NIST Lightweight Cryptography Workshop 2015
Session VII: Implementations & Performance

Performance of State-of-the-Art Cryptography on ARM-based Microprocessors

Hannes Tschofenig & Manuel Pegourie-Gonnard
(Hannes.Tschofenig@arm.com, Manuel.Pegourie-Gonnard@arm.com)

Presented by Hugo Vincent (Hugo.Vincent@arm.com)

IoT Business Unit
Tuesday, July 21, 2015

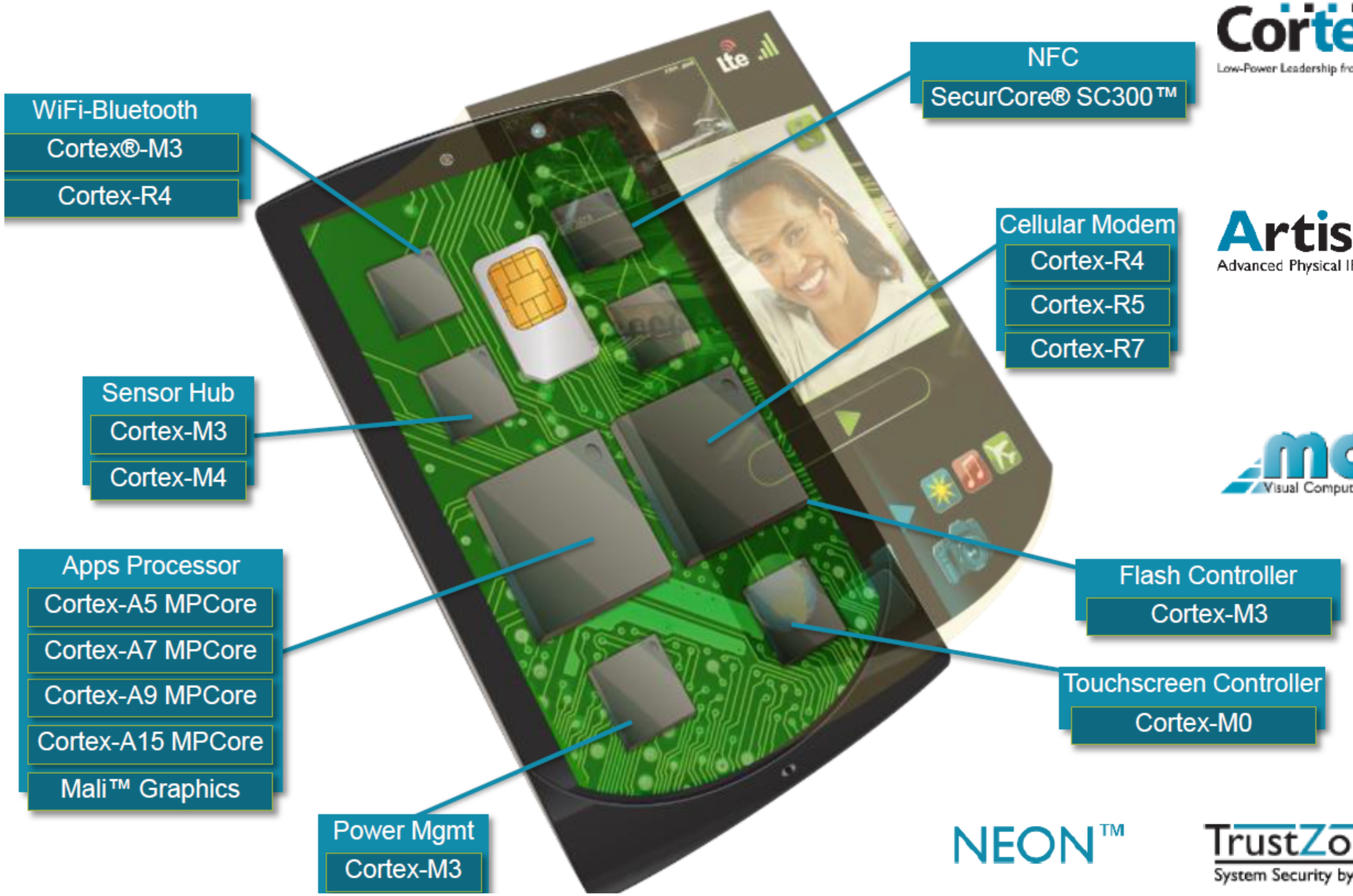
Outline

- Why does ARM care about crypto performance?
  - ARM Cortex-M vs. Cortex-A Class processors.
  - Short overview of the Cortex-M processor family.
  - Internet of Things – a world full of constraints.
- Performance of crypto on Cortex-M class processors
  - Assumptions
  - Hardware used for measurement
  - Symmetric Key Cryptography
  - Public Key Crypto (with different curves)
  - Cortex-M3/M4 Performance
  - Cortex-M0/M0+ Performance
  - Curve25519
  - RAM Usage
  - Applying Results to TLS/DTLS
- Conclusion & Next Steps

Why does ARM care about Crypto Performance?

ARM Processors in Smartphones

- ARM Cortex-A family:
  - Applications processors for feature-rich OS and 3rd party applications
- ARM Cortex-R family:
  - Embedded processors for real-time signal processing, control applications
- ARM Cortex-M family:
  - Microcontroller-oriented processors for MCU, ASSP, and SoC applications



Cortex Low-Power Leadership from ARM

Artisan Advanced Physical IP by ARM

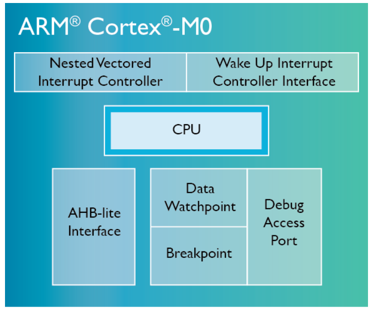
mali Visual Computing by ARM

NEON™

TrustZone® System Security by ARM

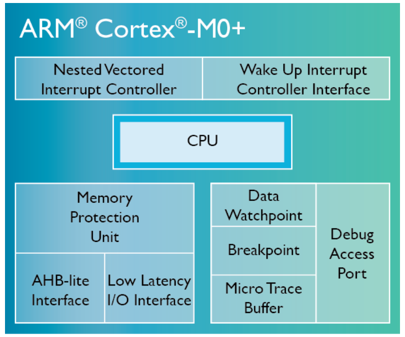
ARM

Cortex-M Processors



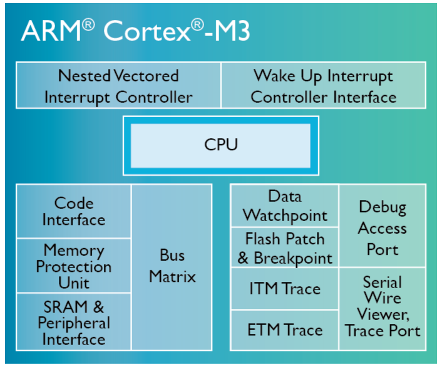
Lowest cost
Low power

Example: Touchscreen Controller



Lowest power
Outstanding energy efficiency

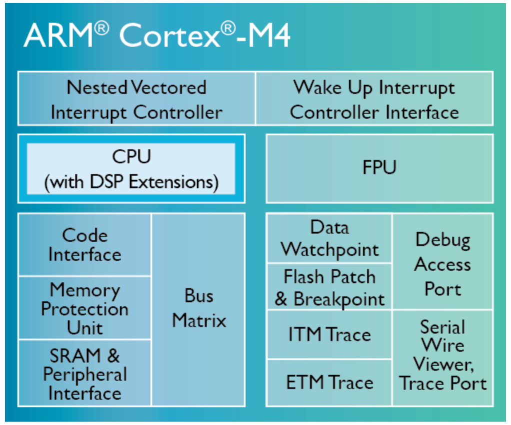
Example: Sensor node
Bluetooth Smart



Performance & efficiency
Feature rich connectivity

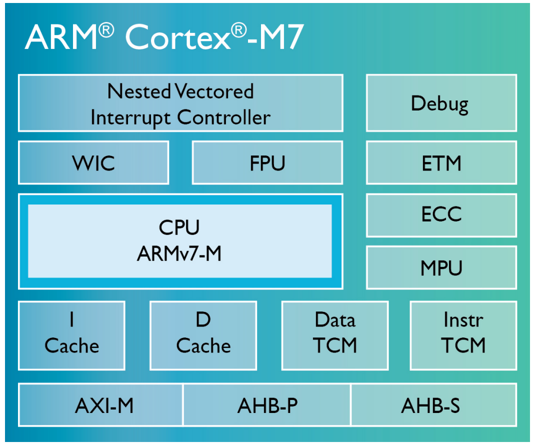
Example: Weables,

Activity trackers, Wifi receiver


ARM® Cortex®-M4
Nested Vectored Interrupt Controller
Wake Up Interrupt Controller Interface
CPU (with DSP Extensions)
FPU
Code Interface
Memory Protection Unit
SRAM & Peripheral Interface
Bus Matrix
Data Watchpoint
Flash Patch & Breakpoint
ITM Trace
ETM Trace
Debug Access Port
Serial Wire Viewer, Trace Port

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

Example: Sensor fusion,
motor control

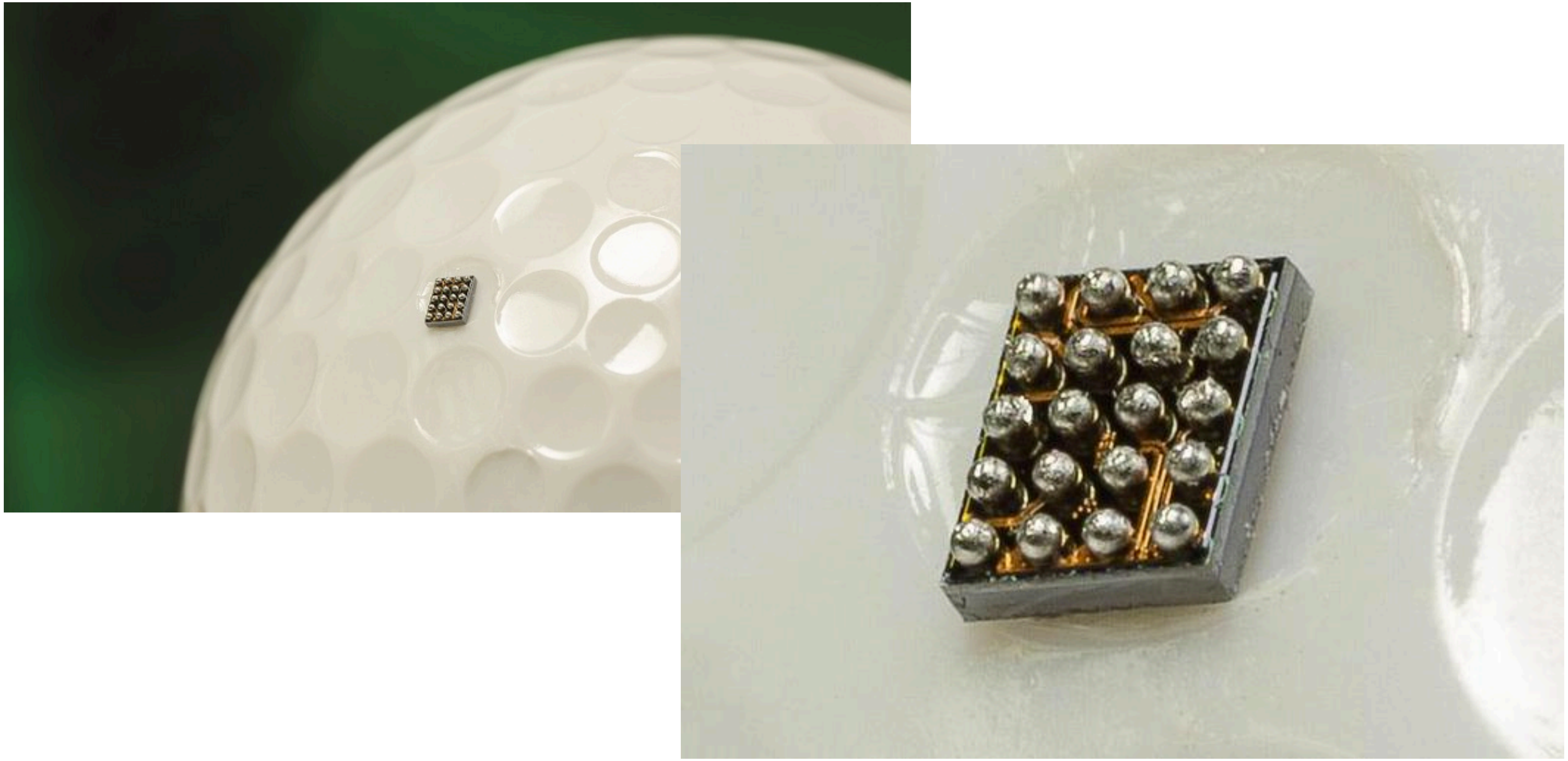


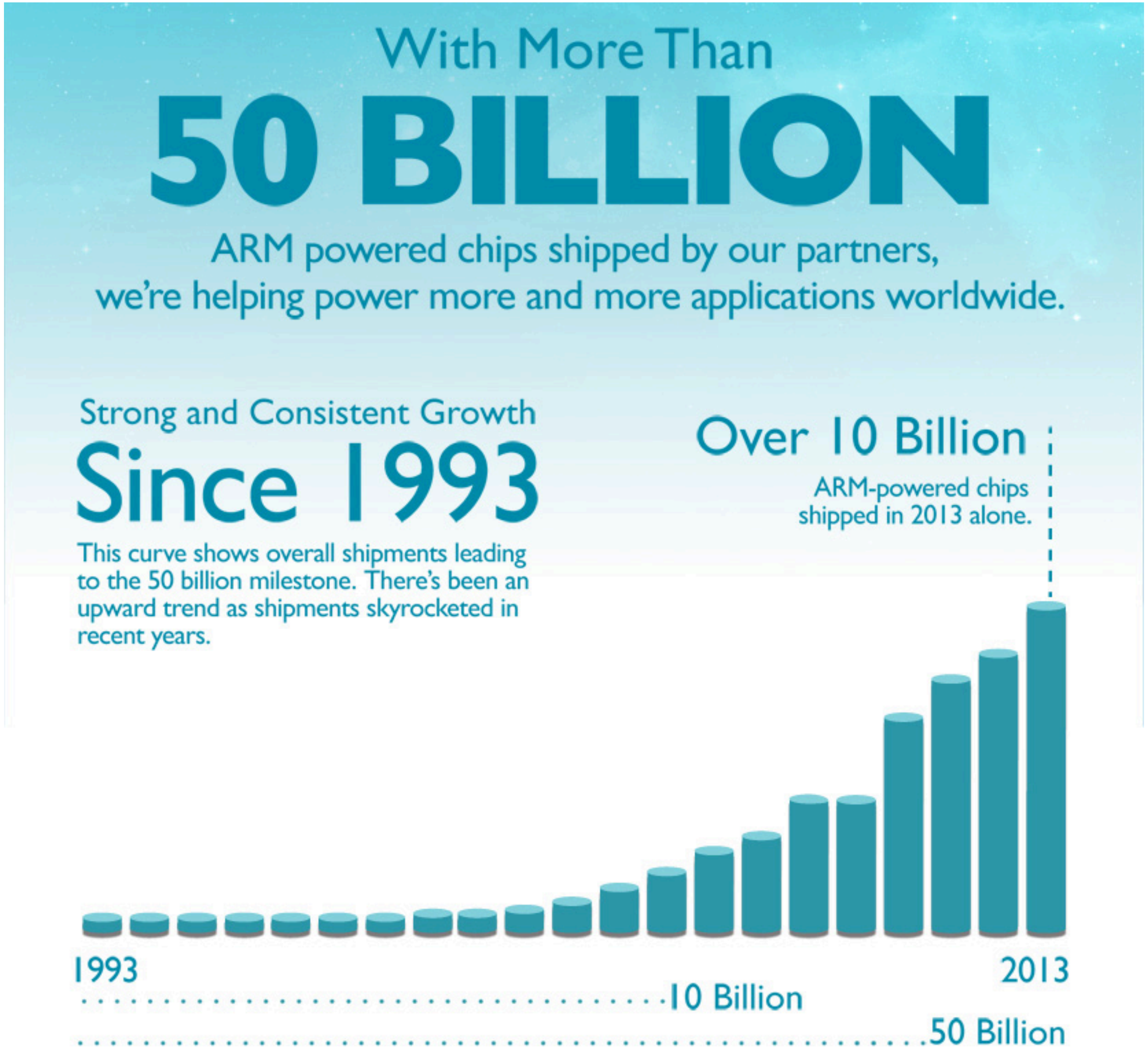
Maximum Performance
Flexible Memory Cache
Single & Double Precision FP

Examples: Automotive,
High-end audio set

ARMv7-M ISA

ARMv6-M Instruction Set Architecture (ISA)





With More Than

50 BILLION

ARM powered chips shipped by our partners, we're helping power more and more applications worldwide.

Strong and Consistent Growth Since 1993

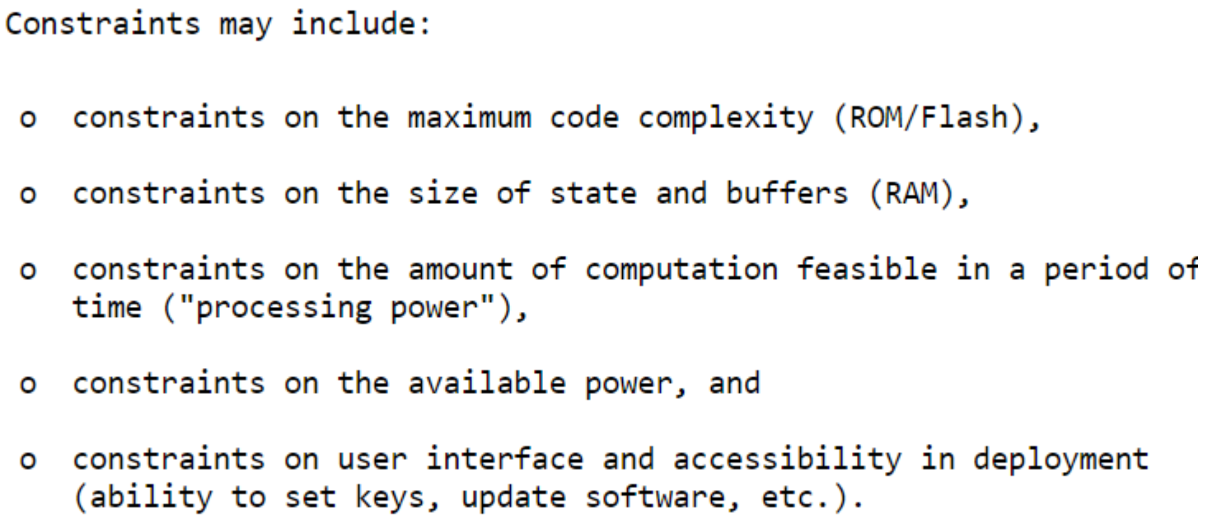
This curve shows overall shipments leading to the 50 billion milestone. There's been an upward trend as shipments skyrocketed in recent years.

Over 10 Billion

ARM-powered chips shipped in 2013 alone.

Wide Range of Constraints

Constrained Node



Constraints may include:

o constraints on the maximum code complexity (ROM/Flash),

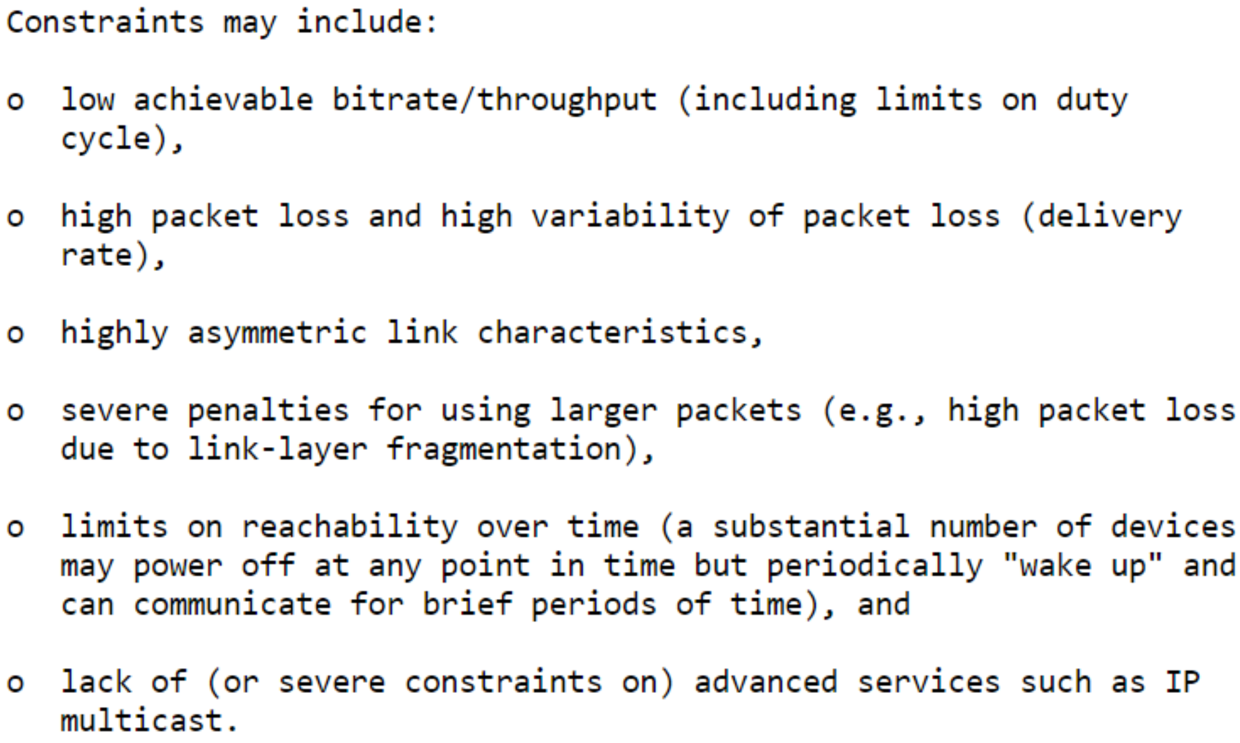
o constraints on the size of state and buffers (RAM),

o constraints on the amount of computation feasible in a period of time ("processing power"),

o constraints on the available power, and

o constraints on user interface and accessibility in deployment (ability to set keys, update software, etc.).

Constrained Networks



Constraints may include:

o low achievable bitrate/throughput (including limits on duty cycle),

o high packet loss and high variability of packet loss (delivery rate),

o highly asymmetric link characteristics,

o severe penalties for using larger packets (e.g., high packet loss due to link-layer fragmentation),

o limits on reachability over time (a substantial number of devices may power off at any point in time but periodically "wake up" and can communicate for brief periods of time), and

o lack of (or severe constraints on) advanced services such as IP multicast.

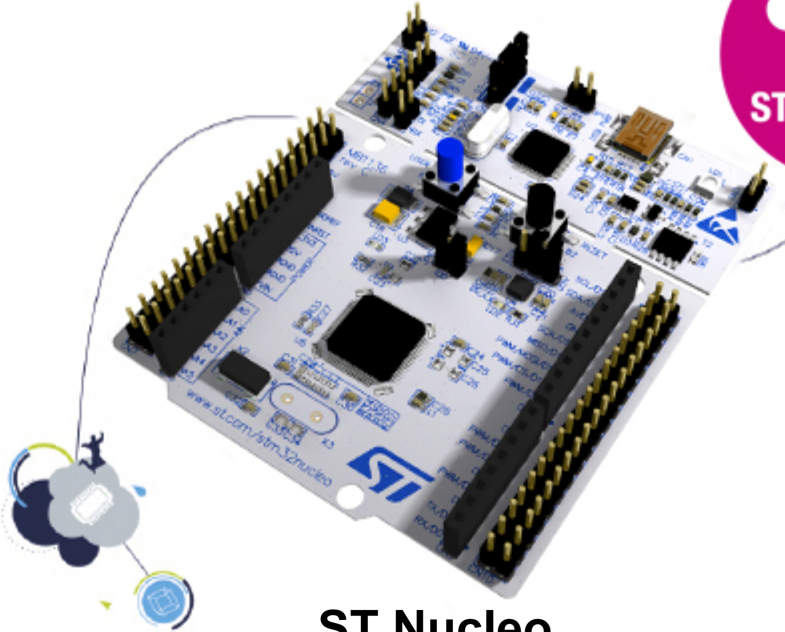
Text copied from RFC 7228 “Terminology for Constrained-Node Networks”

Assumptions

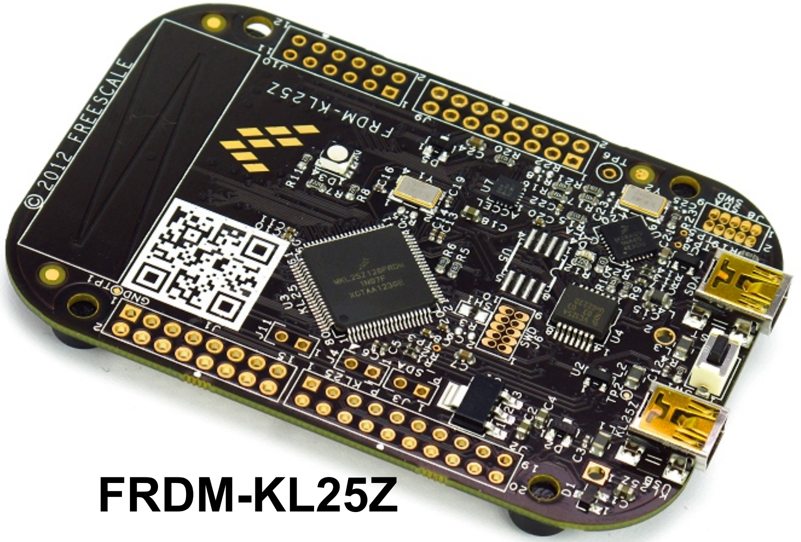
- Main focus of the measurements so far was on
  - Raw crypto primitive performance, not on protocol exchanges
  - Asymmetric crypto: ECC (with several curves) rather than RSA
  - Symmetric crypto
  - Run-time performance (not energy consumption, RAM usage, code size)
- No hardware acceleration was used, pure software
- 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.

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



FRDM-KL25Z

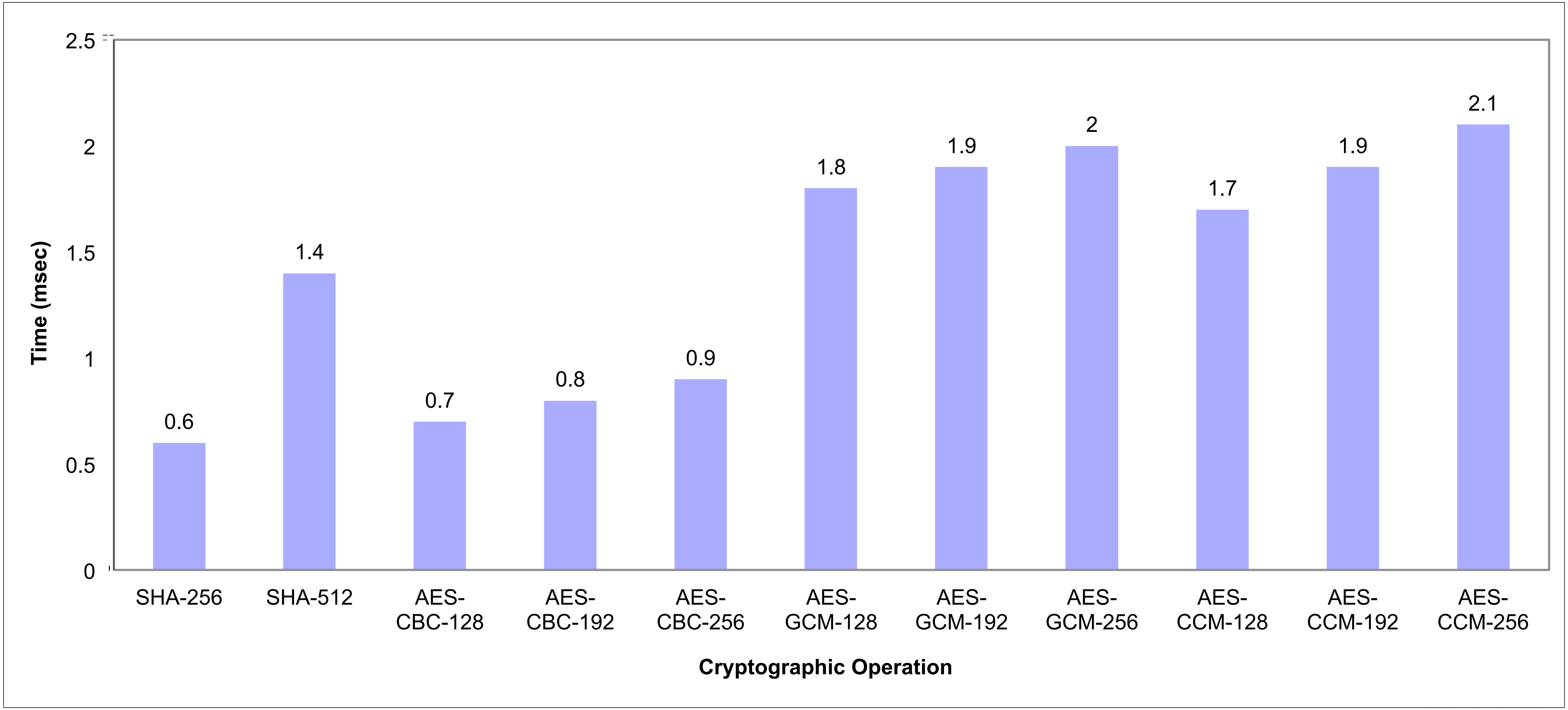
LPC1768

Symmetric Key Cryptography

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 LPC1768



Public Key Cryptography

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).

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: Some of the prototyping boards used here provide true random number generation in hardware, but this hardware was not used in this work.
- 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.
- RFC 7525 recommends at least 112 bits symmetric keys.
- The 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

Performance Figures: A few notes

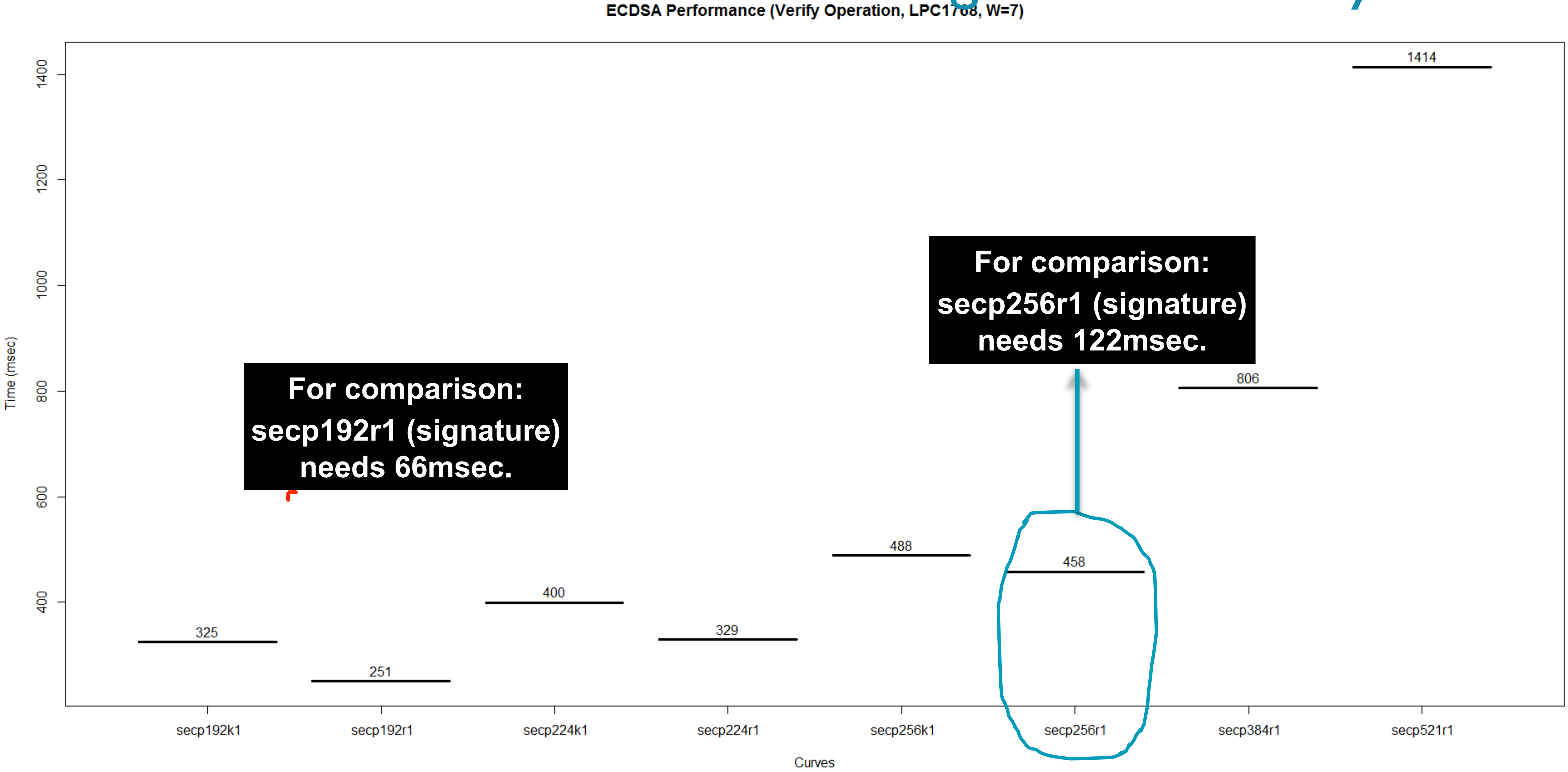
- ECDSA signature operation is faster than ECDSA verify operation.
- Brainpool curves are much slower than NIST curves because Brainpool curves use random primes.
- ECC key sizes above 256 bits are substantially slower than ECC curves with key size 192, 224, and 256.
- ECDH is only slightly faster than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

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

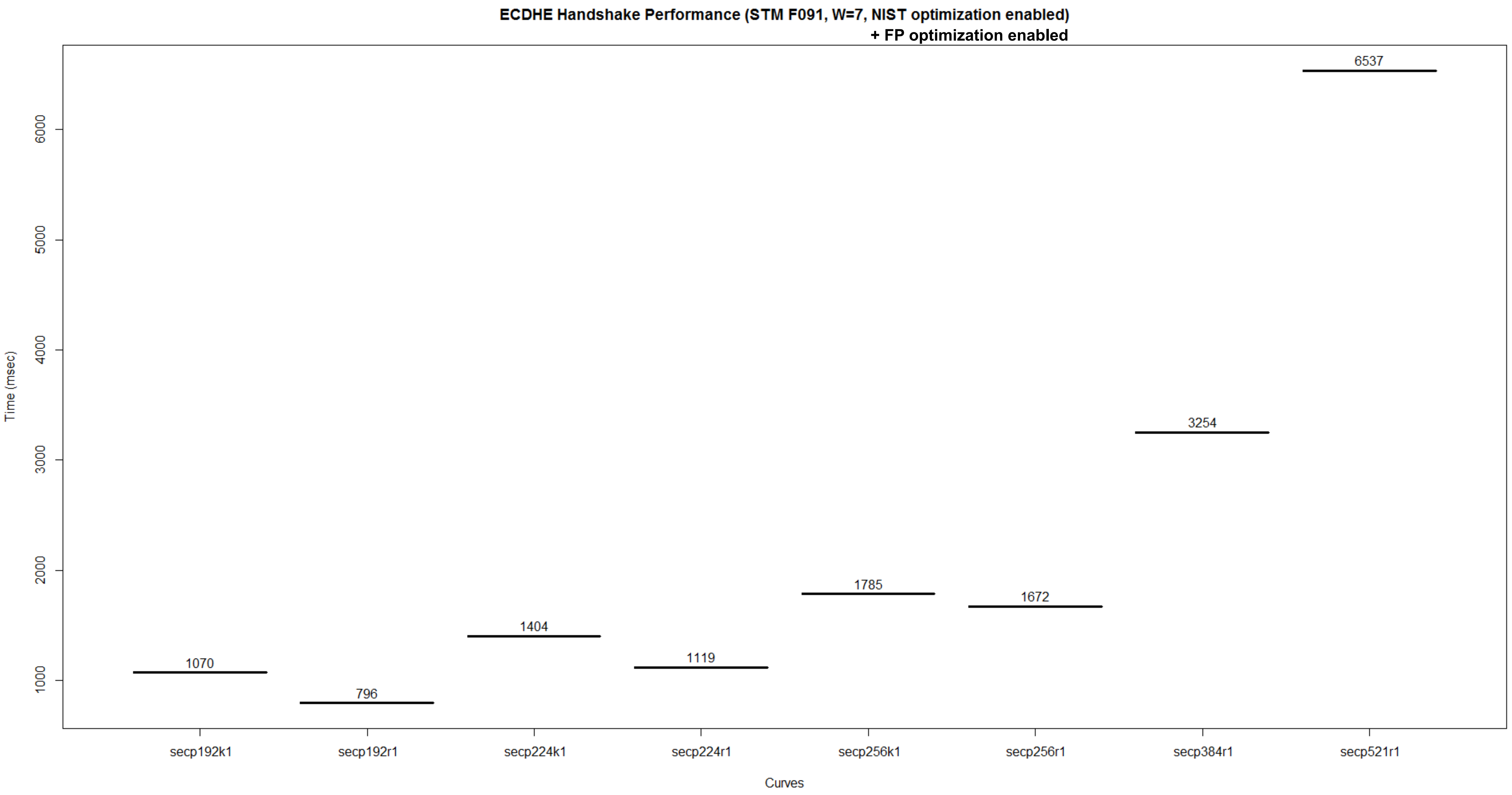
Performance difference between signature vs. verify



For comparison: secp192r1 (signature) needs 66msec.

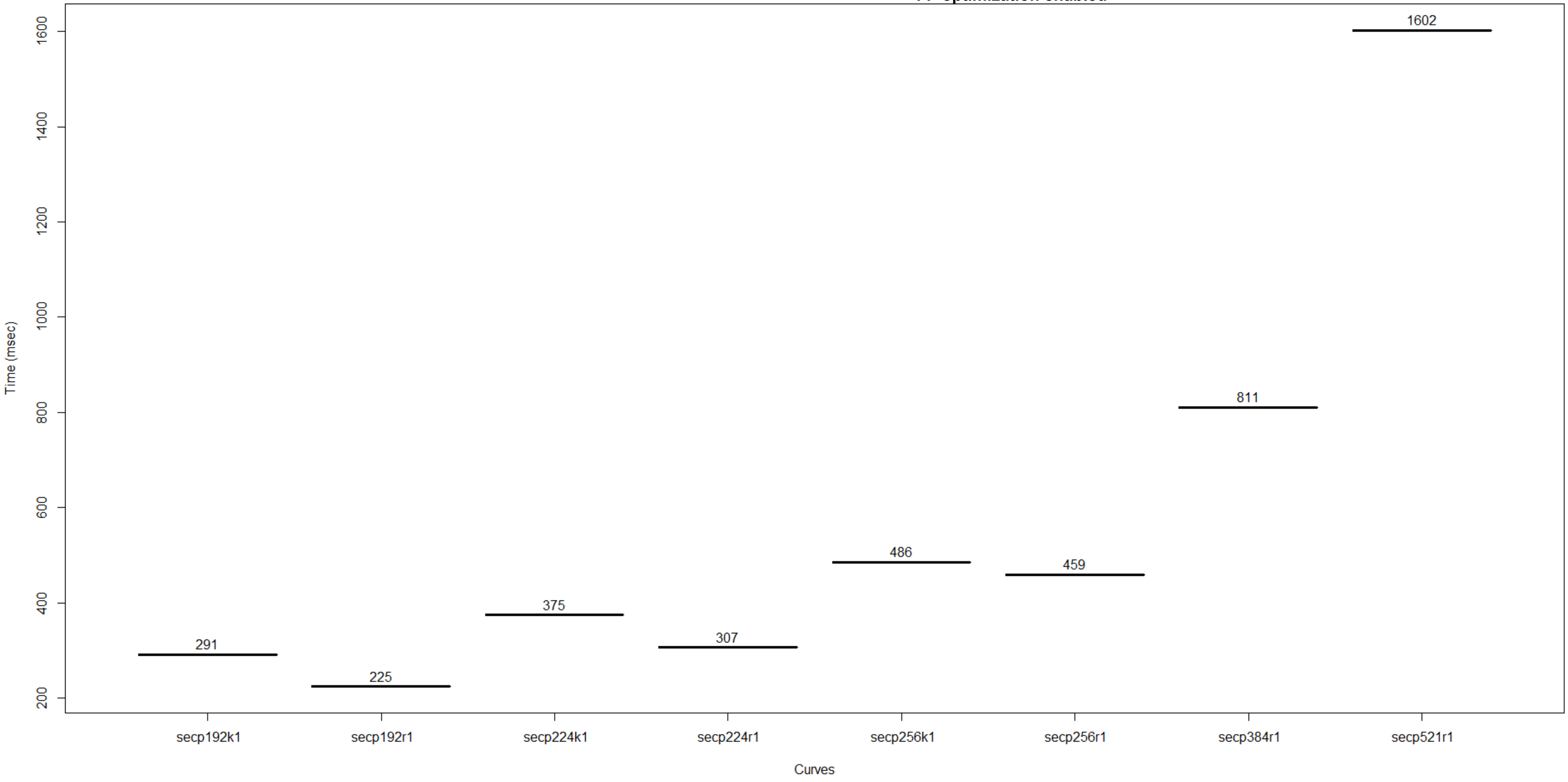
For comparison: secp256r1 (signature) needs 122msec.

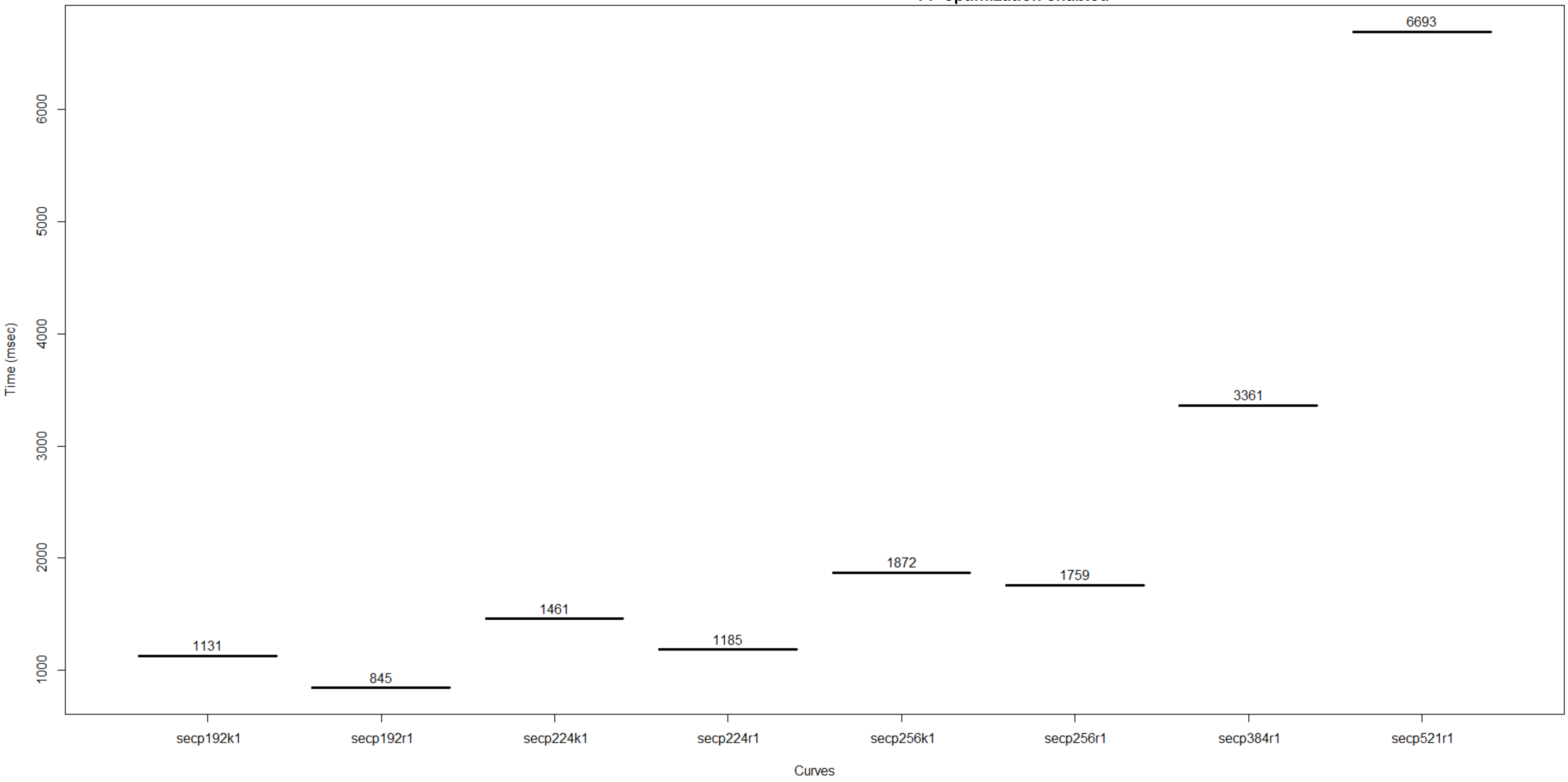
ECC Performance of the Cortex M0/M0+



ECDHE Handshake Performance (STM F091, W=7, NIST optimization enabled)
+ FP optimization enabled

ECDSA Performance (Sign Operation, STM F091, W=7, NIST optimization enabled)
+ FP optimization enabled





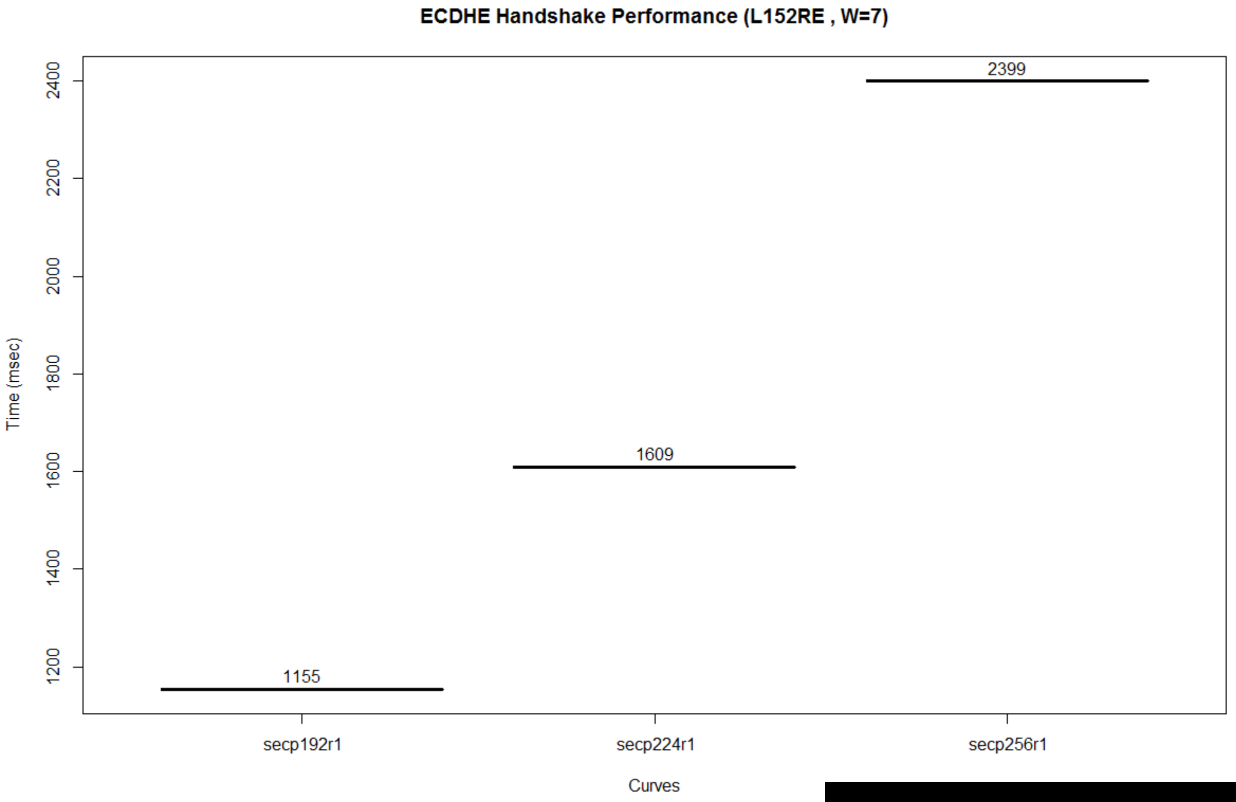
ECDSA Performance (Verify Operation, STM F091, W=7, NIST optimization enabled)
+ FP optimization enabled

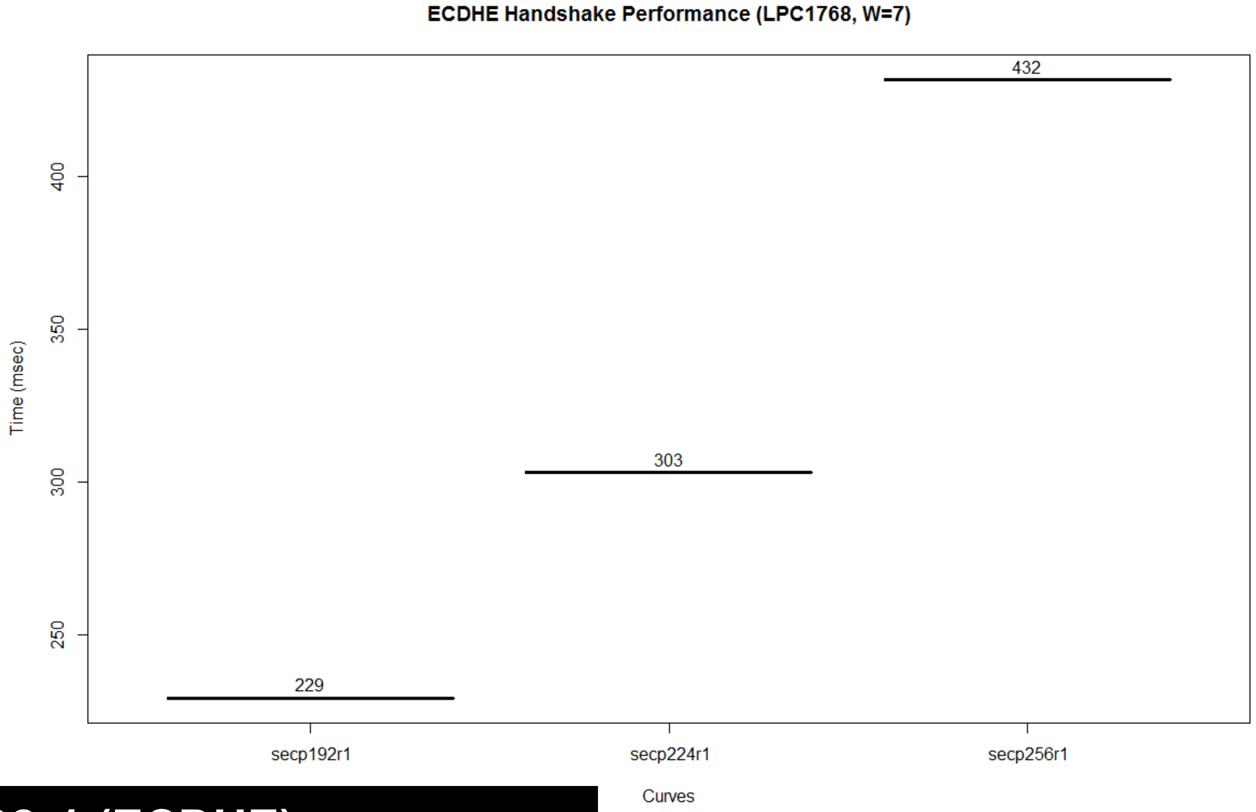
CPU Speed Impact

Performance of ECDHE: L152RE vs. LPC1768

L152RE:
Cortex-M3 with 32MHz

LPC1768:
Cortex-M3 with 96MHz

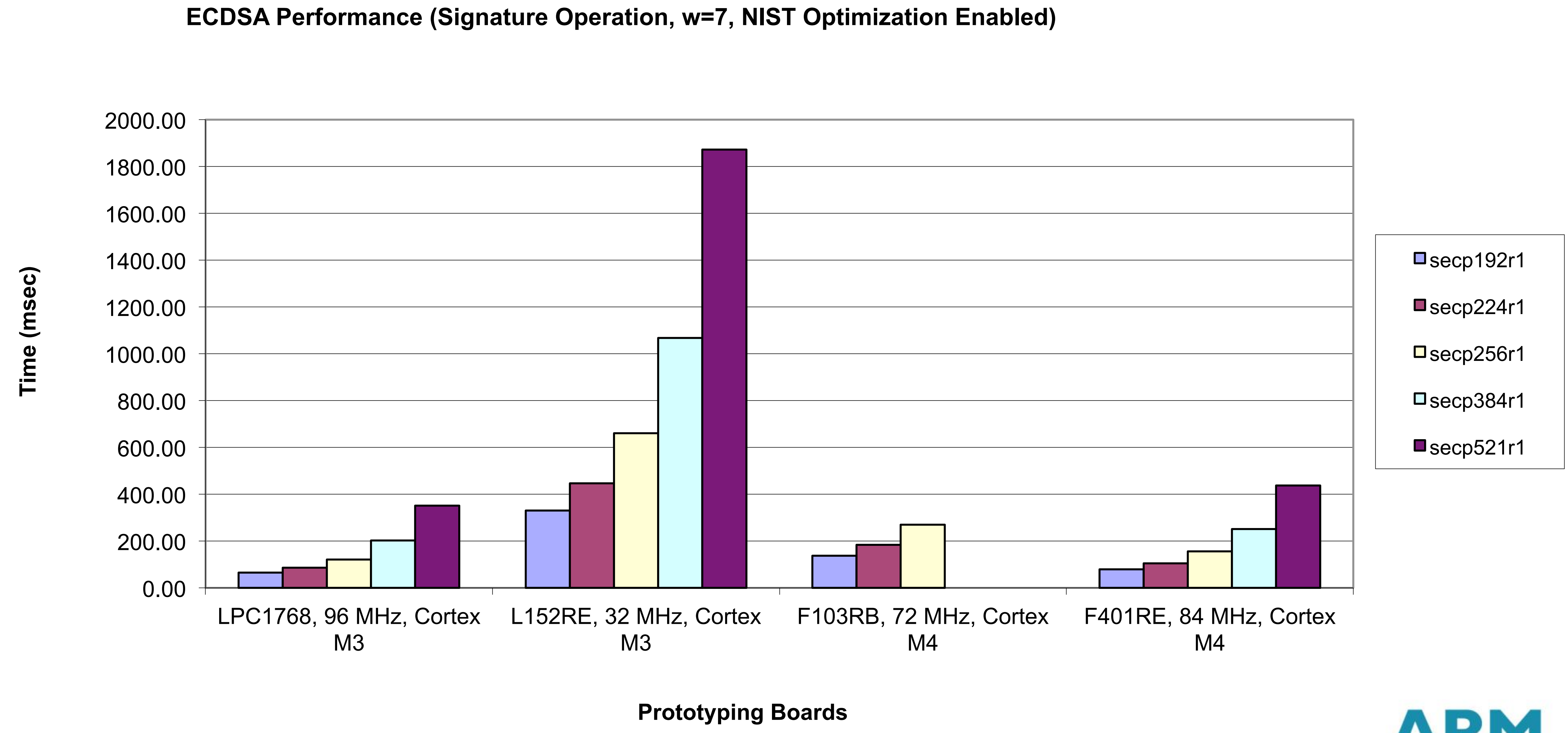




secp192r1 (ECDHE):
1155 msec (L152RE) vs. 229 msec (LPC1768)

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

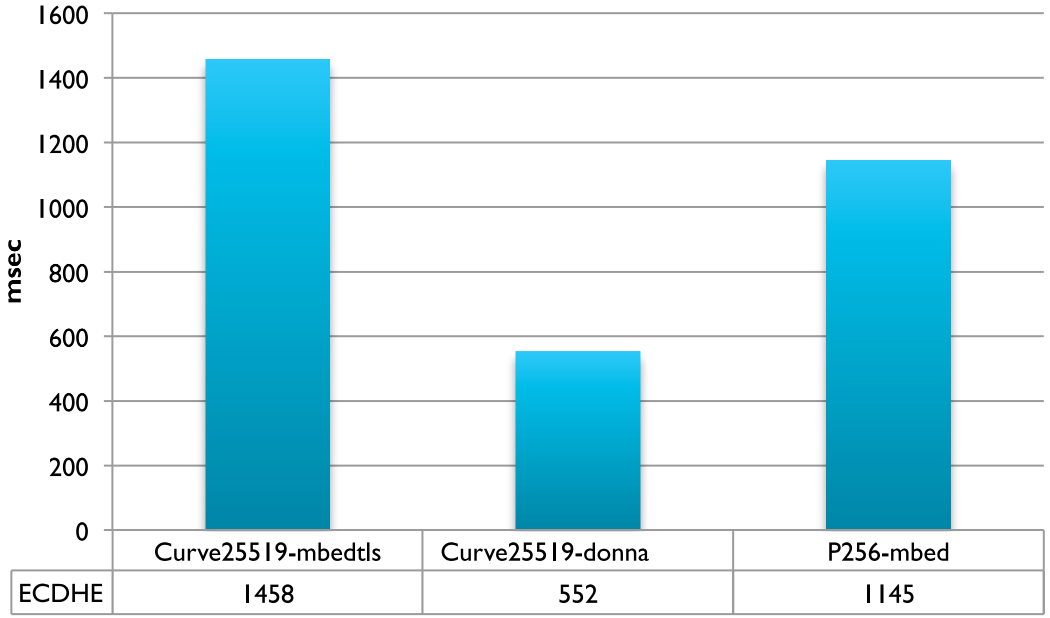
Performance Comparison: Prototyping Boards



Curve25519

(Warning: Preliminary Results)

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

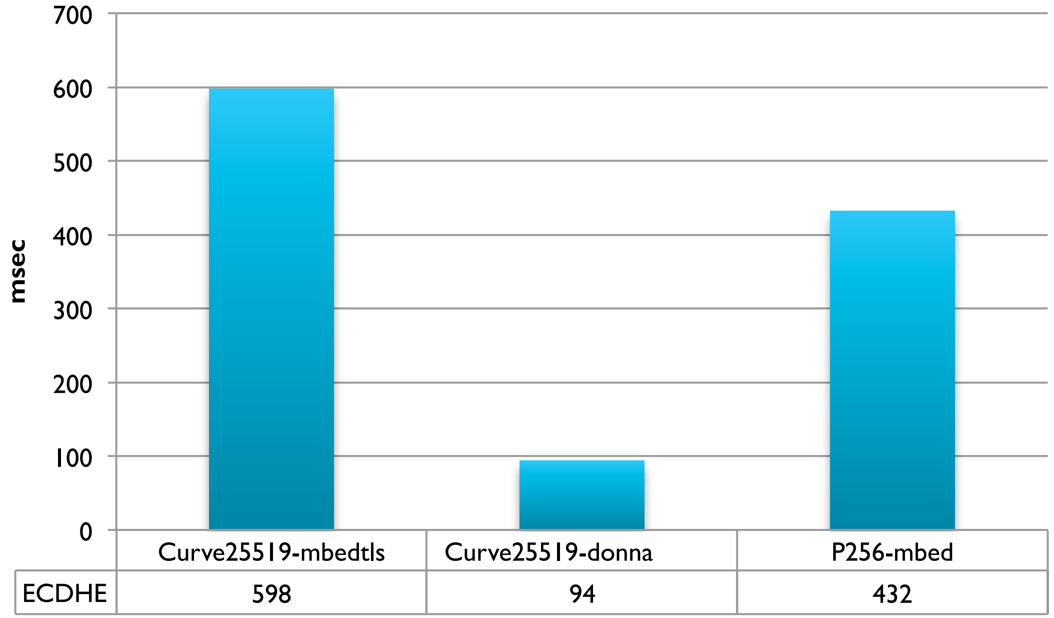


	Curve25519-mbedtls	Curve25519-donna	P256-mbed
ECDHE	1458	552	1145

Notes:

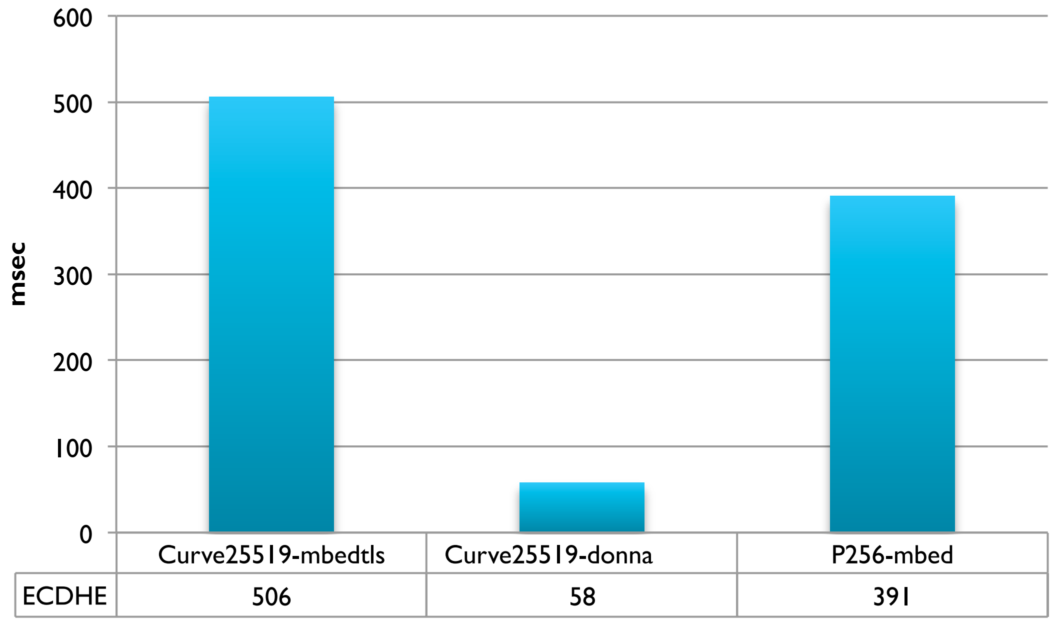
- The Curve25519-mbedtls implementation uses a generic libary. Hence, the special properties of Curve25519 are not utilized.
- Curve25519 has very low RAM requirements (~1 Kbyte only).
- Curve25519-donna is based on the Google implementation. Improvements for M0/M0+ are likely since the code has not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

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

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 %**

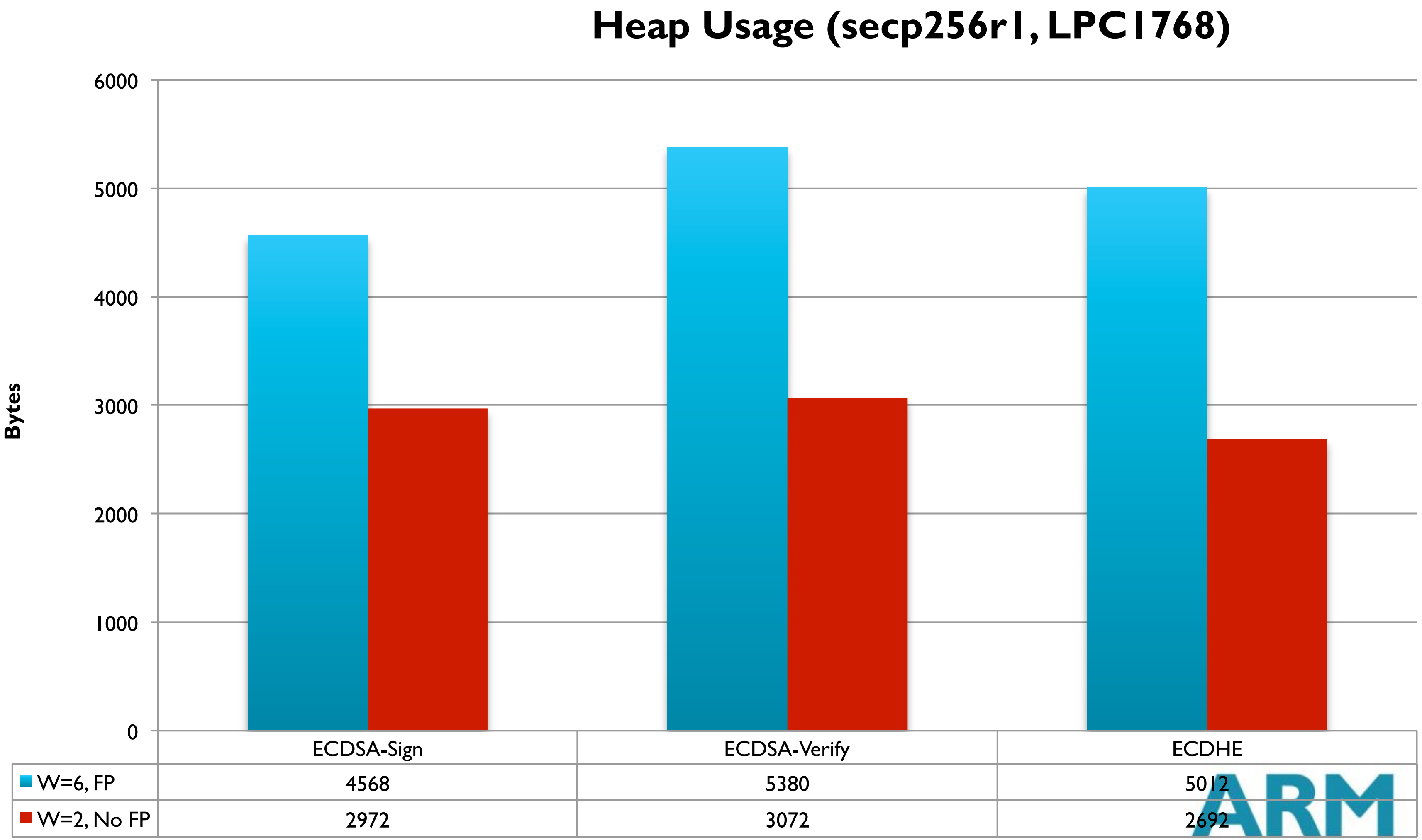
RAM Usage

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.

Summary

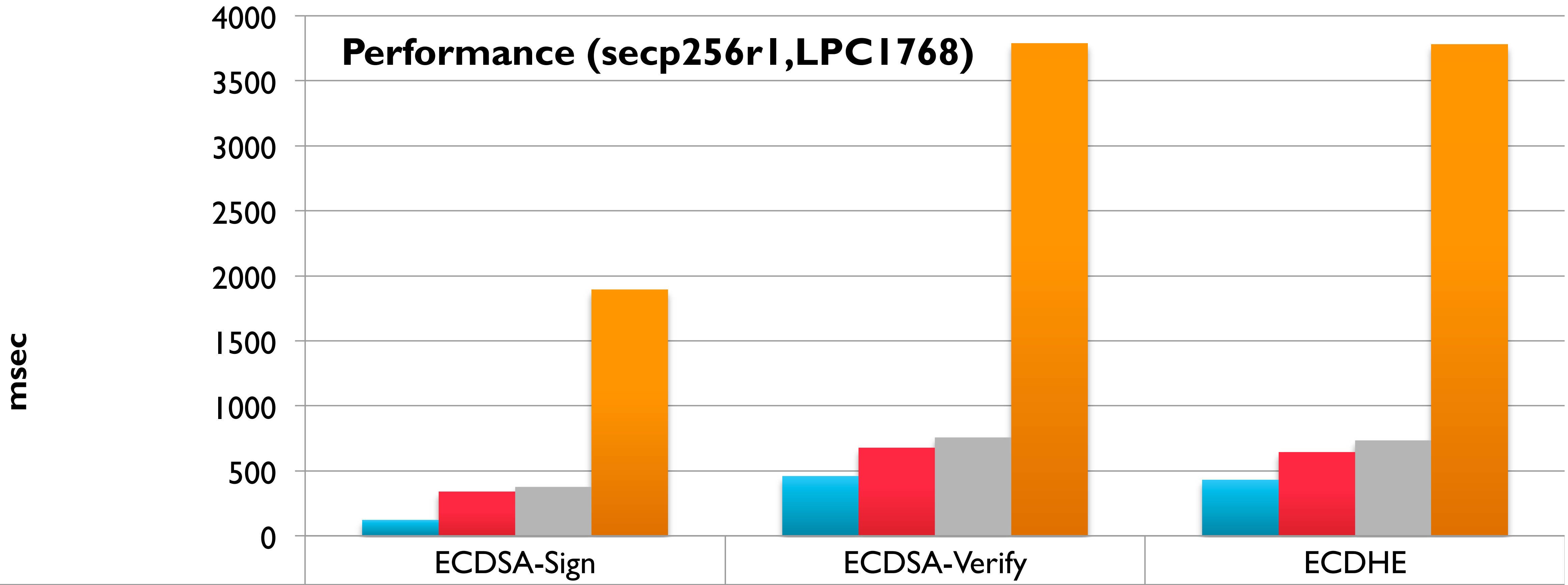
- To enable certain optimizations sufficient RAM is needed. A tradeoff decision between RAM and speed.
- Optimizations pays off.
- This slide shows heap usage (NIST optimization enabled).



Heap Usage (secp256r1, LPC1768)

	ECDSA-Sign	ECDSA-Verify	ECDHE
🟦 W=6, FP	4568	5380	5012
🟥 W=2, No FP	2972	3072	2692

Performance (secp256r1,LPC1768)



	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.

Applying Results to TLS/DTLS

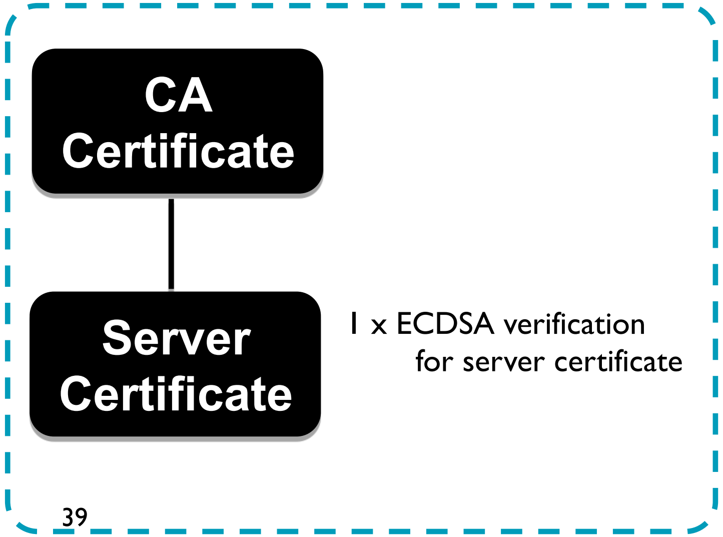
Raw Public Keys with TLS_ECDHE_ECDSA_*

- TLS / DTLS 1.2 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)
  - 329 msec (ECDSA verification)
  - 303 msec (ECDHE computation)
  - 85 msec (ECDSA signature)

  Total: 717 msec

Applying Results to TLS/DTLS Certificates with TLS_ECDHE_ECDSA_*

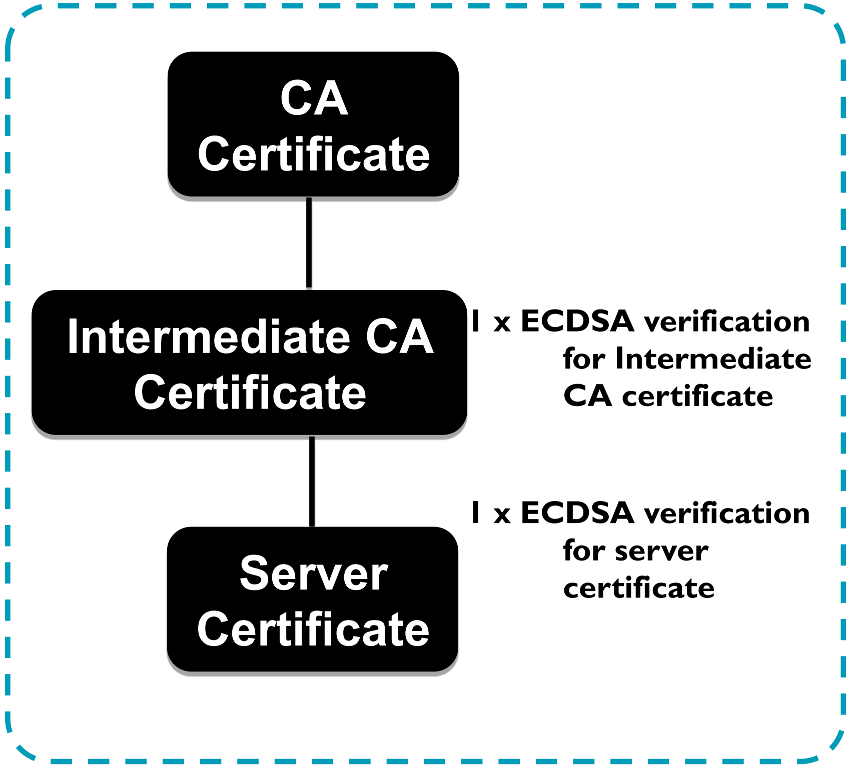
Same as with raw public key *plus*
(assuming no OCSP and certs are signed with ECC certificates)



CA Certificate

Server Certificate

1 x ECDSA verification for server certificate



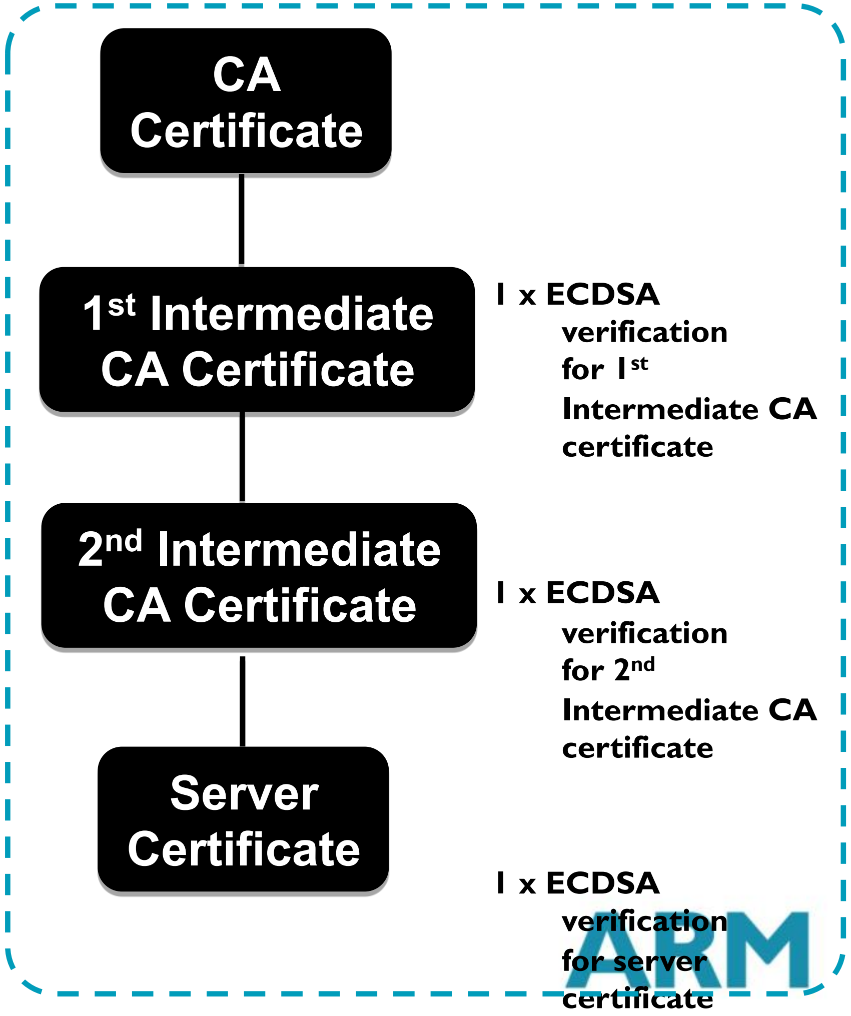
CA Certificate

Intermediate CA Certificate

1 x ECDSA verification for Intermediate CA certificate

Server Certificate

1 x ECDSA verification for server certificate



CA Certificate

1st Intermediate CA Certificate

1 x ECDSA verification for 1st Intermediate CA certificate

2nd Intermediate CA Certificate

1 x ECDSA verification for 2nd Intermediate CA certificate

Server Certificate

1 x ECDSA verification for server certificate

Conclusion

- Block ciphers, hashes, MACs are fast enough already, and often hardware-accelerated in practice anyway.
- 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 DTLS/TLS session resumption symmetric key cryptography is 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 MCU based on the expected usage environment is important.