The AES Cipher - Rijndael

- designed by Rijmen-Daemen in Belgium
- has 128/192/256 bit keys, 128 bit data
- an iterative rather than Feistel cipher
  - processes data as block of 4 columns of 4 bytes
  - operates on entire data block in every round
- designed to be:
  - resistant against known attacks
  - fast and compact on many CPUs
  - simple
Rijndael

- data block of 4 columns of 4 bytes is state
- key is expanded to array of words
- has 9/11/13 rounds in which state undergoes:
  - byte substitution (1 s-box used on every byte)
  - shift rows (permute bytes between columns)
  - mix columns (substitute using matrix multiply)
  - add round key (XOR state with key material)
- view as alternating XOR key & scramble data bytes
- initial XOR key material & incomplete last round
- fast XOR & table lookup implementation
Rijndael
Byte Substitution

- a simple substitution of each byte
- uses one table of 16x16 bytes containing a permutation of all 256 8-bit values
- each byte of state is replaced by byte indexed by row (left 4-bits) & column (right 4-bits)
  - eg. byte {95} is replaced by byte in row 9 column 5
  - which has value {2A}
- S-box constructed using defined transformation of values in GF(2^8)
- designed to be resistant to all known attacks
Byte Substitution
Shift Rows

- a circular byte shift in each row
  - 1\textsuperscript{st} row is unchanged
  - 2\textsuperscript{nd} row does 1 byte circular shift to left
  - 3rd row does 2 byte circular shift to left
  - 4th row does 3 byte circular shift to left
- decrypt inverts using shifts to right
- since state is processed by columns, this step permutes bytes between the columns
Mix Columns

- each column is processed separately
- each byte is replaced by a value dependent on all 4 bytes in the column
- effectively a matrix multiplication in GF(2^8) using prime poly \( m(x) = x^8 + x^4 + x^3 + x + 1 \)
Mix Columns
Mix Columns

- can express each column as 4 equations
  - to derive each new byte in column
- decryption requires use of inverse matrix
  - with larger coefficients, hence a little harder
- have an alternate characterization
  - each column a 4-term polynomial
  - with coefficients in GF(2^8)
  - and polynomials multiplied modulo (x^4 + 1)
Add Round Key

- XOR state with 128-bits of the round key
- again processed by column (though effectively a series of byte operations)
- inverse for decryption identical
  - since XOR its own inverse, with reversed keys
- designed to be as simple as possible
  - a form of Vernam cipher on expanded key
  - requires other stages for complexity / security
Add Round Key
AES Round
AES Key Expansion

- takes 128-bit (16-byte) key and expands into array of 44/52/60 32-bit words
- start by copying key into first 4 words
- then loop creating words that depend on values in previous & 4 places back
  - in 3 of 4 cases just XOR these together
  - 1\textsuperscript{st} word in 4 has rotate + S-box + XOR round constant on previous, before XOR 4\textsuperscript{th} back
AES Key Expansion

The diagram shows the key expansion process for AES. The initial key values $k_0, k_1, k_2, k_3$ are expanded into $w_0, w_1, w_2, w_3$, which are then used to generate the expanded key $w_4, w_5, w_6, w_7$. The expansion involves the use of a specific function $g$ that operates on the initial key values.
Key Expansion Rationale

- designed to resist known attacks
- design criteria included
  - knowing part key insufficient to find many more
  - invertible transformation
  - fast on wide range of CPUs
  - use round constants to break symmetry
  - diffuse key bits into round keys
  - enough non-linearity to hinder analysis
  - simplicity of description
AES Decryption

- AES decryption is not identical to encryption since steps done in reverse
- but can define an equivalent inverse cipher with steps as for encryption
  - use inverses of each step
  - with a different key schedule
- works since result is unchanged when
  - swap byte substitution & shift rows
  - swap mix columns & add (tweaked) round key
AES Decryption
Implementation Aspects

- can efficiently implement on 8-bit CPU
  - byte substitution works on bytes using a table of 256 entries
  - shift rows is simple byte shift
  - add round key works on byte XOR’s
  - mix columns requires matrix multiply in GF(2^8) which works on byte values, can be simplified to use table lookups & byte XOR’s
Implementation Aspects

- can efficiently implement on 32-bit CPU
  - redefine steps to use 32-bit words
  - can pre-compute 4 tables of 256-words
  - then each column in each round can be computed using 4 table lookups + 4 XORs
  - at a cost of 4Kb to store tables

- designers believe this very efficient implementation was a key factor in its selection as the AES cipher