Ullah and Zahilah *Cybersecurity* (2021) 4:11
https://doi.org/10.1186/s42400-021-00078-6

Cybersecurity

RESEARCH Open Access

Curve25519 based lightweight end-to-end encryption in resource constrained autonomous 8-bit IoT devices

Shafi Ullah* and Raja Zahilah

Abstract

Robust encryption techniques require heavy computational capability and consume large amount of memory which are unaffordable for resource constrained IoT devices and Cyber-Physical Systems with an inclusion of general-purpose data manipulation tasks. Many encryption techniques have been introduced to address the inability of such devices, lacking in robust security provision at low cost. This article presents an encryption technique, implemented on a resource constrained IoT device (AVR ATmega2560) through utilizing fast execution and less memory consumption properties of curve25519 in a novel and efficient lightweight hash function. The hash function utilizes GMP library for multi-precision arithmetic calculations and pre-calculated curve points to devise a good cipher block using ECDH based key exchange protocols and large random prime number generator function.

Keywords: Cyber-physical systems, IoT, Resource constrained IoT devices, Lightweight encryption, End-to-end encryption, Elliptic curve cryptography, Curve25519

Introduction

Useful tiny autonomous IoT machines that work endlessly, are used in our daily lives such as smart refrigerator, smart air-condition, smart TV, smart health devices and smart parking. Number of such devices are estimated to reach 50 Billion by 2025 (Song et al. 2017). Most devices share sensitive information over the internet with each other via pre-programmed tasks and make critical and nearly accurate decisions when required. These devices are usually cheap, tiny and retain little computational and memory capabilities, known as IoT devices, machine type communication (MTC) devices and cyber physical systems (CPS).

The data shared among such devices can be extremely sensitive, both for user and operational purposes. Since the devices do not possess security of their own, it is provided either externally or physically by embedded applications (which is not feasible for all devices) within the entire network as the purpose of such devices is to work remotely and autonomously with constrained resources (Rajesh et al. 2019). However, due to low computation and memory capabilities, standard security protocols (as of internet) cannot be applied (Song et al. 2017). Consequently, the devices are only equipped with software-based security in a network for secure operational communication applications. The data shared between the devices must be integrated in its original form without mutation and must also maintain secrecy of the device that sends data. Whereas a network can contain numerous connected devices and every data block that is transmitted from a device, must ensure integrity of data. It is achieved by applying end-to-end encryption.

We propose curve25519 based end-to-end encryption with a novel lightweight hash function. The proposed elliptic curve cryptography (ECC) based end-to-end encryption method enables computational and memory exhaustive IoT devices to communicate securely with

* Correspondence: ullah.shafi@graduate.utm.my
School of Computing, Faculty of Engineering, Universiti Teknologi Malaysia, 81310 Johor Bahru, Johor, Malaysia

efficient memory consumption and affordable computation because of curve25519 that possess strong security (Bernstein 2006) and occupies comparatively smaller key sizes in memory. While most ECC based encryption techniques utilize RSA, AES and Elgamal (Banerjee and Patil 2018) type hash functions to encrypt data based on a private keys and consequently resulting in exhaustive computational power, our proposed method introduces a pre-calculated curve points strategy that proves to be more efficient in computational performance and memory consumption. Moreover, it retains strong 192-bit security with 128-bit encrypted keys. The proposed method is tested on an AVR ATMega2560 microcontroller with ECDH based 128-bit secret keys. Furthermore, the use of GMP library improved scaler multiplication performance and large prime number generation costs. The proposed technique is applicable on all AVR machines with minimum 32Kbytes of flash memory.

Related work on ECC

Elliptic curve over a finite field F is a one-way function and its' inverse is incomputable (Moosavi et al. 2018a). An elliptic curve over a field is denoted by $Fg(a, b)$ where g is a prime number where $g > 3$. g represents x, y set of coordinate points such that $g = (x, y)$ where $(x, y \in Fg(a, b))$ that satisfies the equation $y^2 = x^3 + ax + b \pmod{p}$ where $0 < = x < = p$. Size of p in a curve, governs the essence of n-bit security. The performance efficiency of ECC relies on computing a fixed-point or scalar-point multiplication, given the product points x, y. Performance cost of scaler multiplication mainly depends on two curve operations i.e., PA (Point Addition) and PD (Point Doubling). PA refers to the addition of two similar curve points (x_1, y_1) and (x_2, y_2) on curve in a field f such that $x_1 = x_2$ and $y_1 = y_2$. Whereas PD refers to a Double-&-Add algorithm, used for two different points on curve such that $x_1 \neq x_2$ *and* $y_1 \neq y_2$. The resulting curve point from PA and PD will be a random point on curve that cannot be calculated reversely. Additionally, PA and PD operations cost computational power if large prime numbers are used. Furthermore, ANSI X9.63 and IEEE P1363 provide the public key cryptography (PKC) standards for ECC based lightweight cryptography. These standards offer randomness and robustness per bit compared to the current PKC encryption schemes based on RSA, DES and Elgamal.

Düll et al. (2015) initiated ECDH (Elliptic Curve Diffie-Hellman) key exchange protocol by applying a fixed basepoint and a variable curve point on AVR ATmega, MSP-430X and ARM Cortex-M0 microcontrollers using curve25519. The variance in bandwidth upgraded the robustness of curve including reduction in cost of clock cycles by 18%. Previous methods applied by Liu et al. (2014), Hutter and Schwabe (2013), Hinterwälder et al. (2014), De Clercq et al. (2014), Wenger et al. (2013), Gouvêa et al. (2012), Aranha et al. (2010) and Gura et al. (2004), utilized identical hardware configured IoT devices with different types of elliptic curves where each type resulted in different computational cost and memory consumption. However, the results indicated that these resource constrained computing capable IoT devices handled encryption computational tasks in reasonable time and inexpensive memory. Oualha and Nguyen (2016) applied pre-computation techniques using ECC by discarding scalar point multiplication tasks and resulted in consuming lesser energy. While the technique is computationally efficient but it demanded for large memory. Fujii and Aranha (2018) also instigated curve25519, targeted for ARM Cortex-M4 type microcontrollers that relied on a digital signature (ECqDSA). Moosavi et al. (2018b) worked on developing a refined IoT network structure to achieve optimal security in a three-layer architecture with ECC centered private key generation on random time points using ECG of a patient by calculating difference in two crusts with combination of Fibonacci linear feedback shift register. Numerous applications of ECC discussed in Liu and Ning (2008), used AVR ATmega and targeted low costs with high security. Whereas TinyECC library applied ECDSA, ECDH, and ECIES for 128, 160 and 192-bit security using SECG curves, respectively. Szczechowiak et al. (2008) applied another similar and smaller type of curve called Nano-ECC and used NIST K-163 curve. Though improving cost efficiency in terms of computation and memory, is still a challenging task as recent ECC applications on AVR ATmega machines applied comparatively unsafe curves. Another type of twisted Edwards curves presented in Chu et al. (2013), applied 80-bit and 96-bit security levels for performance enhanced implementation of ECC. De Santis and Sigl (2016) used implemented × 25519 curve on ARM Cortex-M4 device to improve key generation computation. Similarly, Fujii and Aranha (2018) aimed to minimize modular multiplication by 50% on ed25519 curve using ARM Cortex-M4 device. Moreover, a software implementation on Intel's AVX2 Haswell processors which are equipped with vector instruction set architecture for prime number arithmetic in ECC operations (Faz-Hernández et al. 2019). They computed digital signatures algorithms with reduction of 19% and 29% for ed25519 and ed448 in comparing with execution time of × 25519 and × 448, respectively.

Proposed curve25519 based lightweight encryption in resource constrained IoT devices

In the context of Related work on ECC section, resource constrained IoT devices still lack in efficient encryption

algorithm that can deliver robust security against powerful computational statistical attacks. We propose a novel approach in implementation of end-to-end encryption in this section. The section also describes related functional parts of our technique. An example illustrated in Fig. 1 elaborates proposed novel hash function and a pseudo code of encryption and decryption procedures is also presented in Algorithm 1&2, respectively.

The proposed method implements curve25519 based encryption on an AVR ATMega2560 machine through a pre-calculated curve points strategy. The method is designed to accommodate 46 characters (a-z, 0–9 and special characters) to minimize memory usage. Each character (*ci*) is allocated a unique large random prime number (*LRPN*), considered as a generator number point (G) on curve. A generator number point (G) represents the number of times a curve performs PA and PD curve operations from a starting point 1G (i.e., 9x,6278y in our technique). Thus, each character has a particular generator number representing unique curve points. These curve points are then fed to a hash function that encrypts the characters. Moreover, every proceeding character will influence the next character's curve points during encryption, explained briefly in Hash function of the proposed method section.

GMP (GNU multiple precision) arithmetic library

32–64-bit large prime numbers and signed integers have limited compiler support in constrained IoT devices. To use minimum code size, we adopted GMP library that has embedded support libraries for arithmetic operations of 32–64 bits large-signed integers. GMP library is freeware and used for arbitrary multiple precision based arithmetic calculation on singed integers, floating points and rational numbers. It is equipped with fast and precision capable algorithms. Furthermore, it supports C and C++ platform with most inner loops coded in assembly language (depending on CPUs) to deliver speedy calculations.

We used gmp-6.2.0 version that comes with a mini-gmp sub library. This sub library can be used for machines from AVR family to use minimum code size and functionality for all arithmetic operations related to elliptic curves.

Elliptic curve (Curve25519)

Adoption of a particular form of curve is very crucial in elliptic curve cryptography. The proposed technique targets a curve that should perform speedy calculations and consume the least possible memory in addition to robust encryption. In this regard, we chose curve25519 as it retains our targeted features during the curve operations. The curve25519 is a 255-bit based curve with preceding standard curves (NIST P-256 and NIST K233 of 128-bit) that offers up to 256-bit key security with fastest ECC curves. The curve was introduced by Bernstein (2006) and designed particularly for ECDH key agreement schemes. The research in Bernstein (2006), implemented the curve on a public domain software using Intel Pentium III, 4, M and AMD Athlon CPUs and concluded that the curve is fast, retains short secret and public keys, and is free of key validation. Another research in Bernstein and Lange (2014), represented a thorough security and performance analysis of several ECC based curves where the curve25519 is said to gain indistinguishable security and showed resilience against ladder attack, side channel attack, twist attack and exhibited completeness from random strings attacks. Thus, it is considered a safe curve as it repels all known curve attacks.

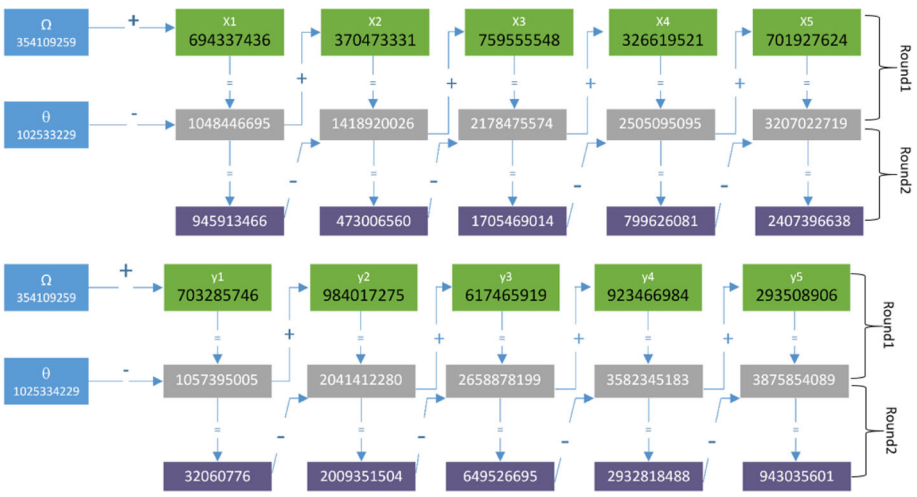

Fig. 1 Hash function mutation of curve points (example)

Curve25519 equation

The curve equation $y^2 = x^3 + 486662x^2 + x \ mod(p)$ is a Montgomery curve (Montgomery 1987) over *Fg* prime field where *mod* (*p*) is defined by $3 < p \leq 2^{255} - 19$, with a based-point of $x = 9$. The curve also uses compressed x-axis coordinate points to allow use of Montgomery ladder in utilizing X, Z coordinates. In our work, we used three (32,43 and 62-bit) *p* values and secret key sizes at 30,43 and 51-bit to limit cost affordability of computation and memory.

Secret key (*SK*) generation

Secret key (*SK*) plays vital role in the proposed hash function. *SK* is primarily adopted to further mutate curve points to improve encryption robustness. It is generated during runtime; hence size of the secret key must be considerably kept as a tradeoff between robustness and computational efficiency. We tested three *SKs* for every *mod*(*p*) as shown in Table 1. Additionally, *SK* can be generated by any of the following two means

1. *SK* can be resultant key provided by ECDH based key exchange protocols.
2. *SK* can be generated via large prime positive integer through random number generator function (RNGF) with seed value applied from a built-in timer function in AVR Atmega2560.

SK in Table 1 is generated by ECDH based key exchange protocols, treated as a starting generator value (1G) with specific curve points. The *SK* generator's execution cost is evaluated through curve operations (PA & PD) during scalar multiplication. It is an efficient way to calculate large curve points on a curve (i.e. If a bit is 1 then PA is applied, and PD is applied if the bit is 0). Similarly, the number of operations (i.e., PA and PD) depends on number of generator (G) bits.

Table 1 shows several *SKs*, generated against three *mod*(*p*) values on runtime in constrained IoT device where scalar multiplication and computational costs are also mentioned. The execution time (ET) increases with larger *mod*(*p*) value due to increase in the curve operations. Note that PD has lesser arithmetic operations than PA. We calculated that PA takes 1.5-2x more execution time than PD.

Reference table (RFTAB)

Reference table (RFTAB) contains the list of ASCII characters which will be used to encrypt message where each character contains a unique curve point. Unlike secret key (*SK*), the corresponding curve points in RFTAB are pre-calculated to limit runtime computation. The proposed method utilizes only 46 ASCII characters in RFTAB including a-z, 0–9 and 10 special characters. Other ASCII characters such as capital letters and special characters are excluded to reduce memory usage and simplify messages. Every character is assigned a large random prime integer which is considered as a generator curve point (G) (as shown in Table 2). These curve points are pre-calculated through scalar multiplication starting from a pre-set generator value (1G). The starting point (1G) can also be set as a network dependent generator number, a public key or specified through constant change on certain timestamps. Furthermore, the RFTAB is flexible enough to incorporate all ASCII and Unicode characters depending upon developers.

Table 2 represents a unique generator number, pre-calculated through large random prime integer function for every ASCII character. Generator numbers are further applied on a curve to calculate its corresponding unique (x, y) curve points, generated from a starting curve point at 1G (9x, 6248y) with *mod* (*p*) = 1019532643. Note that the starting curve point (1G) and *mod*(*p*) are developer dependent values and must be

Table 1 Secret key generation performance in the proposed technique

mod(p)	G	*SK*		*SK* size	PA	PD	ET (ms)	E
		x	y					
30-bit	484,924,606,752,301	826,625,966	80,435,164	30-bit	25	48	1087	0.1087
	2,938,674,132,359,780,000	136,996,696	68,348,709	28-bit	31	62	1406	0.1406
	160,949,512,909,771	497,086,896	418,609,960	29-bit	24	48	1062	0.1062
43-bit	484,924,606,752,301	198,854,107,681	1,226,705,872,672	38-bit	25	48	1891	0.1891
	2,938,674,132,359,780,000	1,323,613,420,771	2,788,657,789,809	42-bit	31	62	2349	0.2349
	160,949,512,909,771	2,741,618,876,124	974,370,806,151	42-bit	24	48	1811	0.1811
62-bit	484,924,606,752,301	775,724,248,908,631,391	3,753,518,844,464,014,710	62-bit	25	48	2623	0.2623
	2,938,674,132,359,780,000	685,910,647,662,771,392	3,518,772,554,951,183,129	61-bit	31	62	2907	0.0290
	160,949,512,909,771	1,295,578,424,185,598,538	933,837,696,196,791,576	60-bit	24	48	2523	0.2523

PA Point addition, *PD* Point doubling, *ET* Execution time, *E* Energy consumption

Table 2 Reference table in proposed hash function

ci	ASCII	*LPRN.G*	Curve Points (x, y)
1	a	1114995945582353G	694,337,436, 703,285,746
2	b	288921697093859G	370,473,331, 984,017,275
3	c	276838389415067G	759,555,548, 617,465,919
4	d	675280302129893G	658,744,261, 842,278,155
5	e	936876179746217G	183,110,606, 110,227,174
6	f	41384689890461G	179,809,473, 782,626,044
7	g	439822307637991G	189,560,000, 939,147,020
8	h	701422480221611G	920,636,675, 800,025,404
9	j	770835426621577G	73,536,773, 384,990,209
10	k	497156241326461G	21,051,887, 664,479,360
11	l	285860567224847G	114,911,791, 95,276,802
12	m	410623294644557G	370,347,429, 177,948,275
13	n	852327385733039G	207,177,991, 1,001,485,169
14	o	441810755274013G	435,796,402, 461,746,467
15	p	230515081172399G	369,263,203, 616,366,034
16	q	945894062588701G	449,183,440, 200,940,412
17	r	535381727096971G	548,976,276, 16,443,194
18	s	796981899680591G	619,312,283, 712,899,022
19	t	163086287441159G	402,748,439, 455,269,340
20	u	26248842277249G	18,422,723, 6,605,984
21	v	14169829565753G	903,414,998, 482,785,657
22	w	138932556985463G	605,945,868, 891,747,116
23	x	2095111821553G	325,942,946, 742,935,628
24	y	779857604431357G	863,486,555, 856,002,957
25	z	400528434601787G	498,942,518, 278,185,069
26	0	83582775965719G	269,100,618, 153,886,684
27	1	892541319139571G	82,566,882, 973,636,287
28	2	755695284041069G	1,010,214,883, 564,478,933
29	3	133882979480429G	225,964,606, 644,839,968
30	4	849266255864027G	558,176,893, 301,038,469
31	5	165074735077181G	326,619,521, 923,466,984
32	6	750654296470627G	2,978,693, 455,928,598
33	7	23192007375533G	106,943,196, 864,459,202
34	8	769762744388587G	701,927,624, 293,508,906
35	9	147950439827947G	222,120,950, 201,464,481
36	@	851254703500049G	647,033,442, 905,238,490
37	#	534309044863981G	735,000,937, 247,572,192
38	:	889480189270559G	653,569,343, 260,941,402
39	^	1052464107493483G	303,172,155, 284,207,106
40	&	20130877506521G	8,074,164, 890,475,165
41	*	735514153890119G	915,720,122, 487,163,851
42	.	629868464322961G	208,512,143, 934,037,045
43	,	723439436145919G	23,290,126, 528,271,011
44	_	89543823906487G	413,027,496, 974,898,543

Table 2 Reference table in proposed hash function *(Continued)*

ci	ASCII	*LPRN.G*	Curve Points (x, y)
45	–	171035783017949G	377,335,563, 513,671,840
46	space	34198337854039G	438,328,264, 902,839,215

Points are generated with *mod*(*p*) = 1019532643
Points are generated from a starting curve point of 1G i.e. 96,248

kept secret. Moreover, change in single bit of starting curve point will result in totally different values in Table 2.

Hash function of the proposed method

Hash function includes a secret key (*SK*) described either from a random number function of hardware machine (discussed in Security analysis section) or from resultant key of ECDH key exchange protocols. We assume generating *SK* from random number generator. The random number generator function takes seed value provided by a timer. The timer starts when the hardware is powered on and the value is taken in microseconds. In the example, the random number function outputs 484,924, 606,752,301 value which is considered as a *SK* generator value (G). Secret key generated at 484924606752301G contains curve points at SK_x = 369,200,743 and SK_y = 723,310,002. The hash function incorporates two more following variables.

$$\Omega = \begin{cases} \mathrm{SK}_x - \mathrm{SK}_y & ,\text{if } \mathrm{SK}_x > \mathrm{SK}_y \\ \mathrm{SK}_y - \mathrm{SK}_x & ,\text{if } \mathrm{SK}_x < \mathrm{SK}_y \end{cases} \quad (1)$$

$$\Theta = SK_x \oplus SK_y \quad (2)$$

Consequently, Ω and Θ are said to be 354,109,259 and 1,025,334,229, respectively from Eqs. 1 & 2. Ω is added to x_1 and y_1 in the first-round while θ is subtracted from resulting values in the second-round. Similarly, we assume encrypting a message that contains five characters (*abc58*), with referenced curve points shown in Table 3. Figure 1 illustrates the further mutation of x and y coordinates into referenced curve points from Table 2. The purpose of this mutation is to increase permutation and range so that the real curve points are kept secret. The decryption process is similar and inverse of the encryption function as described in Algorithm.1 (encryption) and Algorithm.2 (decryption).

Algorithm 1: Encryption Algorithm of proposed technique	**Algorithm 2: Decryption Algorithm of proposed technique**
Start:	**Start:**
Input : $\sum_{46}^{i=1} zi$ *(message, z = characters)*	**Input :** $\sum_{46}^{i=1} zi$ *(encrypted characters , z)*
For *each* zi ***do***	**For** *each* zi ***do***
Input: $\sum_{46}^{zi=1} x_{zi}, \sum_{46}^{zi=1} y_{zi}$ *(coordinate values from Reference Table)*	**Input:** $\sum_{46}^{zi=1} x_{zi}, \sum_{46}^{zi=1} y_{zi}$ *(Separate coordinate values)*
End for	**End for**
Input: SK_x, SK_y *(secret key from RNGF/ECDH)*	**Input:** SK_x, SK_y *(secret key from RNGF/ECDH)*
For *each* x_{zi} & y_{zi} ***do*** *(Round 1)*	**For** *each* x_{zi} & y_{zi} ***do*** *(Round 1)*
ADD: $x_{zi} = \Omega_x + x_{zi}$	**ADD:** $x_{zi} = \Omega_x - x_{zi}$
ADD: $y_{zi} = \Omega_y + y_{zi}$	**ADD:** $y_{zi} = \Omega_y - y_{zi}$
$\Omega_x = x_{zi}$, $\Omega_y = y_{zi}$	$\Omega_x = x_{zi}$, $\Omega_y = y_{zi}$
End for	**End for**
For *each* x_{zi} & y_{zi} ***do*** *(Round 2)*	**For** *each* x_{zi} & y_{zi} ***do*** *(Round 2)*
ADD: $x_{zi} = \Theta_x - x_{zi}$	**ADD:** $x_{zi} = \Theta_x + x_{zi}$
ADD: $y_{zi} = \Theta_y + y_{zi}$	**ADD:** $y_{zi} = \Theta_y + y_{zi}$
$\Theta_x = x_{zi}$, $\Theta_y = y_{zi}$	$\Theta_x = x_{zi}$, $\Theta_y = y_{zi}$
End for	**End for**
	Output: x_{zi} & y_{zi} (characters comparing x, y coordinates from Reference Table)
Stop	**Stop**

RNGF: Random Number Generator Function,
ECDH: Secret Key after ECDH based key exchange protocol is established

Experimental results

This section represents analysis of results taken from implementation of the proposed technique on an IoT resource constrained device. The technique is evaluated in terms of avalanche effect and memory consumption with existing similar encryption techniques. Additionally, computational and memory costs are also presented in the preceding sections.

Experimental setup

This subsection describes the specifications of a resource constrained IoT device's hardware in terms of processing power, availability of internal memory (RAM) and code

Table 3 Hash function encryption (example)

ASCII	a		b		c		5		8	
CP	x_1	y_1	x_2	y_2	x_3	y_3	x_4	y_4	x_5	y_5
CP before Encryption	694,337,436	703,285,746	370,473,331	984,017,275	759,555,548	617,465,919	326,619,521	923,466,984	701,927,624	293,508,906
CP after Encryption	945,913,466	32,060,776	473,006,560	2,009,351,504	1,705,469,014	649,526,695	799,626,081	2,932,818,488	2,407,396,638	943,035,601

CP (Curve Points) of characters are taken from Table 1 (Reference Table)
Example can be redirected to Fig. 1

size (ROM). The proposed method is implemented on an AVR ATmega2560 machine with the following specifications.

- CPU - AVR ATmega2560 with frequency (crystal oscillator) of 16 MHz
- 256 KB of flash memory
- EEPROM of 4 KB
- SRAM of 4 KB
- 20 mA average current ratings
- 5 V of operating voltage

The ATmega2560 machine is tested on three (32, 43 & 63-bit) $mod(p)$, containing 1,019,532,643, 5,171,003, 929,967 and 4,434,155,615,661,930,479 values, respectively. Every p value is further tested against three (30, 43 and 51-bit) *SKs* with the respective curve points at $(640280610_x, 817170226_y)$ – $(5067356135643_x, 223330964472_y)$ and $(2527499838244584_x, 1675273799655845_y)$, respectively. In total, nine message blocks (3 $mod(p)$ values against 3 *SKs*) are encrypted, each containing all the ASCII characters shown in Table 2. Similarly, these encrypted message blocks are then decrypted. Computational cost, memory and energy consumptions are evaluated during the encryption and decryption processes, as mentioned in Table 4.

Avalanche effect

Avalanche effect is known as the computational time taken during encryption and decryption of data block and change in bits before and after encryption. Since significant change in bit pattern does not apply on ECC as all the resulting curve points are not random values but rather the exact curve points. However, the change of bit difference before and after the encryption can be explained in the following example.

Example.1

If a single bit is changed in encrypted block by the adversary. The decryption process will apply *SK's* round 1 & 2 as described in Algorithm 2. The final value will be formed into two different large random prime integers for x and y coordinates. We assume that the bit change resulted as exact points on curve. In this case, the resulted points for one character must be same as assigned in Table 2 (RFTAB). Similarly, the adversary will have to change bits for all characters in encrypted message block which is statistically extremely difficult and computationally exhaustive to predict. In addition, the hash function is designed in a way that change in one character will affect the entire preceding message block. Hence, the entire message will be decrypted as null characters if single bit change in the encrypted message is found wrong.

Execution time (ET) costs during encryption and decryption of 46 characters in different $mod(p)$ values and *SK* sizes, are shown in Table 4. It is observed that the execution time shows significant increase with the increase in *SK* size. However, there is slight difference in encryption time and energy usage between 32-bit and 62-bit $mod(p)$ values with similar *SK* size.

Memory consumption

Resource constrained IoT devices possess very small internal memory. Encryption techniques must address efficiency in terms of memory cost in these devices. Efficient memory usage is one of the main features of the proposed method. Furthermore, the technique enabled such devices with 192-bit security in consuming comparatively smaller internal memory. According to Table 4, 17% of the dynamic memory is occupied (1420 bytes out of 8192 bytes) for 128-bit *SK* and ≈ 192-bit security, leaving rest of the memory for other general-purpose usage. It is also observed that memory consumption mainly depends on the size of *SK*. Similarly, program memory (coded in C) occupied 7% (20,044 bytes out of 253,952 bytes) of total available memory. Hence, the method can be stored and implemented in small computing capable IoT devices that contain minimum 64Kbytes of internal memory.

Figure 2 illustrates nine tests of computational and memory performance in encryption and decryption of 46 ASCII characters. Encryption process appears to consume most of the CPU power while the decryption process utilizes almost uniform CPU power. Furthermore, 51-bit *SK* costs nearly similar computation time in all three $mod(p)$ values which confirms that the considerate size of *SK* plays important role for computational efficiency in the proposed technique. Hence, the size of *SK* can be considered a tradeoff between retaining good security and computational affordability in resource constrained IoT devices, in the proposed method.

Security analysis

This section describes security robustness of the proposed technique. Elliptic curve-based encryption security mainly depends on the keys and the hash function that encrypts data block founded on the devised keys. Hence, these keys must exhibit extreme sensitivity and randomness in devising good ciphers.

Key sensitivity

Key sensitivity means if a single bit change occurs in a key, then the whole encrypted block should also change. In our technique, the change will lead to a non-curve point and the overall output will also change during

Table 4 Computational and memory performance of the proposed technique

Mod(p) bits	SK (bits)	Encryption time (μs)	Encryption Energy (mJs^{-1})	Decryption time (μs)	Decryption energy (mJs^{-1})	SRAM (bytes)	Code Size (bytes)
43	30	300,484	30.0484	79,200	7.92	1386	20,018
	43	292,672	29.2672	76,216	7.6216	1386	20,018
	51	419,908	41.9908	84,700	8.47	1420	20,044
32	30	246,304	24.6304	74,900	7.49	1386	20,018
	43	280,260	28.026	75,496	7.5496	1386	20,018
	51	420,268	42.0268	84,792	8.4792	1420	20,044
62	30	293,284	29.3284	76,020	7.602	1386	20,018
	43	290,720	29.072	75,608	7.5608	1386	20,018
	51	420,428	42.0428	84,692	8.4692	1420	20,044

round operations. Furthermore, scope of curve points in curve25529 are mainly limited to positive integers with maximum value of $p = 2^{255} - 19$. However, our technique expands the scope of values to both positive and negative integers. This scope expansion improves permutation range from $+2^{255} - 19$ *to* $-2^{255} + 19$ which vastly improve immunity against several statistical attacks. Additionally, the method is primarily dependent on manipulation and generation of secret key (*SK*) devised from ECDH key exchange protocols or a large prime random number generator function whose seed value changes every 4 microseconds (limited to AVR ATmega family). Thus, one microsecond change in seed value will lead to different curve points during key predicting statistical attacks. Additionally, ECC curve points are statistically proven to be non-reversible. It is extremely difficult and huge computation costly to accurately calculate or predict previous curve points (Moosavi et al. 2018a). Curve25519 is faster and can withstand timing, side channel, twists on curve, ladder and statistical attacks (Dong et al. 2018). However, *SK* size used in our proposed method achieved only 192-bit security which is kept limited because of the affordability in resource constrained autonomous IoT devices (Table 5).

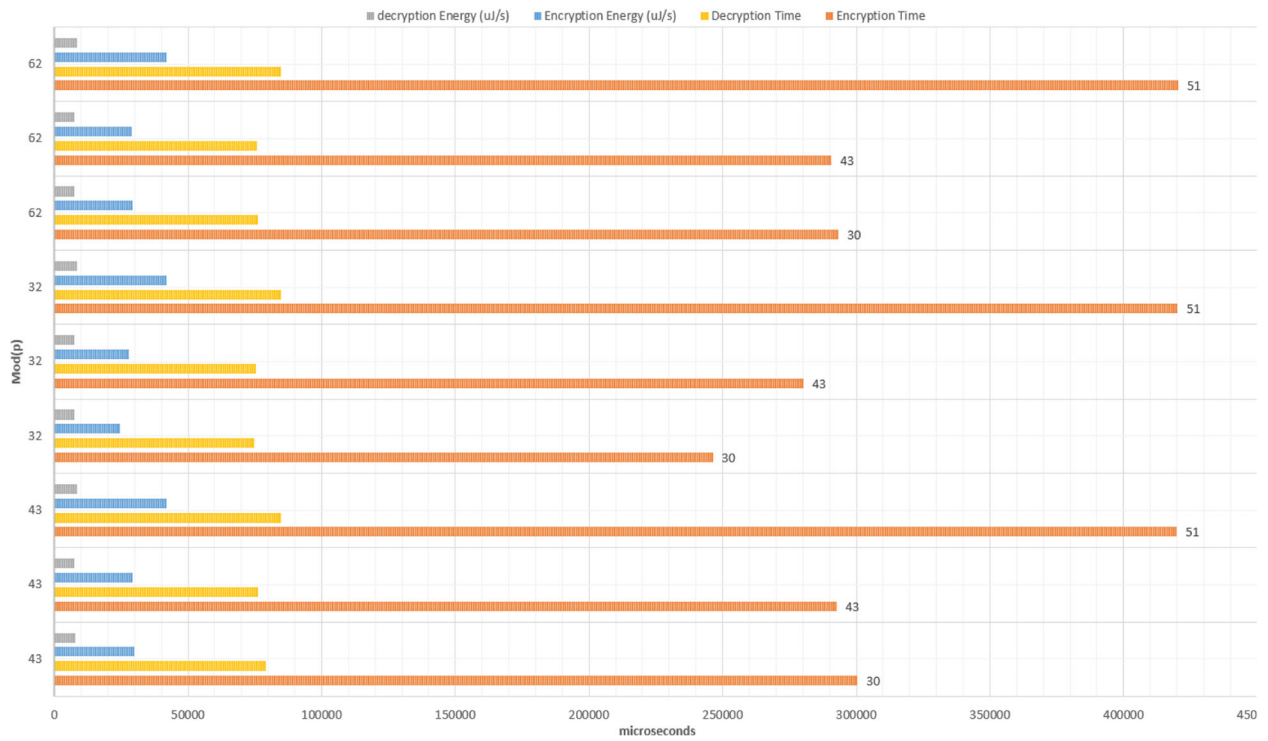

Fig. 2 Computation and memory performance of the proposed technique

Table 5 Cross correlation of encrypted and non-encrypted data and average PDF values

SK	Coordinates	*mod*(5171003929967)	*mod*(4434155615661930000)	*mod*(1019532643)
43-bits	x	−0.62307341	−0.697613234	− 0.667615167
	Y	−0.608143659	−0.741037992	− 0.701866233
	Avg.PDF	**0.071**	**0.46**	**0.067**
30-bits	X	−0.176503123	−0.48110427	− 0.468270225
	Y	−0.171319621	−0.573184911	− 0.46253257
	Avg.PDF	**0.064**	**0.063**	**0.065**
51-bits	x	0.019286371	0.019283871	0.019284215
	y	0.019286496	0.019284471	0.019284614
	Avg.PDF	**0.049**	**0.049**	**0.048**

PDF and cross correlation

An efficient encryption method should ensure high level of uniformity, recurrence, and independence to exhibit strong immunity against statistical attacks. In Uniformity, encrypted key must be correlated in size, bit-pattern and exhibit existence probability near $\frac{1}{n}$ (Noura et al. 2018). To assess uniformity of encrypted data, it is justified visually through the PDF (Probability Density Function) graph. Figure 3 plots PDF graph where horizontal-axis shows original message in ASCII and vertical-axis shows PDF value against every encrypted ASCII. 1, 2 and 3 corresponds to 43, 30 and 51-bit *SKs*, respectively. Similarly, 43, 62 and 32 corresponds to the respective size (in bits) of *mod*(p) values. In case of uniformity, Its observable in PDF plot that every encrypted character exhibit existence probability close to $\frac{1}{n}$.

Recurrence refers to a well scattered data in PDF plot and strict correlation between curve points for robust encrypted blocks. It can be seen in Fig. 3 that the PDF values are well scattered and the average PDF value is also close to $\frac{1}{n}$. Furthermore, Table 6 shows strict cross correlation between the curve points in all three *mod* (p) values. The values for 43-bit *SK* are strictly correlated while showing non uniform strict cross correlation for 30 and 51-bit *SKs*. This non uniformity in cross correlation indicates randomness between the curve points. Moreover, Avg.PDF of all tested variations is near $\frac{1}{n}$ which proves that the keys used in our technique are robust and has high level of randomness. The strict cross-correlation and well scattered PDF values increase the level of permutation due to long random signed integer values which are devised through mutation in the hash function, as shown in Fig. 1.

Computational performance analysis

This section deals with performance analysis of the proposed technique with similar ECC based techniques. The analysis is presented in terms of curve operations

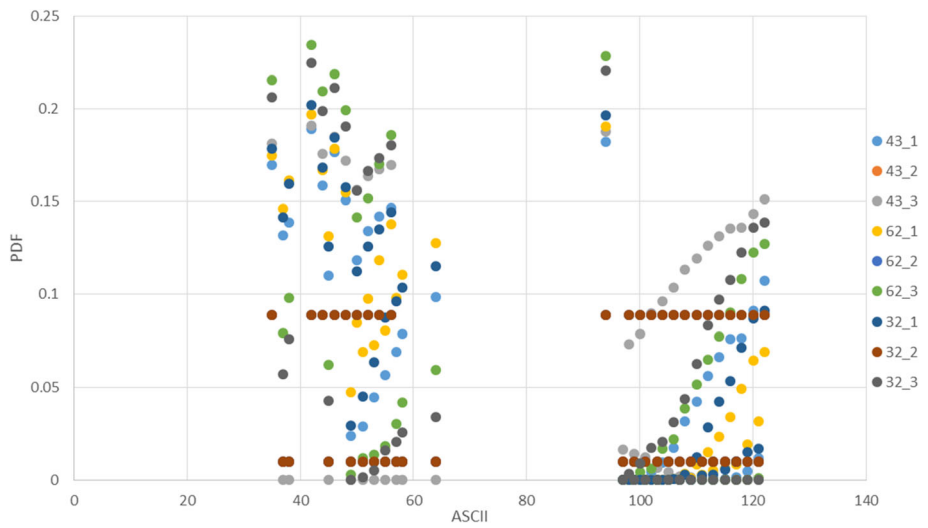

Fig. 3 Probability density function plot

Table 6 Curve operation performance comparison for 128-bit key generation

Work	CPU	Curve	Key Generation
De Santis and Sigl (2016)	ARM Cortex M4 @ 48 MHz	X25519	1563.8 (pa + pd)
Oliveira et al. (2017)	Teensy 3.1 @ 48 MHz	ECqDSA	614 (pa + pd)
Fujii and Aranha (2018)	ARM Cortex M4 @ 48–72 MHz	Ed25519 & Ed448	353 (pa + pd)
Faz-Hernández et al. (2019)	Intel Haswell (AVX2) processors	X25519 & X448	18pa + 12pd
Chou (2015)	Intel Ivy bridge	Curve25519	31pa + 4pd
This Work	**ATmega2560 @ 16 MHz**	**Curve 25,519**	**25pa + 48 pd**

pd Point doubling (Point Multiplication), *pa* Point addition & subtraction

and CPU execution time cost in constructing 128-bit encrypted keys.

Curve operation cost

Robust elliptic curve key generation determines the efficiency of performance due to its heavy computation. Curve operations are executed on large random prime numbers with maximum size of 255-bits. These operations consist of fixed-point multiplication, scaler multiplication, point addition, point doubling, multiplicative inverse and verification. Point addition and point doubling are the main computational exhaustive curve operations and other operations are subparts of these operations. Additionally, Table 6 shows cost comparisons of our technique with similar techniques where all mentioned techniques used comparatively more computational power capable CPUs. Research carried out by Faz-Hernández et al. (2019) and Chou (2015) used advanced curve operations in relatively more powerful CPUs with advance instruction set architectures which cannot be afforded by 8-bit CPUs. That is why our work's CPU expense is slightly higher than Faz-Hernández et al. (2019) and Chou (2015). Moreover, Fig. 4 illustrates the comparison shown in Table 6 where it is observable that our proposed technique utilizes efficient curve operations in creating 128-bit encrypted keys.

Encryption cost analysis

This section is based on cost analysis of the proposed technique in terms of execution time (clock cycles) during encryption of 128-bit keys. Table 7 summarizes comparative cost analysis of encryption performance of existing techniques where the techniques utilized similar curve types (Montgomery) and hardware (AVR family) to generate encrypted keys over 160-bit field. The techniques adopted different libraries and features to improve encrypted keys' security and performance. Liu et al. (2016) used optimal prime field (OPF) library to optimize scaler multiplication process and compared with former implementation on Gallant-Lambert-Vanstone technique on twisted Edward curves. Hutter and Schwabe (2013) applied networking and cryptography library (NaCl) and used Salsa20 encryption method. The method utilized significant clock cycles, whereas use of memory was efficient. Düll et al. (2015) coded in assembly language to reduce CPU cycles by factor of 1.6x comparing to Hutter and Schwabe (2013) in several microcontrollers including AVR ATmega2560 and ATmega128. Inter-pulse interval (IPI) and ECG

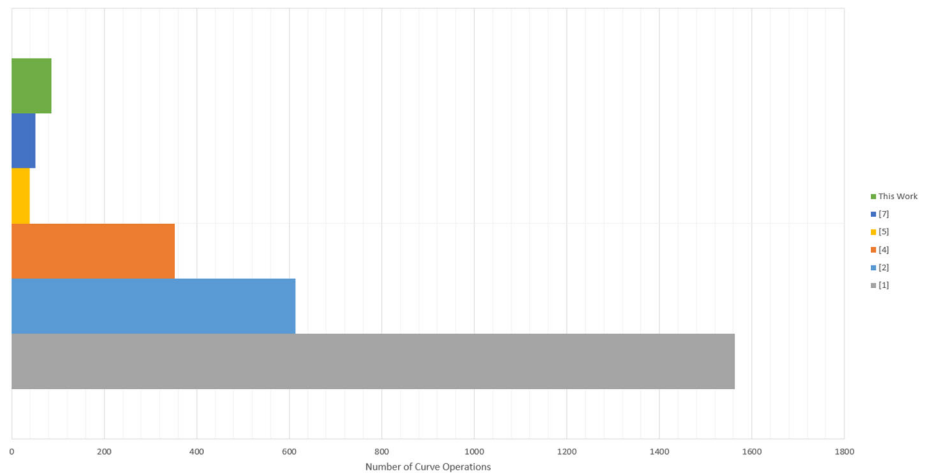

Fig. 4 Curve25519 based curve operation cost comparison of 128-bit encrypted key generation with similar existing techniques

Table 7 Encryption cost comparison with existing technique in AVR hardware

Techniques	Field	Support library / feature	Hardware	Execution Time (ET)	RAM
Liu et al. (2016)	160-bit	OPF lib	ATmega128	5.53 s	n/a
Liu et al. (2014)	160-bit	OPF / GLV	Atmega128	5.5 s	n/a
Hutter and Schwabe (2013)	160-bit	NaCL	Atmega128	22.8	677
Düll et al. (2015)	160-bit	Assembly	ATmega2560	14.15 s	510
Altop et al. (2015)	160-bit	IPI / PPG / BP	ATmega128L	225*16 = 3.6 s	n/a
Zhang et al. (2011)	160-bit	IPI / BSs	ATmega128L	0.278*16 = 2.8 s	n/a
Moosavi et al. (2017)	160-bit	IPI / LSFR PRNG	ATmega128L	0.1884*16 = 3 s	n/a
Our work [a]	**160-bit**	**GMP / Hash**	**ATmega2560**	**2.623 s**	**812**
Our work [b]				**1.08 s**	**812**

[a]192bit security provision
[b]96bit security provision

based key generation methods were developed by Altop et al. (2015) and Zhang et al. (2011) through mixing IPI random intervals with other features such as physiological parameter generation (PPG), blood pressure and biometric binary sequences (BSS). The keys were generated over finalizing 16 sequential samples of IPI. Furthermore, Moosavi et al. (2017) generated 128-bit encrypted key for body area network (BAN) through similar IPI and ECG based features. IPI feature of ECG executed 188 ms for one sample. In total 16 consecutive cycles of IPIs and Fibonacci linear feedback shift register (LFSR) based pseudo random number samples were collected to generate a single key. We utilized GMP library and introduced a novel hash function that achieved better performance in encrypting 128-bit keys with 96-to-192-bit security provisions. Moreover, Fig. 5 illustrates comparison of execution time cost in microseconds. It can be concluded that our work generated 128-bit encrypted keys with affordable internal memory consumption and enabled 8-bit CPUs in achieving elliptic curve based robust end-to-end encryption.

Discussion

There have been several in-depth studies and implementations on elliptic curve cryptographies as the functionality and applications of autonomous resource constrained devices grew. Table 8 provides the summery of similar implementations of elliptic curve cryptography in IoT devices, achievable milestones and weaknesses. Apart from AVR family, ARM machines are comparatively more expensive and cannot be categorized as resource constrained IoT devices for stream/block ciphers because these machines possess 32-bit computation power and high memory capacity. As a result, with faster CPUs, advanced encryption algorithms (that require more memory and clock cycles) can easily be applied.

The novelty in our work is the utilization of GMP library for related arithmetic calculations in retaining

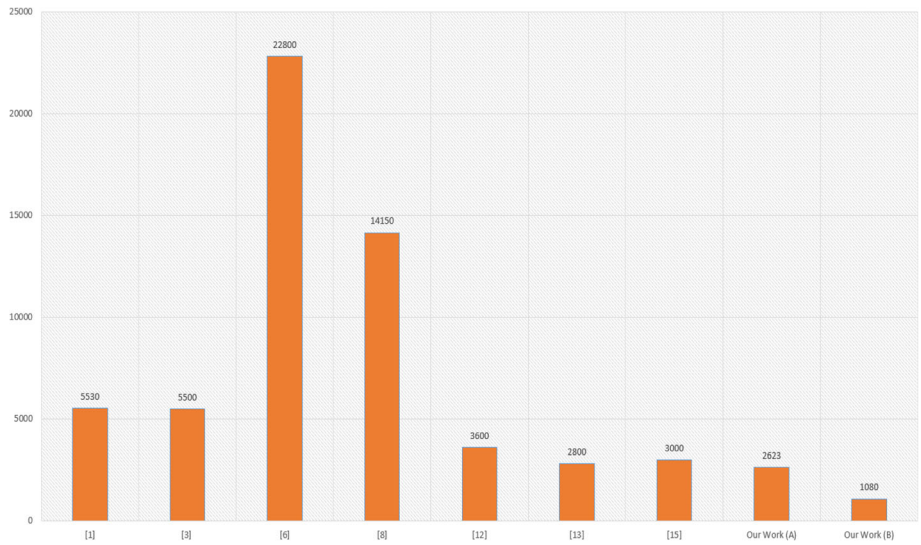

Fig. 5 128-bit encrypted key generation performance comparison

Table 8 Comparison with existing ECC techniques in resource constrained IoT devices

Article	Protocol	Curve	Hardware	Achievement	Weakness
Liu and Ning (2008)	ECDSA, ECIES	SECG Curves	ATmega - @ 16 MHz	Low security levels (80–96 bits)	limited for small MTCDs
Düll et al. (2015)	ECDH	Curve25519	ARM Cortex M0 @ 48 MHz	Robustness Cost effective	Consume 2x computation due to heavy curve
De Clercq et al. (2014)	ECDLP	Mixed curves	MSP430X ARM Cortex	Resource constrained device afforded computation	Side channel attacks horizontal attacks
Liu et al. (2015)	ECDSA, ECDH	NIST P-192	ATmega328P @ 16 MHz	Resource constrained device afforded computation	Ineffective security, Side channel Threat
Fujii and Aranha (2018)	ECqDSA	Curve25519	ARM Cortex-M4 @ 48 MHz	Up to 50% optimized Operations	Inefficient implementation technique
Devi et al. (2015)	ECDH	NIST P-256	3GPP	Low execution requirements	Database related threats and optimization
Moosavi et al. (2018b)	ECDH	ECG based IPI-PRNG	WLAN Configuration	Large prime random curve points	Secure authorization and effective authentication of all devices
De Santis and Sigl (2016)	ECqDSA	X25519	ARM Cortex M4 @ 48 MHz	Efficient DSA	Heavy Curves and computation cost
Oliveira et al. (2017)	ECqDSA	Ed25519	Teensy 3.1 @ 48 MHz	Improved DSA computation	Heavy point arithmetic
Fujii and Aranha (2018)	ECqDSA	Ed25519	ARM Cortex M4 @ 48–72 MHz	Efficient Key generation	Heavy curve and multi point arithmetic
This Work	**ECDH / LPRN**	**Curve25519**	**ATmega2560 @ 16 MHz**	**Strong curve & extremely low memory and exec. cost**	**Performance efficiency depends on Secret key's size**

almost similar security levels and robust encrypted blocks with 8-bit CPU that consumed only 1-2Kbytes of RAM. The GMP library enabled our work for faster execution of the curve operations in generating 128-bit keys and large random prime integers. Furthermore, techniques mentioned in Table 8 applied different ciphering methods such as RSA, AES and PRNG. Whereas our novel hash function performed faster executions during encryption due to pre-calculation strategy and provided affordable 192-bit security.

Conclusion

The proposed method is mainly designed to provide robust ECC based 96-to-192-bit affordable security to resource constrained MTC, CPS and IoT devices. The curve25519 is applied to provide faster execution for generating secret keys with support from GMP library. Furthermore, hash function consumed affordable computational and memory consumption in providing good encryption. Additionally, the proposed method proves to retain good security level (described in Security analysis section) with efficient consumption of internal memory. Moreover, generating keys at very large prime integers, cost computation that might not be affordable as these CPUs are cheap and possess less computation capabilities. It can be elaborated from Table 6 that curve25519 used in our method has generated robust secret keys (as summarized in Table 8) with statistically proven good cipher blocks and costing less computation (as shown in Table 7). The proposed technique can be applied on all AVR ATmega 8-bit machines with capability of 32-64Kbytes internal memory. However, the method is dependent on GMP (GNU Multiple Precision) arithmetic library in handling large prime random integers during scaler multiplication. Devices that lack support from GMP library will have to use alternative libraries that can differ in performance.

Supplementary Information

The online version contains supplementary material available at https://doi.org/10.1186/s42400-021-00078-6.

Additional file 1.

Additional file 2.

Acknowledgments

We acknowledge that the work is not funded by any funding.

Authors' contributions

Shafi Ullah: Concept and Design of Work, Experiments. Dr. Raja Zahilah: Data analysis and interpretation. The author(s) read and approved the final manuscript.

Competing interests

The authors concur that there are no competing interests related to this research artical.

Received: 19 November 2020 Accepted: 7 February 2021
Published online: 02 March 2021

References

Altop DK, et al (2015) Towards using physiological signals as cryptographic keys in body area networks. 2015 9th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth), IEEE
Aranha DF, Dahab R, López J, Oliveira LB (2010) Efficient implementation of elliptic curve cryptography in wireless sensors. Adv Math Commun 4(2):169–187
Banerjee S, Patil A (2018) ECC Based Encryption Algorithm for Lightweight Cryptography. International Conference on Intelligent Systems Design and Applications, Springer
Bernstein DJ (2006) Curve25519: new Diffie-Hellman speed records. International Workshop on Public Key Cryptography, Springer
Bernstein, DJ, Lange, T: SafeCurves: choosing safe curves for elliptic-curve cryptography. 2014. https://safecurves.cr.yp.to. Accessed 1 Dec 2014.
Chou T (2015) Sandy2x: new Curve25519 speed records. International Conference on Selected Areas in Cryptography, Springer
Chu D, et al (2013) Twisted Edwards-form elliptic curve cryptography for 8-bit AVR-based sensor nodes. Proceedings of the first ACM workshop on Asia public-key cryptography, ACM
De Clercq R, et al (2014) Ultra low-power implementation of ECC on the ARM Cortex-M0+. 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC), IEEE
De Santis F, Sigl G (2016) Towards side-channel protected X25519 on ARM Cortex-M4 processors. In: Proceedings of Software performance enhancement for encryption and decryption, and benchmarking, Utrecht, The Netherlands, pp 19–21
Devi GU, Balan EV, Priyan M, Gokulnath C (2015) Mutual authentication scheme for IoT application. Indian J Sci Technol 8(26):15
Dong J, et al (2018) Towards High-performance X25519/448 Key Agreement in General Purpose GPUs. 2018 IEEE Conference on Communications and Network Security (CNS), IEEE
Düll M, Haase B, Hinterwälder G, Hutter M, Paar C, Sánchez AH, Schwabe P (2015) High-speed Curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers. Des Codes Crypt 77(2–3):493–514
Faz-Hernández A, López J, Dahab R (2019) High-performance implementation of elliptic curve cryptography using vector instructions. ACM Trans Math Softw 45(3):1–35
Fujii H, Aranha DF (2018) Efficient Curve25519 implementation for ARM microcontrollers. Anais Estendidos do XVIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, SBC
Gouvêa CP, Oliveira LB, López J (2012) Efficient software implementation of public-key cryptography on sensor networks using the MSP430X microcontroller. J Cryptogr Eng 2(1):19–29
Gura N, et al (2004) Comparing elliptic curve cryptography and RSA on 8-bit CPUs. International workshop on cryptographic hardware and embedded systems, Springer
Hinterwälder G, Moradi A, Hutter M, Schwabe P, Paar C (2014) Full-size high-security ECC implementation on MSP430 microcontrollers. In: International conference on cryptology and information security in Latin America. Springer, pp 31–47
Hutter M, Schwabe P (2013) NaCl on 8-bit AVR microcontrollers. International Conference on Cryptology in Africa, Springer
Liu A, Ning P (2008) TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. Proceedings of the 7th international conference on Information processing in sensor networks, IEEE Computer Society
Liu Z, Seo H, Großschädl J, Kim H (2015) Efficient implementation of NIST-compliant elliptic curve cryptography for 8-bit AVR-based sensor nodes. IEEE Trans Inf Forensics Secur 11(7):1385–1397
Liu Z, Weng J, Hu Z, Seo H (2016) Efficient elliptic curve cryptography for embedded devices. ACM Trans Embed Comput Syst 16(2):1–18
Liu Z, et al (2014) MoTE-ECC: Energy-scalable elliptic curve cryptography for wireless sensor networks. International Conference on Applied Cryptography and Network Security, Springer
Montgomery PL (1987) Speeding the pollard and elliptic curve methods of factorization. Math Comput 48(177):243–264
Moosavi SR, Nigussie E, Levorato M, Virtanen S, Isoaho J (2018a) Low-latency approach for secure ECG feature based cryptographic key generation. IEEE Access 6:428–442
Moosavi SR, Nigussie E, Levorato M, Virtanen S, Isoaho J (2018b) Performance analysis of end-to-end security schemes in healthcare IoT. Procedia Comput Sci 130(C):432–439
Moosavi SR, Nigussie E, Virtanen S, Isoaho J (2017) Cryptographic key generation using ECG signal. In: 2017 14th IEEE annual consumer communications & networking conference (CCNC). IEEE, pp 1024–1031
Noura H, et al (2018) Efficient and secure physical encryption scheme for Low-Power wireless M2M devices. 2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC), IEEE
Oliveira T, et al (2017) How to (pre-) compute a ladder. International Conference on Selected Areas in Cryptography, Springer
Oualha N, Nguyen KT (2016) Lightweight attribute-based encryption for the internet of things. 2016 25th International Conference on Computer Communication and Networks (ICCCN), IEEE
Rajesh S, Paul V, Menon VG, Khosravi MR (2019) A secure and efficient lightweight symmetric encryption scheme for transfer of text files between embedded IoT devices. Symmetry 11(2):293
Song T, Li R, Mei B, Yu J, Xing X, Cheng X (2017) A privacy preserving communication protocol for IoT applications in smart homes. IEEE Internet Things J 4(6):1844–1852
Szczechowiak P, et al (2008) NanoECC: Testing the limits of elliptic curve cryptography in sensor networks. European conference on Wireless Sensor Networks, Springer
Wenger E, et al (2013) 8/16/32 shades of elliptic curve cryptography on embedded processors. International Conference on Cryptology in India, Springer
Zhang G-H, Poon CC, Zhang Y-T (2011) Analysis of using interpulse intervals to generate 128-bit biometric random binary sequences for securing wireless body sensor networks. IEEE Trans Inf Technol Biomed 16(1):176–182

Publisher's Note