

Cryptography

Advanced
Encryption
Standard

Overview of AES

Simplified-AES

Simplified-AES
Example

AES in OpenSSL

AES in Python

Advanced Encryption Standard

Cryptography

School of Engineering and Technology
CQUniversity Australia

Prepared by Steven Gordon on 05 Jan 2022,
aes.tex, r1980

Contents

Overview of AES

Overview of AES

Simplified-AES

Simplified-AES

Simplified-AES
Example

Simplified-AES Example

AES in OpenSSL

AES in OpenSSL

AES in Python

AES in Python

History of AES

- ▶ 1977: DES (56-bit key). NIST published.
- ▶ 1991: IDEA, similar to DES, secure but patent issues
- ▶ 1999: 3DES (168-bit key). NIST recommended 3DES be used (DES only for legacy systems)
 - ▶ 3DES was considered secure (apart from special case attacks)
 - ▶ But 3DES is very slow, especially in software
 - ▶ DES and 3DES use 64-bit blocks – larger block sizes required for efficiency
- ▶ 1997: NIST called for proposals for new Advanced Encryption Standards
 - ▶ Proposals made public and evaluations performed
- ▶ 2001: Selected *Rijndael* as the algorithm for AES

Selecting a Winner

- ▶ Original NIST criteria:
 - ▶ Security: effort to cryptanalyse algorithm, randomness, ...
 - ▶ Cost: royalty-free license, computationally efficient, ...
 - ▶ Algorithm and implementation characteristics: flexibility (different keys/blocks, implement on different systems), simplicity, ...
- ▶ 21 candidate algorithms reduced to 5
- ▶ Updated NIST evaluation criteria for 5 algorithms:
 - ▶ General Security
 - ▶ Software and hardware implementations (needs to be efficient)
 - ▶ Low RAM/ROM requirements (e.g. for smart cards)
 - ▶ Ability to change keys quickly
 - ▶ Potential to use parallel processors

Selecting Rijndael for AES

- ▶ Security: good, no known attacks
- ▶ Software implementation: fast, can make use of parallel processors
- ▶ Hardware implementation: fastest of all candidates
- ▶ Low memory requirements: good, except encryption and decryption require separate space
- ▶ Timing and Power analysis attacks: easiest to defend against
- ▶ Key flexibility: supports on-the-fly change of keys and different size of keys/blocks

Overview of AES

- ▶ NIST Advanced Encryption Standard, FIPS-197, 2001
- ▶ Three variations of same algorithm standardised
 - ▶ AES-128: 128-bit key, 10 rounds
 - ▶ AES-192: 192-bit key, 12 rounds
 - ▶ AES-256: 256-bit key, 14 rounds
- ▶ AES uses 128-bit block size for all variations
- ▶ S-AES used to understand AES (educational only)
- ▶ For details of AES see the Stallings textbook, AES on Wikipedia or the AES standard from NIST

Cryptography

Advanced
Encryption
Standard

Contents

Overview of AES

Overview of AES

Simplified-AES

Simplified-AES

Simplified-AES
Example

Simplified-AES Example

AES in OpenSSL

AES in OpenSSL

AES in Python

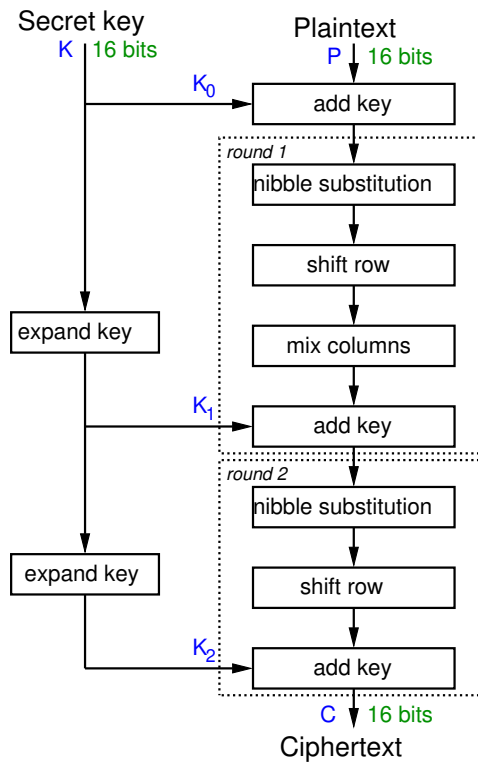
AES in Python

Simplified-AES

- ▶ Educational purposes only. Mohammad A. Musa , Edward F. Schaefer and Stephen Wedig (2003) A Simplified AES Algorithm and its Linear and Differential Cryptanalyses, Cryptologia, 27:2, 148-177, DOI: 10.1080/0161-110391891838
- ▶ Input: 16-bit block of plaintext; 16-bit key
- ▶ Output: 16-bit block of ciphertext
- ▶ Operations:
 - ▶ Add Key: XOR of a 16-bit key and 16-bit state matrix
 - ▶ Nibble Substitution: S-Box table lookup that swaps nibbles (4 bits)
 - ▶ Shift Row: shift of nibbles in a row
 - ▶ Mix Column: re-order columns
 - ▶ Rotate Nibbles: swap the nibbles
- ▶ 3 rounds (although they don't contain same operations)

S-AES operates on 16-bit blocks, with some operations on 8-bit words and others on 4-bit nibbles. For example, a 16-bit block is equivalent to two 8-bit words or four 4-bit nibbles.

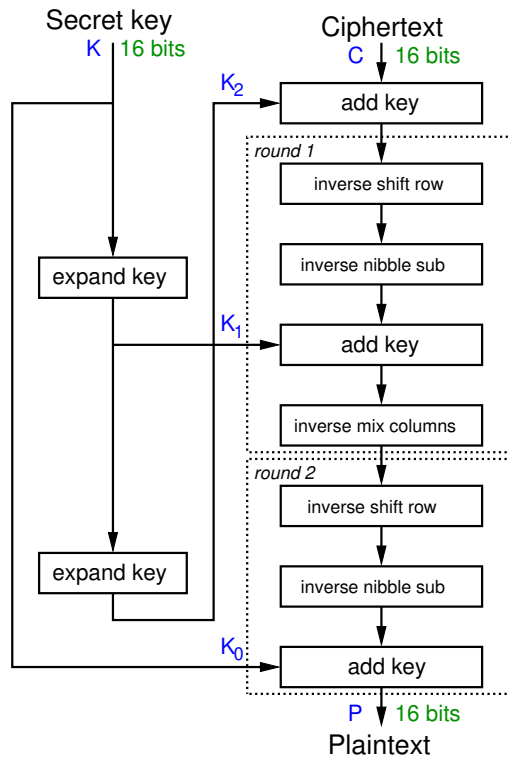
S-AES Encryption



The figure on slide 9 shows the overall steps for S-AES and key expansion and encryption. The key generation takes a 16-bit secret key and expands that into 3 16-bit round keys. The first round key K_0 is simply the original key. The next two round keys, K_1 and K_2 are generated by an expansion algorithm. The figure on slide 11 shows that algorithm for K_1 .

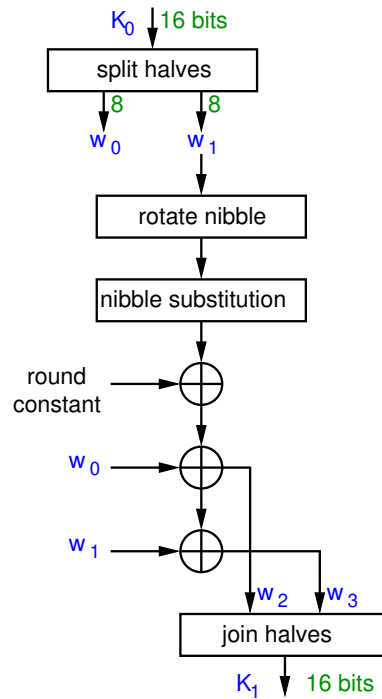
S-AES encryption operates on 16-bit blocks of plaintext. To encrypt, there is an initial *add key*, and then two rounds, where the 2nd round does not include the *mix columns* operation.

S-AES Decryption



The figure on slide 10 shows the decryption operations. Note that it is similar to encryption in reverse, with all operations replaced with their inverse operations. The same round keys are used as in encryption, but in the opposite order.

S-AES Key Generation for Round 1



The figure on slide 11 shows the key generation operations for generated round key K_1 . Similar steps are used to generate K_2 , where the input is K_1 and a different round constant.

S-AES State Matrix (definition)

S-AES operates on a 16-bit state matrix, viewed as 4 nibbles

$$\begin{bmatrix} b_0 b_1 b_2 b_3 & b_8 b_9 b_{10} b_{11} \\ b_4 b_5 b_6 b_7 & b_{12} b_{13} b_{14} b_{15} \end{bmatrix} = \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix}$$

Overview of AES

Simplified-AES

Simplified-AES
Example

AES in OpenSSL

AES in Python

While S-AES operates on 16-bits at a time, those bits are viewed as a state matrix of 4 nibbles. Note the matrix is filled columnwise, with the first 8 bits (2 nibbles) in the first column.

The following shows operations based on the state matrix.

S-AES Shift Row, Add Key and Rotate Nibble operations (definition)

S-AES Shift Row:

$$\begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix} \rightarrow \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,1} & S_{1,0} \end{bmatrix}$$

S-AES Add Key: Exclusive OR (XOR)

S-AES Rotate Nibble: swap the two nibbles

S-AES Nibble Substitution: apply S-Box on each nibble

S-AES Round Constant 1: 10000000

S-AES Round Constant 2: 00110000

Shift Row swaps the 2nd nibble with the 4th nibble. Add Key is a bitwise XOR. The round constants are used in the key generation.

S-AES S-Boxes (definition)

S-Box considered as a matrix: input used to select row/column; selected element is output

Input: 4-bit nibble, $bit_1, bit_2, bit_3, bit_4$

bit_1bit_2 specifies row

bit_3bit_4 specifies column

$$\text{encrypt : } \begin{bmatrix} 1001 & 0100 & 1010 & 1011 \\ 1101 & 0001 & 1000 & 0101 \\ 0110 & 0010 & 0000 & 0011 \\ 1100 & 1110 & 1111 & 0111 \end{bmatrix}$$

$$\text{decrypt : } \begin{bmatrix} 1010 & 0101 & 1001 & 1011 \\ 0001 & 0111 & 1000 & 1111 \\ 0110 & 0000 & 0010 & 0011 \\ 1100 & 0100 & 1101 & 1110 \end{bmatrix}$$

The left-most 2 bits in a nibble determine the row, and the right-most 2 bits in the nibble determine the column. The output nibble is based on the S-Box. The Inverse S-Box is used in decryption.

S-AES Mix Columns (definition)

Mix the columns in the state matrix by performing a matrix multiplication.

Mix Columns:

$$\begin{bmatrix} S'_{0,0} & S'_{0,1} \\ S'_{1,0} & S'_{1,1} \end{bmatrix} = \begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix}$$

Inverse Mix Columns:

$$\begin{bmatrix} S'_{0,0} & S'_{0,1} \\ S'_{1,0} & S'_{1,1} \end{bmatrix} = \begin{bmatrix} 9 & 2 \\ 2 & 9 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix}$$

Galois Field $GF(2^4)$ is used for addition and multiplication operations.

S' denotes the output from the mixing of columns, e.g. $S'_{0,0} = (1 \times S_{0,0}) + (4 \times S_{1,0})$. Importantly, the resulting addition and multiplication operations are in Galois Field $GF(2^4)$. We do not cover (Galois) fields, however in Number Theory we saw modular arithmetic with mod n where all operations produced results within 0 to n . This is a simple case of a field, i.e. all operations produce answers within some finite range. $GF(2^4)$ means all answers will be within range 0 to 15.

$GF(2^4)$ addition is equivalent to bitwise XOR. However $GF(2^4)$ multiplication is more complicated. Therefore, for the purpose of demonstrating S-AES, a simplified view of the mix column operations with a table lookup for multiplication is shown in the following.

S-AES Mix Columns (Simple) (definition)

Mix the columns in the state matrix by performing the following calculations.

Mix Columns:

$$S'_{0,0} = S_{0,0} \oplus (0100 \times S_{1,0})$$

$$S'_{1,0} = (0100 \times S_{0,0}) \oplus S_{1,0}$$

$$S'_{0,1} = S_{0,1} \oplus (0100 \times S_{1,1})$$

$$S'_{1,1} = (0100 \times S_{0,1}) \oplus S_{1,1}$$

Inverse Mix Columns:

$$S'_{0,0} = (1001 \times S_{0,0}) \oplus (0010 \times S_{1,0})$$

$$S'_{1,0} = (0010 \times S_{0,0}) \oplus (1001 \times S_{1,0})$$

$$S'_{0,1} = (1001 \times S_{0,1}) \oplus (0010 \times S_{1,1})$$

$$S'_{1,1} = (0010 \times S_{0,1}) \oplus (1001 \times S_{1,1})$$

For multiplication, lookup using The figure on slide 17.

GF(2⁴) Multiplication Table used in S-AES

Overview of AES

Simplified-AES

Simplified-AES
Example

AES in OpenSSL

AES in Python

x	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0001	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0010	0000	0010	0100	0110	1000	1010	1100	1110	0011	0001	0111	0101	1011	1001	1111	1101
0011	0000	0011	0110	0101	1100	1111	1010	1001	1011	1000	1101	1110	0111	0100	0001	0010
0100	0000	0100	1000	1100	0011	0111	1011	1111	0110	0010	1110	1010	0101	0001	1101	1001
0101	0000	0101	1010	1111	0111	0010	1101	1000	1110	1011	0100	0001	1001	1100	0011	0110
0110	0000	0110	1100	1010	1011	1101	0111	0001	0101	0011	1001	1111	1110	1000	0010	0100
0111	0000	0111	1110	1001	1111	1000	0001	0110	1101	1010	0011	0100	0010	0101	1100	1011
1000	0000	1000	0011	1011	0110	1110	0101	1101	1100	0100	1111	0111	1010	0010	1001	0001
1001	0000	1001	0001	1000	0010	1011	0011	1010	0100	1101	0101	1100	0110	1111	0111	1110
1010	0000	1010	0111	1101	1110	0100	1001	0011	1111	0101	1000	0010	0001	1011	0110	1100
1011	0000	1011	0101	1110	1010	0001	1111	0100	0111	1100	0010	1001	1101	0110	1000	0011
1100	0000	1100	1011	0111	0101	1001	1110	0010	1010	0110	0001	1101	1111	0011	0100	1000
1101	0000	1101	1001	0100	0001	1100	1000	0101	0010	1111	1011	0110	0011	1110	1010	0111
1110	0000	1110	1111	0001	1101	0011	0010	1100	1001	0111	0110	1000	0100	1010	1011	0101
1111	0000	1111	1101	0010	1001	0110	0100	1011	0001	1110	1100	0011	1000	0111	0101	1010

The figure on slide 17 shows the GF(2⁴) multiplication table in binary. The green column is used in encryption (Mix Columns) and the two blue columns are used in decryption (Inverse Mix Columns). For example with encryption, when multiplying a value by 4 (0100 in binary), lookup the value in the first column (e.g. 0111) and the answer will be in the green column (e.g. 1111).

Comparing S-AES and AES-128

▶ S-AES

- ▶ 16-bit key, 16-bit plaintext/ciphertext
- ▶ 2 rounds: first with all 4 operations, last with 3 operations
- ▶ Round key size: 16 bits
- ▶ Mix Columns: arithmetic over $GF(2^4)$

▶ AES-128

- ▶ 128-bit key, 128-bit plaintext/ciphertext
- ▶ 10 rounds: first 9 with all 4 operations, last with 3 operations
- ▶ Round key size: 128 bits
- ▶ Mix Columns: arithmetic over $GF(2^8)$

- ▶ Principles of operation are the same

Contents

Overview of AES

Overview of AES

Simplified-AES

Simplified-AES

Simplified-AES
Example

Simplified-AES Example

AES in OpenSSL

AES in OpenSSL

AES in Python

AES in Python

Encrypt with S-AES (exercise)

Show that when the plaintext **1101 0111 0010 1000** is encrypted using Simplified-AES with key **0100 1010 1111 0101** that the ciphertext obtained is **0010 0100 1110 1100**.

Contents

Overview of AES

Overview of AES

Simplified-AES

Simplified-AES

Simplified-AES
Example

Simplified-AES Example

AES in OpenSSL

AES in OpenSSL

AES in Python

AES in Python

AES Key Generation (exercise)

Generate a shared secret key to be used with AES and share it with another person.

Overview of AES

Simplified-AES

Simplified-AES
Example

AES in OpenSSL

AES in Python

AES Encryption (exercise)

Create a message in a plain text file and after using AES, send the ciphertext to the person you shared the key with.

Overview of AES

Simplified-AES

Simplified-AES
Example

AES in OpenSSL

AES in Python

Cryptography

Advanced
Encryption
Standard

AES Decryption (exercise)

Decrypt the ciphertext you received.

Overview of AES

Simplified-AES

Simplified-AES
Example

AES in OpenSSL

AES in Python

AES Performance Benchmarking (exercise)

Perform speed tests on AES using both the software and hardware implementations (if available). Compare and discuss the impact of the following on performance: key length; software vs hardware; different computers (e.g. compare the performance with another person).

Cryptography

Advanced
Encryption
Standard

Contents

Overview of AES

Overview of AES

Simplified-AES

Simplified-AES

Simplified-AES
Example

Simplified-AES Example

AES in OpenSSL

AES in OpenSSL

AES in Python

AES in Python

Cryptography

Advanced
Encryption
Standard

AES in Python Cryptography Library

- ▶ <https://cryptography.io/en/latest/hazmat/primitives/symmetric-encryption/>

Overview of AES

Simplified-AES

Simplified-AES
Example

AES in OpenSSL

AES in Python