Cryptography: Advanced Encryption Standard (AES)

Alex Corner

Sheffield Hallam University

The Story So Far

- In the 1970s the Data Encryption Standard is introduced.

The Story So Far

- In the 1970s the Data Encryption Standard is introduced.
- It is quickly deemed insecure, but is still in use up to 2023 in the form of 3DES (Triple-DES).

The Story So Far

- In the 1970s the Data Encryption Standard is introduced.
- It is quickly deemed insecure, but is still in use up to 2023 in the form of 3DES (Triple-DES).
- Other developments are happening at the time (public key cryptography), but we'll visit these later.

Development of AES

- Triple-DES was supposed to be a stopgap.

Development of AES

- Triple-DES was supposed to be a stopgap.
- In 1998 the National Institute of Standards and Technology held an open call for a new standard to replace DES.

Development of AES

- Triple-DES was supposed to be a stopgap.
- In 1998 the National Institute of Standards and Technology held an open call for a new standard to replace DES.
- This was to be called **AES: A**dvanced **E**ncryption **S**tandard.

Development of AES

- Triple-DES was supposed to be a stopgap.
- In 1998 the National Institute of Standards and Technology held an open call for a new standard to replace DES.
- This was to be called **AES: A**dvanced **E**ncryption **S**tandard.
- Many of the world's top cryptographers submitted entries to this competition.

Development of AES

- One requirement was the use of variable key sizes: 128, 192, and 256 bits.

Development of AES

- One requirement was the use of variable key sizes: 128, 192, and 256 bits.
- It should work on different hardware: PCs, smart cards, etc.

Development of AES

- One requirement was the use of variable key sizes: 128, 192, and 256 bits.
- It should work on different hardware: PCs, smart cards, etc.
- Speed was also a criterion.

Development of AES

- One requirement was the use of variable key sizes: 128, 192, and 256 bits.
- It should work on different hardware: PCs, smart cards, etc.
- Speed was also a criterion.
- Fifteen candidates were considered and a shortlist of five were chosen as finalists:
  - MARS,
  - RC6,
  - Twofish,
  - Serpent,
  - **Rijndael**: the winner.

Development of AES

- One requirement was the use of variable key sizes: 128, 192, and 256 bits.
- It should work on different hardware: PCs, smart cards, etc.
- Speed was also a criterion.
- Fifteen candidates were considered and a shortlist of five were chosen as finalists:
  - MARS,
  - RC6,
  - Twofish,
  - Serpent,
  - **Rijndael**: the winner.
- All algorithms were up to three times faster than DES.

The Winner: Rijndael

- Designed by Belgian cryptographs Joan Daemen and Vincent Rijmen.

The Winner: Rijndael

- Designed by Belgian cryptographs Joan Daemen and Vincent Rijmen.
- Their algorithm became the official standard in 2002.

The Winner: Rijndael

- Designed by Belgian cryptographs Joan Daemen and Vincent Rijmen.
- Their algorithm became the official standard in 2002.
- It is fast, easy to implement, and requires little memory.

The Winner: Rijndael

- Designed by Belgian cryptographs Joan Daemen and Vincent Rijmen.
- Their algorithm became the official standard in 2002.
- It is fast, easy to implement, and requires little memory.
- **Not** a Feistel system like DES, as everything is considered as one block: no splitting.

The Winner: Rijndael

- Designed by Belgian cryptographs Joan Daemen and Vincent Rijmen.
- Their algorithm became the official standard in 2002.
- It is fast, easy to implement, and requires little memory.
- **Not** a Feistel system like DES, as everything is considered as one block: no splitting.
- It is a substitution-permutation system.

Development of AES

- Like DES it is a block cipher.

Development of AES

- Like DES it is a block cipher.
- Keys are of length 128, 192, or 256 bits, compared to DES's 56-bit keys.

Development of AES

- Like DES it is a block cipher.
- Keys are of length 128, 192, or 256 bits, compared to DES's 56-bit keys.
- For the 128-bit version this means that there are

  $$2^{128} \approx 3.4 \times 10^{38}$$

  possible keys: much more than even Triple DES.

Development of AES

- Like DES it is a block cipher.
- Keys are of length 128, 192, or 256 bits, compared to DES's 56-bit keys.
- For the 128-bit version this means that there are

  $$2^{128} \approx 3.4 \times 10^{38}$$

  possible keys: much more than even Triple DES.
- Each block is 16 bytes (128 bits) arranged as an array of 4 rows and 4 columns.

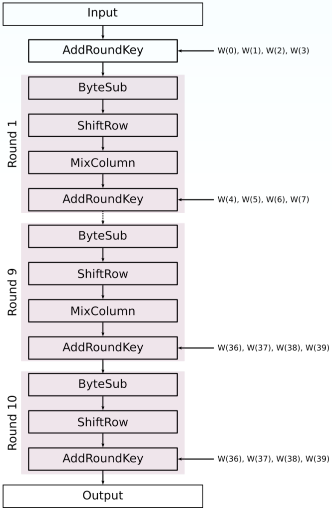
Development of AES

- Like DES it is a block cipher.
- Keys are of length 128, 192, or 256 bits, compared to DES's 56-bit keys.
- For the 128-bit version this means that there are

  $$2^{128} \approx 3.4 \times 10^{38}$$

  possible keys: much more than even Triple DES.
- Each block is 16 bytes (128 bits) arranged as an array of 4 rows and 4 columns.
- AES uses 10 rounds (128-bit key), 12 rounds (192-bit key), or 14 rounds (256-bit key) using round keys based on the original key, plus a 0th round using the original key.

AES Scheme: 128 Bits



The Input

AES-128 is a block cipher, using blocks of 128 bits made up of 16 bytes. These are arranged in an array with 4 rows and 4 columns. E.g., if the plaintext is `to be or not to be that is the question...`, then the array is represented in hex as

$$\begin{bmatrix} t & o & t & e \\ o & r & t & t \\ b & n & o & h \\ e & o & b & a \end{bmatrix} = \begin{bmatrix} 74 & 6F & 74 & 65 \\ 6F & 72 & 74 & 74 \\ 62 & 6E & 6E & 68 \\ 65 & 6F & 62 & 61 \end{bmatrix}.$$

(Recall that each lower case letter is represented by an 8-bit binary number in ASCII. E.g., `t` $= 0111\,0100_2 = 74_{16}$.)

AddRoundKey

The round keys are derived from the original keys (more on that later). At this stage we perform a bitwise `XOR` with the plaintext, before beginning the first round. The 0th round key is the original key, where the columns are denoted by $W(0)$, $W(1)$, $W(2)$, and $W(3)$, respectively.

AddRoundKey

The round keys are derived from the original keys (more on that later). At this stage we perform a bitwise `XOR` with the plaintext, before beginning the first round. The 0th round key is the original key, where the columns are denoted by $W(0)$, $W(1)$, $W(2)$, and $W(3)$, respectively.

$$K \oplus P = \begin{bmatrix} 0F & 47 & 0C & AF \\ 15 & D9 & B7 & 7F \\ 71 & E8 & AD & 67 \\ C9 & 59 & D6 & 98 \end{bmatrix} \oplus \begin{bmatrix} 74 & 6F & 74 & 65 \\ 6F & 72 & 74 & 74 \\ 62 & 6E & 6E & 68 \\ 65 & 6F & 62 & 61 \end{bmatrix} = \begin{bmatrix} 7B & 28 & 78 & CA \\ 7A & AB & C3 & 0B \\ 13 & 86 & C3 & 0F \\ AC & 36 & B4 & F9 \end{bmatrix}.$$

ByteSub

In the ByteSub stage we replace each entry in the array using the Rijndael S-Box. We determine the row number using the first hex digit (first byte) and the column by the second hex digit (second byte).

$$\begin{bmatrix} 7B & 28 & 78 & CA \\ 7A & AB & C3 & 0B \\ 13 & 86 & C3 & 0F \\ AC & 36 & B4 & F9 \end{bmatrix}$$

E.g., the first entry is $7B$, which corresponds to the 7th row and column B, the 11th column.

Rijndael S-Box

	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

Rijndael S-Box

	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

ByteSub

In the ByteSub stage we replace each entry in the array using the Rijndael S-Box. We determine the row number using the first hex digit (first byte) and the column by the second hex digit (second byte).

$$\begin{bmatrix} 7B & 28 & 78 & CA \\ 7A & AB & C3 & 0B \\ 13 & 86 & C3 & 0F \\ AC & 36 & B4 & F9 \end{bmatrix}$$

ByteSub

In the ByteSub stage we replace each entry in the array using the Rijndael S-Box. We determine the row number using the first hex digit (first byte) and the column by the second hex digit (second byte).

$$\begin{bmatrix} 7B & 28 & 78 & CA \\ 7A & AB & C3 & 0B \\ 13 & 86 & C3 & 0F \\ AC & 36 & B4 & F9 \end{bmatrix}$$

E.g., the first entry is $7B$, which corresponds to the 7th row and column B, the 11th column. This gives 21.

ByteSub

In the ByteSub stage we replace each entry in the array using the Rijndael S-Box. We determine the row number using the first hex digit (first byte) and the column by the second hex digit (second byte).

$$\begin{bmatrix} 7B & 28 & 78 & CA \\ 7A & AB & C3 & 0B \\ 13 & 86 & C3 & 0F \\ AC & 36 & B4 & F9 \end{bmatrix}$$

E.g., the first entry is $7B$, which corresponds to the 7th row and column B, the 11th column. This gives 21. Doing this for each entry gives the following array.

$$\begin{bmatrix} 21 & 34 & BC & 74 \\ DA & 62 & 2E & 2B \\ 7D & 44 & 2E & 76 \\ 91 & 05 & 8D & 99 \end{bmatrix}$$

ShiftRow

The ShiftRow stage is quite simple. Each row is shifted by a certain amount:

- The first row doesn't shift.
- The second row shifts one place to the left.
- The third row shifts two places to the left.
- The fourth row shifts three places to the left.

ShiftRow

The ShiftRow stage is quite simple. Each row is shifted by a certain amount:

- The first row doesn't shift.
- The second row shifts one place to the left.
- The third row shifts two places to the left.
- The fourth row shifts three places to the left.

$$\begin{bmatrix} 21 & 34 & BC & 74 \\ DA & 62 & 2E & 2B \\ 7D & 44 & 2E & 76 \\ 91 & 05 & 8D & 99 \end{bmatrix} \mapsto \begin{bmatrix} 21 & 34 & BC & 74 \\ 62 & 2E & 2B & DA \\ 2E & 76 & 7D & 44 \\ 99 & 91 & 05 & 8D \end{bmatrix}$$

MixColumn

We won't fully investigate what happens in the MixColum stage, as the mathematics gets rather complicated. It depends on an area of mathematics called Galois Fields. In particular, the Galois field $GF(2^8)$. In practice, this is achieved by shifting bits and applying `XOR`, but the mathematical underpinnings ensure that the scheme works out nicely.

MixColumn

We won't fully investigate what happens in the MixColum stage, as the mathematics gets rather complicated. It depends on an area of mathematics called Galois Fields. In particular, the Galois field $GF(2^8)$. In practice, this is achieved by shifting bits and applying `XOR`, but the mathematical underpinnings ensure that the scheme works out nicely.

Each column of the array is replaced by a combination of the bytes in the column, where $\oplus$ means the usual bitwise `XOR`, but $2a$, $3a$, $2b$, etc. are **not** by simple multiplication.

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \mapsto \begin{bmatrix} 2a \oplus 3b \oplus c \oplus d \\ a \oplus 2b \oplus 3c \oplus d \\ a \oplus b \oplus 2c \oplus 3d \\ 3a \oplus b \oplus c \oplus 2d \end{bmatrix}$$

In our example, this would be as follows.

$$\begin{bmatrix} 21 & 34 & BC & 74 \\ 62 & 2E & 2B & DA \\ 2E & 76 & 7D & 44 \\ 99 & 91 & 05 & 8D \end{bmatrix} \mapsto \begin{bmatrix} 58 & FD & 66 & 54 \\ 13 & 63 & 68 & 9A \\ B9 & 5E & 62 & AA \\ 0D & 3D & 83 & 03 \end{bmatrix}$$

AddRoundKey

At this stage, we perform another `XOR` operation with the round key. The keys are generated in a fairly straightforward way, using some modular arithmetic, the Rijndael S-Box, and some more mathematical operations in $GF(2^8)$.

AddRoundKey

At this stage, we perform another `XOR` operation with the round key. The keys are generated in a fairly straightforward way, using some modular arithmetic, the Rijndael S-Box, and some more mathematical operations in $GF(2^8)$.

- We then perform another 8 rounds, to get to the end of round 9.

AddRoundKey

At this stage, we perform another `XOR` operation with the round key. The keys are generated in a fairly straightforward way, using some modular arithmetic, the Rijndael S-Box, and some more mathematical operations in $GF(2^8)$.

- We then perform another 8 rounds, to get to the end of round 9.
- The final round is Round 10:
  1. ByteSub Transformation,
  2. ShiftRow Transformation,
  3. AddRoundKey.

AddRoundKey

At this stage, we perform another `XOR` operation with the round key. The keys are generated in a fairly straightforward way, using some modular arithmetic, the Rijndael S-Box, and some more mathematical operations in $GF(2^8)$.

- We then perform another 8 rounds, to get to the end of round 9.
- The final round is Round 10:
  1. ByteSub Transformation,
  2. ShiftRow Transformation,
  3. AddRoundKey.
- Round 10 has no MixColumns Transformation, for decryption purposes.

AddRoundKey

- 10 rounds for 128-bit key, 12 rounds for 192-bit key, 14 rounds for 256-bit key.

AddRoundKey

- 10 rounds for 128-bit key, 12 rounds for 192-bit key, 14 rounds for 256-bit key. Why 10 rounds?

AddRoundKey

- 10 rounds for 128-bit key, 12 rounds for 192-bit key, 14 rounds for 256-bit key. Why 10 rounds?
- There were known attacks up to 6 rounds which were better than brute force.

AddRoundKey

- 10 rounds for 128-bit key, 12 rounds for 192-bit key, 14 rounds for 256-bit key. Why 10 rounds?
- There were known attacks up to 6 rounds which were better than brute force.
- An extra four rounds added a good margin of safety, and the number of rounds could always be increased if required.

AddRoundKey

- 10 rounds for 128-bit key, 12 rounds for 192-bit key, 14 rounds for 256-bit key. Why 10 rounds?
- There were known attacks up to 6 rounds which were better than brute force.
- An extra four rounds added a good margin of safety, and the number of rounds could always be increased if required.
- Decryption is a similar process, but not simply the direct inverse of the encryption process: not symmetric.
    1. AddRoundKey is its own inverse,

AddRoundKey

- 10 rounds for 128-bit key, 12 rounds for 192-bit key, 14 rounds for 256-bit key. Why 10 rounds?
- There were known attacks up to 6 rounds which were better than brute force.
- An extra four rounds added a good margin of safety, and the number of rounds could always be increased if required.
- Decryption is a similar process, but not simply the direct inverse of the encryption process: not symmetric.
  1. AddRoundKey is its own inverse,
  2. InverseByteSub uses an an Inverse S-Box,

AddRoundKey

- 10 rounds for 128-bit key, 12 rounds for 192-bit key, 14 rounds for 256-bit key. Why 10 rounds?
- There were known attacks up to 6 rounds which were better than brute force.
- An extra four rounds added a good margin of safety, and the number of rounds could always be increased if required.
- Decryption is a similar process, but not simply the direct inverse of the encryption process: not symmetric.
  1. AddRoundKey is its own inverse,
  2. InverseByteSub uses an an Inverse S-Box,
  3. InverseShiftRow shifts rows to the right, instead of left,

AddRoundKey

- 10 rounds for 128-bit key, 12 rounds for 192-bit key, 14 rounds for 256-bit key. Why 10 rounds?
- There were known attacks up to 6 rounds which were better than brute force.
- An extra four rounds added a good margin of safety, and the number of rounds could always be increased if required.
- Decryption is a similar process, but not simply the direct inverse of the encryption process: not symmetric.
  1. AddRoundKey is its own inverse,
  2. InverseByteSub uses an an Inverse S-Box,
  3. InverseShiftRow shifts rows to the right, instead of left,
  4. InvMixColumn is more complicated than the encryption process, but relies on a similar mathematical manipulation.

Modes of Operation

Five standard ways of implementing AES:

- Mode 1: Electronic Code Book - not recommended due to plaintext attacks
- Mode 2: Cipher Block Chaining
- Mode 3: Cipher Feedback Mode
- Mode 4: Output Feedback Mode
- Mode 5: Counter Mode - NIST recommended mode (2001)

Where is AES used?

AES was officially ratified in 2002, with the intention that it takes over from Triple-DES in most applications:

Where is AES used?

AES was officially ratified in 2002, with the intention that it takes over from Triple-DES in most applications:

- Compression Tools: WinZip, RAR, 7-Zip.
- Wireless Connections: WPA2.
- VPNs: IPsec.
- Password Managers: LastPass.
- Communication: Signal Protocol in WhatsApp, Signal, Facebook Messenger, etc.

The common Cipher Block Chaining mode of operation is processor intensive, so some chip designs include partial AES instructions to improve efficiency:

- x86-64 architecture processors.

How secure is AES?

- In 2008 AES was reapproved to 2030.
- SECRET: 128, 192, or 256-bit keys.
- TOP SECRET: 192 or 256-bit keys.
- Best brute force attack to date was against a 64-bit key.
- Many cryptographers research potential attacks on AES using more nuanced techniques.

How secure is AES?

- In 2008 AES was reapproved to 2030.
- SECRET: 128, 192, or 256-bit keys.
- TOP SECRET: 192 or 256-bit keys.
- Best brute force attack to date was against a 64-bit key.
- Many cryptographers research potential attacks on AES using more nuanced techniques.

With that in mind, there is some concern that quantum computation will allow various techniques to be brought to bear on the problem of cracking AES. But the NIST doesn't seem too worried for now:

'*Taking these mitigating factors into account, it is quite likely that Grover's algorithm will provide little or no advantage in attacking AES, and AES 128 will remain secure for decades to come.*'

Tutorials

In the tutorial this week we will:

- Create a spreadsheet to perform the first round of AES encryption. (Compare with Q3 on the coursework.)
- Use any remaining time to carry on with the coursework.