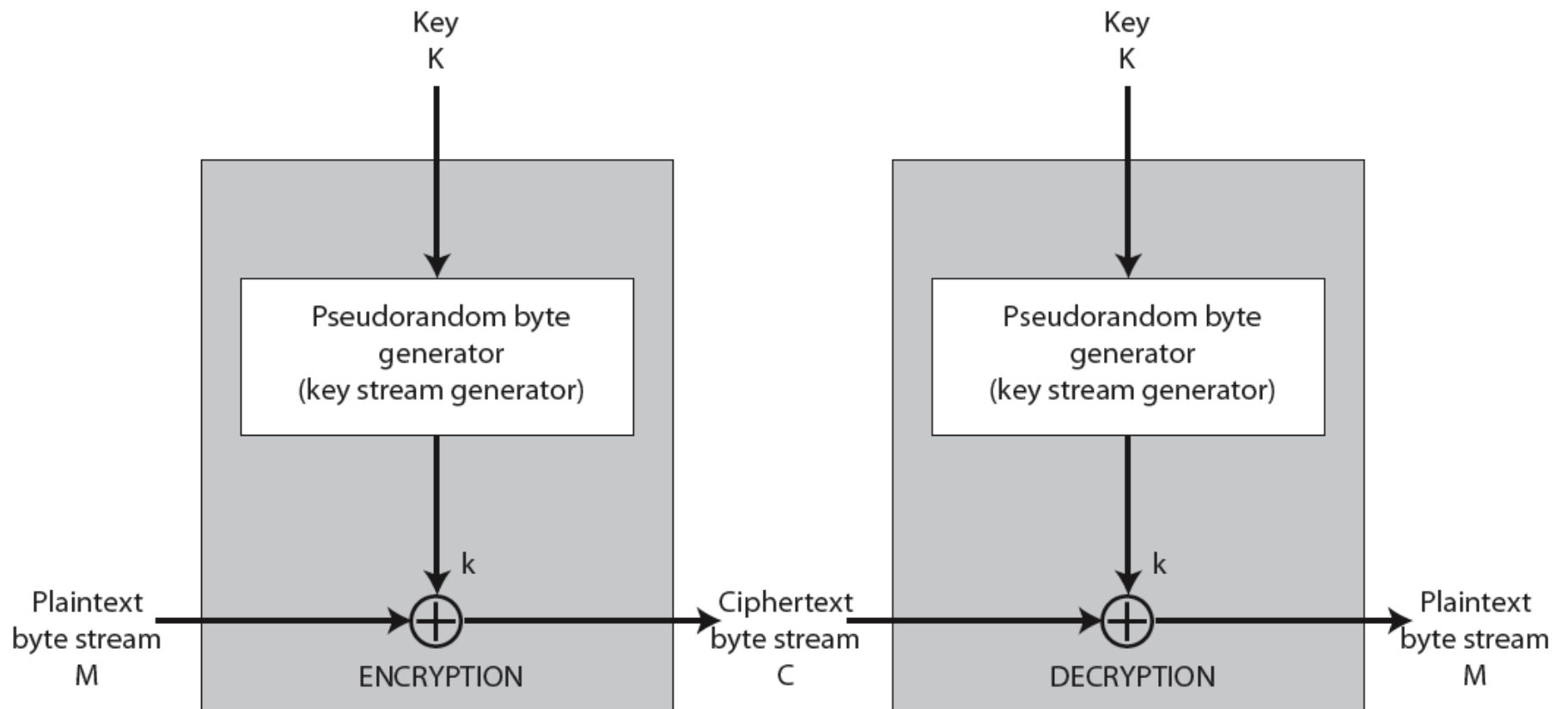


Stream Ciphers

- process message bit by bit (as a stream)
- have a pseudo random **keystream**
- combined (XOR) with plaintext bit by bit
- randomness of **stream key** completely destroys statistically properties in message
 - $C_i = M_i \text{ XOR } \text{StreamKey}_i$
- but must never reuse stream key
 - otherwise can recover messages (cf book cipher)

Stream Cipher Structure



Stream Cipher Properties

- some design considerations are:
 - long period with no repetitions
 - statistically random
 - depends on large enough key
 - large linear complexity
- properly designed, can be as secure as a block cipher with same size key
- but usually simpler & faster

RC4

- a proprietary cipher owned by RSA DSI
- another Ron Rivest design, simple but effective
- variable key size, byte-oriented stream cipher
- widely used (web SSL/TLS, wireless WEP)
- key forms random permutation of all 8-bit values
- uses that permutation to scramble input info processed a byte at a time

RC4 Key Schedule

- starts with an array S of numbers: 0..255
- use key to well and truly shuffle
- S forms **internal state** of the cipher

```
for i = 0 to 255 do
    S[i] = i
    T[i] = K[i mod keylen])
j = 0
for i = 0 to 255 do
    j = (j + S[i] + T[i]) (mod 256)
    swap (S[i], S[j])
```

RC4 Encryption

- encryption continues shuffling array values
- sum of shuffled pair selects "stream key" value from permutation
- XOR $S[t]$ with next byte of message to en/decrypt

$i = j = 0$

for each message byte M_i

$i = (i + 1) \pmod{256}$

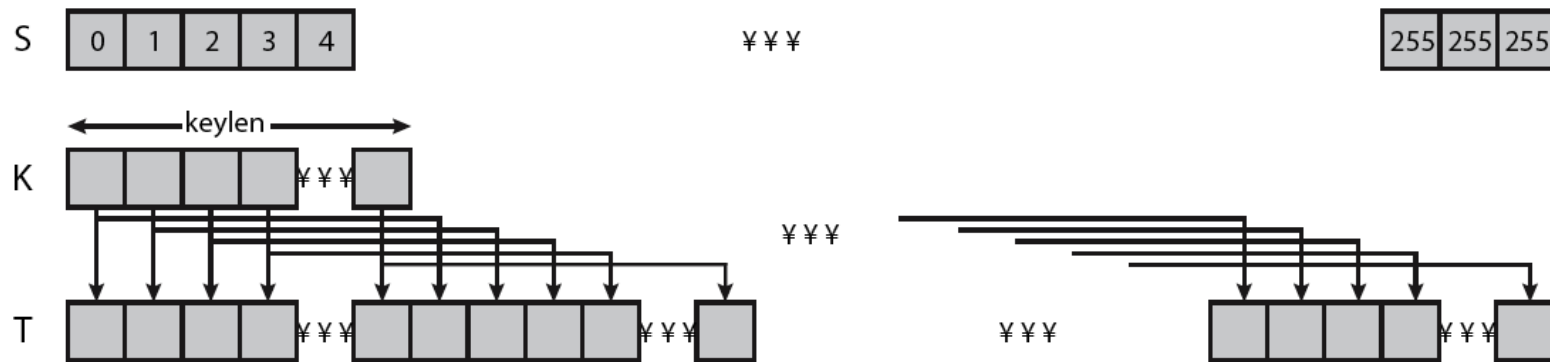
$j = (j + S[i]) \pmod{256}$

swap($S[i], S[j]$)

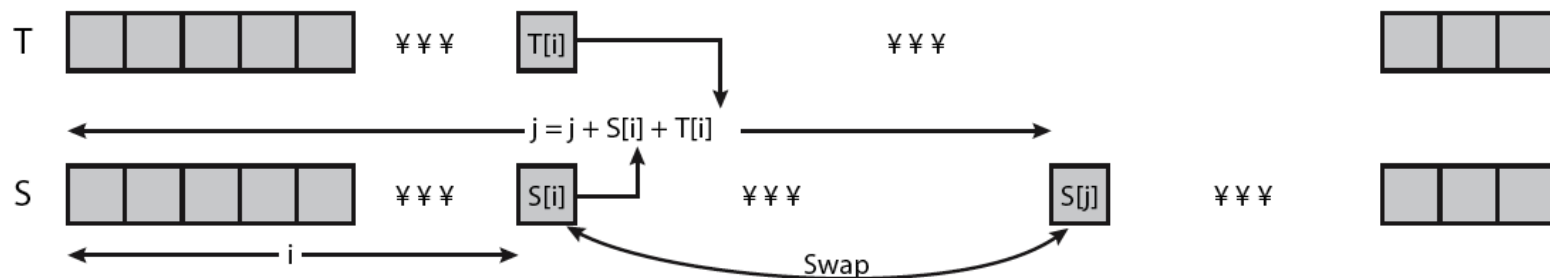
$t = (S[i] + S[j]) \pmod{256}$

$C_i = M_i \text{ XOR } S[t]$

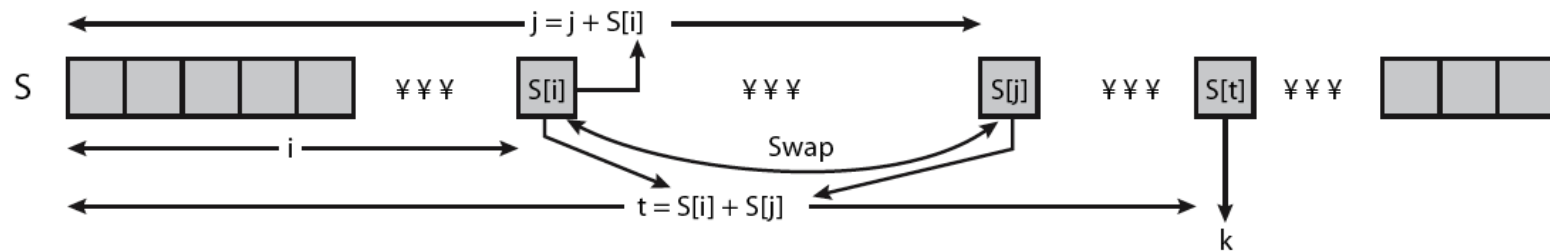
RC4 Overview



(a) Initial state of S and T



(b) Initial permutation of S



(c) Stream Generation

RC4 Security”

- claimed secure against known attacks
 - have some analyses, none practical
- result is very non-linear
- since RC4 is a stream cipher, must **never reuse a key**
- have a concern with WEP, but due to key handling rather than RC4 itself

AES ENCRYPTION Origins

- clear a replacement for DES was needed
 - have theoretical attacks that can break it
 - have demonstrated exhaustive key search attacks
- can use Triple-DES – but slow, has small blocks
- US NIST issued call for ciphers in 1997
- 15 candidates accepted in Jun 98
- 5 were shortlisted in Aug-99
- Rijndael was selected as the AES in Oct-2000
- issued as FIPS PUB 197 standard in Nov-2001

AES Requirements

- private key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- stronger & faster than Triple-DES
- active life of 20-30 years (+ archival use)
- provide full specification & design details
- both C & Java implementations
- NIST have released all submissions & unclassified analyses

AES Evaluation Criteria

- initial criteria:
 - security – effort for practical cryptanalysis
 - cost – in terms of computational efficiency
 - algorithm & implementation characteristics
- final criteria
 - general security
 - ease of software & hardware implementation
 - implementation attacks
 - flexibility (in en/decrypt, keying, other factors)

AES Shortlist

- after testing and evaluation, shortlist in Aug-99:
 - MARS (IBM) - complex, fast, high security margin
 - RC6 (USA) - v. simple, v. fast, low security margin
 - Rijndael (Belgium) - clean, fast, good security margin
 - Serpent (Euro) - slow, clean, v. high security margin
 - Twofish (USA) - complex, v. fast, high security margin
- then subject to further analysis & comment
- saw contrast between algorithms with
 - few complex rounds verses many simple rounds
 - which refined existing ciphers verses new proposals

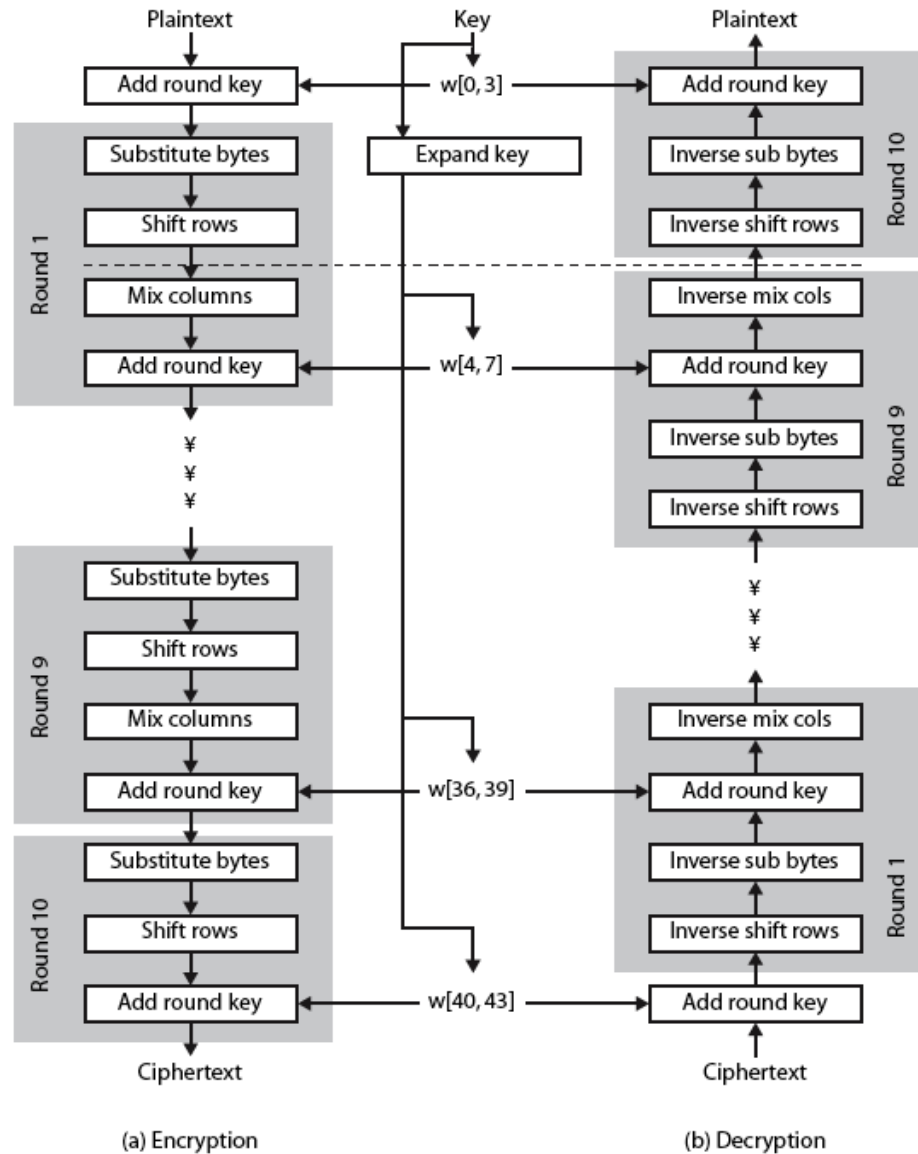
The AES Cipher - Rijndael

- 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
 - speed and code compactness on many CPUs
 - design simplicity

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 groups/columns)
 - mix columns (subs using matrix multiply of groups)
 - add round key (XOR state with key material)
 - view as alternating XOR key & scramble data bytes
- initial XOR key material & incomplete last round
- with fast XOR & table lookup implementation

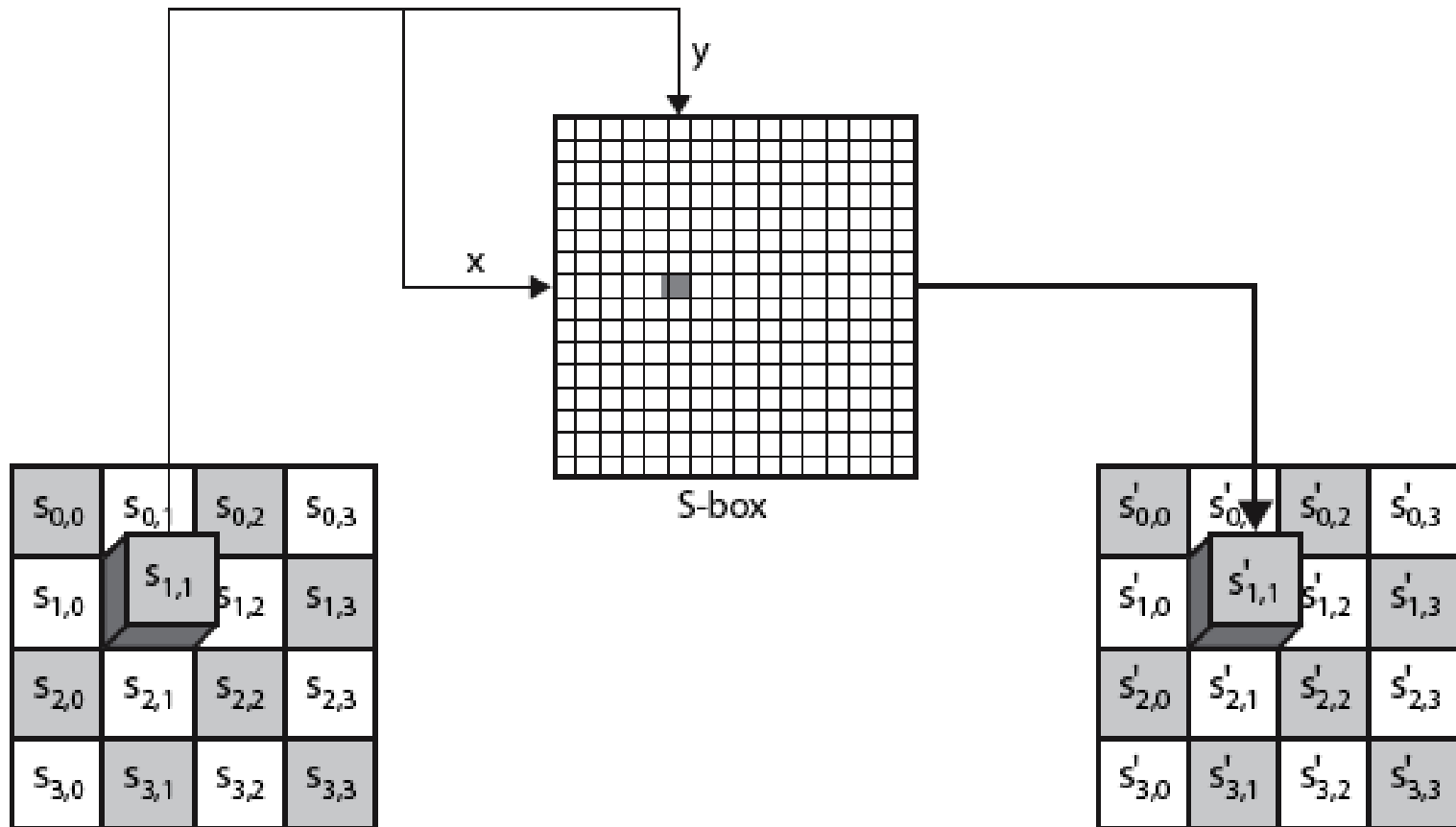
Rijndael



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

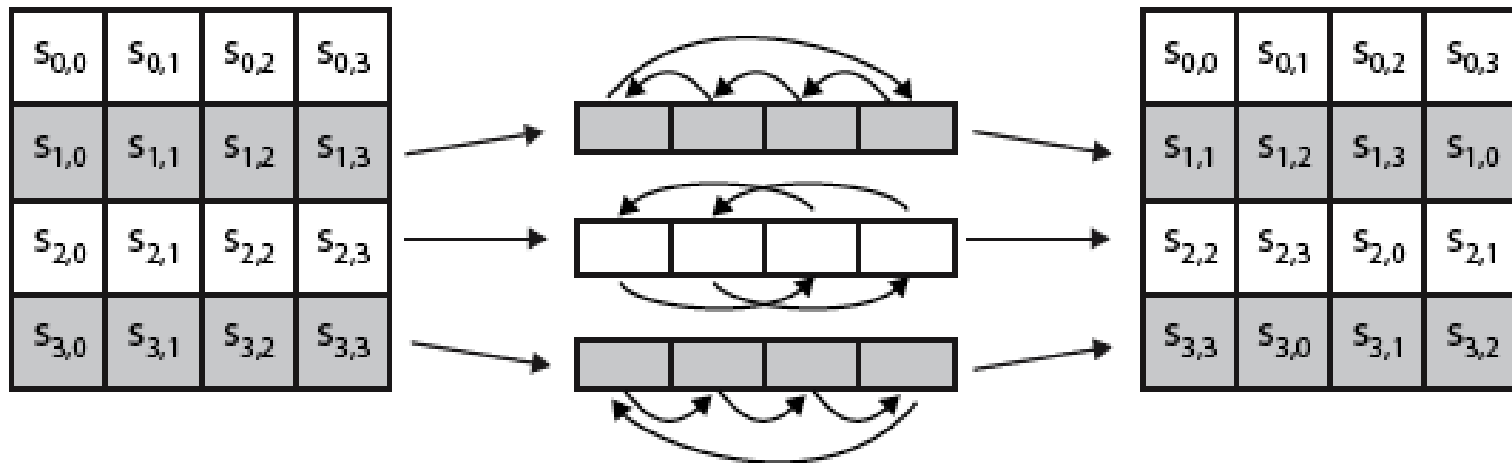
Byte Substitution



Shift Rows

- a circular byte shift in each row
 - 1st row is unchanged
 - 2nd 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

Shift Rows

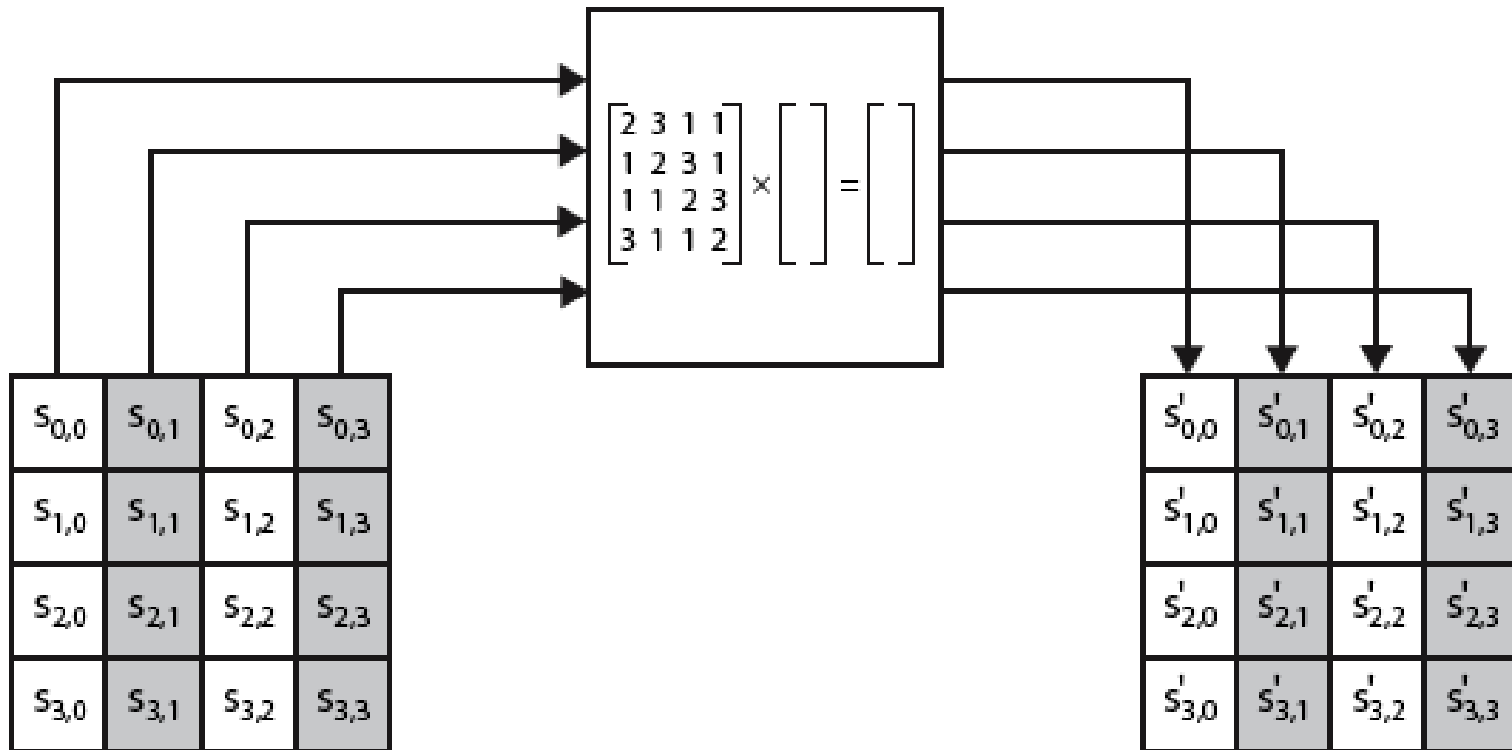


Mix Columns

- 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 $m(x) = x^8 + x^4 + x^3 + x + 1$

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

Mix Columns



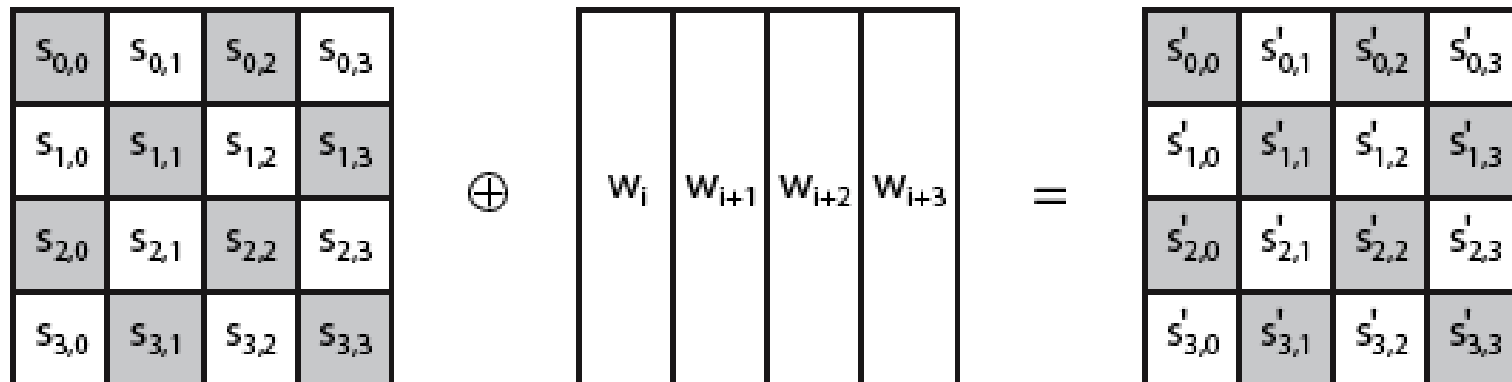
Mix Columns

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

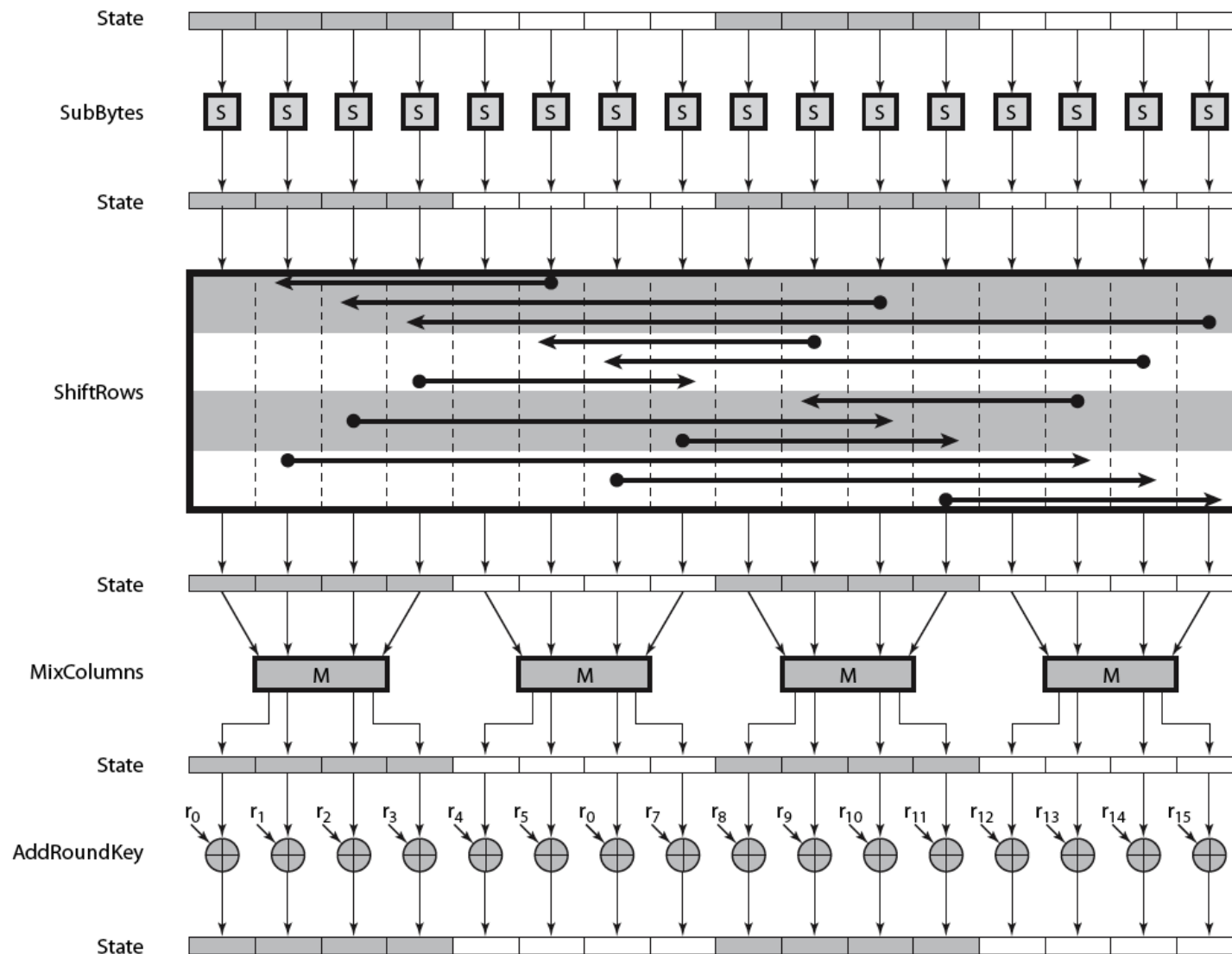
Add Round Key

- 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 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



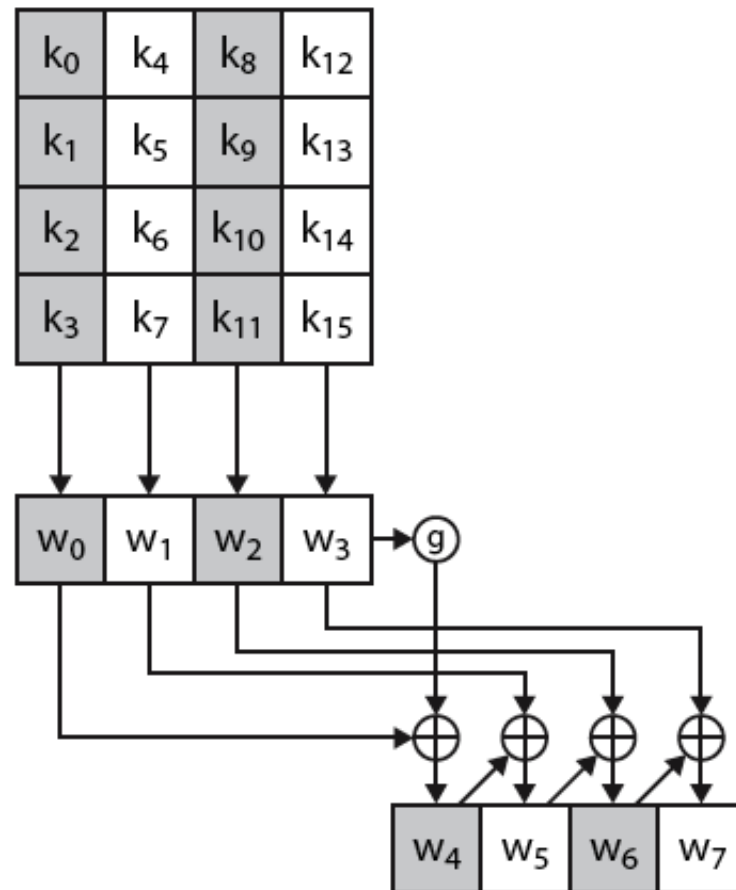
AES Round



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
 - 1st word in 4 has rotate + S-box + XOR round constant on previous, before XOR 4th back

AES Key Expansion



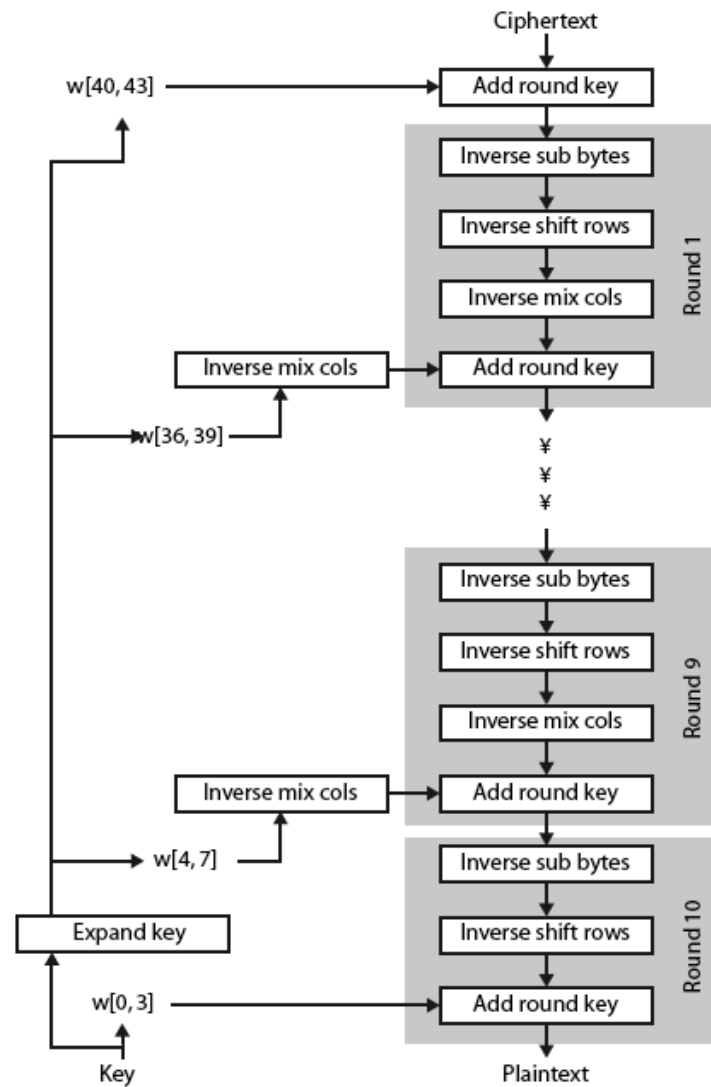
Key Expansion Rationale

- designed to resist known attacks
- design criteria included
 - knowing part key insufficient to find many more
 - invertible transformation
 - fast on wide range of CPU's
 - 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
 - but using 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

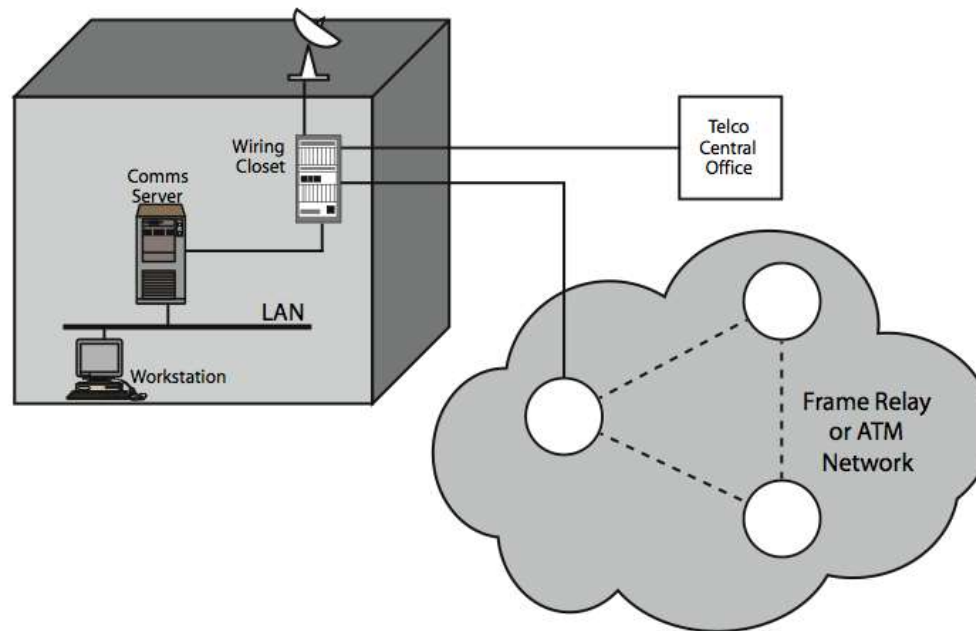
- can efficiently implement on 8-bit CPU
 - byte substitution works on bytes using a table of 256 entries
 - shift rows is simple byte shift
 - add round key works on byte XOR's
 - mix columns requires matrix multiply in $GF(2^8)$ which works on byte values, can be simplified to use table lookups & byte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

Implementation Aspects

- can efficiently implement on 32-bit CPU
 - redefine steps to use 32-bit words
 - can precompute 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

Confidentiality using Symmetric Encryption

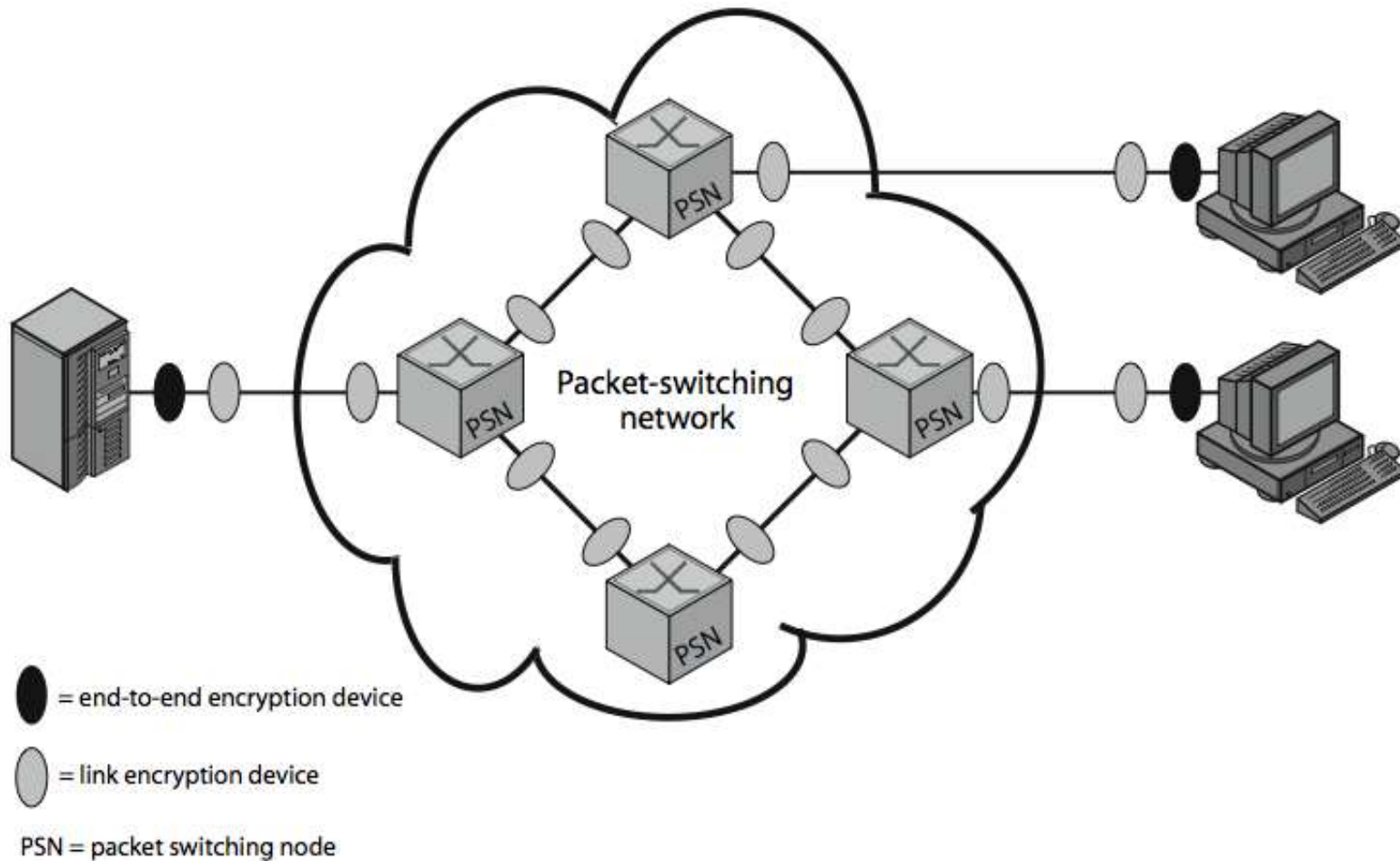
- traditionally symmetric encryption is used to provide message confidentiality



Placement of Encryption

- have two major placement alternatives
- **link encryption**
 - encryption occurs independently on every link
 - implies must decrypt traffic between links
 - requires many devices, but paired keys
- **end-to-end encryption**
 - encryption occurs between original source and final destination
 - need devices at each end with shared keys

Placement of Encryption



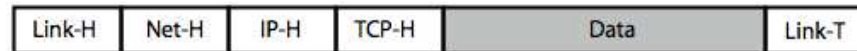
Placement of Encryption

- when using end-to-end encryption must leave headers in clear
 - so network can correctly route information
- hence although contents protected, traffic pattern flows are not
- ideally want both at once
 - end-to-end protects data contents over entire path and provides authentication
 - link protects traffic flows from monitoring

Placement of Encryption

- can place encryption function at various layers in OSI Reference Model
 - link encryption occurs at layers 1 or 2
 - end-to-end can occur at layers 3, 4, 6, 7
 - as move higher less information is encrypted but it is more secure though more complex with more entities and keys

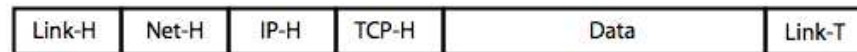
Encryption vs Protocol Level



(a) Application-Level Encryption (on links and at routers and gateways)



On links and at routers

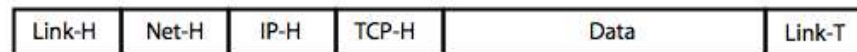


In gateways

(b) TCP-Level Encryption



On links



In routers and gateways

(c) Link-Level Encryption

Shading indicates encryption.

TCP-H = TCP header
 IP-H = IP header
 Net-H = Network-level header (e.g., X.25 packet header, LLC header)
 Link-H = Data link control protocol header
 Link-T = Data link control protocol trailer

Traffic Analysis

- is monitoring of communications flows between parties
 - useful both in military & commercial spheres
 - can also be used to create a covert channel
- link encryption obscures header details
 - but overall traffic volumes in networks and at end-points is still visible
- traffic padding can further obscure flows
 - but at cost of continuous traffic

Key Distribution

- symmetric schemes require both parties to share a common secret key
- issue is how to securely distribute this key
- often secure system failure due to a break in the key distribution scheme

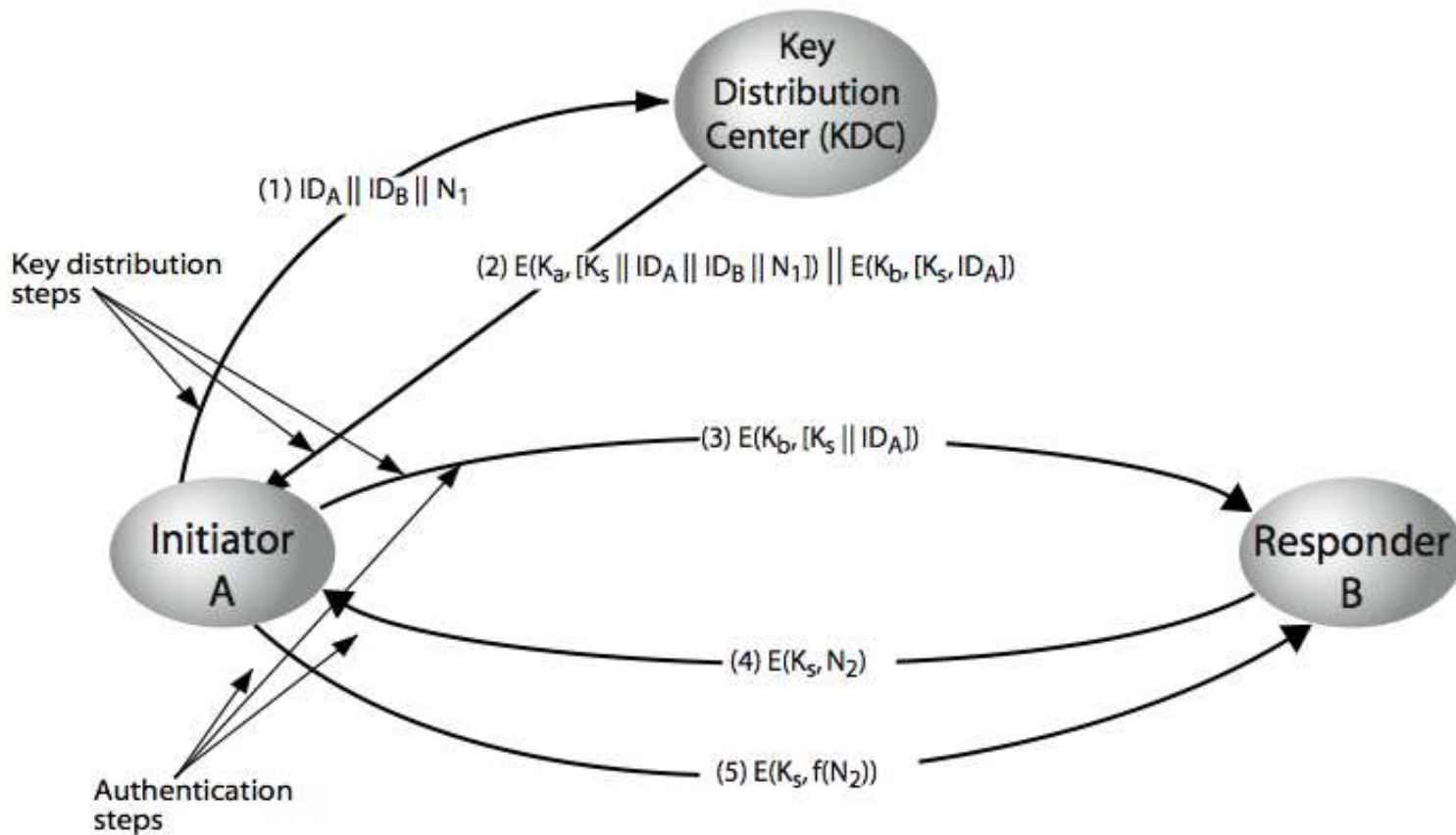
Key Distribution

- given parties A and B have various **key distribution** alternatives:
 1. A can select key and physically deliver to B
 2. third party can select & deliver key to A & B
 3. if A & B have communicated previously can use previous key to encrypt a new key
 4. if A & B have secure communications with a third party C, C can relay key between A & B

Key Hierarchy

- typically have a hierarchy of keys
- session key
 - temporary key
 - used for encryption of data between users
 - for one logical session then discarded
- master key
 - used to encrypt session keys
 - shared by user & key distribution center

Key Distribution Scenario



Key Distribution Issues

- hierarchies of KDC's required for large networks, but must trust each other
- session key lifetimes should be limited for greater security
- use of automatic key distribution on behalf of users, but must trust system
- use of decentralized key distribution
- controlling key usage