Communication Protocols

Lecture 19

TLS

We saw...

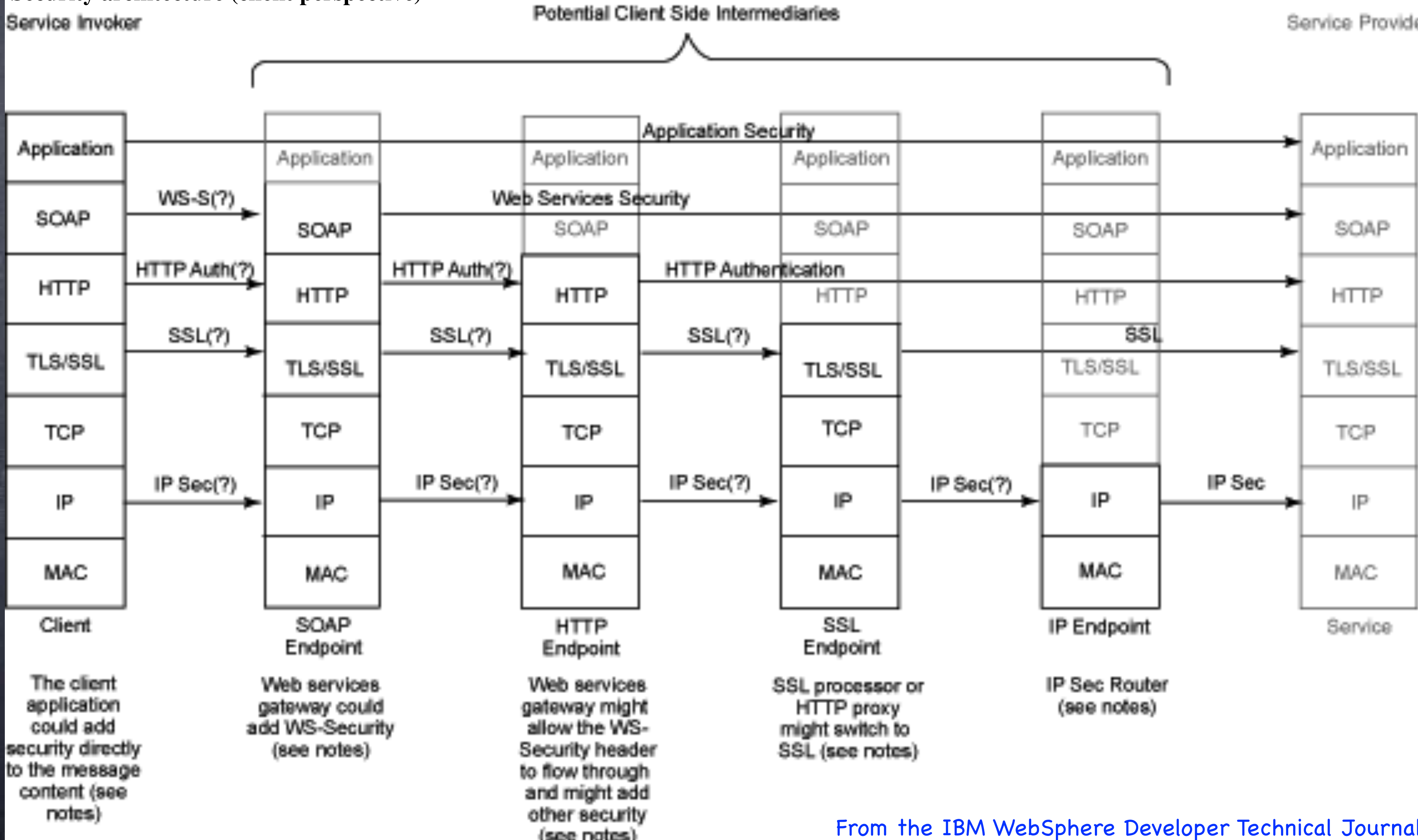
- Symmetric-Key Components
  - SKE, MAC
- Public-Key Components
  - PKE, Digital Signatures
- Building blocks: Block-ciphers (AES), Hash-functions (SHA-3), Trapdoor PRG/OWP for PKE (e.g., DDH, RSA) and Random Oracle heuristics (in RSA-OAEP, RSA-PSS)
- Symmetric-Key primitives much faster than Public-Key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

Security architecture (client perspective)



From the IBM WebSphere Developer Technical Journal

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, typically Alice and Bob engage in "transactions," exchanging multiple messages, maintaining state throughout the transaction
    - Makes several efficiency improvements possible

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 long term encryption keys too
  - Problem: if the long term key is leaked, old communications are also revealed
    - Adversary may have already stored, or even actively participated in old sessions
  - Solution: Do a fresh 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 better
  - Generic composition (encrypt, then MAC) needs two keys and two passes

MAC-then-encrypt is not necessarily CCA-secure

- 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, SIV, ... [included 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

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-cipher, and "stream-MAC"

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-cipher, and "stream-MAC"

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-cipher, and "stream-MAC"

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.)

Server sends a certificate of its PKE public-key, which the client verifies

Server also "contributes" to key-generation (to avoid replay attack issues): Roughly, client sends a key K for a PRF; 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 uses HKDF instead)

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 is tricky:
    - CCA-secure when using SKE implemented using a stream cipher (or block-cipher in CTR mode) or CBC-MAC
    - But insecure if it reveals information from decryption phase.
      - 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)

  - 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