arm

Differences between TF-PSA-Crypto 1.0 and PSA Certified Crypto API 1.4

Gilles Peskine

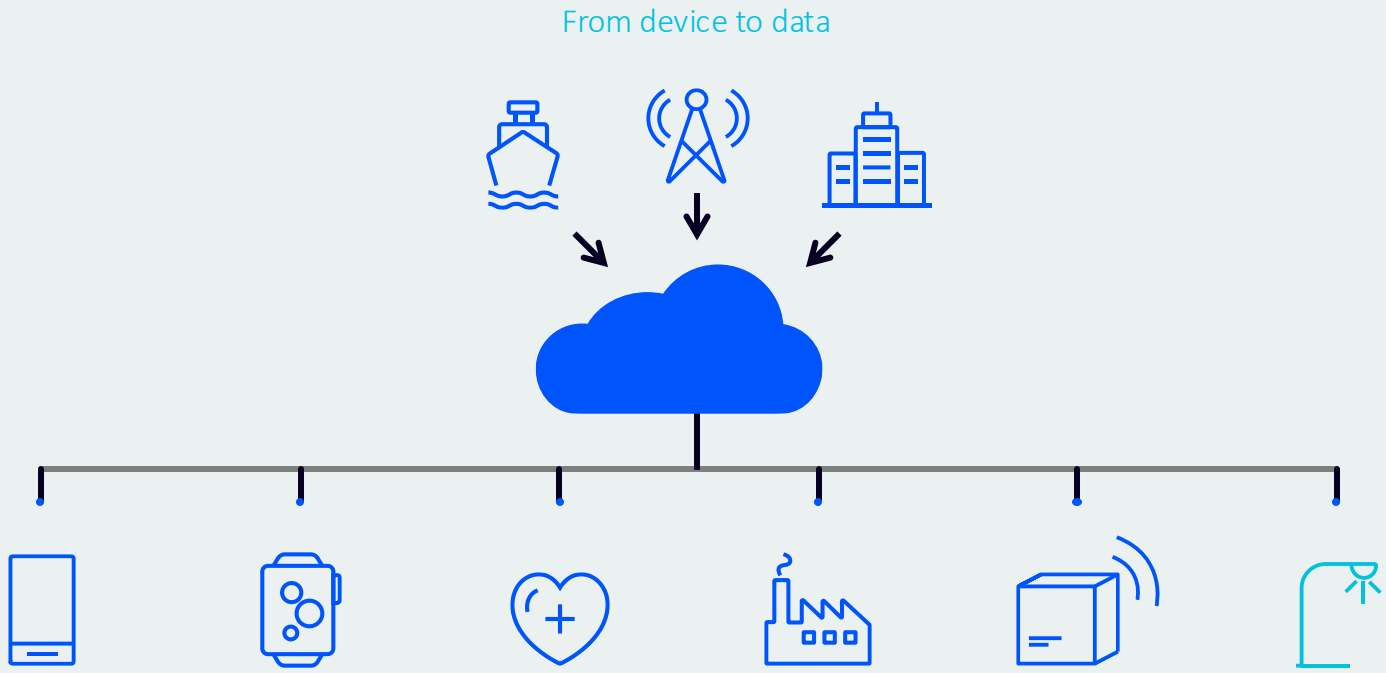
2025-11-25

How it started

1 Trillion securely connected devices

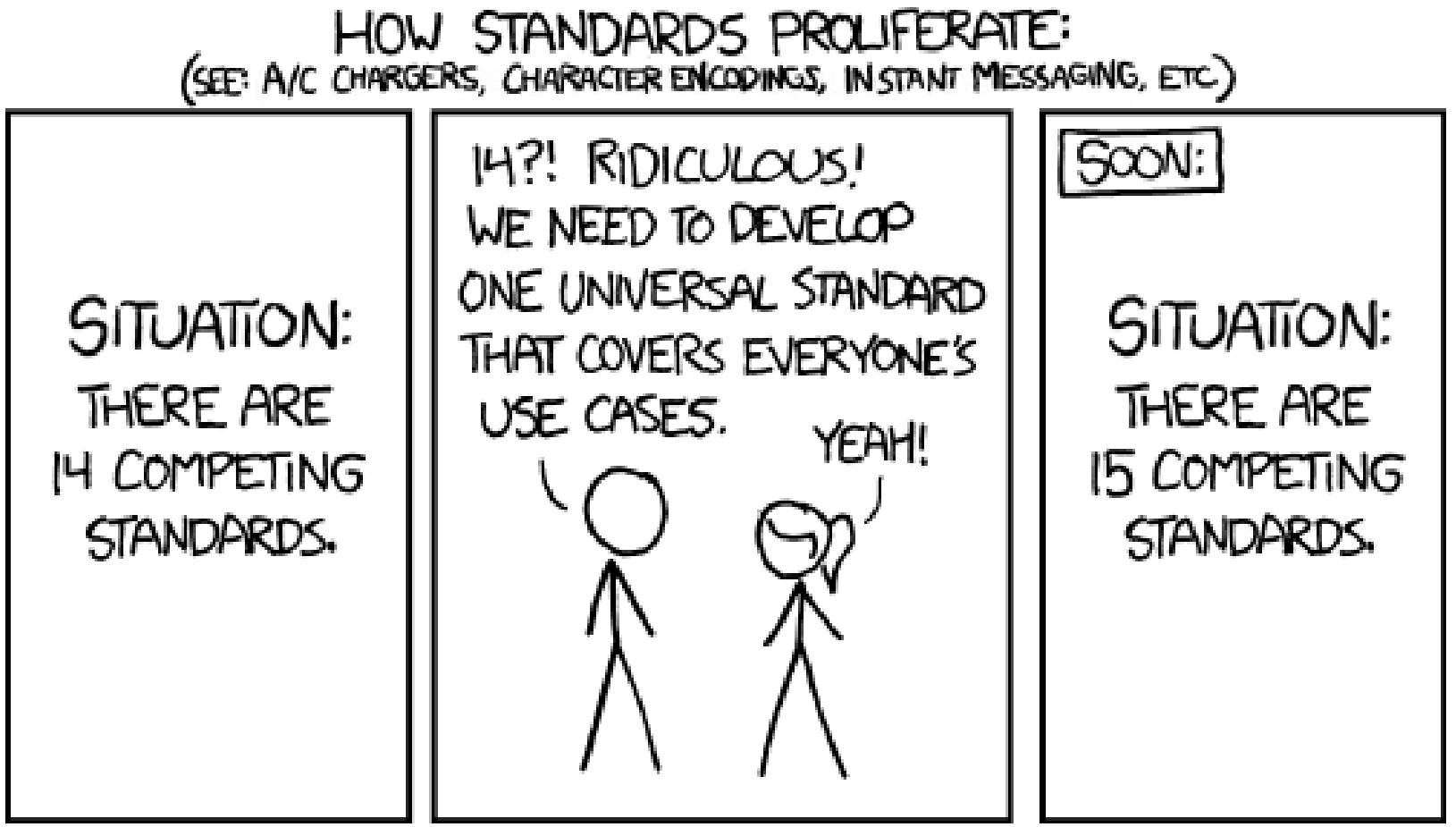
Trust is key for achieving this scale

- Security across entire value chain
- Success built on diversity
- Scalable to low-cost devices



arm

2017: all crypto APIs suck



(In our case it was about robustness and suitability for constrained devices, not features)

Main inspirations

- Tried to be a PKCS#11 light
  - C API + support opaque keys + multipart operations → lots of commonalities
  - No parsing! No variable-sized structures!
- Little inspiration from Mbed TLS
  - Basically only the set of algorithms

The first commit

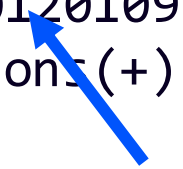
- https://github.com/ARM-software/psa-crypto-api/commit/a430a3b9921b330b53344b61ca8c9032fb85aaa0

```
commit a430a3b9921b330b53344b61ca8c9032fb85aaa0
Author: Gilles Peskine <Gilles.Peskine@arm.com>
Date:   2018-01-11 15:19:07 +0100

    Meeting notes

    Notes from the meeting between Nicolas, Derek and Gilles

notes/psa_crypto.h                          | 401 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
notes/requirements-design-20120109.md | 471 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
2 files changed, 872 insertions(+)
```



(actually 2018)

The first commit in Mbed TLS

- https://github.com/Mbed-TLS/mbedtls/commit/e59236fc17d2b404946a17d4af213c491294a81c

```
commit e59236fc17d2b404946a17d4af213c491294a81c
Author: Gilles Peskine <Gilles.Peskine@arm.com>
Date:   2018-01-27 23:32:46 +0100

    Add PSA crypto module

    New module psa_crypto.c (MBEDTLS_PSA_CRYPTO_C):
    Platform Security Architecture compatibility layer on top of
    libmedcrypto.

    Implement psa_crypto_init function which sets up a RNG.

    Add a mbedtls_psa_crypto_free function which deinitializes the
    library.

    Define a first batch of error codes.
```

The documentation was the specification

- The specification was `include/psa/crypto*.h` until it stabilized in October 2019
  - 1.0.0beta3 was from Doxygen
  - 1.0.0 was its own thing

Things we don't implement

We don't implement all mechanisms

- PSA Crypto normally accepts a mechanism if all are true:
  - There is actual demand for it (or it's from some standard for which there is demand).
  - It fits an existing API (otherwise the bar is higher).
  - There is an existing specification (NIST, RFC, …).
    - Must define full interoperatbility (bit-level format), not just maths in a research paper.
  - It's considered secure OR it has some legacy real-world use.
- Started with almost all that was in Mbed TLS
- Removed some obsolete algorithms
  - MD2, MD4, ARC4, DES, small curves

Missing mechanisms: hash, XOF

Mechanism	In PSA since	TF-PSA-Crypto implementation plans
Obsolete hashes: MD2, MD4	Day 1	Removed
SHA512/224, SHA512/256	Beta 1	If needed
SHA256/192, SHAKE128/256, SHAKE256/192, SHAKE256/256	1.3 ext-pqc	If needed (Intended for hash-based signatures)
SHAKE256_512 hash	1.1.0	When we do Ed448 (Used in Ed448)
SM3 hash	1.0.1	If requested
AES-MMO hash from Zigbee	1.1.0	If requested (specific to the Zigbee protocol)
SHAKE128, SHAKE256	1.4.0	When we get around to it (Likely as part of ML-DSA since it uses SHAKE internally)
Ascon-Hash256, Ascon-XOF128, Ascon-CXOF128	1.4.0	If requested

Missing mechanisms: cipher, AEAD, key wrap

Mechanism	In PSA since	TF-PSA-Crypto implementation plans
Obsolete ciphers: ARC4, DES	Day 1	Removed
SM4 block cipher	1.0.1	If requested
XTS cipher mode	Beta 1	Is in Mbed TLS legacy. PR: #538
XChaCha20 stream cipher **XChaCha20-Poly1305 AEAD**	1.2.0	When we get around to it PR: Mbed-TLS/mbedtls#6556
Ascon-AEAD128	1.4.0	If requested
KW (RFC 3394), **KWP** (RFC 5649)	1.4.0	Currently proprietary `mbedtls/nist_kw.h` PSA when we get around to it

Missing mechanisms: classic asymmetric

Mechanism	In PSA since	TF-PSA-Crypto implementation plans
The PSA spec has all elliptic curves from SEC2	Day 1	No plans to add more SEC2 curves. Removed small curves (<254 bits) in TF-PSA-Crypto 1.0.0.
FRP256v1 elliptic curve	1.0.0 (external request)	No
EdDSA (Ed25519, Ed448)	1.1.0	When we get around to it PR: Mbed-TLS/mbedtls#5819, Mbed-TLS/mbedtls#5824
ECIES	1.3.0	Maybe (can be calculated in the existing API with ECDH)

Missing mechanisms: post-quantum

Mechanism	In PSA since	TF-PSA-Crypto implementation plans
ML-KEM (PQC key exchange using lattices)	1.3 ext-pqc	One of these days
ML-DSA (PQC signature using lattices)	1.3 ext-pqc	Early to mid-2026
SLH-DSA (hash-based signature)	1.3 ext-pqc	If requested
LMS (stateful hash-based signature)	1.3 ext-pqc	Currently proprietary `mbedtls/lms.h` PSA if requested
XMMS (stateful hash-based signature)	1.3 ext-pqc	If requested

Missing mechanisms: derivation and PAKE

Mechanism	In PSA since	TF-PSA-Crypto implementation plans
SP800-108 key derivation (HMAC and CMAC variants)	1.2.0	One of these days
SPAKE2+ HMAC, CMAC and MATTER variants	1.2 ext-pake, 1.3.0	Expected contribution from Silabs
WPA3-SAE (ECC and FFDH keys, H2E key derivation, fixed and GDH variants)	1.4.0	If requested

Missing functions: published API up to 1.3.1

Functions	In PSA since	TF-PSA-Crypto implementation plans
`psa_hash_resume()`, `psa_hash_suspend()`	1.0.0	No (Useful for smartcard emulation)
`psa_key_derivation_verify_bytes()`, `psa_key_derivation_verify_key()`	1.1.0	One of these days (Useful for password verification)
`psa_encapsulate()`, `psa_decapsulate()`	1.3.0	One of these days (Interface of ML-KEM)

Missing functions: upcoming API 1.4.0

Functions	In PSA since	TF-PSA-Crypto implementation plans
`psa_attach_key()`	1.4.0	When we get around to it (Requested by Arm partners)
`psa_check_key_usage()`	1.4.0	When we get around to it (Requested by us, similar to `mbedtls_pk_can_do_psa()`)
Direct key wrapping `psa_wrap_key()`, `psa_unwrap_key()`	1.4.0	When we get around to it (Currently proprietary `mbedtls/nist_kw.h`)
Signature with context `psa_sign_message_with_context()`, `psa_verify_message_with_context()`, `psa_sign_hash_with_context()`, `psa_verify_hash_with_context()`	1.4.0	If ever requested
XOF (Extendable output function API) `psa_xof_setup()`, …	1.4.0	When we get around to it (Likely as part of ML-DSA since it uses SHAKE internally)

Miscellaneous deviations

- Delayed key destruction: when `psa_destroy_key()` returns, copies of the key material can remain in operation structures
  - Hard to implement because our operations are independent (they have a copy of the key)

How we extend the API

Mbed TLS and TF-PSA-Crypto extensions to PSA Crypto

- Extensions are declared in `include/psa/crypto_extra.h` if possible
- Sometimes called `psa_xxx`, sometimes `mbedtls_psa_xxx` (we have been inconsistent)
- Note: here I won't discuss implementation aspects that don't affect the API
  - E.g. RNG choice, client-server builds, key store size, …

Enrolment algorithm in policies

- PSA: a key is used by a single algorithm
  - (Or wildcard policy, e.g. `PSA_ALG_RSA_PSS(PSA_ALG_ANY_HASH)`)
- Extension: a key can have two algorithms

- Motivation: a customer needed ECC keys used for ECDH, but with a certificate whose enrolment involves making an ECDSA signature
- Not standardized because it seemed very ad hoc and I hoped to find a better solution
  - 6 years later, still no better solution

Crypto subsystem deinit

- `mbedtls_psa_crypto_free()`: deinit for `psa_crypto_init()`
- Motivation: we want fully self-contained code
  - Deinit at the end of each test case
  - Deinit at the end of programs so that memory leak detection tools can report no leak
- Not standardized because we didn't want to imply that it's necessary
  - But we could standardize it anyway, just insist that it's optional like everything else
- Related: `mbedtls_psa_get_stats()`
  - Only intended to check that everything has been freed on deinit (and give a bit of info if not)

DSA key types

- Finite-field DSA was in an early draft because GlobalPlatform TEE requires it
  - Requested in Mbed TLS: Mbed-TLS/mbedtls#1321, Mbed-TLS/mbedtls#9318
  - But we'll never implement it in TF-PSA-Crypto (that ship has long sailed)
- Didn't actually make it into the API
  - Nobody wanted it *enough*
  - Doesn't fit well:
    - Key generation needs domain parameters as input
    - The private key size and the public key size are partially independent
- Still present in `crypto_extra.h` in 1.0.0 because merging is hard

Platform random generator (integrator callback)

- Integrators can provide a callback `mbedtls_psa_external_get_random()`
  - High-speed, crypto-quality rangom generator (not just an entropy source)
- Not an *application* interface feature, should probably be in `mbedtls/platform.h`
  - It's in a PSA header because we originally kept PSA out of the way, and then stuck with the habit

Built-in keys (integrator callback)

- Integrators can provide a callback `mbedtls_psa_platform_get_builtin_key()`
- One of many possible ways of giving access to “magically” pre-existing keys
- Not an *application* interface feature, so probably belongs in a different header
- A different method will be standardized with PSA crypto drivers

PAKE

- Was originally an extension to the PSA API
- PAKE has been standardized, so it should be moved out of `crypto_extra.h`

Interruptible asymmetric operations

- Interruptible signature, key agreement, key generation, export-public-key
- Intended for extremely constrained platforms with no OS
  - Need to check radio state while doing lengthy crypto
  - Called “restartable ECC” in Mbed TLS
- PSA standardization in progress
  - ARM-software/psa-api#107, ARM-software/psa-api#199
  - We already implement the draft standard
  - Will be finalized when it escapes the low-priority well

Configuration mechanism

- `#define PSA_WANT_xxx 1` to enable support for `PSA_xxx`
  - A few deviations, see documentation
  - Additive configuration system (“give me this feature”), unlike legacy Mbed TLS (“enable this piece of code”)
- Motivation: we needed something
  - And it's much more convenient than the legacy way
- Not standardized
  - Maybe one day
  - Maybe only as compile-time discovery mechanism?
    - i.e. applications can use `#if PSA_WANT_FOO` to check if `PSA_xxx` is available

Other properties

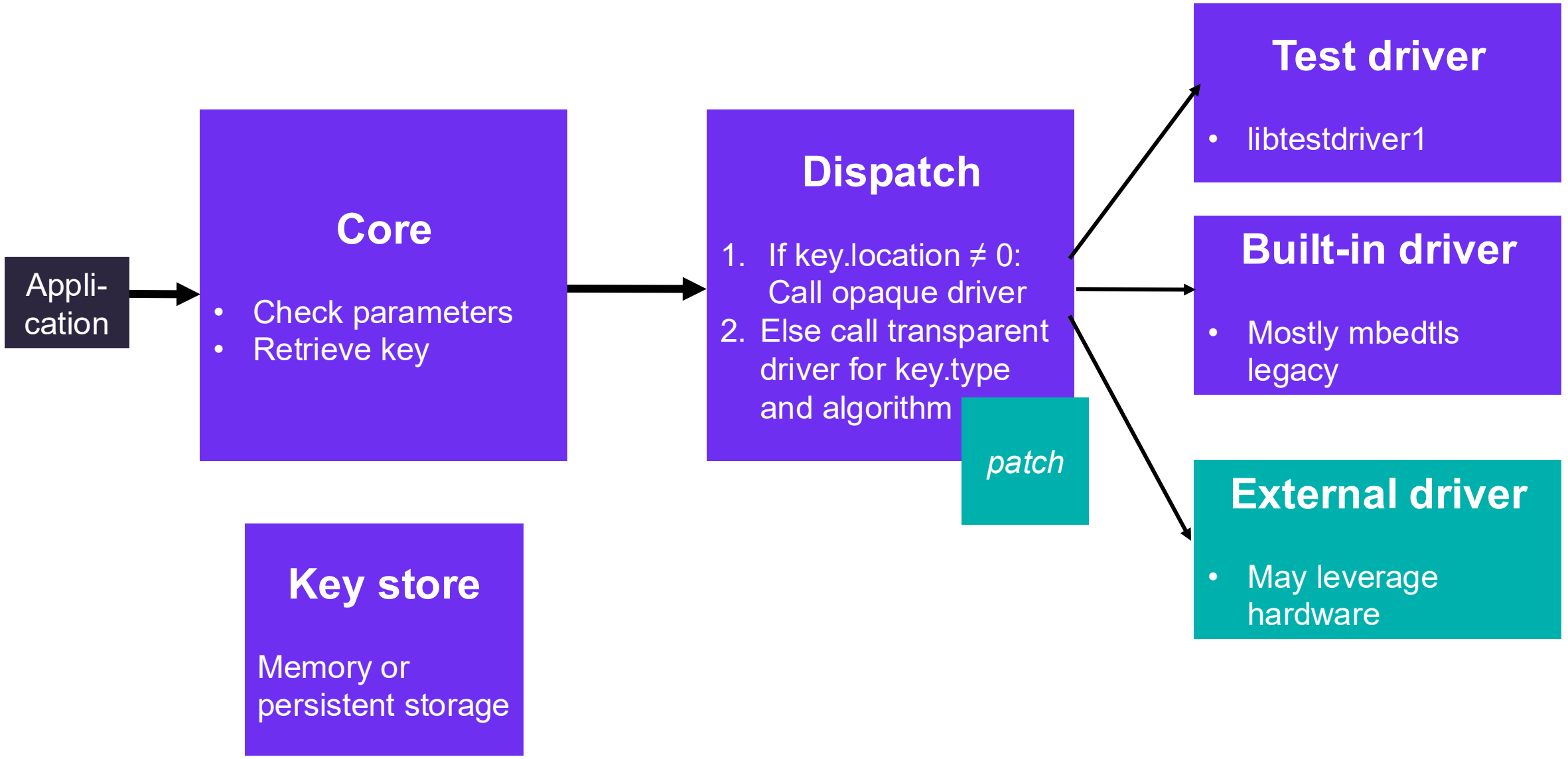
- Safe outputs on failure
  - `*output_length <= output_size`
  - MAC/signature output is something never valid
  - No partial plaintext on decrypt failure
  - etc.
- etc.

Drivers

Why drivers?

- Transparent drivers: benefit from hardware acceleration
  - Or alternative software implementation
  - Driver chosen based on capabilities for the algorithm, key type/size, operation
    - Can do runtime fallback
- Opaque drivers: key material not directly accessible
  - Hardware unique key, protected environment, secure element, …
  - With storage: external hardware stores the key and the core accesses it through a label
  - Without storage: the core has a wrapped blob containing the key in encrypted form
  - Driver determined by the key location (part of the lifetime attribute)
- Random generator drivers
  - Custom entropy sources (currently `MBEDTLS_PSA_DRIVER_GET_ENTROPY`)
  - Or benefit from a secure element's RNG (currently `MBEDTLS_PSA_CRYPTO_EXTERNAL_RNG`)

Anatomy of a crypto operation



Driver interface standardization

- PSA standardization started in Sep 2025
  - Participating implementers: TF-PSA-Crypto, Oberon, RIOT OS
  - https://discord.com/channels/1106321706588577904/1417530000575561891
  - https://github.com/ARM-software/psa-api/issues?q=state%3Aopen%20label%3A%22Crypto%20Driver%22
- Well-established: what C functions the core calls
- Needs work: how to declare driver capabilities to the dispatch core
- Needs work: how can a driver call another crypto operation?
- Needs work: how to plug drivers into an implementation

Driver description file

INITIAL PLAN
only ~5% implemented
(not enough to do
anything useful)

- Describes the capabilities of a driver
  - What key types, algorithms, operations are supported
  - Names of functions to call
- Example: `p256m.json` (alternative implementation of secp256r1)

```
{
    "prefix":       "p256",
    "type":         "transparent",
    "headers":      ["p256-m_driver_entrypoints.h"],
    "capabilities": [
        {
            "entry_points": ["import_key", "export_public_key", "sign_hash", "verify_hash"],
            "algorithms": ["PSA_ALG_ECDH", "PSA_ALG_ECDSA(PSA_ALG_ANY_HASH)"],
            "key_types": [
                "PSA_KEY_TYPE_ECC_KEY_PAIR(PSA_ECC_FAMILY_SECP_R1)",
                "PSA_KEY_TYPE_ECC_PUBLIC_KEY(PSA_ECC_FAMILY_SECP_R1)"
            ],
            "key_sizes": [256],
        }
    ]
}
```

Standardizing the dispatch layer

- All current driver integrators write their own dispatch code
  - Because there was no other way
  - Because they don't like to generate C code as part of the build
    - (But they could do it once and for all)
- Hence:
  - Need standard function prototypes
  - Need to plug external dispatch code into the build
- Also: driver calling back to the core
  - E.g. compute a hash or a block cipher during signature, KDF, PAKE, …
  - Can't call the core
    - Might not be possible in client-server builds
    - Would need to access the key store for operations with a key — problematic for reentrancy
  - So drivers should call the dispatch layer
    - Drivers can't easily call *other* drivers since they don't know which driver to call

Built-in keys

- Vague term covering any key that isn't created by import/generate/... by “the application”
  - Hardware unique key
  - Key injected out of band during storage provisioning
  - Key held by a companion IP
- Strongly related to drivers because they're often provided as part of the same IP as an accelerator or secure element
- Currently: `MBEDTLS_PSA_CRYPTO_BUILTIN_KEYS`
  - Not well-suited to the needs of many partners
  - Design in progress

arm

Merci
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Thank You
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు
Köszönöm