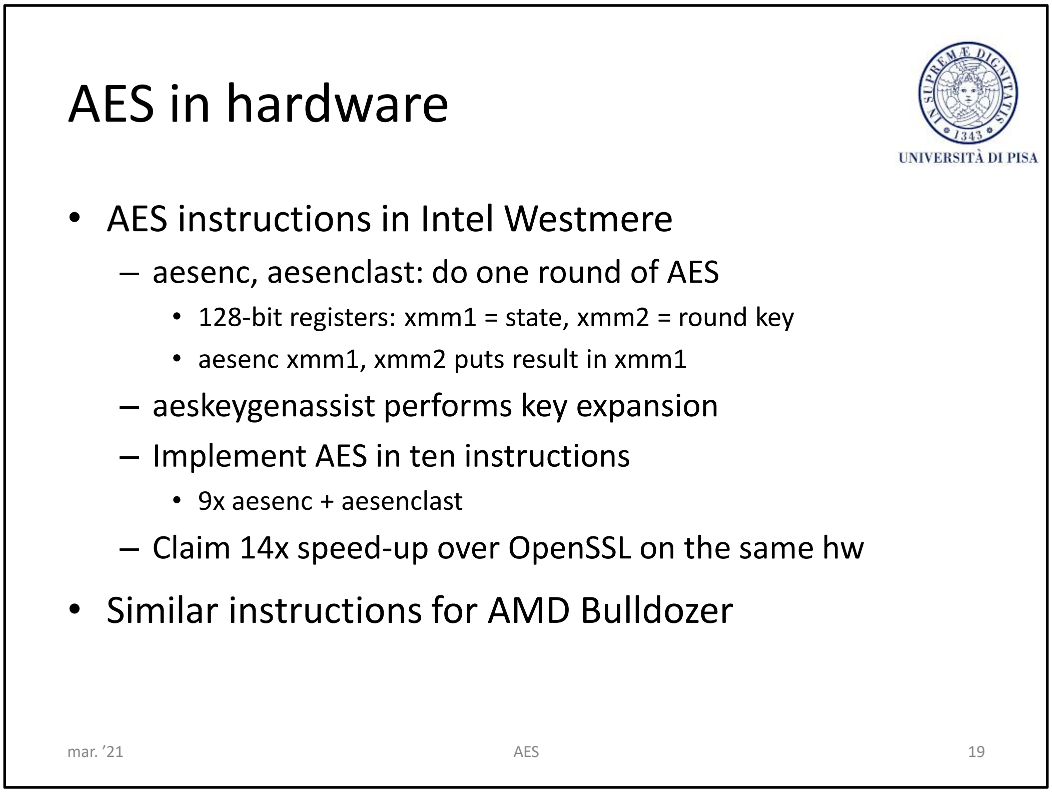
Prior to encryption:
pre-compute tables

Then encrypt using tables

http://crypto.stanford.edu/sjcl/

SJCL: Stanford Javascript Crypto Library

So here's an example that's a little unusual, suppose you wanted to implement AES in Javascript so you can send an AES library to the browser and have the browser actually do AES by itself. So in this case what you'd like to, to is you'd like to both shrink the code size, so that on the network there's minimum traffic to send a library over to the browser but, at the same time, you'd like the browser performance to be as fast as possible. And so this is something that we did a while ago essentially the idea is that the code that actually gets send to the browser doesn't have any pre computed table and as a result is fairly small code. But then the minute it lands on the browser, what the browser will do is actually pre-compute all the tables. So in some sense the code goes from just being small and compact. It gets bloated with all these pre-computed tables. But those are stored on the laptop, which presumably has a lot of memory. And then once you have the pre-computed tables you actually encrypt using them. And that's how you get the best performance. Okay? So if you have to stand in implementation AES over the network, you can kind of get the best of all worlds. Whereas, the code over the network is small, but when it reaches the target client, it can kind of inflate itself. And then get the best performance as it's doing encryption on the clients.



AES in hardware

- AES instructions in Intel Westmere
  - aesenc, aesenclast: do one round of AES
    - 128-bit registers: xmm1 = state, xmm2 = round key
    - aesenc xmm1, xmm2 puts result in xmm1
  - aeskeygenassist performs key expansion
  - Implement AES in ten instructions
    - 9x aesenc + aesenclast
  - Claim 14x speed-up over OpenSSL on the same hw
- Similar instructions for AMD Bulldozer