DPDK

SUMMIT

September 24–25, 2024| Montreal, Canada

#DPDKSUMMIT

OpenSSL PMD: Analysis and Optimisations

Jack Bond-Preston, Arm and Honnappa Nagarahalli, Arm

Introducing the CFP Author

Jack Bond-Preston
Software Engineer
Arm

Jack is a Software Engineer at Arm, which he joined after graduating with a degree in Computer Science from the University of Bristol.

Jack's main interests in the field include networking, performance engineering, software profiling, and security.

Jack's first major contributions to the DPDK project were the OpenSSL changes detailed in this presentation. Since then, Jack has also taken maintainership of several minor DPDK components: Bitops, ARMv8 Crypto, and Ticketlock.

Due to travel constraints, Jack can't travel to the DPDK Summit this year.

Background

Existing DPDK software crypto PMDs

ARMv8

- Arm platforms only
- Utilises Arm's AArch64cryptolib, which provides performant assembly implementations of crypto algorithms – making use of Arm-specific CPU instructions for acceleration, and optimised for specific cores.
- Supported algorithms: AES-GCM-{128,192,256} and AES-CBC-128 (with SHA-{1,256} HMAC authentication)
- Library is low-level, with a thin API
- **Arm plans to deprecate this library, dropping future maintenance and support**

ipsec-mb

- Utilises either Intel's ipsec-mb library, or Arm's ipsec-mb library (fork)
- Arm's fork has a much more minimal set of supported algorithms compared with Intel's ipsec-mb: ZUC-EEA3[-256] and SNOW3G-UEA2 (both 3GPP ciphers)
- Has a thinner API than OpenSSL

OpenSSL

- Utilises the well-known OpenSSL library (widely packaged for distributions)
- OpenSSL supports an extremely large range of algorithms, although the DPDK PMD only supports a (still large) subset of these
- OpenSSL has a much heavier API than the other two libraries, due to providing a highly generic API to use across many algorithms, on many platforms
- OpenSSL is maintained by the community, and managed by the OpenSSL Corporation and Foundation

What are the problems, in the context of Arm platforms?

- High level of fragmentation
  - Depending on which crypto algorithm you wish to use, you may need/want to use different PMDs.
- AArch64cryptolib deprecation
  - Aarch64CryptoLib is a prime candidate for deprecation, however the replacement needs to have similar performance.
  - The replacement needs to have similar performance, otherwise users won't want to switch.
- OpenSSL PMD performance is suboptimal
  - On first look, the OpenSSL PMD performance seems to be much worse than the others.
  - More on this later...

What are the solutions?

- High level of fragmentation
  - Try to reduce the number of SW Crypto PMDs, consolidating algorithm support into fewer libraries – deprecation of Armv8 Crypto PMD.
- AArch64cryptolib deprecation
  - Enable support in the OpenSSL PMD for the algorithms supported by Armv8 crypto PMD, with comparable performance.
- OpenSSL PMD performance is suboptimal
  - Optimise the OpenSSL PMD implementation, within DPDK.
  - Optimise the OpenSSL library itself, especially by adding performant assembly implementations of common algorithms (utilising AArch64 cryptography instructions where possible, for AArch64cryptolib parity).

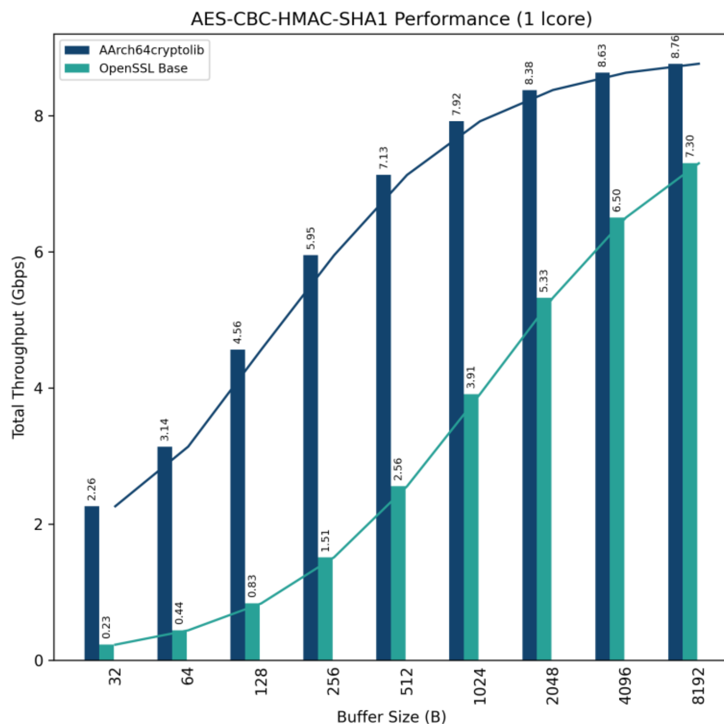
DPDK OpenSSL PMD Performance

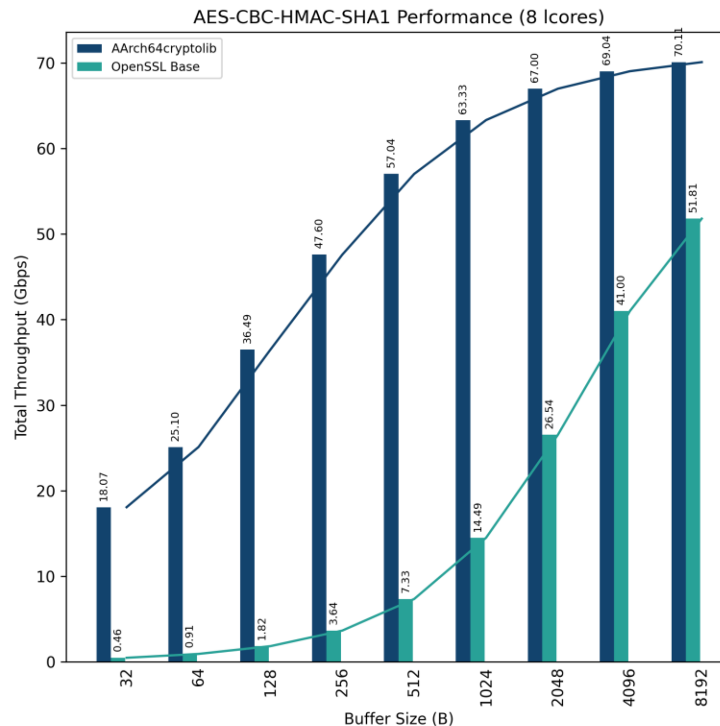
Previous state of OpenSSL PMD

- The OpenSSL PMD is not often updated
  - Vendors/users are largely focussed on other PMDs, such as ipsec-mb or hardware crypto PMDs.
  - Changes and bugfixes have been made in the past, without prioritising performance and/or correctness.[1]
- The PMD already supported the algorithms provided by the ARMv8 PMD, however the performance was much lower.
  - The OpenSSL PMD was spending much less time in the actual crypto assembly implementation, compared with the ARMv8 crypto PMD.
  - In AArch64cryptolib the assembly implementation of the algorithm itself interleaves crypto and auth operations. OpenSSL doesn't do this optimisation.

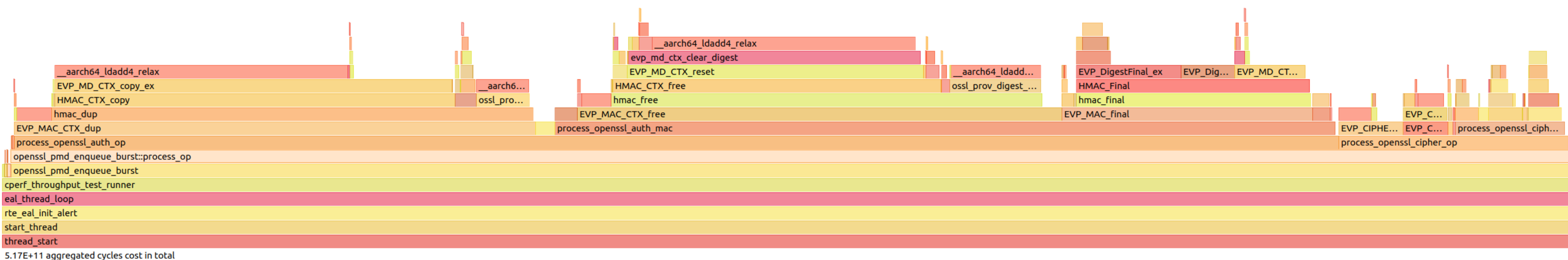
[1] e.g. 75adf1e (introduced a bug), 6b283a0, 67ab783 (fixed bugs but introduced performance hits)

Throughput: ARMv8 PMD vs OpenSSL PMD





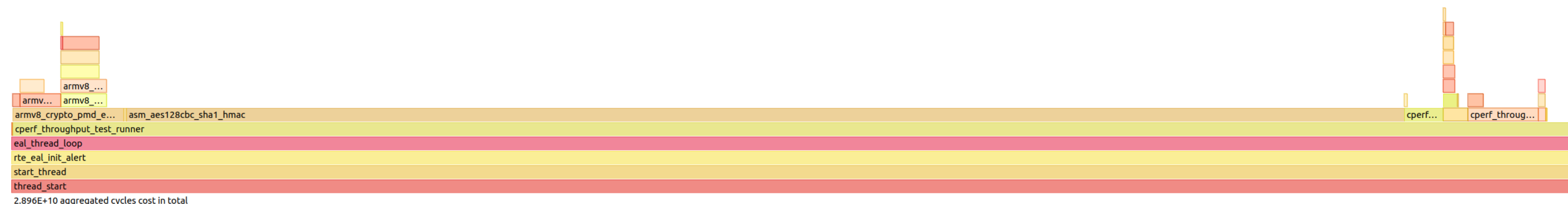
OSSL PMD Flamegraph (AES-128-CBC-HMAC-SHA1 Enc, 8 lcores)



Symbol	Binary	cycles (incl.)
__aarch64_ldadd4_relax	libcrypto.so.3	52.2%
EVP_MD_CTX_copy_ex	libcrypto.so.3	9.68%
EVP_DigestUpdate	libcrypto.so.3	4.84%
EVP_DigestFinal_ex	libcrypto.so.3	4.47%
sha1_block_armv8	libcrypto.so.3	2.41%
OPENSSL_cleanse	libcrypto.so.3	2.19%
[…]		
aes_v8_cbc_encrypt	libcrypto.so.3	1.16%

In green are the actual cipher/auth algorithm implementations, which we want to be consuming the most cycles

ARMv8 PMD Flamegraph (AES-128-CBC-HMAC-SHA1 Enc)



Symbol	Binary	cycles (incl.)
asm_aes128cbc_sha1_hmac	dpdk-test-crypto-p...	81.9%
cperf_throughput_test_runner::rte_cryptodev_enqueue_burst	dpdk-test-crypto-p...	3.53%
armv8_crypto_pmd_enqueue_burst::rte_ring_enqueue_burst::rte_ring_enqueue_burst_elem::rte_ring_sp_enqu...	dpdk-test-crypto-p...	2.33%

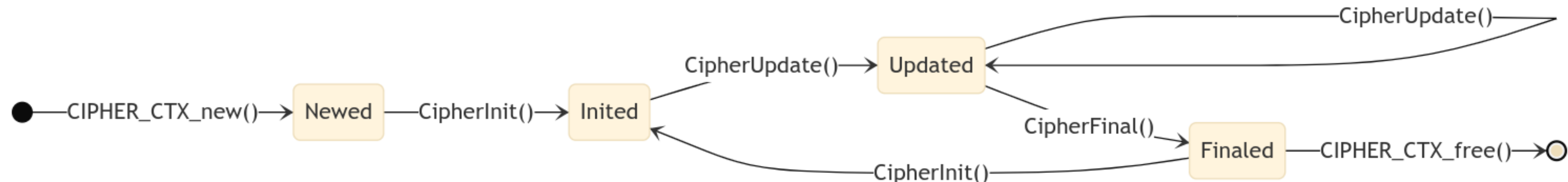
Optimising the DPDK OpenSSL PMD

EVP_CIPHER_CTX management

OpenSSL is interfaced with using the EVP (EnVeloPe) API. This provides access to a wide variety of operations through a unified API

EVP_CIPHER_CTX structures (cipher contexts) store the cipher state:

- This context roughly maps onto one IPsec tunnel.
- The contexts contains things like the cipher implementation being used, the key, the IV, pending unciphered trailing buffer data, etc
- The context has a variety of states with a somewhat complex lifecycle. However, the standard lifecycle we use is:



DPDK stores the cipher context inside the OpenSSL session structure

- The OpenSSL session structure represents one flow – with one key, cipher type, direction, etc.
- **DPDK previously created a duplicate of the cipher context for every (encrypt/decrypt) operation.** This was because cipher contexts are not thread safe. One session can be shared across multiple worker threads, so using the cipher context from the session structure, without cloning it, will lead to issues

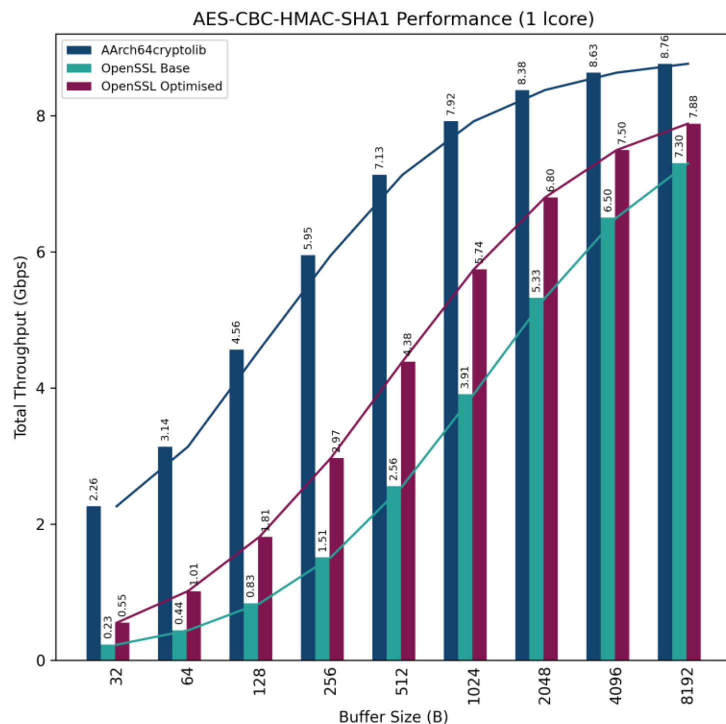
Reducing frequency of cipher context duplication

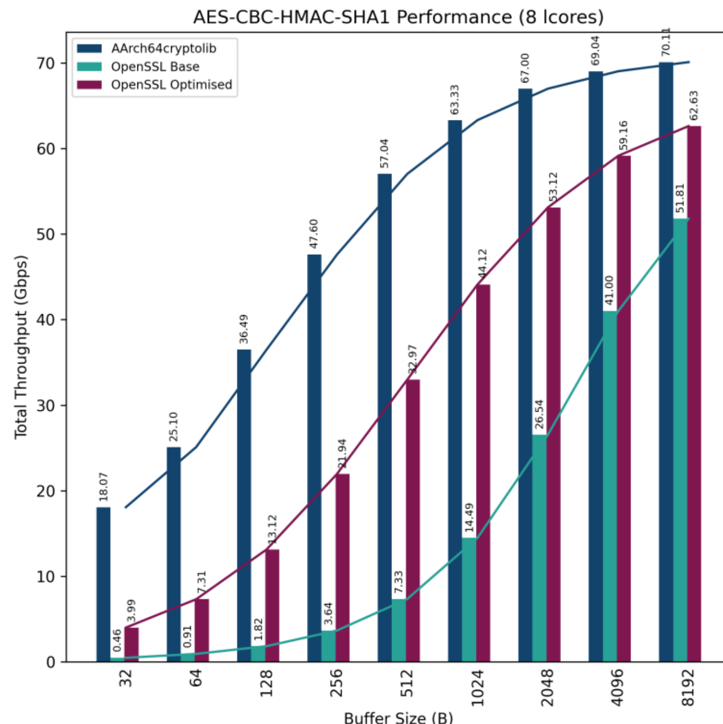
- Duplicating the cipher context per-buffer can be very expensive, especially for smaller buffer sizes where the crypto operation itself takes up less time
- We can use an alternate approach: maintain a per-queue-pair cipher context clone, which has a lifetime across the whole session
  - Queue-pairs can only be used from one thread at a time, so this prevents thread-safety issues. We use per-queue-pair instead of per-lcore as [# QPs allocated to PMD] should be <= [# total lcores]
  - This will increase memory usage (as the lifetime for these clones expands), but drastically reduce the amount of (expensive) cipher context duplications that need to be performed
- This also prevents a multi-threading performance issue that keen-eyed people may have spotted in the flamegraph
  - Each cipher context clone is extra expensive in a multithreaded use-case. This is due to a globally maintained EVP_CIPHER instance inside the EVP_CIPHER_CTX, which uses refcounting to handle its lifecycle. Regularly cloning EVP_CIPHER_CTXs (and thus the contained EVP_CIPHERs) thrashes the refcount and ruins performance scaling. This explains time spent in __aarch64_ldadd4_relax in the hotspot analysis
  - Not cloning the cipher context per-buffer negates this issue

Status

- The aforementioned approach was implemented as a patchset for the DPDK OpenSSL PMD
  - Performance increases for most algorithms are quite large, especially for smaller buffer sizes (where the overhead of the cloning is a larger percentage of runtime)
  - Throughput uplift ranging from ~1000% increase (8 lcores, 32B buffers) to 5% (1 lcore, 8192B buffers)
- Patchset accepted and ships as part of DPDK release 24.07.
  - Thanks all for feedback and reviews, especially Nick Connolly and Wathsala Vithanage from Arm, and Kai Ji and Akhil Goyal from the DPDK community
  - Users of the OpenSSL PMD across all platforms should now be able to see crypto speedups with no code changes necessary

Throughput: ARMv8 PMD vs Optimised OpenSSL PMD





Optimising the OpenSSL AES-CBC-HMAC-SHA Implementation

Summary

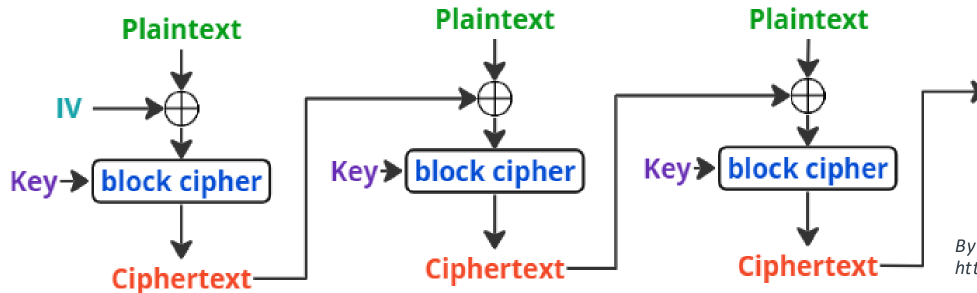
1) AArch64cryptolib already has a well-optimised implementation of interleaving AES-CBC-HMAC-SHA
   - This algorithm combines encryption (AES-CBC) and authentication (HMAC-SHA)
   - The implementation using interleaving, which means we perform the encryption and authentication together, instead of fully encrypting *then* fully authenticating
2) Arm took this assembly implementation and created a PR to add this to OpenSSL
   - Also slightly optimised the routine by taking advantage of some AES instruction fusing – so the implementation should perform a little better than AArch64cryptolib's
3) A patch to DPDK is needed to integrate this into the OpenSSL PMD
   - There are some difficulties here - should this be provided to the user as a separate algorithm (interleaved AES-CBC-HMAC-SHA), or transparently used when the user selects the existing AES-CBC encryption algorithm combined with existing HMAC-SHA authentication algorithm?

Problems with integrating into DPDK's PMD

- The initial implementation doesn't support partial updates
  - Normally, DPDK takes advantage of the ability of OpenSSL to supply buffer data in multiple segments (partial updates), by repeatedly calling EVP_CipherUpdate()
  - This isn't supported for Arm's initial port of interleaving AES-CBC-HMAC-SHA (since AArch64cryptolib never supported this)
  - There is no way for the PMD to inform the user that this isn't supported, on a per-algorithm basis. The only option would be to error out on segmented buffers with this particular algorithm combination, and document that it isn't supported. However, if we want to transparently use interleaving without making the user select it, this would represent a feature regression.
- Solution: implement support for partial updates in OpenSSL's implementation ourselves

Background: AES-CBC

- AES-CBC is a block cipher used to encrypt/decrypt data. AES (Advanced Encryption Standard) is the cipher algorithm, and CBC is the mode of operation (Chained Block Cipher).
  - Data is processed block-by-block, with the output of one block forming an input to the next block
  - The first block takes an IV (Initialisation Vector). For subsequent blocks, the algorithm takes the cipher output of the previous block
  - Due to this dependency, CBCs are poorly parallelisable



By Epachamo - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=147126044

- AES can take key sizes of either 128, 192, or 256 bits.

Confidential © 2024 Arm

Implementing partial updates

- AES-CBC encryption and SHA authentication both process data in set block sizes.
  - OpenSSL supports buffering trailing data and only providing the algorithm with multiples of its block size (until the CipherFinal() call)
  - However, the block sizes are different: AES-CBC uses 128b/16B blocks, and SHA uses 512b/64B, 512b/64B, 1024b/128B blocks for SHA-1, SHA-256, and SHA-512 respectively
  - Solution: Increase the block size of the interleaved cipher + auth to the size of the respective SHA algorithm used, to ensure the data processed always meets the minimum block size
- We need to store some intermediate state to ensure the multiple buffers are treated as segments of the same buffer by the algorithm.
  - For AES-CBC, need to store the output of each chunk of ciphered data, and pass it to the next chunk.
  - For SHA-HMAC, need to save the internal SHA state at the end of the chunk, and use it as the initial state for the next chunk.
  - Solution: store inside the algorithm specific struct, inside the cipher context.

DPDK integration: remaining issues

One problem remains: the new approach cannot handle mismatched authentication and cipher offsets.

- Previously we asked OpenSSL to do the cipher and auth operations individually, so it was trivial to use a different offset for each.
- This is very hard to implement with interleaving operations, and OpenSSL doesn't really have the API features to allow this anyway.
- We probably still need to make the user request the interleaving version of the algorithms, and state that this isn't supported for these in the documentation. Alternatively, we can fall back to re-calculating the auth after the interleaved operation finishes (slower, but produces correct results – may be acceptable if only a few ops use mismatched offsets)
- It would be good to understand the typical usecase for mismatched authentication and cipher offsets. Any opinions on ideal solutions for this issue, from the DPDK crypto PMD side?

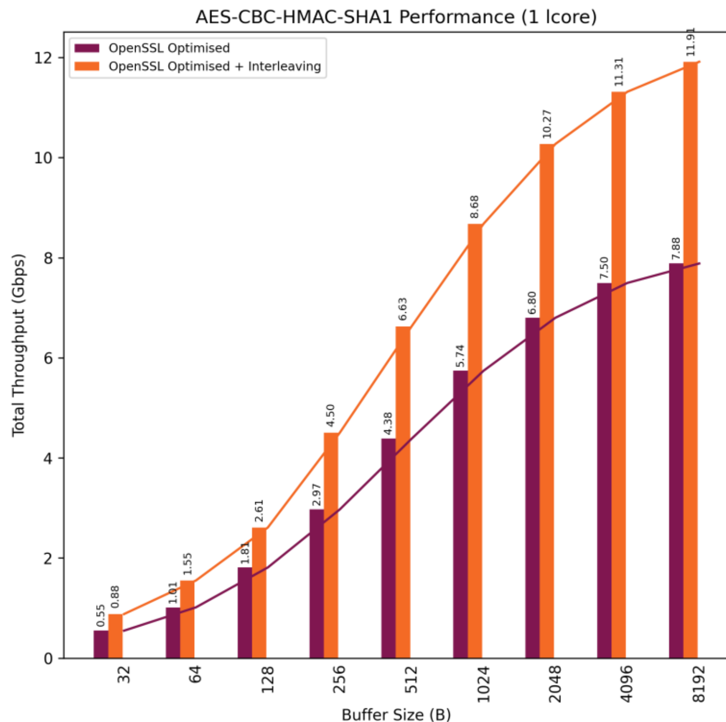
Status

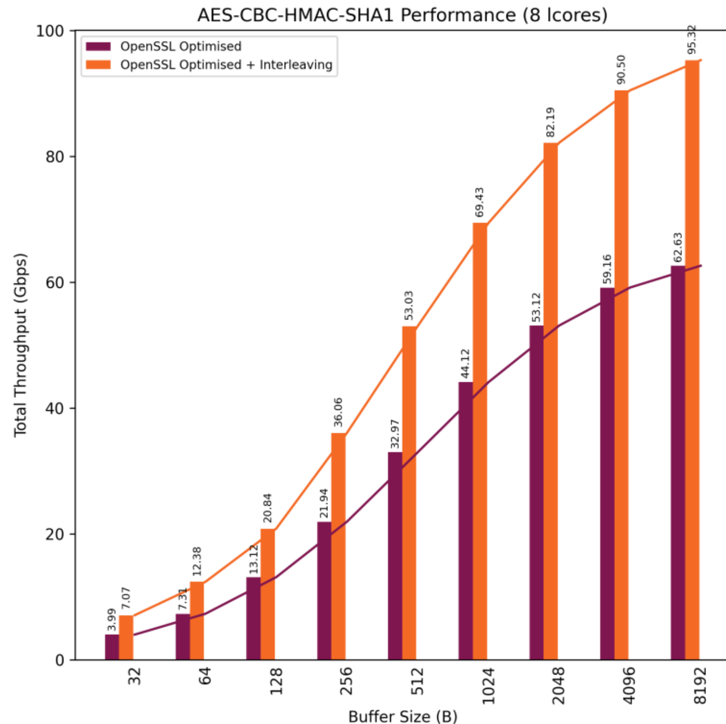
- Arm's patches are upstreamed on Github as an OpenSSL PR, pending merge
  - Hopefully targeting October's OpenSSL release
  - This does not include partial update support, which will be upstreamed later (still needs some final work internally)
- Once these changes are eventually merged into OpenSSL releases, support for the features can be upstreamed for DPDK
  - A work in progress patch has been started inside Arm for this support, but some issues are still pending (e.g. the mismatched auth and cipher offset issue)

Confidential © 2024 Arm

arm

Throughput: ARMv8 PMD vs Optimised OpenSSL PMD w/ Interleaving





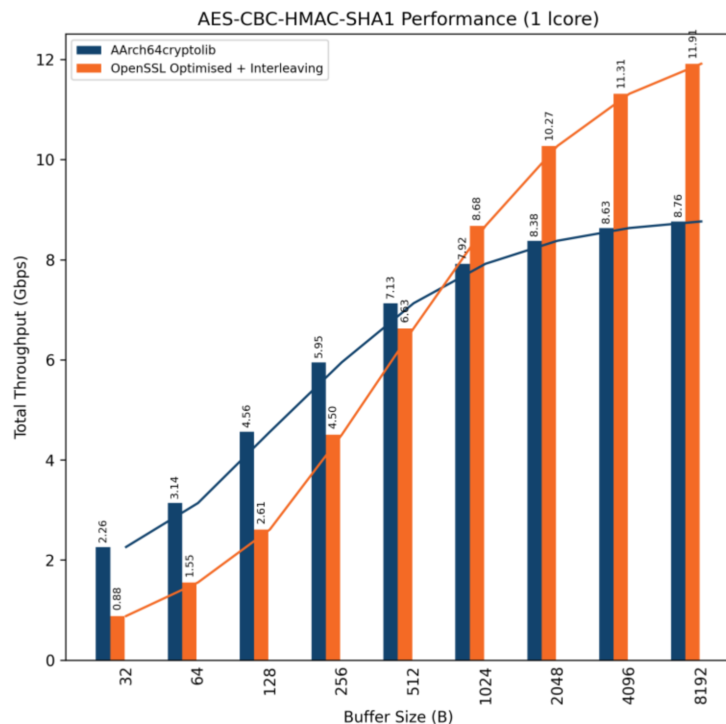
DPDK SUMMIT

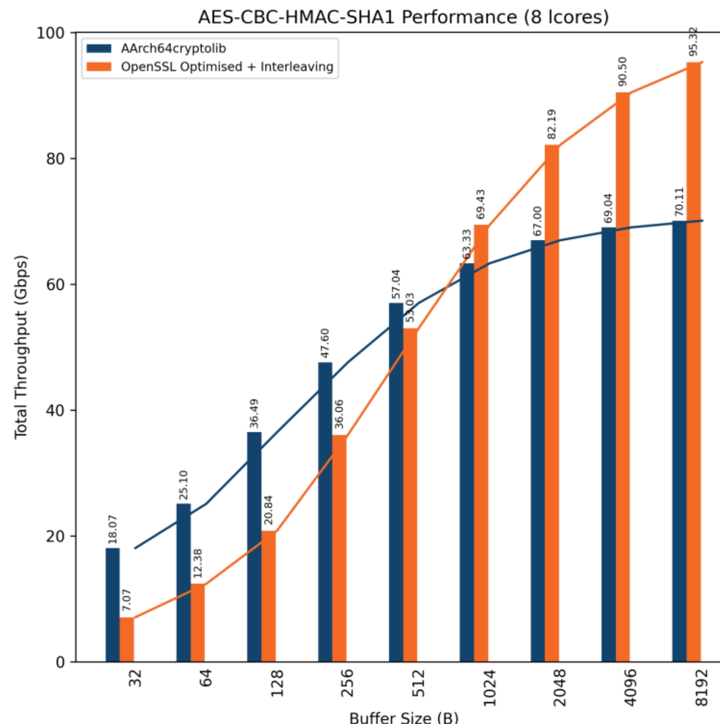
Future Work and Considerations

Future Work

- More algorithms can be supported fairly easily by the DPDK PMD (where support already exists in OpenSSL).
  - Just a case of plumbing through various DPDK crypto PMD API aspects and utilising the OpenSSL EVP API correctly.
- There may be some room to improve OpenSSL's performance more.
  - One major area is EVP parameter handling. This is all string-based and performs a lot of string comparisons on every parameter set/get operation. Plans and work are underway in the OpenSSL project to improve this.
  - Some investigations into whether the performance of the OpenSSL API can be improved are ongoing within Arm, too.

AArch64cryptolib/ARMv8 PMD Deprecation Discussion





DPDK

SUMMIT

September 24–25, 2024| Montreal, Canada

#DPDKSUMMIT

Addendum: Throughput charts for all PMD iterations