Middleware and Distributed Systems

Security

Martin v. Löwis

Introduction

- Threat model: shared resources need to be protected against adversaries
- Security Policy: specification defining what operations are on the resources are acceptable
  - Often declared through *access control*
- Security Mechanism: procedure/infrastructure to enforce a security policy
- Design process
- Cryptography: art of encoding information so that only a designated recipient can understand it; distinct from security

Cryptography

- based in military applications (esp. intelligence and counter-intelligence)
- recently also used for other parts of life (esp. industry)
    - opening of cryptography caused better understanding of the concept, more uniform terminology (e.g. usage of common names Alice, Bob, Carol, Dave, Eve, Mallory, Sara
- Literature
    - Schneier, Applied Cryptography
    - Anderson, Security Engineering

Threats and Attacks

- Threats
  - Leakage: acquisition of information by unauthorized recipients
  - Tampering: unauthorized alteration of information
  - Vandalism: inference with proper operation of a system without gain for the perpetrator
- Attacks depend on gaining access to a channel or a node
  - Eavesdropping: obtaining copies of messages without authority
  - Masquerading: sending or receiving messages using the identity of another participant
  - Replaying: storing intercepted messages and sending them later
  - Denial of service: flooding a channel or node so that access to others is denied

Design process for secure systems

- Assume for the worst
  - interfaces are exposed to attackers
  - networks are insecure
  - algorithms are available to attackers
  - attackers may have access to large resources
- Define policies
- Define list of threats
- Specify how each threat is prevented through mechanism built into the system
  - ideally: formally proof properties
- employ auditing

Cryptography

- Only consider "advanced" techniques here (e.g. no substitution algorithms)
  - Algorithm should have a secret key as its parameter (K_A, K_B)
  - Encryption E(K, M), decryption D(K, M)
- Shared-key algorithms: K_{AB} used both for encryption and decryption
- public/private key algorithms: each participant has a pair of keys
- Applications of cryptography:
  - Secrecy and integrity of messages
  - Authentication
  - Digital Signatures

Secrecy and Integrity

- Secrecy: Alice sends Bob E(K, M); Bob applies D(K, E(K, M))
  - Problem: How can Alice and Bob exchange the key securely? (*key exchange*)
  - Problem: How can Bob know that Alice just send the message, as Mallory might have captured a message and replayed it? (*replay attack*)
- Integrity: may just use encryption; if D yields a meaningful result, it is authentic
  - better: use Message Authentity Codes MAC(M):
    - Alice sends encrypted MAC along with the message
    - Bob decrypts received MAC, computed MAC(M), and compares them

Authentication

- Needham, Schroeder (1978): Using Encryption for Authentication in Large Networks. Communications of the ACM, Vol. 21, pp. 993-999
- Authentication against a central server: Sara has shared secrets with both Alice and Bob
- Alice sends a plain-text message to Sara requesting a ticket for authentication to Bob
- Sara generates a new random secret key K_{AB}, and puts $E(K_B, K_{AB})$ into the ticket
- Sara sends to Alice: $E(K_A, \text{ticket}+K_{AB})$
- Alice decrypts the message, and extracts $E(K_B, \text{ticket})$ and K_{AB}
- Alice sends to Bob: ticket, $E(K_{AB}, M)$
- Bob decrypts the ticket, learns that it is from Sara, retrieves KAB, and decodes the message

Authentication (2)

- Needham/Schroeder algorithm uses concept of a *challenge*: Alice can only use the ticket for B if she really possesses K_A
  - sending her password to Sara is not necessary for authentication
- Problem with that algorithm: a central server is needed which shares a secret key with each user
  - Problem later solved through public/private key cryptography

Digital Signatures

- Scenario 1: Bob wants to make sure the message really originates from Alice
  - Can use MAC as discussed earlier
  - MAC is sometimes also called *message digest*
- Scenario 2: Bob wants to prove to Carol that the message is from Alice
  - Cannot use shared keys anymore, since Bob would need to reveal the key to Carol
  - Solution 1: Alice provides K_{Apub} to Bob in advance, then sends M, E(KApriv, digest(M)); Bob and Carol both decrypt the digest and verify it
    - Problem: How can Carol be sure about K_{Apub}?
  - Solution 2: Digital Certificates

Digital Certificates and Digital Signatures

- Dave, an authority, publishes his public key K_{Dpub}
- Alice identifies herself somehow to Dave, and simultaneously provides K_{Apub}
- Dave returns to Alice $C=E(K_{Dpriv}, K_{Apub})$
- Alice sends C, M, $E(K_{Apriv}, digest(M))$ to Bob
- Bob and Carol decrypt C with K_{Dpub}, obtain *certified* K_{Apub}, then decrypt the digest with K_{Apub}, and compare it with digest(M)

Access Control

- Protection domain: List of <resource, right> pairs given to a set of processes in a distributed system
  - typically established by a principal authenticating to the system, then processes acting on behalf of the principal
  - typically implemented through *capabilities* or *access control lists*
    - variations: role-bases access control (principals act in roles, and gain access based on their roles)

Capabilities

- tokens that enumerate the operations that a process may perform
  - similar to physical keys in the real world
- need to be unforgable in a distributed system
- client passes capability along with the request; server verifies it and performs the operation
- problem 1: key theft
  - may try to revoke capability when it is reported stolen
  - partial solution: include the holder in the capability
- problem 2: capability revocation
  - need to communicate to servers to "exchange the locks"
  - partial solution: add timeout (end of validity) to capability

Access Control Lists

- add a list of <principal, operation> pairs to each resource
  - several variations, e.g. groups of principals, separate allow and deny entries, ...
- problem: assumes that principals can be reliably authenticated

Credentials

- evidence provided by principal when requesting access to a resource
  - certificates, passwords, physical tokens, ...
- *speaks-for* relationship: possession of credentials allows a principal to speak for another one
- delegation: passing of credentials from one process to another, to allow the other process to speak for the principal
  - typically limited by permitted operations and by time

Cryptographic Algorithms

- Cryptoanalysis: known ciphertext, known plaintext, chosen plaintext; differential analysis (similar input data), related key analysis (similar keys)
  - algorithm considered *broken* if a better-than-brute-force attack is known
- Symmetric vs. asymmetric algorithms
- block ciphers: algorithms often operate on fixed-size blocks (e.g. 64 bits)
  - threat: attacker might recognize patterns, perform known plaintext analysis
- cipher block chaining: cipherblock i is XOR'ed with plaintext block i+1 before encryption
  - repeated plaintext data will not result in same cipertext anymore
  - threat: first encrypted block in a communication just based on plaintext

Cryptographic Algorithms (2)

- Stream ciphers: encrypting blocks of data might be inappropriate if data need to be transmitted quickly after the get produced
  - e.g. live AV data
  - encryption must encrypt bit-per-bit
  - solution: keystream generator produces a stream of key bits based on some initial state
- Quality of algorithm: diffusion and confusion (Shannon)
  - confusion: make output look different from input (e.g. combining multiple input bits into one)
  - diffusion: dissipate regular patterns in the input, to make output look "random" (Avalanche Effect)

Shared-key (symmetric) algorithms

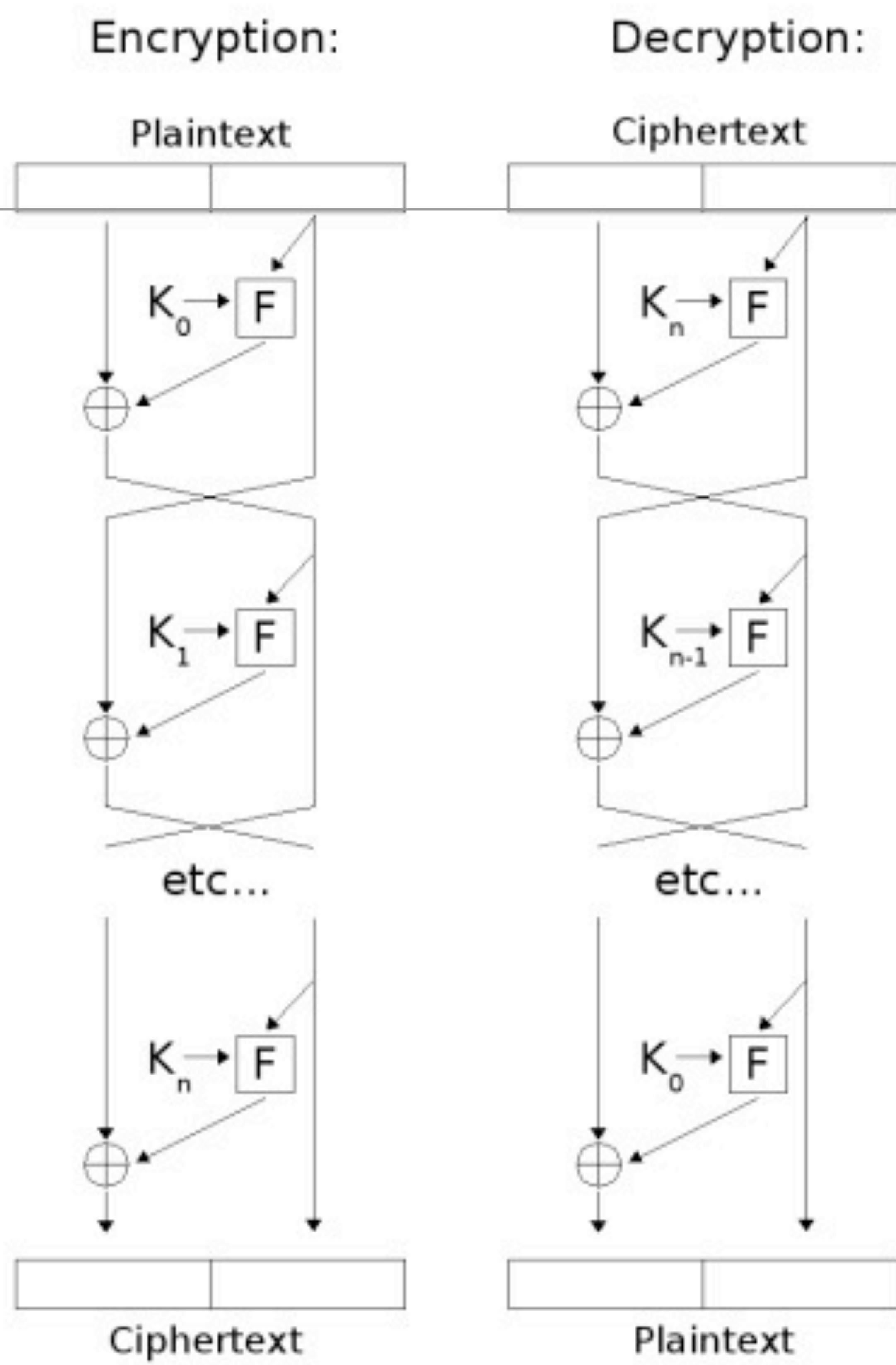
- TEA (Tiny Encryption Algorithm): Wheeler and Needham 1994
  - mainly for educational usage
  - 128 bit keys (4x32), 64 bit blocksize
- DES (Data Encryption Standard): U.S. National Bureau of Standards 1977
  - originally by IBM
  - 56-bit key, 64 bit blocksize
  - designed for efficient implementation in hardware
- IDEA (International Data Encryption Algorithm): Lai and Massey 1990
  - developed as successor to DES; 128-bit keys

Shared-key algorithms (2)

- RC4: Rivest 1992
  - variable-length keys up to 256 bytes
  - allows for efficient implementation in software, used in 802.11
- AES (Advanced Encryption Standard): Daemen and Rijmen 2000
  - submitted to U.S. NIST under the name Rijndael
  - AES: block length 128 bits, key lengths 128, 192, 256
    - usable for U.S. SECRET data (TOP SECRET requires keys >= 192 bits)
  - Rijndael: block and key length multiple of 32, between 128 and 256

TEA

- uses integer addition, XOR and shift for diffusion and confusion
  - bit shuffling: P-boxes (permutation)
  - non-linear functions: S-boxes (substitution)
- Feistel network:
  - encryption and decryption are very similar (reverse key schedule)
  - product cipher (output is product of several rounds)
- 32 rounds
- each round takes as input the two 32-bit parts of the text, and combines them with the 4 32-bit parts of the key and with each other
- delta added to obscure key



Source: Wikipedia

Feistel Cipher

TEA: Encryption

```
void encipher(unsigned long *const v,unsigned long *const w,
   const unsigned long *const k)
{
   register unsigned long       y=v[0],z=v[1],sum=0,delta=0x9E3779B9,
                                a=k[0],b=k[1],c=k[2],d=k[3],n=32;

   while(n-->0)
      {
      sum += delta;
      y += (z << 4)+a ^ z+sum ^ (z >> 5)+b;
      z += (y << 4)+c ^ y+sum ^ (y >> 5)+d;
      }

   w[0]=y; w[1]=z;
}
```

TEA: Decryption

```
void decipher(unsigned long *const v,unsigned long *const w,
   const unsigned long *const k)
{
   register unsigned long      y=v[0],z=v[1],sum=0xC6EF3720,
                               delta=0x9E3779B9,a=k[0],b=k[1],
                               c=k[2],d=k[3],n=32;

   /* sum = delta<<5, in general sum = delta * n */

   while(n-->0)
      {
      z -= (y << 4)+c ^ y+sum ^ (y >> 5)+d;
      y -= (z << 4)+a ^ z+sum ^ (z >> 5)+b;
      sum -= delta;
      }

   w[0]=y; w[1]=z;
}
```

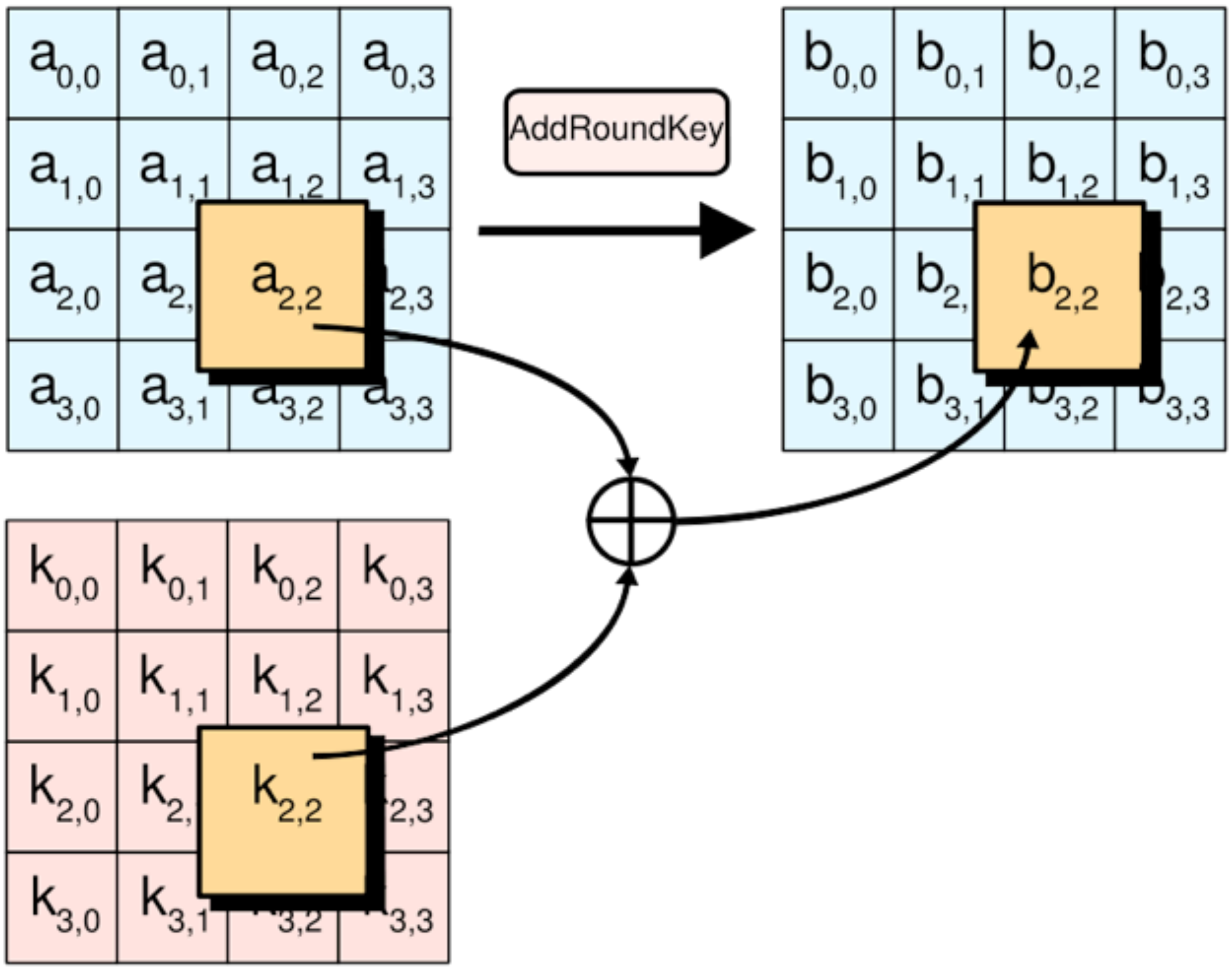
TEA weaknesses

- key equivalence: symmetric usage of key causes certain 4-tuples of keys to be equivalent
  - effective key size is only 126
- related key attacks: similar keys lead to similar output
  - need 2^{23} chosen plaintexts for successful key discovery

AES

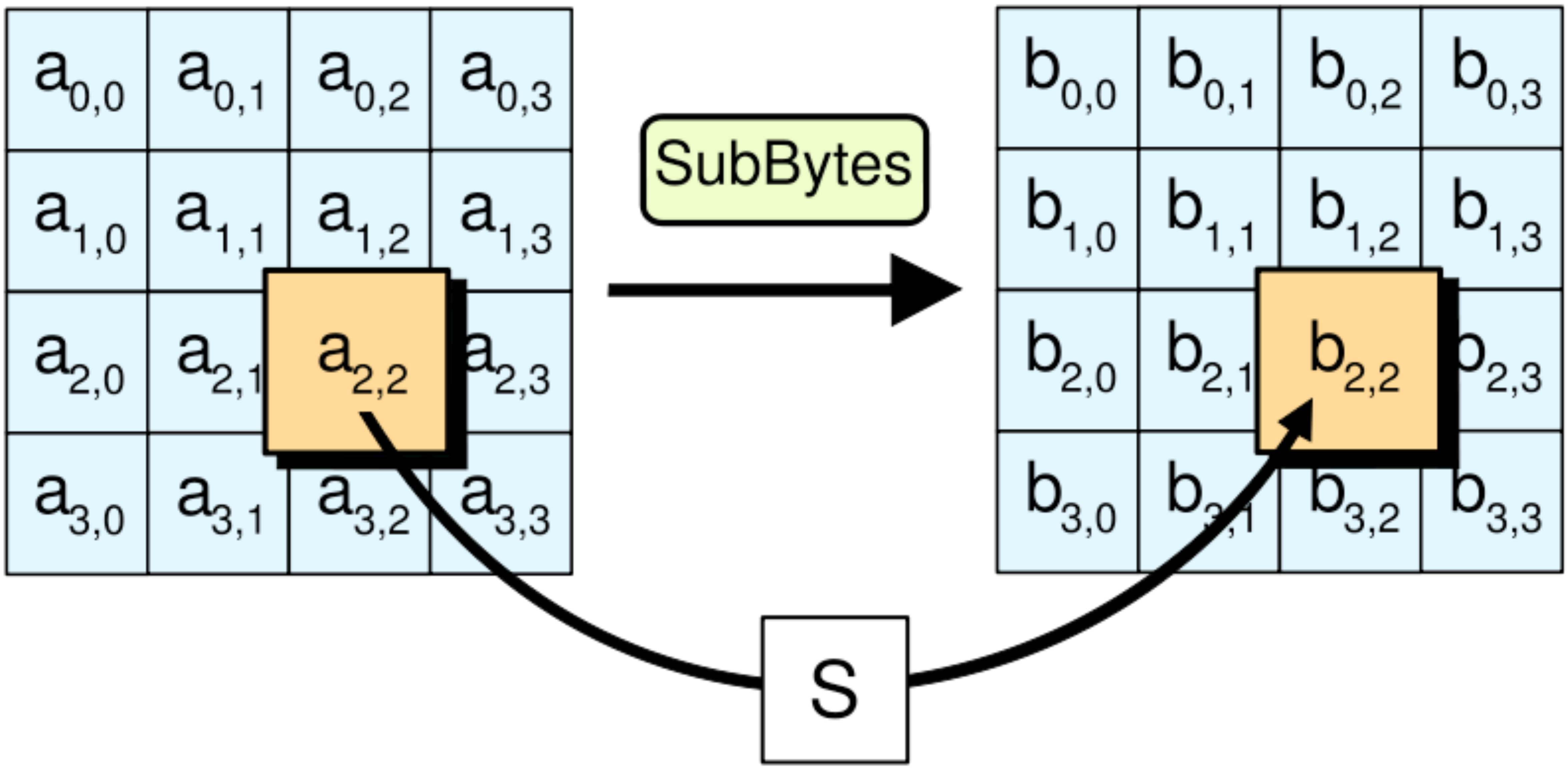
- general substitution-permutation network
- key schedule: generate round keys from encryption key, expanding it to block size
- each round has four steps:
    - AddRoundKey: combine 4x4 bytes with round key
    - SubBytes: substitute each byte with another one according to a specified table
    - ShiftRows: shift each row of the table somewhat
    - MixColumns: apply a linear transformation on each column

AES: AddRoundKey

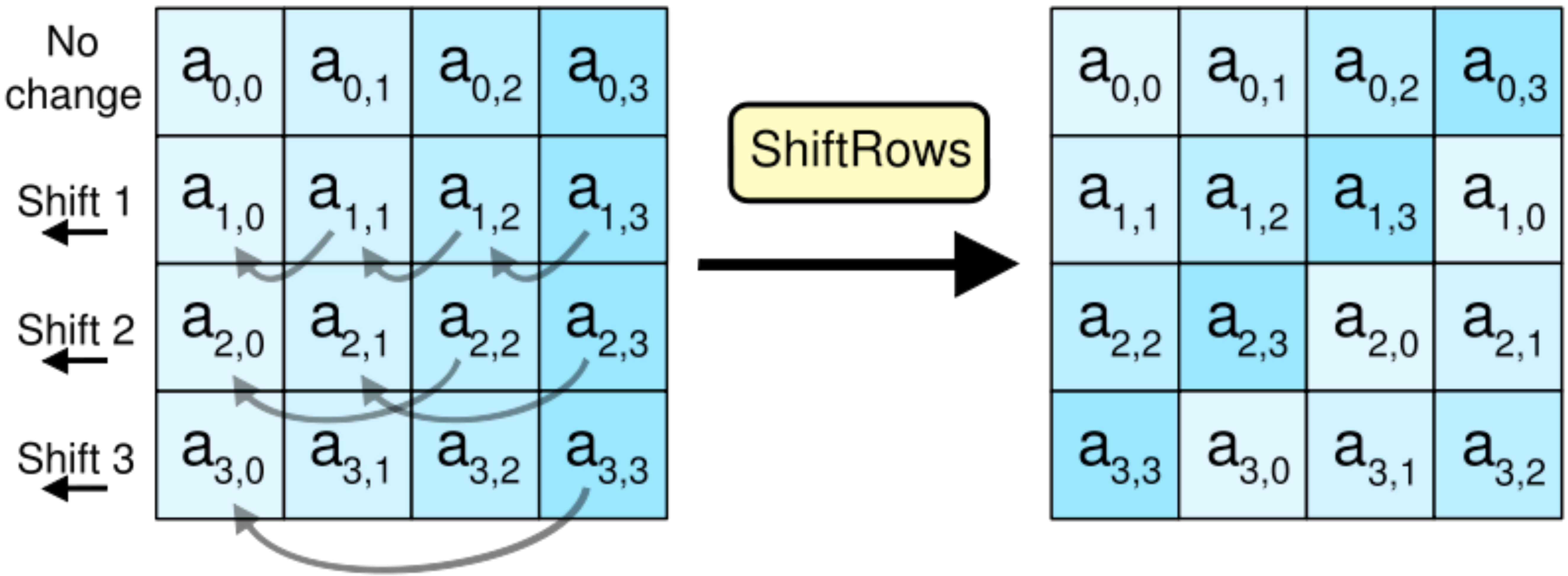


Source: Wikipedia

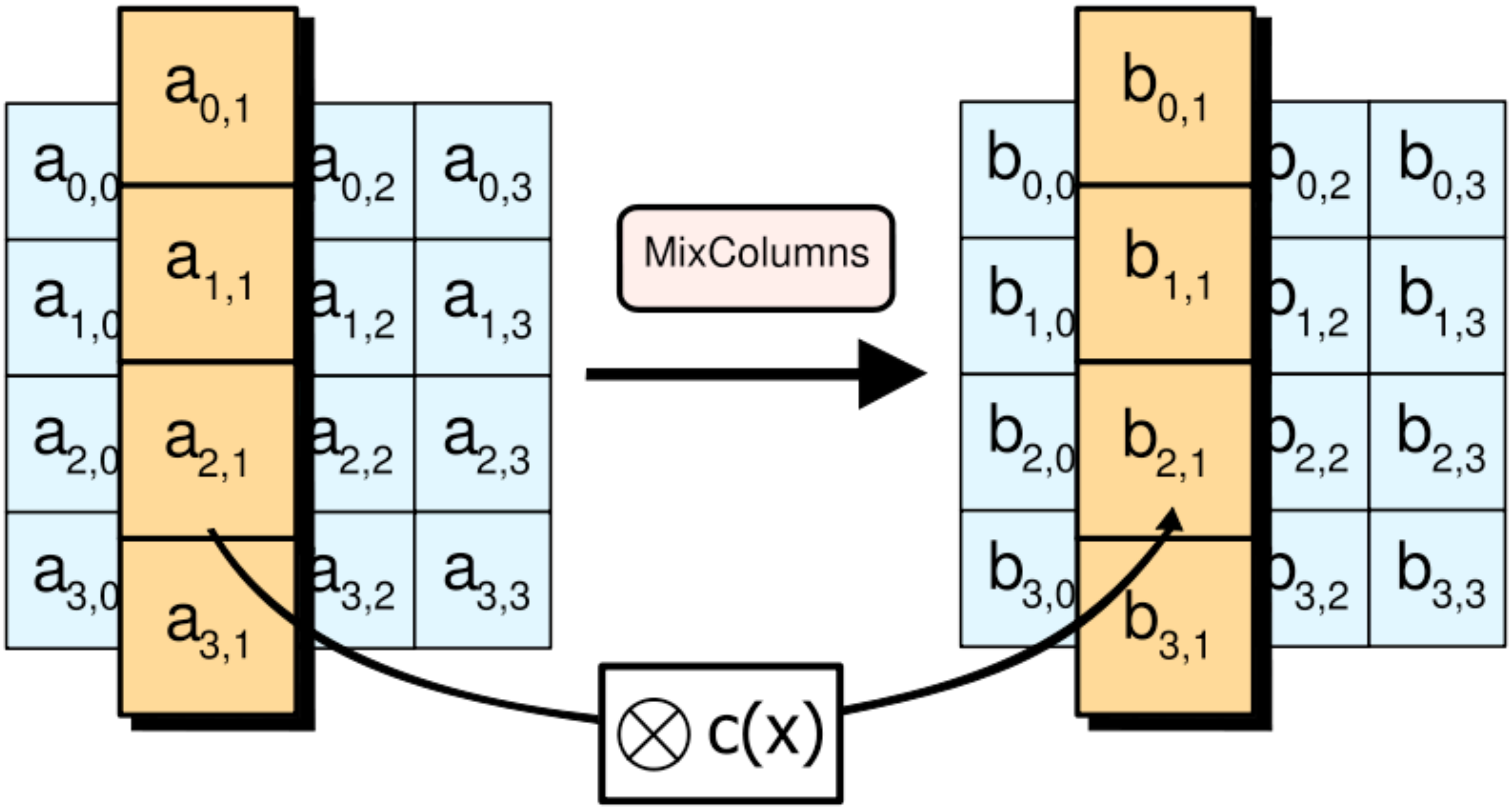
AES: SubBytes



AES: ShiftRows



AES: MixColumns



Public Key (asymmetric) Algorithms

- Key pair: K_e, K_d so that $D(K_d, E(K_e, M)) == M$
  - K_e made known public; K_d kept secret
- Assumption: Derivation of K_d from K_e is computationally expensive; generation of a new pair K_d/K_e is not
- RSA (Rivest, Shamir, Adelman), 1978
  - arbitrary key sizes, must generate two large primes
- ElGamal: Taher Elgamal, 1984
  - need to compute a large prime and its generator
- Various algorithms based elliptic curves

RSA

- Choose two large primes P and Q, N = P·Q, Z = (P-1)·(Q-1)
  - Publish N, keep Z secret
- Choose d so that it is relatively prime with Z (lcd(d, Z) = 1)
- Choose e so that $e \cdot d \equiv 1 \pmod{Z}$
  - Compute through extended Euclidean algorithm
- $E(e, M) = M^e \bmod N$
  - Compute through repeated quadration
- $D(d, M) = M^d \bmod N$
  - Idea: $M^{ed} \equiv M \pmod{Z}$
  - Fermat-Euler-Theorem: $M^Z \equiv 1 \pmod{N}$
    - hence $M^{ed} \equiv M^{1+kZ} \equiv M^1 M^{Zk} \equiv M^1 1^k \equiv M (\mathrm{mod}\ N)$

RSA Analysis

- Chosen plain-text attack: Encrypt all messages with the public key until the encrypted message is found
  - needs block size large enough to make this attack infeasible
- Compute private key from public key: Needs to factor N=PQ
  - feasibility depends on efficient factorization algorithm; none is known today
- Key generation: need to test primality of large numbers quickly
  - probabilistic tests: determine whether P, Q are "probable primes"
  - fast deterministic tests: cyclotomy test, elliptic curve primality test, AKS test
- N not a power of two - need padding to achieve bit-oriented block sizes
  - introduce randomized padding to protect better against brute force attacks

AKS Primality Test

- Manindra Agrawal, Neeraj Kayal, Nitin Saxena, 2002: PRIMES is in P
- deterministic primality test, in time polynomial with the bit length of candidate
- History:
  - Agrawal 1999: probabilistic primality test: N is prime, iff, for all a,

$$(x+a)^N \equiv x^N + a \mod N$$

  - Bhattarcharjee, Pandey, 2001 (Bachelor's thesis): study modulus of $x^r - 1$
  - Kayal, Saxena, 2002 (Bachelor's thesis): Assuming Riemann's thesis, prove assumption of Bhattarcharjee, Pandey that if r is a prime and not a factor of N, and $(x+1)^N \equiv x^N + 1 \pmod{x^r - 1, N}$

    then either N is a prime, or $N^2 \equiv 1 \pmod{r}$

AKS Primality Test (2)

- Lenstra/Pomerance optimization of original algorithm

  Find the smallest r so that $O_r(n) > ld^2(n)$

  $O_r(n)$: smallest k, so that $n^k = 1 \pmod r$

  ld: binary logarithm

  for a = 1 to r:

  if gcd(a, n) != 1: output *composite*

  for a = 1 to $\lfloor 2\sqrt{r}\log(n)\rfloor + 1$:

  if $(x + a)^n \equiv x^n + a \pmod{x^r - 1, n}$: output *prime*

Key Agreement

- Objective: Alice and Bob want to agree on a shared key
  - Eve must not find out what the key is
- Solution 1: use asymmetric algorithm - Alice generates key, encrypts with Bob's public key
- Solution 2: Diffie-Hellman-Merkle Key Agreement (1976)
  - Alice computes large prime P, primitive root g, chooses private key a
  - Alice sends Bob P, g, $A=g^a \pmod{P}$
  - Bob chooses private key b, sends Alice $B=g^b \pmod{P}$
  - Alice computes $K = B^a = g^{ab} \pmod{P}$
  - Bob computes $K = A^b = g^{ab} \pmod{P}$

Secure Hashing

- fixed-length bit pattern that characterizes a message
  - Digest/Hash function H(M)
  - ideally: collision-free; $M_1 \neq M_2 \Rightarrow H(M_1) \neq H(M_2)$
  - practically: if hash is short than message, there will be collisions
    - collisions should not appear in practice
  - ideally: irreversible (*one-way hash functions*)
- Birthday paradox (birthday attack): for a random sample of

$$\sqrt{2 * ln(2) * N} \approx 1.2\sqrt{N}$$

elements from a total of N elements, there is a 50% probability of duplicates

Secure Hashing (2)

- MD5 (Message Digest 5): Rivest 1992
  - arbitrary-sized input, 128 bit hash
  - broken; vulnerable to suffix attack (if MD5(A)==MD5(B) then for all X, Y MD5(X+A+Y) == MD5(X+B+Y))
- SHA-1 (US Secure Hash Algorithm 1): NIST 1993
  - based on MD4, 160 bit hash
  - assumed broken: Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu report that fewer than 2^{69} operations are necessary to produce collision
- SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512): NIST 2002
  - different size of resulting hash

MD-5

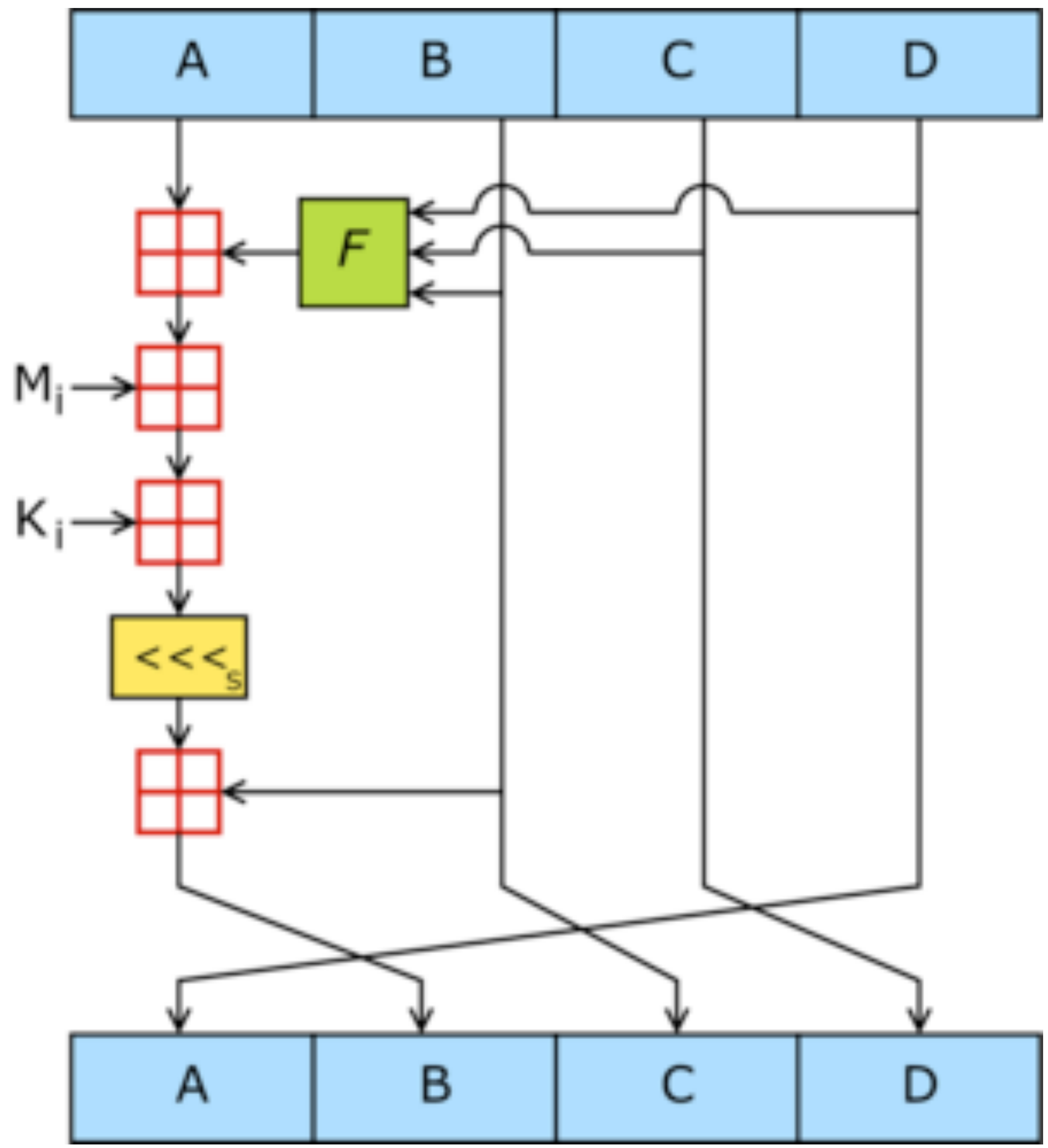
- Input message split into 512-bit chunks; potentially padded
  - padding: last 64 bits specify size of the original message, preceded by zeros, preceded by a single 1-bit, preceded by the original message
  - multiple blocks are fed to algorithm
- state: 4 words
  - A = 01 23 45 67, B = 89 ab cd ef, C = fe dc ba 98, D = 76 54 32 10
- each 32-bit word is processed in four rounds
  - 16 words per chunk -> 64 rounds

MD5 (2)

- 64 constant K_i, one per round (computed from sine values)
- 4 round functions:

$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$

MD5 (3)



Public Key Infrastructure (PKI)

- X.509: Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks, CCITT 1988
  - part of OSI Directory
  - RFC 3280 (obsoletes 2459): Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
- Certificate Authority (CA): issuer/signer of public keys
  - removes need for independent verification of public keys, assuming CA is trusted
  - CA policy: under what conditions are certificates issued? how is the private key of the CA protected against theft? what information about the *subject* will be included?

PKI (2)

- CA hierarchy: individual users don't obtain certificate from a single authority, but *root* CA certifies sub-ordinate CA
  - certificate chain: sequence of certificates leading up to root
- signed information in certificate:
  - subject: who is being certified (distinguished name, public key
  - issuer: what CA issued the certificate (distinguished name, signature)
  - period of validity
  - additional attributes

X.509: Certificate

```
Certificate  ::=  SEQUENCE  {
        tbsCertificate       TBSCertificate,
        signatureAlgorithm   AlgorithmIdentifier,
        signatureValue       BIT STRING  }
```

TBSCertificate

```
TBSCertificate  ::=  SEQUENCE  {
    version         [0]  EXPLICIT Version DEFAULT v1, -- today, always v3
    serialNumber         CertificateSerialNumber,
    signature            AlgorithmIdentifier,
    issuer               Name,
    validity             Validity,
    subject              Name,
    subjectPublicKeyInfo SubjectPublicKeyInfo,
    issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
    subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
    extensions      [3]  EXPLICIT Extensions OPTIONAL
    }

 Version  ::=  INTEGER  {  v1(0), v2(1), v3(2)  }
```

Certificate fields

```
CertificateSerialNumber  ::=  INTEGER
 Validity ::= SEQUENCE {
     notBefore      Time,
     notAfter       Time }
 Time ::= CHOICE {
     utcTime        UTCTime,  -- UNIVERSAL 23
     generalTime    GeneralizedTime }
 UniqueIdentifier  ::=  BIT STRING
 SubjectPublicKeyInfo  ::=  SEQUENCE  {
     algorithm            AlgorithmIdentifier,
     subjectPublicKey     BIT STRING  }
 Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension
```

Algorithms

```
AlgorithmIdentifier  ::=  SEQUENCE  {
    algorithm               OBJECT IDENTIFIER,
    parameters              ANY DEFINED BY algorithm OPTIONAL  }
```

Names

Name ::= CHOICE { RDNSequence }

RDNSequence ::= SEQUENCE OF RelativeDistinguishedName

RelativeDistinguishedName ::= SET OF AttributeTypeAndValue

AttributeTypeAndValue ::= SEQUENCE {

  type   AttributeType,

  value   AttributeValue }

AttributeType ::= OBJECT IDENTIFIER

AttributeValue ::= ANY DEFINED BY AttributeType

Extensions

```
Extension ::= SEQUENCE {
    extnID      OBJECT IDENTIFIER,
    critical    BOOLEAN DEFAULT FALSE,
    extnValue   OCTET STRING }
```

Predefined extensions:

```
id-ce   OBJECT IDENTIFIER ::= { joint-iso-ccitt(2) ds(5) 29 }
```

OIDs for Algorithms

- 1.3.14.3.2.3 md5WithRSA
- 1.3.14.3.2.6 des-ecb
- 1.3.14.3.2.7 des-cbc
- 1.3.14.3.2.13 DSA-SHA
- 1.3.14.3.2.15 RSA-SHA
- 1.3.14.3.2.26 sha1
- 1.2.840.113549.1.1.1 rsaEncryption
- 1.2.840.113549.1.1.4 md5WithRSAEncryption
- 1.2.840.113549.1.1.5 sha1WithRSAEncryption
- 1.2.840.113549.1.1.11 sha256WithRSAEncryption
- 1.2.840.113549.1.1.13 sha512WithRSAEncryption
- 1.3.6.1.4.1.18832.11.3.1 Elliptic-Curve Nyberg-Rueppel with SHA-1 signature

OIDs for Names

2.5.4.3	CN
2.5.4.4	SN
2.5.4.5	serialNumber
2.5.4.6	C
2.5.4.7	L
2.5.4.8	ST
2.5.4.9	streetAddress
2.5.4.10	O
2.5.4.11	OU
2.5.4.72	role
0.9.2342.19200300.100.1.1	userId
1.2.840.113549.1.9.1	emailAddress

OIDs for Extensions

- 2.5.29.14 Subject Key Identifier
- 2.5.29.15 Key Usage
- 2.5.29.17 Subject Alternative Name
- 2.5.29.18 Issuer Alternative Name
- 2.5.29.19 Basic Constraints
- 2.5.29.37 Extended Key Usage
- 2.16.840.1.113730.1.2 Netscape Base Url
- 2.16.840.1.113730.1.3 Netscape Revocation Url
- 2.16.840.1.113730.1.13 Netscape Comment
- 1.3.6.1.4.1.311.20.2 Microsoft Certificate Type

Key Usage Extension

```
id-ce-keyUsage OBJECT IDENTIFIER ::=  { id-ce 15 }

 KeyUsage ::= BIT STRING {
      digitalSignature        (0),
      nonRepudiation          (1),
      keyEncipherment         (2),
      dataEncipherment        (3),
      keyAgreement            (4),
      keyCertSign             (5),
      cRLSign                 (6),
      encipherOnly            (7),
      decipherOnly            (8) }
```

Subject Alternative Name Extension

```
GeneralNames ::= SEQUENCE SIZE (1..MAX) OF GeneralName
GeneralName ::= CHOICE {
    otherName                   [0]  OtherName,
    rfc822Name                  [1]  IA5String,
    dNSName                     [2]  IA5String,
    x400Address                 [3]  ORAddress,
    directoryName               [4]  Name,
    ediPartyName                [5]  EDIPartyName,
    uniformResourceIdentifier   [6]  IA5String,
    iPAddress                   [7]  OCTET STRING,
    registeredID                [8]  OBJECT IDENTIFIER }
 OtherName ::= SEQUENCE {
    type-id   OBJECT IDENTIFIER,
    value     [0] EXPLICIT ANY DEFINED BY type-id }
```

Extended Key Usage Extension

- ExtKeyUsageSyntax ::= SEQUENCE SIZE (1..MAX) OF KeyPurposeId
- KeyPurposeId ::= OBJECT IDENTIFIER
- 1.3.6.1.5.5.7.3.1 Server Authentication
- 1.3.6.1.5.5.7.3.2 Client Authentication
- 1.3.6.1.5.5.7.3.3 Code Signing
- 1.3.6.1.5.5.7.3.4 Email Protection
- 1.3.6.1.5.5.7.3.8 Time Stamping
- 1.3.6.1.5.5.7.3.9 OCSP Signing
- 1.3.6.1.4.1.311.10.3.4 Microsoft Encrypting File System

Transport Layer Security

- RFC 5246 (TLS version 1.2)
- goal: provide privacy and data integrity, interoperable, extensible, efficient
- two layers: record protocol, handshake protocol
- record protocol:
  - private connection (DES, RC4, ...)
  - reliable transport (SHA, MD5, ...)
- handshake protocol: allow client and server to authenticate to each other, using asymmetric algorithms
  - one peer does not need to authenticate
  - negotiate shared secret for communication

Kerberos

- Originally RFC 1510 (Kerberos v5), recently revised in RFC 4120
- based on Needham/Schroeder algorithm
- developed for MIT Project Athena
- KDC: Key Distribution Center
- Realm: Scope of a KDC
- Principal: uniquely named client or server
- Server: principal which provides a resource to clients
- Ticket: record to authenticate a client to a server

Kerberos: Basic Operation

- Client requests TGT (Ticket Granting Ticket) from AS (Authentication service)
  - KRB_AS_REQ and KRB_AS_REP
- Client requests ticket for specific service from TGS (Ticket-Granting Service)
  - KRB_TGS_REQ and KRB_TGS_REP; ticket carries session key
  - client caches all tickets in ticket cache
- Client communicates session key to service
  - KRB_AP_REQ and KRB_AP_REP (only for mutual authentication)
  - Messages include "authenticator": nonce values computed from system time and principal name, signed with session key
    - time stamp prevents replay attacks; server needs replay cache for clock skew
- Client and server exchange KRB_PRIV and KRB_SAFE messages

Kerberos: Tickets

```
Ticket ::=     [APPLICATION 1] SEQUENCE {
               tkt-vno[0]                     INTEGER, -- 5
               realm[1]                       Realm,
               sname[2]                       PrincipalName,
               enc-part[3]                    EncryptedData
}
Realm ::=           GeneralString
PrincipalName ::= SEQUENCE {
        name-type[0]     INTEGER,
        name-string[1]   SEQUENCE OF GeneralString
}
```

Kerberos: Tickets (2)

```
EncTicketPart ::= [APPLICATION 3] SEQUENCE {
                  flags[0]                TicketFlags,
                  key[1]                  EncryptionKey,
                  crealm[2]               Realm,
                  cname[3]                PrincipalName,
                  transited[4]            TransitedEncoding,
                  authtime[5]             KerberosTime,
                  starttime[6]            KerberosTime OPTIONAL,
                  endtime[7]              KerberosTime,
                  renew-till[8]           KerberosTime OPTIONAL,
                  caddr[9]                HostAddresses OPTIONAL,
                  authorization-data[10]   AuthorizationData OPTIONAL
}

KerberosTime ::=   GeneralizedTime
                               -- Specifying UTC time zone (Z)
```

TicketFlags

```
TicketFlags ::=   BIT STRING {
                  reserved(0),
                  forwardable(1),
                  forwarded(2),
                  proxiable(3),
                  proxy(4),
                  may-postdate(5),
                  postdated(6),
                  invalid(7),
                  renewable(8),
                  initial(9),
                  pre-authent(10),
                  hw-authent(11)
                }
```

TicketFlags (2)

- initial: ticket originates from AS
- pre-authenticated: the client has authenticated itself to the KDC
  - required in Active Directory; client needs to encrypt time stamp with password hash
  - designed to prevent offline attacks against the shared secret
- HW-authenticated: the client has authenticated itself using a hardware token
- forwardable/forwarded: TGT can be moved to a different network
- proxiable/proxy: ticket allows the service to act on the principal's behalf
- renewable: ticket can be renewed until renew-till; KDC might check whether it was reported stolen
- may-postdate/postdated: ticket starts validity at a future point in time

Application Programming Interfaces and Layering

- GSSAPI (RFC 2743): Generic Security Service API
  - Attempt to integrate multiple security mechanisms into single API
  - Implies wire protocol for interoperability
  - different mechanisms are not interoperable: Kerberos, NTLM, DCE, SPKM, ...
- CryptoAPI and SSPI (Microsoft): single API for multiple mechanisms
  - CSP: Cryptographic Service Provider
  - offers various cryptographic functions
  - SSPI authentication mechanisms: Schannel (TLS), Kerberos, Negotiate (SPNEGO), NTLM, DIGEST