Mbed TLS workshop — PSA Cryptography API

Gilles Peskine

2020-11-02
Platform Security Architecture

A framework for building secure devices – openly published.

- **Analyze**
  - Threat models & security analyses

- **Architect**
  - Hardware & firmware architect specifications

- **Implement**
  - Firmware source code

- **Certify**
  - Independently tested
PSA Developer APIs – making security easy to use

A consistent set of APIs simplifies developer access to security functions across the industry.

Example security functions:
- PSA RoT
- Crypto
- Attestation
- Trusted boot
- Secure Storage

Developers

Any RTOS

Any architecture
Balancing ease of use vs flexibility

- Cryptography is hard to use, easy to misuse
  - Functional tests don’t tell you your code is insecure
- Make the most obvious path secure
  - But please do read the documentation!
- The best crypto API is no API
  - “If you’re typing the letters A-E-S into your code you’re doing it wrong” — Thomas Ptacek
    - Secure storage:
      f = open("/ext/myfile"); read(f);
    - Secure communication: TLSSocket sock;
      sock.connect("example.com", 443);
      But how does this work under the hood?
- Need low-level primitives to implement TLS, IPsec, WPA, LoRaWAN, Bluetooth, GSM, ZigBee, ...
- Need to do dodgy-but-ok-in-this-context things sometimes
  - Deprecated crypto is still in use: MD5 (TLS 1.1), CBC (TLS 1.1), unauthenticated ciphers (storage), RSA PKCS#1 v1.5 (TLS 1.2), 3DES (banking), ...
  - Key derivation in the real world is a mess
  - A key should only be used for one purpose... except when protocols dictate otherwise
Use an existing crypto API?

• Evolve Mbed TLS?
  • Would be hard to add support for opaque keys
  • Too flexible: gives you a lot of rope to hang yourself
  • Very transparent data structures with visible pointers
    – Cumbersome to plug in hardware acceleration or keystore isolation
    – Relies heavily on malloc (so not suitable for e.g. MISRA)

• `cryptlib`? OpenSSL/BoringSSL/LibreSSL/...?
  • Too big, not easy to use

• `NaCl/libsodium`?
  • Not flexible enough: only includes black-box primitives

• Any API in C+/Rust/Go/...?
  • We need C, the common denominator
What about PKCS#11?

PKCS#11 = Cryptoki: standard interface for smartcards

The elephant in the room!

Not so easy to use

- Example: sign with existing key
- /* Discover the key */
  CK_ATTRIBUTE label_attribute =
  {CKA_LABEL, "Fred",
   strlen("Fred")};
  C_FindObjectsInit(hSession,
  &label_attribute, 1);
  C_FindObjects(hSession,
  &hKey, 1, &count);

  /* Sign with the key */
  CK_MECHANISM mechanism =
  {CKM_ECDSA_SHA256, NULL_PTR, 0};
  C_SignInit(hSession,
  &mechanism, hKey);
  C_Sign(hSession, msg, msg_len,
  &sig, &sig_len);

Not the right shape

- Big, we’d have to define a subset
- Key discovery is complex
- Lots and lots of parsing
- Standard compliance is poor in practice
- Not good at access control
  - Designed for a single user
Some API design guidelines

• Make it easy to use, hard to misuse
  • KISbntS: keep it simple, (but not too) stupid

• Uniform interface to memory buffers
  • Explicit sizes throughout
  • You don’t need to understand the algorithm to know how much memory to allocate

• Cryptographic agility
  • Select a key type and mode during key creation
  • Call sequence, buffer size calculations are uniform across algorithms of the same kind

• “Security agility”
  • Single API, multiple isolation levels under the hood
Suitable for limited resources

- Includes multipart APIs for messages that don’t fit in RAM
- The API can be implemented without malloc
  - (Mbed TLS currently uses malloc — maybe Mbed TLS 4.0 will be malloc-free?)
- All algorithms are optional
  - You can build a device with just what you need
Main features

• Cryptographic primitives
  • Symmetric: hash, MAC, unauthenticated cipher, AEAD, key derivation
  • Asymmetric: signature, encryption, key agreement

• Key store
  • All keys are accessed through identifiers
    – No need to know where a key is to use it (RAM, internal storage, secure element, ...)
    – Can run as a library in the same memory space, or as a separate service protected by MPU, MMU, TrustZone, TrustZone-M, ...
  • Simple key policies
    – Declare what operations are allowed (sign, export, ...) and what algorithm

• Random generation

• https://armmbed.github.io/mbed-crypto/psa/
Driver interface

• Combine a core (e.g. Mbed TLS) with one or more drivers

• Transparent drivers
  • For accelerators
  • Operations receive keys in cleartext
  • Can fall back to software (e.g. to deploy the same image on different hardware)

• Opaque drivers
  • For external secure elements, secure enclaves, accelerators with their own key encryption key, ...
  • Operations receive keys in custom format:
    – wrapped key material, or
    – slot number or label of a key stores inside the secure element

• Entropy drivers
Building with drivers

Mbed TLS

Driver 1

acme.c

acme_sign_hash() {
    ...
}

driver.json

{"prefix":"acme",
 "type":"transparent",
 "capabilities":
 [{"entry_points":
  ["sign_hash"]}]}

Driver 2

apex.c

apex_sign_hash() {
    ...
}

driver.json

{"prefix":"acme",
 "type":"opaque",
 "capabilities":
 [{"entry_points":
  ["sign_hash"]}]}

driver_wrappers.c automatically generated

if (location==APEX)
    apex_sign_hash();
else
    acme_sign_hash();

/src/mbedtls$ make PSA_DRIVERS="../acme/driver.json ../apex/driver.json"
/src/myapp$ ld myapp.o ../mbedtls/libmbedcrypto.a ../acme/acme.a ../apex/apex.a
Crypto APIs in Mbed TLS: `mbedtls_xxx` vs `psa_xxx`

- **Functionality**

- **Performance & code size**

<table>
<thead>
<tr>
<th>Mbed TLS versions</th>
<th>2.17</th>
<th>2.x</th>
<th>3.0</th>
<th>3.x</th>
<th>4.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance &amp; code size</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functionality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Useful links

- Arm Platform Security Architecture (PSA):
  https://developer.arm.com/architectures/security-architectures/platform-security-architecture

- PSA Cryptography API information: https://armmbed.github.io/mbed-crypto/psa/
  - Reference documentation: PDF, HTML
  - Driver interfaces (DRAFT): accelerators and secure elements, entropy source

- Mbed TLS: https://github.com/ARMmbed/mbedtls

- Trusted Firmware-M (TF-M):

- We welcome feedback!
  - Public: on the psa-crypto mailing list (psa-crypto@lists.trustedfirmware.org)
  - Confidential: email us at mbed-crypto@arm.com
Thank You
Danke
Merci
谢谢
ありがとうございます
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
toda