Application Report
SWRA667–January 2020

Cryptographic Performance and Energy Efficiency on SimpleLink™ CC13x2/CC26x2 Wireless MCUs

Vincent Steil — *SimpleLink Software Development Manager*
Bhargavi Nisarga — *SimpleLink Systems Engineer*

ABSTRACT

This application report describes the benefits of cryptographic acceleration and provides performance and energy consumption measurements of on-chip cryptographic accelerators integrated in the SimpleLink CC13x2/CC26x2 family of wireless microcontrollers (MCUs). It also benchmarks these measurements against Arm® Cortex®-M4F software-based implementations of cryptographic operations. This document also describes device power management and TI driver concepts to consider for enabling efficient usage of SimpleLink cryptographic drivers.

Contents

List of Figures

List of Tables

Trademarks

1 Abbreviations and Acronyms

AES	Advanced Encryption Standard
CBC	Cipher Block Chaining
CCM	Counter with CBC-MAC
CPU	Central Processing Unit
CSPRNG	Cryptographically Secure Pseudo-Random Number Generator
CTR	Counter Mode of Operation
DRBG	Deterministic Random Bit Generator
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Helfman
ECDSA	Elliptic Curve Digital Signature Algorithm
ECJPAKE	Elliptic Curve Password Authenticated Key Exchange by Juggling
GCM	Gallois Counter Mode
HW	Hardware
HWI	Hardware Interrupt
ISR	Interrupt Service Routine
LFSR	Linear Feedback Shift Register
MAC	Message Authentication Code
MCU	Microcontroller Unit
MIPS	Millions of Instructions Per Second
NIST	National Institute of Standards and Technology
OS	Operating System
PKA	Public Key Accelerator
PRNG	Pseudo-Random Number Generator
RAM	Random Access Memory
SHA	Secure Hash Algorithm
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SW	Software
SWI	Software Interrupt
TLS	Transport Layer Security
TRNG	True Random Number Generator
UART	Universal Asynchronous Receiver/Transmitter

2 Introduction

Security in network connected systems has become increasingly critical. There is increased motivation for compromising network connected products over closed systems due to factors such as: a larger attack surface through remote and local access and the potential for successful attacks to leverage compromising nodes in the field into impacting large groups of persons or organizations in a significant way. Recent product exploits have demonstrated that security in network connected ecosystems is only as strong as the weakest link; thus, pushing for increased security, not just in gateways and servers but also in the end nodes, for strengthening the security of the entire system.

Security solutions are implemented across network components to mitigate security risks. Cryptography is a foundational component used in these security solutions that aid in protection of security assets (code, data, or keys) from adversaries. Cryptography is used to provide secrecy and integrity for data and enables both authentication and anonymity to entities involved in communication. Modern cryptography is heavily based on mathematical theory and computer science practice and can impact the performance and energy consumption of resource-constrained embedded systems. Resource-constrained embedded systems typically refer to microcontrollers with limited hardware resources like CPU MIPS (millions of instructions per second) or memory. They are often powered from batteries and have specific battery-life requirements to meet.

The following sections discuss the benefits of cryptographic acceleration, followed by basic concepts of device power management and TI drivers for SimpleLink™ MCUs in the context of using cryptographic APIs for the development of security-focused applications. Next, the document covers benchmarking results for various cryptographic accelerators integrated in the SimpleLink CC13x2/CC26x2 family of wireless microcontrollers (MCUs). The benchmarking results show cryptographic performance and energy consumption using on-chip cryptographic accelerators compared to an Arm® Cortex®-M4F software-based implementation. In this benchmarking effort, we have used Arm mbed TLS software cryptographic functions to compare performance with on-chip cryptographic accelerators.

3 Benefits of Cryptographic Acceleration in Embedded Security Solutions

Cryptographic acceleration offers many benefits including:

- **Increased performance:** On embedded systems with limited CPU performance, cryptographic accelerators can speed up the crypto operations. This improves throughput and eases latency requirements in the application.
- **Concurrency:** It lets applications offload the processing of cryptographic operations from the CPU to the accelerator so the CPU can focus on other operational tasks. This can improve CPU availability for other tasks.
- **Reduced energy consumption:** It reduces the energy consumption of cryptographic operations compared to software implementations of the cryptographic operations. That is, the overall power and time needed for the crypto operations are less compared to a software implementation. For embedded applications that are battery powered, this can help extend battery life.

Here are some examples of security solutions in resource-constrained embedded systems that benefit from cryptographic acceleration:

1. **Reduce latency and optimize energy for implementing networking security**
   a. Commissioning devices into a network with security credentials typically prescribes asymmetric cryptography operations (for example, *Bluetooth*® Low Energy Secure Connections pairing or Thread network commissioning) to mitigate risks of eavesdropping and man-in-the-middle attacks during the commissioning process. The handshakes use asymmetric cryptography based key exchanges and, as a result, are often computationally intensive. It is not uncommon for these asymmetric cryptography operations to be on the order of hundreds of milliseconds to seconds if implemented in software alone. By using hardware cryptographic accelerators, the overall handshake can be sped up by a factor of 10x and greater. This enables a more responsive user experience for end-users commissioning devices onto network.
   b. The data communicated over the network is typically encrypted at different network protocol layers (for example, MAC or link layer, network layer, session layer, application layer) and requires encryption and decryption of these messages at corresponding layers. Cryptographic accelerators not only help meeting critical latency requirements (for example, packet ACK turnaround times within the MAC layer) but can also optimize the overall energy consumed for cryptographic

operations implementing networking security. This improves energy efficiency and can extend the battery life of applications depending on network data traffic to and from device.

2. **Speed up secure boot operation to reduce overall device boot time**
   a. Secure boot is a fundamental security task performed upon device boot to validate if the firmware to be executed by the device is valid. This involves computing the hash of the new firmware image and using this hash to verify the firmware image's signature (stored along with the firmware image) with the root of trust secure boot authentication key stored on-chip.
   b. Using asymmetric cryptography for firmware image authentication only requires storing the public key for signature verification and is therefore a preferred method for secure boot operations. See Reference [1] for secure boot in SimpleLink CC13x2/CC26x2 Wireless MCUs. Using symmetric key based firmware image authentication on the other hand requires storing the symmetric key used for signing the image on the device. If the symmetric key is not properly secured, arbitrary signed images can be used to execute invalid SW images on the device. If the same symmetric key is used in multiple devices, compromising one device may be leveraged into compromising all devices using this key.
   c. The time required to compute the hash is dependent on the size of the firmware image to be verified and the throughput of the hash algorithm implementation. Hardware acceleration of the hash algorithm can help shorten the duration (esp. for larger image sizes). In MIPS constrained microcontrollers, asymmetric signature verification is typically on the order of hundreds of milliseconds to multiple seconds with software-based implementations. Cryptographic accelerators can speed up this operation in embedded microcontrollers. In applications, where the boot up time is critical, cryptographic accelerators are beneficial to reduce the overall boot time.
3. **Reduce application downtime during secure firmware updates**
   a. During device firmware update, regular application operation is halted to perform image verification and programming of a new firmware image. This results in application downtime. Many applications have restrictions on application downtime; that is, the maximum acceptable period of time the application may be down during firmware updates. A device firmware update is comprised of new image verification, followed by programming the new image into the device's non-volatile memory. This order may be reversed depending on whether the new firmware image received is stored on-chip or off-chip. Depending on the image size of the new firmware image to be updated and flash programming time (comprised of flash erase and write times – see the device-specific data sheet for flash timing), the image verification step could contribute significantly towards the overall application downtime. Cryptographic acceleration can help reduce the application downtime during firmware updates and also optimize overall energy consumed during firmware updates.
   b. Secure firmware updates require validating the new firmware image before programming and/or executing the new image on the device. Similar to secure boot, this involves computing the hash of the new firmware image to verify the firmware image signature (sent along with the new firmware image) using the root of trust firmware update authentication key stored on-chip. Asymmetric cryptography for firmware image authentication is preferred over combined asymmetric/symmetric cryptography schemes when only image integrity and authenticity are required as this only requires storing the public key on-chip for signature verification.
   c. As discussed in 2(c), cryptographic accelerators can speed up image verification operation involving hash computation and signature verification. This enables reducing the overall application downtime during firmware updates.

4 TI Drivers for SimpleLink MCUs

To efficiently use the SimpleLink crypto drivers, a basic understanding of power management on the SimpleLink CC13x2/CC26x2 devices is needed. The following sections provide an introduction to power management and driver concepts that affect it.

4.1 Power Management Overview

Power management on the SimpleLink CC13x2/CC26x2 devices is intended to be transparent to the application. The device peripheral drivers notify the power driver of the resources they require when they need them without the application directly interacting with the power driver. However, it can be useful to understand what type of events trigger transitions between active, idle, and standby power modes Reference [14] as well as at what time peripherals are powered on.

While the CPU is executing code, the device is in active mode [14]. It consumes 2.9 mA at 3.3 V if no peripherals are powered [2]. When there is no hardware interrupt (HWI) or software interrupt (SWI) running and the operating system has no tasks ready to run, the power driver automatically transitions the device power mode from active mode into either idle or standby. If a peripheral is performing an operation in the background, the driver requests that the power driver does not transition into standby. The power driver instead transitions the device into an idle power state [14] and turns off the CPU power domain. In idle state, the device consumes 590 µA plus any additional current required by the peripheral running in the background [2]. The device returns to active mode when the peripheral's interrupt wakes the CPU to execute the interrupt service routine (ISR).

In general, all of the peripherals that a driver depends on are powered by the driver when opening a driver instance. Some drivers, such as the SPI or UART, map a physical peripheral to each driver instance and assume that when a driver instance is opened, it has exclusive access to that hardware. Other drivers, such as the crypto drivers, share physical peripherals between multiple open instances and between different types of drivers. Mutual exclusion is handled internally in the driver implementations.

Opening a driver instance takes a small amount of time. The default recommendation is to open a driver instance, use it to perform an operation, and then close it again. This usage pattern minimizes the amount of time the peripheral remains powered. It may make more sense to keep the driver instance open if the application uses it multiple times between standby periods. The same is true if the application has stringent latency requirements for transitioning between power modes (for example, transitioning from idle or standby to active mode).

4.2 Return Behavior

The drivers on the SimpleLink CC13x2/CC26x2 devices support three types of return behavior: blocking, polling, and callback.

- **Blocking** return behavior involves the driver pending on a semaphore until the hardware completes the operation. The running task blocks until an asynchronous event such as an HWI or SWI posts the semaphore when the operation is complete. Blocking return behavior is synchronous from a caller perspective.
- **Polling** return behavior involves continuously polling a hardware or software flag until the operation completes. The device remains in active power state and does not enter idle power state. Polling return behavior is synchronous from a caller perspective.
- **Callback** return behavior involves the initial function call triggering the operation and then returning. When the operation completes, an application provided callback function is called. Callback return behavior is asynchronous from a caller perspective.

There are restrictions on the context an application is permitted to make a call depending on the return behavior configured. Table 1 lists the permitted calling contexts by configured return behavior.

Table 1. Restrictions on Calling Context by Return Behavior

	Task	SWI	HWI
Blocking	Allowed	Not allowed	Not Allowed
Polling	Allowed	Allowed	Allowed
Callback	Allowed	Allowed	Allowed

The choice of return behavior is influenced by the application design. If your application makes crypto calls from an asynchronous state machine in interrupt context or an event driven task, you should choose polling for short operations and callback for long ones. If your application makes crypto calls from a synchronous task, you should choose polling for short operations and blocking for long operations.

4.2.1 Runtime Overhead

We use *overhead* to describe the CPU cycles spent over the course of an operation on anything that is not directly related to configuring the hardware or returning to the original calling context.

Polling return behavior has the least overhead. This is because its implementations are the closest to bare-metal we can provide. Apart from acquiring the mutex protecting the hardware from concurrent access, there are no OS calls and interrupts are avoided when hardware flags are available. How this affects power and call duration depends on how computationally expensive the operation is.

Callback return behavior has more inherent overhead. It requires the handling of an asynchronous event. That means that at least one context switch is required to handle the asynchronous event.

Blocking return behavior is effectively a specialized case of callback return behavior in terms of overhead. It will always post a semaphore and unblock a pending Task when the asynchronous event triggers.

If the power driver puts the device into idle when using callback or blocking return behavior, that comes with the associated overhead of going into and coming out of idle.

Table 2. Relative Runtime Overhead of Different Return Behaviors

Return Behavior	Overhead
Polling	Low
Callback	Medium-High
Blocking	High

4.3 Efficient Power Management

Blocking and callback return behavior allow the power driver to opportunistically put the device into the idle state when the CPU is not needed. While the hardware peripheral performs the operation in the background, the power driver turn off the CPU power domain while it waits for the peripheral to trigger an interrupt upon completion of the operation.

Though polling return behavior offers the least overhead, it provides no integrated power management to place the CPU in idle state when not in use. Because the CPU is in active mode continuously polling a flag to signal the operation's completion, there is no opportunity for the power driver to put the device into a lower power state. Because polling return behavior should only be used for short operations, the lack of power management is not a concern. The overhead of using callback or blocking return behavior outweighs any power savings derived from spending time in idle power mode for short operations.

5 CC13x2/CC26x2 Crypto Peripherals

There are multiple crypto accelerators on the CC13x2/CC26x2 devices: the AES and hash accelerator, the public key accelerator (PKA) engine, and the true random number generator (TRNG).

5.1 AES and Hash Crypto Accelerator

The AES and hash crypto accelerator is responsible for AES (see Reference [3]) and SHA-2 (see Reference [4]) functionality. It supports numerous AES block cipher modes of operation and all SHA-2 output digest sizes. It also has its own integrated DMA that can concurrently stream in input and out output. As a result, the drivers generally set up the crypto accelerator, start it, and then wait for the accelerator to complete the entire operation without further intervention.

The time required to process an additional block of input data is relatively small compared to the overhead of setting up the accelerator. It is therefore beneficial from an energy consumption perspective to perform fewer longer operations rather than multiple shorter ones when using drivers based on the AES and hash crypto accelerator.

Because of the trade-off between marginal energy cost per input block and setup overhead, there is a message length where an AES or SHA2 operation consumes less total energy using blocking return behavior than polling return behavior despite having a longer duration.

Table 3. Recommended Return Behavior Based on Payload Length

Driver	Polling Recommended Interval (bytes)	Blocking or Callback Recommended Interval (bytes)
AES ECB	< 1350	≥ 1350
AES CBC	< 1350	≥ 1350
AES CTR	< 1350	≥ 1350
AES CCM	< 675	≥ 675
AES GCM	< 1475	≥ 1475
AES CTR DRBG	< 1350	≥ 1350
SHA-224	< 4650	≥ 4650
SHA-256	< 4650	≥ 4650
SHA-384	< 7100	≥ 7100
SHA-512	< 7100	≥ 7100

See Appendix: Plots of Blocking vs Polling Performance for further benchmarking data of the cryptographic accelerators in Table 3 when using drivers with polling vs. blocking return behavior.

5.2 Public Key Accelerator

The public key accelerator (PKA) provides access to large number math operations and several dedicated elliptic curve cryptography (ECC) (see Reference [5]) primitives. It also contains 2k bytes of dedicated SRAM to manage intermediate results and is used as workspace for the PKA itself. The PKA based drivers listed below manage all required PKA RAM accesses themselves.

Unlike the AES and hash crypto accelerator, there are no true fire-and-forget operations when using the PKA engine. Every ECC driver requires multiple PKA operations to perform any driver operation. This is abstracted away from the application by the drivers.

PKA operations vary highly in their duration. A simple 128-bit addition might take a few microseconds while a Short-Weierstrass scalar point multiplication takes 113 ms. The computational cost of ECC driver operations is dominated by scalar point multiplications and point additions as they are the most computationally expensive operations by several orders of magnitude and are executed in every ECC driver call. With such operation durations, the overhead costs are only marginal. Blocking and callback return behaviors perform well with ECC operations compared to polling return behavior. Polling return behavior is implemented for portability between devices within the SimpleLink ecosystem and for completeness.

Table 4. Drivers Using the PKA Engine

Driver	Recommended Return Behavior
ECDH	Blocking or Callback
ECDSA	Blocking or Callback
ECJPAKE	Blocking or Callback

5.2.1 ECDH Power Management Driver Example

When combined, the peripheral driver and power driver opportunistically put the device into the lowest power state it can be in at any point during an operation. Figure 1 shows a simplified example of how the ECDH and power driver interact when generating a public key. The device starts in active mode after booting and no peripherals are turned on yet.

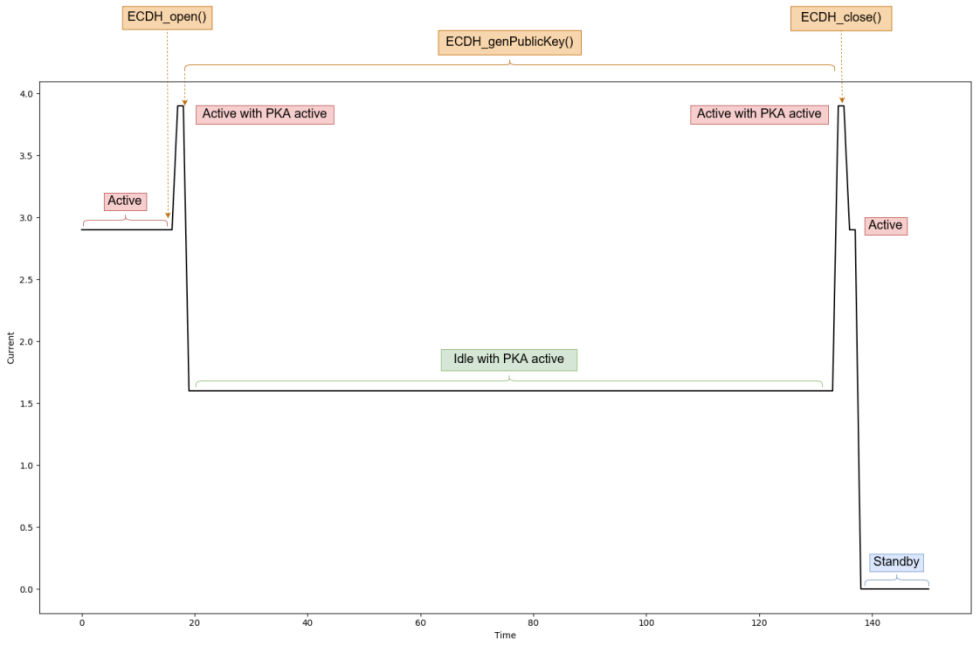

Figure 1. Simplified Power Transition Diagram for ECC Public Key Generation

1. The application opens an ECDH driver instance configured with blocking return behavior by calling ECDH_open(). The driver turns on the PKA peripheral.
2. The application calls ECDH_genPublicKey(). The driver sets up the PKA and appends an internal semaphore.
3. The power driver puts the device into the idle power mode because the CPU is not currently busy but the PKA is active.
4. The PKA triggers an interrupt that wakes up the CPU, and the power driver brings the device into active power mode. The semaphore the thread is pending on is posted, and ECDH_genPublicKey() returns.
5. The application calls ECDH_close(). The ECDH driver turns off the PKA peripheral.
6. The device has nothing else to do, and the power driver transitions the device into standby.

The concepts in Figure 1 also transfer to the other SimpleLink crypto drivers.

5.3 TRNG

The true random number generator (TRNG) in the SimpleLink CC13x2/CC26x2 devices uses 24 free running oscillators (FRO) that are repeatedly sampled and collated in a linear feedback shift register (LFSR). After a sufficient number of samples have been collected and merged into the LFSR, the entropy can be read out from the registers and the TRNG begins collecting more samples. At minimum, the TRNG generates 64 bits of random output. The entropy content of these 64 bits depends on the amount of samples gathered from the FROs. The default number of samples the driver requires before reading out entropy from the TRNG is 240000. This works out to 5 ms at the highest FRO sampling rate and is enough to generate 64 bits of entropy.

The most commonly requested random number sizes will be 128 bits and 256 bits to generate symmetric or asymmetric keying material. These operations will take the TRNG driver 10 ms and 20 ms respectively. The CPU is not needed while the TRNG samples the FROs in the background. The sampling of the FROs takes up the vast majority of the overall duration. This makes blocking or callback return behavior attractive to save power or enable the CPU to perform other operations in the interim.

Newer TRNG driver implementations feature an entropy pool that contains pre-generated entropy. When the application requests a number of random bytes, the driver will first deplete its entropy pool before using the TRNG hardware for a specific request. If there was enough entropy in the pool to fulfil the request, the driver returns immediately regardless of return behavior and refills the entropy pool in the background. The choice of return behavior is consequently only relevant during periods of high TRNG driver activity when the entropy pool is depleted.

For benchmark purposes, we always assume that the entropy pool is empty when a TRNG_generateEntropy() call is made. The use of the entropy pool only allows us to move the entropy generation to earlier in time. This reduces latency and allows for increased parallelism between entropy generation and operations that consume entropy. That parallelism can reduce the amount of time the device spends in active or idle compared to standby and thus reduces power consumption.

Table 5. Drivers Using the TRNG

Driver	Recommended Return Behavior
TRNG	Blocking or Callback

6 Benchmarks

To showcase the power and speed benefit of using the hardware accelerated drivers in comparison to a software based implementation, we have run several benchmarks. The metrics of interest are:

- Crypto operation duration
- Energy consumption
- Relative runtime performance and energy efficiency versus a software implementation

To compute these metrics, we collected the following data points for each benchmark:

- Duration HW: The duration of the operation when using hardware accelerators through the crypto drivers.
- Duration SW mbed TLS: The duration of the operation when using mbed TLS. mbed TLS is a widely adopted open source software crypto package that is optimized for embedded applications. This provides a reference for the durations of operations on an equivalent device without hardware accelerators.
- Average Current HW: The average current consumed during the operation. Because the device is turning on and off accelerators and entering idle dynamically, the current consumption is not constant. Therefore, the average current consumption during this operation duration is considered.
- Average Current SW mbed TLS: The average current consumed during a software-only run of a benchmark.

The comparison metrics *Duration Improvement* and *Energy Efficiency Improvement* both describe the relative performance increase obtained by using the hardware accelerated drivers compared to the pure software implementation. They are computed by dividing the mbed TLS duration or energy used by the equivalent accelerated driver value.

$$K_{\text{Duration Improvement}} = \frac{t_{\text{mbed TLS}}}{t_{\text{HW}}} \quad (1)$$

$$K_{\text{Energy Efficiency Improvement}} = \frac{t_{\text{mbed TLS}} \times \overline{I_{\text{mbed TLS}}}}{t_{\text{HW}} \times \overline{I_{\text{HW}}}} \quad (2)$$

The larger the number, the more efficient it is to use the drivers.

All benchmarks were run on a SimpleLink multi-standard CC26x2R wireless MCU LaunchPad™ development kit (LAUNCHXL-CC26X2R1) running at 48 MHz and powered with 3.3 V. The benchmark project was compiled with SimpleLink CC13x2-CC26x2 SDK version 3.20.00.68. The mbed TLS library was built with mbed TLS version 2.7.9. The project and mbed TLS library were compiled with the TI ARM C/C++ 18.12.1.LTS compiler using -O3 optimization settings. Current measurements were taken using an Agilent N6705B Power Analyzer. Time measurements were made with a Saleae Logic 8 measuring GPIO signals. Results are comparable between the CC26x2 and CC13x2 device families.

6.1 AES and Hash Crypto Accelerator Based Drivers

6.1.1 AES CBC

The AES CBC (see Reference [6]) benchmark consists of encrypting messages of varying sizes. The duration of an AES CBC operation is affected by the key length and input length. We used the default key length of 128 bits and input lengths of 64, 8k, and 16k bytes.

Table 6. AES CBC Benchmark Results

Payload Length (bytes)	Duration HW (ms)	Duration SW mbed TLS (ms)	Duration Improvement	Average Current HW (mA)	Average Current SW mbed TLS (mA)	Energy Efficiency Improvement
64	0.028	0.122	4.4	3.84	3.10	3.5
8000	0.640	14.7	23.0	2.87	3.10	30.7
16000	1.171	29.3	25.0	2.76	3.10	34.6

6.1.2 AES CCM

The AES CCM (see Reference [7]) benchmark consists of encrypting and authenticating messages of varying sizes and authenticating additional authenticated data (AAD) of a constant size. The duration of an AES CCM operation is governed by the key size, input length, and AAD length. We used a key length of 128 bits, a fixed AAD length of 32 bytes, and input lengths of 64, 8k, and 16k bytes. We chose a constant AAD length because the AAD usually contains metadata that does not scale in length with the message length.

Table 7. AES CCM Benchmark Results

Payload Length (bytes)	Duration HW (ms)	Duration SW mbed TLS (ms)	Duration Improvement	Average Current HW (mA)	Average Current SW mbed TLS (mA)	Energy Efficiency Improvement
64	0.041	0.435	10.5	3.90	3.10	8.3
8000	1.198	32.4	27.0	2.00	3.10	41.9
16000	2.294	64.7	28.2	1.94	3.10	45.1

6.1.3 AES GCM

The AES GCM (see Reference [8]) benchmark consists of encrypting and authenticating messages of varying sizes and authenticating additional authenticated data (AAD) of a constant size. The duration of an AES GCM operation is governed by the key size, input length, and AAD length. We used a key length of 128 bits, a fixed AAD length of 32 bytes, and input lengths of 64, 8k, and 16k bytes. We chose a constant AAD length because the AAD usually contains metadata that does not scale in length with the message length.

Table 8. AES GCM Benchmark Results

Payload Length (bytes)	Duration HW (ms)	Duration SW mbed TLS (ms)	Duration Improvement	Average Current HW (mA)	Average Current SW mbed TLS (mA)	Energy Efficiency Improvement
64	0.036	0.533	14.8	3.94	3.10	11.6
8000	0.646	36.8	56.9	2.00	3.10	88.2
16000	1.174	73.3	62.4	1.94	3.10	99.8

6.1.4 AES CTR DRBG

AES CTR DRBG (see Reference [9]) is a deterministic random bit generator using the AES block cipher algorithm in counter mode. The AES CTR DRBG benchmark consists of three operations: initializing the internal state, reseeding the internal state, and producing a number of bytes of cryptographically random output. The most important of these operations is the byte generation as it is run far more frequently compared to initializing and reseeding an AES CTR DRBG instance.

The duration of these operations is governed by the key size and output length. We used a key length of 256 bits and an output length of 32 bytes. Unlike all the other AES benchmarks, we used a 256-bit AES key length to ensure that the AES CTR DRBG instance has a security strength of 256 bits. This is required to efficiently use the AES CTR DRBG output as a private key.

Initial seed entropy generation is not considered by the benchmark.

Table 9. AES CTR DRBG Benchmark Results

Operation	Duration HW (ms)	Duration SW mbed TLS (ms)	Duration Improvement	Average Current HW (mA)	Average Current SW mbed TLS (mA)	Energy Efficiency Improvement
Initialize	0.066	2.041	30.9	3.98	3.10	24.1
Reseed	0.047	1.019	21.7	3.38	3.10	19.9
Generate bytes (32)	0.072	0.227	3.2	3.63	3.10	2.7

6.1.5 SHA-224

SHA-224 (see Reference [4]) is one of the SHA-2 hash algorithms and has a hash digest size of 224 bits. The SHA-224 benchmark consists of hashing a variable length message in one go. The duration of a SHA-224 hashing operation is affected only by the input length. We used input lengths of 64, 8k, and 16k bytes.

Table 10. SHA-224 Benchmark Results

Payload Length (bytes)	Duration HW (ms)	Duration SW mbed TLS (ms)	Duration Improvement	Average Current HW (mA)	Average Current SW mbed TLS (mA)	Energy Efficiency Improvement
64	0.024	0.179	7.4	3.80	3.10	6.0
8000	0.258	10.4	40.2	2.87	3.10	43.4
16000	0.385	20.6	53.5	2.76	3.10	54.0

6.1.6 SHA-256

SHA-256 (see Reference [4]) is one of the SHA-2 hashing algorithms and has a hash digest size of 256 bits. The SHA-256 benchmark consists of hashing a variable length message in one go. The duration of a SHA-256 hashing operation is affected only by the input length. We used input lengths of 64, 8k, and 16k bytes.

Table 11. SHA-256 Benchmark Results

Payload Length (bytes)	Duration HW (ms)	Duration SW mbed TLS (ms)	Duration Improvement	Average Current HW (mA)	Average Current SW mbed TLS (mA)	Energy Efficiency Improvement
64	0.023	0.173	7.6	3.80	3.10	6.2
8000	0.261	10.4	39.6	3.00	3.10	40.9
16000	0.429	20.6	48.0	2.76	3.10	54.0

6.1.7 SHA-384

SHA-384 (see Reference [4]) is one of the SHA-2 hashing algorithms and has a hash digest size of 384 bits. The SHA-384 benchmark consists of hashing a variable length message in one go. The duration of a SHA-384 hashing operation is affected only by the input length. We used input lengths of 64, 8k, and 16k bytes.

Table 12. SHA-384 Benchmark Results

Payload Length (bytes)	Duration HW (ms)	Duration SW mbed TLS (ms)	Duration Improvement	Average Current HW (mA)	Average Current SW mbed TLS (mA)	Energy Efficiency Improvement
64	0.022	0.45	20.7	3.74	3.10	17.1
8000	0.200	26.9	134.3	3.51	3.10	118.6
16000	0.306	53.8	175.9	3.38	3.10	161.3

6.1.8 SHA-512

SHA-512 (see Reference [4]) is one of the SHA-2 hashing algorithms and has a hash digest size of 512 bits. The SHA-512 benchmark consists of hashing a variable length message in one go. The duration of a SHA-512 hashing operation is affected only by the input length. We used input lengths of 64, 8k, and 16k bytes.

Table 13. SHA-512 Benchmark Results

Payload Length (bytes)	Duration HW (ms)	Duration SW mbed TLS (ms)	Duration Improvement	Average Current HW (mA)	Average Current SW mbed TLS (mA)	Energy Efficiency Improvement
64	0.022	0.44	31.8	3.63	3.10	17.1
8000	0.200	26.9	213.1	3.56	3.10	116.7
16000	0.301	53.8	284.3	3.37	3.10	164.5

6.2 PKA Engine Based Drivers

6.2.1 ECDH

Elliptic Curve Diffie Hellman (ECDH) (see Reference [5] and Reference [10]) is used to establish a shared secret over an insecure channel. The ECDH benchmark consists of two operations: generating a public key from a private key and computing a shared secret. Private key generation is not considered by the benchmark. Both operations are dominated by the cost of an ECC scalar multiplication. Their differences lie in which public and private key is used and what type of parameter and curve validation is run.

The variable that affects performance for ECDH operations is the choice of elliptic curve. We have chosen to benchmark the NIST-P256 [10] and Curve25519 [11] curves.

Table 14. ECDH Benchmark Results

Operation	Curve	Duration HW (ms)	Duration SW mbed TLS (ms)	Duration Improvement	Average Current HW (mA)	Average Current SW mbed TLS (mA)	Energy Efficiency Improvement
Generate Public Key	NIST-P256	114.0	231	2.0	1.68	3.10	3.7
Compute Shared Secret	NIST-P256	114.3	668	5.8	1.69	3.10	10.7
Generate Public Key	Curve25519	54.9	585	10.6	1.68	3.10	19.6
Compute Shared Secret	Curve25519	54.9	620	11.3	1.68	3.10	20.8

6.2.2 ECDSA

Elliptic Curve Digital Signature Algorithm (ECDSA) (see Reference [12]) is used to asymmetrically sign and verify messages. The ECDSA benchmark consists of two operations: signing a hash of a message and verifying a hash of a message. Both operations are dominated by the cost of ECC scalar multiplications.

The benchmarks were run using the NIST-P256 curve. The ECDSA driver only supports Short-Weierstrass curves as Montgomery curve point addition is not available to implement ECDSA on Curve25519.

Generating the per-message secret number used to sign the message is not considered by the benchmark.

Table 15. ECDSA Benchmark Results

Operation	Duration HW (ms)	Duration SW mbed TLS (ms)	Duration Improvement	Average Current HW (mA)	Average Current SW mbed TLS (mA)	Energy Efficiency Improvement
Sign	115.7	269	2.3	1.68	3.10	4.3
Verify	230.7	943	4.1	1.67	3.10	7.5

6.2.3 ECJPAKE

ECJPAKE is the elliptic curve cryptography (ECC) (see Reference [13]) variant of the Password Authenticated Key Exchange by Juggling (J-PAKE) algorithm. The ECJPAKE benchmark consists of only one operation: running the entire key exchange. ECJPAKE is always used in its entirety and with a specific sequence of sub-operations. The benchmark therefore runs both the client and server sides of the ECJPAKE algorithm on the same device. The resulting time is then divided in half.

The benchmarks were run using the NIST-P256 curve. Private key and private v generation before round one of the exchange is not considered by the benchmark.

Table 16. ECJPAKE Benchmark Results

Duration HW (ms)	Duration SW mbed TLS (ms)	Duration Improvement	Average Current HW (mA)	Average Current SW mbed TLS (mA)	Energy Efficiency Improvement
1012.5	12485	12.3	1.67	3.10	22.9

6.3 TRNG Based Drivers

6.3.1 TRNG

The only factor influencing the duration of a TRNG operation is the amount of output bytes requested and the number of cycles the TRNG hardware is asked to sample the 24 free-running oscillators for before processing the result. The default setting is approximately 5 ms per 64 bits of output.

We requested 16 and 32 bytes to simulate generating the private key for a symmetric or asymmetric key.

On embedded platforms, sources of entropy usually require dedicated TRNG hardware. mbed TLS cannot reasonably provide a purely software based module that supplies reliable entropy. We have thus omitted the comparison between the TRNG drivers and mbed TLS for the TRNG benchmark.

Table 17. TRNG Benchmark Results

Length (bytes)	Duration HW (ms)	Average Current HW (mA)
16	9.980	2.20
32	19.920	2.19

7 Conclusion

Based on the benchmarks presented in this application report for the various cryptographic functions, with and without using cryptographic accelerators, it is evident that the cryptographic accelerators integrated in SimpleLink CC13x2/CC26x2 Wireless MCUs speed up cryptographic operations and enable energy-efficient security solutions.

With security for network connected devices being critical, most systems are required to support security solutions such as network communication security, secure boot, and secure firmware updates. Embedded systems designers should review their application security requirements and associated cryptographic functions to determine if cryptographic accelerators and their performance can help meet the latency and energy consumption requirements in their systems.

8 References

1. Secure Boot in SimpleLink™ CC13x2/CC26x2 Wireless MCUs, Texas Instruments
2. CC1352R SimpleLink™ High-Performance Multi-Band Wireless MCU data sheet, Texas Instruments
3. Advanced Encryption Standard (AES) (FIPS 197), NIST
4. Secure Hash Standard (SHS) (FIPS 180-4), NIST
5. Standards for Efficient Cryptography SEC 1: Elliptic Curve Cryptography, Certicom Research
6. Recommendation for Block Cipher Modes of Operation: Methods and Techniques (NIST SP 800-38A), NIST
7. Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality (NIST SP 800-38C), NIST
8. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC (NIST SP 800-38D), NIST
9. Recommendation for Random Number Generation Using Deterministic Random Bit Generators (NIST SP 800-90A), NIST
10. Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography (NIST SP 800-56A), NIST
11. Curve25519: new Diffie-Hellman speed records, Bernstein
12. Digital Signature Standard (DSS) (FIPS 186-4), NIST
13. Elliptic Curve J-PAKE Cipher Suites for Transport Layer Security (TLS), Cragie & Hao
14. Power Management for CC26xx SimpleLink Wireless MCUs User's Guide, Texas Instruments

Appendix: Plots of Blocking vs Polling Performance

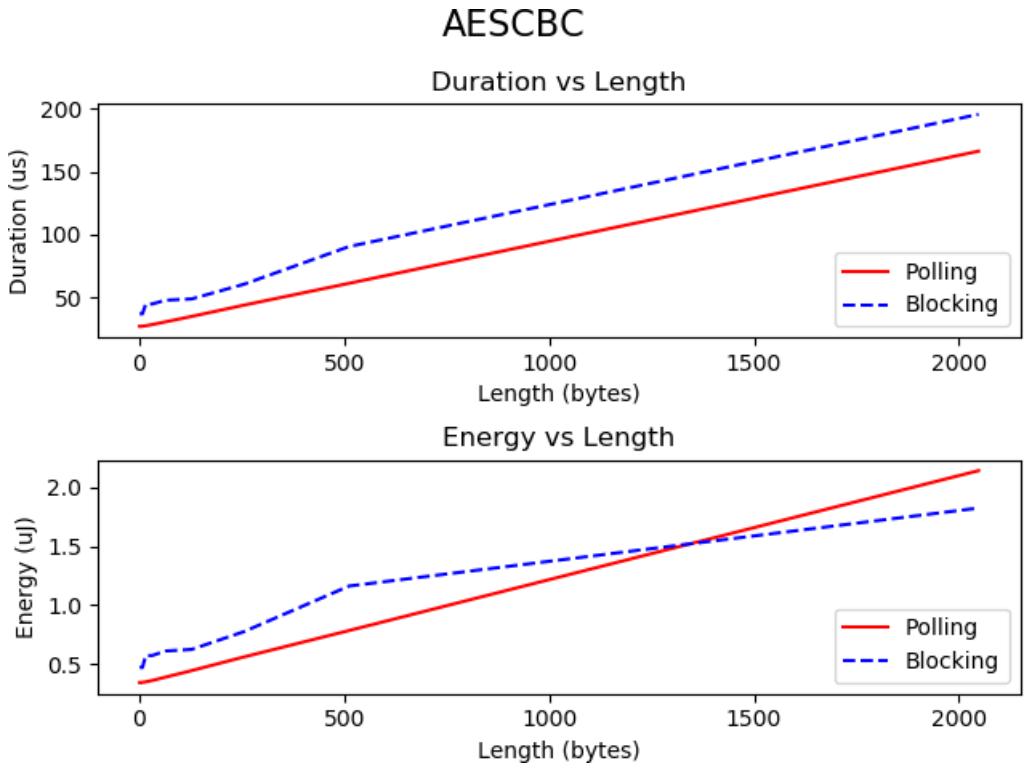

Figure 2. AES CBC Durations and Energy Consumption vs Message Length

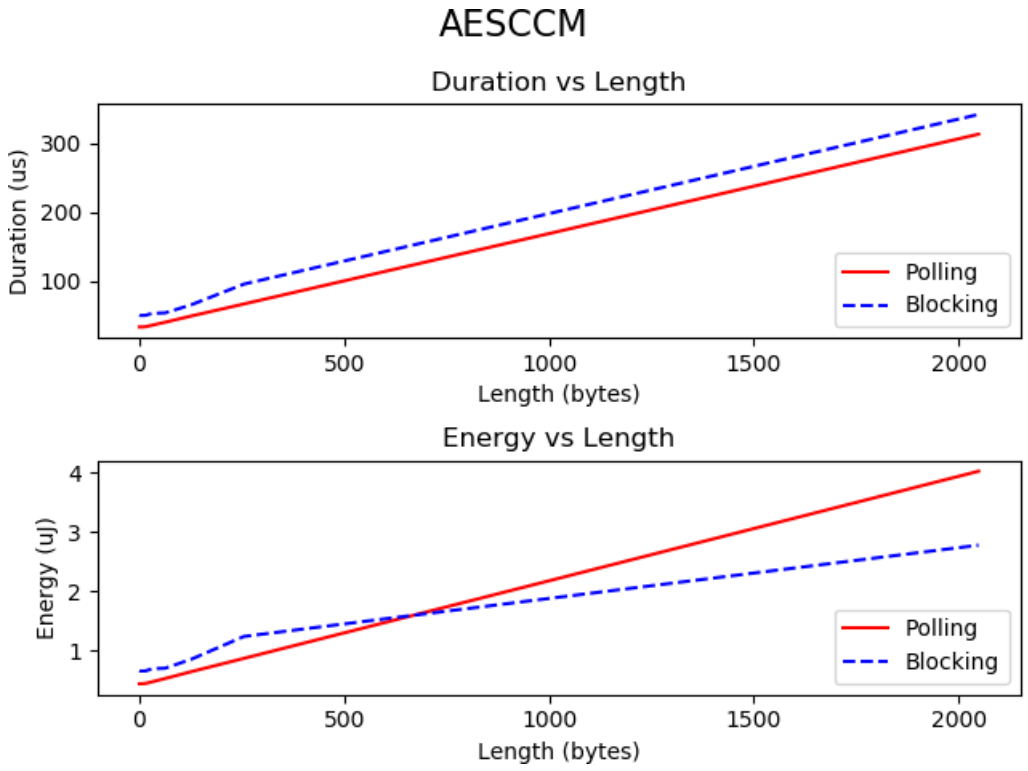

Figure 3. AES CCM Durations and Energy Consumption vs Message Length

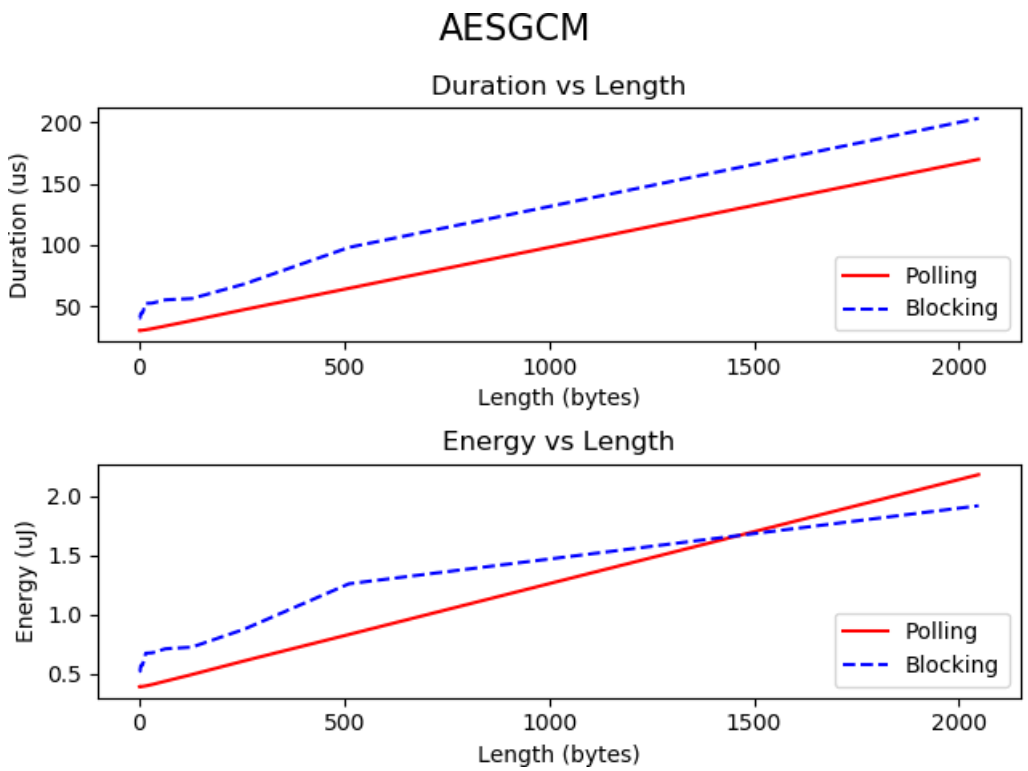

Figure 4. AES GCM Durations and Energy Consumption vs Message Length

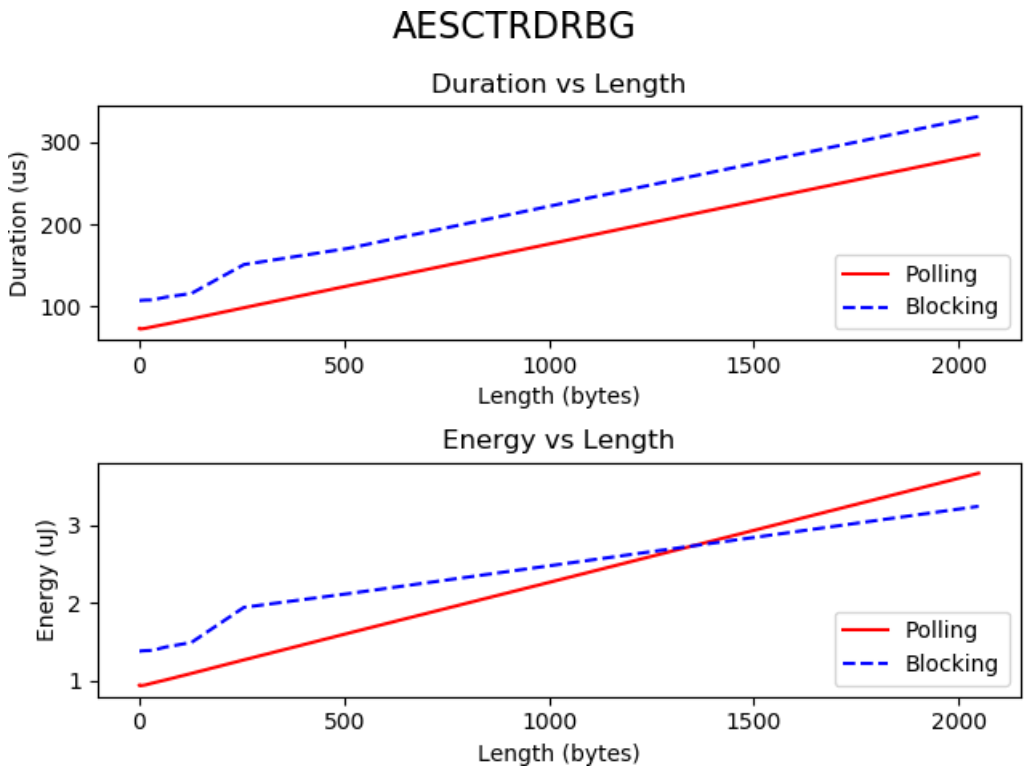

Figure 5. AES CTR DRBG Durations and Energy Consumption vs Message Length

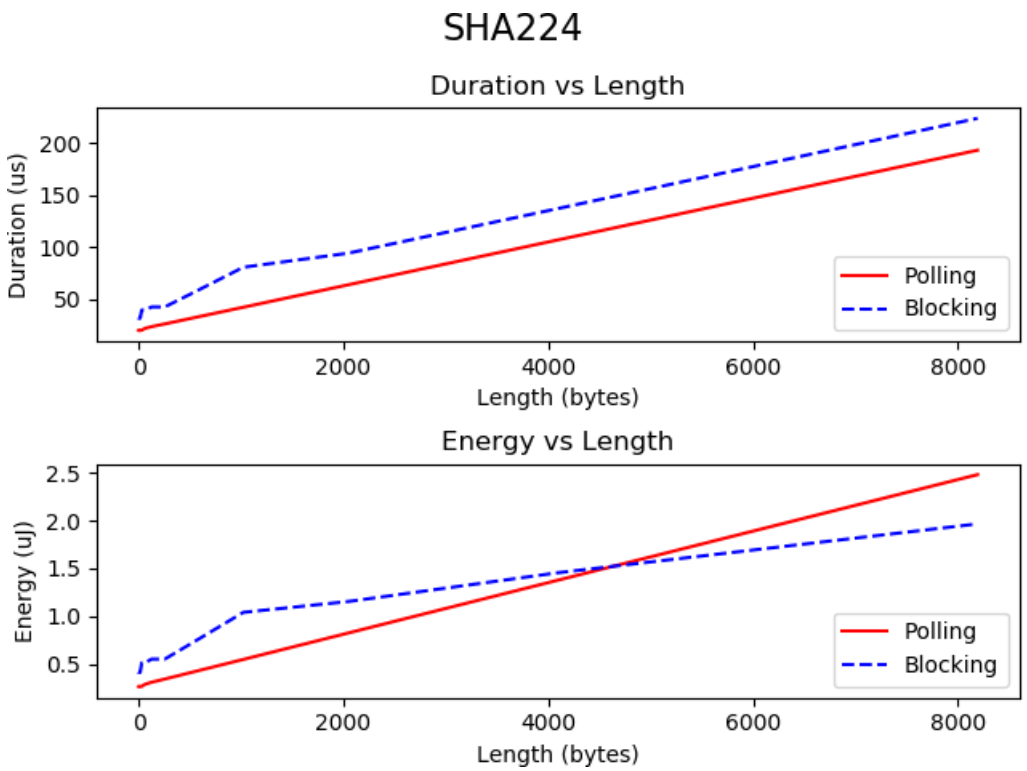

Figure 6. SHA-224 Durations and Energy Consumption vs Message Length

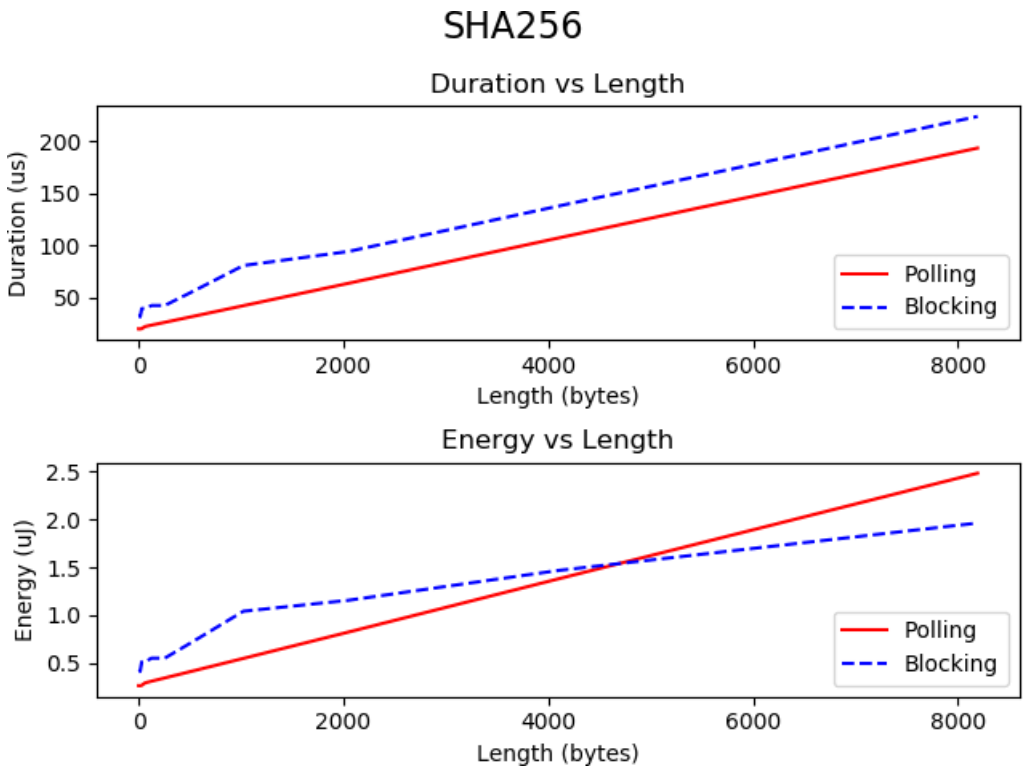

Figure 7. SHA-256 Durations and Energy Consumption vs Message Length

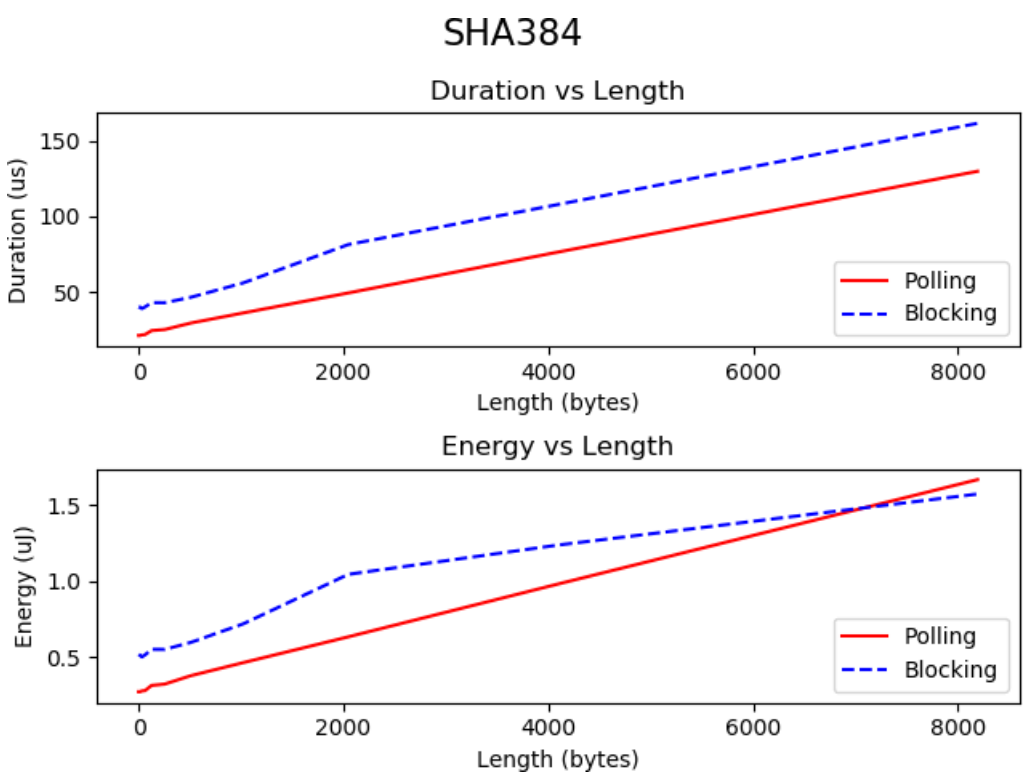

Figure 8. SHA-384 Durations and Energy Consumption vs Message Length

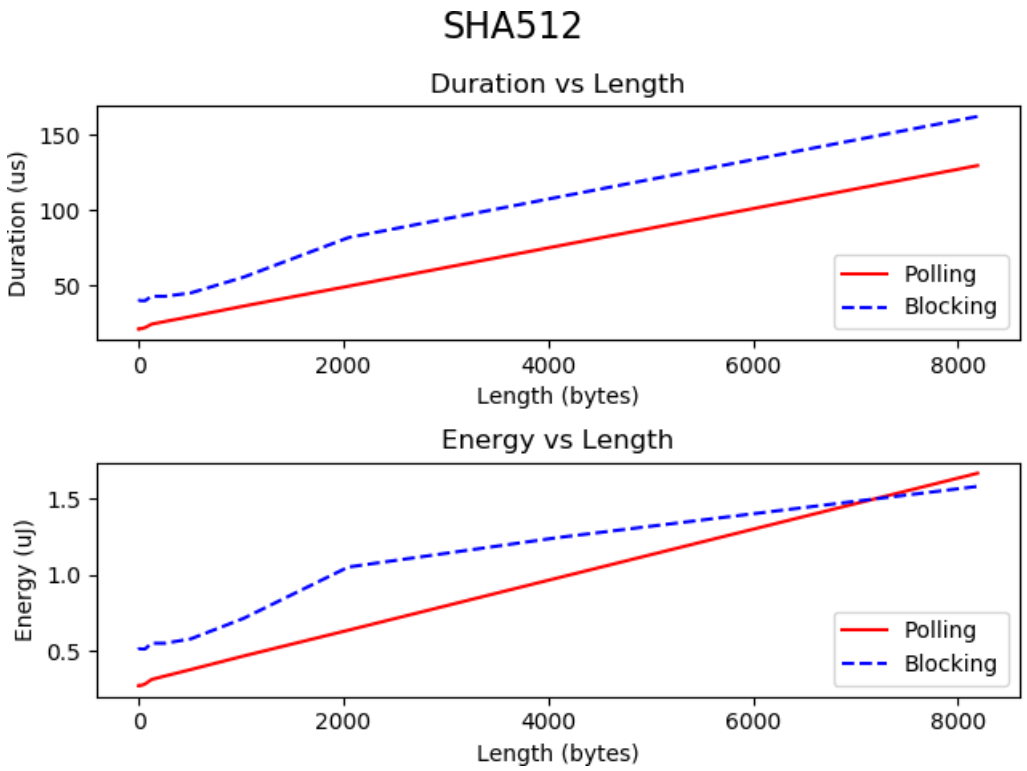

Figure 9. SHA-512 Durations and Energy Consumption vs Message Length

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265