CH 7

Friday, January 4, 2019 9:22 PM

All secure protocols should do the following:

- Confidentially: Protect data from being read
- Integrity: Protect data from being modified
- Prevent attacker from impersonating server/client via server/client auth

Encryption: Data confidentiality
Signing: Data integrity/auth

Substitution ciphers	**Simplest form of encryption** ▪ Alg to encrypt a value based on a sub table that contains 1-to-1 mapping bet plaintxt/cipher txt value ▪ Cipher value is looked up in a table/original txt replaced ▪ Fails to withstand cryptanalysis **Frequency analysis:** Commonly used to crack substitution ciphers ▪ Correlates frequency of symbols found in cipher txt w/plaintxt data sets
XOR Encryption	**Simple:** Applies bitwise XOR op bet byte of plaintxt/byte of key: Results in cipher txt Example: Byte 0x48 & key byte 0x82 and result of XORing would be 0xCA ▫ **Symmetric**: Applying same key byte to cipher returns original plaintxt **Only way to securely use XOR encryption?** ▪ Message/values in key chosen completely at random ▪ **OTP: One-Time Pad encryption:** Hard to break ▪ Alg also has problems/rarely used in practice ▪ OTP: Size of key material you send must be same size as any msg to sender/recipient ▪ Only secure if every byte in msg encrypted w/completely random value ▪ Can never re-use a OTP: If attacker decrypts msg: Can recover key: Compromised
RND	**Random Number Generators:** ▪ Computers are deterministic: Getting truly random data difficult ▪ Sampling physical processes can generate relatively unpredictable data ▫ Don't provide much data: Few hundred bytes every second at best ▫ 4096bit-RSA key requires at least 2 random 256-byte numbers: 7 sec to generate **PRNG: Pseudorandom Number Generators:** ▪ Use initial seed value/generate seq of num that shouldn't be predictable ▫ w/out knowledge of internal state of generator ▫ C lib rand(): Completely useless for crypto secure protocols

Symmetric Key: Send completely random key that's same size as msg before encryption can take place as OTP

- Can construct symmetric key alg that uses math constructs to make cipher
- Easier to distribute if alg has no obv weakness: Limiting factor for sec key size

2 types of symmetric ciphers

1. **Block**
2. **Stream**

Block Ciphers	**AES: Advanced Encryption Standard, DES: Data Encryption Standard** ▪ Encrypt/decrypt fixed number of bits (block) every time encryption alg applied ▪ To encrypt/decrypt msg: Alg reqs key ▪ If msg longer than size of block: Must be split into smaller blocks/alg applied to each ▪ Each app of alg uses same key
DES	**Data Encryption Standard:** ▪ Oldest block cipher still used in modern apps: Dev by IBM: Published as FIPS: 1979 ▪ FIPS: Federal Info Processing Standard **Fiestel network**: Repeatedly applies function to input for number of rounds ▪ Takes input value from previous round [original plaintxt] ▪ Specific subkey derived from original key using key-scheduling alg

DES: 64-bit block size/64-bit key
- 8 bits of key used for error checking [so 56 bits]
- Unsuitable for modern apps: 1998: EFF's DES cracker
  - HW-key brute-force attacker that discovered unknown DES key in 56 hrs

3DES **Modified form that applies alg 3 times:**
- Uses 3 separate DES keys: 168 bits
- Encrypt function applied 1st time
- Output decrypted using 2nd key
- Output encrypted again using 3rd key
  - Ops reversed perform decryption

AES **Advanced Encryption Standard:**
- Rijndael alg: Fixed 128 bit block size
- Can use 3 diff key lengths: 128/192/256 bits

Substitution-permutation network: 2 main components
1. **S-Box: Substitution Boxes**
2. **P-Box: Permutation Boxes**

2 components chained together to form round of alg:
- As with Fiestel network: Can be applied many times w/diff values of S/P-Box

S-box: Basic mapping table unlike sub cipher
- Takes input/looks up in table/produces output

Other Block Ciphers

Name	Block size (bits)	Key size (bits)	Year introduced
DES	64	56	1979
Blowfish	64	32-448	1993
3DES	64	56, 112, 168	1998
Serpent	128	128, 192, 256	1998
Twofish	128	128, 192, 256	1998
Camellia	128	128, 192, 256	2000
AES	128	128, 192, 256	2001

Block Cipher Modes: Defines how cipher ops on blocks of data
Mode of operation: Cipher combined w/this alg: Provides addl sec properties: Less predictability

ECB **Electronic Code Book:**
- Simplest/default mod of op: Alg applied to each fixed-size block from plaintxt to generate cipher blocks
- Size of block defined by alg in use

CBC **Cipher Block Chaining:**
- Encryption of single plaintxt block depends on encrypted value of previous block
- Previous encrypted block is XORed w/current plaintxt block: Alg applied to combined result
- 1st block of plaintxt no previous cipher block: Combined w/IV: Initialization Vector

Common Block Cipher Modes of Operation:

Name	Abbreviation	Mode Type
Electronic Code Block	ECB	Block
Cipher Block Chaining	CBC	Block
Output Feedback	OFB	Stream
Cipher Feedback	CFB	Stream
Counter	CTR	Stream
Galois Counter Mode	GCM	Stream w/data integrity

Block Cipher Padding: Op on fixed-size msg unit: Block
- Padding schemes determine how to handle unused remainder of a block during encryption/decryption

PKCS #7: Public Key Crypto Standard #7: All padded bytes set to value that represents how many padded bytes

present

- Each byte set to value 3

Padding Oracle Attack	Occurs when CBC mode of op combined w/PKCS #7 padding scheme ▪ Allows attacker to decrypt data: Some cases encrypt own data (session token) via protocol ▪ If attacker can decrypt a session token: Might recover sensitive info If they can encrypt the token: ▪ Might be able to circumvent access controls on a website for ex

Common Stream Ciphers

Cipher Name	Key size (bits)	Year Introduced
A5/1 and A5/2 (GSM voice)	54/64	1989
RC4	Up to 2048	1993
CTF: Counter Mode	Depends on block cipher	NA
OFB: Output Feedback Mode	Depends on block cipher	NA
CFB: Cipher Feedback Mode	Depends on block cipher	NA

Signature algs: Generate a unique signature for a msg: Msg recipient can use same alg to generate sig to prove auth

- Protects against tampering over untrusted network
- Built on crypto hashing algs

Cryptographic Hashing Algs: AKA Message Digest Algs

- Funcs applied to a msg to generated fixed-length summary of msg: Usually shorter than original

For hashing alg to be suitable needs 3 reqs:

Pre-Image resistance	Given a hash value: Diff to recover msg
Collision resistance	Diff to find 2 diff msgs that has to same value
Nonlinearity	Diff to create a msg that hashes to any given value

Most common:

- **MD: Message Digest:** MD4/5: Ron Rivest
- **SHA: Secure Hashing Alg:** SHA-1/2: NIST
- **CRC:** Useful for detecting changes in data: Not useful for secure protocols: Can change checksum

Asymmetric Signature Algorithms: Properties of asymmetric crypto to generate msg signature

- **DSA: Digital Signature Alg:** Designed for sigs only
- **RSA:** Possible to encrypt msg using private key/decrypt w/public one (no longer secure)

Message Authentication Codes: MACs: Symmetric sig algs: Rely on sharing key bet sender/recipient

HMAC: Hashed Message Auth Code: Counters attacks

- Instead of directly appending key to msg/using hashed output to produce sig: Splits process into 2 parts
  1. **Key is XORed w/padding block equal to block size of hashing alg**
     - 1st padding block filled w/repeating value: Typically byte 0x36
     - Prefixed to msg/hashing alg applied
  2. **Takes hash value from 1st step: Prefixes hash w/new key (outer padding block: Uses constant 0x5C)**
     - Applies hash alg again

PKI: Public Key Infrastructure:

- Combined set of protocols/encryption key fmts/usr roles/policies to manage asymmetric public key info across network

WOT	**Web of Trust:** Used by PGP: ID of public key attested to by someone you trust

X.509 Certificates: Generate strict hierarchy of trust rather than relying on directly trusting peers

Used to: Verify web servers | Sign exe programs | Auth to a network service

- Trust provided through hierarchy of certs using asymmetric sig algs like RSA/DSA

Chain of trust: Certs must contain at least 4 pieces of info:

1. **Subject: Specifies ID for cert**
2. **Subject's public key**
3. **Issuer: ID's signing cert**
4. **Valid sig applied over cert/auth by issuer's priv key**

TLS: Transport Layer Security: Formerly SSL: Secure Sockets Layer:

- Most common sec protocol in use on the internet
- Originally dev as SSL by Netscape in mid-90's for securing HTTP connections

Protocol went through multiple revisions: SSL ver 1-3.0 || TLS 1.0-1.2

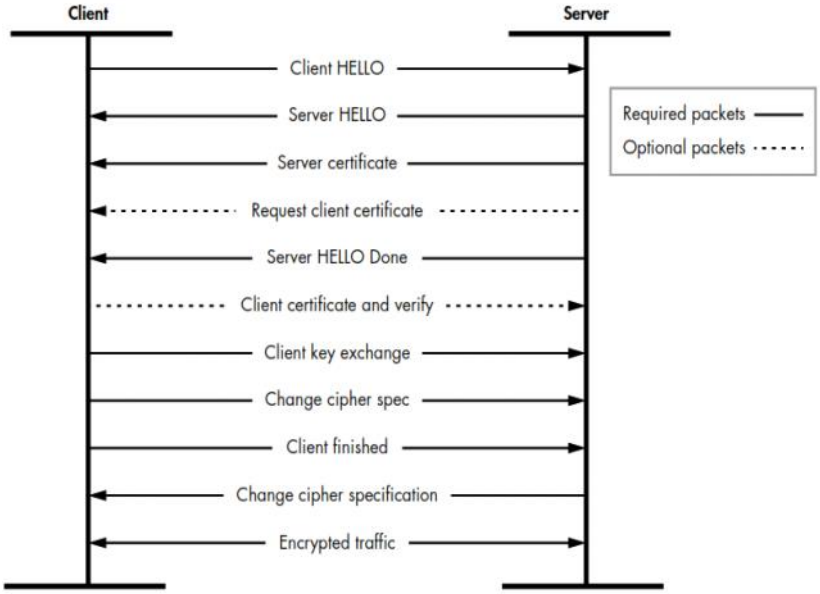
- Originally designed for HTTP: Can use TLS for any TCP protocol

DTLS: Datagram Transport Layer Security: For use w/UDP/unreliable protocols

TLS: Symmetric/asymmetric encryption/MACs/Secure key exchange/PKI

TLS Handshake: Client/server negotiate type of encryption they'll use: Exchange unique key for connection/verify ID's

- TLS Record protocol: All comm uses a predefined TLV structure
  - Allows protocol parser to extract individual records from stream of bytes
  - Handshake packets assigned a tag value of 22 to distinguish from other packets
- Handshake process can be time-intensive
- Sometimes truncated/bypassed entirely by caching previously negotiated session key
  - Or by client's asking server to resume previous session by unique session identifier
  - Client still won't know private negotiated session key sec wise



Initial Negotiation:

- **1st step:** Client/server negotiate sec params they want to use for TLS connection using **HELLO** msg
  - Piece of info in **HELLO** client random
  - **Client random:** Random value ensures connection can't be easily replayed
  - **HELLO** msg indicates types of ciphers supported
- **TLS designed to be flexible w/regard to what encryption algs it uses:**
  - Only supports symmetric ciphers like RC4/AES
  - Using public key encryption would be too expensive from computational perspective
- **Server responds w/its own HELLO msg:** Indicates what cipher chosen from avail list
- **Server HELLO also contains server random**
  - **Server random:** Value that adds addl replay protection to connection
- **Server sends its X.509 cert/any necessary intermediate CA certs so client can make informed decision about ID**
- **Server sends HELLO done packet to inform client it can proceed to auth connection**

Endpoint Authentication: Client must verify server certs legitimate/meet client's sec reqs

- **Client must verify ID in cert by matching Subject field to server's domain name**
  - Cert's Subject/Issuer fields not simple strings but X.500 names
  - Contains other fields like Organization/Email
  - Only CN ever checked during handshake to verify ID
  - Possible to have wildcards in CN field: Sharing certs w/multiple servers running on a subdomain
    - *.domain for www.domain.com or blog.domain.com
- **After client checks ID of endpoint**: Ensures cert trusted
  - Builds chain of trust for cert/any intermediate CA certs
  - Checks to make sure none of them appear on revocation lists

- **Optional: Client certificate:** Allows server to auth client
  - If server reqs client cert: Sends list of acceptable root certs to client during HELLO
  - Client can search avail certs/choose most useful to send back to server
  - Cert + verification msg containing hash of all handshake msgs sent/received/signed w/cert's priv key
  - Sig proves to server client possesses priv key associated w/cert

Establishing Encryption: When endpoint auth: Client/server finally establish encrypted connection

- Client sends randomly generated pre-master secret to server encrypted w/server's cert public key
- Both client/server combine pre-master secret w/client/server randoms
- They use this combined value to seed a RNG that generates a 48-byte master secrete
  - Will be session key for encrypted connection
- When both endpoints have master secret:
  - Client issues change cipher spec packet: Tells server it will only send encrypted msgs from there on
  - Client needs 1 msg to server before normal traffic: Finished packet
    - Packet encrypted w/session key/contains hash of all handshake msgs sent/received during process
    - Crucial against downgrade attack

Downgrade attack: Attacker mods handshake process to reduce sec of connection by selecting weak encryption algs

- Once server receives finished msg: Can validate negotiated session key correct: If not: Closes connection

How TLS Meets Sec Reqs

Sec Req	How met
Confidentiality	Selectable strong cipher suites: Sec key exchange
Integrity	Encrypted data protected by an HMAC: Handshake packets verified by final hash verification
Server Auth	Client can choose to verify server endpoint using PKI + issued cert
Client Auth	Optional cert-based client auth

Problems:

- Reliance on cert-based PKI
- Depends entirely on trust that certs issued to correct people/orgs
- Subversion of CA process to generate certs an issue
- CA's don't always perform due diligence/issued bad certs

Certificate pinning: App restricts acceptable certs/CA issuers for certain domains

- Issue: Management of pinning list: Dev can't easily migrate/change certs to another CA
- TLS connections can be captured from network/stored by attacker until needed
- If attacker obtains server's priv key: All historical traffic could be decrypted

Why using DH alg in addition to certs for ID verification impt

Perfect Forward Secrecy: Even if private key compromised: Not easy to also calc DH-generated key