Computer Security

08. Cryptography – Part II

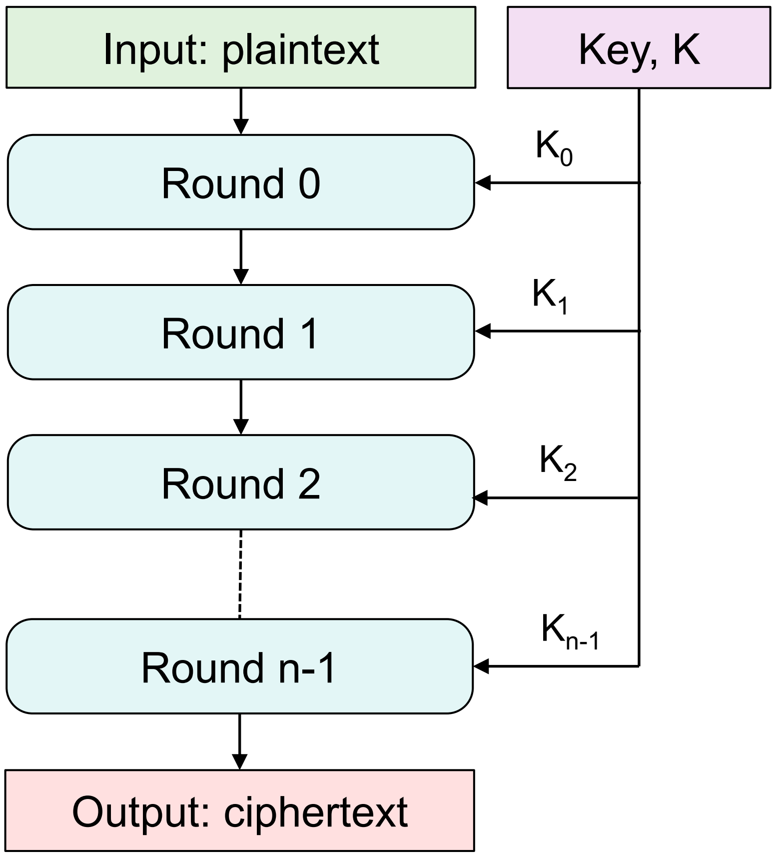
Paul Krzyzanowski

Rutgers University

Spring 2018

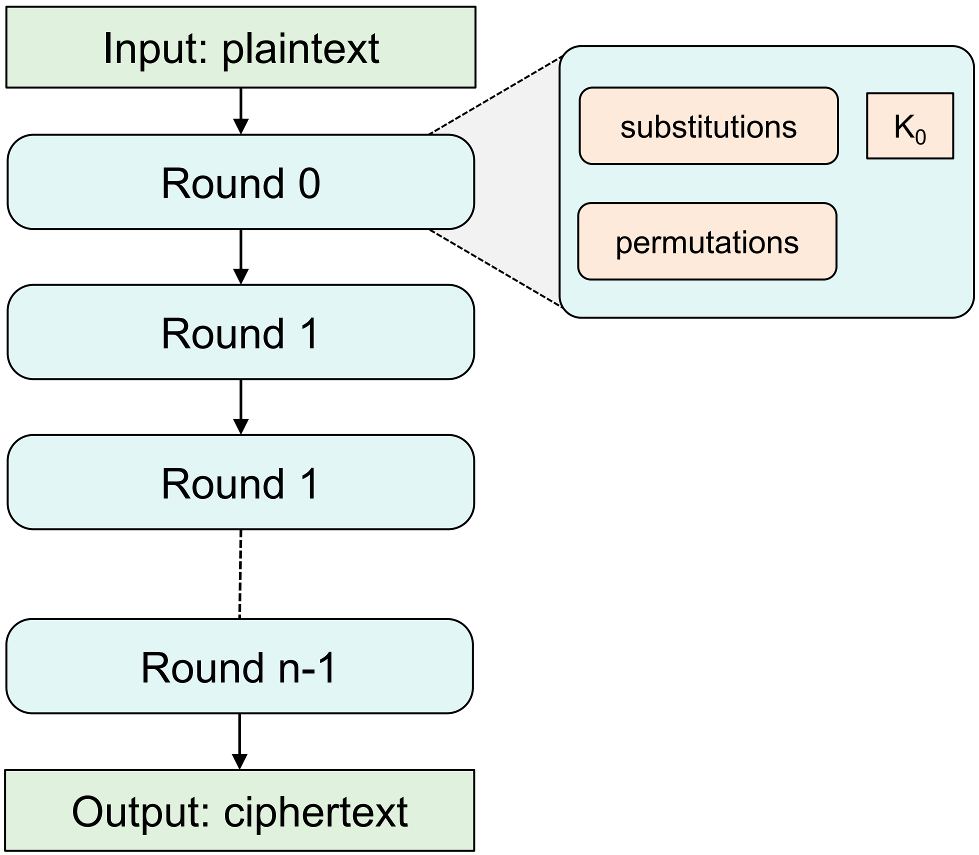
Block ciphers

- Block ciphers encrypt a ***block*** of plaintext at a time and produce ciphertext
- DES & AES are two popular block ciphers
  - DES: 64 bit blocks
  - AES: 128 bit blocks
- Block ciphers are usually iterative ciphers
  - The encryption process is an iteration through several *round* operations



Block cipher rounds

Each round consists of substitutions & permutations



Substitution = **S-box**

- Table lookup
- Converts a small block of input to a block of output
- Changing one bit of input should change approximately ½ of output bits

Permutation

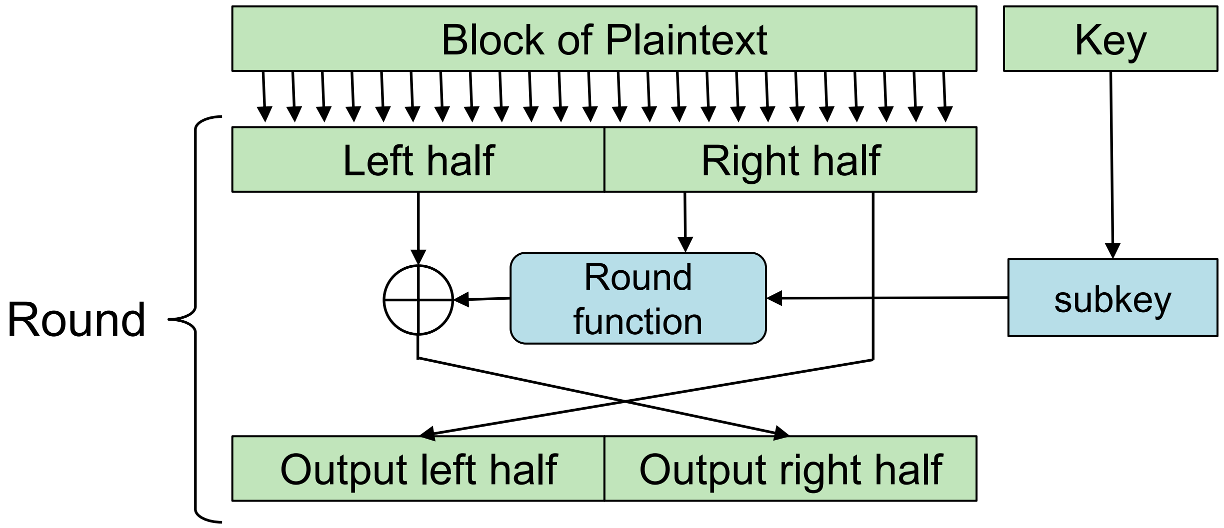
- Scrambles the bits in a prescribed order

Key application per round

- Subkey per round derived from the key
- Can drive behavior of s-boxes
- May be XORed with the output of each round

Feistel cipher

- DES is a type of **Feistel cipher**, which is a form of a **block cipher**
- Plaintext block is split in two
  - Round function applied to one half of the block
  - Output of the round function is XORed with other half of the block
  - Halves are swapped
- AES is *not* a Feistel cipher



AES (Advanced Encryption Standard)

- Block cipher: 128-bit blocks
  - DES used 64-bit blocks
- Successor to DES as a standard encryption algorithm
  - DES: 56-bit key
  - AES: 128, 192, or 256 bit keys

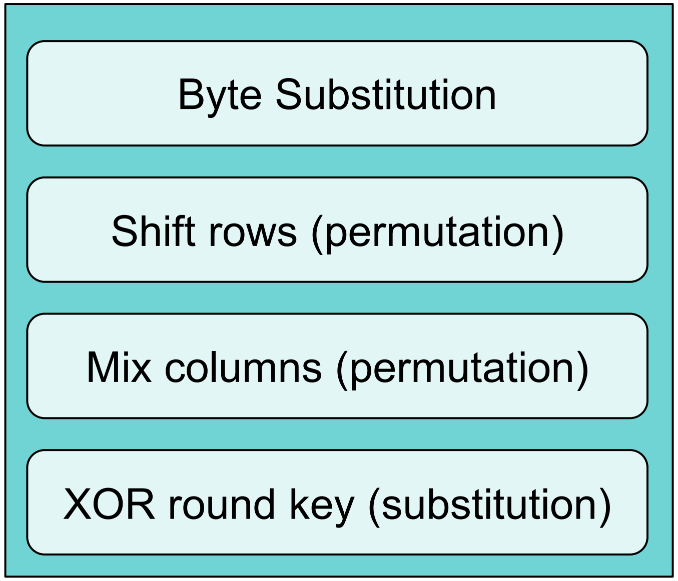
AES (Advanced Encryption Standard)

- Iterative cipher, just like most other block ciphers
  - Each round is a set of substitutions & permutations

- Variable number of rounds
  - DES always used 16 rounds
  - AES:
    - 10 rounds: 128-bit key
    - 12 rounds: 192-bit key
    - 14 rounds: 256-bit key
  - A subkey ("round key") derived from the key is computed for each round
    - DES did this too

Each AES Round

- Step 1: Byte Substitution (s-boxes)
  - Substitute 16 input bytes by looking each one up in a table (S-box)
  - Result is a 4x4 matrix
- Step 2: Shift rows
  - Each row is shifted to the left (wrapping around to the right)
  - 1st row not shifted; 2nd row shifted 1 position to the left;
    3rd row shifted 2 positions; 4th row shifted three positions
- Step 3: Mix columns
  - 4 bytes in each column are transformed
  - This creates a new 4x4 matrix
- Step 4: XOR round key
  - XOR the 128 bits of the round key with the 16 bytes of the matrix in step 3



AES Decryption

Same rounds
… but in reverse order

DES Disadvantages

- DES has been shown to have some weaknesses
  - Key can be recovered using 2^{47} chosen plaintexts or 2^{43} known plaintexts
  - *Note that this is not a practical amount of data to get for a real attack*
- Short block size (8 bytes = 2^8 = 64 bits)
- The real weakness of DES is its 56-bit key
  - Exhaustive search requires 2^{55} iterations on average
- 3DES solves the key size problem: we can have keys up to 168 bits.
  - Differential & linear cryptanalysis is not effective here: the three layers of encryption use 48 rounds instead of 16 making it infeasible to reconstruct s-box activity.
- DES is relatively slow
  - It was designed with hardware encryption in mind: 3DES is 3x slower than DES
  - *Still much faster than RSA public key cryptosystems!*

AES Advantages

- Larger block size: 128 bits vs 64 bits
- Larger & varying key sizes: 128, 192, and 256 bits
  - 128 bits is complex enough to prevent brute-force searches
- No significant academic attacks beyond brute force search
  - Resistant against linear cryptanalysis thanks to bigger S-boxes
    - S-box = lookup table that adds non-linearity to a set of bits via transposition & flipping
  - DES: 6-bit inputs & 4-bit outputs
  - AES: 8-bit inputs & 8-bit outputs
- Typically 5-10x faster in software than 3DES

Attacks against AES

- Attacks have been found
  - This does ***not*** mean that AES is insecure!
- Because of the attacks:
  - AES-128 has computational complexity of $2^{126.1}$ (~126 bits)
  - AES-192 has computational complexity of $2^{189.7}$ (~189 bits)
  - AES-256 has computational complexity of $2^{254.9}$ (~254 bits)
- The security of AES can be increased by increasing the number of rounds in the algorithm
- However, AES-128 still has a sufficient safety margin to make exhaustive search attacks impractical

Cryptographic attacks

- Chosen plaintext
  - Attacker can create plaintext and see the corresponding ciphertext
- Known plaintext
  - Attacker has access to both plaintext & ciphertext but doesn’t get to choose the text
- Ciphertext-only
  - The attacker only sees ciphertext
  - Popular in movies but rarely practical in real life

Differential Cryptanalysis

Examine how changes in input affect changes in output

- Discover where a cipher exhibits non-random behavior
  - These properties can be used to extract the secret key
  - Applied to block ciphers, stream ciphers, and hash functions (functions that flip & move bits vs. mathematical operations)

- Chosen plaintext attack is normally used
  - Attacker must be able to choose the plaintext and see the corresponding cipher text

Differential Cryptanalysis

- Provide plaintext with known differences
  - See how those differences appear in the ciphertext
- The properties depend on the key and the s-boxes in the algorithm
- Do this with lots and lots of known *plaintext-ciphertext* sets
- Statistical differences, if found, may allow a key to be recovered faster than with a brute-force search
  - You may deduce that certain keys are not worth trying

Linear Cryptanalysis

Create a predictive approximation of inputs to outputs

- Instead of looking for differences, linear cryptanalysis attempts to come up with a linear formula (e.g., a bunch of xor operations) that connects certain input bits, output bits, and key bits with a probability higher than random
  - Goal is to approximate the behavior of s-boxes

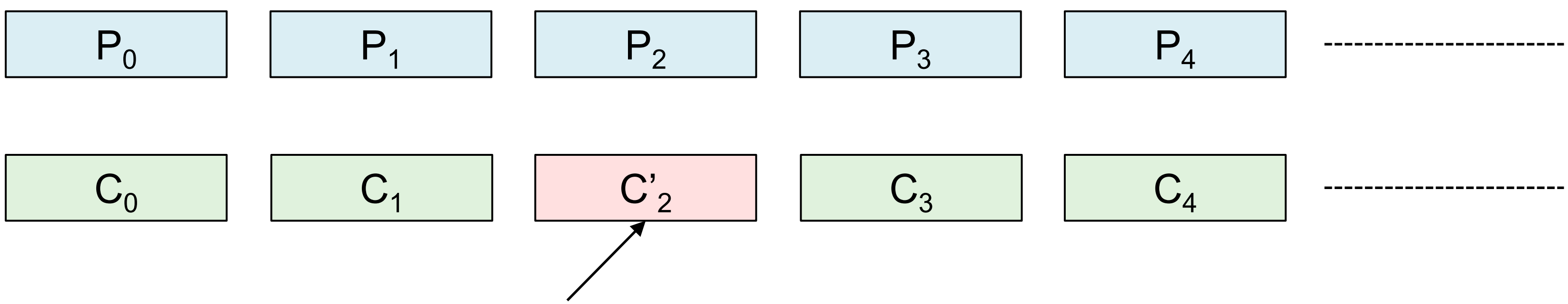
- It will *not* recreate the working of the cipher
  - You just hope to find non-random behavior that gives you insight on what bits of the key might matter

- Works better than differential cryptanalysis for known plaintext Differential cryptanalysis works best with chosen plaintext

- Linear & differential cryptanalysis will rarely recover a key but may be able to reduce the number of keys that need to be searched.

Not a good idea to use block ciphers directly

- Streams of data are broken into *k*-byte blocks
  - Each block encrypted separately
  - This is called Electronic Codebook (ECB)
- Problems
  1. Same plaintext results in identical encrypted blocks
     Enemy can build up a code book of plaintext/ciphertext matches
  2. Attacker can add/delete/replace blocks

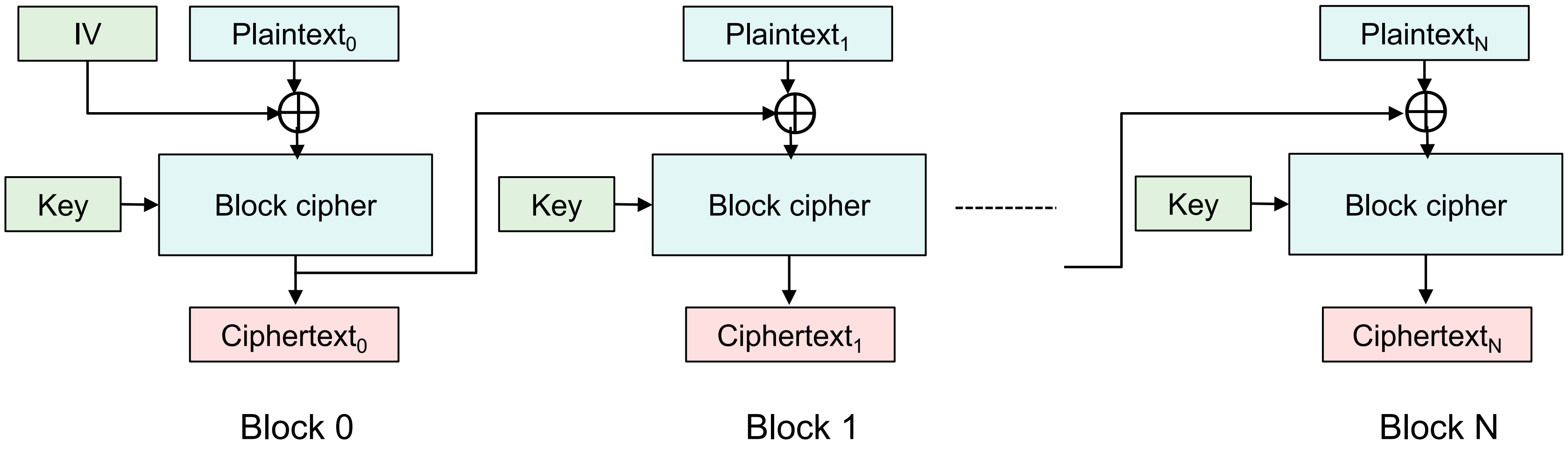


Intruder can replace individual blocks

Cipher Block Chaining (CBC) mode

- Random initialization vector (IV) = bunch of *k* random bits
  - Non-secret: both parties need to know this
- Exclusive-or with first plaintext block – then encrypt the block
- Take exclusive-or of the result with the next plaintext block

$$c_i = E_K(m) \oplus c_{i-1}$$

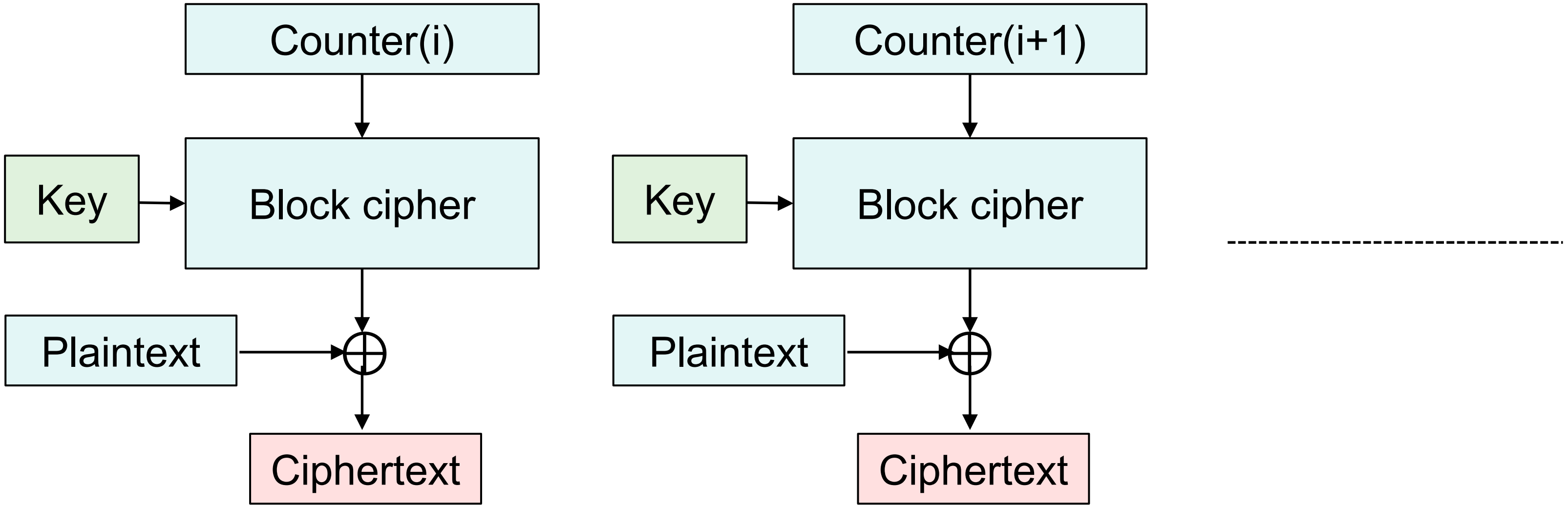


CBC Observations

- Identical blocks of plaintext do not produce the same ciphertext
- Each block is a function of all previous blocks
- But an attacker can still cause data corruption

Block encryption: Counter (CTR) mode

- Random starting counter = bunch of *k* random bits, just like IV
  - Any function producing a non-repeating sequence (an incrementing number is a common function)
- **Encrypt the counter with the key**
- Exclusive-or result with plaintext block

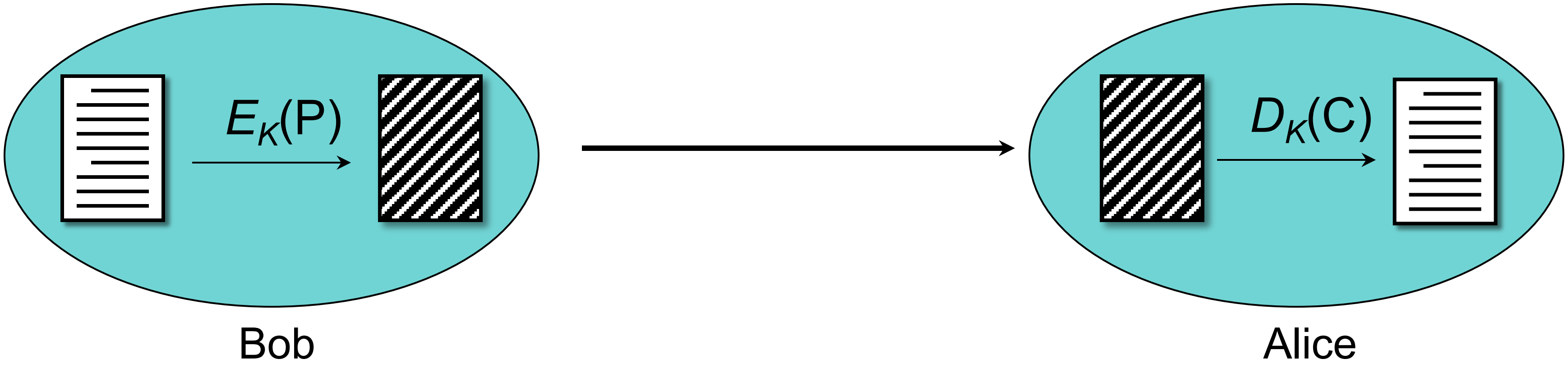


Popular symmetric algorithms

- AES (Advanced Encryption Standard)
  - FIPS standard since 2002
  - 128, 192, or 256-bit keys; operates on 128-bit blocks
- DES, 3DES
  - FIPS standard since 1976
  - 56-bit key; operates on 64-bit (8-byte) blocks
  - Triple DES recommended since 1999 (112 or 168 bits)
- Blowfish
  - Key length from 23-448 bits; 64-bit blocks
- IDEA
  - 128-bit keys; operates on 64-bit blocks
  - More secure than DES but faster algorithms are available

Communicating with symmetric cryptography

- Both parties must agree on a secret key, *K*
- Message is encrypted, sent, decrypted at other side

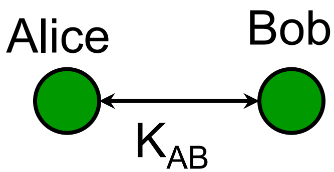


- Key distribution must be secret
  - otherwise messages can be decrypted
  - users can be impersonated

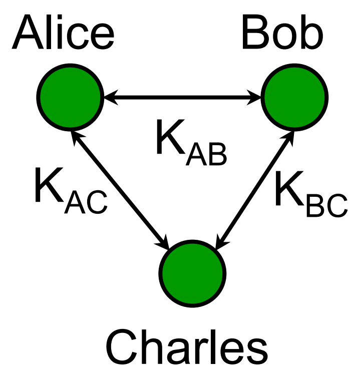
Key Distribution

Key explosion

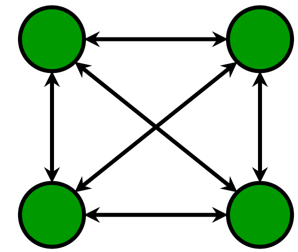
Each pair of users needs a separate key for secure communication



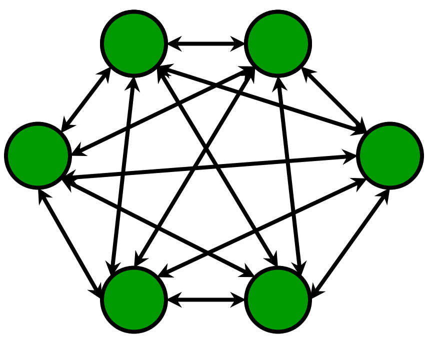
2 users: 1 key



3 users: 3 keys



4 users: 6 keys



6 users: 15 keys

100 users: 4,950 keys

1000 users: 399,500 keys

n users: $\frac{n(n-1)}{2}$ keys

Key distribution

Secure key distribution is the biggest problem with symmetric cryptography

Public-key algorithm

- Two related keys.

$$C = E_{K1}(P) \qquad P = D_{K2}(C)$$
$$C' = E_{K2}(P) \qquad P = D_{K1}(C')$$

K_1 is a **public** key
K_2 is a **private** key

- Examples:
  – RSA, Elliptic curve algorithms, DSS (digital signature standard)

- Key length
  – Unlike symmetric cryptography, not every number is a valid key
  – 3072-bit RSA = 256-bit elliptic curve = 128-bit symmetric cipher
  – 15360-bit RSA = 521-bit elliptic curve = 256-bit symmetric cipher

RSA Public Key Cryptography

- Ron Rivest, Adi Shamir, Leonard Adleman created a true public key encryption algorithm in 1977
- Each user generates two keys:
  - Private key (kept secret)
  - Public key (can be shared with anyone)
- Difficulty of algorithm based on the difficulty of factoring large numbers
  - keys are functions of a pair of large (~300 digits) prime numbers

RSA algorithm

How to generate keys

- choose two random large prime numbers p, q
- Compute the product $n = pq$
- randomly choose the encryption key, e, such that:
  e and $(p - 1)(q - 1)$ are relatively prime
- Compute a decryption key, d such that:

  $ed = 1 \bmod ((p - 1)(q - 1))$

  $d = e^{-1} \bmod ((p - 1)(q - 1))$
- discard p, q

The security of the algorithm rests on our understanding that factoring n is extremely difficult

RSA Encryption

- What you need:
  - Key pair: *e, d*
  - Agreed-upon modulus: *n*

- Encrypt:
  - divide data into numerical blocks < *n*
  - encrypt each block:

    $c = m^e \bmod n$

- Decrypt:

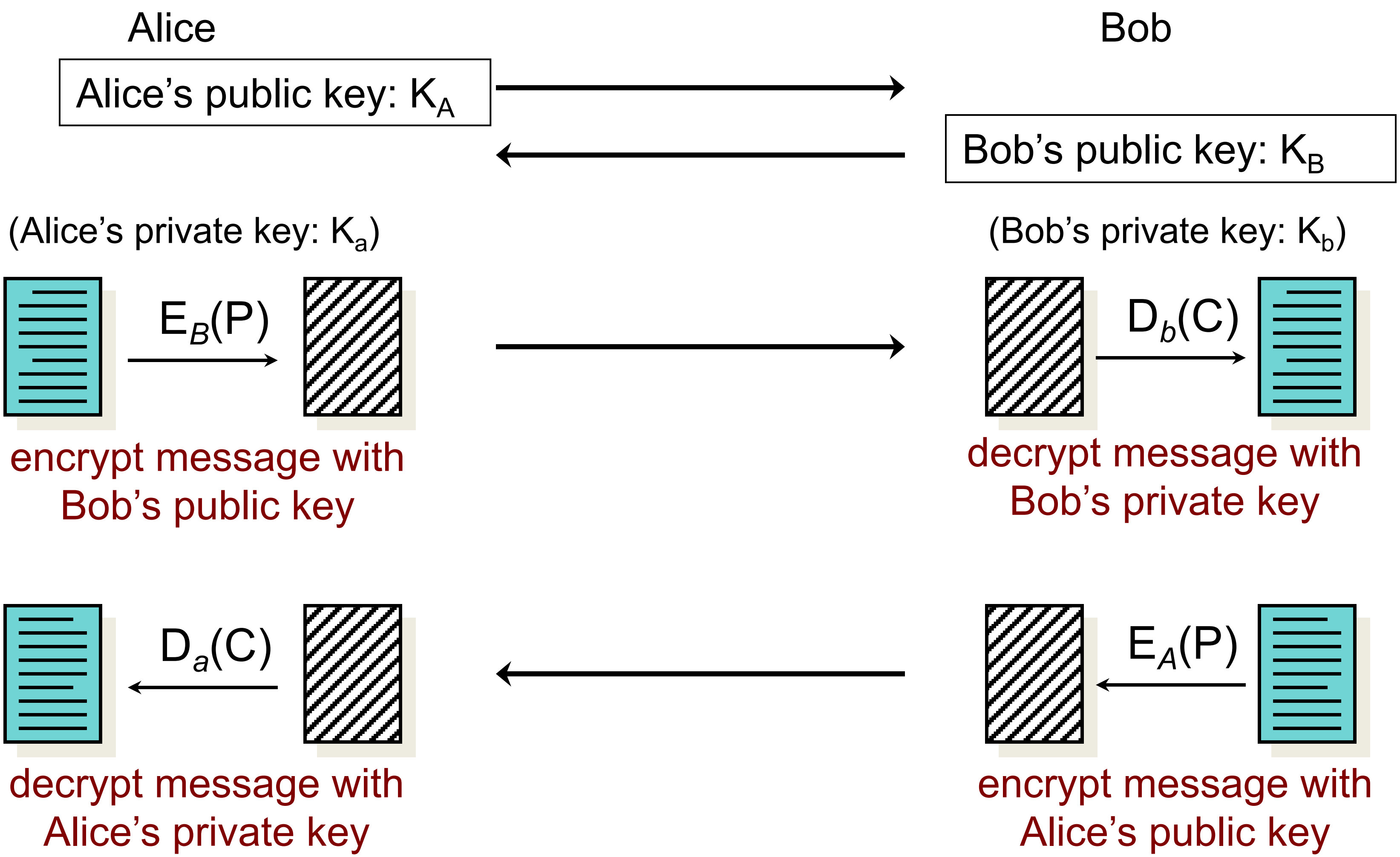
    $m = c^d \bmod n$

Communication with public key algorithms

Different keys for encrypting and decrypting

- No need to worry about key distribution

Communication with public key algorithms



RSA isn't good for large-scale encryption

Calculations are very expensive (& key generation is slow)

Common speeds:

Algorithm	Bytes/sec
AES-128-ECB	148,000,000
AES-128-CBC	153,000,000
AES-256-ECB	114,240,000
RSA-2048 encrypt	3,800,000
RSA-2048 decrypt	96,000

- AES ~1500x faster to decrypt; 40x faster to encrypt
- RSA is also subject to mathematical attacks
  - Certain keys (numbers) may expose weaknesses
- If anyone learns your private key, they can read all your messages

Diffie-Hellman Key Exchange

Key distribution algorithm

- Allows two parties to exchange keys securely
- *Not* public key encryption
- Based on difficulty of computing discrete logarithms in a finite field compared with ease of calculating exponentiation

Allows us to negotiate a secret **common key** without fear of eavesdroppers

Diffie-Hellman Key Exchange

- All arithmetic performed in a field of integers modulo some large number
- Both parties agree on
  - a **large prime number *p***
  - and a number $\alpha < p$
- Each party generates a public/private key pair

Private key for user *i*: X_i

Public key for user *i*: $Y_i = \alpha^{X_i} \bmod p$

Diffie-Hellman exponential key exchange

- Alice has secret key X_A
- Alice sends Bob public key Y_A
- Alice computes

- Bob has secret key X_B
- Bob sends Alice public key Y_B

$$K = Y_B^{X_A} \bmod p$$

K = (Bob's public key) $^{\textbf{\textit{(Alice's private key)}}}$ ***mod p***

Diffie-Hellman exponential key exchange

- Alice has secret key X_A
- Alice sends Bob public key Y_A
- Alice computes

$$K = Y_B^{X_A} \bmod p$$

- Bob has secret key X_B
- Bob sends Alice public key Y_B
- Bob computes

$$K = Y_A^{X_B} \bmod p$$

$$\boldsymbol{K' = (\text{Alice's public key})^{(\text{Bob's private key})} \bmod p}$$

Diffie-Hellman exponential key exchange

- Alice has secret key X_A
- Alice sends Bob public key Y_A
- Alice computes

$$K = Y_B^{X_A} \bmod p$$

- expanding:

$$\begin{aligned} K &= Y_B^{X_A} \bmod p \\ &= (\alpha^{X_B} \bmod p)^{X_A} \bmod p \\ &= \alpha^{X_B X_A} \bmod p \end{aligned}$$

- Bob has secret key X_B
- Bob sends Alice public key Y_B
- Bob computes

$$K = Y_A^{X_B} \bmod p$$

- expanding:

$$\begin{aligned} K &= Y_B^{X_B} \bmod p \\ &= (\alpha^{X_A} \bmod p)^{X_B} \bmod p \\ &= \alpha^{X_A X_B} \bmod p \end{aligned}$$

K = K′

K is a *common key*, known *only* to Bob and Alice

Hybrid Cryptosystems

- Session key: randomly-generated key for one communication session
- Use a public key algorithm to send the session key
- Use a symmetric algorithm to encrypt data with the session key

Public key algorithms are almost never used to encrypt messages

- Vulnerable to chosen plaintext attacks
- MUCH slower; RSA-2048 approximately 40x slower to encrypt and 1,500x slower to decrypt than AES-256

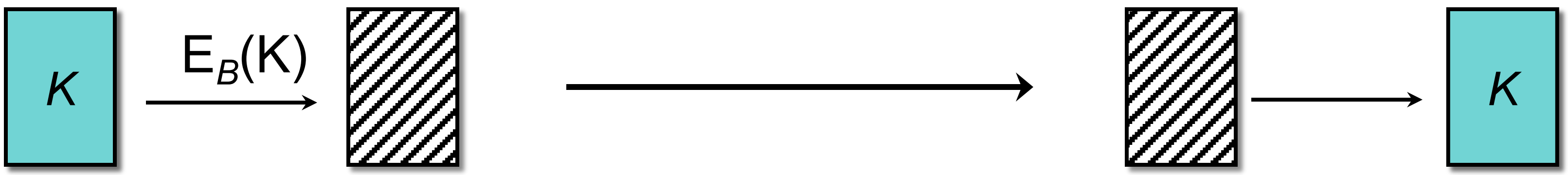
Communication with a hybrid cryptosystem

Alice

Bob

Bob's public key: K_B

Pick a random session key, *K*



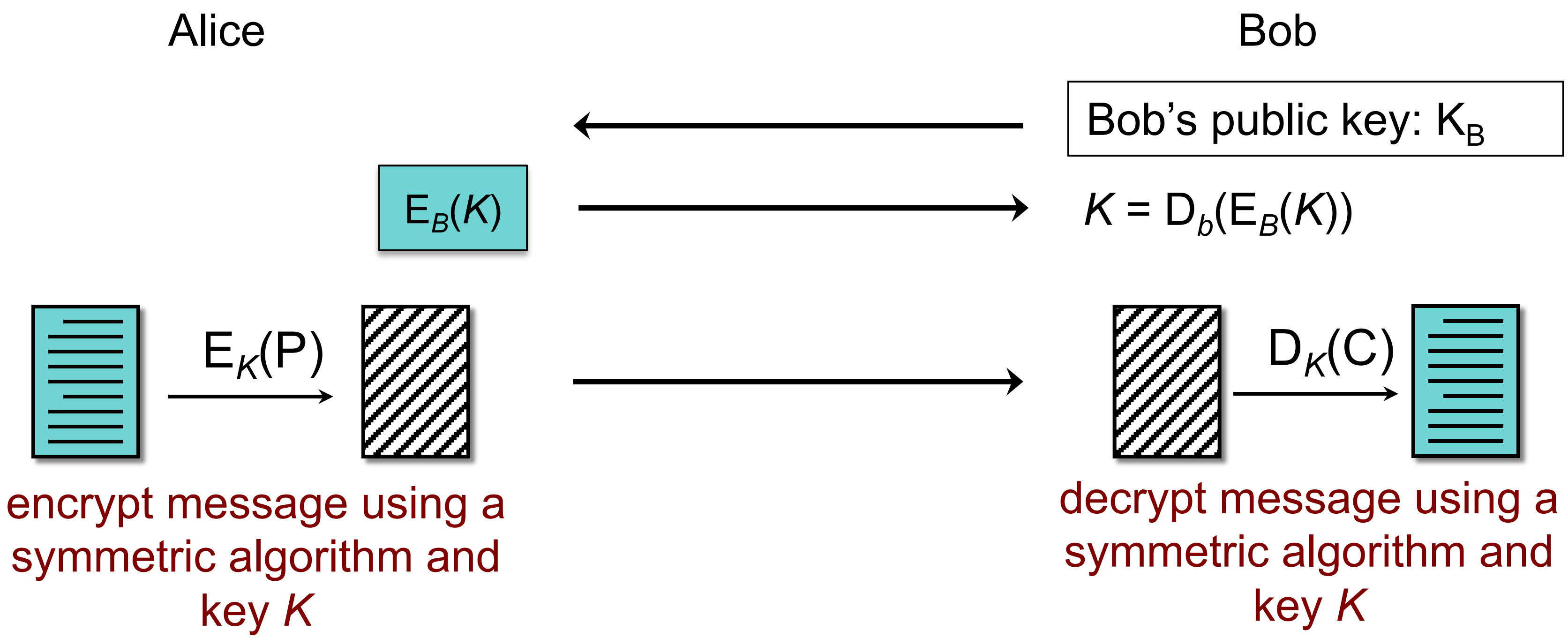
encrypt session key with Bob's public key

$K = D_b(E_B(K))$

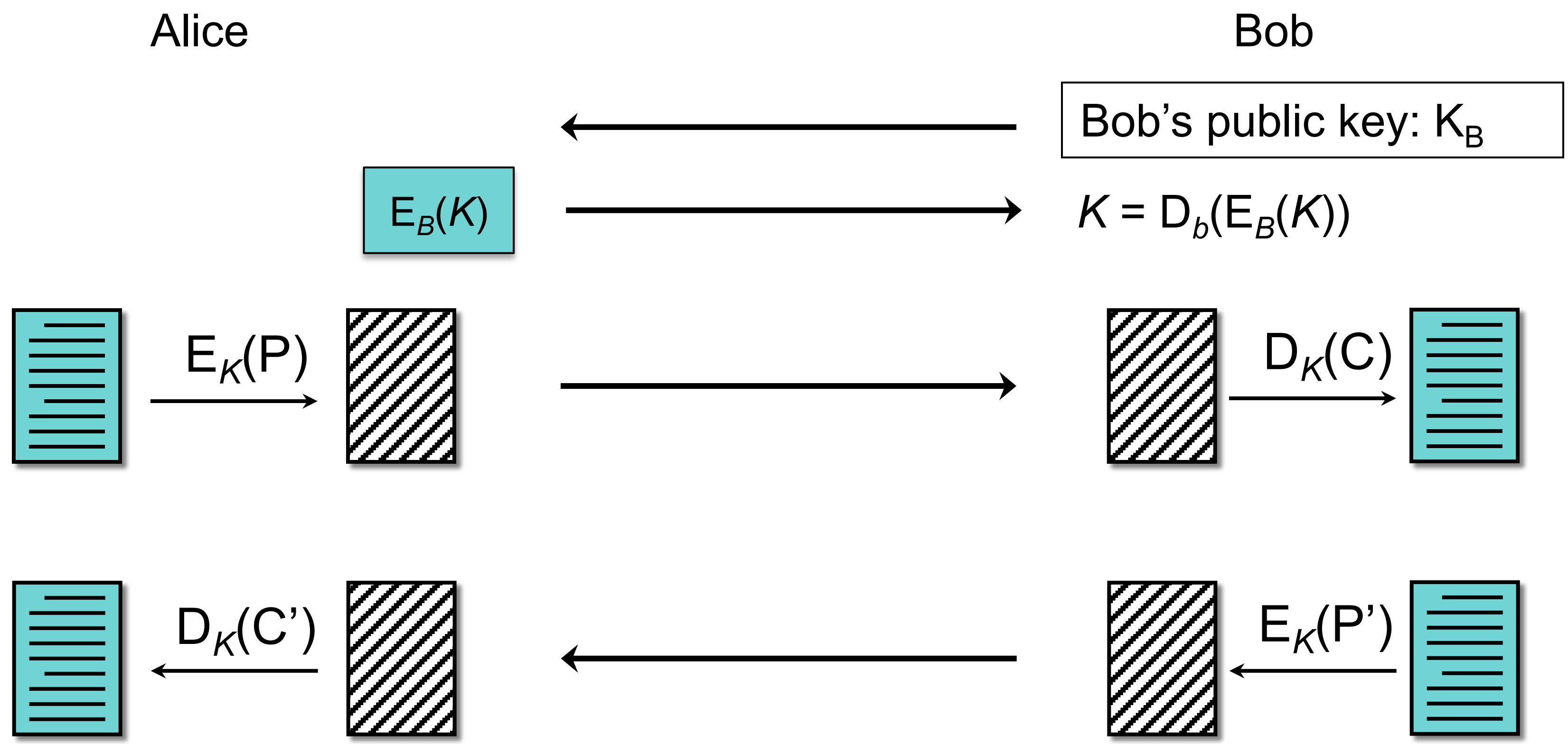
Bob decrypts *K* with his private key

Now Bob knows the secret session key, K

Communication with a hybrid cryptosystem



Communication with a hybrid cryptosystem



decrypt message using a symmetric algorithm and key *K*

encrypt message using a symmetric algorithm and key *K*

Forward Secrecy

Suppose an attacker steals Bob’s private key

- Future messages can be compromised
- But past messages that used the key can also be decrypted

Pick a session key
Encrypt it with the Bob's public key → Bob decrypts the session key

Security rests entirely on the secrecy of Bob's private key

If Bob's private key is compromised, all recorded past traffic can be decrypted

Forward Secrecy

- **Forward secrecy**
  - ***Compromise of long term keys does not compromise past session keys***
  - There is no one secret to steal that will compromise multiple messages
- Diffie-Hellman
  - Use common key as the encryption/decryption key
    - Or as a key to encrypt a session key
  - Not recoverable as long as long as private keys are thrown away after each session
  - Unlike RSA keys, Diffie Hellman makes key generation simple
- Keys must be ephemeral
  - Client & server will generate new Diffie-Hellman parameters for each session – all will be thrown away after the session

Diffie-Hellman is preferred over RSA for key exchange to achieve forward secrecy – generating Diffie-Hellman keys is a rapid, low-overhead process

Cryptographic systems: summary

- Symmetric ciphers
  - Based on “SP networks” = substitution & permutation sequences
- Asymmetric ciphers – public key cryptosystems
  - Based on trapdoor functions
    Easy to compute in one direction; difficult to compute in the other direction without special information (the trapdoor)
- Hybrid cryptosystem
  - Pick a random session key
  - Use a public key algorithm to send
  - Use a symmetric key algorithm to encrypt traffic back & forth
- Key exchange algorithms (more to come later)
  - Diffie Hellman
  - Public key

  } *Enables secure communication without knowledge of a shared secret*

RSA cryptography in the future

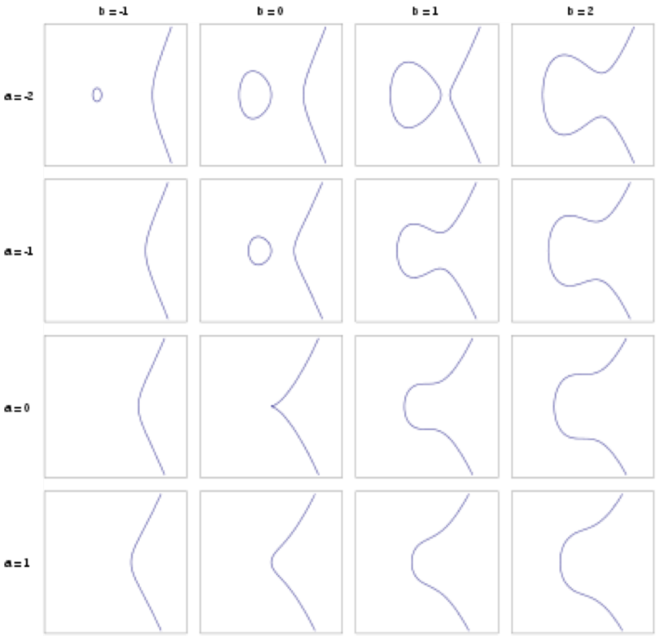
- Based on the difficulty of factoring products of two large primes
- Factoring algorithms get more efficient as numbers get larger
  - As the ability to decrypt numbers increases, the key size must therefore grow even faster
  - This is not sustainable (especially for embedded devices)

Elliptic Curve Cryptography

- Alternate approach: elliptic curves

$$y^2 = x^3 + ax + b$$

- Using discrete numbers, pick
  – A prime number as a maximum (modulus)
  – A curve equation
  – A public base point on the curve
  – A random private key
  – Public key is derived from the private key, the base point, and the curve



Catalog of elliptic curves
https://en.wikipedia.org/wiki/Elliptic_curve

- To compute the private key from the public,
  – We need an elliptic curve discrete logarithm function
  – This is difficult and is the basis for ECC's security

ECC vs. RSA

- RSA is still the most widely used public key cryptosystem
  - Mostly due to inertia & widespread implementations
  - Simpler implementation & faster decryption
- ECC offers higher security with fewer bits than RSA
  - ECC is also faster (for key generation & encryption) and uses less memory
  - NIST defines 15 standard curves for ECC
    - Many implementations support only a couple (P-256, P-384)
- For long-term security

  The European Union Agency for Network and Information Security (ENISA) and the National Institute for Science & Technology (NIST) recommend:
  - AES: 256 bit keys
  - RSA: 15,360 bit keys
  - ECC: 512 bit keys

https://www.keylength.com/en/4/

http://https://www.enisa.europa.eu/publications/algorithms-key-size-and-parameters-report-2014

Quantum Computers

- Once (if) real quantum computers can be built, they can
  - Crack a symmetric cipher in time proportional to the square root of the key space size: $2^{n/2}$
    - Use 256-bit AES to be safe
  - Factor efficiently
    - RSA and many elliptic curve algorithms will not be secure anymore
- NSA called for a migration to “post-quantum cryptographic algorithms” – but no agreement yet on what they are

The End