Communication Protocols

Lecture 19

TLS

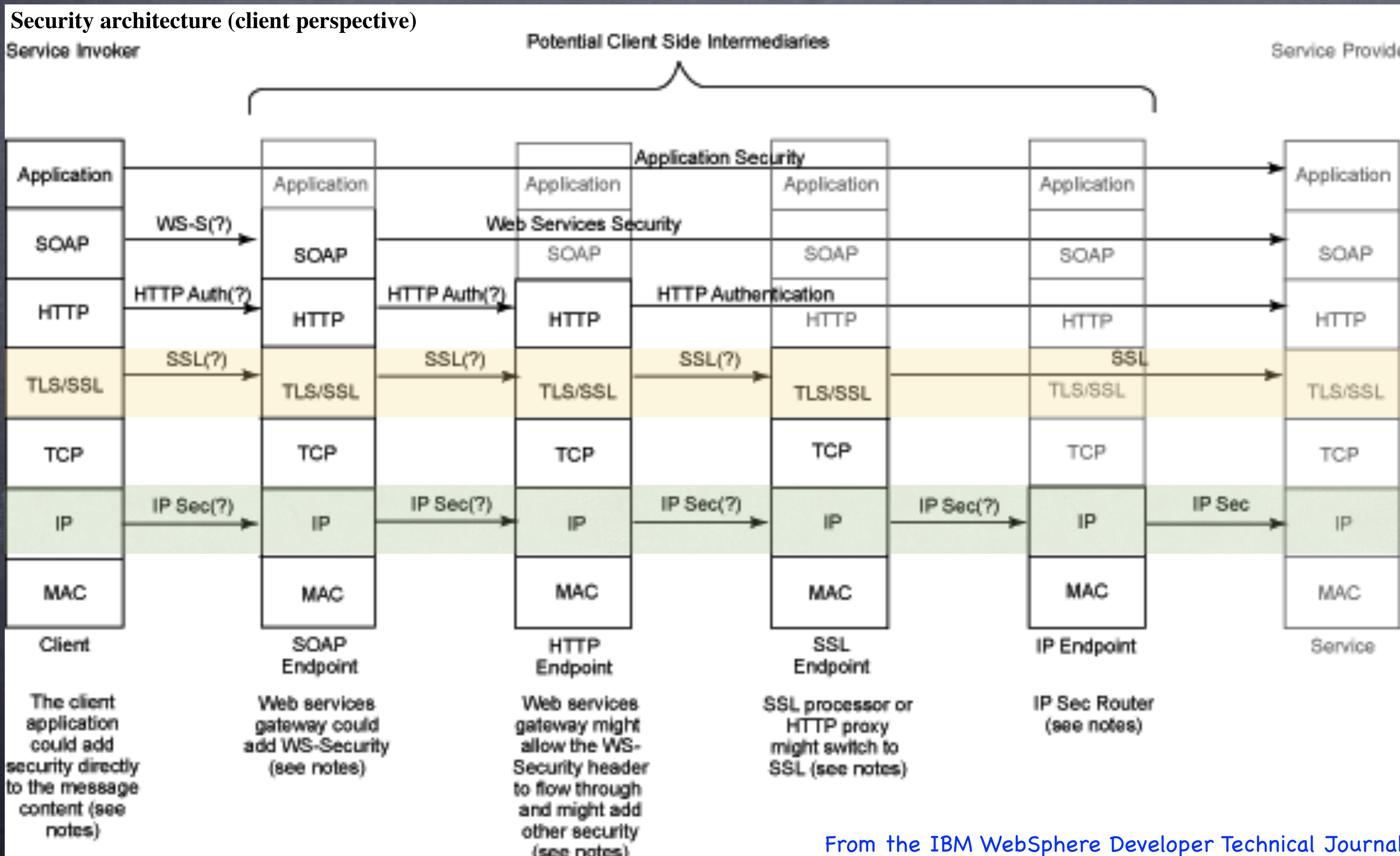
We saw...

- Symmetric-Key Components
  - SKE, MAC
- Public-Key Components
  - PKE, Digital Signatures
- Building blocks: Block-ciphers (AES), Hash-functions (SHA-3), Trapdoor PRG/OWP for PKE (e.g., DDH, RSA) and Random Oracle heuristics (in RSA-OAEP, RSA-PSS)
- Symmetric-Key primitives much faster than Public-Key ones
  - Hybrid Encryption gets best of both worlds

Secure Communication in Practice

- Can do at different levels of the "network stack"
  - e.g., "application layer", "transportation layer" or "network layer"
- Protocol standards in all cases
  - To be interoperable
  - To not insert bugs by doing crypto engineering oneself
  - e.g.: SSL/TLS (used in https), IPSec (in the "network layer")
  - Allows implementation in libraries or within OS kernels

Security Architectures
(An example)



Secure Communication Infrastructure

- Goal: a way for Alice and Bob to setup a private and authenticated communication channel (can give a detailed SIM-definition)
- Simplest idea: Use a (SIM-CCA secure) public-key encryption (possibly a hybrid encryption) to send signed (using an existentially unforgeable signature scheme) messages (with sequence numbers and channel id)
  - Limitation: Alice, Bob need to know each other's public-keys
  - Also, room for efficiency improvements if Alice and Bob engage in "sessions"
    - Can maintain state (keys, counters) throughout the session
    - If fresh PKE key in each authenticated session, only CPA security needed

Secure Communication Infrastructure

- Secure Communication Sessions
  - Handshake protocol: establish private shared keys
  - Record protocol: use efficient symmetric-key schemes
- Server-to-server communication: Both parties have (certified) public-keys
- Client-server communication: server has (certified) public-keys
  - Client "knows" server. Server willing to talk to all clients
- Client-Client communication (e.g., email) Clients share public-keys in ad hoc ways

(Authenticated) Key-Exchange

Server may "know" (some) clients too, using passwords, pre-shared keys, or if they have (certified) public-keys. Often implemented in application-layer

Certificate Authorities

- How does a client know a server's public-key?
  - Based on what is received during a first session? (e.g., first ssh connection to a server)
- Better idea: Chain of trust
  - Client knows a certifying authority's public key (for signature)

Certificate Authorities

- How does a client know a server's public-key?
  - Based on what is received during a first session? (e.g., first ssh connection to a server)
- Better idea: Chain of trust
  - Client knows a certifying authority's public key (for signature)
  - Bundled with the software/hardware
- Certifying Authority signs the signature PK of the server
  - CA is assumed to have verified that the PK was generated by the "correct" server before signing
  - Validation standards: Domain/Extended validation

Forward Secrecy

- Servers have long term public keys that are certified
  - Would be enough to have long term signature keys, but in practice sometimes long term decryption keys too
  - Problem: if the long term decryption key is leaked, old communications are also revealed
    - Adversary may have already stored, or even actively participated in old sessions
  - Solution: Do a fresh secure key-exchange for each session (authenticated using signatures)
    - TLS 1.3 removes support for static keys (except for externally prepared Pre-Shared Keys)

Authenticated Encryption

- Doing encryption + authentication efficiently
  - Generic composition (encrypt, then MAC) needs two keys and two passes

MAC-then-encrypt is not necessarily CCA-secure

- Authenticated Encryption (AE) aims to do this more efficiently (one single module, which can be optimised together)
  - Several constructions based on block-ciphers (modes of operation) provably secure modeling block-cipher as PRP
    - One pass: IAPM, OCB, ... [patented]
    - Two pass: CCM, GCM, AES-SIV, ... [in NIST standards]
  - AE with Associated Data (AEAD): Allows unencrypted (but authenticated) parts of the plaintext, for headers etc.
    - Used as the basic symmetric key primitive in TLS 1.3

Authenticated Encryption

GCM

- Galois/Counter Mode: Encrypt using a block-cipher in counter mode, and authenticate the ciphertext using a MAC based on operations in a "Galois field"
  - GHASH: uses arithmetic in a finite field where field elements are 128-bit blocks, addition is bit-wise XOR, and multiplication is quite fast (compared to block-ciphers)
    - Treat an (arbitrarily long) message m as the coefficients of a polynomial M, and evaluate M(K), where K, the key, is a random field element
    - An approximate universal hash function: Given M(K), M'(K) is still almost uniform (degrades with message length)
  - GCM: (r,C,T) where $C = F_K(r+1) \oplus m$, $T = F_K(r) \oplus GHASH_{K'}(C)$

Authenticated Encryption

GCM-SIV

- Synthetic IV: To provide security against "nonce reuse"
  - Recall SKE $(r, F_K(r)\oplus m)$ where F is a PRF (with extended output). The "IV" r should be "fresh".
  - Instead of picking r = nonce, let $r = F_K(\text{nonce}\oplus H_{K'}(m))$, where H is (say) GHASH
    - If nonce is fixed deterministically, a deterministic scheme, and hence not CPA secure. But secure upto revealing message repetition pattern.
    - Letting ciphertext be $(\text{nonce}, r, F_K(r)\oplus m)$ works as an authenticated encryption
      - (nonce, r) is a MAC tag on m which hides m

A Simple Secure Communication Scheme

- Handshake
  - Client sends fresh session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)
- For authentication only: use MAC
  - In fact, a "stream-MAC": To send more than one message, but without allowing reordering
- For authentication + encryption, encrypt-then-MAC ("stream" versions)
  - Or better, use Authenticated Encryption

Server's PK either trusted (from a previous session for e.g) or certified by a trusted CA, using a Digital Signature scheme

Does not have forward secrecy!
Not allowed in TLS 1.3

A Simple Secure Communication Scheme

- Handshake
  - Client sends first message of a key exchange protocol and server responds with the second message. Symmetric keys derived from the resulting secret.
- For authentication only: use MAC
  - In fact, a "stream-MAC": To send more than one message, but without allowing reordering
- For authentication + encryption, encrypt-then-MAC ("stream" versions)
  - Or better, use Authenticated Encryption

Server's message is authenticated, and can include additional data, encrypted using the newly defined key. Also, includes a certificate of its signature key.

Need to avoid replay attacks (infeasible for server to explicitly check for replayed ciphertexts)

Recall "inefficient" domain-extension of MAC: Add a sequence number (and a session-specific nonce) to each message before MAC'ing

MAC serves dual purposes of CCA security and authentication

TLS (SSL)

- Handshake
  - Client sends first message of a key exchange protocol and server responds with the second message. Symmetric keys derived from the resulting secret.
- For authentication only: use MAC
  - In fact, a "stream-MAC": To send more than one message, but without allowing reordering
- For authentication + encryption, encrypt-then-MAC ("stream" versions)
  - Or better, use Authenticated Encryption

Negotiations on protocol version, "cipher suites" for SKE (block-ciphers & hash), PKE & signature algorithms.

e.g. cipher-suite: RSA-OAEP for key-exchange, AES for SKE, HMAC-SHA256 for MAC (In TLS 1.3, Auth. Enc.)

TLS 1.2 allows server to send a certified PKE public-key (RSA), which the client uses to send a "pre-key" K. Server also "contributes" to key-generation (to avoid replay attack issues): a master key generated as $PRF_K(x,y)$ where x from client and y from server. SKE and MAC keys derived from master key

(TLS 1.3 allows only Diffie-Hellman key-exchange followed by HKDF)

TLS 1.2 uses MAC-then-encrypt! Not CCA secure in general, but secure with stream-cipher (and CBC mode). TLS 1.3 uses AEAD.

Several details on closing sessions, session caching, resuming sessions, using pre-shared keys ...

TLS: Some Considerations

- Overall security goal: Authenticated and Confidential Channel Establishment (ACCE), or Server-only ACCE
- Handshake Protocol
  - Cipher suites are negotiated, not fixed → "Downgrade attacks"
  - Doesn't use CCA secure PKE, but is overall CCA secure if error in decryption "never revealed" (tricky to ensure!)
- Record Protocol
  - Using MAC-then-Encrypt (as in TLS 1.2) is tricky:
    - CCA-secure when using SKE implemented using a stream cipher (or block-cipher in CTR mode) or CBC-MAC
    - But insecure if more information revealed on decryption fails
      - e.g., different times taken by MAC check (or different error messages!) when a format error in decrypted message
- TLS 1.3 uses easier to analyse protocols

TLS: Some Considerations

- Numerous vulnerabilities keep surfacing

  FREAK, DROWN, POODLE, Heartbleed, Logjam, ...

  And numerous unnamed ones: www.openssl.org/news/vulnerabilities.html

  Listed as part of Common Vulnerabilities and Exposures (CVE) list: cve.mitre.org/
- Bugs in protocols
  - Often in complex mechanisms created for efficiency
  - Often facilitated by the existence of weakened "export grade" encryption and improved computational resources
  - Also because of weaker legacy encryption schemes (e.g. Encryption from RSA PKCS#1 v1.5 — known to be not CCA secure and replaced in 1998 — is still used in TLS)
- Bugs in implementations
- Side-channels that are not originally considered
- Back-Doors (?) in the primitives used in the standards

TLS: Some Considerations

- Numerous vulnerabilities keep surfacing

  FREAK, DROWN, POODLE, Heartbleed, Logjam, …

  And numerous unnamed ones: www.openssl.org/news/vulnerabilities.html

  Listed as part of Common Vulnerabilities and Exposures (CVE) list: cve.mitre.org/

(Kenny Paterson & Thyla van der Merwe, Dec 2016)

  Encryption from RSA PKCS#1 v1.5 — known to be not CCA secure and replaced in 1998 — is still used in TLS)

- Bugs in implementations
- Side-channels that are not originally considered
- Back-Doors (?) in the primitives used in the standards