We’re going to talk about:

1. What is SSL
2. What’s new in TLS 1.3
3. RIOT-OS wolfSSL pkg design and implementation
What is SSL
And why it is important for the IoT
What is SSL?

- Enables **security** in network communications, defined as:
  
  - **Confidentiality** + Prevent eavesdropping
  - **Authentication** + Prevent impersonation
  - **Integrity** + Prevent modification
What is SSL?

• Provides **end-to-end** security
  - Using the same standard **protocols** and **ciphers** as the remote endpoint
  - Enabling built-in security for the most popular communication protocols (**https**, **ssh**), even IoT specific (**mqtt**)
  - Not relying on security features from third party technology, like data link for the first leg of the communication path
Security in IoT

• It’s no longer a myth
  - Most connected embedded systems require secure communication
• Easy interaction with the existing IT infrastructure and cloud servers
  - Same families of protocols...
Security in IoT

...different **technologies**, so different implementation **approach**:

- Resources required by the SSL implementation (RAM, flash, ...)
- Computational power/time to execute encryption operations
- Integration with communication libraries (TCP/IP or other communication stacks)
wolfSSL

• Designed for embedded systems
  – Small footprint, limited amount of resource required
  – Built-in hardware acceleration and assembly optimization
  – Modular to allow scalability to the single algorithm/feature
• Portable and easy to integrate
  – Callback-based API for bare metal and OS integration
  – Built-in support for many OS/environment/platforms
• Mature codebase
• Professional support
• Fast release cycle
• GPL
What is new in TLS 1.3

The new standard protocol for secure communication
Protocols

• Timeline of the protocols standard

Notes:
• SSL 2.0/3.0 are insecure
• SSL = “Secure Sockets Layer”
• TLS = “Transport Layer Security”
• DTLS = “Datagram TLS”
TLS 1.3: major improvements

- Faster handshake (1-RTT/0-RTT)
- Full session encryption
- New cipher suites
- Deprecated vulnerable ciphers and algorithms
- Removed obsolete/insecure features
TLS 1.3: improved handshake

TLS Handshake now requires only one RTT instead of two

Client can start sending data immediately after the first reply from the server

Less RTT == faster handshake, less traffic, less power used
Classic TLS v1.1/v1.2 handshake

Client

Hello!

Key exchange,
Change cipher spec,
Finished.

{encrypted data}

Server

Hello!
Here is my certificate.

Change cipher spec,
Finished.

{encrypted data}

{encrypted data}
TLS 1.3 handshake

Hello!
Key share

Finished+ {encrypted data}

{encrypted data}

Hello! Key share, Certificate, Cert verify, Finished.

{encrypted data}
Encryption algorithms

- TLS uses a variety of encryption algorithms to secure data

**Hashing Functions**
- MD4, MD5, SHA ...

**Block and Stream Ciphers**
- DES, 3DES, AES, ARC4 ...

**Public Key Options**
- RSA, DSS ...

CIPHER SUITE
Ciphers

- Does the configuration support the needed cipher suites?

<table>
<thead>
<tr>
<th>Public Key</th>
<th>Block / Stream</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA, DSS, DH, NTRU</td>
<td>DES, 3DES, AES, ARC4,</td>
<td>MD2, MD4, MD5, SHA-128, SHA-256, RIPEMD</td>
</tr>
<tr>
<td>...</td>
<td>RABBIT, HC-128</td>
<td>...</td>
</tr>
</tbody>
</table>

Ex: **TLS_RSA_WITH_AES_128_CBC_SHA**
A common **CIPHER SUITE** is negotiated during the initial handshake:

- SSL_RSA_WITH_DES_CBC_SHA
- SSL_DHE_RSA_WITH_DES_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS13-AES128-GCM-SHA256
- TLS13-AES256-GCM-SHA384
- TLS13-CHACHA20-POLY1305-SHA256
- TLS13-AES128-CCM-SHA256
- TLS13-AES128-CCM-8-SHA256
TLS 1.3: abandoned algorithms

- The following algorithms are obsolete and should not be used:
  - RC4
  - SHA1
  - MD5
  - SHA224
The following features are obsolete and are no longer part of TLS:

- Compression
- Renegotiation
- All non-AEAD ciphers
- Non-PFS Key exchange (static RSA and static DH)
- Custom DHE groups
- Change Cipher Spec
- Fallback to old SSL standard during negotiation
TLS 1.3: Added algorithms

- The following algorithms are now part of TLS:
  - ChaCha20 symmetric key stream cipher
  - Poly1305 message authentication code
  - Ed25519 and Ed448 digital signature algorithms
  - curve25519 and x448 key-exchange protocols
TLS 1.3: Session resumption

• Older TLS version allowed the client to resume a previously interrupted session
  - Server must look up the session id from its cache
  - Multiple servers should share the same cache

• TLS 1.3 uses session tickets
  - The ticket contains the server state for the session
  - The ticket is stored by the client and used for resumption, but it can only be decrypted and used by the server to resume the session
  - Stateless servers == less resources
The Riot-OS wolfSSL pkg

Design and implementation
First approach: alpha version

- PR #6197: wolfSSL alpha examples
  - Self-contained application to show possible integration
- PR #7348: wolfSSL first pkg with TLS examples
  - Integration via Berkeley socket interface
  - Requires **traditional TCP/IP stack** (e.g. LwIP)
  - TLS client/server (TCP) examples provided
First approach: alpha version

Traditional TCP/IP stack

- IEEE 802.3 or IEEE 802.11
- IPv6
- ICMP6
- UDP
- TCP
- POSIX socket API
- wolfSSL
- TLS Application
Second version: GNRC support and DTLS demo

- PR #9894: wolfSSL pkg with `sock_udp` API integration
  - Built-in callbacks in wolfSSL
  - DTLS 1.2 client/server demo (tested on native)
  - Requires `gnrc_sock_udp`
  - Application API: `sock_tls_t` provided by the module `sock_tls`
Second version: GNRC support and DTLS demo

GNRC UDP/IP stack

- 6LoWPAN
- IEEE 802.15.4
- IPv6/RPL
- ICMP6
- UDP
- sock udp
- wolfSSL
- sock tls
- DTLS Application
Second version: implementation details

- **sock_tls**
  - Front-end to create TLS/DTLS sessions on top of sock_udp
  - Type: `sock_tls_t`
    - Groups together wolfSSL context, session, udp sock and endpoint address for DTLS session
    - Used as context by wolfSSL callbacks

- System integration:
  - Uses `random_uint32()` as random source
Next steps

- Status of native TCP support?
  - TLS over gnrc_tcp?
  - Plans for a generic sock_tcp?
  - HTTPS/wolfMQTTS demos
  - wolfSSH

- Feedback is appreciated. Let’s work together on the best solution!
Thanks!

www.wolfssl.com

daniele@wolfssl.com

info@wolfssl.com