Standardized signature algorithms on ultra-constrained 4-bit MCU

Chen, Chien-Ning; Jacob, Nisha; Kutzner, Sebastian; Ling, San; Poschmann, Axel; Saetang, Sirote

2012

Chen, C. N., Jacob, N., Kutzner, S., Ling, S., Poschmann, A. & Saetang, S. (2012).
Standardized signature algorithms on ultra-constrained 4-bit MCU. Proceedings of the 7th
International Workshop on Security, IWSEC 2012, 7631, pp.37-50.

<https://hdl.handle.net/10356/99792>

https://doi.org/10.1007/978-3-642-34117-5_3

© 2012 Springer-Verlag Berlin Heidelberg. This is the author created version of a work that has been peer reviewed and accepted for publication by Proceedings of the 7th International Workshop on Security, IWSEC 2012, Springer-Verlag Berlin Heidelberg. It incorporates referee's comments but changes resulting from the publishing process, such as copyediting, structural formatting, may not be reflected in this document. The published version is available at: [http://dx.doi.org/10.1007/978-3-642-34117-5_3].

Downloaded on 26 Apr 2023 22:51:53 SGT

Standardized Signature Algorithms on Ultra-Constrained 4-bit MCU ^{*}

Chien-Ning Chen¹, Nisha Jacob², Sebastian Kutzner¹,
San Ling², Axel Poschmann¹², and Sirote Saetang²

Physical Analysis & Cryptographic Engineering (PACE)
Nanyang Technological University, Singapore
{chienning,skutzner}@ntu.edu.sg

School of Physical and Mathematical Sciences
Nanyang Technological University, Singapore
{njacob,lingsan,aposchmann,sirote.tang}@ntu.edu.sg

Abstract. In this work, we implement all three digital signature schemes specified in Digital Signature Standard (FIPS 186-3), including DSA and RSA (based on modular exponentiation) as well as ECDSA (based on elliptic curve point multiplication), on an ultra-constrained 4-bit MCU of the EPSON S1C63 family. Myriads of 4-bit MCUs are widely deployed in legacy devices, and some in security applications due to their ultra low-power consumption. However, public-key cryptography, especially digital signature, on 4-bit MCU is usually neglected and even regarded as infeasible. Our highly energy-efficient implementation can give rise to a variety of security functionalities for these ultra-constrained devices.

Keywords: 4-bit MCU, DSA, ECDSA, Elliptic Curve Cryptography, Lightweight Cryptography, RSA, SHA-1.

1 Introduction

In recent years, the area footprint of hardware implementations of standardized algorithms has been continuously brought down to a level, where it is hard to yield any further gain, e.g. for AES [31] from 5,400 GE [37] down to 2400 GE [30]. In the meantime, a great deal of research work has been spent on the design of new lightweight cryptographic primitives. Notably examples for block ciphers and hash functions include KLEIN [12], KATAN [5], LED [14], PICCOLO [43] and PRESENT [4] for the former, and QUARK [1], PHOTON[13] and SPONGENT[3] for the latter, amongst many others. A major optimization goal for those lightweight algorithms is to reduce the area footprint in silicon in order to reduce the cost and the power consumption. The recent adoption of PRESENT as an ISO standard [17] shows the maturity of the field, and, hence, it is no wonder that state-of-the-art lightweight algorithms require close to the theoretical optimal area [13].

^{*} The authors were supported in part by the Singapore National Research Foundation under Research Grant NRF-CRP2-2007-03.

At the same time, there is a surprising lack of improvements on the software side. 8-bit microcontrollers (MCUs) have been long used as the platform of choice to evaluate the efficiency of cryptographic algorithms in embedded devices. However, one of the simplest, cheapest and most-abundant computing platforms is 4-bit MCUs that are embedded in a wide variety of everyday items. Applications range from watches and toys to security sensitive applications such as remote access and control systems, car immobilizers, one-time password generators, and all sorts of sensors. The ultra low power consumption of a few micro ampere [40] makes it a fitting choice for passive RFID-tags and a reasonable choice for active RFID-tags as well.

Previous works on 4-bit MCUs are mostly on symmetric crypto, i.e. block cipher implementations using a legacy device from ATMEL. **PRESENT** is reported in [44], **HUMMINGBIRD** [9] in [10], and **AES** in [18, 22]. [18] also reports the first implementations of the hash function, **SHA-1** [33], and the public key primitive, **ECC**. In this work we partially build on the results of [18] and combine for the first time **SHA-1** and **ECC** to **ECDSA** on a 4-bit MCU. We also present the first implementations of **DSA**, **RSA**, and Rabin cryptosystem on a 4-bit MCU and compare the results. Our implementations provide functionalities of digital signature on 4-bit MCU for applications that are not timing critical, e.g., legally binding sensor/meter readings and secure firmware updates.

The remainder of this work is organized as follows. In Section 2 the target platform and the design flow are briefly introduced. Section 3 discusses modular exponentiation and in particular the Montgomery multiplication. Subsequently, **DSA** is treated in Section 4, before Section 5 describes our **ECDSA** implementation. Finally, we conclude this paper in Section 6.

2 Target Platform and Design Flow

The Epson S1C63 family of MCUs was introduced in 2011 and is one of the most recent 4-bit low-power architectures. All members of the S1C63 family have a 4-bit core along with ROM, RAM, LCD drivers, and I/O ports. It also has a two-stage pipeline (fetch and execute) and a maximum of 15 and 63 hardware and software interrupt vectors respectively, depending on the model being used. The MCUs differ mainly in the memory size and on-board components, such as UART or hardware multiplier [39]. In this work, due to the extensive space requirement for public-key cryptography, we use S1C63016, which has 26kB (16k*13 bits) of code ROM, 1kB (2k*4 bits) of RAM and 2kB (4k*4 bits) of data ROM as well as an integer multiplier/divider communicated through memory I/O.

The S1C63 MCU core supports a wide instruction set with a linear addressing space without pages. It has two 4-bit data registers **A** and **B**; a 4-bit flag register **F** consisting of extension **E**, interrupt **I**, carry **C** and zero flag **Z**; two 16-bit index registers **X** and **Y** supporting post increment instructions; two stack pointers, **SP1** for address and **SP2** for data. Table 1 gives a list of some frequently used

instructions and their instruction cycles. One instruction cycle is equal to 2 clock cycles¹.

Table 1. Frequently used instruction list [39] of S1C63 family MCU.

Mnemonic*				Cycles
LD [%ir]+,%r	LD %r, [%ir]+	ADC %r, [%ir]	ADC %r, [%ir]+	1
CMP %r, [%ir]+	CMP [%ir]+,%r	AND %r, imm4	OR %r, imm4	1
JR sign8	JRNC sign8	CALR imm8	RET	1
LDB %EXT,%BA	LDB %rr, imm8	ADD %ir,%BA	ADD %ir, sign8	1
LD [%ir]+, [%ir]+	LDB [%X]+,%BA	ADC [%ir]+,%r		2
INC [addr6]	DEC [addr6]	XOR [%ir]+,%r	EX %r, [%ir]+	2

*ir = index register (X or Y); r = data register (A or B); rr = XL, XH, YL, YH;
imm4 = 4-bit immediate data; imm8 = 8-bit immediate data;
sign8 = signed 8-bit digit; addr6 = 6-bit absolute data address.

Details of the design flow of this MCU can be found in [42]. For debugging we use a software simulator on PC (Fig. 1(a)) and a FPGA-based hardware emulation board, called In-Circuit Emulator (ICE) (Fig. 1(b)) [41]. The code will be tested first on the software simulator or on the ICE and then burned on the target board (Fig. 1(c)). The advantage of using the ICE over the software simulator is to ensure the proper operation of the system before burning it on the target board. The software simulator is also used to get the cycle count and code size of our implementations, which are the two most common performance metrics for embedded platforms. Furthermore, energy consumptions are estimated based on datasheets.

3 Modular Exponentiation

Modular exponentiation is widely used in public-key cryptosystems, like RSA [36] and DSA [32]. It is the most time consuming operation in these cryptosystems and determines their performance. This section presents our implementation of 512-bit and 1024-bit modular exponentiation on the EPSON 4-bit MCU, S1C63016.

The computation of modular exponentiation can be divided into two parts: modular multiplication of multi-precision integers at the bottom and exponentiation evaluation on the top. In our implementation, modular multiplication is realized by using the Montgomery multiplication [29] to avoid expensive modular operations, and exponentiation is evaluated by the binary left-to-right exponentiation algorithm. The implementation details of these two parts are provided in Sec. 3.1 and Sec. 3.2.

¹ In the remainder of the paper we refer to instruction cycles as cycles

Fig. 1. S1C63 family development tools [41].

3.1 Montgomery Multiplication

Montgomery multiplication introduced by Peter Montgomery is commonly used in modular arithmetic. It computes $(A \times B \times 2^{-nt} \bmod M)$ instead of $(A \times B \bmod M)$ to avoid expensive modular operations (divisions). Figure 2 provides the typical Montgomery multiplication, where A and B are the two operands, M is the modulus, $A[]$, $B[]$, $M[]$ are their (2^t) -ary representation, $m' = (-M[0])^{-1} \bmod 2^t$, and $0 \leq A, B, M < 2^{nt}$ as well as $0 \leq A[i], B[i], M[i] \leq 2^t - 1$ for all $0 \leq i \leq n-1$.

Input: $A[], B[], M[]$
Output: $R[] = \text{MontMul}(A[], B[])$

```

01  $T[] = 0$ 
02 for  $i = 0$  to  $n-1$ 
03    $T[] = T[] + A[i]B[]$ 
04    $u = (T[0] \times m') \bmod 2^t$ 
05    $T[] = (T[] + u \times M[]) / 2^t$ 
06 output  $R[] = T[]$  or  $R[] = T[] - M[]$ 

```

Fig. 2. Montgomery multiplication algorithm.

In a naïve implementation, the inputs and the result of the Montgomery multiplication will satisfy $0 \leq A, B, R < M$. It needs to check if $T \geq M$ and optionally performs a subtraction $T - M$ before outputting the result. The check and the optional subtraction will cause the execution time to depend on the operands. In addition, the downward scanning in the check is less efficient in both computational time and code size because the EPSON 4-bit MCU only supports post-increment instructions.

C. Walter [45] as well as G. Hachez and J.-J. Quisquater [16] proposed some techniques to eliminate the check and the subtraction, in order to have a constant run-time. In their methods, the parameters will satisfy $A, B, R < 2M$ as well as $2M < 2^{(n'-1)t}$ or $M < 2^{(n'-1)t}$. However, in order to satisfy the extra condition for the modulus, we will have $n' = n + 1$ or $n + 2$ for an nt -bit modulus, which will cause a large overhead on ultra-constrained devices. When n is replaced by $n' = n + 1$, the Montgomery multiplication will require $2(n + 1)^2 + 1$ t -bit multiplications instead of $2n^2 + 1$ multiplications. In addition, since n is usually of 2's power, replacing n by $n' = n + 1$ might also cause some extra costs in memory management.

In order to avoid either the slow check or the extra cost of extending M to $2M$, our implementation only keeps the inputs and the result within $0 \leq A, B, R < 2^{nt}$ (i.e., A , B , and R might be greater than M). The temporary result after each iteration (lines 02–05 in Fig.2) will satisfy $T \leq 2^{nt} + M - 1$. After the whole n iterations, we only check if $T \geq 2^{nt}$, which is much easier than checking $T \geq M$, and a final subtraction $T - M$ is required when $T \geq 2^{nt}$. To achieve a constant time implementation, the optional final subtraction can be evaluated by

$$T[i] = T[i] - (\text{mask AND } M[i]) - c$$

from $i = 0$ to $n - 1$, where c is the carry (borrow) flag and $\text{mask} = (-T[n] \bmod 2^t) = 0$ or $2^t - 1$.

Our implementation achieves the constant execution time of 242,916 and 960,944 cycles for a 512-bit and 1024-bit Montgomery multiplication, respectively. Detailed results are provided in Table 2.

3.2 Exponentiation Computation

We implement the binary left-to-right exponentiation algorithm. In order to use the Montgomery multiplication, some additional computations are required before and after the exponentiation. When computing $X^E \bmod M$, the base number X will be converted to $X' = (X \times 2^{nt} \bmod M)$ before exponentiation. After exponentiation, one extra Montgomery multiplication $R = \text{MontMul}(R', 1) = R' \times 2^{-nt} \bmod M$ is required to get the final result. Although the Montgomery multiplication in our implementation only ensures its output being smaller than 2^{nt} (i.e., might be greater than the modulus M), the output of $\text{MontMul}(R', 1)$ will always be smaller than M when $R' \neq M$ and $M > 2^{nt-1}$ (i.e., M is nt -bit).

Table 2 provides the implementation results including the code size² and the execution time. The execution time of the exponentiation with a full-length

² Each instruction takes 13 bits, and we provide the code size in byte (8 bits).

exponent is the average value by assuming the Hamming weight of exponent is equal to half of its bit length.

Table 2. Implementation results of Montgomery multiplication and exponentiation.

Operation	Code Size [bytes]	Cycles [million]		Energy [mJ] @3V	
		512-bit	1024-bit	512-bit	1024-bit
Montgomery multiplication	260	0.243	0.961	0.0801	0.317
Exponentiation (full exponent)	499	187.1	1,476	61.74	487.08
Exponentiation (exponent = $2^{16} + 1$)	463	5.156	19.15	1.70	6.31

It is clear that exponentiation with a full length exponent (e.g, RSA signature generation) is impracticality for this ultra-constrained MCU. However, exponentiation with a short exponent (e.g., RSA signature verification with public key $e = 2^{16} + 1$) might still be practical. For Rabin cryptosystem [35], only one modular squaring is required for signature verification. The computation can be further reduced to one Montgomery multiplication (i.e., without pre- and post-computation) by using a modified signature $S' = S \times 2^{nt/2} \bmod M$.

4 Digital Signature Algorithm

The digital signature standard was announced by the US National Institute of Standards and Technology (NIST) in 1991, of which the latest specification can be found in FIPS 186-3 [32]. It includes the secure hash algorithm (SHA) specified in FIPS 180-4 [33], and the digital signature algorithm (DSA). In this section, we combine the SHA-1 implementation in [18] and the modular exponentiation in the previous section and then implement DSA with domain parameters $L = 1024$ bits and $N = 160$ bits (i.e., 1024-bit modulus and 160-bit exponent in modular exponentiation).

4.1 SHA-1 Implementation

SHA-1 is a secure hash standard published by NIST in 1995. It processes arbitrary messages up to a length of 2^{64} bits and produces a 160-bit message digest. There are two stages of SHA-1 computation, *preprocessing* and *hash computation*.

Preprocessing stage of SHA-1 consists of the following three steps.

1. *Padding*: The message is padded by a bit ‘1’ followed by the necessary number ($0 \sim 511$) of bits ‘0’, and then the bit length of the original message (a 64-bit integer) is appended. The length of the message after padding will be a multiple of 512 bits.
2. *Parsing the padded message*: This step divides the padded message into blocks of 512 bits.

3. *Initialize hash value:* The 160-bit starting value is initialized by the five 32-bit words: $A = 0x67452301$, $B = 0xEFCDAB89$, $C = 0x98BADCFE$, $D = 0x10325476$ and $E = 0xC3D2E1F0$ in big-endian.

Hash computation: SHA-1 consists of 80 rounds for each block (512 bits) of the message. A block of message will be divided into 16 32-bit words, $M_0 \sim M_{15}$. In each round, $W_t = M_t$ for $0 \leq t \leq 15$, or $W_t = \text{ROTL}^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$ for $16 \leq t \leq 79$, where $\text{ROTL}^n()$ is the n -bit rotate left (circular left shift) operation. The round function $f_t(B, C, D)$, constants K_t , and the round computations are described in Table 3 and Fig. 3.

Table 3. SHA-1 function $f_t(B, C, D)$ and constants K_t .

Round (t)	$f_t(B, C, D)$	K_t
0 to 19	$(B \wedge C) \oplus (\neg B \wedge D)$	0x5A827999
20 to 39	$B \oplus C \oplus D$	0x6ED9EBA1
40 to 59	$(B \wedge C) \oplus (B \wedge D) \oplus (C \wedge D)$	0x8F1BBCDC
60 to 79	$B \oplus C \oplus D$	0xCA62C1D6

Fig. 3. One round of SHA-1 computation.

Details of our SHA-1 implementation can be found in [18]. Table 4 summaries the results for space (code size) and speed optimization.

Table 4. Implementation results of SHA-1.

Optimization	Code Size [bytes]	Cycles	Energy Consumption [μ J] @3V
Space	2,038	108,666	35.85
Speed	2,324	87,788	28.97

4.2 DSA Implementation

The digital signature algorithm provides the capability of generation and verification of a digital signature. The system parameters include two prime numbers p and q , satisfying $2^{1023} < p < 2^{1024}$, $2^{159} < q < 2^{160}$, and q divides $(p - 1)$, as well as a base number $g \in \mathbb{Z}_p^*$ of the order q . Signer's private key is x , satisfying $0 < x < q$, and the public key is $y = g^x \bmod p$. The signature generation and verification algorithms are given below:

Signature generation

1. Compute $h(m)$ by using SHA-1.
2. Generate a random ephemeral key k satisfying $0 < k < q$.
3. Compute $k^{-1} \bmod q$.
4. Compute $r = (g^k \bmod p) \bmod q$.
5. Compute $s = (k^{-1}(h(m) + x \times r)) \bmod q$.

Signature verification

1. Verify the signature (r', s') satisfying $0 < r' < q$ and $0 < s' < q$.
2. Compute $w = (s')^{-1} \bmod q$.
3. Compute $u_1 = (h(m) \times w) \bmod q$.
4. Compute $u_2 = (r' \times w) \bmod q$.
5. Compute $v = (g^{u_1} \times y^{u_2} \bmod p) \bmod q$.
6. If $v = r'$, the signature is valid.

We implement both DSA signature generation and verification. The SHA-1 hash function has been implemented in [18] and introduced in Sec. 4.1, other atomic computations are described as follows, and the implementation results are summarized in Table 5.

Modular exponentiation: We employ the exponentiation algorithm described in Sec. 3.2, which is based on the Montgomery multiplication. The modulus is the 1024-bit prime p , and the length of the exponents k , u_1 and u_2 are 160-bit. We also implement Shamir's double-exponentiation algorithm [11, section V.B] for signature verification.

Multiplication and reduction: Except the modular exponentiation, other multiplications modulo q are achieved by using row-wise multiplication and Barrett reduction [2]. When reducing a 1024-bit integer by the 160-bit modulus q , seven reductions are required, starting from MSB of the 1024-bit integer.

Inversion: We employ the binary extended GCD algorithm [23, Ch 4.5.2] implemented in [18].

5 Elliptic Curve Digital Signature Algorithm

Elliptic curve cryptography (ECC), introduced independently by Neil Koblitz [24] and Victor Miller [27], is an alternative of public-key cryptography. Similar to the discrete logarithm problem on modular exponentiation, ECC can be employed

Table 5. Implementation results of DSA.

Operation	Code Size [bytes]	Cycles [million]	Energy Consumption [mJ] @ 3V
Exponentiation (160-bit exponent)	499	232.68	76.78
Double-exp (160-bit exponents)	655	274.69	90.65
Barrett reduction	425	0.036	0.011
Inversion	703	0.13	0.043
Signature generation	3,951	239.18	78.92
Signature verification	4,154	290.78	95.96

in a variety of applications, like key exchange (e.g., ECDH [38]), digital signature (e.g., ECDSA [19]). The main advantage of ECC is the small key size. ECC with much smaller key size can provide the same level of security as RSA or DLP-based cryptography, e.g., ECC with 160-bit key is as secure as RSA with 1024-bit key [15]. The small key size reduces the cost of communication, storage, and even computation, and makes it particularly suitable for constrained devices.

ECC relies upon group operations in an elliptic curve group, and a group $\mathbb{E}$ over field $\mathbb{F}_p$ can be defined by the points (x, y) satisfying the short Weierstrass form:

$$\mathbb{E} : y^2 = x^3 + ax + b, \text{ where } 4a^3 + 27b^2 \neq 0.$$

In “Standards for Efficient Cryptography 2” (SEC2) [7], an elliptic curve over a prime field is specified by a sextuple: (p, a, b, G, n, h) , where p is the prime, a and b are the curve parameters, G is a base point with order n , and h is the cofactor.

In this section, we combine the SHA-1 and the SEC2 curve `secp160r1` implementation in [18] and then provide the first implementation of the standardized elliptic curve digital signature, ECDSA, on the 4-bit MCU, S1C63016.

5.1 ECDSA Implementation

ECDSA is one of the standardized digital signature schemes. The system parameters of ECDSA include the specification of the underlying curve (`secp160r1` for our implementation), signer’s private key d ($0 < d < n$) and public key $Q = dG$. The signature generation and verification algorithms are described as follows.

ECDSA signature generation

1. Compute $h(m)$ by using SHA-1.
2. Generate a random number k satisfying $0 < k < n$.
3. Compute $k^{-1} \pmod{n}$.
4. Compute $r = x \pmod{n}$, where $(x, y) = kG$.
5. Compute $s = (k^{-1}(h(m) + d \times r)) \pmod{n}$.

ECDSA signature verification

1. Verify the signature (r', s') satisfying $0 < r' < n$ and $0 < s' < n$.
2. Compute $w = (s')^{-1} \bmod n$.
3. Compute $u_1 = (h(m) \times w) \bmod n$.
4. Compute $u_2 = (r' \times w) \bmod n$.
5. $P = (x, y) = u_1G + u_2Q$.
6. If $r' = x \bmod n$, the signature is valid.

The computations in ECDSA can be divided into three parts: prime field arithmetic, point arithmetic, and protocol layer. There are two different types of prime field arithmetic in ECDSA, where the moduli are the pseudo Mersenne prime p in the point arithmetic and the curve order n in the protocol layer, respectively.

Prime Field Arithmetic The curve `secp160r1` in SEC2 employs the pseudo Mersenne prime, $p = 2^{160} - 2^{31} - 1$, which makes the modulo operations much more efficient compared to the Barrett reduction [2] and the Montgomery multiplication [29]. The computations modulo the 164-bit curve order n in the protocol layer are performed by using Barrett reduction. We have implemented the following operations for both moduli p and n , and the results are summarized in Table 6.

Modular Addition and Subtraction: We further optimize the implementation in [18] and generalize it for both 160-bit and 164-bit moduli by using two entry points. The functions for the two moduli share most of the code, but some additional code is required for processing the 164-bit modulus.

Multiplication (M) and Squaring (S): According to the implementation result in [18], the row-wise multiplication is more efficient for this MCU than the column-wise or hybrid multiplication. We separately implement the modular multiplication for the modulus n because a generalized implementation for both moduli will cause 10% increase in execution time for the 160-bit multiplication and this significantly slows down the overall run time (as about 90% run time of a point multiplication is spent on underlying field multiplications).

Bisection is required only for point arithmetic. For an even number, division by 2 is a right shift of its binary representation. For an odd number, an extra addition of the odd prime p is required before the right shift.

Inversion (I) is achieved by the binary extended GCD algorithm [23, Ch 4.5.2], which requires multi-precision addition/subtraction and bisection. The implementation in [18] only supports the 160-bit modulus p , and we extend it to support both 160-bit and 164-bit moduli.

Reduction: Since the prime number of the curve `secp160r1` is a *pseudo mersenne prime*, the reduction modulo p can be implemented efficiently by using only shifts and additions. However, the reduction modulo the curve order n requires the Barrett reduction (BR). Each Barrett reduction requires 2M and some additions/subtractions. It also requires some space to store the pre-computed values.

Table 6. Implementation results of prime field arithmetic.

Operation	modulo 160-bit p		modulo 164-bit n	
	Code Size [bytes]	Cycles	Code Size [bytes]	Cycles
Modular add/sub	292	340	302	344
Multiplication	318	16,226	333	17,836
Bisection	208	207	299	212
Fast reduction	624	679	-	-
Barrett reduction	-	-	425	36,194

Point Arithmetic The major computation in ECC is the point multiplication nP which can be evaluated through the combination of point doubling $2P$ and point addition $P_1 + P_2$. Instead of representing points in affine coordinates ($\mathcal{A}$), we employ Jacobian projective coordinates ($\mathcal{J}$) to implement point doubling and addition. A point (x, y) in $\mathcal{A}$ can be represented by $(x, y, 1)$ in $\mathcal{J}$, and a point (X, Y, Z) in $\mathcal{J}$ is identical to the point $(X/Z^2, Y/Z^3)$ in $\mathcal{A}$. The following are the three point operations, and detailed results are summarized in Table 7 and Table 8.

Point Doubling (D): We implement point doubling in Jacobian coordinates ($2\mathcal{J} \rightarrow \mathcal{J}$). It requires 4M and 4S as well as some minor field operations, or alternatively 3M and 5S by using the trick $\alpha\beta = \frac{1}{2}((\alpha + \beta)^2 - \alpha^2 - \beta^2)$ [25].

Point Addition/Subtraction (A): Point addition in mixed coordinates ($\mathcal{J} + \mathcal{A} \rightarrow \mathcal{J}$) takes 8M and 3S, or 7M and 4S by using the trick described above. Point subtraction is similar to point addition but has an extra subtraction to calculate the y -coordinate.

Point Multiplication (PM): As shown in [18], we have implemented various scalar multiplication algorithms, including the basic binary left-to-right method, the left-to-right NAF recoding [21], and some side-channel countermeasures³ [8, 26]. For signature verification, we also need double point multiplication (d-PM). Employing NAF recoding in either Shamir’s double-exponentiation algorithm [11] or Möller’s interleaving algorithm [28] can achieve the average complexity of $1.55 \log_2 n$ or $1.66 \log_2 n$, respectively. However, recoding both scalars into NAF will cause huge overhead in code size on this MCU. We only implement Shamir’s method with binary scalars which achieves the average complexity of $1.75 \log_2 n$.

Table 7. Implementation results of ECC point arithmetic.

Operation	Description	Code Size [bytes]	Cycles
Point doubling	4M + 4S	900	128,453
$2\mathcal{J} \rightarrow \mathcal{J}$	3M + 5S	940	123,781
Point addition	8M + 3S	1,700	178,956
$\mathcal{J} + \mathcal{A} \rightarrow \mathcal{J}$	7M + 4S	1,748	176,601

³ Please refer to [18, Sec. 5.2] for the security-efficiency trade-off on 4-bit MCU.

Table 8. Implementation results of ECC point multiplication.

Point Multiplication Algorithm	Code Size [bytes]	Cycles [millions]	Side-channel Immunity
Left-to-right PM	5,724	34.37	-
Left-to-right PM with left-to-right NAF	8,127	29	-
Double-and-add-always	6,562	48.04	SPA
BRIP	7,681	49	SPA,RPA,ZPA,DPA
Randomization of scalar (20-bit)	8,215	32.21	DPA
Randomization of scalar (64-bit)	8,342	42.04	DPA
Randomized projective coordinates	8,093	30.50	DPA
Randomization of scalar (20-bit) & Randomized projective coordinates	8,312	32.52	DPA

Protocol Layer We implement the ECDSA signature generation and verification. Besides hash computation, for signature generation, we need $2M_{164} + I + 2BR$ (modulo 164-bit n) and one PM. For signature verification, we need $2M_{164} + I + 2BR$ and one d-PM. Table 9 shows the implementation results using a 160-bit message m (including one SHA-1 computation).

Table 9. Implementation results of ECDSA.

Operation*	Code Size [Bytes]	Cycles [Million]	Energy Consumption [mJ] @ 3V
Signature generation	8,546	35.28	11.64
Signature verification	8,611	41.9	13.87

* Using Left-to-Right PM or d-PM

6 Conclusion

In this work, we implement the three standardized signature schemes, RSA (512- and 1024-bit), DSA (1024-bit) and ECDSA (160-bit) on a 4-bit MCU. Our implementation results show that ECDSA is the most practical signature scheme for ultra-constrained devices, and when only signature verification is required, RSA with small public key is also practical. Through this work, we show that public-key cryptography is possible on constrained devices. Future work includes the investigation of side-channel immunity of our implementations.

References

1. J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia. Quark: A lightweight hash. In S. Mangard and F.-X. Standaert, editors, *CHES*, volume 6225 of *LNCS*, pages 1–15. Springer, 2010. <http://131002.net/quark/>.

2. P. Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In A. M. Odlyzko, editor, *CRYPTO*, volume 263 of *LNCS*, pages 311–323. Springer, 1986.
3. A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici, and I. Verbauwhede. spongent: A lightweight hash function. In Preneel and Takagi [34], pages 312–325.
4. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Viskelsoe. PRESENT: An ultra-lightweight block cipher. In P. Paillier and I. Verbauwhede, editors, *CHES*, volume 4727 of *LNCS*, pages 450–466. Springer, 2007. <http://lightweightcrypto.org/present/>.
5. C. D. Cannière, O. Dunkelman, and M. Knezevic. KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers. In C. Clavier and K. Gaj, editors, *CHES*, volume 5747 of *LNCS*, pages 272–288. Springer, 2009.
6. Çetin Kaya Koç and C. Paar, editors. *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *LNCS*. Springer, 1999.
7. Certicom Research. Standards for efficient cryptography, SEC 2: Recommended elliptic curve domain parameters, 2000.
8. J.-S. Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Çetin Kaya Koç and Paar [6], pages 292–302.
9. D. Engels, X. Fan, G. Gong, H. Hu, and E. M. Smith. Ultra-lightweight cryptography for low-cost RFID tags: Hummingbird algorithm and protocol. Technical report, Centre for Applied Cryptographic Research (CACR), 2009. <http://cacr.uwaterloo.ca/techreports/2009/cacr2009-29.pdf>.
10. X. Fan, H. Hu, G. Gong, E. M. Smith, and D. Engels. Lightweight implementation of Hummingbird cryptographic algorithm on 4-bit microcontrollers. In *International Conference for Internet Technology and Secured Transactions, 2009*, pages 1–5, 2009.
11. T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.
12. Z. Gong, S. Nikova, and Y. W. Law. KLEIN: A new family of lightweight block ciphers. In A. Juels and C. Paar, editors, *RFIDSec*, volume 7055 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011.
13. J. Guo, T. Peyrin, and A. Poschmann. The PHOTON family of lightweight hash functions. In P. Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 222–239. Springer, 2011.
14. J. Guo, T. Peyrin, A. Poschmann, and M. J. B. Robshaw. The LED block cipher. In Preneel and Takagi [34], pages 326–341.
15. N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing elliptic curve cryptography and RSA on 8-bit cpus. In Joye and Quisquater [20], pages 119–132.
16. G. Hachez and J.-J. Quisquater. Montgomery exponentiation with no final subtractions: Improved results. In Çetin Kaya Koç and C. Paar, editors, *CHES*, volume 1965 of *LNCS*, pages 293–301. Springer, 2000.
17. ISO/IEC. 29192-2: Information technology – security techniques – lightweight cryptography – part 2: Block ciphers. available via http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=56552.
18. N. Jacob, S. Saetang, C.-N. Chen, S. Kutzner, S. Ling, and A. Poschmann. Feasibility and practicability of standardized cryptography on 4-bit micro controllers. to appear in SAC 2012.

19. D. Johnson, A. Menezes, and S. Vanstone. The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, 2001.
20. M. Joye and J.-J. Quisquater, editors. *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *LNCS*. Springer, 2004.
21. M. Joye and S.-M. Yen. Optimal left-to-right binary signed-digit recoding. *IEEE Trans. Computers*, 49(7):740–748, July 2000.
22. T. Kaufmann and A. Poschmann. Enabling standardized cryptography on ultra-constrained 4-bit microcontrollers. In *IEEE International Conference on RFID*, pages 32–39, Orlando, USA, April 2012.
23. D. E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, Third Edition*. Addison-Wesley, 1997.
24. N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, January 1987.
25. P. Longa and A. Miri. Fast and flexible elliptic curve point arithmetic over prime fields. *IEEE Trans. Computers*, 57(3):289–302, March 2008.
26. H. Mamiya, A. Miyaji, and H. Morimoto. Efficient countermeasures against RPA, DPA, and SPA. In Joye and Quisquater [20], pages 343–356.
27. V. S. Miller. Use of elliptic curves in cryptography. In H. C. Williams, editor, *CRYPTO*, volume 218 of *LNCS*, pages 417–426. Springer, 1985.
28. B. Möller. Algorithms for multi-exponentiation. In S. Vaudenay and A. M. Youssef, editors, *Selected Areas in Cryptography*, volume 2259 of *Lecture Notes in Computer Science*, pages 165–180. Springer, 2001.
29. P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.
30. A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. Pushing the limits: A very compact and a threshold implementation of the AES. In K. G. Paterson, editor, *EUROCRYPT*, volume 6632 of *LNCS*. Springer, 2011.
31. National Institute of Standards and Technology. FIPS 197: Announcing the advanced encryption standard (AES). <http://csrc.nist.gov/publications/PubsFIPS.html>, November 2001.
32. National Institute of Standards and Technology. FIPS 186-3: Digital signature standard (DSS). <http://csrc.nist.gov/publications/PubsFIPS.html>, June 2009.
33. National Institute of Standards and Technology. FIPS 180-4: Secure hash standard (SHS). <http://csrc.nist.gov/publications/PubsFIPS.html>, March 2012.
34. B. Preneel and T. Takagi, editors. *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *LNCS*. Springer, 2011.
35. M. O. Rabin. Digitalized signatures and public key functions as intractable as factorization. <http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-212.pdf>, 1979.
36. R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.
37. A. Satoh, S. Morioka, K. Takano, and S. Munetoh. A compact Rijndael hardware architecture with S-Box optimization. In C. Boyd, editor, *ASIACRYPT*, volume 2248 of *Lecture Notes in Computer Science*, pages 239–254. Springer, 2001.
38. R. Schroepel, H. K. Orman, S. W. O’Malley, and O. Spatscheck. Fast key exchange with elliptic curve systems. In D. Coppersmith, editor, *CRYPTO*, volume 963 of *LNCS*, pages 43–56. Springer, 1995.

39. Seiko Epson Corporation. CMOS 4-bit single chip microcomputer S1C63000 core CPU manual. http://www.epson.jp/device/semicon_e/product/index_mcu.htm, 2011.
40. Seiko Epson Corporation. CMOS 4-bit single chip microcontroller S1C63003/004/008/016 technical manual. http://www.epson.jp/device/semicon_e/product/index_mcu.htm, 2011.
41. Seiko Epson Corporation. Microcontrollers 2011. http://www.epsondevice.com/webapp/docs_ic/DownloadServlet?id=ID000463, 2011.
42. Seiko Epson Corporation. Program development process. http://www.epson.jp/device/semicon_e/product/mcu/development/tool.htm, 2011.
43. K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai. Piccolo: An ultra-lightweight blockcipher. In Preneel and Takagi [34], pages 342–357.
44. M. Vogt, A. Poschmann, and C. Paar. Cryptography is feasible on 4-bit microcontrollers - a proof of concept. In *IEEE International Conference on RFID*, pages 267–274, Orlando, USA, 2009.
45. C. D. Walter. Montgomery’s multiplication technique: How to make it smaller and faster. In Çetin Kaya Koç and Paar [6], pages 80–93.