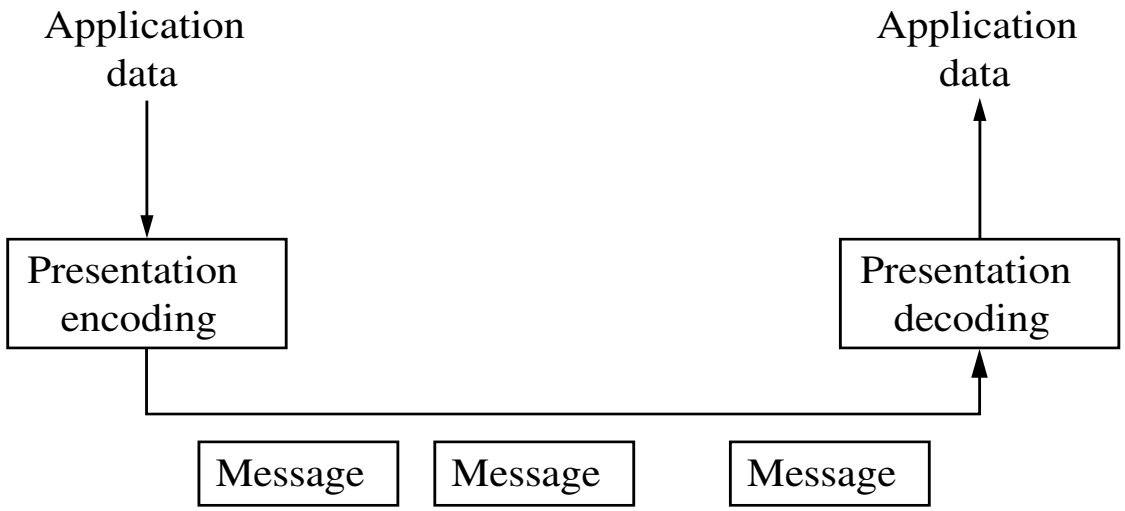
Copyright 1996. All rights reserved. Permission to reproduce this document for not for profit educational purposes is hereby granted. This document may not be reproduced for commercial purposes without the express written consent of the author.

Presentation Formatting

Overview

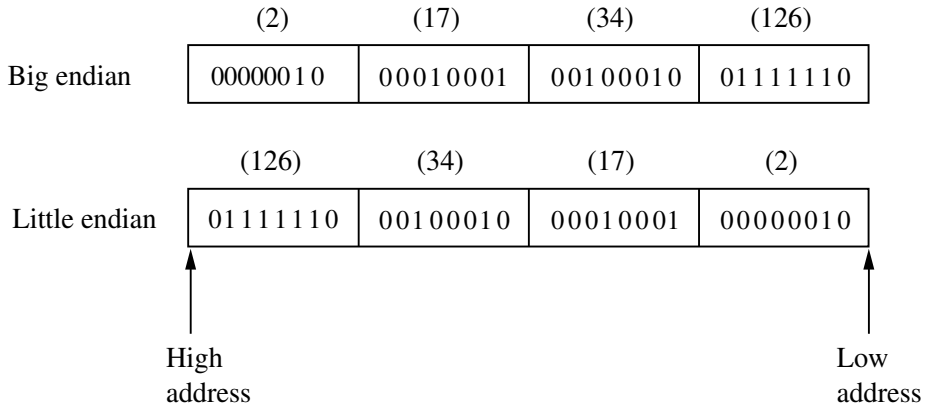


- Marshalling (encoding) application data into messages
- Unmarshalling (decoding) messages into application data

- Data types we consider:
  - integers
  - floating point numbers
  - character strings
  - arrays
  - structures
- Types of data we do not consider (now):
  - images
  - video
  - multimedia documents

Difficulties

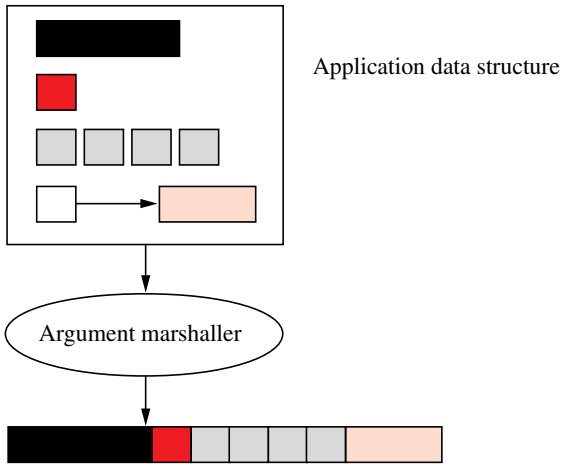
- Representation of base types
  - floating point: IEEE 754 versus non-standard
  - integer: big-endian versus little-endian (e.g., 34,677,374)



- Compiler layout of structures

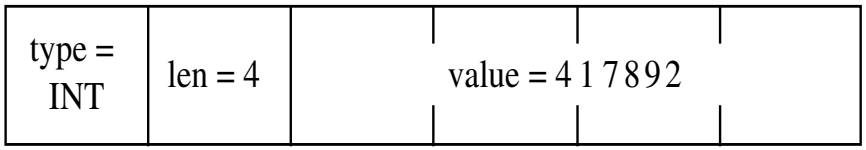
Taxonomy

- Data types
  - base types (e.g., ints, floats); must convert
  - flat types (e.g., structures, arrays); must pack
  - complex types (e.g., pointers); must linearize

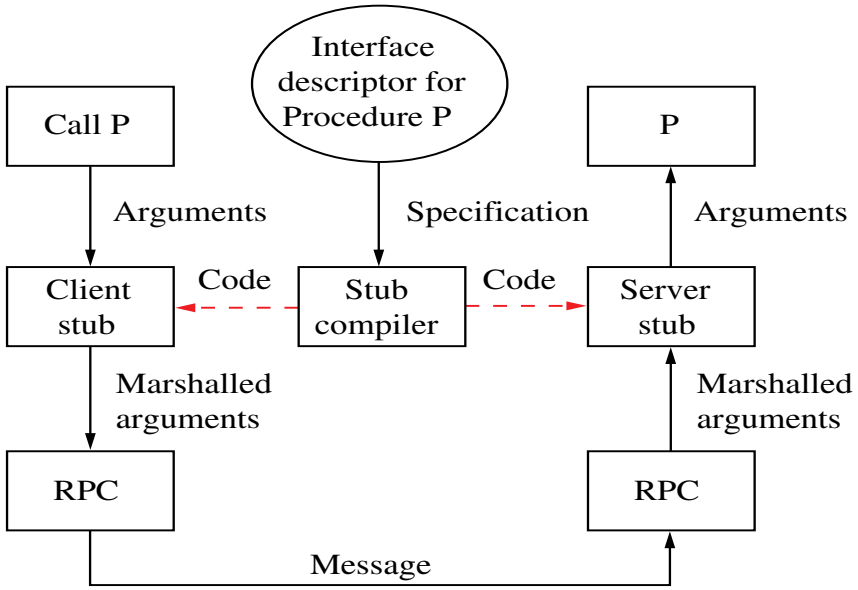


- Conversion Strategy
  - canonical intermediate form
  - receiver-makes-right (an $N \times N$ solution)

- Tagged versus Untagged data



- Stubs
  - compiled
  - interpreted

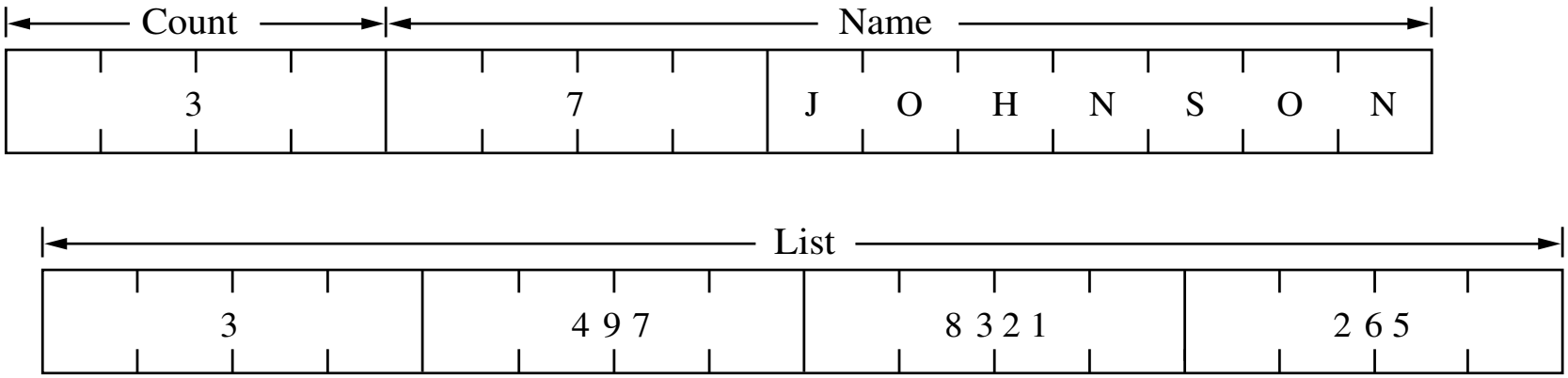


Examples

XDR: eXternal Data Representation

- Defined by Sun for use with SunRPC
- C type system (without function pointers)
- Canonical intermediate form
- Untagged (except array length)
- Compiled stubs

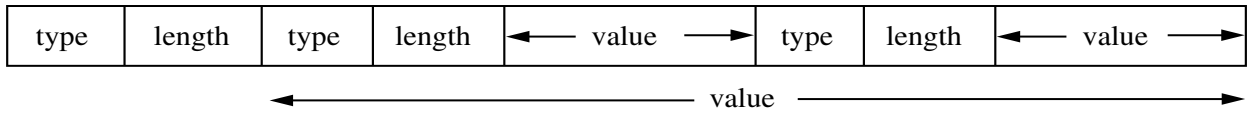
```
#define MAXNAME 256;
#define MAXLIST 100;
struct item {
   int     count;
   char    name[MAXNAME];
   int     list[MAXLIST];
};

bool_t
xdr_item(XDR *xdrs, struct item *ptr)
{
   return(xdr_int(xdrs, &ptr->count) &&
      xdr_string(xdrs, &ptr->name, MAXNAME) &&
      xdr_array(xdrs, &ptr->list, &ptr->count,
                MAXLIST, sizeof(int), xdr_int));
}
```

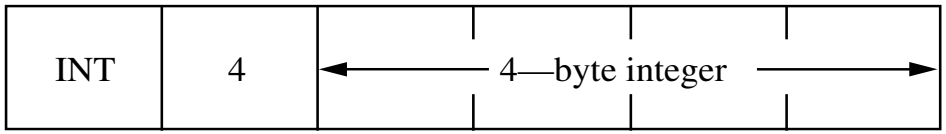

Count
Name
3
7
J O H N S O N
List
3
4 9 7
8 3 2 1
2 6 5

ASN.1: Abstract Syntax Notation One

- An ISO standard
- Essentially the C type system
- Canonical intermediate form
- Tagged
- Compiled or interpretted stubs
- BER: Basic Encoding Rules

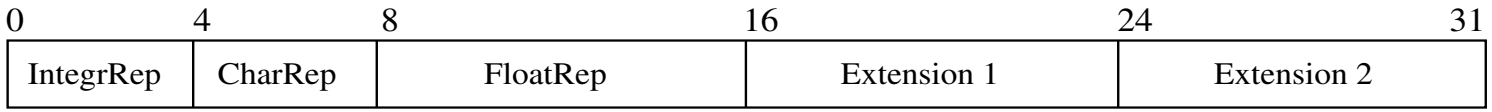
<tag, length, value>





NDR: Network Data Representation

- Defined by DCE
- Essentially the C type system
- Receiver-makes-right (architecture tag)
- Individual data items untagged
- Compiled stubs from IDL



- 4-byte architecture definition tag
  - **`IntegrRep`**
    - 0 = big-endian
    - 1 = little-endian
  - **`CharRep`**
    - 0 = ASCII
    - 1 = EBCDIC
  - **`FloatRep`**
    - 0 = IEEE 754
    - 1 = VAX
    - 2 = Cray
    - 3 = IBM

Data Compression

Data must be encoded into a message. Compression is concerned with with removing *redundancy* from that encoding. There are two classes of compression:

- Lossless: ensures that the data recovered from the compression/decompression process is exactly the same as the original data. Commonly used to compress executable code, text files, and numeric data.

- Lossy: does not promise that the data received is exactly the same as the data sent; removes removes information that it cannot later restore. (Hopefully, no one will notice.) Commonly used to compress digital imagery, including video.

Note: The compression/decompression process takes time. Whether or not you compress data (and how much you compress it) depends on whether you have more cycles (for compression) or bandwidth (for transmission).

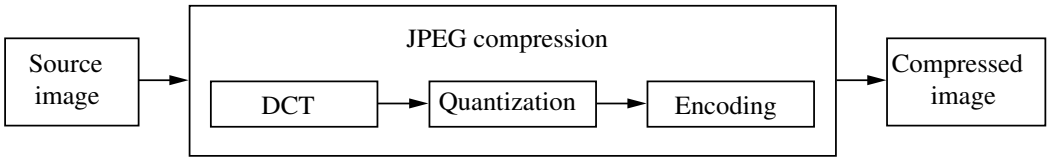
Loss-Less Compression Algorithms

- Run Length Encoding (RLE)
  - example: AAABBCDDDD encoded as 3A2B1C4D
  - good for scanned text (8-to-1 compression ratio)
  - can increase size for data with variation (e.g., some images)
- Differential Pulse Code Modulation (DPCM)
  - example: AAABBCDDDD encoded as A0001123333
  - change reference symbol if delta becomes too large
  - works better than RLE for many digital images (1.5-to-1)

- Dictionary-Based Methods
  - build dictionary of common terms (variable length strings)
  - transmit index into dictionary for each term
  - Lempel-Ziv (LZ) compression is the best-known example
  - commonly achieve 2-to-1 ration on text
  - variation of LZ used to compress GIF images
    - first reduce 24-bit color to 8-bit color
    - treat common sequences of pixels as terms in dictionary
    - not uncommon to achieve 10-to-1 compression (x3)

Image Compression

- JPEG: Joint Photographic Expert Group (ISO/ITU)
- Lossy still image compression
- Three phase process



  - process in 8x8block chunks (macroblock)
  - greyscale: each pixel is a single value
  - color: each pixel is three values (YUV)
  - DCT: transforms signal from spatial domain into an equivalent signal in the frequency domain (loss-less)
  - apply a quantization to the results (lossy)
  - RLE-like encoding (loss-less)

Quantization Phase

$$\mathbf{QuantizedValue}(i, j) = \mathbf{IntegerRound}(\mathbf{\mathit{DCT}}(i, j)/\mathbf{Quantum}(\mathbf{i, j}))$$

$$\mathbf{IntegerRound}(x) = \begin{cases} [x + 0.5] \text{ if } x \text{ Š } 0 \\ [x - 0.5] \text{ if } x > 0 \end{cases}$$

Decompression is then simply defined as

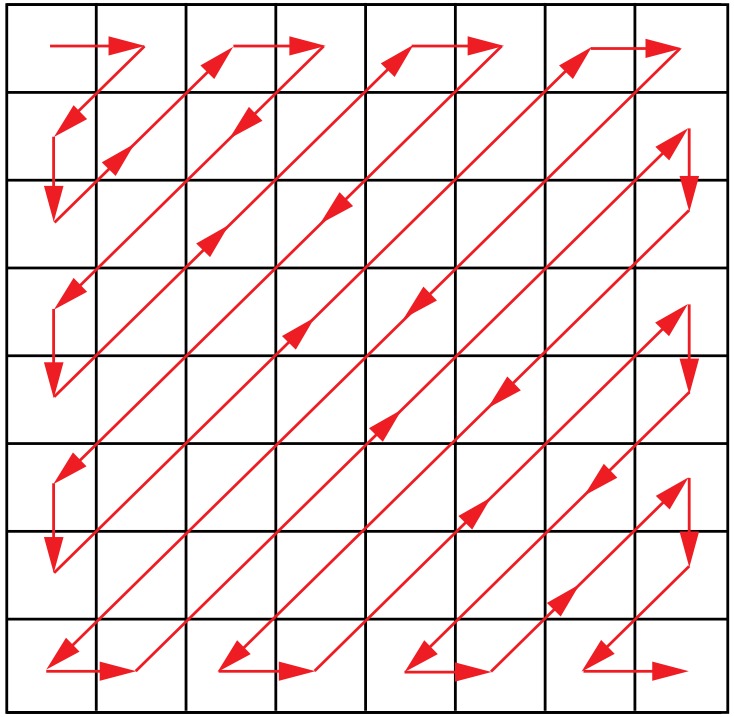
Decompression is then simply defined as

$DCT\,(i, j) = \mathbf{QuantizedValue}\,(i, j)$ x $\mathbf{Quantum}\,(i, j)$

$\mathbf{Quantum} =$

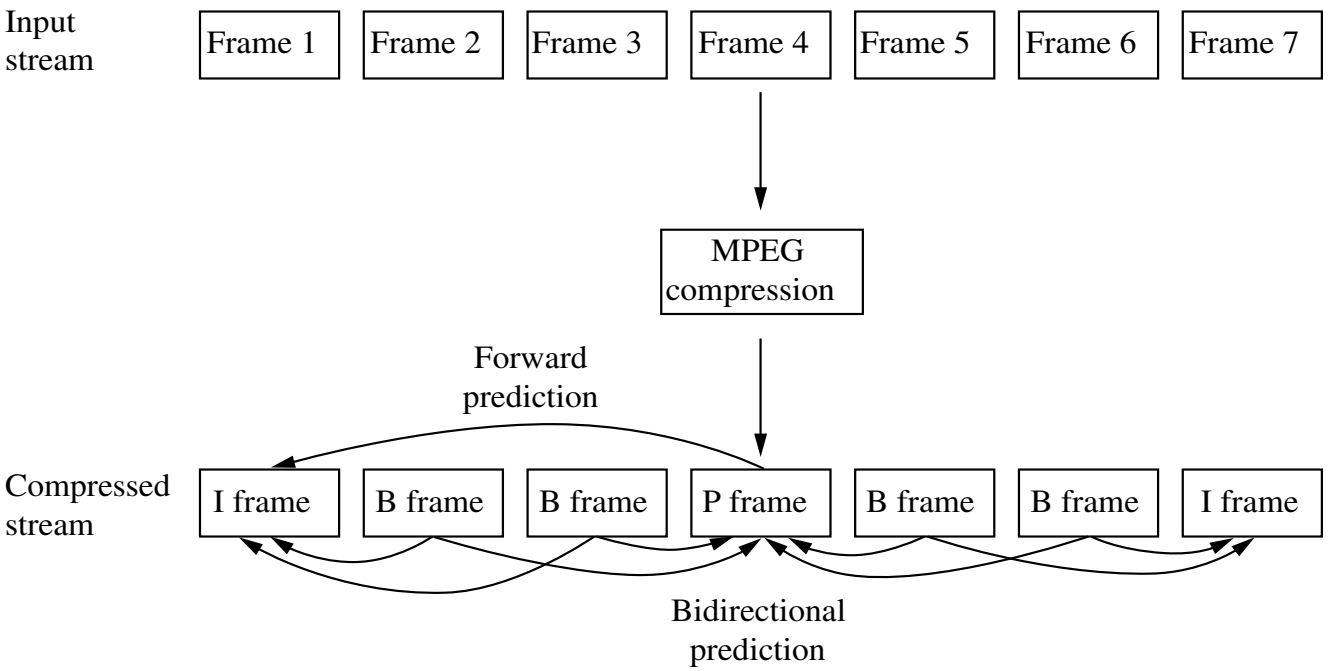
$$\begin{bmatrix} 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 \\ 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 \\ 7 & 9 & 11 & 13 & 15 & 17 & 19 & 21 \\ 9 & 11 & 13 & 15 & 17 & 19 & 21 & 23 \\ 11 & 13 & 15 & 17 & 19 & 21 & 23 & 25 \\ 13 & 15 & 17 & 19 & 21 & 23 & 25 & 27 \\ 15 & 17 & 19 & 21 & 23 & 25 & 27 & 29 \\ 17 & 19 & 21 & 23 & 25 & 27 & 29 & 31 \end{bmatrix}$$

Encoding Phase



Video Compression

- MPEG: Motion Picture Expert Group
- Lossy compression of video
- First approximation: JPEG on each frame
- Added compression: remove inter-frame redundancy
- Frame types
  - I frames: intrapicture
  - P frames: predicted picture
  - B frames: bidirectional predicted picture

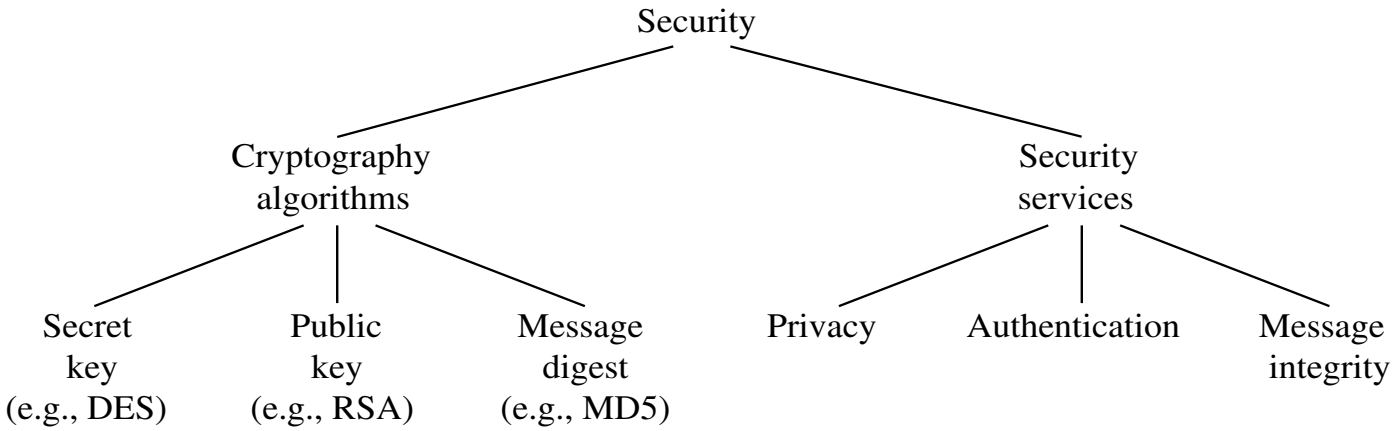


- Example sequence transmitted as I P B B I B B

- B and P frames
  - coordinate for the macroblock in the frame
  - motion vector relative to previous reference frame
  - motion vector relative to subsequent reference frame (B only)
  - delta for each pixel in the macro block
- Effectiveness
  - typically 90-to-1
  - as high as 150-to-1
  - 30-to-1 for I frames
  - P & B frames get another 3 to 5x

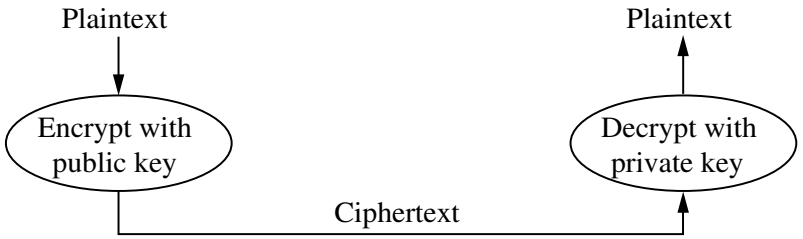
Security

- Cryptography functions
  - Secret key (e.g., DES)
  - Public key (e.g., RSA)
  - Message digest (e.g., MD5)
- Security services
  - Privacy: preventing unauthorized release of information
  - Authentication: verifying identity of the remote participant
  - Integrity: making sure message has not been altered

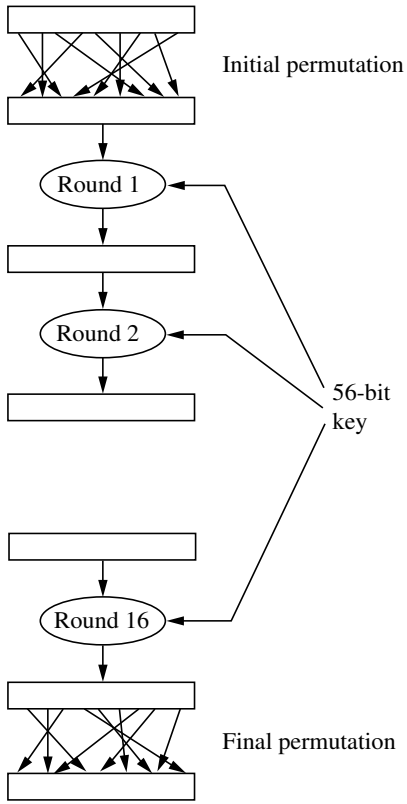

Security
Cryptography algorithms
Security services
Secret key (e.g., DES)
Public key (e.g., RSA)
Message digest (e.g., MD5)
Privacy
Authentication
Message integrity

Encryption Algorithms

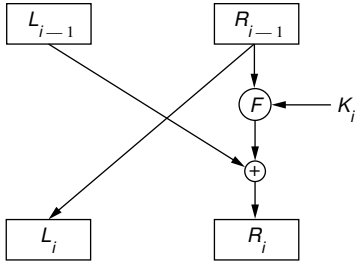
Private Key (DES)



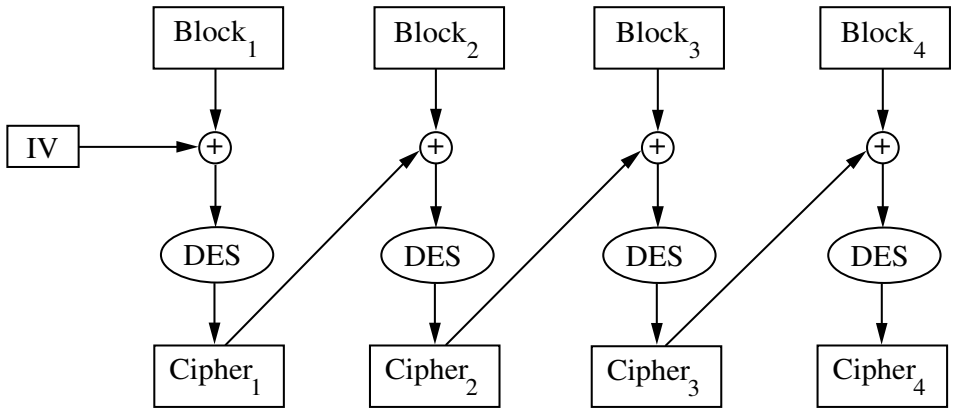
- 64-bit key (56-bits + 8-bit parity)
- 16 rounds



- Each round



- Function F and generation of K_i for each round not shown
- Repeat for larger messages



Public Key (RSA)

- Generate a public and private key
  - choose two large prime numbers *p* and *q* (each 256 bits)
  - multiply *p* and *q* together to get *n*
  - chose the encryption key *e*, such that *e* and $(p^{-1}) \times (q^{-1})$ are relatively prime.
  - two numbers are relatively prime if they have no common factor greater than one.
  - compute decryption key *d* such that

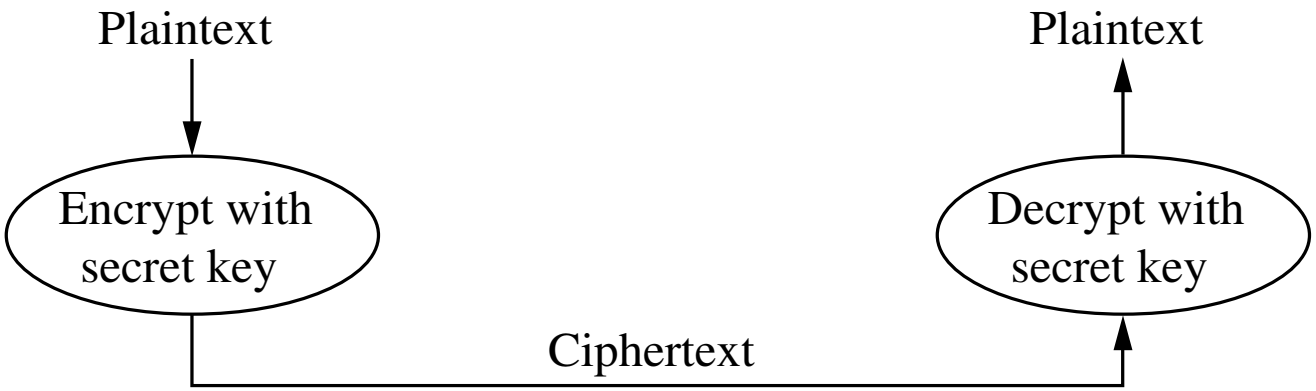
    $$d = e^{-1} \bmod((p^{-1}) \times (q^{-1}))$$

  - construct public key as <*e*, *n*>
  - construct private key as <*d*, *n*>
  - discard (do not disclose) original primes *p* and *q*

- Encryption & Decryption

$c = m^e \bmod n$

$m = c^d \bmod n$



Message Digest

- Cryptographic checksum:

  just as a regular checksum protects the receiver from accidental changes to the message, a cryptographic checksum protects the receiver from malicious changes to the message.

- One-way function:

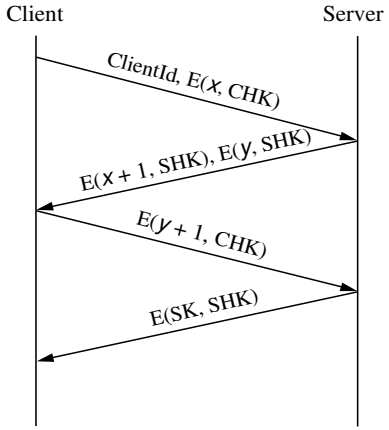
  given a cryptographic checksum for a message, it is virtually impossible to figure out what message produced that checksum; it is not computationally feasible to find two messages that hash to the same cryptographic checksum.

- Relevance:

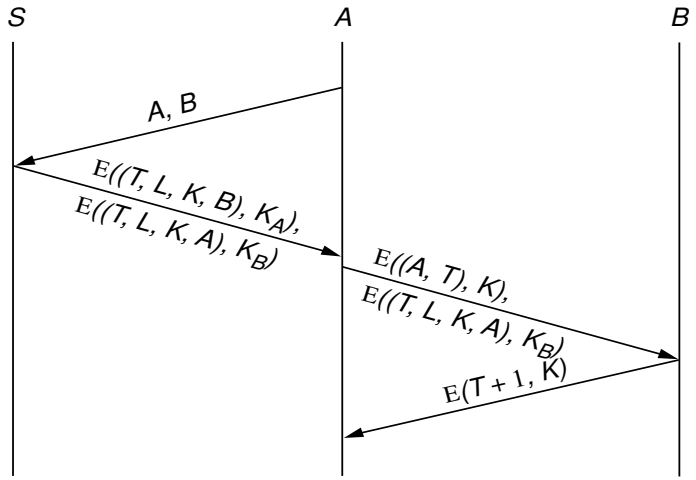
  if you are given a checksum for a message and you are able to compute exactly the same checksum for that message, then it is highly likely this message produced the checksum you were given.

Authentication Protocols

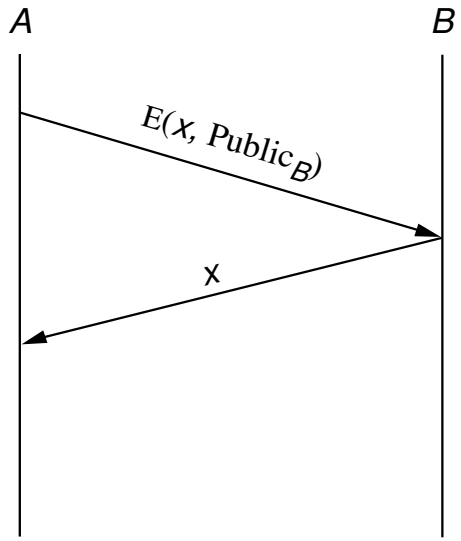
- Three-way handshake



- Trusted third party (Kerberos)



- Public key authentication



Message Integrity Protocols

- Digital signature using RSA
  - special case of a message integrity where the code can only have been generated by one participant
  - compute signiture with private key and verify with public key
- Keyed MD5
  - sender

    ***m* + MD5(*m* + *k*) + E(*k*, *private*)**
  - receiver
    - recovers random key using the sender's public key
    - applies MD5 to the concatenation of this random key message
    - compares result with checksum sent with message

- MD5 with RSA signature
  - sender
    - ***m* + MD5(*m*) + E(MD5(*m*), *private*)**
  - receiver
    - decrypts signature with sender's pubic key
    - compares result with MD5 checksum sent with message