Performance Investigations

Hannes Tschofenig, Manuel Pégourié-Gonnard
25th March 2015

Motivation

- In <draft-ietf-lwig-tls-minimal> we tried to provide **guidance** for the use of DTLS (TLS) when used in IoT deployments and included **performance data** to help understand the design tradeoffs.
- Later, work in the IETF DICE was started with the profile draft, which **offers detailed guidance** concerning credential types, communication patterns. It also indicates which extensions to use or not to use.
- Goal of <draft-ietf-lwig-tls-minimal> is to offer performance data based on the recommendations in the profile draft.
- This presentation is about the current status of gathering performance data for later inclusion into the <draft-ietf-lwig-tls-minimal> document.

Performance Data

- This is the data we want:
  - Flash code size
  - Message size / Communication Overhead
  - CPU performance
  - Energy consumption
  - RAM usage
- Also allows us to judge the improvements of various extensions and gives engineers a rough idea what to expect when planning to use DTLS/TLS in an IoT product.
- <draft-ietf-lwig-tls-minimal-01> offers preliminary data about
  - Code size of various basic building blocks (data from one stack only)
  - Memory (RAM/flash) (pre-shared secret credential only)
  - Communication overhead (high level only)

Overview

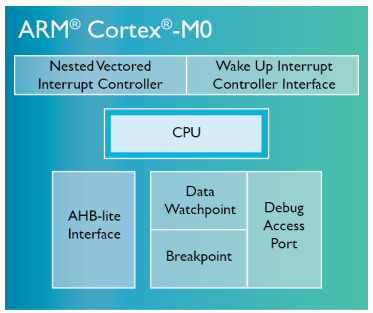
- Goal of the authors: Determine performance of asymmetric cryptography on ARM-based processors.
- Next slides explains
  - Assumptions for the measurements,
  - ARM processors used for the measurements,
  - Development boards used,
  - Actual performance data, and
  - Comparison with other algorithms.

Assumptions

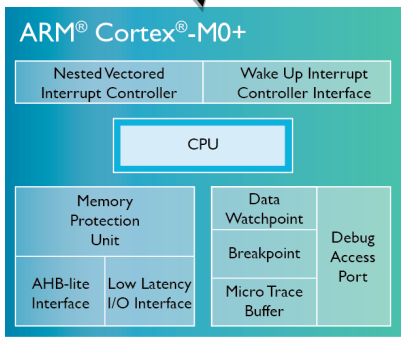
- Main focus of the measurements so far was on
  - raw crypto (and not on protocol exchanges)
  - ECC rather than RSA
  - Different ECC curves
  - Run-time performance (not energy consumption, RAM usage, code size)
- No hardware acceleration was used.
- Used open source software; code based on PolarSSL/mbed TLS stack.
- No hardware-based random number generator in the development platform was used → Not fit for real deployment.

ARM Cortex-M Processors

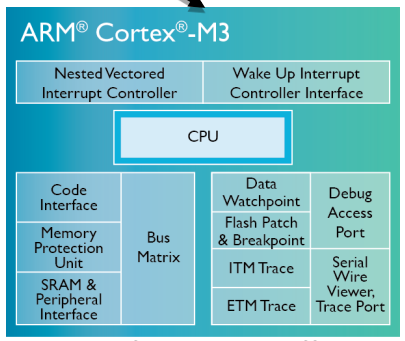
Processors used in the performance tests



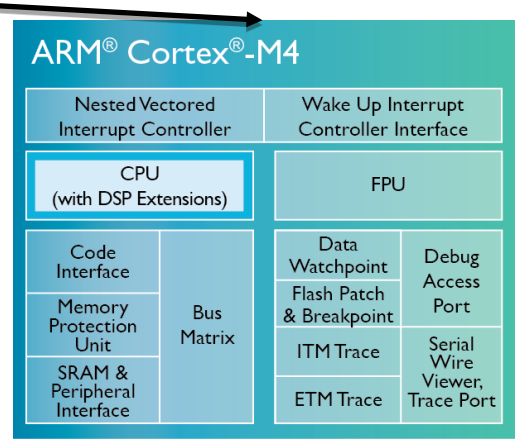
Lowest cost
Low power



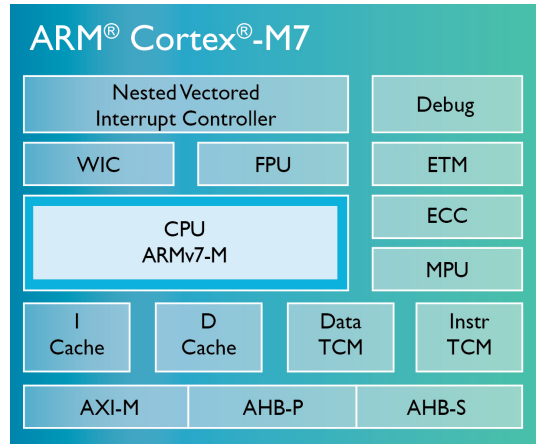
Lowest power
Outstanding energy efficiency



Performance efficiency
Feature rich connectivity



Digital Signal Control (DSC)
Processor with DSP
Accelerated SIMD
Floating point (FP)

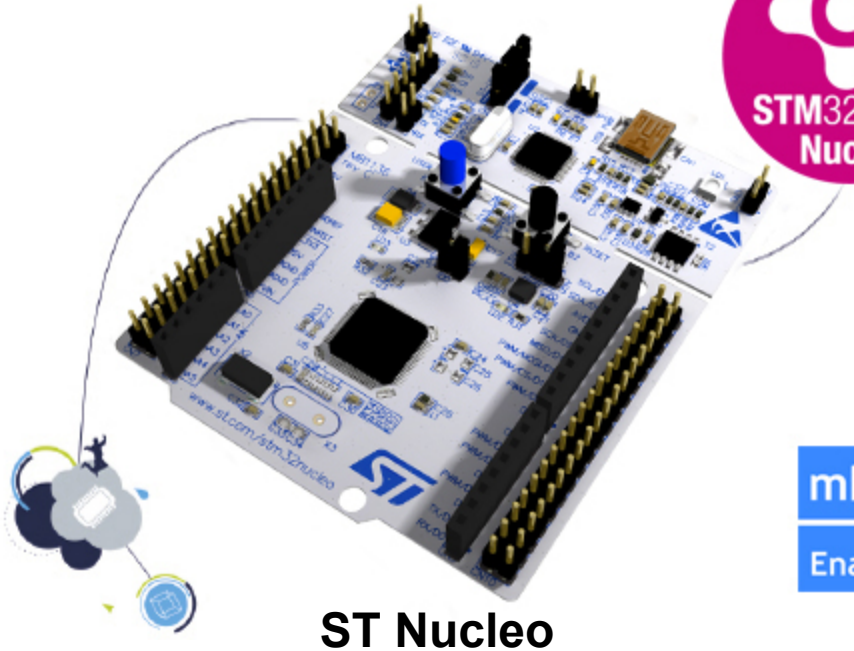


Recently released;
Best performance

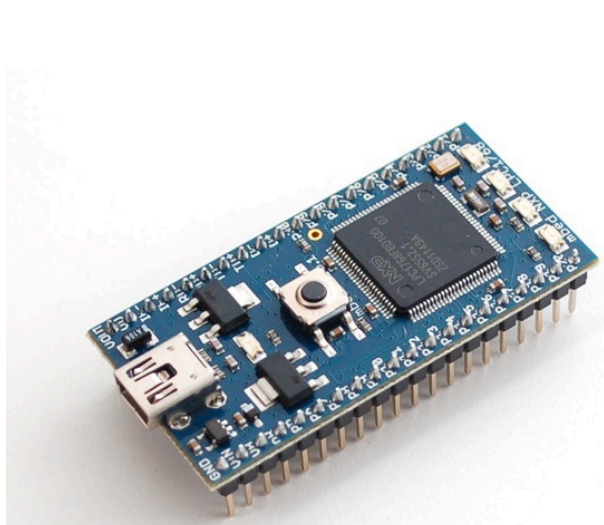
Processors use the 32-bit RISC architecture
http://www.arm.com/products/processors/cortex-m/index.php

Prototyping Boards used in Performance Tests

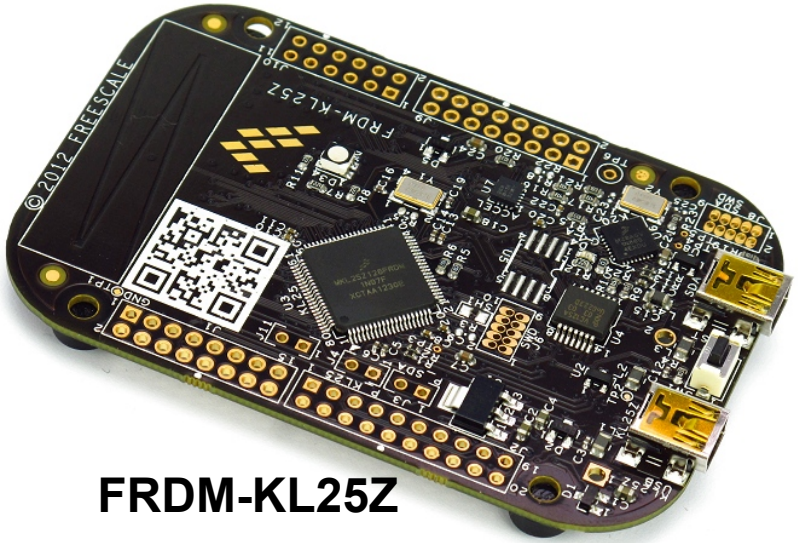
- **ST Nucleo F401RE** (STM32F401RET6)
  - ARM Cortex-M4 CPU with FPU at 84MHz
  - 512KB Flash, 96KB SRAM
- **ST Nucleo F103** (STM32F103RBT6)
  - ARM Cortex-M4 CPU with FPU at 72MHz
  - 128KB Flash, 20KB SRAM
- **ST Nucleo L152RE** (STM32L152RET6)
  - ARM Cortex-M3 CPU at 32MHz
  - 512 KBytes Flash, 80KB RAM
- **ST Nucleo F091** (STM32F091RCT6)
  - ARM Cortex-M0 CPU at 48MHz
  - 256 KBytes Flash, 32KB RAM
- **NXP LPC1768**
  - ARM Cortex-M3 CPU at 96MHz
  - 512KB Flash, 32KB RAM
- **Freescale FRDM-KL25Z**
  - ARM Cortex-M0+ CPU at 48MHz
  - 128KB Flash, 16KB RAM



ST Nucleo



LPC1768



FRDM-KL25Z

ECC Curves

- NIST curves: secp521r1, secp384r1, secp256r1, secp224r1, secp192r1
- "Koblitz curves": secp256k1, secp224k1, secp192k1
- Brainpool curves: brainpoolP512r1, brainpoolP384r1, brainpoolP256r1
- Curve25519 (only preliminary results).

- Note that FIPS186-4 refers to secp192r1 as P-192, secp224r1 as P-224, secp256r1 as P-256, secp384r1 as P-384, and secp521r1 as P-521.

Optimizations

- NIST Optimization
  - Utilizes special structure of NIST chosen curves.
  - Appendix 1 of http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf
  - Longer version in FIPS PUB 186-4:
  - http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
  - Relevant configuration parameter: POLARSSL_ECP_NIST_OPTIM
- Fixed Point Optimization:
  - Pre-computes points
  - Described in https://eprint.iacr.org/2004/342.pdf
  - Relevant configuration parameter: POLARSSL_ECP_FIXED_POINT_OPTIM
- Window:
  - Technique for more efficient exponentation
  - Sliding window technique described in https://en.wikipedia.org/wiki/Exponentiation_by_squaring
  - Relevant configuration parameter: POLARSSL_ECP_WINDOW_SIZE (min=2, max=7).

ARM

ECDSA, ECDHE, and ECDH

- Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve variant of the Digital Signature Algorithm (DSA) or, as it is sometimes called, the Digital Signature Standard (DSS).
- It is used in TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 ciphersuite recommended in CoAP (and consequently also in the DTLS profile draft).
- ECDSA, like DSA, has the property that poor randomness used during signature generation can compromise the long-term signing key.
  - For this reason the deterministic variant of (EC)DSA (RFC 6979) is implemented, which uses the private key as a source or "entropy" to seed a PRNG.
  - Note: None of the prototyping boards listed in the slide deck provide true random number generation.
- CoAP recommends this ciphersuite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 that makes use of the Ephemeral Elliptic Curve Diffie-Hellman (ECDHE).
  - The Elliptic Curve Diffie-Hellman (ECDH) is only used for comparison purposes in this slide deck but not used in the recommended ciphersuites.

Key Length

- Tradeoff between security and performance.
- Values based on recommendations from RFC 4492.
- [I-D.ietf-uta-tls-bcp] recommends at least 112 bits symmetric keys.
- A 2013 ENISA report states that an 80bit symmetric key is sufficient for legacy applications but recommends 128 bits for new systems.

Symmetric	ECC	DH/DSA/RSA
80	163	1024
112	233	2048
128	283	3072
192	409	7680
256	571	15360

Observations: Performance Figures

- ECDSA signature operation is faster than ECDSA verify operation.
- Brainpool curves are slower than NIST curves because Brainpool curves use random primes.
- ECC key sizes above 256 bits are substantially slower than ECC curves with key size 192, 224, and 256.
- ECDH is only slightly faster than ECDHE (when fixed point optimization is enabled).
- CPU speed has a significant impact on the performance.
- The performance of symmetric key cryptography (keyed hash functions, encryption functions) is neglectable.

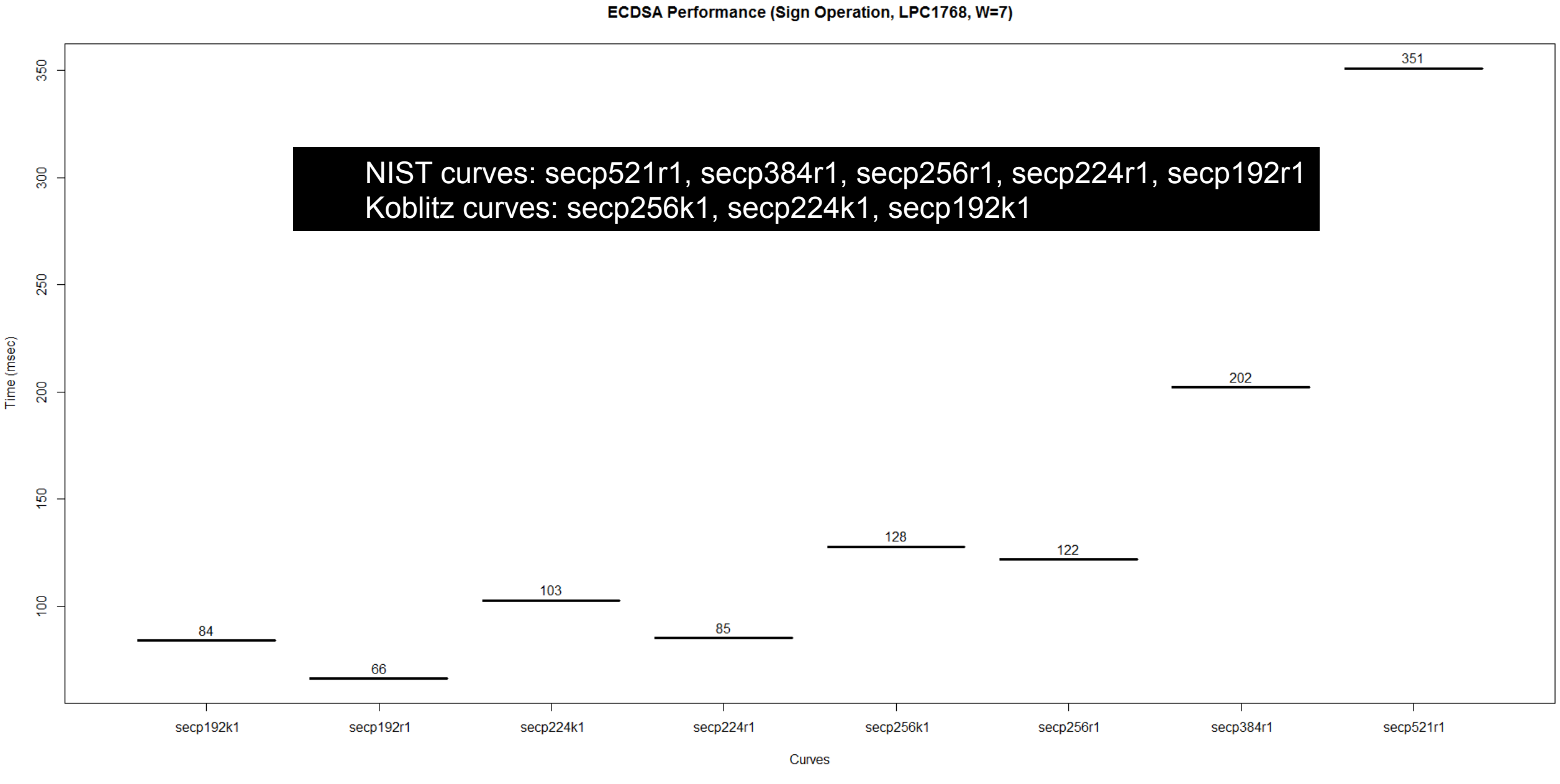
Observations: Optimizations

- NIST curve optimization provides substantial benefit for NIST secp*r1 curves.
- Fixed point optimization has a significant influence on the performance.
- There is a performance – RAM usage tradeoff: increased performance comes at the expense of additional RAM usage.
- ECC library increases code size but also requires a fair amount of RAM for optimizations (for most curves).

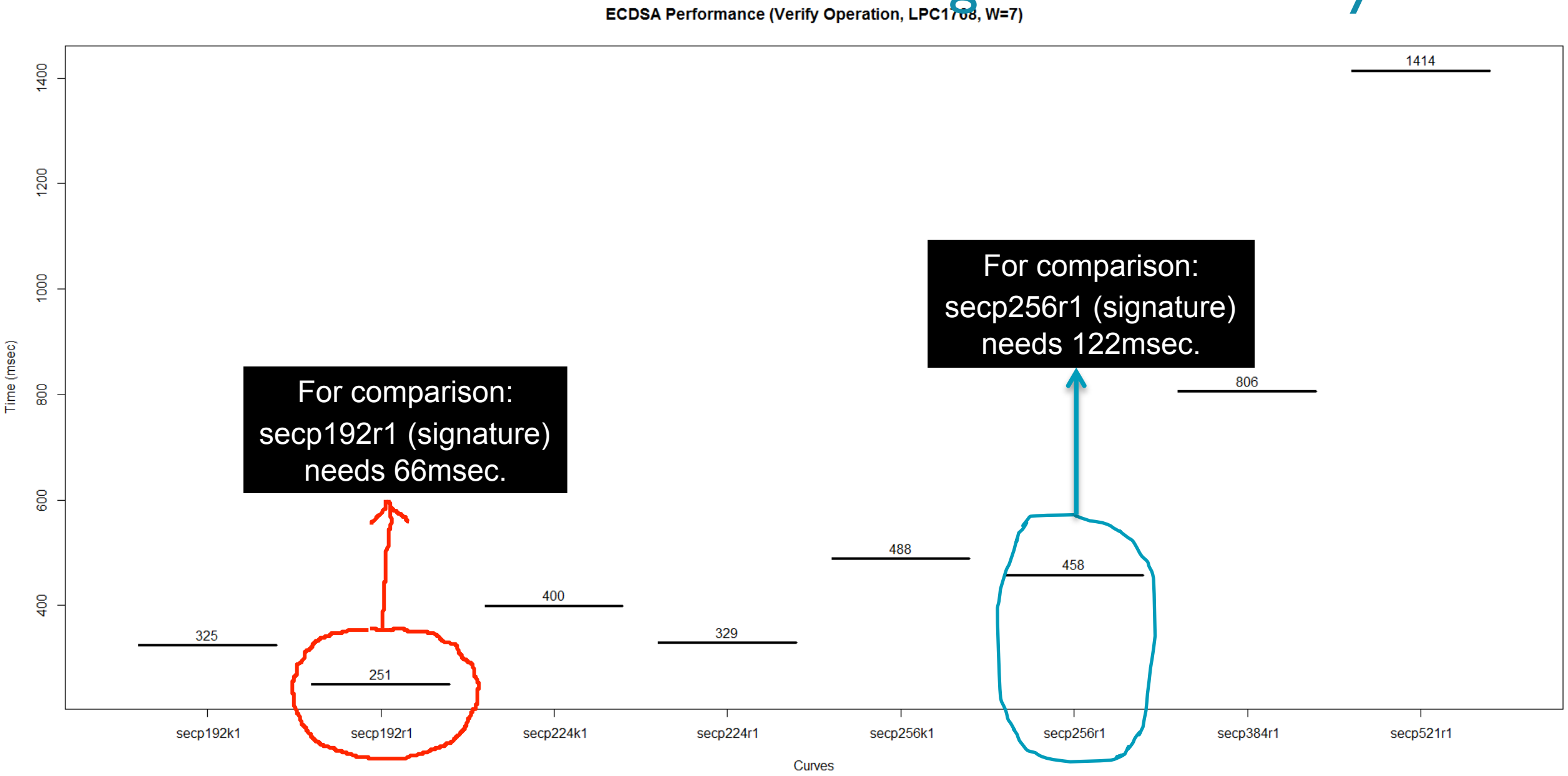
ECC Performance of the Cortex M3/M4

ARM

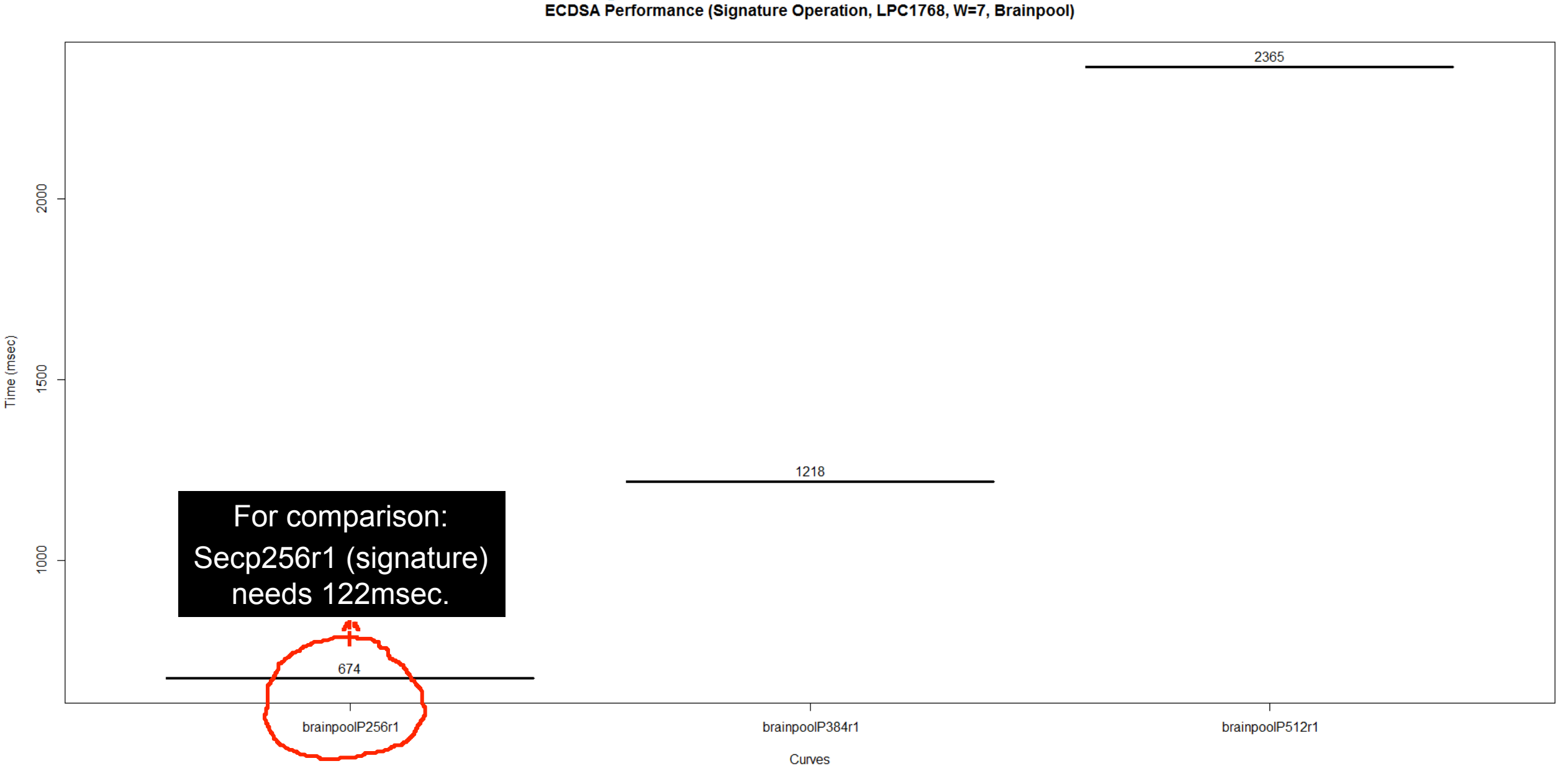
Performance of various NIST/Koblitz ECC Curves



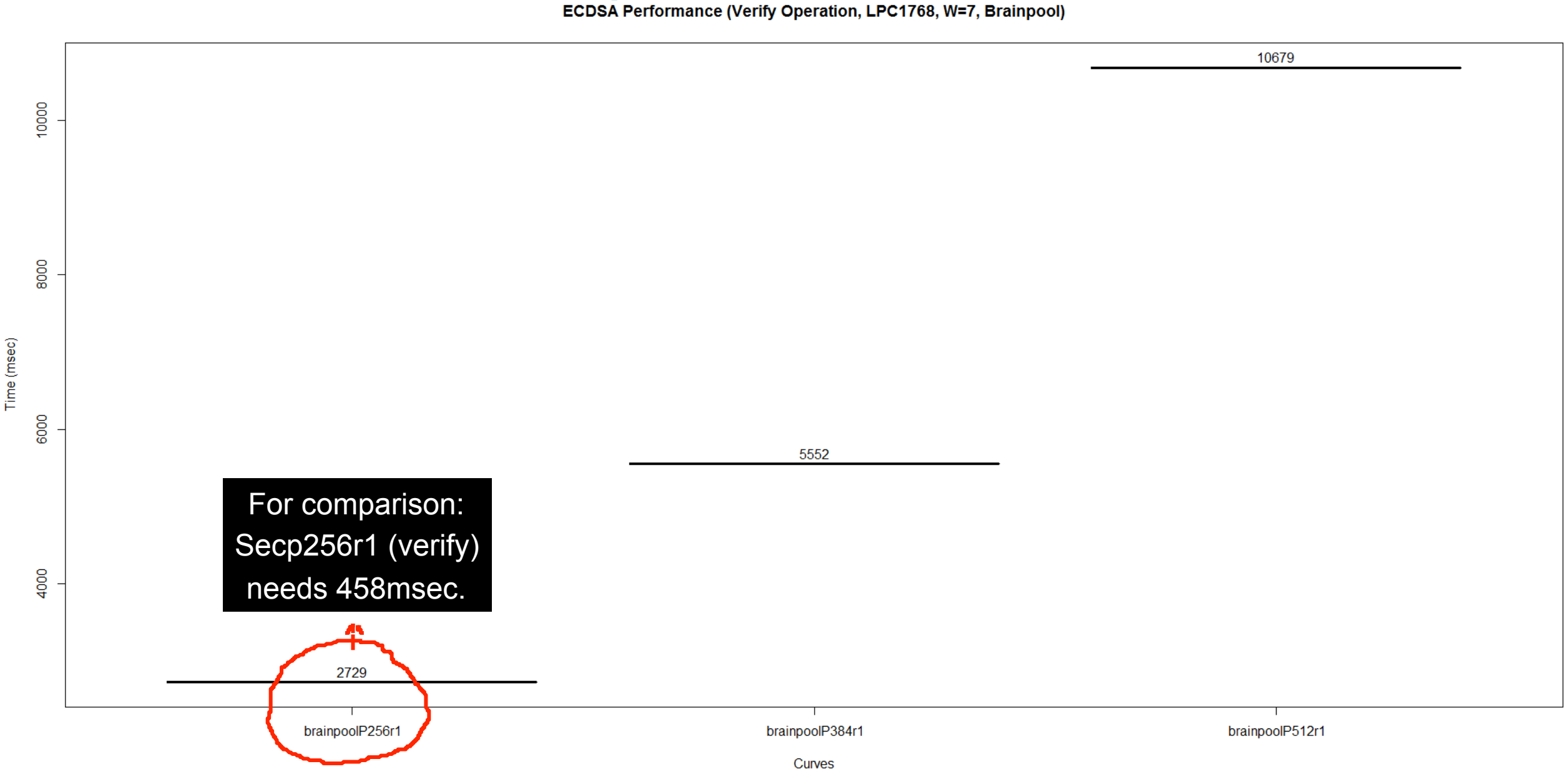
Performance difference between signature vs. verify



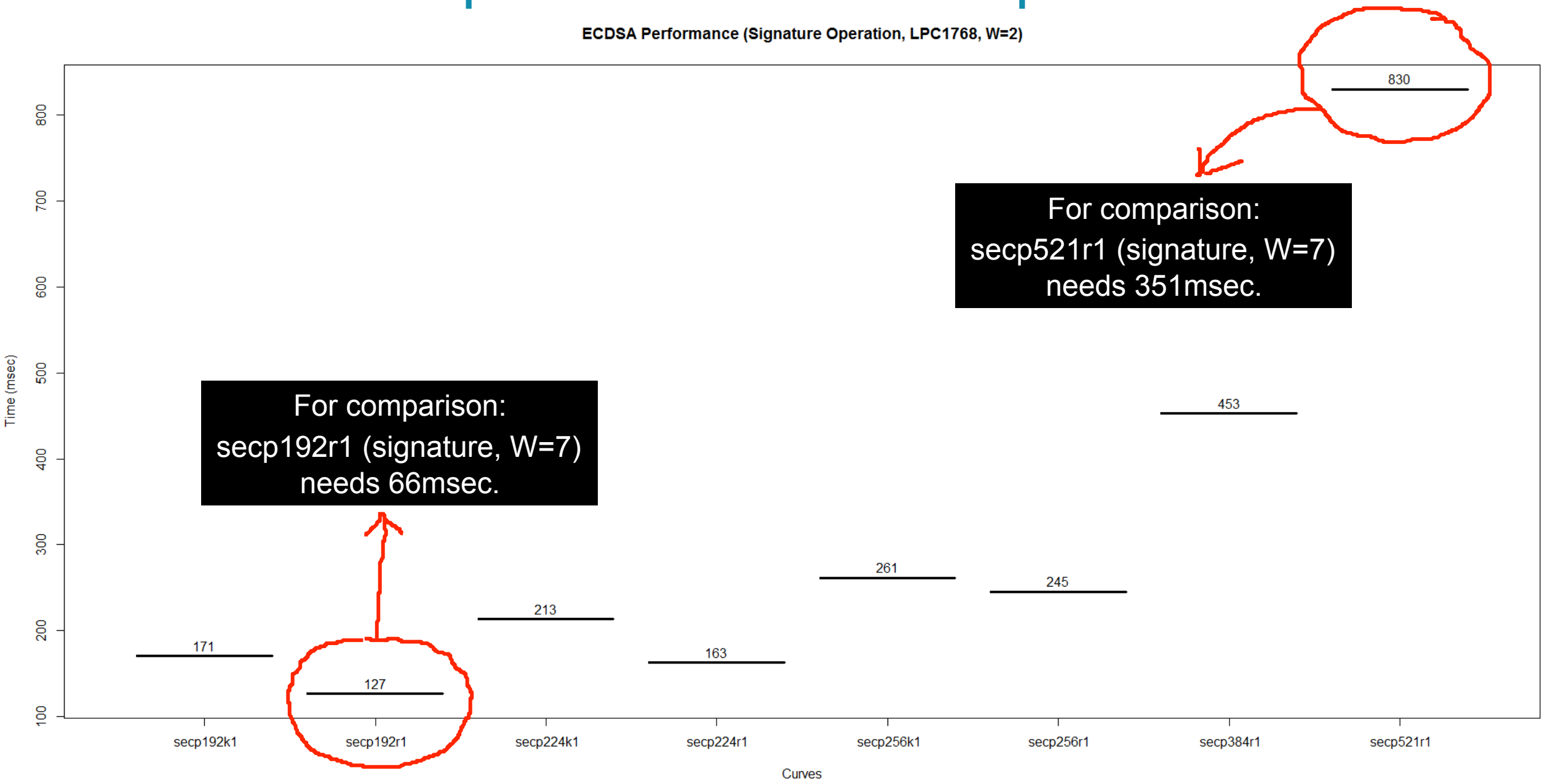
Performance of Brainpool Curves



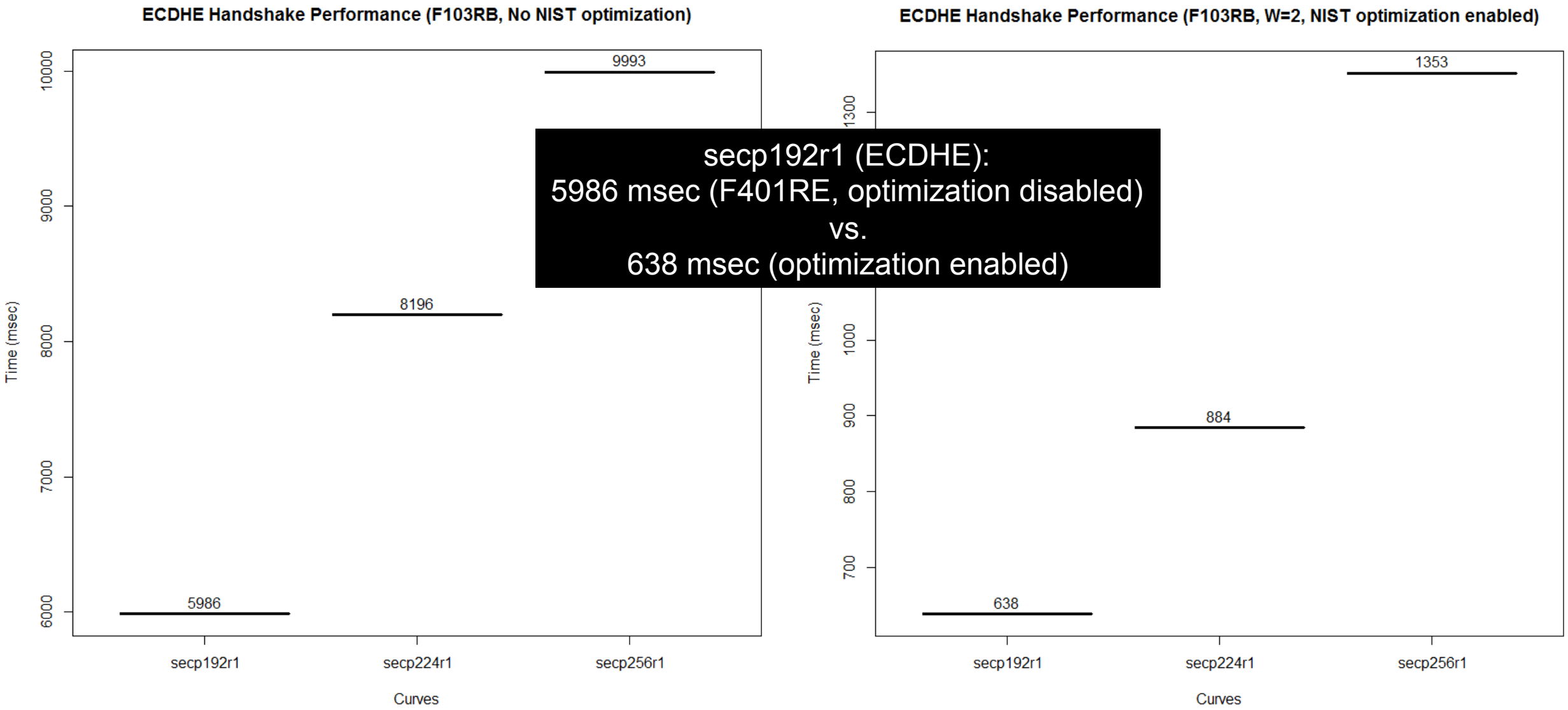
Performance of Brainpool Curves



Performance impact of the "window" parameter



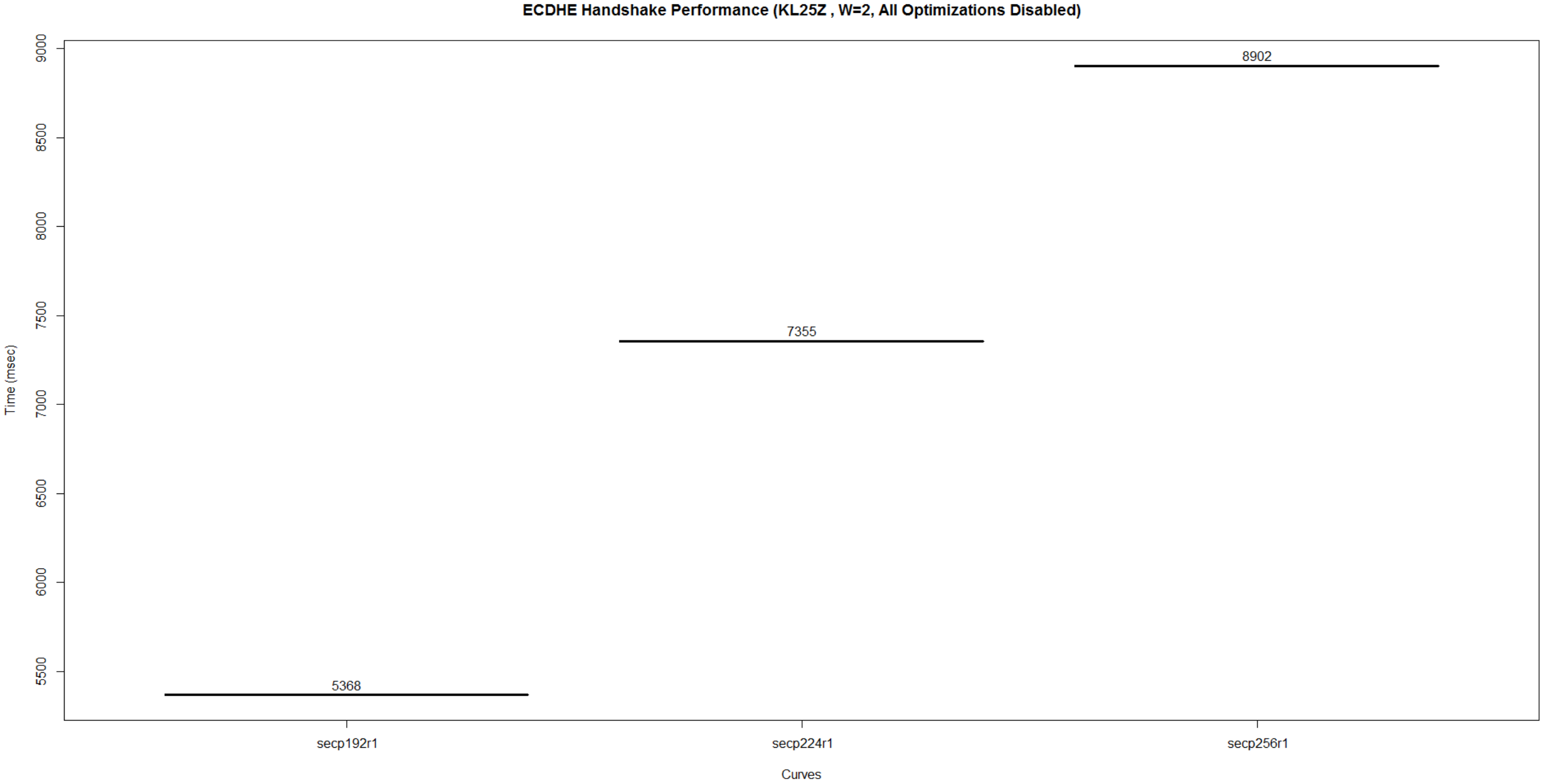
The Performance Impact of the NIST Optimization



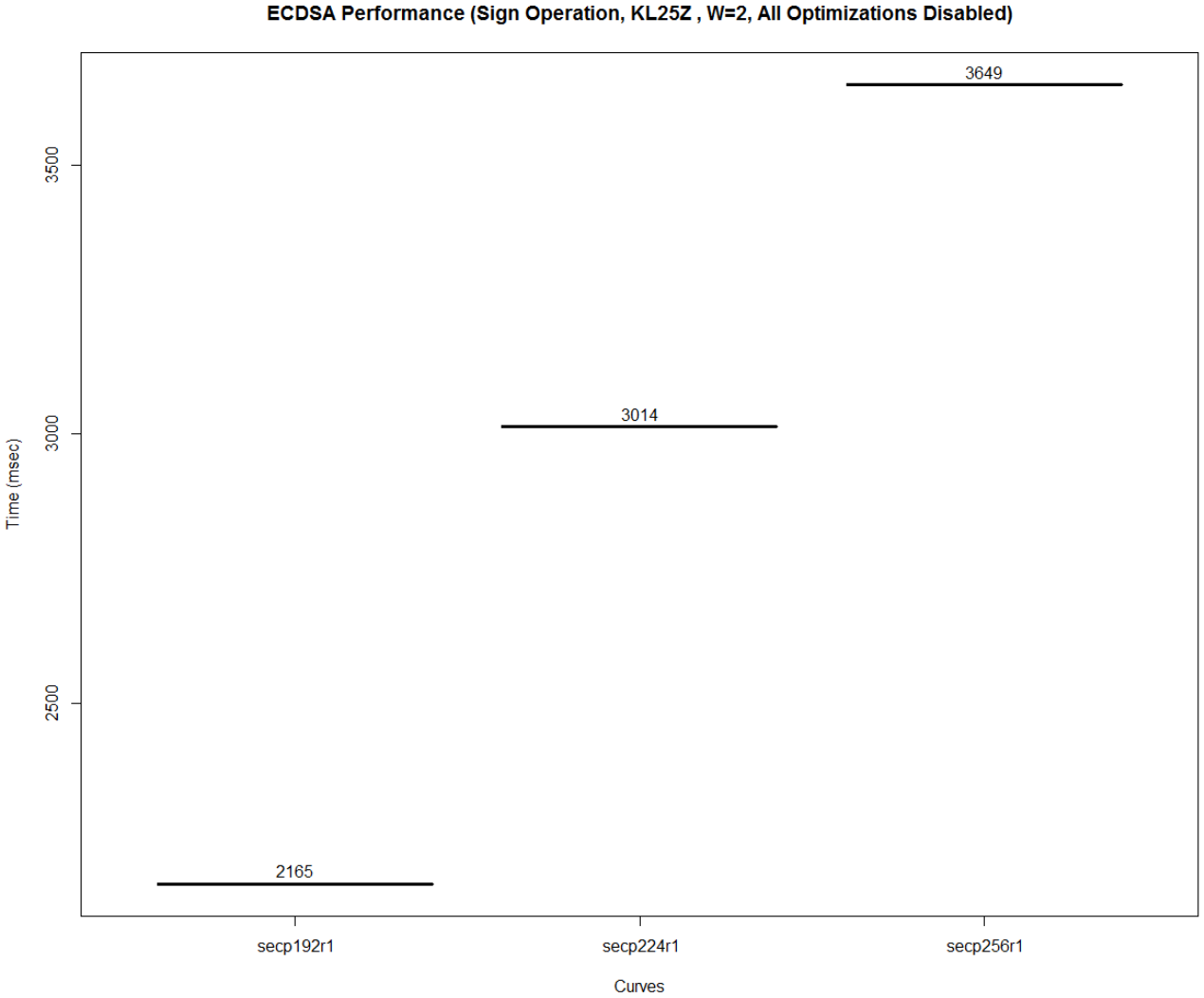
ECC Performance of the Cortex M0/M0+

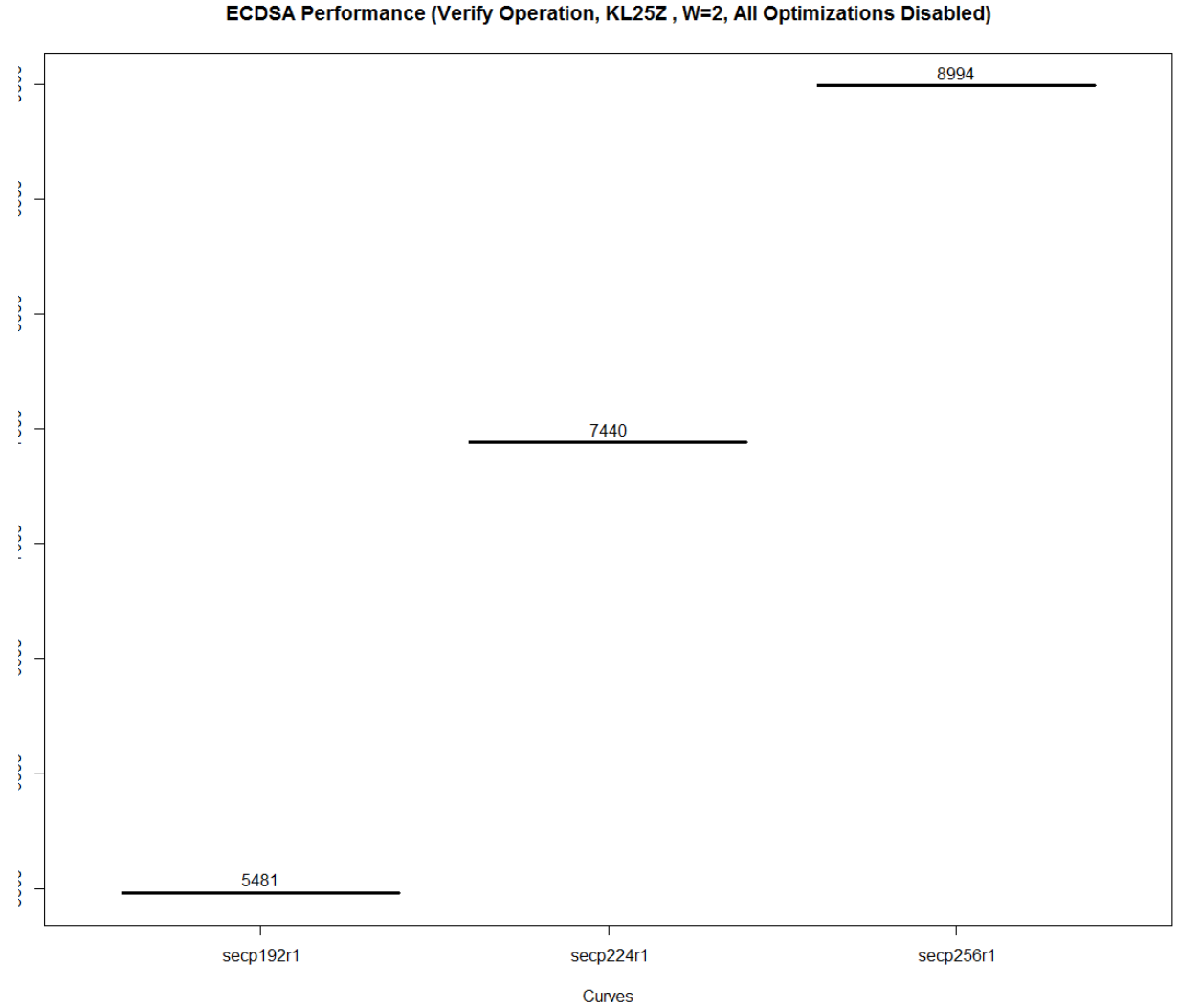
ARM

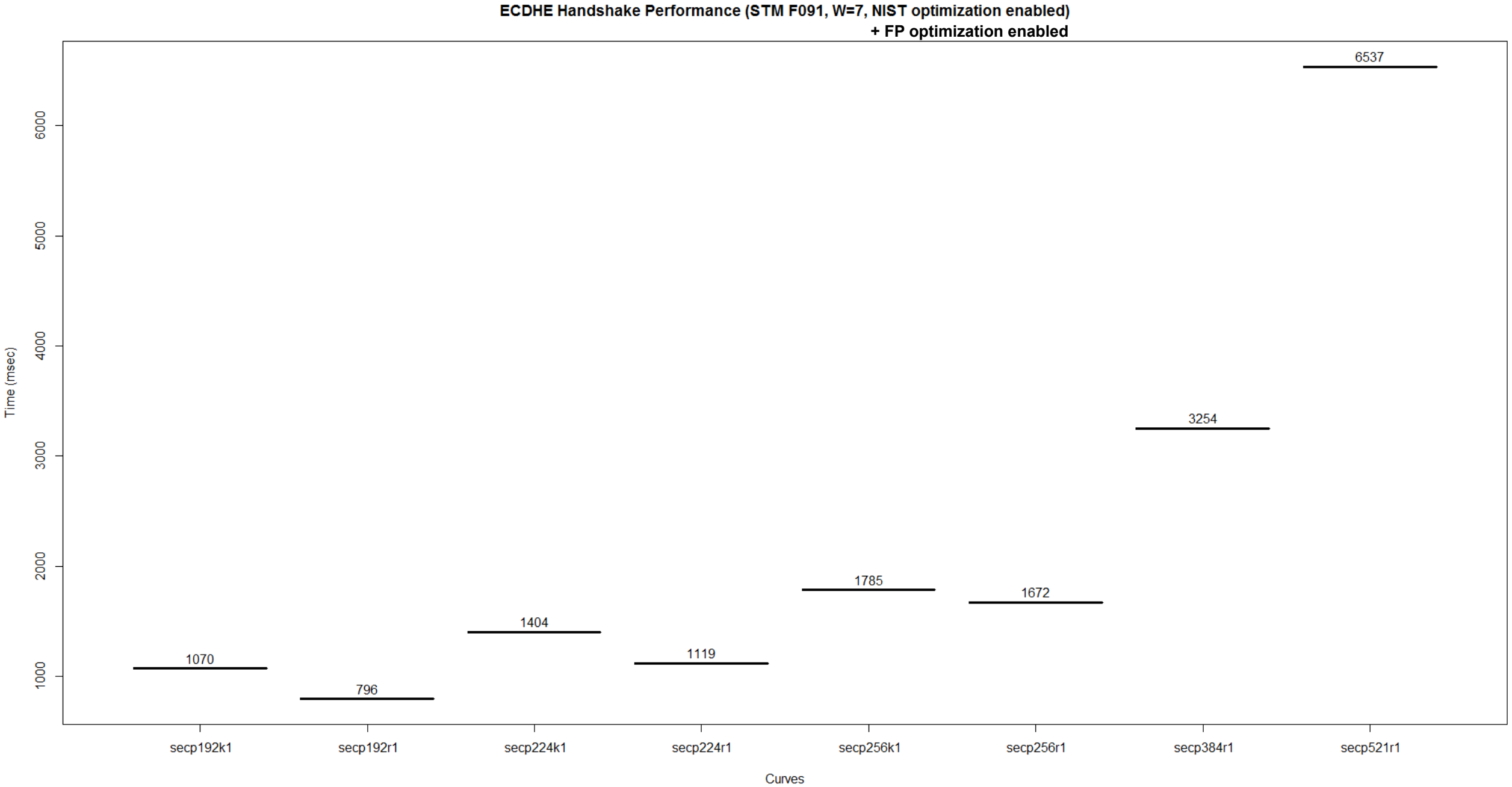
ECDHE Performance of the KL25Z

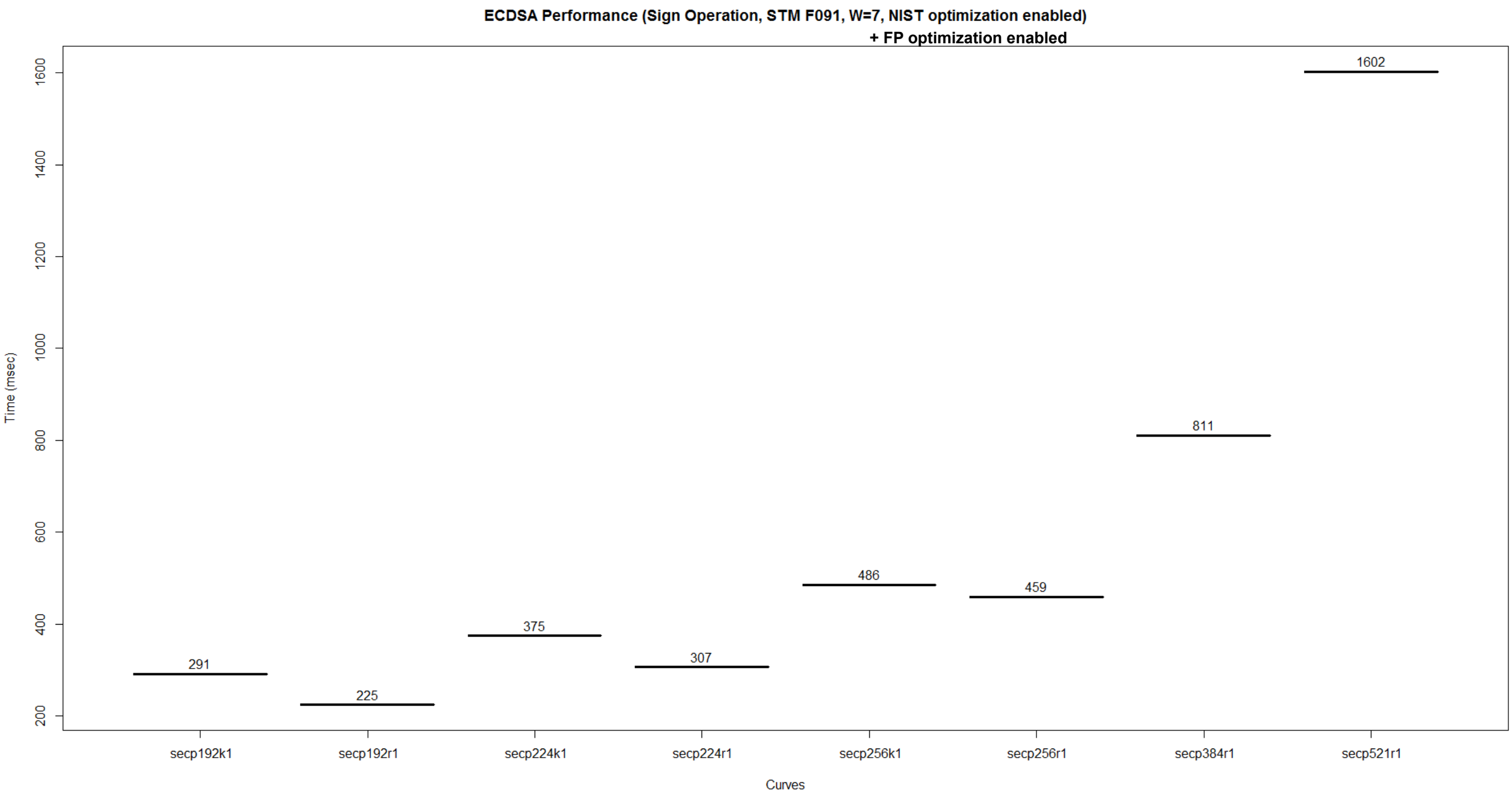


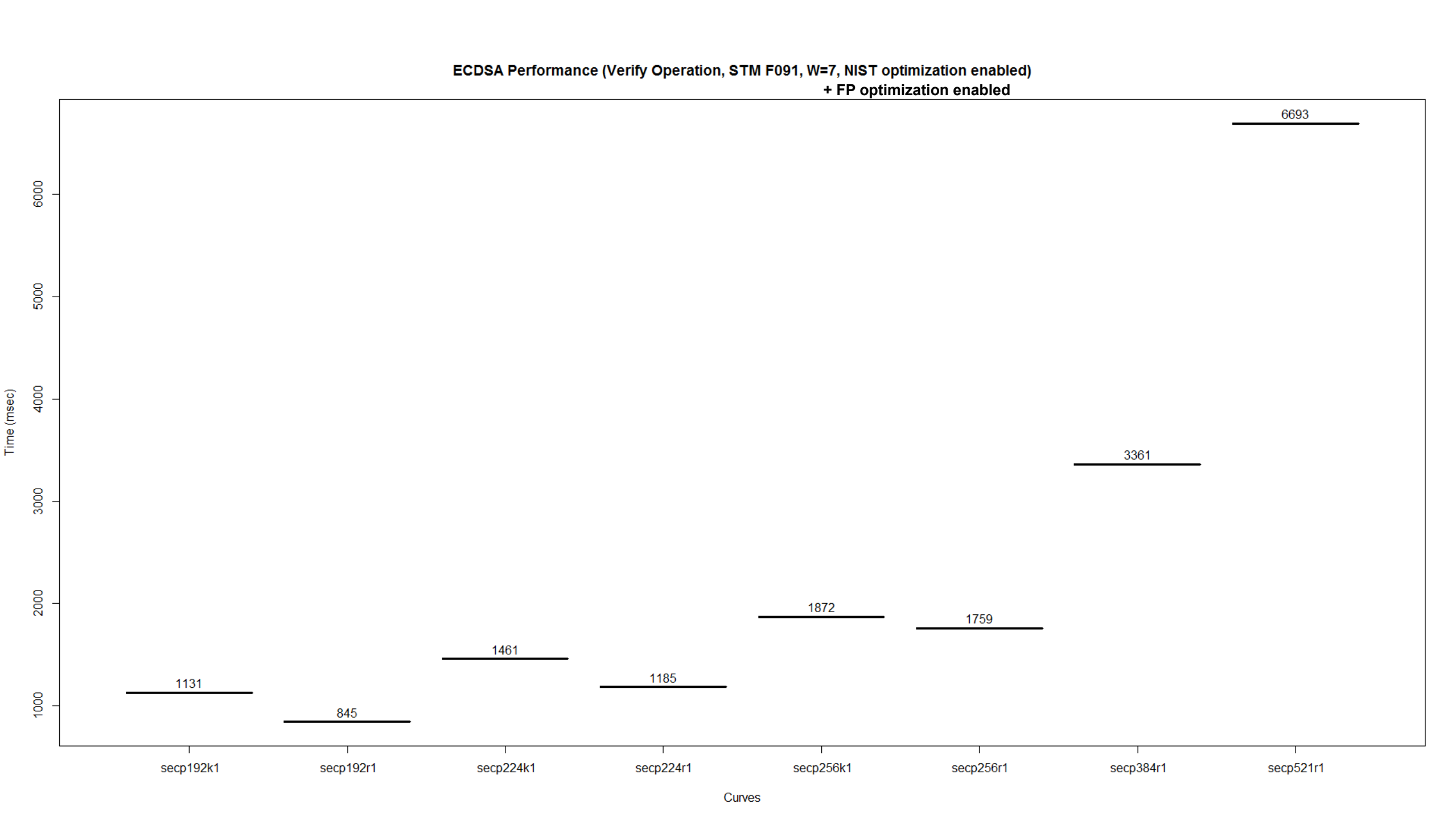
ECDSA Performance of the KL25Z






ECDHE Handshake Performance (STM F091, W=7, NIST optimization enabled)
+ FP optimization enabled
Time (msec)
1000
2000
3000
4000
5000
6000
1070
796
1404
1119
1785
1672
3254
6537
secp192k1
secp192r1
secp224k1
secp224r1
secp256k1
secp256r1
secp384r1
secp521r1
Curves


ECDSA Performance (Sign Operation, STM F091, W=7, NIST optimization enabled)
+ FP optimization enabled
Time (msec)
200
400
600
800
1000
1200
1400
1600
291
225
375
307
486
459
811
1602
secp192k1
secp192r1
secp224k1
secp224r1
secp256k1
secp256r1
secp384r1
secp521r1
Curves


ECDSA Performance (Verify Operation, STM F091, W=7, NIST optimization enabled)
+ FP optimization enabled
Time (msec)
1000
2000
3000
4000
5000
6000
1131
845
1461
1185
1872
1759
3361
6693
secp192k1
secp192r1
secp224k1
secp224r1
secp256k1
secp256r1
secp384r1
secp521r1
Curves

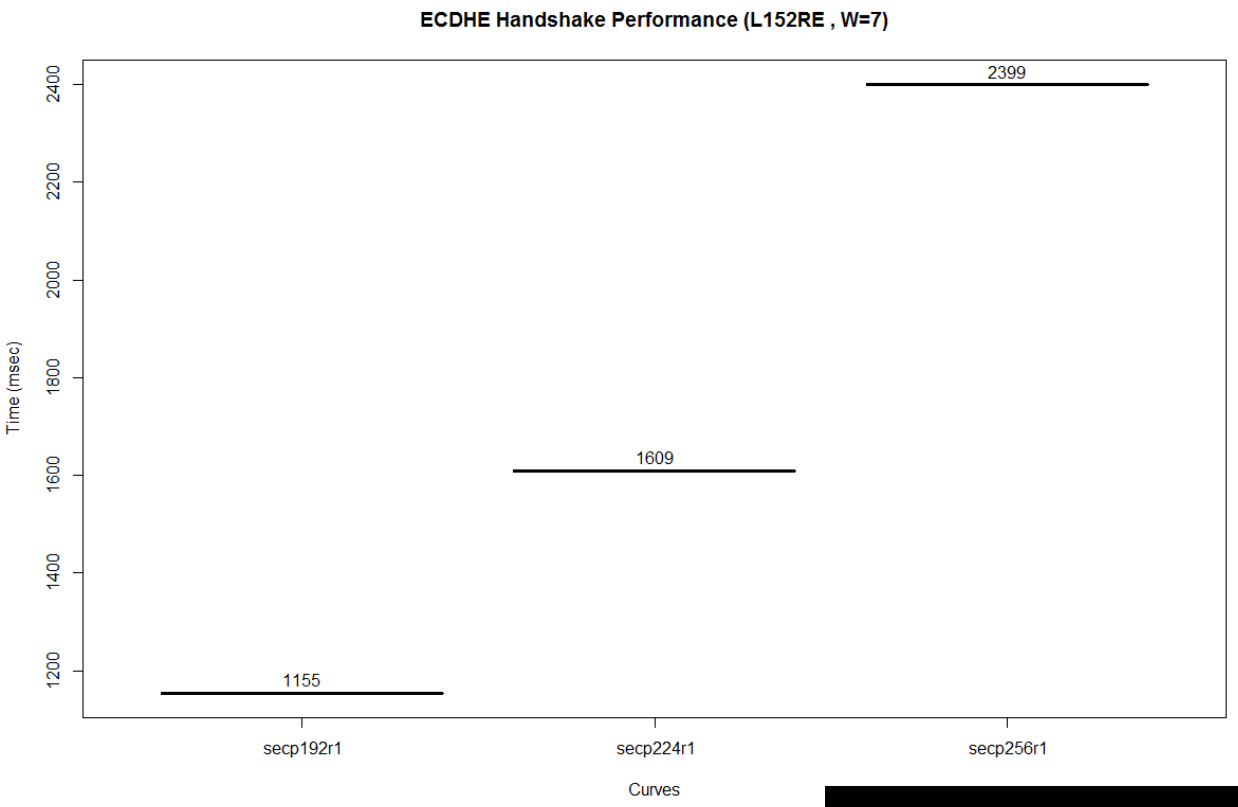
CPU Speed Impact

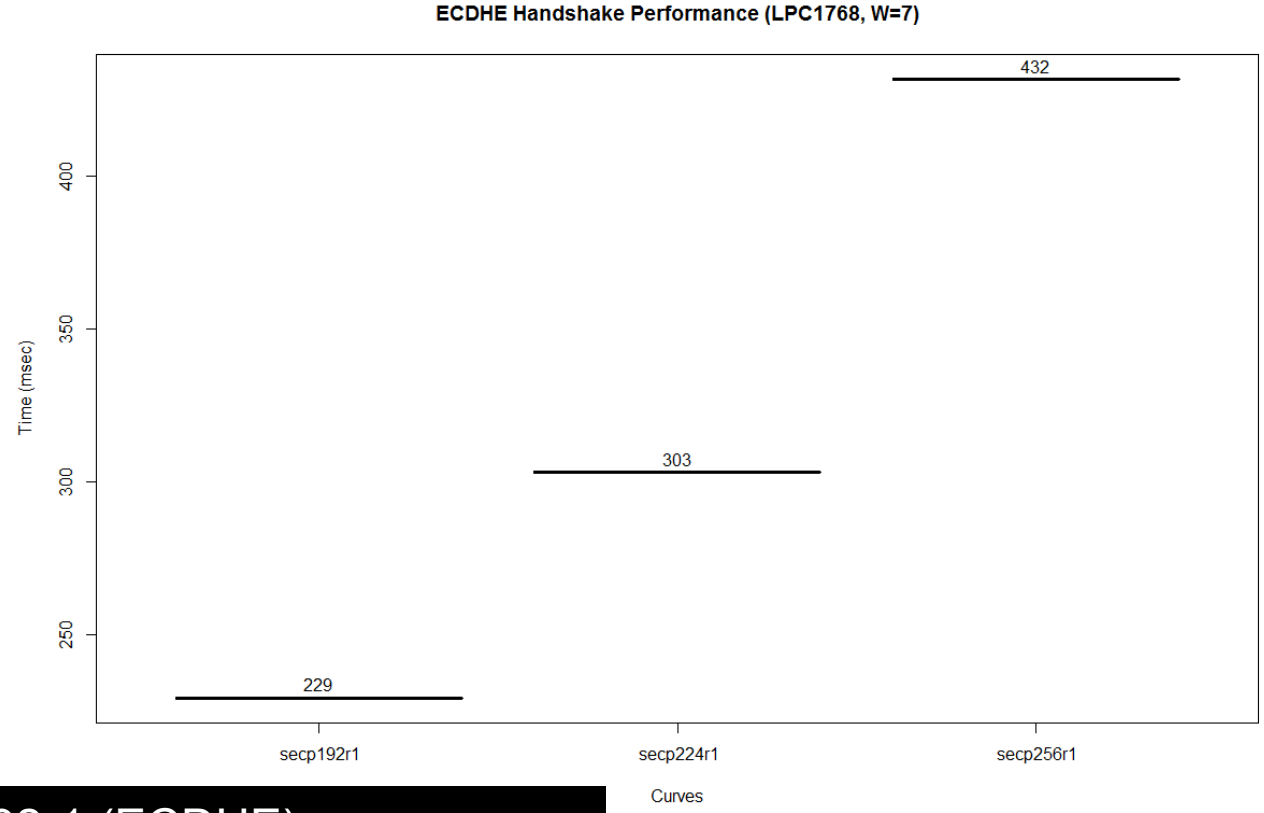
ARM

Performance of ECDHE: L152RE vs. LPC1768

L152RE:
Cortex-M3 with 32MHz

LPC1768:
Cortex-M3 with 96MHz



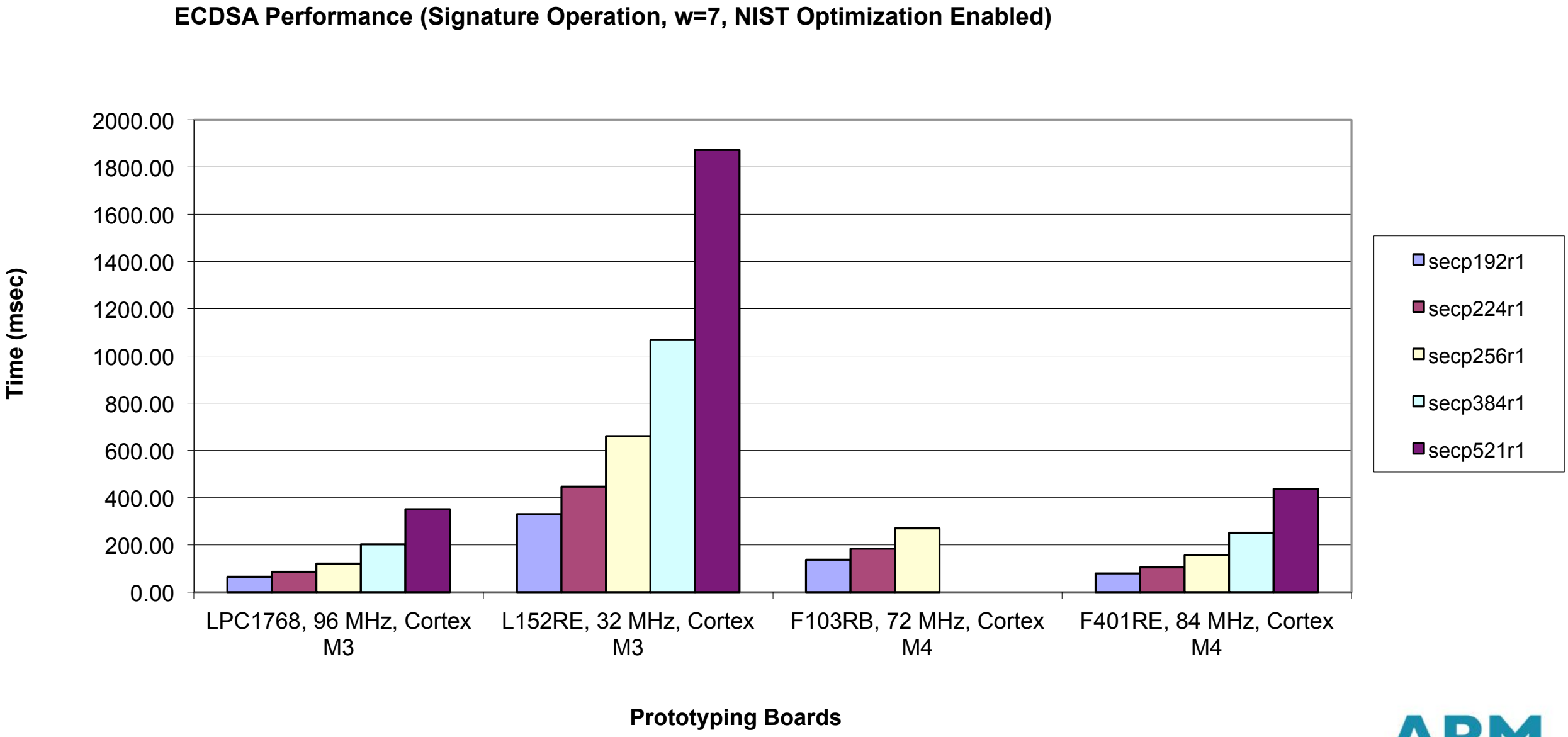


secp192r1 (ECDHE):
1155 msec (L152RE) vs. 229 msec (LPC1768)

NIST optimization enabled.
Fixed-point speed-up enabled.

ARM

Performance Comparison: Prototyping Boards



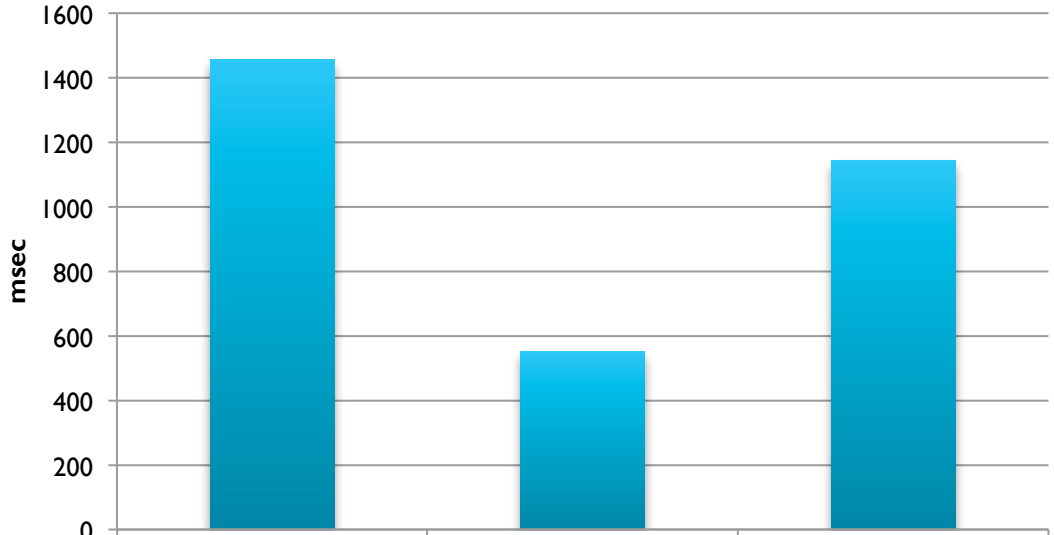
ARM

Curve25519

(Warning: Preliminary Results)

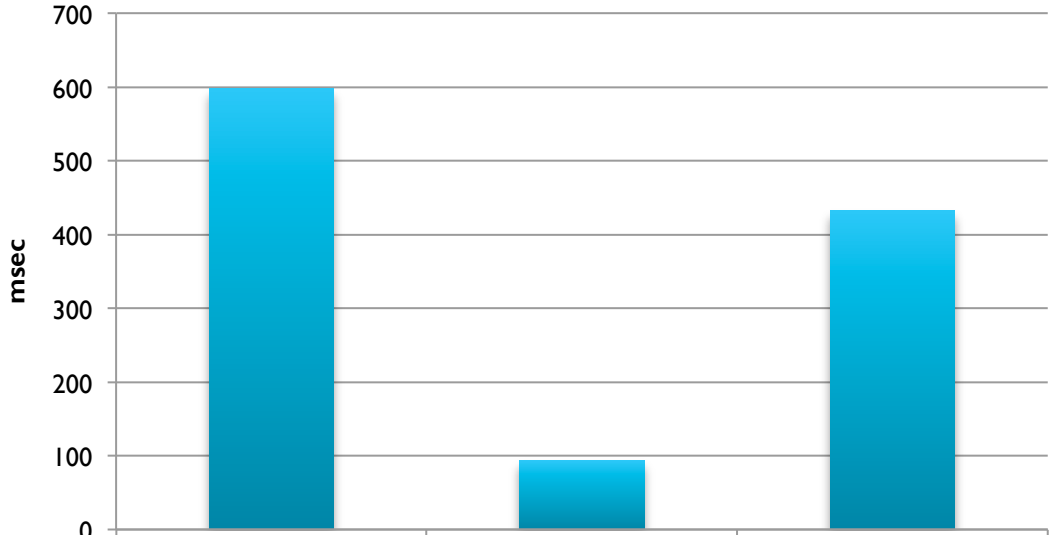
ARM

FRDM-KL46Z (Cortex-M0+, 48 MHz)



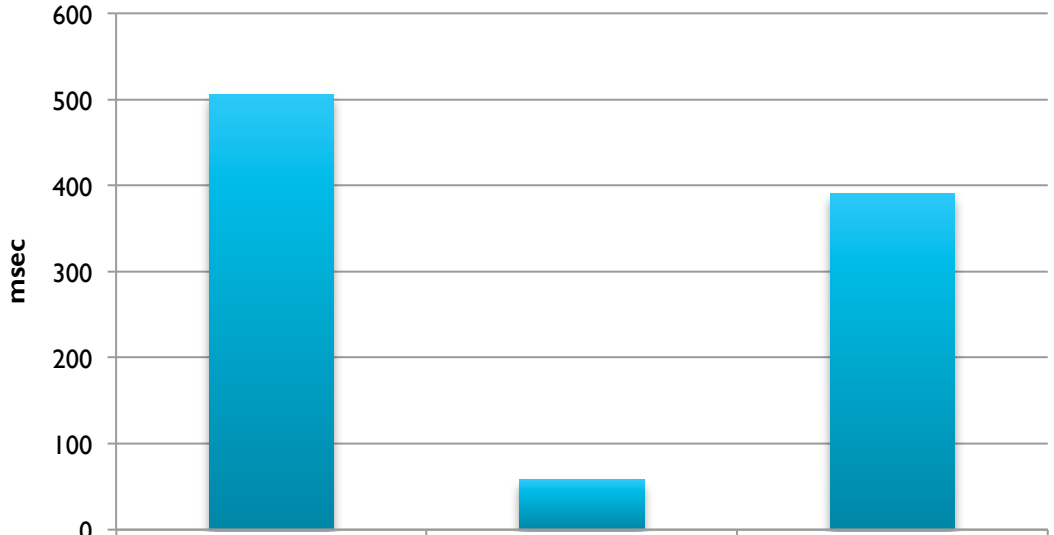
	Curve25519-mbedtls	Curve25519-donna	P256-mbed
ECDHE	1458	552	1145

LPC1768 (Cortex-M3, 96 MHz)



	Curve25519-mbedtls	Curve25519-donna	P256-mbed
ECDHE	598	94	432

FRDM-K64F (Cortex-M4, 120 MHz)



	Curve25519-mbedtls	Curve25519-donna	P256-mbed
ECDHE	506	58	391

Notes:

- The Curve25519-mbedtls implementation uses a generic libary. Hence, the special properties of Curve25519 are not utilized.
- Curve25519 has very low RAM requirements (~1 Kbyte only).
- Curve25519-donna is based on the Google implementation. Improvements for M0/M0+ are likely since the code has not been tailored to the architecture.
- Question: Is Curve25519 a way to get ECC on M0/M0+?

The Power of Assembly Optimizations

- Example: micro-ecc library
  - https://github.com/kmackay/micro-ecc/tree/old
  - Written in C, with optional inline assembly for ARM and Thumb platforms.
  - LPC1114 at 48MHz (ARM Cortex-M0)

ECDH time (ms)	secp192r1	secp256r1
LPC1114	175.7	465.1
STM32F091	604,55	1260.9

ECDSA verify time (ms)	secp192r1	Secp256r1
LPC1114	217.1	555.2
STM32F091	845.5	1758.8

- Performance improvement between 200 and 300 %

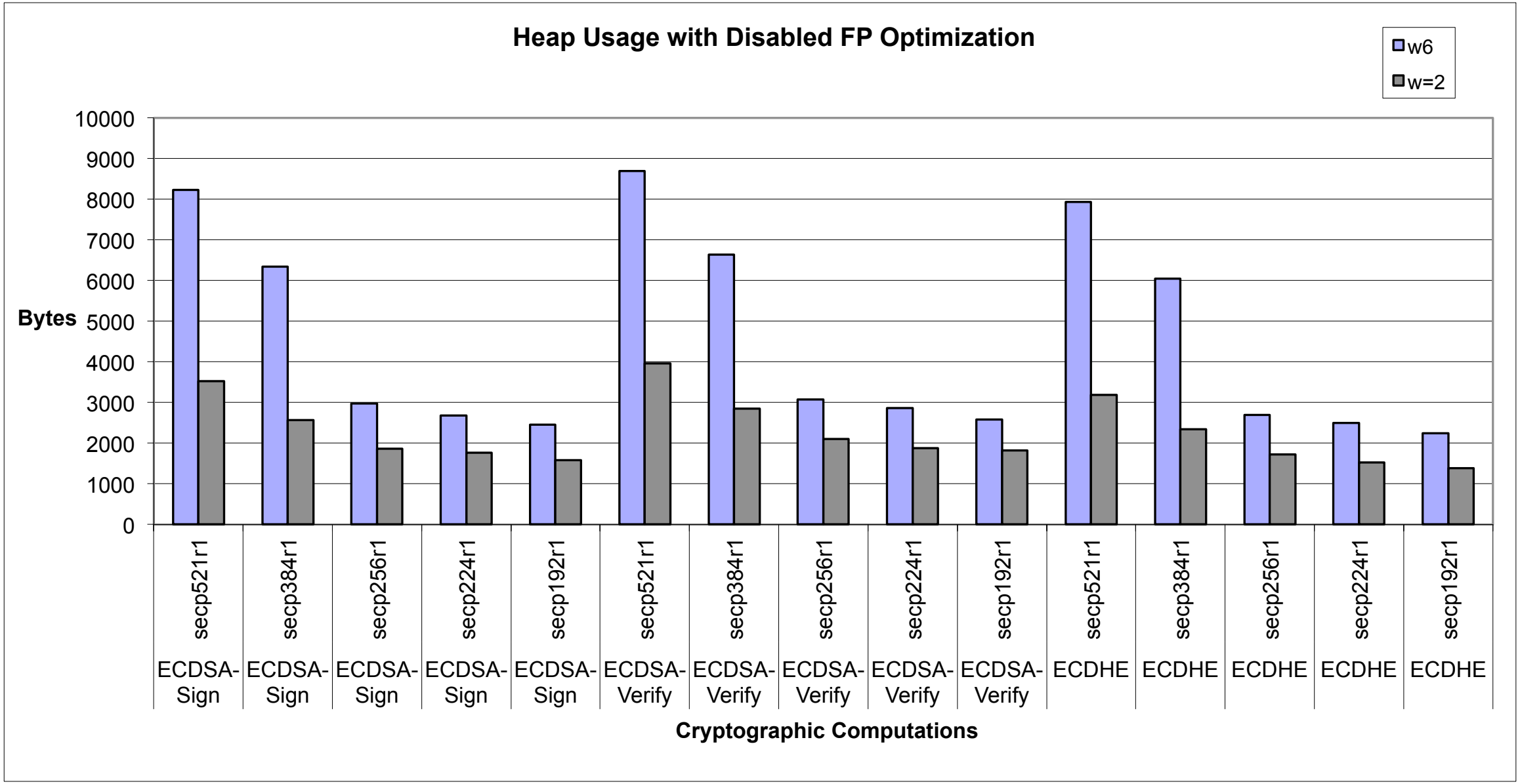
ARM

RAM Usage

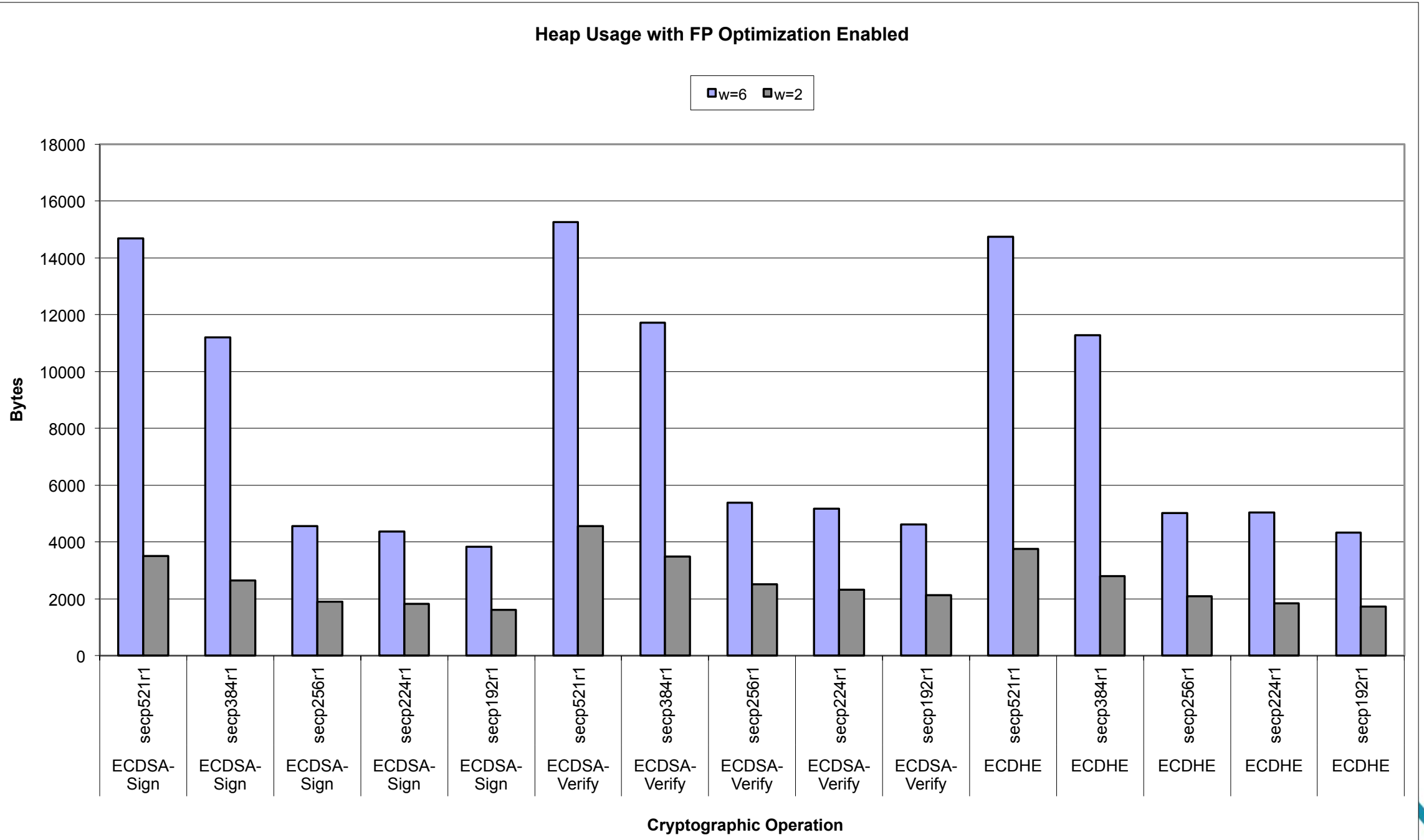
ARM

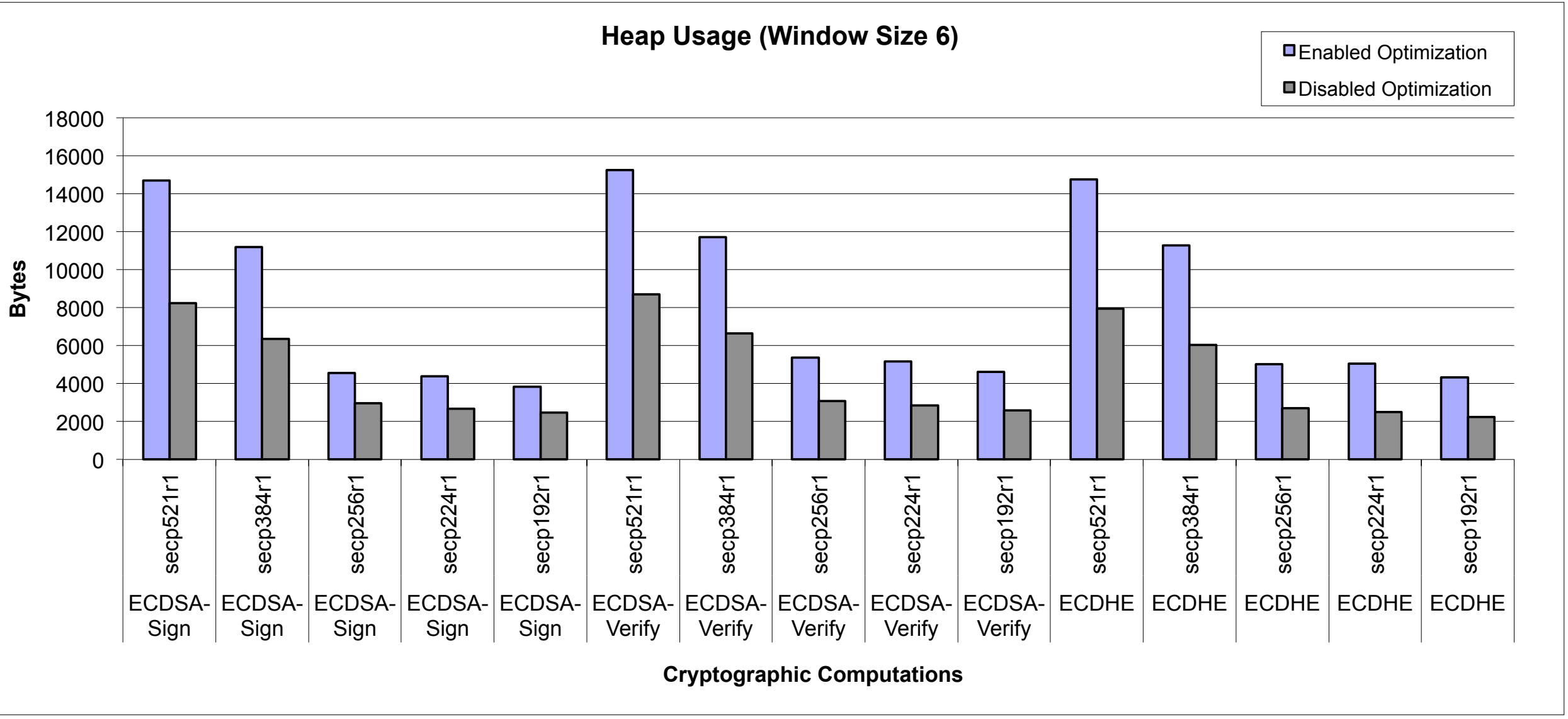
What was measured?

- Heap using a custom memory allocation handler (instead of malloc).
- Memory allocated on the stack was not measured (but it is negligible).
- Measurement was done on a Linux PC (rather than on the embedded device itself for convenience reasons).
- Two aspects investigated:
  - Memory impact caused by different window parameter changes.
  - Memory impact caused by FP performance optimization.


Heap Usage with Disabled FP Optimization
w6
w=2
Bytes
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
secp521r1
secp384r1
secp256r1
secp224r1
secp192r1
secp521r1
secp384r1
secp256r1
secp224r1
secp192r1
secp521r1
secp384r1
secp256r1
secp224r1
secp192r1
ECDSA-Sign
ECDSA-Sign
ECDSA-Sign
ECDSA-Sign
ECDSA-Sign
ECDSA-Verify
ECDSA-Verify
ECDSA-Verify
ECDSA-Verify
ECDSA-Verify
ECDHE
ECDHE
ECDHE
ECDHE
ECDHE
Cryptographic Computations

ARM


Heap Usage with FP Optimization Enabled
w=6
w=2
Bytes
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
secp521r1
secp384r1
secp256r1
secp224r1
secp192r1
secp521r1
secp384r1
secp256r1
secp224r1
secp192r1
secp521r1
secp384r1
secp256r1
secp224r1
secp192r1
ECDSA-Sign
ECDSA-Sign
ECDSA-Sign
ECDSA-Sign
ECDSA-Sign
ECDSA-Verify
ECDSA-Verify
ECDSA-Verify
ECDSA-Verify
ECDSA-Verify
ECDHE
ECDHE
ECDHE
ECDHE
ECDHE
Cryptographic Operation

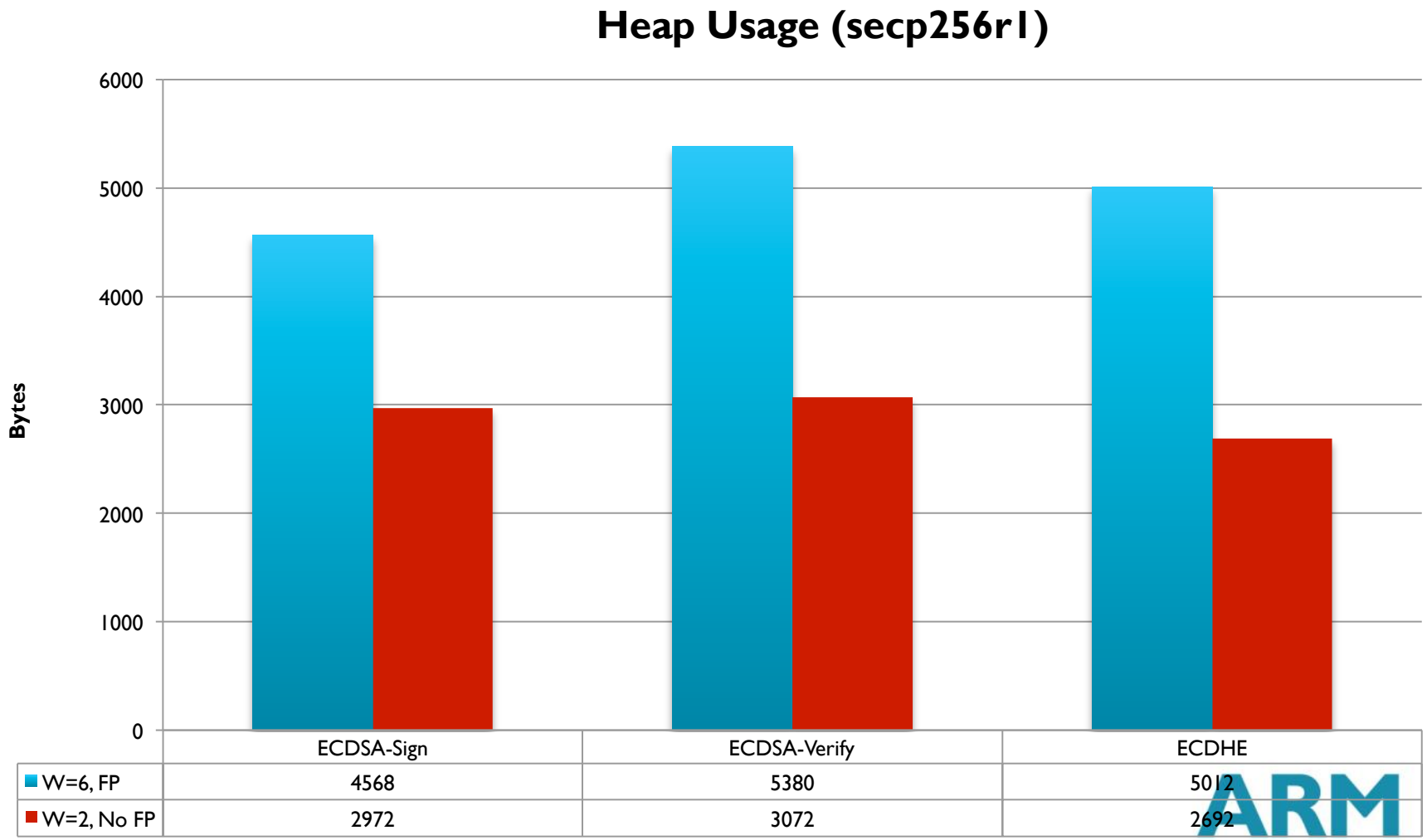


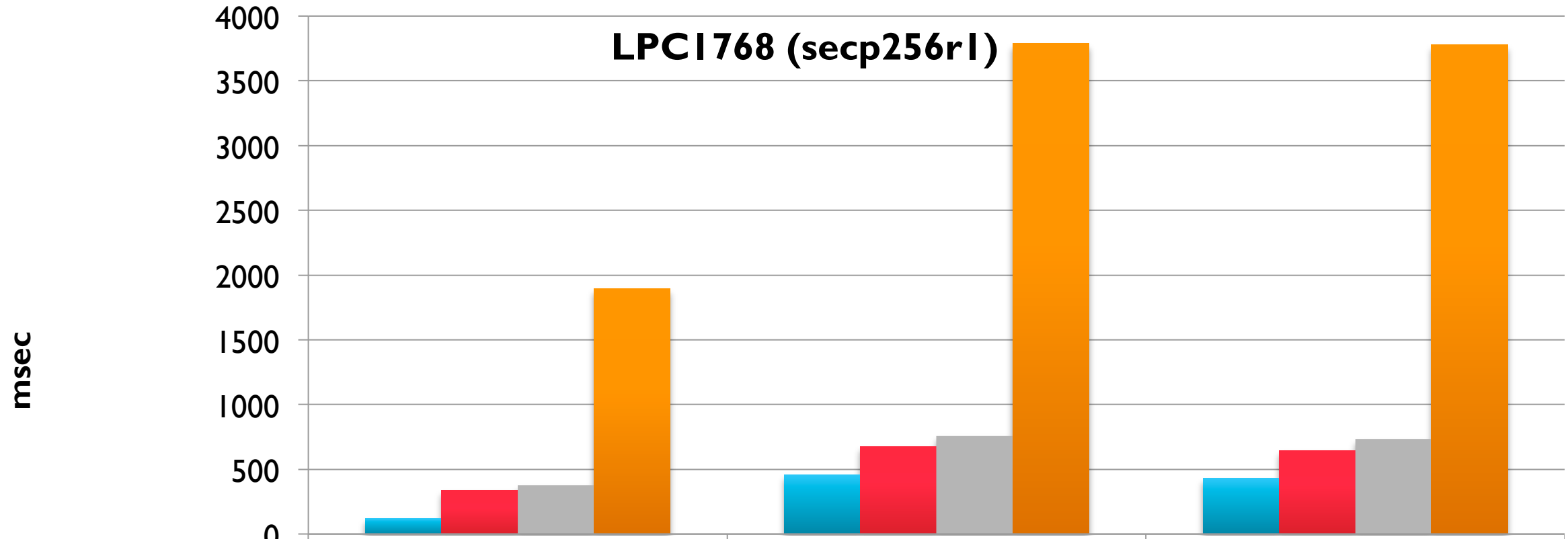
Note: NIST optimization enabled in both cases since it does not have an impact on the heap usage.

ARM

Summary

- To enable certain optimizations sufficient RAM is needed. A tradeoff decision between RAM and speed.
- Optimizations pays off.
- This slide shows heap usage (NIST optimization enabled).





	ECDSA-Sign	ECDSA-Verify	ECDHE
w=6, FP, NIST	122	458	431
w=6, no FP, NIST	340	677	644
w=2, no FP, NIST	378	759	734
w=2, no FP, no NIST	1893	3788	3781

Using ~50 % more RAM increases the performance by a factor 8 or more.

ARM

Applying Results to TLS/DTLS

ARM

Raw Public Keys with TLS_ECDHE_ECDSA_*

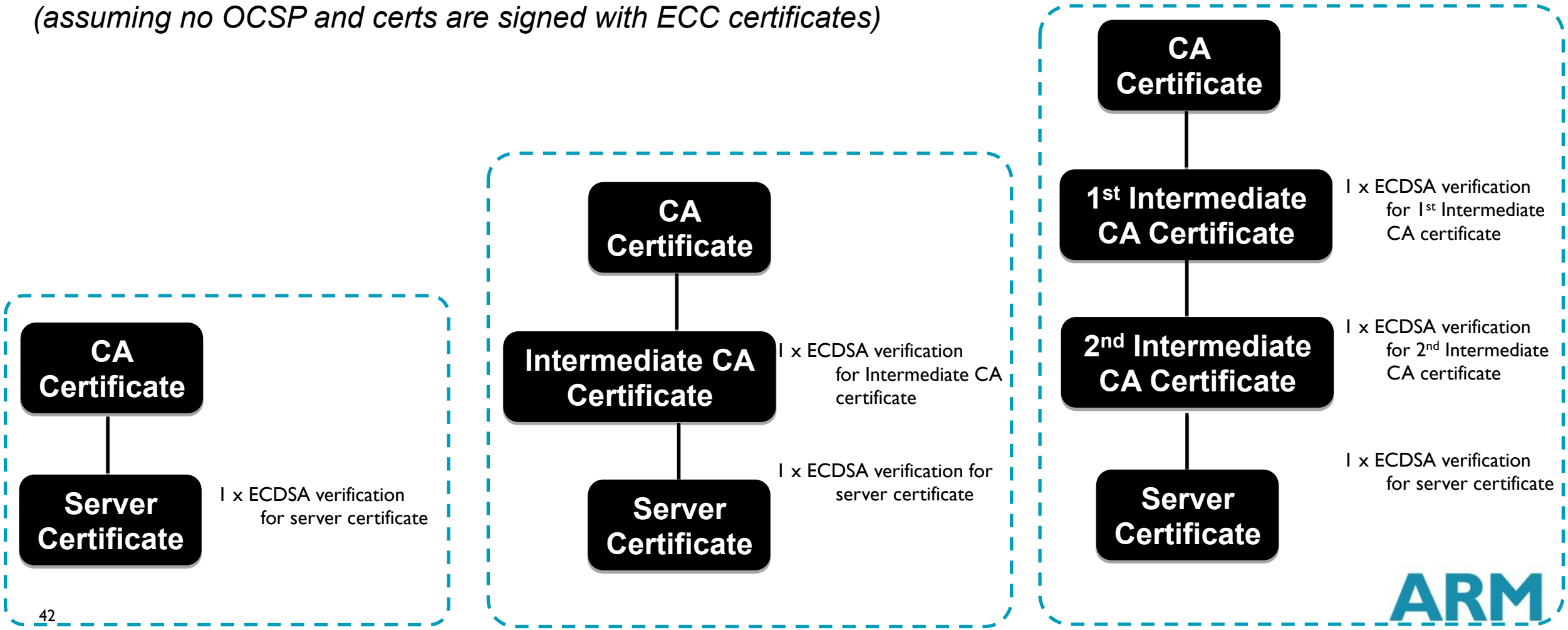
- TLS / DTLS client needs to perform the following computations:
  1. Client verifies the signature covering the Server Key Exchange message that contains the server's ephemeral ECDH public key (and the corresponding elliptic curve domain parameters).
  2. Client computes ECDHE.
  3. Client creates signature over the Client Key Exchange message containing the client's ephemeral ECDH public key (and the corresponding elliptic curve domain parameters).
- Summary:
  - 1 x ECDSA verification for step (1)
  - 1 x ECDHE computation for step (2)
  - 1 x ECDSA signature for step (3)
- Example (LPC1768, secp224r1, W=7, FP and NIST optimization enabled)
  - 329msec (ECDSA verification)
  - 303 msec (ECDHE computation)
  - 85 msec (ECDSA signature)

  Total: 717 msec

ARM

Applying Results to TLS/DTLS Certificates with TLS_ECDHE_ECDSA_*

Same as with raw public key *plus*
(assuming no OCSP and certs are signed with ECC certificates)



ARM

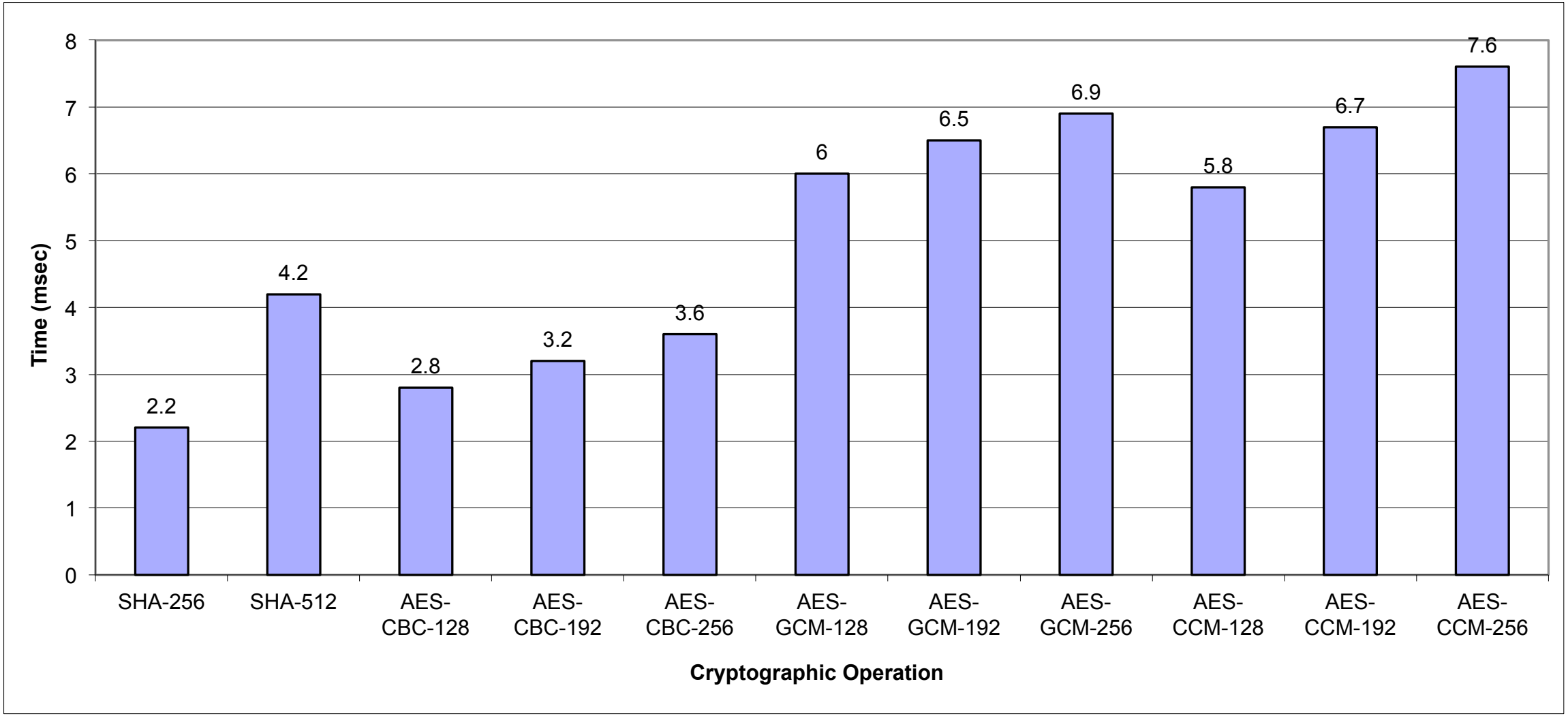
Symmetric Key Cryptography

ARM

Symmetric Key Cryptography

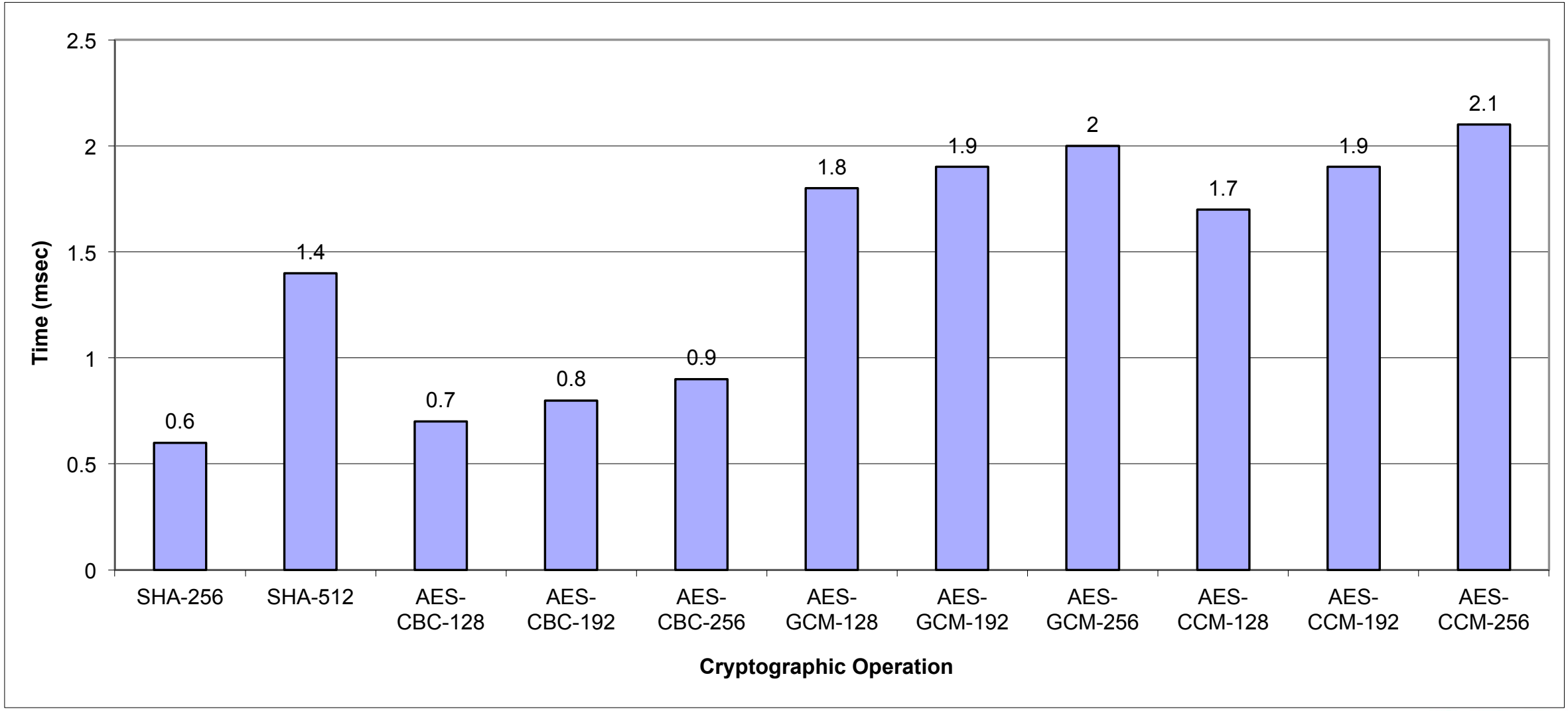
- Secure Hash Algorithm (SHA) creates a fixed length fingerprint based on an arbitrarily long input. The output length of the fingerprint is determined by the hash function itself. For example, SHA256 produces an output of 256 bits.
- Advanced Encryption Standard (AES) is an encryption algorithm, which has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits.
  - A mode of operation describes how to repeatedly apply a cipher's single-block operation to securely transform amounts of data larger than a block.
  - Examples of modes of operation: CCM, GCM, CBC.
- Test relevant information:
  - SHA computes a hash over a buffer with a length of 1024 bytes.
  - AES-CBC: 1024 input bytes are encrypted. No integrity protection is used. IV size is 16 bytes.
  - AES-CCM and AES-GCM: 1024 input bytes are encrypted and integrity protected. No additional data is used. In this version of the test a 12 bytes nonce value is used together with the input data. In addition to the encrypted data a 16 byte tag value is produced.

Symmetric Key Crypto: Performance of the KL25Z



ARM

Symmetric Key Crypto: Performance of the LPC1768



ARM

Conclusion

- ECC requires performance-demanding computations. Those take time.
  - What an acceptable delay is depends on the application.
  - Many applications only need to run public key cryptographic operations during the initial (session) setup phase and infrequently afterwards.
  - With session resumption DTLS/TLS uses symmetric key cryptography most of the time (which is lightning fast).
- Detailed performance figures depend on the enabled performance optimizations (and indirectly the available RAM size), the key size, the type of curve, and CPU speed.
- Choosing the microprocessor based on the expected usage environment is important.

Next Steps

- Collecting performance data on IoT devices is time-consuming.
  We would appreciate help.
- In particular, we need
  - Verification of the gathered data
  - Data from other crypto libraries
  - Further tests (energy efficiency, complete DTLS/TLS handshake data, data about various extensions, more data for Curve25519, etc.).
- We plan to update <draft-ietf-lwig-tls-minimal> accordingly.

ARM