Network Security fiche

Pierre Colson

January 2023

Contents

Markdown version on *github*
Compiled using *pandoc* and *`gpdf` script*

TLS

General

- **TLS**: **T**ransport **L**ayer **S**ecurity
- It's goal is to provide a **secure channel between two peers**
- **Entity authentication**
  - **Server** side of the channel is *always authenticated*
  - **Client** side is *optionally authenticated*
  - Via **Assymetric crypto** or a symmetric *pre-shared key*
- **Confidentiality**
  - *Data* send over the channel is *only visible to the endpoints*
  - TLS does *not hide the length* of the data it transmits (but allows padding)
- **Integrity**
  - *Data* sent over the channel *cannot be modified* without detection
  - Integrity guarantees also cover reordering, insertion, deletion of data
- **Efficiency**
  - Attempt to minimise crypto overhead
  - Minimal use of public key techniques; maximal use of symmetric key techniques
  - Minimise number of communication round trips before secure channel can be used
- **Flexibility**
  - Protocol supports flexible choices of algorithms and authentication
- **Self negociation**
  - The choice is done in hand, i.e. as part of the protocol itself

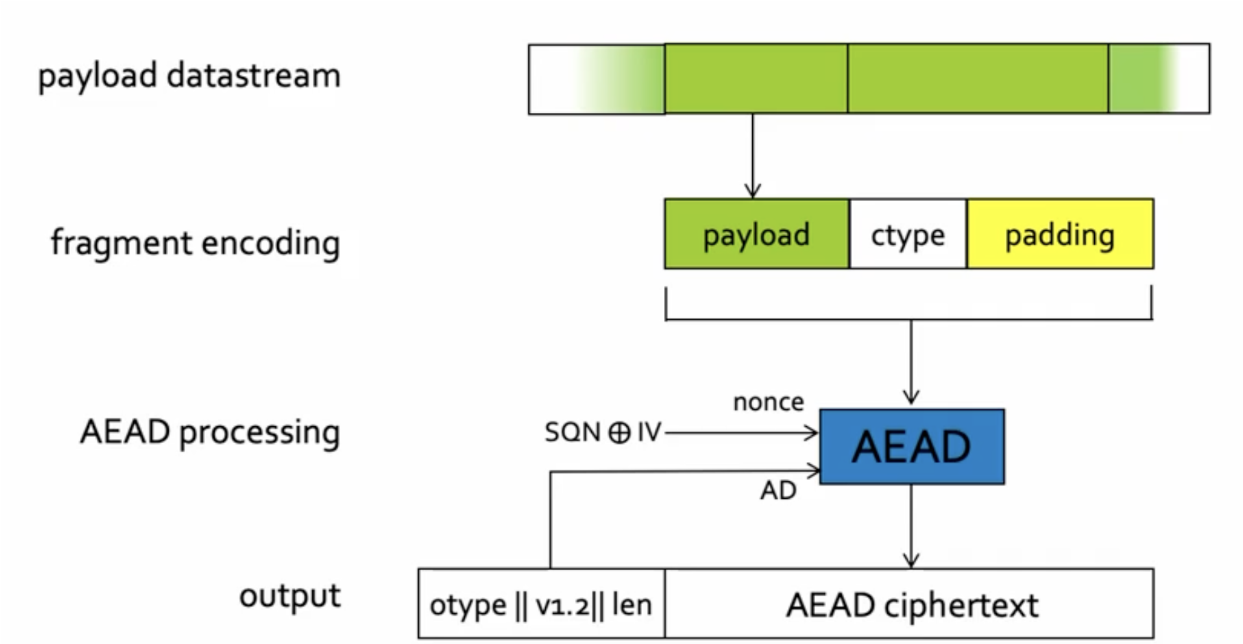
  - The is done through the version negociation and cipher suite negociation process: the client offers, server selects
- **Protection of negocation**
  - Aim to prevent MITM attacker from performing version and cipher suite downgrade attacks

- So the cryptography used in the protocol should also protect the hsoice of cryptography made
- **TLS** aims for security in the face of *attacker who has complete control of the network*
- Only requirement from underlying transport: reliable, in order data-stream
- **Handshake protocol**: Authentication, negociation and key agreement
- **Record protocol**: Use those keys to provide confidentiality and integrity
- **TLS 1.3** design process goals
  - *Clean up*: get rid ot flawed and unused crypto & features
  - *Improve latency* : for main handshake and repeated connections (while maintaining security)
  - *Improve privacy*: hide as much of the handshake as possible
  - *Continuity*: maintain interoperability with previous versions and support exisiting important use cases
  - *Security Assurance (added later)*: have supporting analyses for changes
- TLS uses mostrly 'boring' cryptography yet is a very complex protocol suite
- Some protocol design errors were made, but not too many
- Legacy support for EXPORT cipher suites and liong tial of old versions opened up seious vulnerabilities
- Lack of formal state-machine description, lack of API specification, and sheer complexity of specifications have let to many serious implementations errors
- Poor algorithm choices in the Record Protocol should have been retired more aggressively
- Most of this had been fixed in TLS 1.3
- TLS 1.3 was developed hand-in-hand with formal security analysis
- The design changed many times, often changes driven by security concerns identified through the analysis
- Cryptography has evolved significantly in TLS
- The largest shift was from RSA key transport to elliptic curve Diffie-Hellman, and from CBC/RC4 to AES-GCM
- A second shift now underway is to move to using newer elliptic curves, allowing greater and better implementation security
- A third shift is the move away from SHA1 in certs
- A future shift is being considered to incorporate post-quantum algorithm
- But Implementation vulnerabilities are bound to continue to be discovered

Record Protocol

- The TLS Record Protocol provides a **stream oriented** API for applications making use of it
  - Hence TLS may fragment into smaller units or coalesce into larger units any data supplied by the calling application
  - Protocol data units in TLS are called **records**
  - So each record is a fragment from a **data stream**
- Cryptographic protectionin the TLS Record Protocol
  - Data origin authentication & integrity for records using a MAC
  - Confidentiality for records using a symmetric encryption algorithm
  - Prevention of replay, reordering, deletion of records using per record sequence number protected by the MAC
  - Encryption and MAC provided simultaneously by use of AEAD in TLS 1.3
  - Prevention of reflection attack by key separation
- *Datastream* is divided in different **payload**
- Each *payload* in concanated with a bit (**ctype**) and an optional **padding**; this give a **fragment**
- This is then given to **AEAD** encryption
  - Needs in input a *nonce*, some *associated data* (AD) (otype, v1.2, and len field) and a plaintext
- **ctype field**
  - Single byte representing content type - indicates wheter contetn is handshake message, alert message or application data
  - AEAD-encryption inside record; header contains dummy value otype to limit traffic analysis
- **padding**
  - Optional features that can be used ot hide true length of fragments

  - Not needed for encryption
  - Sequence of 0x00 bytes afer non-0x00 content type field
  - Removed after integrity check, so no padding oracle issues arise (Time side channel attack to recover lenght on plaintext)
- **AEAD nonce**
  - $Nonce = SQN \bigoplus IV$
  - Constructed from 64 bits sequence number (SQN)
  - SQN is incremented for each record sent on a connection
  - SQN is masked by XOR with IV field
  - IV is a fixed (per TLS connection) pseudorandom value deirved from secrets in TLS handshake protocol
  - IV masking ensures nonce sequence is 'unique' per connection, good for security in multi-connection setting
- **Record header**
  - Contains dummy field, legacy version field, length of AEAD ciphertext
  - Version field is always securely negociated during handshake
  - SQN is not included in header, but is maintained as a counter at each end of the connection (send and receive)



Handshake Protocol

- TLS 1.3: **full handshake in 1 RTT**
  - Achieved via feature reduction: we always do (EC)DHE in one of a shortlist of groups
  - Client includes DH shares in its first message, along with `Clienthello`, anticipating groups that server will accept
  - Server respons with single DH share in its `ServerKeyShare` response
  - If this works, a forward-secure key is established after 1 round trip
  - If server dos not like DH groups offered by client, it sends a `HelloRetryRequest` and a group description back to client
    * In this case, the handshake will be 2 round trips
- **0-RTT handshake** when resuming a previously established connection
  - Client + server keep shared state enabling them to derive a PSK (pre-shared key)
  - Client derives an 'early data' emcryption key from the PSK and can use it to include encrypted

  application data along with its first handshake message
  - *sacrifices* certain securitty properties
- Because of reliance oc Ephemeral DS key exchange, TLS 1.3 handshake is **forward secure**
- This means: compromise of all session keys, DH values and signing keys has no impact on the security of earlier sessions
- Use of ephemeral DH also means: if a server's long term (signing) key is compromised, then an attacker cannot passively decrypt future sessions
- Compare to RSA key transport option in TLS 1.2 and earlier: past and future passive interception using compromised server RSA private key

Public Key Infrastructure (PKI)

- In symmetric cryptography, main challenge is key distribution as keys need to be distributed via **confidential and authentic** channels
- In public-key system, main challenge is key authentication (i.e., which key belongs to who) as keys need to be distributed via **authentic channel**
- **Public-key infrastructure (PKIs)** provide a way to validate public keys
- **CA**: certificate Authority
- A **public-key certificate** (or simply **certificate**) is signed and dinds a name to a public key
- **Trust anchor, trust root**: self-signed certificates of public keys that are allowed to sign other certificate
- **X.509** strandard format of digital certificate
- **Root of trust** is used to establish trust in other entities
- *Cryptography operations enable transfer of trust from one entity to another*
- Trust roots do not scale to the world
  - *Monopoly model*: single root of trust
    - Problem: world cannot agree on who controls root of trust
  - Obligarchy model: numerous roots of trust
    - Problems: Weakest link security: single compromised enables man-in-the-niddle attack; not trusting some trust roots results in unverifiable entities
- **Let's Encrypt**
  - Goal: provide free certificate based on automated domain validation, issurance, and renewal
  - Based on ACME; Automated Certificate Managment Environment
- **Certificate Revocation**
  - Certificate revocation is a mechanism to invalidate certificates
    - After a private key is disclosed
    - Trusted employee / administrator leaves corporation
    - Certificate expiration time is usually chosen too long
  - CA periodically publishes Certificate Revocation List (CRL)
    - Delta CRLs only contains changes
    - What to do if we miss CRL update?
  - What is general problem with revocation
    - CAP theorem (Consistency, Availability, tolerance to partition): impossible to achieve all 3, must select one to sacrifice
- **DANE**
  - DNS-Based Authentication of Named Entities
  - Goal: Authenticate TLS servers without a certificate authority
  - Idea: use DNSSEC to bind certificate to names
- **Certificate Transparency**
  - Will make all public end-entity TLS certificate public knowledge, and will hold CAs publicaly accountable for all certificates they issue
  - And it will do so withou introducing another trusted third party
  - A CT log is an append-only list of certificate
  - The log server verifies the certificate chain
  - Periodically append all new certificates to the append-only log and sign that list

  - Publish all updates of the signed list of certificates to the world
  - A CT log is not a "Super CA"
    * The log does not testify the goodness of certificates; it merely notes their presence
    * The log is public: everyone can inspect all the certificates
    * The log is untrusted: since the log is signed, the face that everyone sees the same list of certificate is cryptographically verifiable
  - How CT improves security
    * Browser would require SCT for opening connection
    * Browser contacts log server to ensure that certificate is listed in the log
  - Consequence
    * Attack certificate would have to be listed in public log
    * Attacks become publicly known
  - Advantages
    * CT is fully operational today
    * No change to domain's web server required
  - Disadvantages
    * MitM attacks can still proceed
    * Browser still needs to contact Log eventually to verify that certificate is listed in log
    * Current CT does not support revocation
    * Malicious Log server can add bogous certificate
    * Management of list of trusted log server can introduce a kill switch
- **Summary**
  - Cannot tolerate additional latency of contacting additional server during SSL/TLS handshake
  - A key has to be immediately usable and verifiable after initial registration
  - Users shouldn't be bothered in the decision process if certificate is legitimate
  - Need to cover entire certificate life cycle, including revocation, handing stolen and lost certificate
  - Secure crypto and secure protocols are insufficient
    * Numerous failure possibilities
    * User interface security and certificate management are critically important
  - The entity who controls the root keys, controls all authentication and verification operations
  - PKI and revocation can result in a powerfull 'kill switch', which can enable shouting down part of internet
    * Sovereign PKI continues to be an important research challenge