Advanced Encryption Standard (AES)

Sources:

- Advanced Encryption Standard, NIST CSRC, http://csrc.nist.gov/CryptoToolkit/aes/rijndael/
- *Practical Cryptography* by N. Ferguson and B. Schneier
- *Modern Cryptography: Theory and Practice* by W. Mao
- AES Lounge, http://www.iaik.tu-graz.ac.at/research/krypto/AES/
- *A Simple Algebraic Representation of Rijndael* by N. Ferguson, R. Schroeppel, and D. Whiting, Selected Areas in Cryptography, Proc. SAC 2001, Lecture Notes in Computer Science #2259, pp. 103–111, Springer Verlag, 2001.

In 1997, the U.S. National Institute of Standards and Technology (NIST) made an open call for algorithms for the *Advanced Encryption Standard* (*AES*).

The call stipulated that:

- AES would specify an unclassified, publicly disclosed symmetric-key encryption algorithm
- The algorithm must support (at least) block sizes of 128 bits, key sizes of 128, 192, and 256 bits
- Should have strength at the level of triple DES, but should be more efficient
- If selected, the algorithm must be available royalty-free, worldwide

In 1998, NIST announced 15 AES candidate algorithms.

Public comments were solicited.

Five finalist algorithms were announced in 1999: MARS, RC6, Rijndael, Serpent, and Twofish.

Further comments and analysis were sought.

In 2000, NIST announced it had selected Rijndael for the AES.

It was designed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen.

Overview of Rijndael [Mao]

Rijndael is a block cipher with a variable block size and a variable key size.

The key size and block size can be independently specified to 128, 192, or 256 bits.

For simplicity, we describe the case with 128 bit key and block sizes.

A 128-bit message block and key block are each segmented into 16 bytes:

InputBlock = $m_0, m_1, \ldots, m_{15}$

InputKey = $k_0, k_1, \ldots, k_{15}$

The data structure for their internal representation is a 4×4 matrix:

$$\text{InputBlock} = \begin{pmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{pmatrix}$$

$$\text{InputKey} = \begin{pmatrix} k_0 & k_4 & k_8 & k_{12} \\ k_1 & k_5 & k_9 & k_{13} \\ k_2 & k_6 & k_{10} & k_{14} \\ k_3 & k_7 & k_{11} & k_{15} \end{pmatrix}$$

The Rijndael algorithm comprises a number of *rounds* – ten for 128 bit keys and blocks.

A round transformation is denoted by

Round(*State*, *RoundKey*)

State is a round-message matrix and is treated as both input and output.

RoundKey is a round-key matrix and is derived from the input key via a *key schedule*.

Execution of a round will change the values in *State*.

For encryption (respectively, decryption):

- *State* input to the first round is InputBlock, which is the plaintext (respectively, ciphertext) message matrix

- *State* output from the final round is the ciphertext (respectively, plaintext) message matrix

The non-final round transformation is composed of four transformation functions:

```
Round(State, RoundKey) {
    SubBytes(State);
    ShiftRows(State);
    MixColumns(State);
    AddRoundKey(State, RoundKey);
}
```

The final round, denoted by

FinalRound(*State, RoundKey*)

is slightly different.

It is equal to Round(*State*, *RoundKey*) with the call to MixColumns function removed.

For decryption, the round transformations are *inverted*:

Round^{-1}(*State, RoundKey*)

The four internal functions are each invertible.

Their inverses are applied in *reverse order* during decryption.

Internal Functions of Rijndael

These functions work in a *finite field*, called the *Rijndael field*.

This is realized as all polynomials modulo the irreducible polynomial

$$f(x) = x^8 + x^4 + x^3 + x + 1$$

over $\mathbb{F}_2$ (the finite field with two elements).

An element of this field is a polynomial over $\mathbb{F}_2$ of degree less than 8.

This field has 256 elements.

Each can be represented by an 8-bit byte, whose bits represent the coefficients $b_7b_6b_5b_4b_3b_2b_1b_0$.

Conversely, any byte can be viewed as an element of this field.

Addition can be done by adding coefficients modulo 2.

Multiplication in the Rijndael field is multiplication of two polynomials modulo *f*.

The modulo operation can be implemented by applying the *Extended Euclidean Algorithm* for polynomials.

Function SubBytes(*State*)

This function provides a *nonlinear substitution* for each byte of *State*.

Any nonzero byte *x* is replaced using the following transformation:

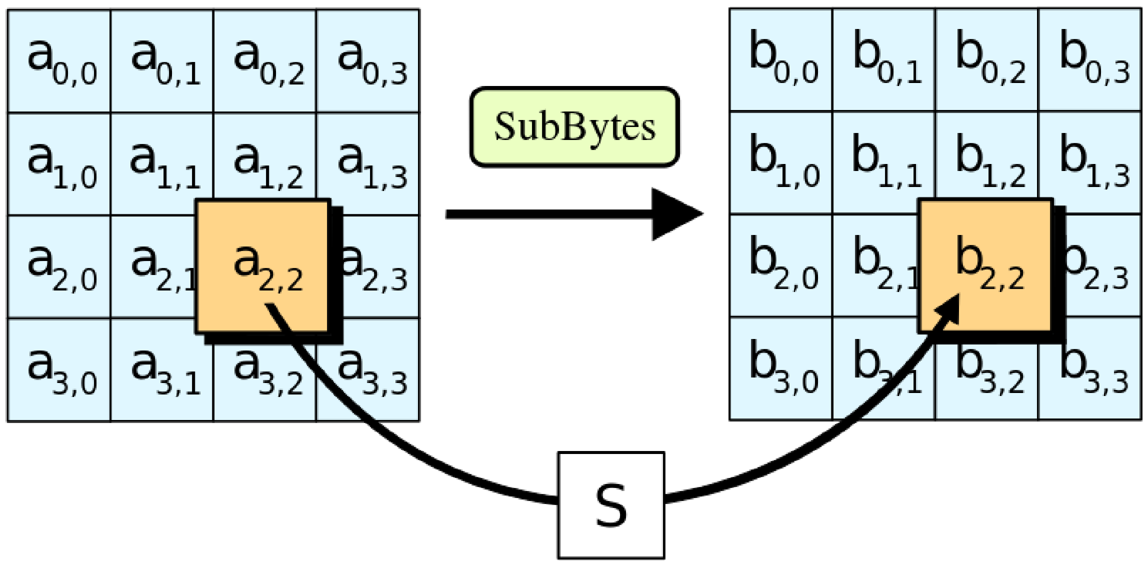
$$y = Ax^{-1} + b$$

where *A* is a fixed 8×8 invertible binary matrix and *b* is a fixed 8-element binary vector:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \text{ and } b = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

The nonlinearity of this transformation comes from the inversion of *x*.

If *x* is the zero byte, then $y = x$ is the transformation result.



[en.wikipedia.org/wiki/Advanced_Encryption_Standard#mediaviewer/File:AES-SubBytes.svg]

Function ShiftRows(*State*)

This function operates on each row of *State*.

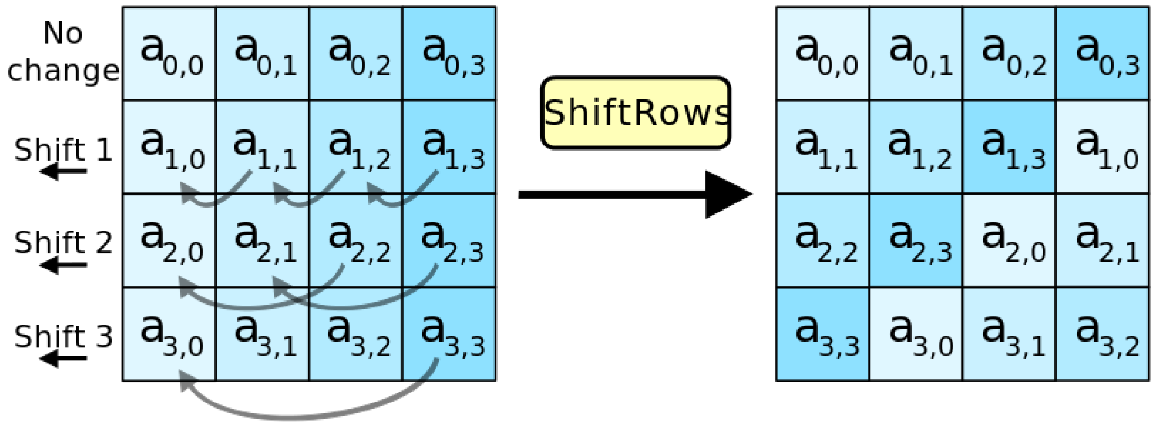
For 128 bit blocks, it is the following transformation:

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \rightarrow \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{pmatrix}$$

This operation is actually a *transposition cipher*.

It rearranges the positions of the elements without changing their identities.

The elements in the *i*th row (*i* = 0, 1, 2, 3) are shifted cyclically to the right by 4 – *i* positions.


No change
Shift 1
Shift 2
Shift 3
ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

Function MixColumns(***State***)

This function iterates over the columns of *State* one at a time.

The output of an iteration is still a column.

Let

$$\begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

be a column from the output of ShiftRows(*State*).

This column is interpreted as a degree-3 polynomial:

$$s(x) = s_3x^3 + s_2x^2 + s_1x + s_0.$$

This is a polynomial *over* the Rijndael field, not an element of it.

It is multiplied by a fixed degree-3 polynomial $c(x)$, modulo $x^4 + 1$:

$$c(x) = c_3x^3 + c_2x^2 + c_1x + c_0$$

If the coefficient bytes are represented in *hexadecimal*, we have:

$$c(x) = \text{‘03’}x^3 + \text{‘01’}x^2 + \text{‘01’}x + \text{‘02’}$$

The transformation $c(x)\cdot s(x) \pmod{x^4 + 1}$ can be viewed as a *polyalphabetic substitution cipher* using a known key.

In such a cipher a plaintext message element can be substituted into many ciphertext message elements.

The following equation over $\mathbb{F}_2$ may be confirmed by long division:

$$x^i \pmod{x^4 + 1} = x^{i \pmod 4}$$

Hence, the coefficient of x^i (for $i = 0, 1, 2, 3$) must be the sum of c_js_k satisfying $j + k = i \pmod 4$ (where $j, k = 0, 1, 2, 3$).

For example, the coefficient of x^2 is

$$c_2 s_0 + c_1 s_1 + c_0 s_2 + c_3 s_3$$

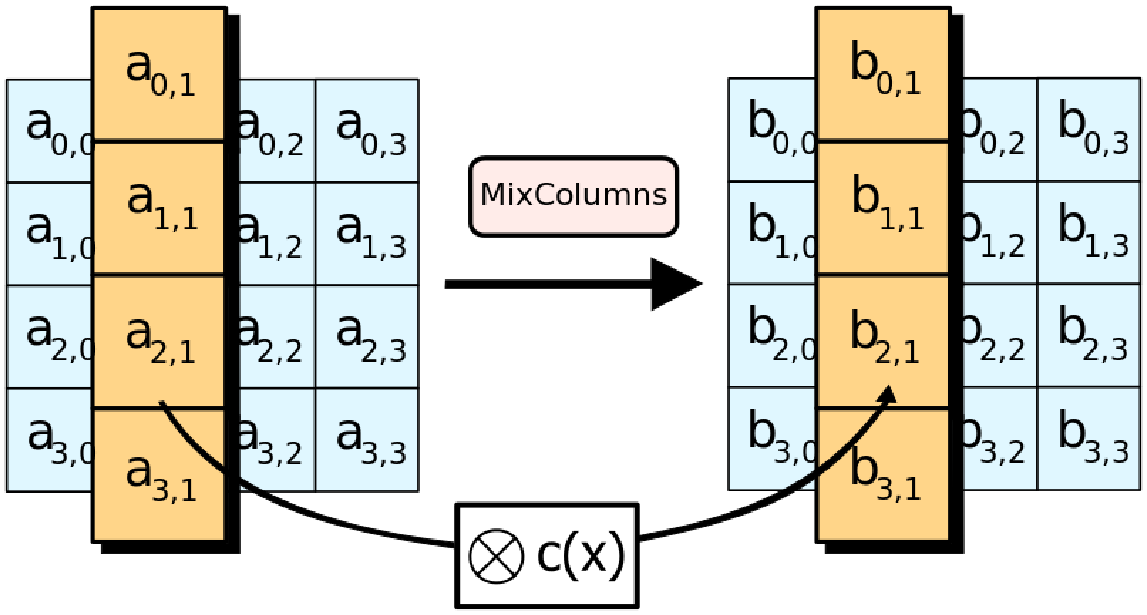
The multiplication and addition are in the Rijndael field.

It can be verified that the polynomial multiplication in $c(x) \cdot s(x)$ (mod $x^4 + 1$) can be achieved by taking the following matrix-vector product:

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} c_0 & c_3 & c_2 & c_1 \\ c_1 & c_0 & c_3 & c_2 \\ c_2 & c_1 & c_0 & c_3 \\ c_3 & c_2 & c_1 & c_0 \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$
$$= \begin{pmatrix} '02' & '03' & '01' & '01' \\ '01' & '02' & '03' & '01' \\ '01' & '01' & '02' & '03' \\ '03' & '01' & '01' & '02' \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

Because $c(x)$ is *relatively prime* to $x^4 + 1$ over $\mathbb{F}_2$, the inverse $c(x)^{-1}$ (mod $x^4 + 1$) exists.

Hence, the transformation above is invertible.



[en.wikipedia.org/wiki/Advanced_Encryption_Standard#mediaviewer/File: AES-MixColumns.svg]

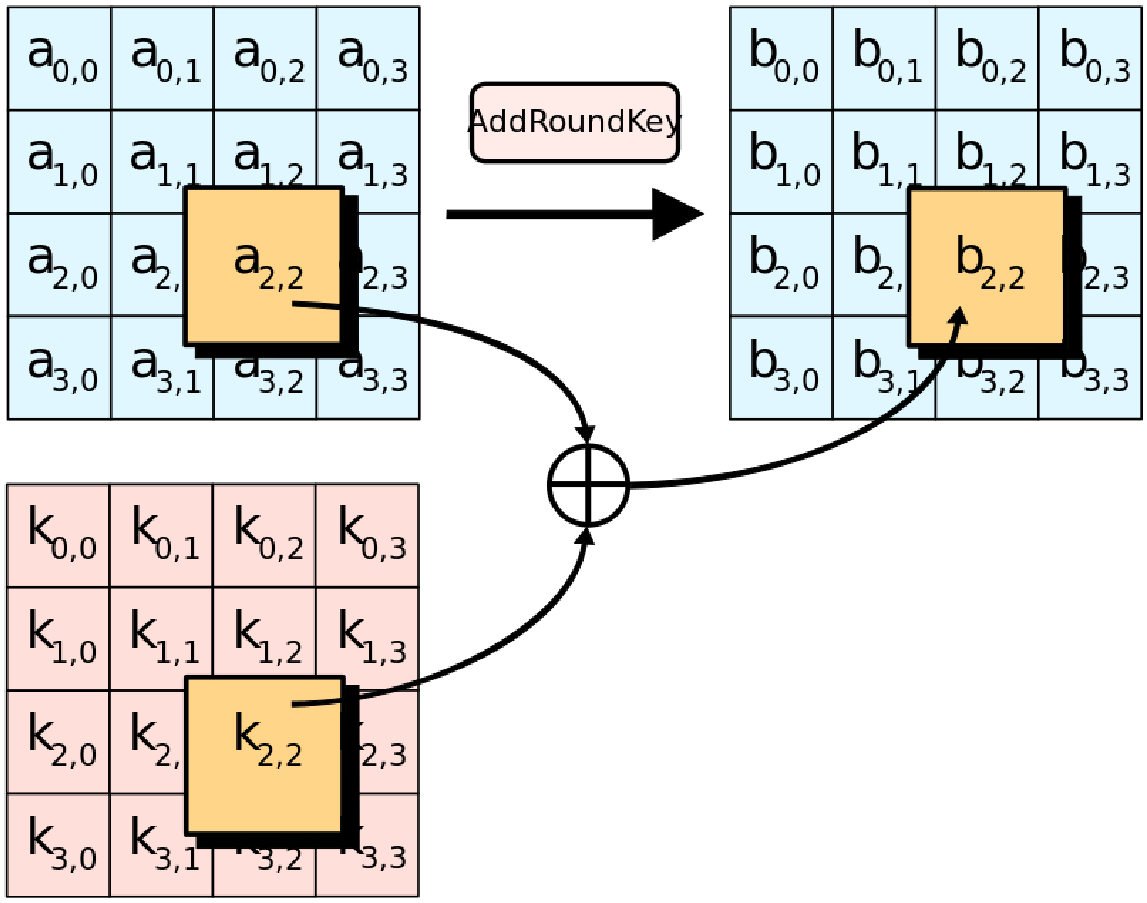
Function AddRoundKey(*State*, *RoundKey*)

This function simply adds the elements of *RoundKey* to those of *State*, byte by byte and bit by bit.

Here addition is in $\mathbb{F}_2$ (bitwise XOR), which is invertible.

The key bits for different rounds are different.

They are derived from the key using a fixed "key schedule".



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
AddRoundKey
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$
$k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$
$k_{2,0}$ $k_{2,2}$ $k_{2,3}$
$k_{3,0}$ $k_{3,1}$ $k_{3,2}$ $k_{3,3}$

Decryption in Rijndael

A *decryption round* is accomplished by the following sequence of operations:

AddRoundKey^{-1}(*State*, *RoundKey*);
MixColumns^{-1}(*State*);
ShiftRows^{-1}(*State*);
SubBytes^{-1}(*State*);

With Rijndael, different circuits or functions must be used for encryption and decryption.

Summary of the Roles of the Rijndael Internal Functions

- SubBytes is intended to be a *non-linear substitution cipher*.

  Nonlinearity is important for *preventing differential cryptanalysis*.

- ShiftRows and MixColumns are intended to achieve a *mixture* of the bytes in different positions of the plaintext block.

  This is Shannon's mixing property.

- AddRoundKey provides the needed *secret randomness* to the message distribution.

These functions are very simple(!) and operate in small algebraic spaces.

They can be implemented very efficiently in hardware or software by exploiting *table lookups*.

Impact of AES

AES obviates the need for multiple encryption as with triple-DES.

This reduces the number of keys an application has to manage.

It “should” also lead to the emergence of new cryptographic hash functions, but there are complications.

(It is standard practice to use block cipher encryption algorithms to play the role of one-way hash functions.)

Research on the security of AES continues apace.

Security of Rijndael

The security of Rijndael depends on a new and untested hardness assumption.

This assumption is that it is computationally infeasible to solve certain equations.

These equations have a very simple algebraic form.

For example, a single round of Rijndael can be expressed as follows [Ferguson]:

$$a_{i,j}^{(r+1)} = k_{i,j}^{(r)} + \sum_{\substack{e_r \in \mathcal{E} \\ d_r \in \mathcal{D}}} w_{i,e_r,d_r} \left(a_{e_r,e_r+j}^{(r)}\right)^{-2^{d_r}} \qquad (1)$$

AES Attacks

[Wikipedia, 2013]

On July 1, 2009, Bruce Schneier blogged[17] about a related-key attack on the 192-bit and 256-bit versions of AES, discovered by Alex Biryukov and Dmitry Khovratovich,[18] which exploits AES's somewhat simple key schedule and has a complexity of 2^{119}. In December 2009 it was improved to $2^{99.5}$. This is a follow-up to an attack discovered earlier in 2009 by Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolić, with a complexity of 2^{96} for one out of every 235 keys.[19]

Another attack was blogged by Bruce Schneier[20] on July 30, 2009 and released as a preprint[21] on August 3, 2009. This new attack, by Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich, and Adi Shamir, is against AES-256 that uses only two related keys and 2^{39} time to recover the complete 256-bit key of a 9-round version, or 2^{45} time for a 10-round version with a stronger type of related subkey attack, or 2^{70} time for an 11-round version. 256-bit AES uses 14 rounds, so these attacks aren't effective against full AES.

In November 2009, the first known-key distinguishing attack against a reduced 8-round version of AES-128 was released as a preprint.[22] This known-key distinguishing attack is an improvement of the rebound or the start-from-the-middle attacks for AES-like permutations, which view two consecutive rounds of permutation as the application of a so-called Super-Sbox. It works on the 8-round version of AES-128, with a time complexity of 2^{48}, and a memory complexity of 2^{32}.

In July 2010 Vincent Rijmen published an ironic paper on "chosen-key-relations-in-the-middle" attacks on AES-128.[23]
The first key-recovery attacks on full AES were due to Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger, and were published in 2011.[24] The attack is based on bicliques and is faster than brute force by a factor of about four. It requires $2^{126.1}$ operations to recover an AES-128 key. For AES-192 and AES-256, 2189.7 and 2254.4 operations are needed, respectively.