Cryptography Overview

John Mitchell

Announcement: Homework 1

- Homework 1 has 3 problems
  - Problems 1 and 2 on the web now
  - Problem 3 to be added soon
- Due in two weeks, on January 24
  - Install Murphi and run problems 1,2 by 1/17

Cryptography

- Is
  - A tremendous tool
  - The basis for many security mechanisms
- Is not
  - The solution to all security problems
  - Reliable unless implemented properly
  - Reliable unless used properly
  - Something you should try to invent yourself unless
    - you spend a lot of time becoming an expert
    - you subject your design to outside review

Basic Cryptographic Concepts

- Encryption scheme:
  - functions to encrypt, decrypt data
  - key generation algorithm
- Secret key vs. public key
  - Public key: publishing *key* does not reveal key^{-1}
  - Secret key: more efficient, generally $key = key^{-1}$
- Hash function, MAC
  - Map any input to short hash; ideally, no collisions
  - MAC (keyed hash) used for message integrity
- Signature scheme
  - Functions to sign data, verify signature

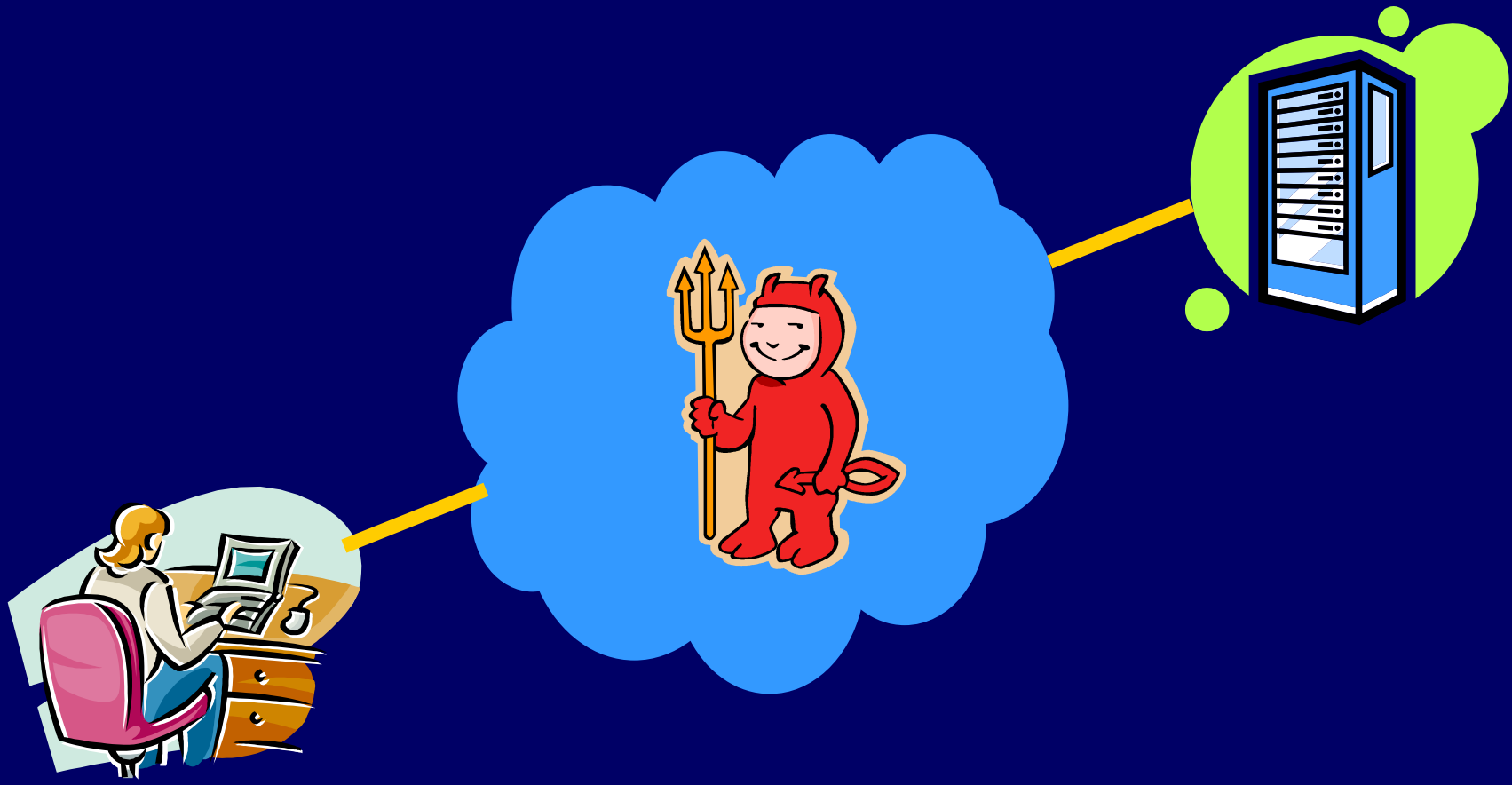
Five-Minute University

Father Guido Sarducci

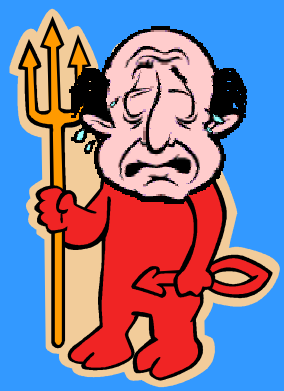
- Everything you might remember, five years after taking CS255 ... ?

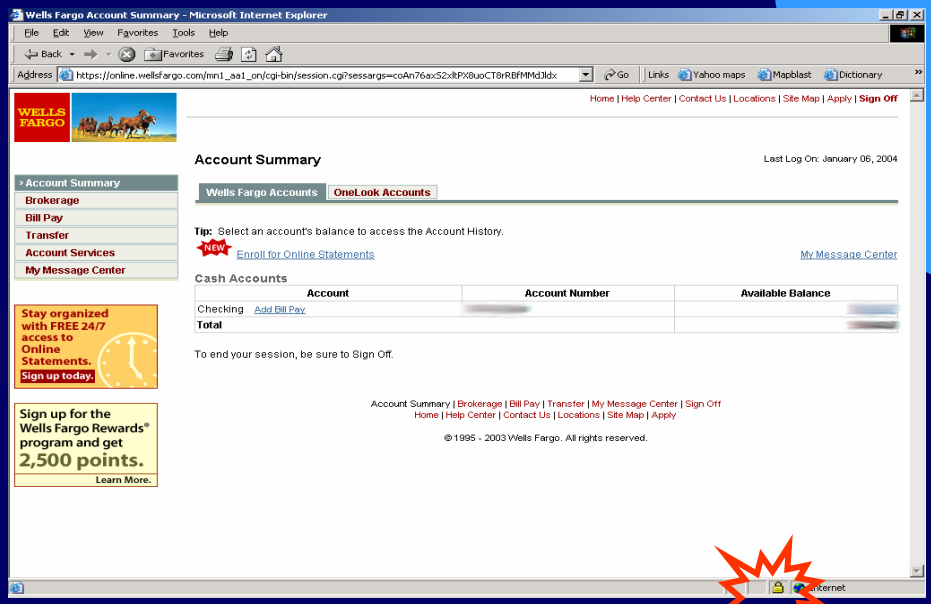
This lecture describes basic functions *and* example constructions. Constructions *not* needed for CS259.

Web Purchase



Secure communication





Secure Sockets Layer / TLS

- Standard for Internet security
  - Originally designed by Netscape
  - Goal: “… provide privacy and reliability between two communicating applications”
- Two main parts
  - Handshake Protocol
    - Establish shared secret key using public-key cryptography
    - Signed certificates for authentication
  - Record Layer
    - Transmit data using negotiated key, encryption function

SSL/TLS Cryptography

- Public-key encryption
  - Key chosen secretly (handshake protocol)
  - Key material sent encrypted with public key
- Symmetric encryption
  - Shared (secret) key encryption of data packets
- Signature-based authentication
  - Client can check signed server certificate
  - And vice-versa, in principal
- Hash for integrity
  - Client, server check hash of sequence of messages
  - MAC used in data packets (record protocol)

Example cryptosystems

- One-time pad
  - "Theoretical idea," but leads to stream cipher
- Feistel construction for symmetric key crypto
  - Iterate a "scrambling function"
  - Examples: DES, Lucifer, FREAL, Khufu, Khafre, LOKI, GOST, CAST, Blowfish, ...
  - AES (Rijndael) is also block cipher, but different
- Complexity-based public-key cryptography
  - Modular exponentiation is a "one-way" function
  - Examples: RSA, El Gamal, elliptic curve systems, ...

One-time pad

- Secret-key encryption scheme (symmetric)
  - Encrypt plaintext by xor with sequence of bits
  - Decrypt ciphertext by xor with same bit sequence
- Scheme for pad of length n
  - Set P of plaintexts: all n-bit sequences
  - Set C of ciphertexts: all n-bit sequences
  - Set K of keys: all n-bit sequences
  - Encryption and decryption functions

$$\text{encrypt}(\text{key}, \text{text}) = \text{key} \oplus \text{text} \quad \text{(bit-by-bit)}$$

$$\text{decrypt}(\text{key}, \text{text}) = \text{key} \oplus \text{text} \quad \text{(bit-by-bit)}$$

Evaluation of one-time pad

- Advantages
  - Easy to compute encrypt, decrypt from key, text
  - As hard to break as possible
    - This is an information-theoretically secure cipher
    - Given ciphertext, all possible plaintexts are equally likely, assuming that key is chosen randomly
- Disadvantage
  - Key is as long as the plaintext
    - How does sender get key to receiver securely?

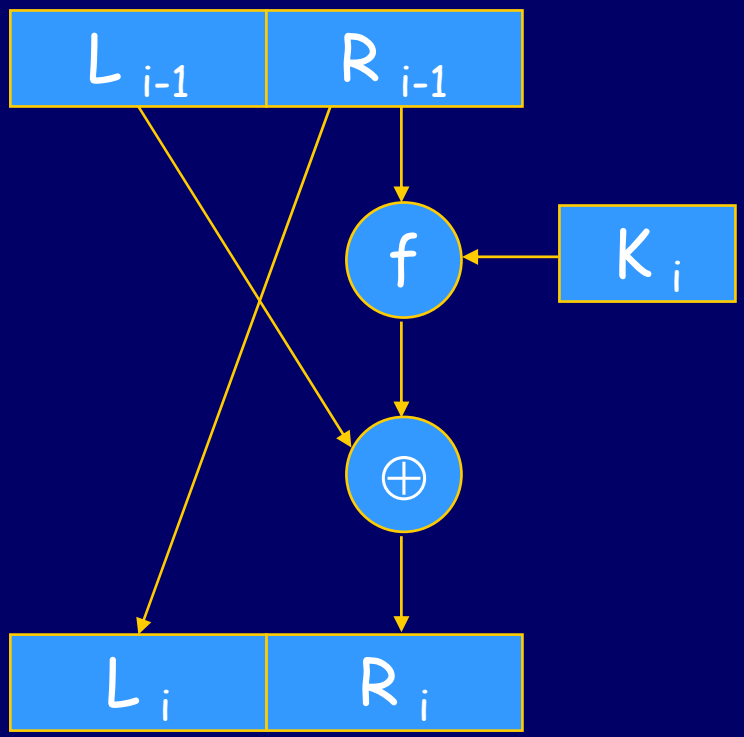
Idea for stream cipher: use pseudo-random generators for key…

Feistel networks

- Many block algorithms are *Feistel networks*
  - A block cipher encrypts data in blocks
    - Encryption of block n+1 *may* depend on block n
  - Feistel network is a standard construction for
    - Iterating a function f on parts of a message
    - Producing an invertible transformation
- AES (Rijndael) is related but different
  - Also a block cipher with repeated rounds
  - Not a Feistel network

Feistel network: One Round

Divide n-bit input in half and repeat



- Scheme requires
  - Function $f(R_{i-1}, K_i)$
  - Computation for K_i
    - e.g., permutation of key K
- Advantage
  - Systematic calculation
    - Easy if f is table, etc.
  - Invertible if K_i known
    - Get R_{i-1} from L_i
    - Compute $f(R_{i-1}, K_i)$
    - Compute L_{i-1} by $\oplus$

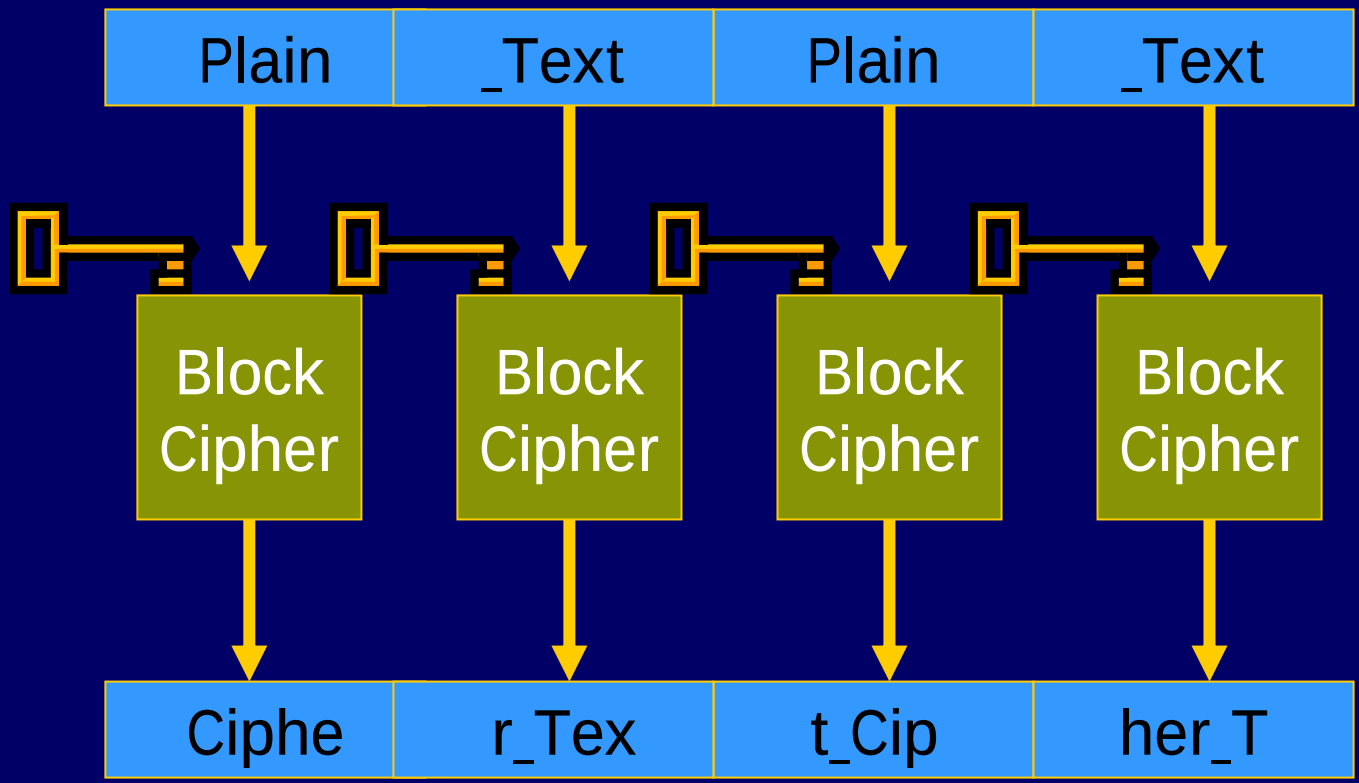
Data Encryption Standard

- Developed at IBM, some input from NSA, widely used
- Feistel structure
  - Permute input bits
  - Repeat application of a *S-box* function
  - Apply inverse permutation to produce output
- Worked well in practice (but brute-force attacks now)
  - Efficient to encrypt, decrypt
  - Not provably secure
- Improvements
  - Triple DES, AES (Rijndael)

Block cipher modes (for DES, AES, ...)

- ECB – Electronic Code Book mode
  - Divide plaintext into blocks
  - Encrypt each block independently, with same key
- CBC – Cipher Block Chaining
  - XOR each block with encryption of previous block
  - Use initialization vector IV for first block
- OFB – Output Feedback Mode
  - Iterate encryption of IV to produce stream cipher
- CFB – Cipher Feedback Mode
  - Output block y_i = input $x_i \oplus$ encyrpt$_K(y_{i-1})$

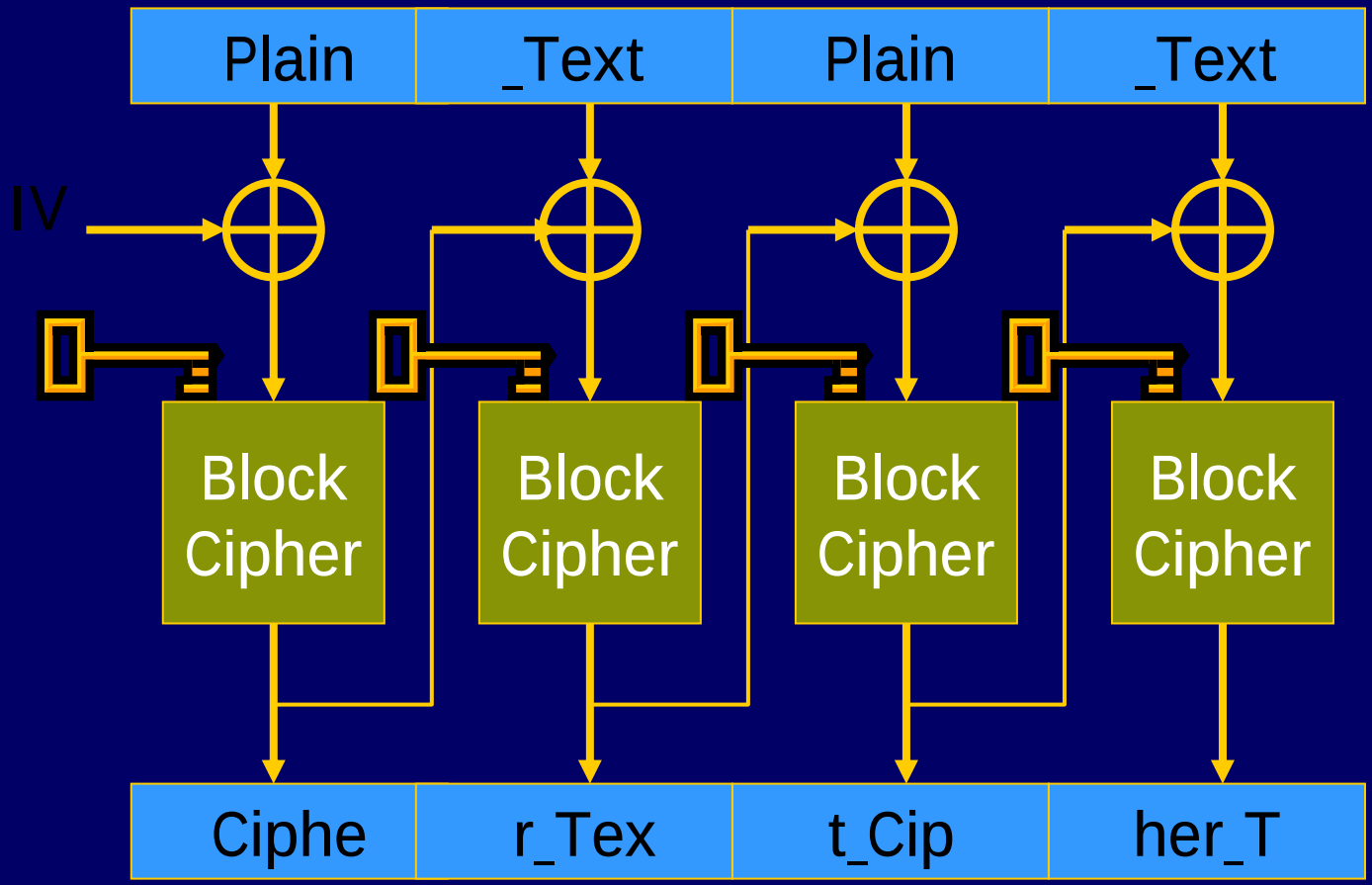
Electronic Code Book (ECB)



Problem: Identical blocks encrypted identically

No integrity check

Cipher Block Chaining (CBC)



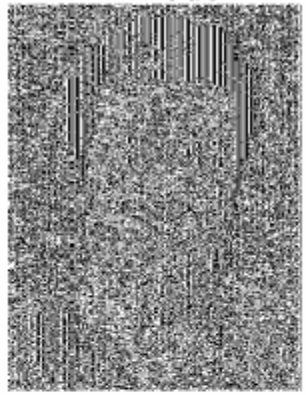
Advantages: Identical blocks encrypted differently

Last ciphertext block depends on entire input

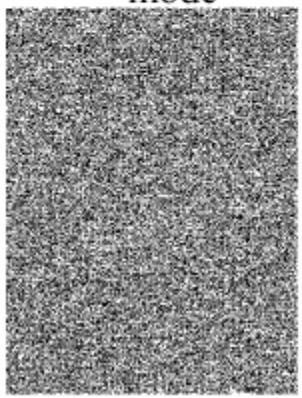
Comparison (for AES, by Bart Preneel)

An example plaintext

Encrypted with AES in ECB mode



Encrypted with AES in CBC mode



Similar plaintext blocks produce similar ciphertext (see outline of head)

No apparent pattern

RC4 stream cipher – “Ron’s Code”

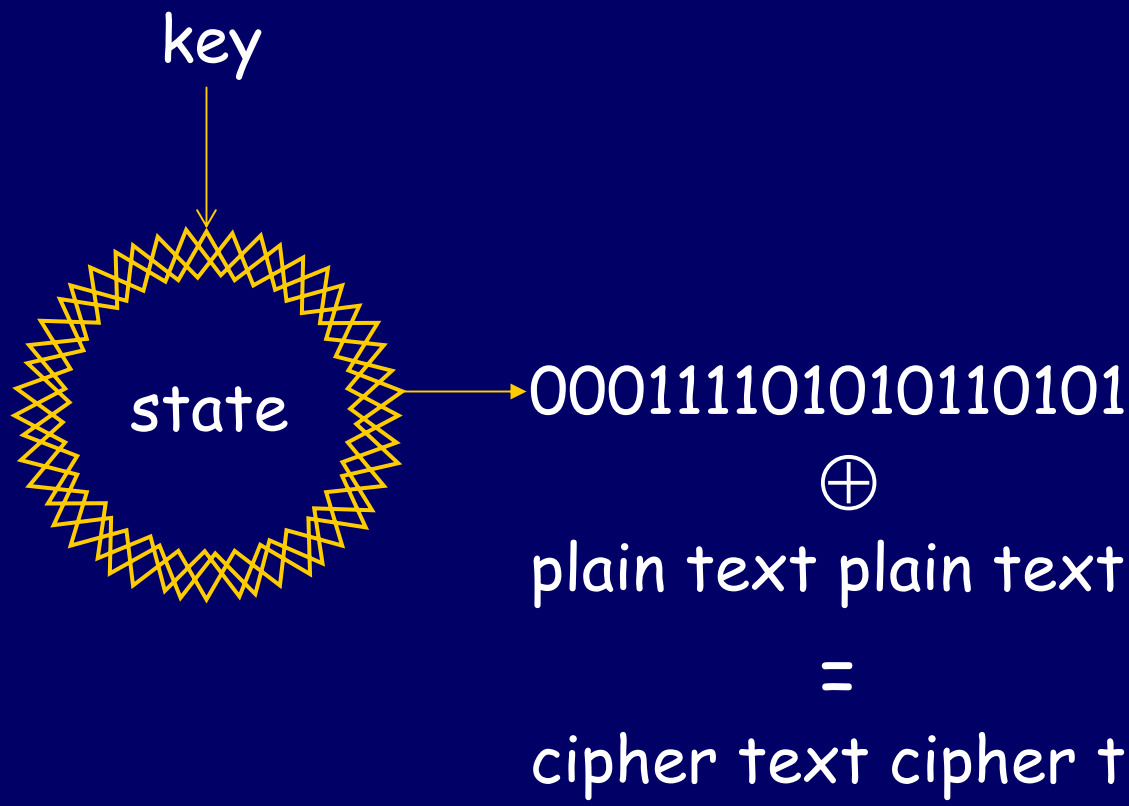
- Design goals (Ron Rivest, 1987):
  - speed
  - support of 8-bit architecture
  - simplicity (circumvent export regulations)
- Widely used
  - SSL/TLS
  - Windows, Lotus Notes, Oracle, etc.
  - Cellular Digital Packet Data
  - OpenBSD pseudo-random number generator

RSA Trade Secret

- History
  - 1994 – leaked to cypherpunks mailing list
  - 1995 – first weakness (USENET post)
  - 1996 – appeared in Applied Crypto as “alleged RC4”
  - 1997 – first published analysis

Weakness is predictability of first bits; best to discard them

Encryption/Decryption



Stream cipher: one-time pad based on pseudo-random generator

Security

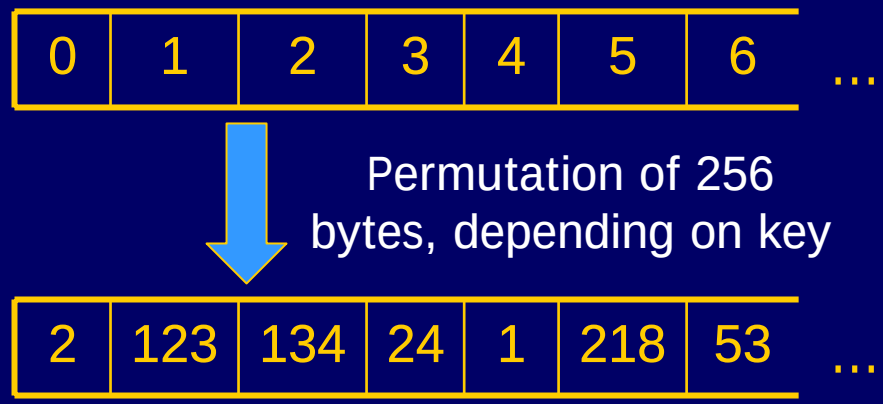
- Goal: indistinguishable from random sequence
  - given part of the output stream, it is impossible to distinguish it from a random string
- Problems
  - Second byte [MS01]
    - Second byte of RC4 is 0 with twice expected probability
  - Related key attack [FMS01]
    - Bad to use many related keys (see WEP 802.11b)
- Recommendation
  - Discard the first 256 bytes of RC4 output [RSA, MS]

Complete Algorithm (all arithmetic mod 256)

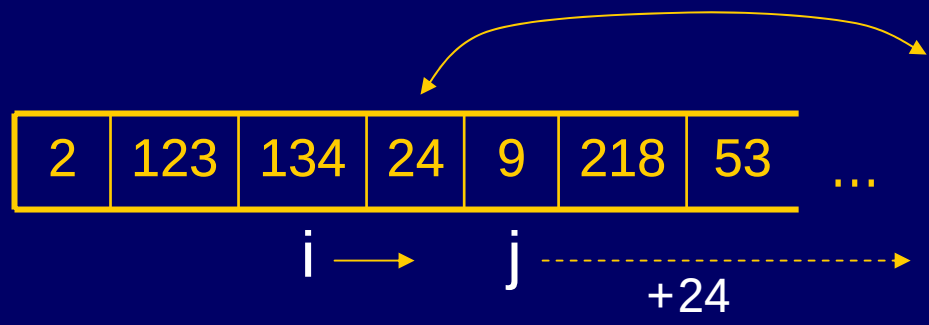
```
for i := 0 to 255    S[i] := i
j := 0
for i := 0 to 255
    j := j + S[i] + key[i]
    swap (S[i], S[j])

i, j := 0
repeat
    i := i + 1
    j := j + S[i]
    swap (S[i], S[j])
    output (S[ S[i] + S[j] ])
```

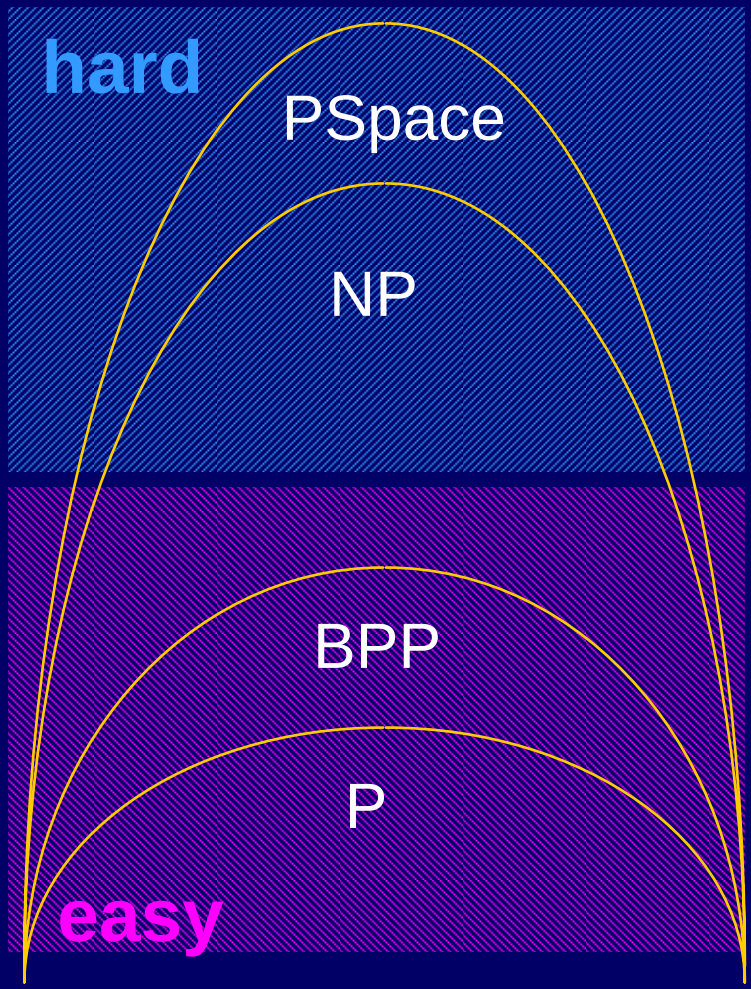
Key scheduling



Random generator



Complexity Classes



Answer in polynomial space
may need exhaustive search

If yes, can guess and check in polynomial time

Answer in polynomial time, with high probability

Answer in polynomial time
compute answer directly

One-way functions

- A function f is one-way if it is
  - Easy to compute f(x), given x
  - Hard to compute x, given f(x), for most x

- Examples (we believe they are one way)
  - f(x) = divide bits x = y@z and multiply f(x)=y*z
  - f(x) = 3^x mod p, where p is prime
  - f(x) = x^3 mod pq, where p,q are primes with |p|=|q|

One-way trapdoor

- A function f is *one-way trapdoor* if
  - Easy to compute f(x), given x
  - Hard to compute x, given f(x), for most x
  - Extra “trapdoor” information makes it easy to compute x from f(x)

- Example (we believe)
  - $f(x) = x^3 \bmod pq$, where p,q are primes with |p|=|q|
  - Compute cube root using (p-1)*(q-1)

Public-key Cryptosystem

- Trapdoor function to encrypt and decrypt
  - encrypt(key, message)

  - decrypt(key $^{-1}$, encrypt(key, message)) = message

- Resists attack
  - Cannot compute m from encrypt(key, m) and key, unless you have key^{-1}

Example: RSA

- Arithmetic modulo pq
  - Generate secret primes p, q
  - Generate secret numbers a, b with $x^{ab} \equiv x \bmod \overbrace{pq}^{n}$
- Public encryption key $\langle n, a \rangle$
  - Encrypt($\langle n, a \rangle$, x) = $x^a \bmod n$
- Private decryption key $\langle n, b \rangle$
  - Decrypt($\langle n, b \rangle$, y) = $y^b \bmod n$
- Main properties
  - This works
  - Cannot compute b from n,a
    - *Apparently*, need to factor n = pq

How RSA works (quick sketch)

- Let p, q be two distinct primes and let n=p*q
  - Encryption, decryption based on group Z_n^*
  - For n=p*q, order $\phi(n) = (p-1)*(q-1)$
    - Proof: (p-1)*(q-1) = p*q - p - q + 1
- Key pair: $\langle a, b \rangle$ with $ab \equiv 1 \bmod \phi(n)$
  - Encrypt(x) = $x^a \bmod n$
  - Decrypt(y) = $y^b \bmod n$
  - Since $ab \equiv 1 \bmod \phi(n)$, have $x^{ab} \equiv x \bmod n$
    - Proof: if gcd(x,n) = 1, then by general group theory, otherwise use “Chinese remainder theorem”.

How well does RSA work?

- Can generate modulus, keys fairly efficiently
  - Efficient rand algorithms for generating primes p,q
    - May fail, but with low probability
  - Given primes p,q easy to compute $n=p*q$ and $\phi(n)$
  - Choose a randomly with $\gcd(a, \phi(n))=1$
  - Compute $b = a^{-1} \bmod \phi(n)$ by Euclidean algorithm
- Public key n, a does not reveal b
  - This is not proven, but believed
- But if n can be factored, all is lost …

Public-key crypto is significantly slower than symmetric key crypto

Message integrity

- For RSA as stated, integrity is a weak point
  - encrypt(k*m) = $(k*m)^e = k^e * m^e$
    $= encrypt(k)*encrypt(m)$
  - This leads to “chosen ciphertext” form of attack
    - If someone will decrypt *new* messages, then can trick them into decrypting m by asking for decrypt(k^e *m)
- Implementations reflect this problem
  - “The PKCS#1 ... RSA encryption is intended primarily to provide confidentiality. ... It is not intended to provide integrity.” RSA Lab. Bulletin
- Additional mechanisms provide integrity

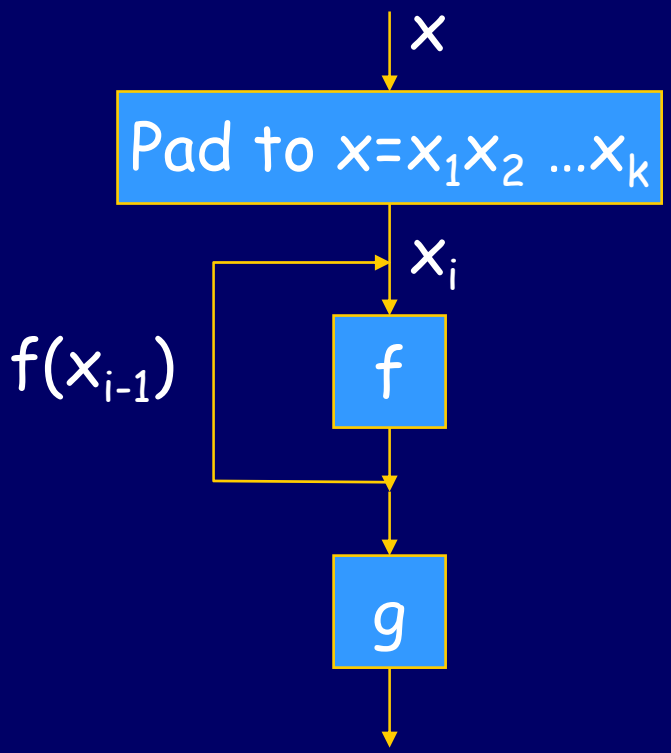
Cryptographic hash functions

- Length-reducing function h
  - Map arbitrary strings to strings of fixed length
- One way ("preimage resistance")
  - Given y, hard to find x with h(x)=y
- Collision resistant
  - Hard to find any distinct m, m' with h(m)=h(m')
- Also useful: 2^{nd} preimage resistance
  - Given x, hard to find $x' \neq x$ with h(x')=h(x)
  - Collision resistance $\Rightarrow$ 2^{nd} preimage resistance

Iterated hash functions

- Repeat use of block cipher or custom function
  - Pad input to some multiple of block length
  - Iterate a length-reducing function f
    - f : 2^{2k} -> 2^k reduces bits by 2
    - Repeat h_0= some seed

      $h_{i+1} = f(h_i, x_i)$
  - Some final function g completes calculation



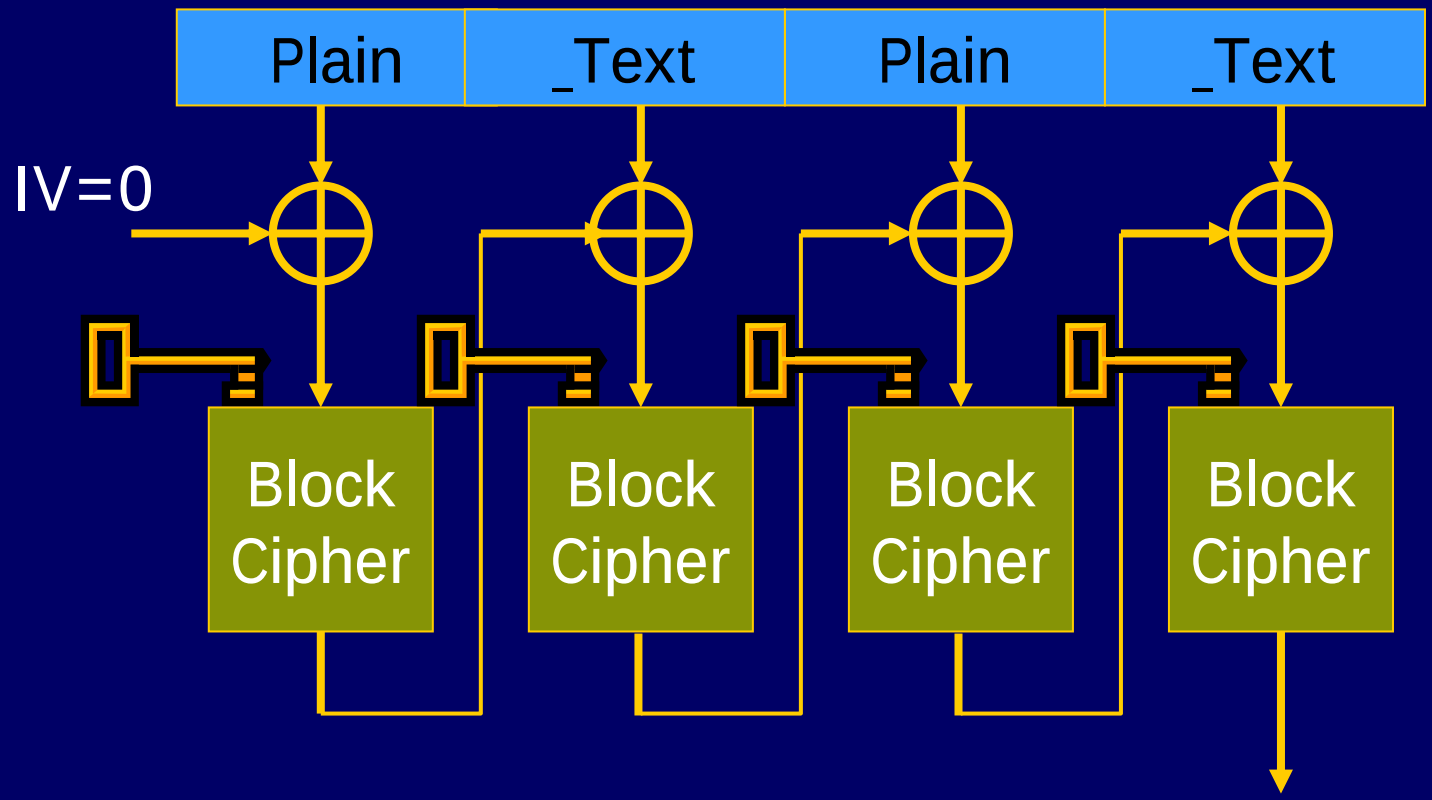
Applications of one-way hash

- Password files (one way)
- Digital signatures (collision resistant)
  - Sign hash of message instead of entire message
- Data integrity
  - Compute and store hash of some data
  - Check later by recomputing hash and comparing
- Keyed hash for message authentication
  - MAC – Message Authentication Code

MAC: Message Authentication Code

- General pattern of use
  - Sender sends Message & MAC(Message), M1
  - Receiver receives both parts
  - Receiver makes his own MAC(Message),M2
    - If M2 != M1, data has been corrupted
    - If M2 == M1, data is valid
- Need for shared key
  - Suppose an attacker can compute MAC(x)
  - Intercept M and Hash(M) and resend as M' and Hash(M')
  - Receiver cannot detect that message has been altered.

Basic CBC-MAC



CBC block cipher, discarding all but last output block

Additional post-processing (e.g, encrypt with second key) can improve output

Digital Signatures

- Public-key encryption
  - Alice publishes encryption key
  - Anyone can send encrypted message
  - Only Alice can decrypt messages with this key
- Digital signature scheme
  - Alice publishes key for verifying signatures
  - Anyone can check a message signed by Alice
  - Only Alice can send signed messages

Properties of signatures

- Functions to sign and verify
  - Sign(Key^{-1}, message)
  - Verify(Key, x, m) = $\begin{cases} true & \text{if } x = Sign(Key^{-1}, m) \\ false & \text{otherwise} \end{cases}$
- Resists forgery
  - Cannot compute Sign(Key^{-1}, m) from m and Key
  - Resists existential forgery:

    given Key, cannot produce Sign(Key^{-1}, m)

    for any random or otherwise arbitrary m

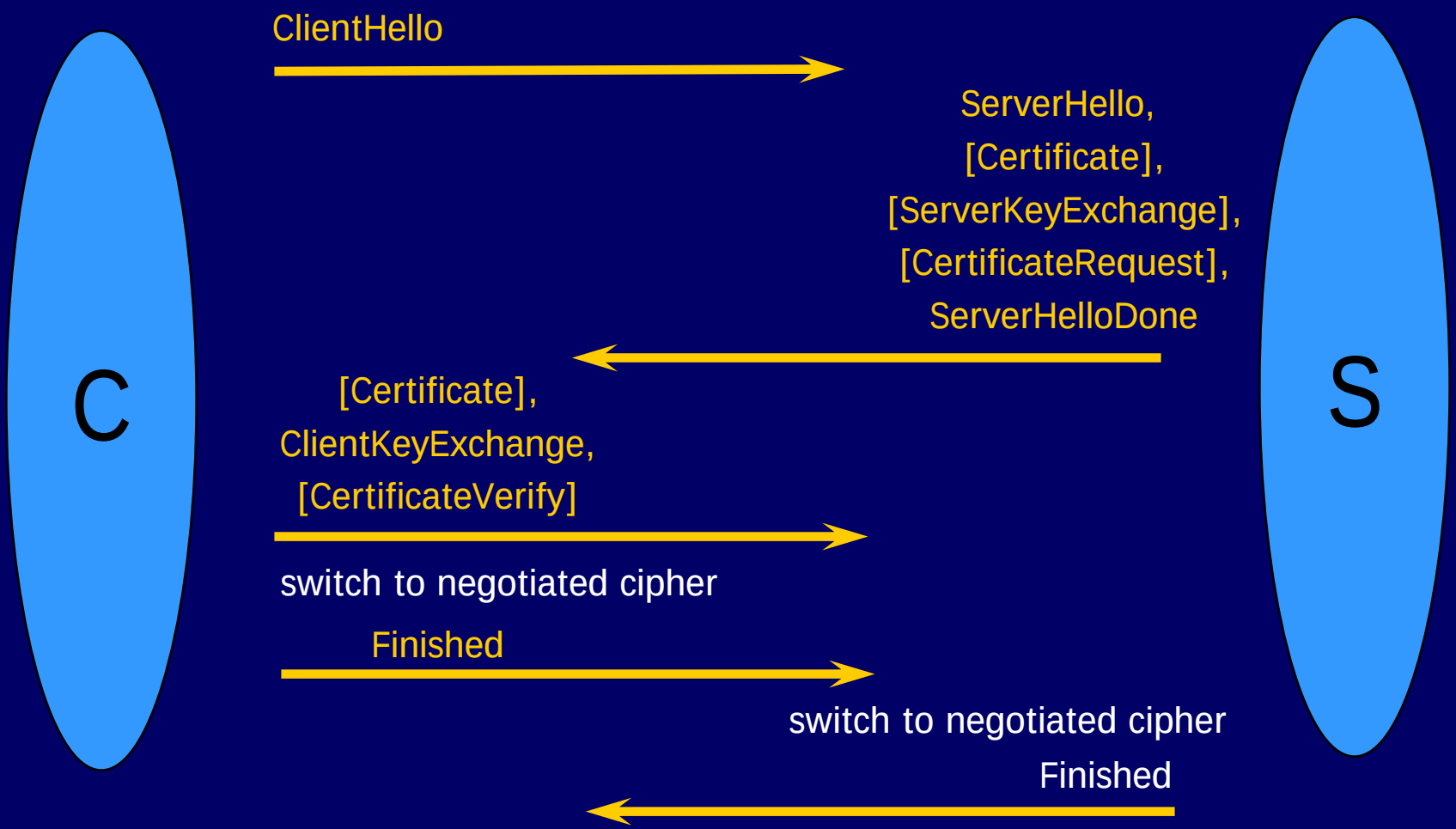
RSA Signature Scheme

- Publish decryption instead of encryption key
  - Alice publishes decryption key
  - Anyone can decrypt a message encrypted by Alice
  - Only Alice can send encrypt messages
- In more detail,
  - Alice generates primes p, q and key pair $\langle a, b \rangle$
  - Sign(x) = $x^a \bmod n$
  - Verify(y) = $y^b \bmod n$
  - Since $ab \equiv 1 \bmod \phi(n)$, have $x^{ab} \equiv x \bmod n$

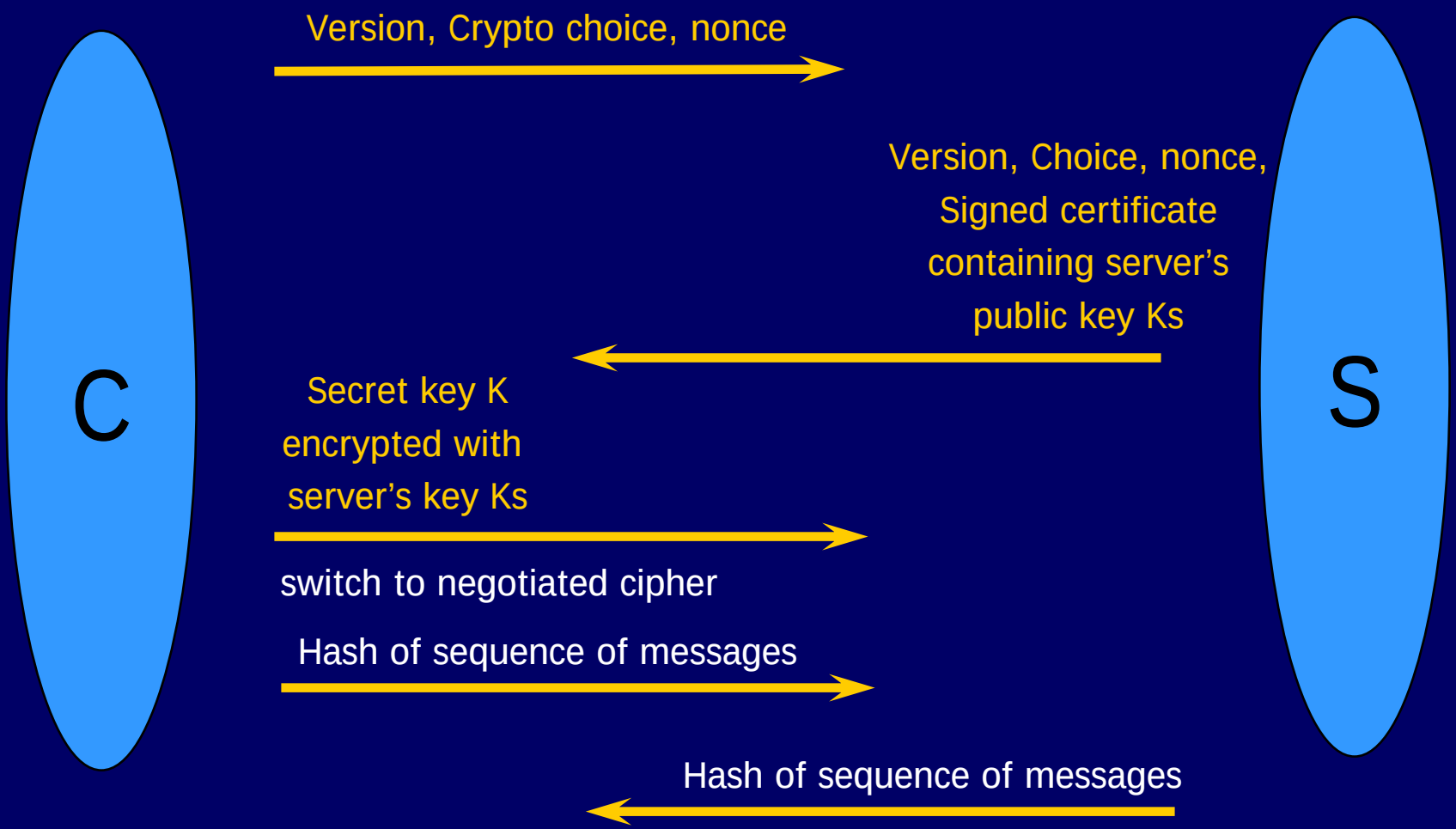
Public-Key Infrastructure (PKI)

- Anyone can send Bob a secret message
  - Provided they know Bob's public key
- How do we know a key belongs to Bob?
  - If imposter substitutes another key, can read Bob's mail
- One solution: PKI
  - Trusted root authority (VeriSign, IBM, United Nations)
    - Everyone must know the verification key of root authority
    - Check your browser; there are hundreds!!
  - Root authority can sign certificates
  - Certificates identify others, including other authorities
  - Leads to certificate chains

Back to SSL/TLS



Use of cryptography



Crypto Summary

- Encryption scheme:

  encrypt(key, plaintext) decrypt(key^{-1}, ciphertext)

- Secret vs. public key
  - Public key: publishing key does not reveal key $^{-1}$
  - Secret key: more efficient, but key = key $^{-1}$
- Hash function
  - Map long text to short hash; ideally, no collisions
  - Keyed hash (MAC) for message authentication
- Signature scheme
  - Private key $^{-1}$ and public key provide authentication

Limitations of cryptography

- Most security problems are not crypto problems
  - This is good
    - Cryptography works!
  - This is bad
    - People make other mistakes; crypto doesn’t solve them