Complexity and Cryptography

Thomas Zeugmann

Hokkaido University
Laboratory for Algorithmics

https://www-alg.ist.hokudai.ac.jp/~thomas/COCR/

Lecture 3: The Advanced Encryption Standard

Motivation I

As we have seen, there is an unconditionally secure cryptosystem, i.e., the one-time pads. But, as said before, its usability is somehow restricted. All keys must be at least as long as the message to be sent, and the keys must be equally likely. It is not really clear how to achieve this in practice.

Motivation I

As we have seen, there is an unconditionally secure cryptosystem, i.e., the one-time pads. But, as said before, its usability is somehow restricted. All keys must be at least as long as the message to be sent, and the keys must be equally likely. It is not really clear how to achieve this in practice.

There are two main possibilities to recover. One can try to use so-called pseudo-random keys, i.e., only a short true random secret key is exchanged in advance. Furthermore, the parties agree on a method how to generate long keys from this short secret key. These long keys should be practically indistinguishable from true random keys. This leads to stream-ciphers.

Motivation II

Second, one uses keys of a fixed length. Consequently, then one has to use much more complicated encryption and decryption techniques than in the one-time-pad cryptosystem. The advantage is speed. From 1976 to 2001 the Data Encryption Standard (DES) was widely used. But its key length is 56 bits which is nowadays considered to be small.
We call such cryptosystems block ciphers.

Motivation II

Second, one uses keys of a fixed length. Consequently, then one has to use much more complicated encryption and decryption techniques than in the one-time-pad cryptosystem. The advantage is speed. From 1976 to 2001 the Data Encryption Standard (DES) was widely used. But its key length is 56 bits which is nowadays considered to be small.
We call such cryptosystems block ciphers.

Because of the weaknesses of the DES, in 1997 the National Institute of Standards and Technology of the United States called for proposals for a new block cipher that may serve as a replacement of the DES. The selection process was much more open and transparent than for its predecessor the DES. In response to this call, fifteen different designs were submitted and Rijndael won this competition in the years to follow. In 2001 it got adopted as Advanced Encryption Standard (AES).

Motivation III

The call for proposals contained the following demands.

1. The cryptosystem should be a block cipher.
2. It should use a 128-bit block size and keys of length 128-bits, 192-bits, and 256-bits.
3. It should be easy to implement in hardware and software and it should be very fast.
4. It should be usable in smartcards with low resources (short code, low storage requirements).
5. The algorithm should be royalty-free and license-free for any use.
6. Is should resist all known cryptanalytic methods.

Joan Daemon and Vincent Rijmen presented the winning design which they called Rijndael.

Motivation IV

To understand the design of the AES it is helpful to look again into Shannon's (1949) paper *Communication Theory of Secrecy Systems*. In this paper he identified two important criteria that should be fulfilled by every "good" block cipher, i.e., *confusion* and *diffusion*.

Motivation IV

To understand the design of the AES it is helpful to look again into Shannon's (1949) paper *Communication Theory of Secrecy Systems*. In this paper he identified two important criteria that should be fulfilled by every "good" block cipher, i.e., *confusion* and *diffusion*.

1. *Confusion.* The statistical properties of the ciphertext cannot be derived from the statistical properties of the plaintext. That is, a redundant plaintext should be encrypted in a way such the ciphertext looks (almost) truly random.

Motivation IV

To understand the design of the AES it is helpful to look again into Shannon's (1949) paper *Communication Theory of Secrecy Systems*. In this paper he identified two important criteria that should be fulfilled by every "good" block cipher, i.e., *confusion* and *diffusion*.

1. *Confusion.* The statistical properties of the ciphertext cannot be derived from the statistical properties of the plaintext. That is, a redundant plaintext should be encrypted in a way such the ciphertext looks (almost) truly random.
2. *Diffusion.* Each bit of the plaintext and each bit of the key should influence "many" bits of the cipher. Ideally, if one bit of the plaintext or key is changed then the ciphertext should change completely, in an unpredictable or pseudorandom manner.

Motivation V

More recently, the demands on diffusion were concretized as follows.

Avalanche Criterion. Changing one input bit should change at least half of the output bits. In its strict form, the *avalanche criterion* demands that changing one input bit changes each bit in the cipher with probability 1/2.
Furthermore, in addition to the two criteria given by Shannon (1949), the state of the art also requires *non-linearity* defined as follows.

Non-Linearity. None of the bits in the cipher depends linearly on a single input bit.

Mathematical Background I

The basic unit for computations in the AES is the byte. Thus, many operations are defined in $\mathbb{F}_{2^8}$, i.e., the Galois field having 2^8 many elements. To represent a byte we either use binary numbers, hexadecimal numbers, or the polynomial representation of the elements of $\mathbb{F}_{2^8}$.

Mathematical Background I

The basic unit for computations in the AES is the byte. Thus, many operations are defined in $\mathbb{F}_{2^8}$, i.e., the Galois field having 2^8 many elements. To represent a byte we either use binary numbers, hexadecimal numbers, or the polynomial representation of the elements of $\mathbb{F}_{2^8}$.
The AES uses the *irreducible polynomial*

$$m(x) \quad = \quad x^8 + x^4 + x^3 + x + 1\,. \tag{1}$$

This polynomial m is irreducible, since
$m(0) = m(1) = 1 \bmod 2$.

Mathematical Background I

The basic unit for computations in the AES is the byte. Thus, many operations are defined in $\mathbb{F}_{2^8}$, i.e., the Galois field having 2^8 many elements. To represent a byte we either use binary numbers, hexadecimal numbers, or the polynomial representation of the elements of $\mathbb{F}_{2^8}$.
The AES uses the *irreducible polynomial*

$$m(x) \quad = \quad x^8 + x^4 + x^3 + x + 1\,. \tag{1}$$

This polynomial m is irreducible, since
$m(0) = m(1) = 1 \bmod 2$.

The advantage is that addition in $\mathbb{F}_{2^8}$ corresponds to take the bitwise *EX-OR* of the coefficients. Multiplication in $\mathbb{F}_{2^8}$ is then just polynomial multiplication modulo $m(x)$, where the coefficients are taken modulo 2.

Mathematical Background II

For example, adding $x^7 + x^6 + x^2 + x + 1$ and $x^6 + x^5 + x^3 + x$ which gives $x^7 + x^5 + x^3 + x^2 + 1$ is performed as

$$11000111 \oplus 01101010 \quad = \quad 10101101\,. \tag{2}$$

Mathematical Background II

For example, adding $x^7 + x^6 + x^2 + x + 1$ and $x^6 + x^5 + x^3 + x$ which gives $x^7 + x^5 + x^3 + x^2 + 1$ is performed as

$$11000111 \oplus 01101010 \quad = \quad 10101101\,. \tag{2}$$

Multiplying $(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1)$ modulo $m(x)$ results in $x^7 + x^6 + 1$.

Mathematical Background II

For example, adding $x^7 + x^6 + x^2 + x + 1$ and $x^6 + x^5 + x^3 + x$ which gives $x^7 + x^5 + x^3 + x^2 + 1$ is performed as

$$11000111 \oplus 01101010 \quad = \quad 10101101\,. \tag{2}$$

Multiplying $(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1)$ modulo $m(x)$ results in $x^7 + x^6 + 1$.

Note that this multiplication can be precomputed for all values that may appear during the execution of the AES, and may be thus realized by simple table look ups.

Mathematical Background III

Furthermore, the multiplicative inverse of an element in $\mathbb{F}_{2^8}$ can be computed by using a straightforward modification of our Algorithm ECL. That is, for $p \in \mathbb{F}_{2^8}$, now we have to compute a polynomial p^{-1} such that $p(x) \cdot p^{-1}(x) = 1$.

Mathematical Background III

Furthermore, the multiplicative inverse of an element in $\mathbb{F}_{2^8}$ can be computed by using a straightforward modification of our Algorithm ECL. That is, for $p \in \mathbb{F}_{2^8}$, now we have to compute a polynomial p^{-1} such that $p(x) \cdot p^{-1}(x) = 1$.

In dependence on the operation to be performed, sometimes one has to convert a given representation of a byte into another one.
If $b_7 b_6 \cdots b_0$ the representation of a byte in binary notation, then

$$b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0 \qquad (3)$$

is the representation of the same byte as a polynomial (i.e., an element of $\mathbb{F}_{2^8}$).

Mathematical Background IV

To convert the binary notation into hexadecimal notation, the following table is used for $b_7b_6b_5b_4$ and $b_3b_2b_1b_0$, respectively.

0000_2	$=$	0_{16}	1000_2	$=$	8_{16}
0001_2	$=$	1_{16}	1001_2	$=$	9_{16}
0010_2	$=$	2_{16}	1010_2	$=$	A_{16}
0011_2	$=$	3_{16}	1011_2	$=$	B_{16}
0100_2	$=$	4_{16}	1100_2	$=$	C_{16}
0101_2	$=$	5_{16}	1101_2	$=$	D_{16}
0110_2	$=$	6_{16}	1110_2	$=$	E_{16}
0111_2	$=$	7_{16}	1111_2	$=$	F_{16}

Mathematical Background IV

To convert the binary notation into hexadecimal notation, the following table is used for $b_7b_6b_5b_4$ and $b_3b_2b_1b_0$, respectively.

0000_2	$=$	0_{16}	1000_2	$=$	8_{16}
0001_2	$=$	1_{16}	1001_2	$=$	9_{16}
0010_2	$=$	2_{16}	1010_2	$=$	A_{16}
0011_2	$=$	3_{16}	1011_2	$=$	B_{16}
0100_2	$=$	4_{16}	1100_2	$=$	C_{16}
0101_2	$=$	5_{16}	1101_2	$=$	D_{16}
0110_2	$=$	6_{16}	1110_2	$=$	E_{16}
0111_2	$=$	7_{16}	1111_2	$=$	F_{16}

Besides the elements of $\mathbb{F}_{2^8}$ the AES uses also polynomials of degree 3 to realize certain operations. Note that these polynomials do *not* form a subfield of $\mathbb{F}_{2^8}$.

Mathematical Background V

Let $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$. For such a polynomial we sometimes write $[a_0, a_1, a_2, a_3]$.
Let $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ and
$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$ be such polynomials. We define addition as follows.

$$a(x) + b(x) \quad = \quad (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0)\,.$$

The multiplication is defined via polynomial multiplication modulo $x^4 + 1$.

Mathematical Background VI

Recall that $x^i \equiv x^{i \bmod 4} \bmod (x^4 + 1)$. Thus, the multiplication of these polynomials can be simplified as follows. Let $d(x) = d_3x^3 + d_2x^2 + d_1x + d_0 = a(x) \otimes b(x)$; then we have

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} a_0 a_3 a_2 a_1 \\ a_1 a_0 a_3 a_2 \\ a_2 a_1 a_0 a_3 \\ a_3 a_2 a_1 a_0 \end{pmatrix} \cdot \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} .$$

Note that this multiplication is not always invertible, since the inverse of the polynomial b may not be of degree 3.

Mathematical Background VI

Recall that $x^i \equiv x^{i \bmod 4} \bmod (x^4 + 1)$. Thus, the multiplication of these polynomials can be simplified as follows. Let $d(x) = d_3x^3 + d_2x^2 + d_1x + d_0 = a(x) \otimes b(x)$; then we have

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} a_0 a_3 a_2 a_1 \\ a_1 a_0 a_3 a_2 \\ a_2 a_1 a_0 a_3 \\ a_3 a_2 a_1 a_0 \end{pmatrix} \cdot \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} .$$

Note that this multiplication is not always invertible, since the inverse of the polynomial b may not be of degree 3.
However, the AES uses here the multiplication with a fixed polynomial c which has an inverse of degree 3, i.e., c is fixed as

$$\begin{aligned} c(x) &= 03_{16}x^3 + 01_{16}x^2 + 01_{16}x + 02_{16} \\ c(x)^{-1} &= 0B_{16}x^3 + 0D_{16}x^2 + 09_{16}x + 0E_{16} . \end{aligned}$$

Mathematical Background VI

Note that the numbers in the definition of the polynomial c are to be read as two half bytes, i.e., 03_{16} means that first four bits are 0000 and the second four bits are 0011.
Note that the polynomial c has the multiplicative inverse given above, since $x^4 + 1$ and $c(x)$ have the gcd 1.

Mathematical Background VI

Note that the numbers in the definition of the polynomial c are to be read as two half bytes, i.e., 03_{16} means that first four bits are 0000 and the second four bits are 0011.
Note that the polynomial c has the multiplicative inverse given above, since $x^4 + 1$ and $c(x)$ have the gcd 1.

Now we are ready to describe the AES.

The AES is based on a design principle known as a *substitution permutation network* (abbr. SPN). An SPN takes a block of the plaintext and the key as inputs, and applies several alternating "rounds" or "layers" of substitution boxes (S-boxes) and permutation boxes (P-boxes) to produce the ciphertext block.

The AES I

We start with a high-level description. The input is a (plaintext) block of 128-bits and a key. Possible keys length are 128 bits, 192 bits or 256 bits. Then the AES proceeds in rounds, where the number of rounds depends on the key length (10 rounds, 12 rounds, 14 rounds, respectively).

1. KeyExpansion – the round keys are derived from the input key.
2. Initial Round – **AddRoundKey()**.
3. Rounds – **SubBytes()**, **ShiftRows()**, **MixColumns()**, **AddRoundKey()** (* in this order *).
4. Final Round – **SubBytes()**, **ShiftRows()**, **AddRoundKey()**.
5. Output the cipher

Note that all rounds in (3) do essentially the same. So, it remains to explain what the different functions compute.

The AES II

Recall that the basis unit in the AES is the byte. Let $b_7, \ldots, b_0$ be the stored bits. Then we use $(b_7, \ldots, b_0)_2$ if we mean a binary number and we write $b_7x^7 + \cdots + b_1x + b_0$ when dealing with elements of $\mathbb{F}_{2^8}$. If x is a block of n bytes then we write $x\{i\}$ to denote the ith byte of the block x.

The AES II

Recall that the basis unit in the AES is the byte. Let $b_7, \ldots, b_0$ be the stored bits. Then we use $(b_7, \ldots, b_0)_2$ if we mean a binary number and we write $b_7x^7 + \cdots + b_1x + b_0$ when dealing with elements of $\mathbb{F}_{2^8}$. If x is a block of n bytes then we write $x\{i\}$ to denote the ith byte of the block x.

The AES operates on states which are defined as follows. The plaintext and the intermediate results are written as a 4×4-matrix, whose entries are bytes. Thus we describe the states as follows.

input					state					output			
in_0	in_4	in_8	in_{12}		$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$		out_0	out_4	out_8	out_{12}
in_1	in_5	in_9	in_{13}	$\rightarrow$	$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$\rightarrow$	out_1	out_5	out_9	out_{13}
in_2	in_6	in_{10}	in_{14}		$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$		out_2	out_6	out_{10}	out_{14}
in_3	in_7	in_{11}	in_{15}		$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$		out_3	out_7	out_{11}	out_{15}

The AES III

To convert a 1-dimensional array of 16 bytes in a 2-dimensional 4×4-matrix the function **Square**: $(\{0,1\}^8)^{16} \to (\{0,1\}^8)^{4\times 4}$ is used which is defined as follows. Let

$$s \quad = \quad (s_{i,j})_{i,j\in\{0,\ldots,3\}} \quad = \quad \textbf{Square}(x) \qquad \text{for } x \in (\{0,1\}^8)^{16} \ ,$$

then $\quad s_{i,j} = x\{i + 4 \cdot j\} \quad$ for all $i, j \in \{0, \ldots, 3\}$.

The AES III

To convert a 1-dimensional array of 16 bytes in a 2-dimensional 4×4-matrix the function **Square**: $(\{0,1\}^8)^{16} \to (\{0,1\}^8)^{4\times 4}$ is used which is defined as follows. Let

$$s \quad = \quad (s_{i,j})_{i,j\in\{0,\ldots,3\}} \quad = \quad \textbf{Square}(x) \qquad \text{for } x \in (\{0,1\}^8)^{16} \ ,$$

then $\quad s_{i,j} = x\{i + 4 \cdot j\} \quad$ for all $i, j \in \{0, \ldots, 3\}$.

The function **InvSquare**: $(\{0,1\}^8)^{4\times 4} \to (\{0,1\}^8)^{16}$ computes the inverse of **Square**, i.e., for

$$x \quad = \quad \textbf{InvSquare}(s)$$

we have $\quad x\{i + 4 \cdot j\} = s_{i,j} \quad$ for all $i, j \in \{0, \ldots, 3\}$.

The AES IV

Let $s = (s_{i,j})_{i,j\in\{0,\ldots,3\}}$ and $s' = (s'_{i,j})_{i,j\in\{0,\ldots,3\}}$, then we define

$$s \oplus s' \quad = \quad (s_{i,j} \oplus s'_{i,j})_{i,j\in\{0,\ldots,3\}}\,, \tag{4}$$

i.e., as element-wise addition in $\mathbb{F}_{2^8}$.

The AES IV

Let $s = (s_{i,j})_{i,j\in\{0,\ldots,3\}}$ and $s' = (s'_{i,j})_{i,j\in\{0,\ldots,3\}}$, then we define

$$s \oplus s' \quad = \quad (s_{i,j} \oplus s'_{i,j})_{i,j\in\{0,\ldots,3\}}\,, \tag{4}$$

i.e., as element-wise addition in $\mathbb{F}_{2^8}$.

Next, we describe the functions already mentioned in the high-level description and their inverses.
We start with **AddRoundKey**() and **InvAddRoundKey**() which are identical.

Input: 4×4-matrix s and a 16-byte round key K_i.
Output: the 4×4-matrix $s \oplus$ **Square**(K_i).

The AES V

How the 16-byte round key K_i is computed will be explained later. Note that the 16-bytes correspond to the size of a state.

If the key length is 128 bit, then in the initial round, the function **AddRoundKey**() uses the main key, i.e., the key provided as input.

The AES VI

Next, we consider the function **SubBytes**(), which is the substitution mapping of the AES. It is the only *non-linear* part of the AES and guarantees the necessary confusion.

Input: 4×4-matrix s.
Output: the 4×4-matrix $s' = (s'_{i,j})_{i,j \in \{0,\dots,3\}}$, where for $s_{i,j} = uv$ with $u, v \in \{0,1\}^4$ we define

$$s'_{i,j} = \text{the entry in the } u\text{th-row and } v\text{th-column from the S-Box table shown below .}$$

To understand the table below, recall that each matrix entry is a byte. So u and v are the first and last 4 bits of a byte, respectively. Thus, u and v can be regarded as binary numbers in the range from 0 to 15, which we write in hexadecimal notation below.

The S-Box Table

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

The S-Box Table

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

The function **InvSubBytes**() is essentially the same with the only difference that now the Inverse S-Box table is used.

The Inverse S-Box Table

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

The AES VII

The entries in the S-Box table correspond to the following operations. Let $s_{i,j}$ be a byte for which we like to compute the corresponding S-Box entry. Then

1. Compute the *inverse* $s_{i,j}^{-1}$ of $s_{i,j}$ in $\mathbb{F}_{2^8}$ ($*$ if $s_{i,j} = 0$ then we set $s_{i,j}^{-1} = 0$ $*$).
2. The new byte is transformed by the following affine transformation $s'_{i,j} = M \cdot s_{i,j}^{-1} \oplus b$.

It is known that the *inverse* is highly *non-linear*.

Here M and b are defined as follows by the AES. Note that we have to write the bytes as column vectors.

The AES VIII

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \qquad b = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Note that M is invertible.

The AES IX

We continue with **ShiftRows**() and **InvShiftRows**().
The function **ShiftRows**() executes a cyclic shift on the rows of the state matrix. If the block length is 128 bit then row i is shifted i positions. That is, formally we have:

Input: 4×4-matrix s.
Output: the 4×4-matrix s', where $s'_{i,j} = s_{i,(j+i) \mod 4}$.

The AES IX

We continue with **ShiftRows**() and **InvShiftRows**().
The function **ShiftRows**() executes a cyclic shift on the rows of the state matrix. If the block length is 128 bit then row i is shifted i positions. That is, formally we have:

Input: 4×4-matrix s.
Output: the 4×4-matrix s', where $s'_{i,j} = s_{i,(j+i) \mod 4}$.

For **InvShiftRows**() we then clearly have:

Input: 4×4-matrix s.
Output: the 4×4-matrix s', where $s'_{i,(j+i) \mod 4} = s_{i,j}$.

The AES X

Next, let us look at **MixColumns**() and **InvMixColumns**(). The function **MixColumns**() provides together with **ShiftRows** the *diffusion* of the cipher. As its name says, it operates on the columns of the state matrix.

MixColumns() interprets every column of the state matrix as a polynomial of degree 3 with coefficients from $\mathbb{F}_{2^8}$, i.e.,

$$s_j = s_{3,j}x^3 + s_{2,j}x^2 + s_{1,j}x + s_{0,j} \ . \tag{5}$$

Each of these polynomials is multiplied ($\otimes$) by the fixed polynomial $c(x)$ and the result is taken modulo $(x^4 + 1)$. Recall that

$$c(x) = 03_{16}x^3 + 01_{16}x^2 + 01_{16}x + 02_{16} \ .$$

The AES XI

So, we obtain the following.

Input: 4×4-matrix s.
Output: the 4×4-matrix s' with

$$\begin{pmatrix} s'_{0,j} \\ s'_{1,j} \\ s'_{2,j} \\ s'_{3,j} \end{pmatrix} = \begin{pmatrix} 02_{16} & 03_{16} & 01_{16} & 01_{16} \\ 01_{16} & 02_{16} & 03_{16} & 01_{16} \\ 01_{16} & 01_{16} & 02_{16} & 03_{16} \\ 03_{16} & 01_{16} & 01_{16} & 02_{16} \end{pmatrix} \cdot \begin{pmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{pmatrix}$$

InvMixColumns() is obtained analogously, where one has to use the polynomial $c(x)^{-1} = 0B_{16}x^3 + 0D_{16}x^2 + 09_{16}x + 0E_{16}$ instead of c.

The AES XII

The input key is called the *main key*. For each round, a subkey is derived from the main key. The subkey has the same size as the state. It remains to explain how the round keys are computed. In order to do so, we need several auxillary functions that act on a 4-byte word $w = w\{0\}w\{1\}w\{2\}w\{3\}$.

The AES XII

The input key is called the *main key*. For each round, a subkey is derived from the main key. The subkey has the same size as the state. It remains to explain how the round keys are computed. In order to do so, we need several auxillary functions that act on a 4-byte word $w = w\{0\}w\{1\}w\{2\}w\{3\}$.

RotWord()

- takes a word $w\{0\}w\{1\}w\{2\}w\{3\}$ as input, performs a cyclic permutation, and returns the word $w\{1\}w\{2\}w\{3\}w\{0\}$.

The AES XII

The input key is called the *main key*. For each round, a subkey is derived from the main key. The subkey has the same size as the state. It remains to explain how the round keys are computed. In order to do so, we need several auxillary functions that act on a 4-byte word $w = w\{0\}w\{1\}w\{2\}w\{3\}$.

RotWord()

- takes a word $w\{0\}w\{1\}w\{2\}w\{3\}$ as input, performs a cyclic permutation, and returns the word $w\{1\}w\{2\}w\{3\}w\{0\}$.

SubWord()

- is a function that takes a four-byte input word $w\{0\}w\{1\}w\{2\}w\{3\}$ and applies the S-box to each of the four bytes to produce an output word (* in the same way as described for **SubBytes**() *). For example, $6C_{16}76_{16}05_{16}2A_{16}$ is transformed into $50_{16}38_{16}6B_{16}E5_{16}$.

The AES XIII

Rcon(i)

- for $i \in \mathbb{N}$ with $i > 0$ we define

$$\textbf{Rcon}(i) \quad = \quad c_i 00_{16} 00_{16} 00_{16} \, ,$$

where the *round constant* c_i is determined by the operations in $\mathbb{F}_{2^8}$ and the polynomial representation to be

$$c_i \quad = \quad x^{i-1} \mod m(x) \, .$$

For example, $c_2 = x$ and thus we get the byte 01_{16}.

The complete key expansion algorithm is shown below.

Algorithm KeyExpansion(K)

Input: $4 \cdot k$-byte key K (* the main key *)
Output: The sequence $(K_0, \ldots, K_\ell)$ of round keys

1: $\ell := 6 + k$
2: **for** $i = 0$ to $k - 1$ **do** $h_i := K\{4i\}K\{4i+1\}K\{4i+2\}K\{4i+3\}$ **end for**
3: **for** $i = k$ to $4 \cdot \ell$ **do**
4: $\quad t := h_{i-1}$
5: $\quad$ **if** $(i \bmod k) = 0$ **then**
6: $\quad\quad t :=$ **SubWord**(**RotWord**$(t)) \oplus$ **Rcon**(i/k)
7: $\quad$ **else**
8: $\quad\quad$ **if** $k = 8 \wedge (i \bmod k) = 4$ **then** $t :=$ **SubWord**(t) **end if**
9: $\quad$ **end if**
10: $\quad h_i := h_{i-k} \oplus t$
11: **end for**
12: **for** $i = 0$ to ℓ **do** $K_i := h_{4i}h_{4i+1}h_{4i+2}h_{4i+3}$ **end for**
13: Return $(K_0, \ldots, K_\ell)$

Note that k is determined by the length of the main key.

Remarks I

1. In the initial round, we only add the main key to the plaintext. Since the AES is known, also the three functions **SubBytes**(), **ShiftRows**(), and **MixColumns**() are known and a cryptanalyst could compute their values without knowing the key. So, if we add the key in the initial round then a cryptanalyst does not know the input to the first round.
2. The same reasoning applies to the last round. It can be shown that changing the order of **MixColumns**() and **AddRoundKey**() does not change the result. So, the last security relevant operation is adding the key.
3. The deciphering is just done in reverse order and by using the inverse functions.

Remarks II

1. The AES has a very transparent algebraic design and it seems to be safe againt all known attacks to block ciphers. In particular, it seems to be safe against differential and linear cryptanalytic methods. This safety is caused by order in which the single mappings are applied.
2. On the other hand, there is a debate in the literature whether or not the clear, and in some sense simple algebraic design, is also a weak point of the AES. Clearly, in order to prove that the AES is safe against a particular type of attack, it is an advantage to have a clear mathematical description of the cipher. But it may also ease potential attacks.
   Some attacks were presented at Crypto 2009, Asiacrypt 2009, and Eurocrypt 2010.

Thank you!

$\mathbb{F}_{3^2}$ and its Multiplication Table

We need a polynomial f of degree 2 which is irreducible over $\mathbb{Z}_3$. Here we take $f(x) = x^2 - x + 2$. Then we can represent the elements of $\mathbb{F}_{3^2}$ and its multiplication table as follows.

$\cdot$	1	2	ϑ	2ϑ	$\vartheta+1$	$\vartheta+2$	$2\vartheta+1$	$2\vartheta+2$
1	1	2	ϑ	2ϑ	$\vartheta+1$	$\vartheta+2$	$2\vartheta+1$	$2\vartheta+2$
2	2	1	2ϑ	ϑ	$2\vartheta+2$	$2\vartheta+1$	$\vartheta+2$	$\vartheta+1$
ϑ	ϑ	2ϑ	$\vartheta+1$	$2\vartheta+2$	$2\vartheta+1$	1	2	$\vartheta+2$
2ϑ	2ϑ	ϑ	$2\vartheta+2$	$\vartheta+1$	$\vartheta+2$	2	1	$2\vartheta+1$
$\vartheta+1$	$\vartheta+1$	$2\vartheta+2$	$2\vartheta+1$	$\vartheta+2$	2	ϑ	2ϑ	1
$\vartheta+2$	$\vartheta+2$	$2\vartheta+1$	1	2	ϑ	$2\vartheta+2$	$\vartheta+1$	2ϑ
$2\vartheta+1$	$2\vartheta+1$	$\vartheta+2$	2	1	2ϑ	$\vartheta+1$	$2\vartheta+2$	ϑ
$2\vartheta+2$	$2\vartheta+2$	$\vartheta+1$	$\vartheta+2$	$2\vartheta+1$	1	2ϑ	ϑ	2