Performance Evaluation of Elliptic Curve Libraries on Automotive-Grade Microcontrollers

Lucian Popa
lucian.popa@aut.upt.ro
Faculty of Automatics and Computers,
Politehnica University of Timisoara
Timisoara, Romania

Bogdan Groza
bogdan.groza@aut.upt.ro
Faculty of Automatics and Computers,
Politehnica University of Timisoara
Timisoara, Romania

Pal-Stefan Murvay
pal-stefan.murvay@aut.upt.ro
Faculty of Automatics and Computers,
Politehnica University of Timisoara
Timisoara, Romania

ABSTRACT

As cryptography is quickly entering the automotive domain, public-key cryptographic functions are a vital building block and are part of recent industry-proposed standards. Elliptic curves provide a more compact representation for public/private keys making them more suitable for embedded devices with limited amounts of memory. Nonetheless, they provide more compact signatures and open road for identity-based cryptographic primitives by exploiting the flexibility of bilinear pairings. In this work we carry a performance evaluation on some modern libraries, e.g., MIRACL, RELIC, and compare them to the more classical WolfSSL. The evaluation is carried on a state-of-the-art representative controller from the automotive industry, i.e., a 32 bit Infineon TC297. Having a crisper image on computational requirements is relevant for future automotive and industrial applications.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; • **Security and privacy → Software security engineering**; *Domain-specific security and privacy architectures.*

KEYWORDS

embedded systems, elliptic curve cryptography, short signatures

ACM Reference Format:
Lucian Popa, Bogdan Groza, and Pal-Stefan Murvay. 2019. Performance Evaluation of Elliptic Curve Libraries on Automotive-Grade Microcontrollers. In *Proceedings of ARES Conference (ARES'19).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Cars are vulnerable to well determined adversaries, as proved by recent research [8], [14]. Cryptography provides the main solution for mitigating such threats. This only complements the image since the industry has done lots of efforts in the previous decade to secure industrial networks, supervisory control and data acquisition systems (SCADA), distributed-control systems, etc.

ARES'19, August 26 - 29, 2019, University of Kent, Canterbury, UK

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The recent introduction of standards for cryptographic primitives in automotives [4], [5] is a clear statement that the industry is adopting cryptographic security for in-vehicle units. Moreover, adoption will be at a large scale and manufacturers will follow common design goals. But implementing cryptography on automotive-grade controllers is not so immediate due to high computation demands which become more problematic in case of public-key primitives.

In this paper we take into account elliptical-curve based cryptographic primitives (ECC) which form an indispensable building block for integrating security in modern cars or other systems that are build around embedded devices. There are numerous applications that can benefit from such primitives including key-exchange for in-vehicle or industrial controllers, car access control by using modern smart-phones, etc. We focus on two state-of-the-art libraries MIRACL [13] and RELIC [2] that have extensive support for elliptic curves and moreover, they have support for pairing-friendly curves that open road to compact signatures [7] and identity-based cryptographic primitives [6]. As expected, these libraries are contrasted with the more classical WolfSSL library [22] which is widely employed in the real-world. We find it appealing that the MIRACL [13] and RELIC [2] libraries include support for pairing-friendly curves that set room for the compact Boneh-Lynn-Shacham signature (BLS) [7]. This signature is the most compact signature known to this day as it has only 160 bits (this is half of a regular DSA signature that has 320 bits). Table 1 provides a summary for the libraries and functions that we evaluate. The Diffie-Hellman key exchange is evaluated in all three libraries: MIRACL, RELIC and WolfSSL. The BLS signature is only evaluated on MIRACL and RELIC since there is no implementation for BLS in WolfSSL. The DSA signature is evaluated only from WolfSSL and RELIC since for MIRACL we did not manage to adapt the code to work on the Infineon TriCore. Still, the image is comprehensive since our experiments give results for almost all curves that are present in these libraries and these are quite many.

The rest of the paper is organized as follows. In Section 2 we discuss related work that has already focused on evaluating embedded crypto-libraries and present our experimental setup. Section 3 makes an overview of the libraries and presents the computational results. Finally, section 4 holds the conclusion of our work.

2 RELATED WORK AND SETUP FOR EXPERIMENTS

In this section we discuss related work on evaluating performances of crypto-libraries and the we present the embedded platform that is target for our work.

Table 1: Summary of libraries and evaluated cryptographic primitives

Library	BLS Signature	DSA Signature	DH-Key Exchange
MIRACL [13]	✓	-	✓
RELIC [2]	✓	✓	✓
WolfSSL [22]	N/A	✓	✓

2.1 Related work

Pigatto et. al [16] have evaluated the time performance of MIRACL [13] and RELIC [2] using two types of curves (having key size of 160 and 256 bits) and different message lengths (50 KB and 100 KB) as inputs for the Elliptic Curve ElGamal encryption executed in Ubuntu Linux 11.04 operating system running on a 2.10 GHz Pentium Dual-Core. The encryption performed using 160 bit/256 bit key took 3.3 s/10.6 s with RELIC and 9 s/20.9 s when using MIRACL having as input a 50 KB message and 6.6 s/21.1 s with RELIC and 18.1 s/41.7 s when using MIRACL having as input a 100 KB message. It is quite obvious that performance characteristics of an Intel Pentium do not match our embedded platform so a comparison of the results would be out of scope.

Ruan de Clercq et. al [9] have implemented a cryptographic library on the ARM Cortex-M0+ evaluating fixed point multiplication (kG) and random point multiplication (kP) using the Koblitz NIST K-233 elliptic curve $y^2 + xy = x^3 + 1$ over the binary field defined by 2^{233}. The computational time for random point multiplication was of 59.18 ms and for fixed point multiplication was of 39.7 ms when testing their implementation compared with 117.1 ms and 115.7 ms when testing with RELIC [2] on the same curve. The random point multiplication required 2,814,827 clock cycles on the ARM Cortex-MO+ which has the base frequency of 48 MHz.

Hinterwälder et. al [12] have evaluated the power consumption of ECDH on the MSP430 microcontroller by testing different configurations (Karatsuba, Carry-save, Operand-caching) and using the Curve25519 elliptic curve $y^2 = x^3 + 486662x^2 + x$ over the prime field defined by $2^{255} - 19$ measuring the lowest power consumption of 14.046 μW using 32-bit operand-caching. The number of operations performed by MSP430 was of 6,513,011 clock cycles having the base frequency of 8 MHz so the ECDH using Curve25519 was completed in ~814 ms.

Previous efforts have been focusing on evaluating performances of cryptographic functions on automotive-grade controllers exist as well. In [15] an evaluation that takes into account AUTOSAR compliance is presented. However, the evaluation considers only symmetric cryptographic functions. More expensive, public-key primitives, including pairing-based primitives, have been accounted in [1]. This evaluation is done however on a distinct implementation from [20].

Zelle et. al [23] have analyzed the integration of TLS in Automotive Ethernet considering different types of certificates to be used with RSA-3072 or ECDSA-256 as asymmetric algorithms and (AES-128 CBC + SHA) or (HC-128 + Poly1305) as symmetric ciphers. The certificate generation and certificate-based encryption were implemented using wolfSSL [22] while the communication was tested between two Infineon TriCore TC297 based development boards. The time for ECDSA signing and verifying is, in average, 177 ms and 345 ms based on 100 trials. All the time measurements were analyzed in order to determine which is the maximum achievable data rate and the additional latency if transferring encrypted information considering small data types transferred in the in-vehicle networks but also big data types like sensor data or video streams.

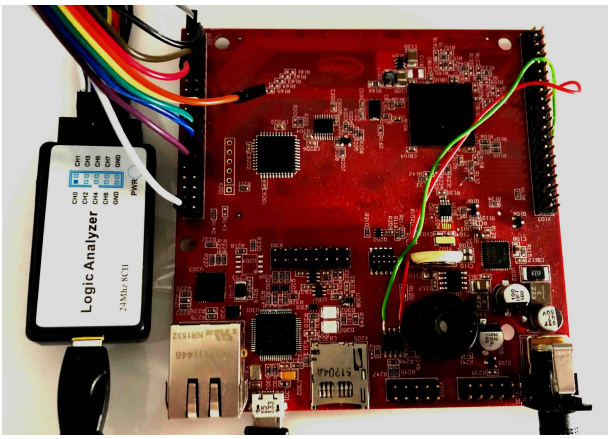

Figure 1: Experimental setup

2.2 Experimental setup and target platform

The large variaety of automotive applications, e.g., related to powertrain, body, chasis, etc., have led to the development of a great variety of automotive-grade microcontrollers that cover distinct needs of the market. Available automotive embedded platforms vary in a number of aspects, e.g., processor architecture, number of cores, operating frequency, available memory and on-chip peripheral modules.

For implementing security mechanisms the microcontroller's ability to implement and perform cryptographic operations is limited by its clock and available memory. Some platforms benefit from additional support provided by hardware security modules (HSM) for basic operations such as random number generation, hashing and encryption. However, this support is commonly limited to several standardized primitives, e.g., SHA-2 and AES, and require software implementations for any other cryptographic algorithm.

Not all automotive embedded platforms are capable of implementing public key cryptography. This is mainly due to the small amount of memory available (i.e., flash required for storing the program as well as RAM needed during execution which becomes more problematic for RSA-like functions that require a large modulus of 1024–2048 bits). Moreover, some of the platforms that are equipped with sufficient memory to support such implementations exhibit poor performance due to their simpler architecture or lower operating frequency.

For our work we considered high-performance 32-bit automotive microcontrollers and selected the Infineon TC297, a member of the Tricore Aurix family, as being representative for the high-end of the market. The TC297 is dedicated to high performance applications such as radar and camera systems for advanced driver assistance. It features three cores each capable of operating at up to 300MHz

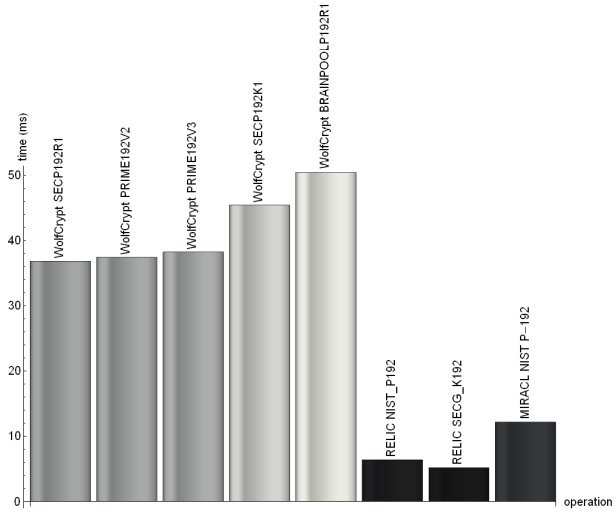

Figure 2: Barcharts of ECDH time for share generation

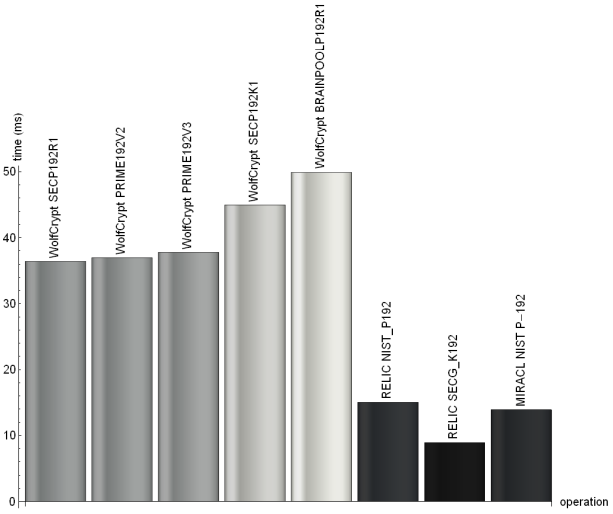

Figure 3: Barcharts of ECDH time for key recovery

as well as DSP functionality. The TC297 provides a total of 728 KB of RAM, 8 MB of program memory and 384 KB of EEPROM. The chip is also equiped with a HSM that provides a trusted execution environment complete with dedicated 32-bit CPU and protected memory. A true random number generator and hardware acceleration for AES-128 encryption are also available. Figure 1 presents our experimental setup and the Infineon Tricore board.

3 BENCHMARKED LIBRARIES AND RESULTS

In this section we give more insights on the libraries that we benchmark and then present the concrete experimental results.

3.1 Libraries and cryptographic primitives

We choose RELIC as one of our targets because it is a cryptographic library containing C implementations for many cryptographic algorithms such as RSA [17], the elliptic curve version of the Diffie-Hellman key exchange [10], SOKAKA [18], ECDSA [11] and BLS [7]. One of the many library's configurable properties is the digit size which can be set to match the CPU architecture enabling faster operation execution. We configured this parameter to 32 bits since we are using a 32 bit microcontroller, i.e. the TriCore TC297. For ECDSA we employed SHA-1, SHA-256 and also BLAKE2S-160, BLAKE2S-256 [3] for computing the hash value of the message to be signed. We have considered the following prime fields based on the curves used in our implementation:

- Prime field size (in bits) for BLS : 158, 256, 381, 382, 638, 1536
- Prime field size (in bits) for ECDH : 158, 160, 192, 221, 224, 226, 251, 254, 255, 256, 381, 382, 383, 384, 455, 477, 508, 511, 521, 638, 1536
- Prime field size (in bits) for ECDSA : 160, 192, 224, 256, 384

The MIRACL library contains C and C++ implementations for the various cryptographic algorithms contained. The BLS signature is implemented in C++ while the ECDH is implemented in C. MIRACL also allows the configuration of digit size which we set to 32 bits as in the case of RELIC. For ECDH we have used only the NIST P-192 as curve over prime field and for BLS we have used the following pairing-friendly curves:

- CP curve with Tate pairing embedding degree 2 and prime field size of 512 bits
- MNT curve with ate pairing embedding degree 6 and prime field size of 160 bits
- BN curve with ate pairing embedding degree 12 and prime field size of 256 bits
- KSS curve with ate pairing embedding degree 18 and prime field size of 512 bits

WolfSSL [22] represents an SSL library with implementation in C of SSL/TLS functions including support for server and client, various ciphers, key and certificate generation and revocation lists. WolfCrypt [21] is the library which contains the cryptographic primitives (hash functions, symmetric ciphers, public key algorithms) used by wolfSSL. It is also certified by the Federal Information Processing Standards (FIPS) 140-2, so it can be regarded as a standard library that is used in the industry. As configuration we used the default wolfCrypt setting for using 32-bit operations. For ECDSA we used SHA-1 and SHA-256 for hashing the message to be signed. We configured the library to generate ECC key pairs for ECDH and ECDSA using the following standardized curves :

- NIST Prime Curves : SECP192R1, PRIME192V2, PRIME192V3, PRIME239V1, PRIME239V2, PRIME239V3, SECP256R1
- SECP Curves : SECP112R1, SECP112R2, SECP128R1, SECP128R2, SECP160R1, SECP160R2, SECP224R1, SECP384R1, SECP521R1
- Koblitz Curves : SECP160K1, SECP192K1, SECP224K1, SECP256K1
- Brainpool Curves : BRAINPOOLP160R1, BRAINPOOLP192R1, BRAINPOOLP224R1, BRAINPOOLP256R1, BRAINPOOLP320R1, BRAINPOOLP384R1, BRAINPOOLP512R1

In order to evaluate the elliptic curve cryptographic capabilities of RELIC [2], MIRACL [13] and wolfCrypt [21] we considered the following primitives to be tested for all the mentioned libraries:

- BLS signature (except for wolfCrypt),
- the ECDH key agreement protocol,

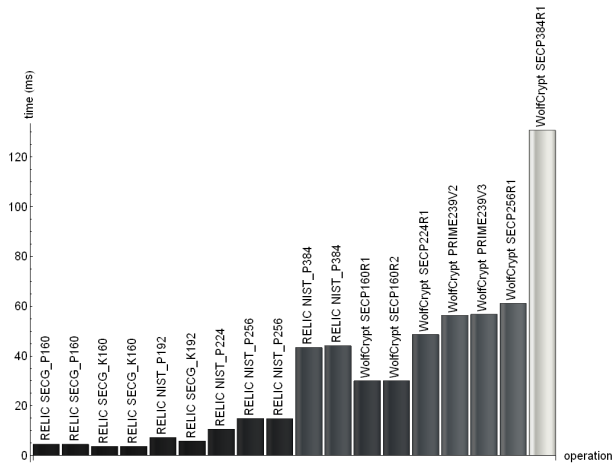

Figure 4: Barcharts of DSA time for key generation

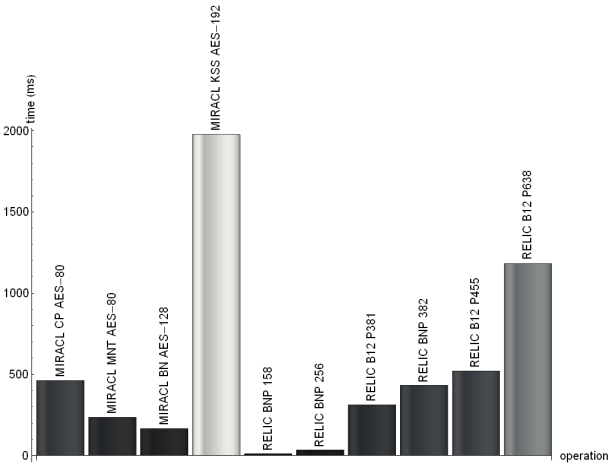

Figure 5: Barcharts of BLS time for key generation

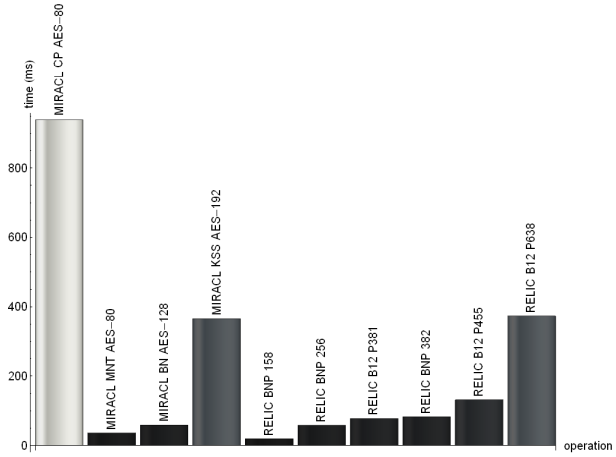

Figure 6: Barcharts of BLS time for signing

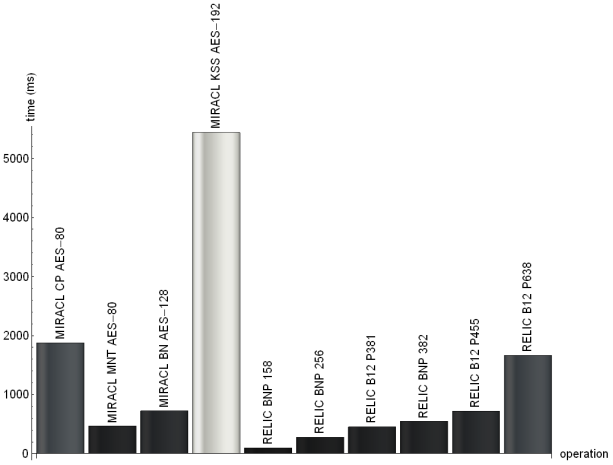

Figure 7: Barcharts of BLS time for signature verification

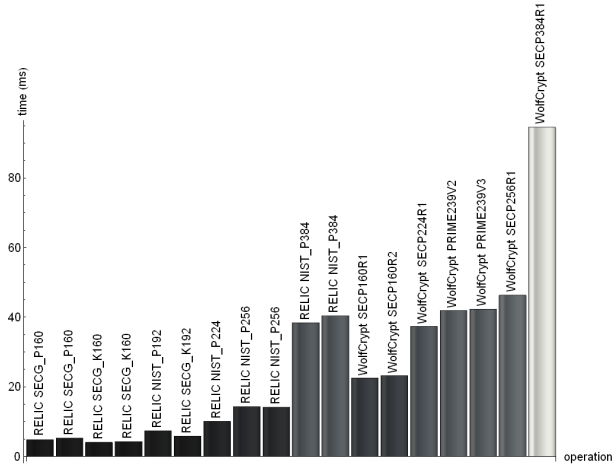

Figure 8: Barcharts of DSA time for signing

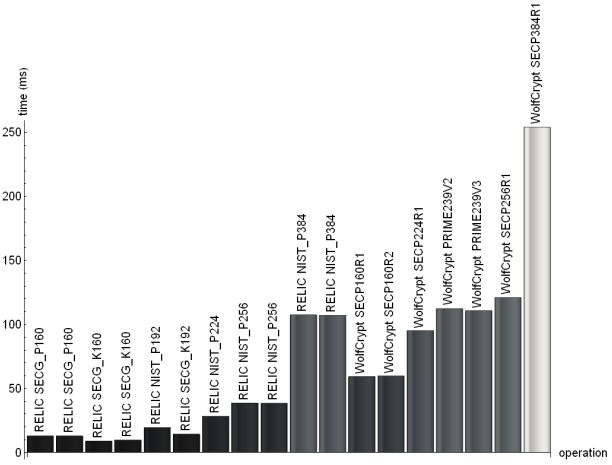

Figure 9: Barcharts of DSA time for signature verification

Table 2: Operations benchmarked for each cryptographic protocol

Security protocol	Key generation	Signature	Verification	Shared secret
BLS	✓	✓	✓	N/A
ECDSA	✓	✓	✓	N/A
ECDH	✓	N/A	N/A	✓

- ECDSA signature (except for MIRACL).

3.2 Experimental results

In this section we present the results of our measurements considering timing tests performed on all three libraries. As hardware equipment we have used the Infineon TC297 Application kit and an 8 channel logic analyzer. We built our applications based on the TriCore software framework by integrating the source code from the mentioned libraries. As cryptographic operations we have used all functions mentioned in Table 2.

Measuring the execution time of each operation was done using the Salae 1.2.18 logic analyzer connected to a general purpose I/O pin from the TC297 development board. The pin is configured as an output and its state is toggled at the start and end of each evaluated operation to generate a pulse width representing the execution duration.

For testing the cryptographic algorithms for BLS, ECDH and ECDSA we configured the prime field size in order to define the size of the resulting signature or keys.

In order to evaluate the ECDH key exchange we have tested all the available prime fields and associated curves with the purpose to measure and compare the timing performance for all three libraries. The number of elliptic curves which were tested were 52 from which 26 were available in wolfCrypt [21], 25 were available in RELIC [2] and 1 was provided in the main sample code from MIRACL [13] (more can be set manually). The prime fields which were available for wolfCrypt [21] were from 112 bits up to 512 bits, for RELIC [2] were from 158 bits up to 1536 bits and for MIRACL [13] 192 bits by considering the sample code. The full results are shown in Table 3 and Table 4 and a graphic summary is in Figures 2 and Figure 3. The RELIC [2] library was faster than wolfCrypt [21] and MIRACL [13]. The generation time of the key-pair and computation time of the shared secret over SECP192R1/NISTP192 was performed in 6.36 ms/15.06 ms for RELIC [2], 12.12 ms/13.9 ms for MIRACL[13] and 36.8 ms/36.4 ms for wolfCrypt [21]. When considering elliptic curves over larger prime fields, the generation and shared secret computation times for wolfCrypt [21] with the duration of 132.2 ms/131.5ms for SECP384R1/NISTP384 are quite high compared with RELIC [2] where we measured the duration of 37.44 ms/84.48 ms for these operations.

For BLS, the implementation is available in C++ for MIRACL [13] and in C for RELIC [2]. We considered as input a string with 20 bytes size for both MIRACL [13] and RELIC [2]. The execution time is graphically summarized in Figure 5, Figure 6 and Figure 7. The AES bits near the curve name represent the security level of the curve as depicted inside the MIRACL library. The complete data is in Table 5. Again, for RELIC [2] it took less time to execute the key generation, short signature and verification compared to

Table 3: Operation time for ECDH ordered by output size

Library	Elliptic curve	Output size	Operation type	Duration [ms]
wolfCrypt	SECP112R1	112 bits	generate	19.8
			compute shared secret	19.6
wolfCrypt	SECP112R2	112 bits	generate	23.7
			compute shared secret	23.8
wolfCrypt	SECP128R1	128 bits	generate	21.9
			compute shared secret	21.6
wolfCrypt	SECP128R2	128 bits	generate	27.6
			compute shared secret	27.2
RELIC	BNP158	158 bits	generate	3.29
			compute shared secret	6.01
wolfCrypt	SECP160R1	160 bits	generate	30
			compute shared secret	29.6
wolfCrypt	SECP160R2	160 bits	generate	30.2
			compute shared secret	29.8
wolfCrypt	SECP160K1	160 bits	generate	34.6
			compute shared secret	34.3
wolfCrypt	BRAINPOOLP160R1	160 bits	generate	37.5
			shared secret	37.2
RELIC	SECGP160	160 bits	generate	4.01
			compute shared secret	9.78
RELIC	SECGK160	160 bits	generate	3.27
			compute shared secret	5.78
wolfCrypt	SECP192R1	192 bits	generate	36.8
			compute shared secret	36.4
wolfCrypt	PRIME192V2	192 bits	generate	37.4
			compute shared secret	37
wolfCrypt	PRIME192V3	192 bits	generate	38.2
			compute shared secret	37.8
wolfCrypt	BRAINPOOLP192R1	192 bits	generate	50.4
			compute shared secret	49.9
wolfCrypt	SECP192K1	192 bits	generate	45.4
			compute shared secret	45
RELIC	SECGK192	192 bits	generate	5.15
			compute shared secret	8.94
RELIC	NISTP192	192 bits	generate	6.36
			compute shared secret	15.06
MIRACL	NISTP192	192 bits	generate	12.127
			compute shared secret	13.93
RELIC	CURVE22103	221 bits	generate	10.14
			compute shared secret	25.92
wolfCrypt	SECP224R1	224 bits	generate	48.8
			compute shared secret	48.4
wolfCrypt	SECP224K1	224 bits	generate	57.3
			compute shared secret	56.8
wolfCrypt	BRAINPOOLP224R1	224 bits	generate	62.8
			compute shared secret	62.4
RELIC	NISTP224	224 bits	generate	9.28
			compute shared secret	21.31
RELIC	SECGK224	224 bits	generate	7.71
			compute shared secret	13.01
RELIC	CURVE4417	226 bits	generate	12.73
			compute shared secret	31.84
wolfCrypt	PRIME239V1	239 bits	generate	56.6
			compute shared secret	56.2
wolfCrypt	PRIME239V2	239 bits	generate	55.7
			compute shared secret	55.2
wolfCrypt	PRIME239V3	239 bits	generate	56.1
			compute shared secret	55.7
RELIC	CURVE1174	251 bits	generate	13.88
			compute shared secret	35.27
RELIC	BNP254	254 bits	generate	10.52
			compute shared secret	17.54
RELIC	CURVE25519	255 bits	generate	14.72
			compute shared secret	36.7
wolfCrypt	BRAINPOOLP256R1	256 bits	generate	78.4
			compute shared secret	77.9
wolfCrypt	SECP256R1	256 bits	generate	60.3
			compute shared secret	59.8
wolfCrypt	SECP256K1	256 bits	generate	69.1
			compute shared secret	68.7
RELIC	NISTP256	256 bits	generate	13.04
			compute shared secret	29.48
RELIC	BSIP256	256 bits	generate	14.64
			compute shared secret	35.64
RELIC	SECGK256	256 bits	generate	10.36
			compute shared secret	18.07
RELIC	BNP256	256 bits	generate	10.52
			compute shared secret	18.22
wolfCrypt	BRAINPOOLP320R1	320 bits	generate	116.2
			compute shared secret	115.7

Table 4: Operation time for ECDH ordered by output size (continued)

Library	Elliptic curve	Output size	Operation type	Duration [ms]
RELIC	B12P381	381 bits	generate	18.45
			compute shared secret	62.06
RELIC	CURVE67254	382 bits	generate	41.72
			compute shared secret	101.33
RELIC	BNP382	382 bits	generate	29.88
			compute shared secret	50.23
RELIC	CURVE383187	383 bits	generate	41.15
			compute shared secret	102.8
wolfCrypt	SECP384R1	384 bits	generate	132.2
			compute shared secret	131.5
wolfCrypt	BRAINPOOLP384R1	384 bits	generate	168.4
			compute shared secret	167.9
RELIC	NISTP384	384 bits	generate	37.44
			compute shared secret	84.48
RELIC	B12P455	455 bits	generate	33.88
			compute shared secret	107.8
RELIC	B24P477	477 bits	generate	51.44
			compute shared secret	144.7
RELIC	KSSP508	508 bits	generate	48.02
			compute shared secret	128.75
RELIC	CURVE511187	511 bits	generate	91.09
			compute shared secret	221.7
wolfCrypt	BRAINPOOLP512R1	512 bits	generate	314.2
			compute shared secret	313.4
RELIC	SECP521R1	521 bits	generate	262.4
			compute shared secret	261.5
RELIC	NISTP521	521 bits	generate	94.28
			compute shared secret	208.22
RELIC	BNP638	638 bits	generate	115.62
			compute shared secret	194.76
RELIC	B12P638	638 bits	generate	83.72
			compute shared secret	136.05
RELIC	SSP1536	1536 bits	generate	327.24
			compute shared secret	4241.16

Table 5: Operation time for BLS ordered by output size

Library	Elliptic curve	Output size	Operation type	Duration [ms]
RELIC	BNP158	158 bits	generate	11.89
			sign	19.5
			verify	99
MIRACL	MNT	160 bits	generate	237
			sign	36
			verify	470
RELIC	BNP256	256 bits	generate	35
			sign	58.4
			verify	280
MIRACL	BN	256 bits	generate	166.5
			sign	59.2
			verify	726
RELIC	B12P381	381 bits	generate	313
			sign	78
			verify	456
RELIC	BNP382	382 bits	generate	434
			sign	83
			verify	548
RELIC	B12P455	455 bits	generate	522
			sign	132
			verify	722
MIRACL	CP	512 bits	generate	463
			sign	939
			verify	1878
MIRACL	KSS	512 bits	generate	1978
			sign	366
			verify	5443
RELIC	B12P638	638 bits	generate	1182
			sign	373
			verify	1664
RELIC	SSP1536	1536 bits	generate	4355
			sign	4854
			verify	7032

the time of using MIRACL [13]. Considering the prime fields of 256 bits we have measured the execution time for generation, signature and verification as 35 ms/58.4 ms/280 ms when using RELIC [2] compared with 166.5 ms/59.2 ms/726 ms when using MIRACL [13]. Also, there are more pairing-friendly curves available for BLS in RELIC [2] than in MIRACL [13].

The ECDSA was tested only on RELIC [2] and wolfCrypt [21] and we were able to test the signature using more than one hash function as it can be seen in Table 6. All the operations were executed faster in RELIC [2] than in wolfCrypt [21]. We have performed the tests over similar prime fields being able to compare the execution time which was, for example, in case of SECP256R1/NISTP256, 14.3 ms/38.6 ms when using BLAKE2S-256 and 14.1 ms/38.4 ms when using SHA-256 for RELIC [2] and 46.3 ms/121 ms when using SHA-256 for wolfCrypt [21]. The choice of the hash function has less impact since the elliptical curve operations are the most computational intensive. The total of 18 measurements is compared in Figures 4, 8 and Figure 9.

Table 6: Operation time for ECDSA ordered by output size

Library	Elliptic curve	Hash function	Output size	Operation type	Duration [ms]
wolfCrypt	SECP160R1	SHA-1	320 bits	generate	30
				sign	22.56
				verify	59.3
wolfCrypt	SECP160R2	SHA-1	320 bits	generate	30.1
				sign	23.2
				verify	59.8
RELIC	SECGP160	BLAKE2S-160	320 bits	generate	4.54
				sign	4.86
				verify	13
RELIC	SECGP160	SHA-1	320 bits	generate	4.54
				sign	5.27
				verify	13
RELIC	SECGK160	BLAKE2S-160	320 bits	generate	3.65
				sign	4.01
				verify	8.94
RELIC	SECGK160	SHA-1	320 bits	generate	3.65
				sign	4.26
				verify	9.78
RELIC	NISTP192	SHA-1	384 bits	generate	7.23
				sign	7.34
				verify	19.43
RELIC	SECGK192	SHA-1	384 bits	generate	5.81
				sign	5.83
				verify	14.37
wolfCrypt	SECP224R1	SHA-1	448 bits	generate	48.7
				sign	37.4
				verify	95.2
RELIC	NISTP224	SHA-1	448 bits	generate	10.6
				sign	10.1
				verify	28.31
wolfCrypt	PRIME239V2	SHA-1	478 bits	generate	56.3
				sign	41.9
				verify	112.4
wolfCrypt	PRIME239V3	SHA-1	478 bits	generate	56.8
				sign	42.3
				verify	110.7
wolfCrypt	SECP256R1	SHA-256	512 bits	generate	61.2
				sign	46.3
				verify	121
RELIC	NISTP256	BLAKE2S-256	512 bits	generate	14.9
				sign	14.3
				verify	38.6
RELIC	NISTP256	SHA-256	512 bits	generate	14.8
				sign	14.1
				verify	38.4
wolfCrypt	SECP384R1	SHA-256	768 bits	generate	130.8
				sign	94.6
				verify	254.1
RELIC	NISTP384	BLAKE2S-256	768 bits	generate	43.4
				sign	38.4
				verify	107.6
RELIC	NISTP384	SHA-256	768 bits	generate	44.1
				sign	40.4
				verify	107.1

Based on the results from our measurements, visible differences exist between the same operations performed using different libraries. From the execution time point of view we notice that RELIC [2] is the fastest library among the three evaluated libraries. However, wolfCrypt [21] has been FIPS 140-2 validated by security experts, and thus may be safer to integrate in applications from various industries (e.g., connected car systems) as a cryptographic library on numerous embedded devices in order to implement the standard security protocols. In case an application requires security protocols implemented in C++, MIRACL [13] is the only one of the benchmarked open-source projects which offers support for this object-oriented programming language. Moreover, MIRACL and RELIC have extensive support for pairing-friendly curves which is not available in wolfCrypt. Further optimizations may be possible as recently suggested in [19].

4 CONCLUSION

The high-end 32-bit automotive-grade Infineon TC297 was able to handle well all the ECC-based cryptographic primitives from the three state-of-the-art libraries: MIRACL, RELIC and WolfSSL. This proves that in-vehicle units are ready for adopting some of the most recent developments in the field of cryptography, e.g., pairing-friendly cryptographic operations. This enables them to implement secure interactions with other devices from the IoT ecosystem. Our measurements indicate RELIC [2] as the fastest and with the most configuration possibilities compared with MIRACL [13] and wolfCrypt [21]. Nonetheless, RELIC [2] and MIRACL [13] have extensive support for pairing-friendly curves. Still, wolfCrypt [21] is a more popular choice and may have been analyzed by a larger number of community members which may give additional security guarantees.

ACKNOWLEDGMENTS

We thank the reviewers for helpful comments that improved our work. This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS-UEFISCDI, project number PN-III-P1-1.1.-TE-2016-1317 (2018-2020).

REFERENCES

[1] Tudor Andreica, Bogdan Groza, and Pal-Stefan Murvay. 2018. Applications of Pairing-Based Cryptography on Automotive-Grade Microcontrollers. In *International Conference on Computer Safety, Reliability, and Security, SAFECOMP 2018*. Springer, 331–343. https://doi.org/10.1007/978-3-319-99229-7_28

[2] Diego F. Aranha and Conrado Porto Lopes Gouvêa. 2019. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic Accessed: 2019-02-22.

[3] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C-W Phan. 2008. SHA-3 proposal BLAKE. *Submission to NIST* 229 (2008), 230.

[4] AUTOSAR. 2015. Specification of Crypto Abstraction Library. In *AUTOSAR 4.2.2*. Foundation of Computer Science.

[5] AUTOSAR. 2015. Specification of Crypto Service Manager. In *AUTOSAR 4.2.2*. Foundation of Computer Science.

[6] Dan Boneh and Matt Franklin. 2001. Identity-based encryption from the Weil pairing. In *Annual international cryptology conference*. Springer, 213–229.

[7] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short signatures from the Weil pairing. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 514–532. https://doi.org/10.1007/3-540-45682-1_30

[8] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. 2011. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*, Vol. 4. San Francisco, USENIX Association, 447–462.

[9] Ruan de Clercq, Leif Uhsadel, Anthony Van Herrewege, and Ingrid Verbauwhede. 2014. Ultra Low-Power Implementation of ECC on the ARM Cortex-M0+. In *Proceedings of the 51st Annual Design Automation Conference (DAC '14)*. ACM, New York, NY, USA, Article 112, 6 pages. https://doi.org/10.1145/2593069.2593238

[10] Whitfield Diffie and Martin Hellman. 1976. New directions in cryptography. *IEEE transactions on Information Theory* 22, 6 (1976), 644–654. https://doi.org/10.1109/TIT.1976.1055638

[11] PUB FIPS. 2013. 186-4: Federal information processing standards publication. Digital Signature Standard (DSS). *Information Technology Laboratory, National Institute of Standards and Technology (NIST), Gaithersburg, MD* (2013), 20899–8900.

[12] Gesine Hinterwälder, Amir Moradi, Michael Hutter, Peter Schwabe, and Christof Paar. 2014. Full-size high-security ECC implementation on MSP430 microcontrollers. In *International Conference on Cryptology and Information Security in Latin America*. Springer, 31–47. https://doi.org/10.1007/978-3-319-16295-9_2

[13] MIRACL Ltd. 2019. Multiprecision Integer and Rational Arithmetic C Library – the MIRACL Crypto SDK. https://github.com/miracl/MIRACL Accessed: 2019-01-07.

[14] Charlie Miller and Chris Valasek. 2014. A survey of remote automotive attack surfaces. *Black Hat USA* 2014 (2014), 94.

[15] Pal-Stefan Murvay, Alexandru Matei, Cristina Solomon, and Bogdan Groza. 2016. Development of an AUTOSAR Compliant Cryptographic Library on State-of-the-Art Automotive Grade Controllers. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*. IEEE, 117–126. https://doi.org/10.1109/ARES.2016.60

[16] Daniel Fernando Pigatto, Natássya Barlate Floro da Silva, and Kalinka Regina Lucas Jaquie Castelo Branco. 2011. Performance evaluation and comparison of algorithms for elliptic curve cryptography with El-Gamal based on MIRACL and RELIC libraries. *Journal of Applied Computing Research* 1, 2 (2011), 95–103. https://doi.org/10.4013/jacr.2011.12.04

[17] Ronald L Rivest, Adi Shamir, and Leonard Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 21, 2 (1978), 120–126. https://doi.org/10.1145/357980.358017

[18] R Sakai, K Ohgishi, and M Kasahara. 2000. Cryptosystem based on Pairings. *Symposium on Cryptography and Information Security* 45 (01 2000), 26–28.

[19] Michael Scott. 2019. Pairing Implementation Revisited. (2019). https://eprint.iacr.org/2019/077.pdf

[20] Thomas Unterluggauer and Erich Wenger. 2014. Efficient pairings and ECC for embedded systems. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 298–315. https://doi.org/10.1007/978-3-662-44709-3_17

[21] wolfSSL Inc. 2019. wolfCrypt Embedded Crypto Engine. https://www.wolfssl.com/products/wolfcrypt/ Accessed: 2019-03-10.

[22] wolfSSL Inc. 2019. wolfSSL embedded SSL library. https://www.wolfssl.com/products/wolfssl/ Accessed: 2019-03-10.

[23] Daniel Zelle, Christoph Krauß, Hubert Strauß, and Schmidt Karsten. 2017. On Using TLS to Secure In-Vehicle Networks. In *ARES '17 Proceedings of the 12th International Conference on Availability, Reliability and Security, Article No. 67*. ACM. https://doi.org/10.1145/3098954.3105824