Chapter 11: Cipher Techniques

- Some Problems
- Types of Ciphers
- Networks
- Examples

Overview

- Problems
  - What can go wrong if you naively use ciphers
- Cipher types
  - Stream or block ciphers?
- Networks
  - Link vs end-to-end use
- Examples
  - Privacy-Enhanced Electronic Mail (PEM)
  - Secure Socket Layer (SSL)
  - Security at the Network Layer (IPsec)

Problems

- Using cipher requires knowledge of environment, and threats in the environment, in which cipher will be used
  - Is the set of possible messages small?
  - Do the messages exhibit regularities that remain after encipherment?
  - Can an active wiretapper rearrange or change parts of the message?

Attack #1: Precomputation

- Set of possible messages *M* small
- Public key cipher *f* used
- Idea: precompute set of possible ciphertexts *f*(*M*), build table (*m*, *f*(*m*))
- When ciphertext *f*(*m*) appears, use table to find *m*
- Also called *forward searches*

Example

- Cathy knows Alice will send Bob one of two messages: enciphered BUY, or enciphered SELL
- Using public key e_{Bob}, Cathy precomputes $m_1 = \{ \text{BUY} \}\, e_{Bob}$, $m_2 = \{ \text{SELL} \}\, e_{Bob}$
- Cathy sees Alice send Bob m_2
- Cathy knows Alice sent SELL

May Not Be Obvious

- Digitized sound
  - Seems like far too many possible plaintexts
    - Initial calculations suggest 2^{32} such plaintexts
  - Analysis of redundancy in human speech reduced this to about 100,000 ($\approx 2^{17}$)
    - This is small enough to worry about precomputation attacks

Misordered Blocks

- Alice sends Bob message
  - $n_{Bob} = 77$, $e_{Bob} = 17$, $d_{Bob} = 53$
  - Message is LIVE (11 08 21 04)
  - Enciphered message is 44 57 21 16
- Eve intercepts it, rearranges blocks
  - Now enciphered message is 16 21 57 44
- Bob gets enciphered message, deciphers it
  - He sees EVIL

Notes

- Digitally signing each block won’t stop this attack
- Two approaches:
  - Cryptographically hash the *entire* message and sign it
  - Place sequence numbers in each block of message, so recipient can tell intended order
    - Then you sign each block

Statistical Regularities

- If plaintext repeats, ciphertext may too
- Example using DES:
  - input (in hex):

    3231 3433 3635 3837 3231 3433 3635 3837
  - corresponding output (in hex):

    ef7c 4bb2 b4ce 6f3b ef7c 4bb2 b4ce 6f3b
- Fix: cascade blocks together (chaining)
  - More details later

What These Mean

- Use of strong cryptosystems, well-chosen (or random) keys not enough to be secure
- Other factors:
  - Protocols directing use of cryptosystems
  - Ancillary information added by protocols
  - Implementation (not discussed here)
  - Maintenance and operation (not discussed here)

Stream, Block Ciphers

- E encipherment function
  - $E_k(b)$ encipherment of message b with key k
  - In what follows, $m = b_1 b_2 \ldots$, each b_i of fixed length
- Block cipher
  - $E_k(m) = E_k(b_1)E_k(b_2) \ldots$
- Stream cipher
  - $k = k_1 k_2 \ldots$
  - $E_k(m) = E_{k1}(b_1)E_{k2}(b_2) \ldots$
  - If $k_1 k_2 \ldots$ repeats itself, cipher is *periodic* and the kength of its period is one cycle of $k_1 k_2 \ldots$

Examples

- Vigenère cipher
  - b_i = 1 character, $k = k_1k_2 \ldots$ where k_i = 1 character
  - Each b_i enciphered using $k_{i \text{ mod length}(k)}$
  - Stream cipher
- DES
  - b_i = 64 bits, k = 56 bits
  - Each b_i enciphered separately using k
  - Block cipher

Stream Ciphers

- Often (try to) implement one-time pad by xor'ing each bit of key with one bit of message
  - Example:

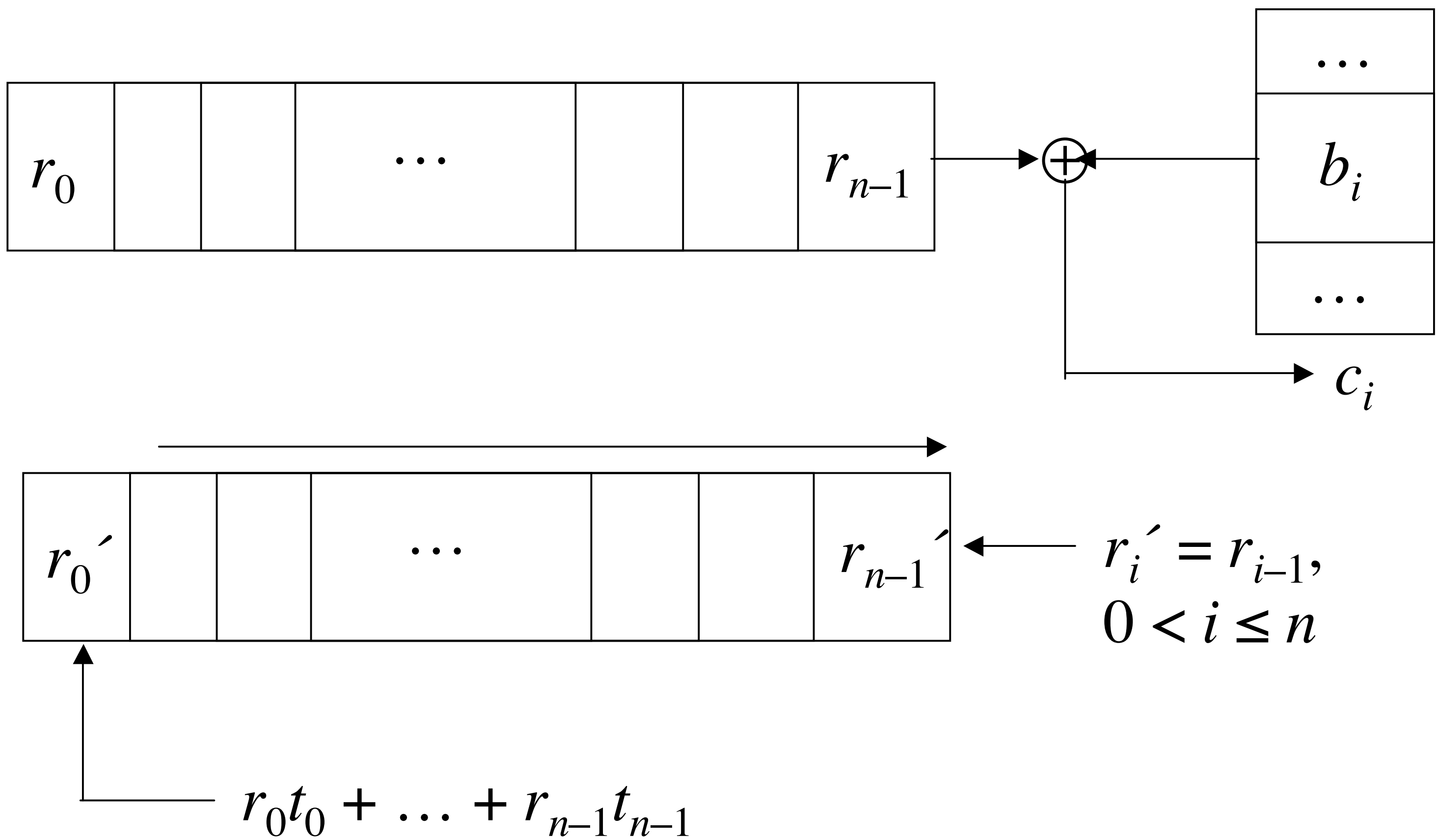
$$m = 00101$$
$$k = 10010$$
$$c = 10111$$

- But how to generate a good key?

Synchronous Stream Ciphers

- n-stage Linear Feedback Shift Register: consists of
  - n bit register $r = r_0 \ldots r_{n-1}$
  - n bit tap sequence $t = t_0 \ldots t_{n-1}$
  - Use:
    - Use r_{n-1} as key bit
    - Compute $x = r_0 t_0 \oplus \ldots \oplus r_{n-1} t_{n-1}$
    - Shift r one bit to right, dropping r_{n-1}, x becomes r_0

Operation



Example

- 4-stage LFSR; t = 1001

r	k_i	*new bit computation*	*new r*
0010	0	01⊕00⊕10⊕01 = 0	0001
0001	1	01⊕00⊕00⊕11 = 1	1000
1000	0	11⊕00⊕00⊕01 = 1	1100
1100	0	11⊕10⊕00⊕01 = 1	1110
1110	0	11⊕10⊕10⊕01 = 1	1111
1111	1	11⊕10⊕10⊕11 = 0	0111
1110	0	11⊕10⊕10⊕11 = 1	1011

  – Key sequence has period of 15 (010001111010110)

NLFSR

- n-stage Non-Linear Feedback Shift Register: consists of
  - n bit register $r = r_0 \ldots r_{n-1}$
  - Use:
    - Use r_{n-1} as key bit
    - Compute $x = f(r_0, \ldots, r_{n-1})$; f is any function
    - Shift r one bit to right, dropping r_{n-1}, x becomes r_0

  Note same operation as LFSR but more general bit replacement function

Example

- 4-stage NLFSR; $f(r_0, r_1, r_2, r_3) = (r_0 \,\&\, r_2) \mid r_3$

r	k_i	*new bit computation*	*new r*
1100	0	(1 & 0) | 0 = 0	0110
0110	0	(0 & 1) | 0 = 0	0011
0011	1	(0 & 1) | 1 = 1	1001
1001	1	(1 & 0) | 1 = 1	1100
1100	0	(1 & 0) | 0 = 0	0110
0110	0	(0 & 1) | 0 = 0	0011
0011	1	(0 & 1) | 1 = 1	1001

  – Key sequence has period of 4 (0011)

Eliminating Linearity

- NLFSRs not common
  - No body of theory about how to design them to have long period
- Alternate approach: *output feedback mode*
  - For E encipherment function, k key, r register:
    - Compute $r' = E_k(r)$; key bit is rightmost bit of r'
    - Set r to r' and iterate, repeatedly enciphering register and extracting key bits, until message enciphered
  - Variant: use a counter that is incremented for each encipherment rather than a register
    - Take rightmost bit of $E_k(i)$, where i is number of encipherment

Self-Synchronous Stream Cipher

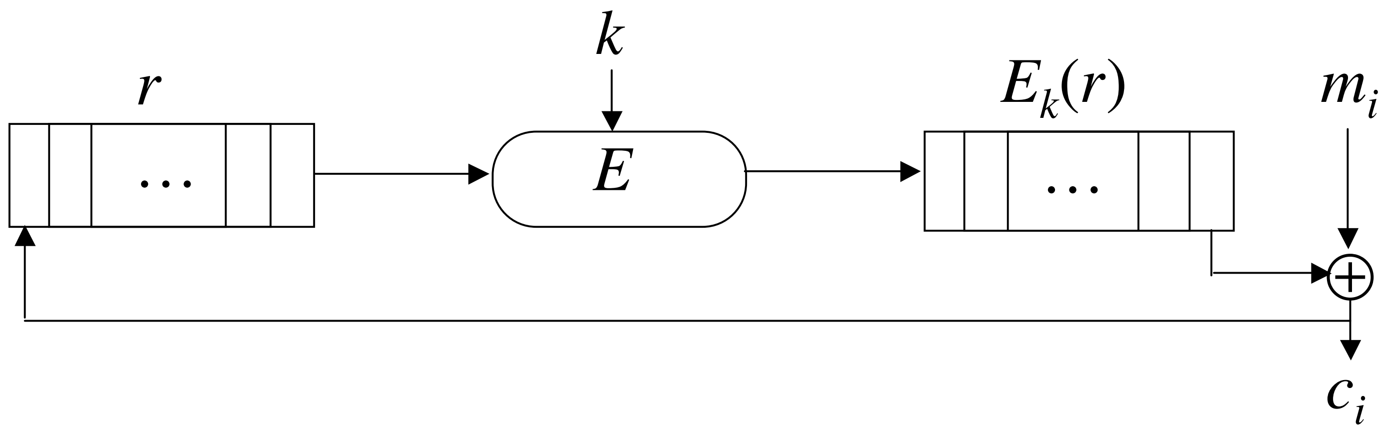
- Take key from message itself (*autokey*)
- Example: Vigenère, key drawn from plaintext
  - *key* `XTHEBOYHASTHEBA`
  - *plaintext* `THEBOYHASTHEBAG`
  - *ciphertext* `QALFPNFHSLALFCT`
- Problem:
  - Statistical regularities in plaintext show in key
  - Once you get any part of the message, you can decipher more

Another Example

- Take key from ciphertext (*autokey*)
- Example: Vigenère, key drawn from ciphertext
  - *key* `XQXBCQOVVNGNRTT`
  - *plaintext* `THEBOYHASTHEBAG`
  - *ciphertext* `QXBCQOVVNGNRTTM`
- Problem:
  - Attacker gets key along with ciphertext, so deciphering is trivial

Variant

- Cipher feedback mode: 1 bit of ciphertext fed into *n* bit register
  - Self-healing property: if ciphertext bit received incorrectly, it and next *n* bits decipher incorrectly; but after that, the ciphertext bits decipher correctly
  - Need to know *k*, *E* to decipher ciphertext



Block Ciphers

- Encipher, decipher multiple bits at once
- Each block enciphered independently
- Problem: identical plaintext blocks produce identical ciphertext blocks
  - Example: two database records
    - `MEMBER: HOLLY INCOME $100,000`
    - `MEMBER: HEIDI INCOME $100,000`
  - Encipherment:
    - `ABCQZRME GHQMRSIB CTXUVYSS RMGRPFQN`
    - `ABCQZRME ORMPABRZ CTXUVYSS RMGRPFQN`

Solutions

- Insert information about block's position into the plaintext block, then encipher
- *Cipher block chaining*:
  - Exclusive-or current plaintext block with previous ciphertext block:
    - $c_0 = E_k(m_0 \oplus I)$
    - $c_i = E_k(m_i \oplus c_{i-1})$ for i > 0

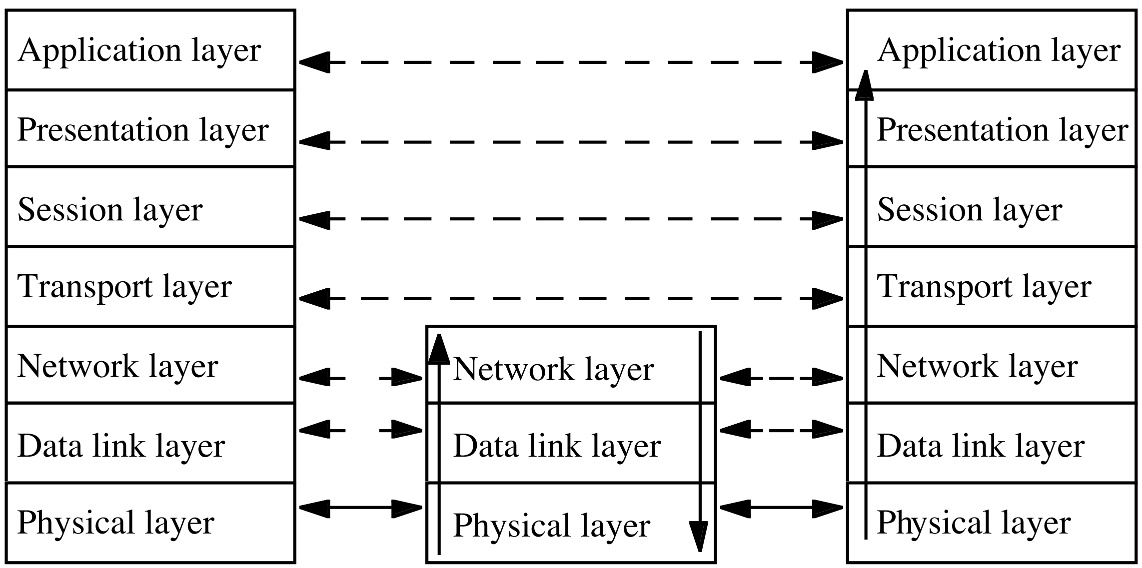
    where *I* is the initialization vector

Multiple Encryption

- Double encipherment: $c = E_{k'}(E_k(m))$
  - Effective key length is $2n$, if k, k' are length n
  - Problem: breaking it requires 2^{n+1} encryptions, not 2^{2n} encryptions
- Triple encipherment:
  - EDE mode: $c = E_k(D_{k'}(E_k(m))$
    - Problem: chosen plaintext attack takes $O(2^n)$ time using 2^n ciphertexts
  - Triple encryption mode: $c = E_k(E_{k'}(E_{k''}(m))$
    - Best attack requires $O(2^{2n})$ time, $O(2^n)$ memory

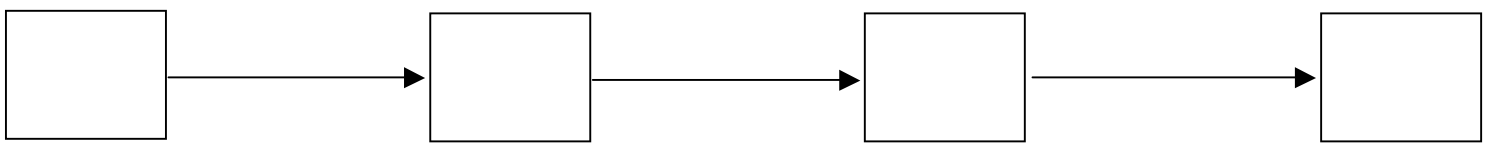
Networks and Cryptography

- ISØOSI model
- Conceptually, each host has peer at each layer
  - Peers communicate with peers at same layer

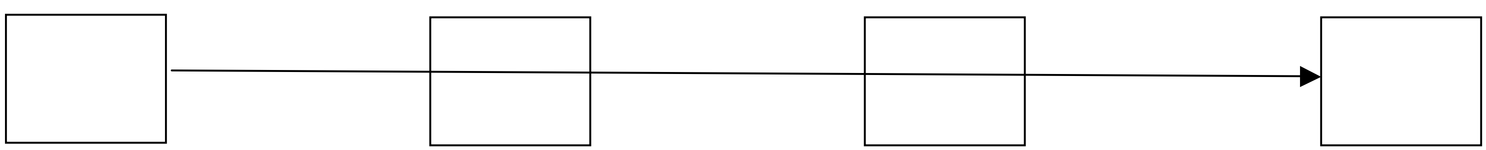


Link and End-to-End Protocols

Link Protocol



End-to-End (or E2E) Protocol



Encryption

- Link encryption
  - Each host enciphers message so host at "next hop" can read it
  - Message can be read at intermediate hosts
- End-to-end encryption
  - Host enciphers message so host at other end of communication can read it
  - Message cannot be read at intermediate hosts

Examples

- TELNET protocol
  - Messages between client, server enciphered, and encipherment, decipherment occur only at these hosts
  - End-to-end protocol
- PPP Encryption Control Protocol
  - Host gets message, deciphers it
    - Figures out where to forward it
    - Enciphers it in appropriate key and forwards it
  - Link protocol

Cryptographic Considerations

- Link encryption
  - Each host shares key with neighbor
  - Can be set on per-host or per-host-pair basis
    - Windsor, stripe, seaview each have own keys
    - One key for (windsor, stripe); one for (stripe, seaview); one for (windsor, seaview)
- End-to-end
  - Each host shares key with destination
  - Can be set on per-host or per-host-pair basis
  - Message cannot be read at intermediate nodes

Traffic Analysis

- Link encryption
  - Can protect headers of packets
  - Possible to hide source and destination
    - Note: may be able to deduce this from traffic flows
- End-to-end encryption
  - Cannot hide packet headers
    - Intermediate nodes need to route packet
  - Attacker can read source, destination

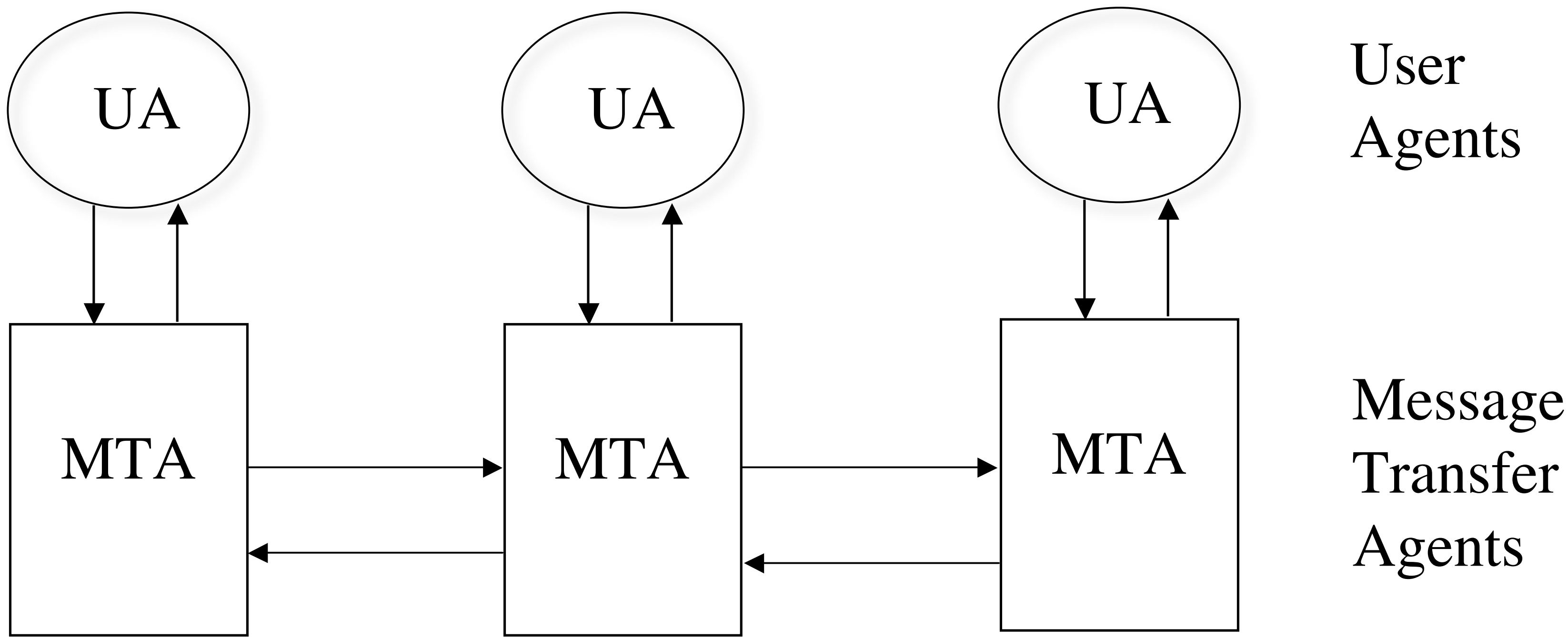
Example Protocols

- Privacy-Enhanced Electronic Mail (PEM)
  - Applications layer protocol
- Secure Socket Layer (SSL)
  - Transport layer protocol
- IP Security (IPSec)
  - Network layer protocol

Goals of PEM

1. Confidentiality
   - Only sender and recipient(s) can read message
2. Origin authentication
   - Identify the sender precisely
3. Data integrity
   - Any changes in message are easy to detect
4. Non-repudiation of origin
   - Whenever possible …

Message Handling System



Design Principles

- Do not change related existing protocols
  - Cannot alter SMTP
- Do not change existing software
  - Need compatibility with existing software
- Make use of PEM optional
  - Available if desired, but email still works without them
  - Some recipients may use it, others not
- Enable communication without prearrangement
  - Out-of-bands authentication, key exchange problematic

Basic Design: Keys

- Two keys
  - *Interchange keys* tied to sender, recipients and is static (for some set of messages)
    - Like a public/private key pair
    - Must be available *before* messages sent
  - *Data exchange keys* generated for each message
    - Like a session key, session being the message

Basic Design: Sending

Confidentiality

- m message
- k_s data exchange key
- k_B Bob's interchange key

$$\text{Alice} \xrightarrow{\{ m \} k_s \,||\, \{ k_s \} k_B} \text{Bob}$$

Basic Design: Integrity

Integrity and authentication:

- m message
- $h(m)$ hash of message m —Message Integrity Check (MIC)
- k_A Alice's interchange key

$$m \{ h(m) \} k_A$$

Alice ⟶ Bob

Non-repudiation: if k_A is Alice's private key, this establishes that Alice's private key was used to sign the message

Basic Design: Everything

Confidentiality, integrity, authentication:

- Notations as in previous slides
- If k_A is private key, get non-repudiation too

$$\text{Alice} \xrightarrow{\{ m \}\, k_s \;||\; \{ h(m) \}\, k_A \;||\; \{ k_s \}\, k_B} \text{Bob}$$

Practical Considerations

- Limits of SMTP
  - Only ASCII characters, limited length lines
- Use encoding procedure
  1. Map local char representation into canonical format
     - Format meets SMTP requirements
  2. Compute and encipher MIC over the canonical format; encipher message if needed
  3. Map each 6 bits of result into a character; insert newline after every 64th character
  4. Add delimiters around this ASCII message

Problem

- Recipient without PEM-compliant software cannot read it
  - If only integrity and authentication used, should be able to read it
- Mode MIC-CLEAR allows this
  - Skip step 3 in encoding procedure
  - Problem: some MTAs add blank lines, delete trailing white space, or change end of line character
  - Result: PEM-compliant software reports integrity failure

PEM vs. PGP

- Use different ciphers
  - PGP uses IDEA cipher
  - PEM uses DES in CBC mode
- Use different certificate models
  - PGP uses general “web of trust”
  - PEM uses hierarchical certification structure
- Handle end of line differently
  - PGP remaps end of line if message tagged “text”, but leaves them alone if message tagged “binary”
  - PEM always remaps end of line

SSL

- Transport layer security
  - Provides confidentiality, integrity, authentication of endpoints
  - Developed by Netscape for WWW browsers and servers
- Internet protocol version: TLS
  - Compatible with SSL
  - Not yet formally adopted

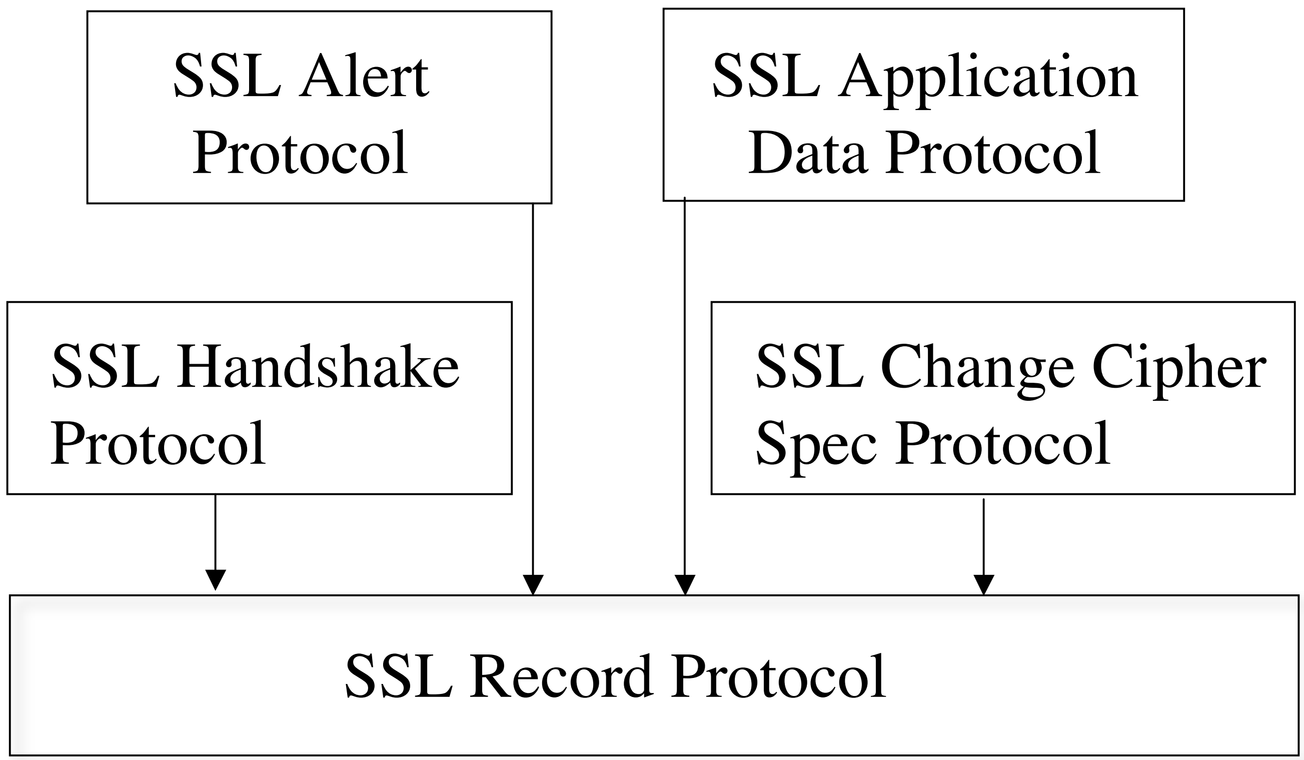
SSL Session

- Association between two peers
  - May have many associated connections
  - Information for each association:
    - Unique session identifier
    - Peer's X.509v3 certificate, if needed
    - Compression method
    - Cipher spec for cipher and MAC
    - "Master secret" shared with peer
      - 48 bits

SSL Connection

- Describes how data exchanged with peer
- Information for each connection
  - Random data
  - Write keys (used to encipher data)
  - Write MAC key (used to compute MAC)
  - Initialization vectors for ciphers, if needed
  - Sequence numbers

Structure of SSL



Supporting Crypto

- All parts of SSL use them
- Initial phase: public key system exchanges keys
  - Messages enciphered using classical ciphers, checksummed using cryptographic checksums
  - Only certain combinations allowed
    - Depends on algorithm for interchange cipher
  - Interchange algorithms: RSA, Diffie-Hellman, Fortezza

RSA: Cipher, MAC Algorithms

Interchange cipher	*Classical cipher*	*MAC Algorithm*
RSA, key ≤ 512 bits	*none*	MD5, SHA
	RC4, 40-bit key	MD5
	RC2, 40-bit key, CBC mode	MD5
	DES, 40-bit key, CBC mode	SHA
RSA	*None*	MD5, SHA
	RC4, 128-bit key	MD5, SHA
	IDEA, CBC mode	SHA
	DES, CBC mode	SHA
	DES, EDE mode, CBC mode	SHA

Diffie-Hellman: Types

- Diffie-Hellman: certificate contains D-H parameters, signed by a CA
  - DSS or RSA algorithms used to sign
- Ephemeral Diffie-Hellman: DSS or RSA certificate used to sign D-H parameters
  - Parameters not reused, so not in certificate
- Anonymous Diffie-Hellman: D-H with neither party authenticated
  - Use is "strongly discouraged" as it is vulnerable to attacks

D-H: Cipher, MAC Algorithms

Interchange cipher	*Classical cipher*	*MAC Algorithm*
Diffie-Hellman, DSS Certificate	DES, 40-bit key, CBC mode	SHA
	DES, CBC mode	SHA
	DES, EDE mode, CBC mode	SHA
Diffie-Hellman, key ≤ 512 bits RSA Certificate	DES, 40-bit key, CBC mode	SHA
	DES, CBC mode	SHA
	DES, EDE mode, CBC mode	SHA

Ephemeral D-H: Cipher, MAC Algorithms

Interchange cipher	*Classical cipher*	*MAC Algorithm*
Ephemeral Diffie-Hellman, DSS Certificate	DES, 40-bit key, CBC mode	SHA
	DES, CBC mode	SHA
	DES, EDE mode, CBC mode	SHA
Ephemeral Diffie-Hellman, key ≤ 512 bits, RSA Certificate	DES, 40-bit key, CBC mode	SHA
	DES, CBC mode	SHA
	DES, EDE mode, CBC mode	SHA

Anonymous D-H: Cipher, MAC Algorithms

Interchange cipher	*Classical cipher*	*MAC Algorithm*
Anonymous D-H, DSS Certificate	RC4, 40-bit key	MD5
	RC4, 128-bit key	MD5
	DES, 40-bit key, CBC mode	SHA
	DES, CBC mode	SHA
	DES, EDE mode, CBC mode	SHA

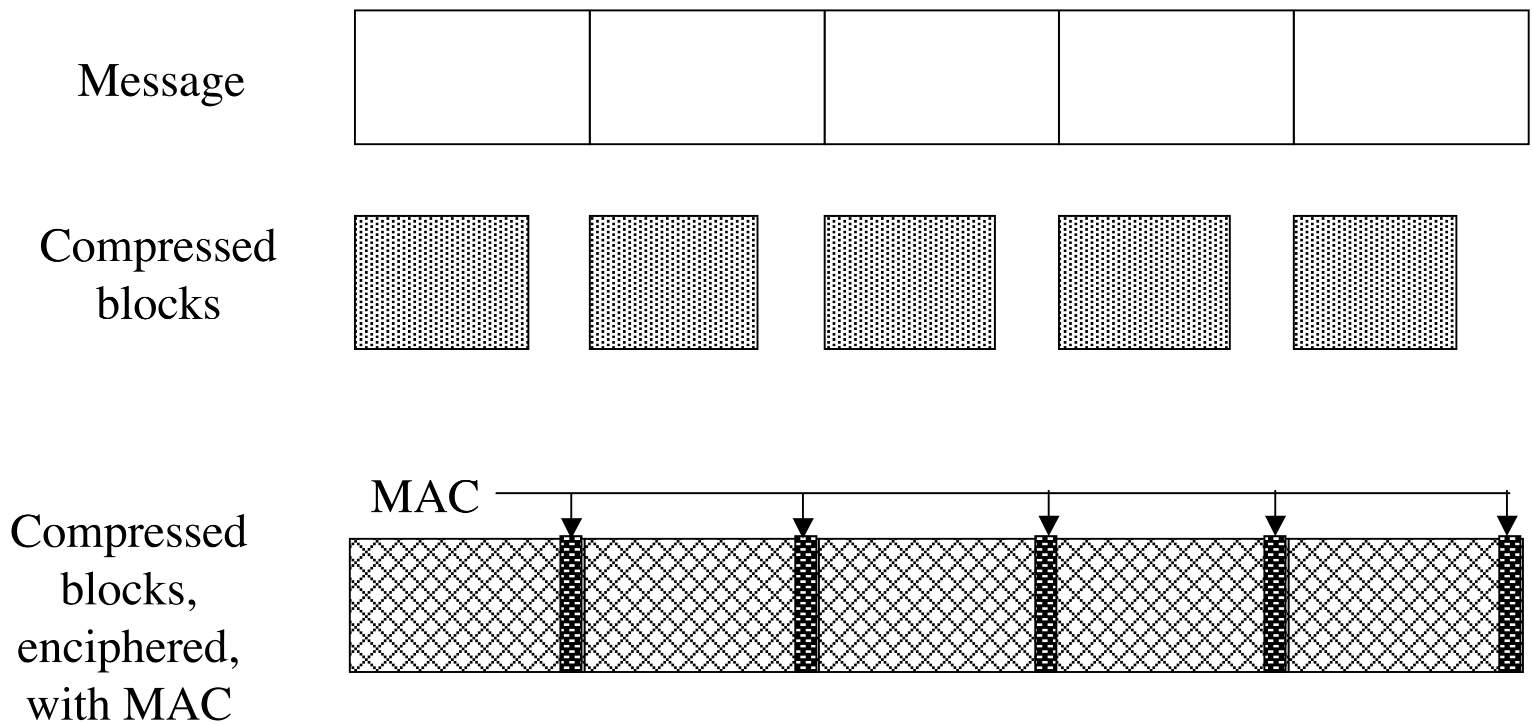
Fortezza: Cipher, MAC Algorithms

Interchange cipher	***Classical cipher***	***MAC Algorithm***
Fortezza key exchange	*none*	SHA
	RC4, 128-bit key	MD5
	Fortezza, CBC mode	SHA

Digital Signatures

- RSA
  - Concatenate MD5 and SHA hashes
  - Sign with public key
- Diffie-Hellman, Fortezza
  - Compute SHA hash
  - Sign appropriately

SSL Record Layer



Record Protocol Overview

- Lowest layer, taking messages from higher
  - Max block size 16,384 bytes
  - Bigger messages split into multiple blocks
- Construction
  - Block b compressed; call it b_c
  - MAC computed for b_c
    - If MAC key not selected, no MAC computed
  - b_c, MAC enciphered
    - If enciphering key not selected, no enciphering done
  - SSL record header prepended

SSL MAC Computation

- Symbols
  - h hash function (MD5 or SHA)
  - k_w write MAC key of entity
  - *ipad* = 0x36, *opad* = 0x5C
    - Repeated to block length (from HMAC)
  - *seq* sequence number
  - *SSL_comp* message type
  - *SSL_len* block length
- MAC

$$h(k_w || opad || h(k_w || ipad || seq || SSL\_comp || SSL\_len || block))$$

SSL Handshake Protocol

- Used to initiate connection
  - Sets up parameters for record protocol
  - 4 rounds
- Upper layer protocol
  - Invokes Record Protocol
- Note: what follows assumes client, server using RSA as interchange cryptosystem

Overview of Rounds

1. Create SSL connection between client, server
2. Server authenticates itself
3. Client validates server, begins key exchange
4. Acknowledgments all around

Handshake Round 1

$\{ v_C \,||\, r_1 \,||\, sid \,||\, ciphers \,||\, comps \}$

Client ⟶ Server

$\{ v \,||\, r_2 \,||\, sid \,||\, cipher \,||\, comp \}$

Client ⟵ Server

v_C	Client's version of SSL
v	Highest version of SSL that Client, Server both understand
r_1, r_2	nonces (timestamp and 28 random bytes)
s_1	Current session id (0 if new session)
s_2	Current session id (if s1 = 0, new session id)
ciphers	Ciphers that client understands
comps	Compression algorithms that client understand
cipher	Cipher to be used
comp	Compression algorithm to be used

Handshake Round 2

Client ← $\{certificate\}$ — Server

Client ← $\{mod \,||\, exp \,||\, \{ h(r_1 \,||\, r_2 \,||\, mod \,||\, exp) \}\, k_S \}$ — Server

Client ← $\{ctype \,||\, gca\}$ — Server

Client ← $\{er2\}$ — Server

Note: if Server not to authenticate itself, only last message sent; third step omitted if Server does not need Client certificate

k_S	Server's private key
ctype	Certificate type requested (by cryptosystem)
gca	Acceptable certification authorities
er2	End round 2 message

Handshake Round 3

Client —— { *pre* } ——→ Server

Both Client, Server compute master secret *master*:

master = MD5(*pre* || SHA('A' || *pre* || r_1 || r_2) ||
MD5(*pre* || SHA('BB' || *pre* || r_1 || r_2) ||
MD5(*pre* || SHA('CCC' || *pre* || r_1 || r_2)

Client —— { *h*(*master* || *opad* || *h*(*msgs* || *master* | *ipad*)) } ——→ Server

msgs	Concatenation of previous messages sent/received this handshake
opad, *ipad*	As above

Handshake Round 4

Client sends "change cipher spec" message using that protocol

Client → Server

$\{ h(master \,||\, opad \,||\, h(msgs \,||\, \mathrm{0x434C4E54} \,||\, master \,||\, ipad)) \}$

Client → Server

Server sends "change cipher spec" message using that protocol

Client ← Server

$\{ h(master \,||\, opad \,||\, h(msgs \,||\, master \,|\, ipad)) \}$

Client ← Server

msgs Concatenation of messages sent/received this handshake in *previous* rounds (does notinclude these messages)

opad, *ipad*, *master* As above

SSL Change Cipher Spec Protocol

- Send single byte
- In handshake, new parameters considered "pending" until this byte received
  - Old parameters in use, so cannot just switch to new ones

SSL Alert Protocol

- Closure alert
  - Sender will send no more messages
  - Pending data delivered; new messages ignored
- Error alerts
  - Warning: connection remains open
  - Fatal error: connection torn down as soon as sent or received

SSL Alert Protocol Errors

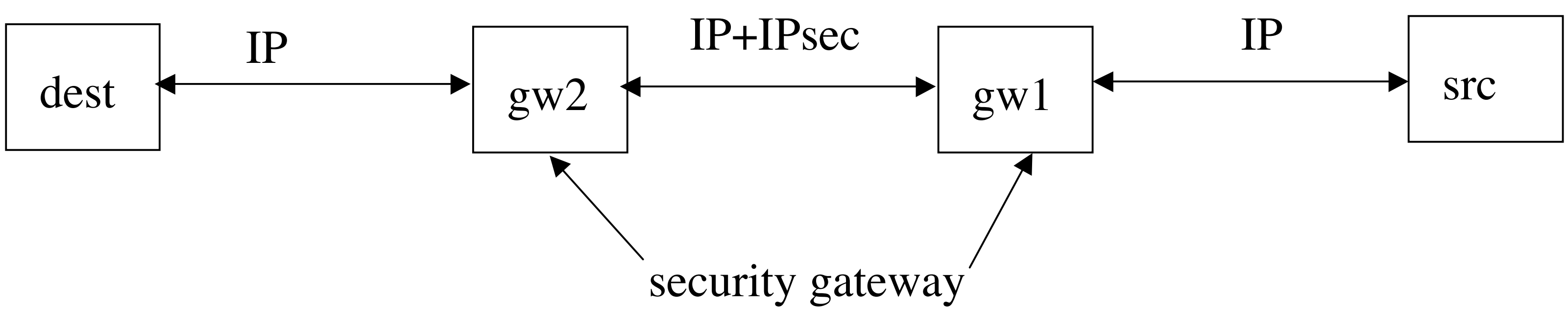
- Always fatal errors:
  - unexpected_message, bad_record_mac, decompression_failure, handshake_failure, illegal_parameter
- May be warnings or fatal errors:
  - no_certificate, bad_certificate, unsupported_certificate, certificate_revoked, certificate_expired, certificate_unknown

SSL Application Data Protocol

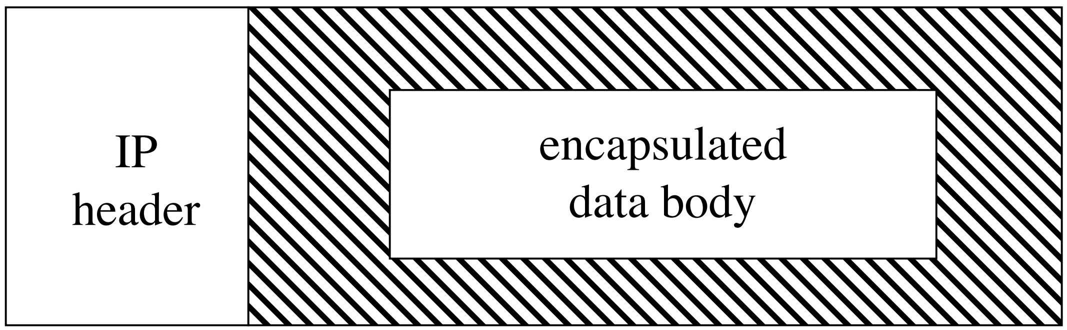
- Passes data from application to SSL Record Protocol layer

IPsec

- Network layer security
  - Provides confidentiality, integrity, authentication of endpoints, replay detection
- Protects all messages sent along a path

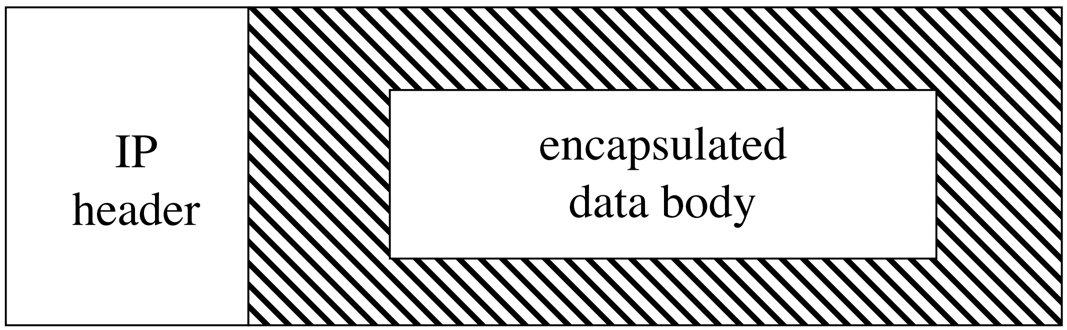


IPsec Transport Mode



- Encapsulate IP packet data area
- Use IP to send IPsec-wrapped data packet
- Note: IP header not protected

IPsec Tunnel Mode



- Encapsulate IP packet (IP header *and* IP data)
- Use IP to send IPsec-wrapped packet
- Note: IP header protected

IPsec Protocols

- Authentication Header (AH)
  - Message integrity
  - Origin authentication
  - Anti-replay
- Encapsulating Security Payload (ESP)
  - Confidentiality
  - Others provided by AH

IPsec Architecture

- Security Policy Database (SPD)
  - Says how to handle messages (discard them, add security services, forward message unchanged)
  - SPD associated with network interface
  - SPD determines appropriate entry from packet attributes
    - Including source, destination, transport protocol

Example

- Goals
  - Discard SMTP packets from host 192.168.2.9
  - Forward packets from 192.168.19.7 without change
- SPD entries

```
src 192.168.2.9, dest 10.1.2.3 to 10.1.2.103, port 25, discard
src 192.168.19.7, dest 10.1.2.3 to 10.1.2.103, port 25, bypass
dest 10.1.2.3 to 10.1.2.103, port 25, apply IPsec
```

- Note: entries scanned in order
  - If no match for packet, it is discarded

IPsec Architecture

- Security Association (SA)
  - Association between peers for security services
    - Identified uniquely by dest address, security protocol (AH or ESP), unique 32-bit number (security parameter index, or SPI)
  - Unidirectional
    - Can apply different services in either direction
  - SA uses either ESP or AH; if both required, 2 SAs needed

SA Database (SAD)

- Entry describes SA; some fields for all packets:
  - AH algorithm identifier, keys
    - When SA uses AH
  - ESP encipherment algorithm identifier, keys
    - When SA uses confidentiality from ESP
  - ESP authentication algorithm identifier, keys
    - When SA uses authentication, integrity from ESP
  - SA lifetime (time for deletion or max byte count)
  - IPsec mode (tunnel, transport, either)

SAD Fields

- Antireplay (inbound only)
  - When SA uses antireplay feature
- Sequence number counter (outbound only)
  - Generates AH or ESP sequence number
- Sequence counter overflow field
  - Stops traffic over this SA if sequence counter overflows
- Aging variables
  - Used to detect time-outs

IPsec Architecture

- Packet arrives
- Look in SPD
  - Find appropriate entry
  - Get dest address, security protocol, SPI
- Find associated SA in SAD
  - Use dest address, security protocol, SPI
  - Apply security services in SA (if any)

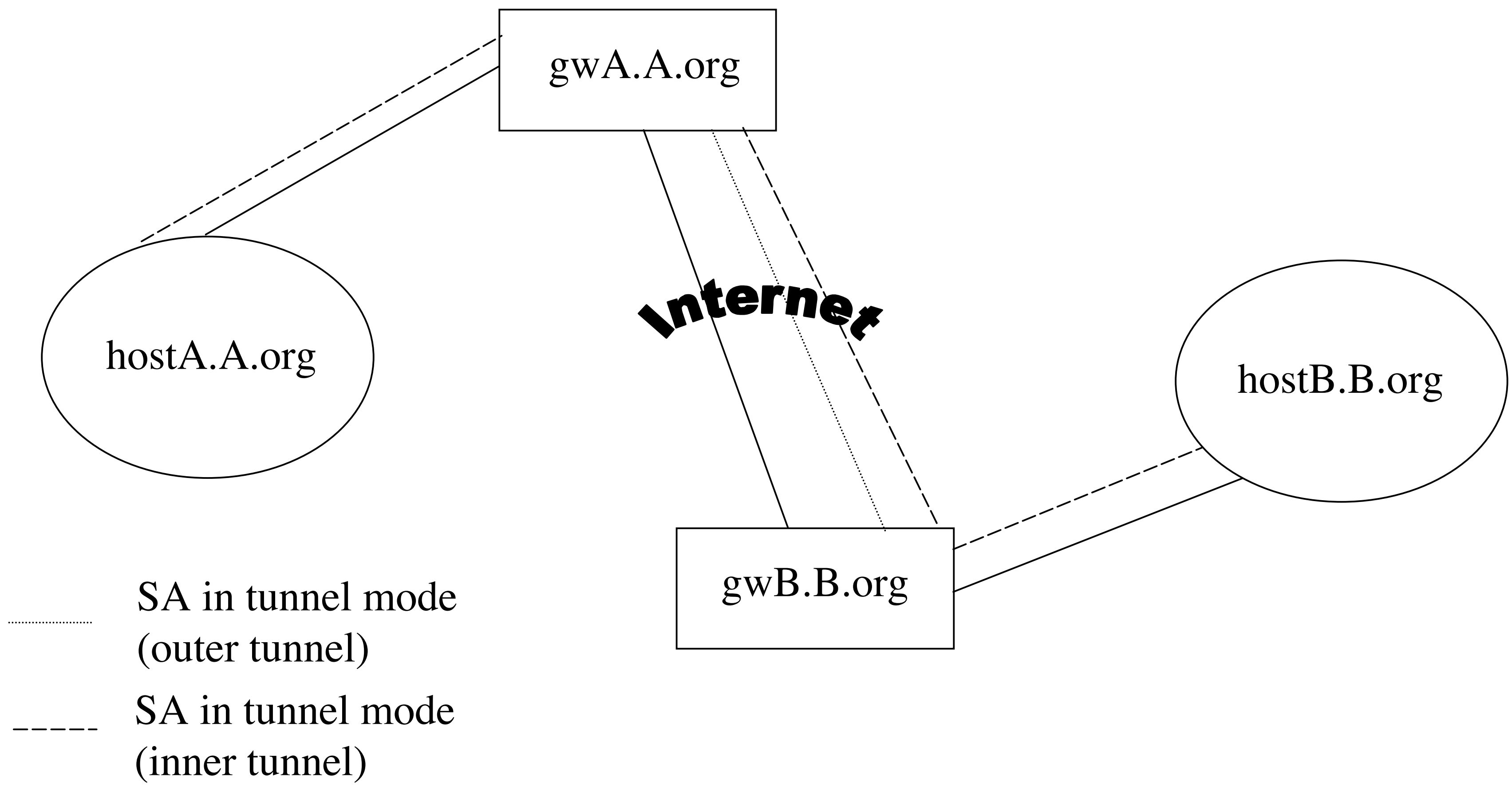
SA Bundles and Nesting

- Sequence of SAs that IPsec applies to packets
  - This is a *SA bundle*
- Nest tunnel mode SAs
  - This is *iterated tunneling*

Example: Nested Tunnels

- Group in A.org needs to communicate with group in B.org
- Gateways of A, B use IPsec mechanisms
  - But the information must be secret to everyone except the two groups, even secret from other people in A.org and B.org
- Inner tunnel: a SA between the hosts of the two groups
- Outer tunnel: the SA between the two gateways

Example: Systems



Example: Packets

IP header from gwA	AH header from gwA	ESP header from gwA	IP header from hostA	AH header from hostA	ESP header from hostA	IP header from hostA	Transport layer headers, data

- Packet generated on hostA
- Encapsulated by hostA’s IPsec mechanisms
- Again encapsulated by gwA’s IPsec mechanisms
  - Above diagram shows headers, but as you go left, everything to the right would be enciphered and authenticated, *etc*.

AH Protocol

- Parameters in AH header
  - Length of header
  - SPI of SA applying protocol
  - Sequence number (anti-replay)
  - Integrity value check
- Two steps
  - Check that replay is not occurring
  - Check authentication data

Sender

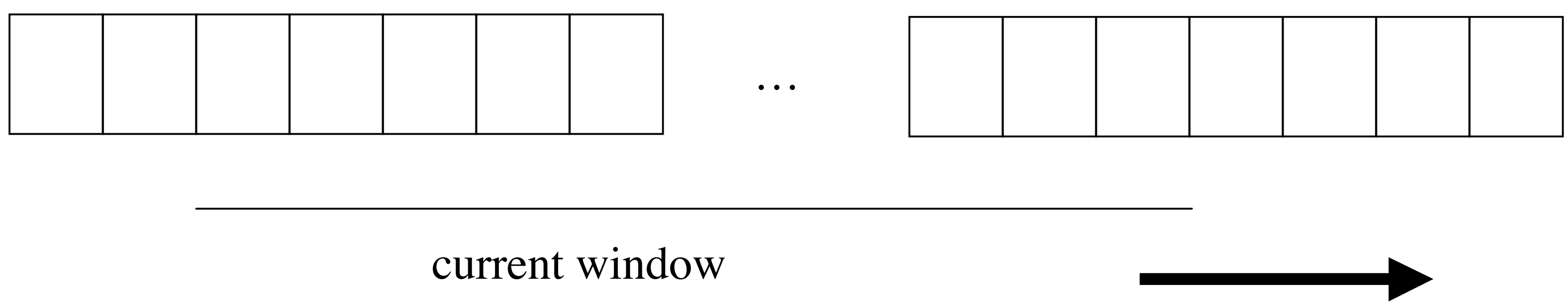
- Check sequence number will not cycle
- Increment sequence number
- Compute IVC of packet
  – Includes IP header, AH header, packet data
    - IP header: include all fields that will not change in transit; assume all others are 0
    - AH header: authentication data field set to 0 for this
    - Packet data includes encapsulated data, higher level protocol data

Recipient

- Assume AH header found
- Get SPI, destination address
- Find associated SA in SAD
  - If no associated SA, discard packet
- If antireplay not used
  - Verify IVC is correct
    - If not, discard

Recipient, Using Antireplay

- Check packet beyond low end of sliding window
- Check IVC of packet
- Check packet's slot not occupied
  - If any of these is false, discard packet



AH Miscellany

- All implementations must support:
  HMAC_MD5
  HMAC_SHA-1
- May support other algorithms

ESP Protocol

- Parameters in ESP header
  - SPI of SA applying protocol
  - Sequence number (anti-replay)
  - Generic “payload data” field
  - Padding and length of padding
    - Contents depends on ESP services enabled; may be an initialization vector for a chaining cipher, for example
    - Used also to pad packet to length required by cipher
  - Optional authentication data field

Sender

- Add ESP header
  - Includes whatever padding needed
- Encipher result
  - Do not encipher SPI, sequence numbers
- If authentication desired, compute as for AH protocol *except* over ESP header, payload and *not* encapsulating IP header

Recipient

- Assume ESP header found
- Get SPI, destination address
- Find associated SA in SAD
  - If no associated SA, discard packet
- If authentication used
  - Do IVC, antireplay verification as for AH
    - Only ESP, payload are considered; *not* IP header
    - Note authentication data inserted after encipherment, so no deciphering need be done

Recipient

- If confidentiality used
  – Decipher enciphered portion of ESP heaser
  – Process padding
  – Decipher payload
  – If SA is transport mode, IP header and payload treated as original IP packet
  – If SA is tunnel mode, payload is an encapsulated IP packet and so is treated as original IP packet

ESP Miscellany

- Must use at least one of confidentiality, authentication services
- Synchronization material must be in payload
  - Packets may not arrive in order, so if not, packets following a missing packet may not be decipherable
- Implementations of ESP assume classical cryptosystem
  - Implementations of public key systems usually far slower than implementations of classical systems
  - Not required

More ESP Miscellany

- All implementations must support (encipherment algorithms):
  - DES in CBC mode
  - NULL algorithm (identity; no encipherment)
- All implementations must support (integrity algorithms):
  - HMAC_MD5
  - HMAC_SHA-1
  - NULL algorithm (no MAC computed)
- Both cannot be NULL at the same time

Which to Use: PEM, SSL, IPsec

- What do the security services apply to?
  - If applicable to one application *and* application layer mechanisms available, use that
    - PEM for electronic mail
  - If more generic services needed, look to lower layers
    - SSL for transport layer, end-to-end mechanism
    - IPsec for network layer, either end-to-end or link mechanisms, for connectionless channels as well as connections
  - If endpoint is host, SSL and IPsec sufficient; if endpoint is user, application layer mechanism such as PEM needed

Key Points

- Key management critical to effective use of cryptosystems
  - Different levels of keys (session *vs*. interchange)
- Keys need infrastructure to identify holders, allow revoking
  - Key escrowing complicates infrastructure
- Digital signatures provide integrity of origin and content

  Much easier with public key cryptosystems than with classical cryptosystems