High-speed implementation of an ECC-based wireless authentication protocol on an ARM microprocessor

M.Aydos, T.Yanık and Ç.K.Koç

Abstract: The results of the implementation of elliptic curve cryptography (ECC) over the field $GF(p)$ on an 80MHz, 32-bit ARM microprocessor are presented. A practical software library has been produced which supports variable length implementation of the elliptic curve digital signature algorithm (ECDSA). The ECDSA and a recently proposed ECC-based wireless authentication protocol are implemented using the library. Timing results show that the 160-bit ECDSA signature generation and verification operations take around 46ms and 94ms, respectively. With these timings, the execution of the ECC-based wireless authentication protocol takes around 140ms on the ARM7TDMI processor, which is a widely used, low-power core processor for wireless applications.

1 Introduction

The rapid progress in wireless communication systems, personal communication systems, and smartcard technologies has brought new opportunities and challenges to be met by engineers and researchers working on the security aspects of the new communication technologies. Public-key cryptography offers robust solutions to many of the existing problems in communication systems, although excessive computational demands (on-line memory, code size and speed) have made the use of public-key cryptography limited, particularly on wireless communication systems. The implementation of public-key cryptography on server and client platforms rarely leads to problems, due to the availability of high-speed processors and extensive memory space. However, in restricted hardware environments with limited computational power and small memory, e.g. smartcards and cellular phones, we meet more challenges. The integration of the public-key cryptographic techniques is often delayed or completely ruled out due to the difficulty of obtaining efficient, reliable solutions. It is obvious that we need:

- Public-key cryptographic systems with higher strength per key bit.
- Efficient, platform-specific, and optimised implementations for a given restricted environment.

The benefits of the 'higher strength per key bit' include higher speeds, lower power consumption, smaller bandwidth requirements and smaller certificate sizes. These advantages are particularly beneficial in applications where the bandwidth, computational strength, power availability, or storage are highly constrained.

Elliptic curve cryptography [1–3] offers secure and efficient solutions for the new communication technologies. It requires fewer bits than the RSA for a similar amount of security. For example, 1024-bit RSA seems to be equivalent to 139-bit ECC, since it requires approximately the same amount of computational power to break [4]. While the ECC provides shorter key sizes, the time and code size requirements may still be excessive. Thus, efficient and optimised implementations are required for the restricted platforms found in wireless communication.

Certicom's SigGen smartcard [5] is a good example of an ECC software implementation on a restricted platform. It is a prototype smartcard with an 8-bit microprocessor that generates digital signatures using a conventional core from Motorola (68SC28). Developed in cooperation with Schlumberger, Siggen combines the Multifiex card technology with the Certicom Elliptic Curve Engine based on the field $GF(2^k)$, and provides fast public-key operations. This card demonstrates that effective digital signature applications can be implemented on standard processors. The digital signatures are generated in less than 600ms while using only 90 bytes of RAM. It has been implemented in less than 4K code. SigGen is ideally suited for applications requiring end-user identification and strong authentication.

Another interesting implementation of the ECC over the field $GF(p)$ on a 16-bit micro-computer was introduced in [6]. A practical cryptographic library has been designed, which supports the elliptic curve arithmetic operations, the digital signature generation and verification, and the secure hash algorithm SHA-1. Their target processor was Mitsubishi's 10MHz, 16-bit microcomputer M16C, which has been used in various applications in mobile telecommunication systems, e.g. cellular phones, pagers, etc. They designed two independent integer arithmetic modules: one for executing the modular arithmetic operations with respect to a fixed prime p, and the other for general integer routines which accept any positive integers with arbitrary length for wider applicability. Their goal here was to support not only the ECC but also the RSA. They have reported a speed of 150ms for generating a 160-bit ECDSA signature, and 630ms for verifying the signature. The total code size was 4 kbyte, including the SHA-1. There are much faster implementations of the ECC [7], although these implementations are obtained on high-end microprocessors.

IEE Proceedings online no. 20010511

DOI: 10.1049/ip-com:20010511

Paper first received 5th May 2000 and in revised form 10th May 2001

The authors are with the Electrical and Computer Engineering Department, Oregon State University, Owen Hall 220, Corvallis, Oregon 97331, USA

Our goal is to design a high-speed and scalable cryptographic library suitable for implementation on low-power microprocessors and digital signal processors. The library supports the ECDSA signature generation and verification and also contains SHA and DES algorithms, which are necessary for the implementation of the wireless authentication protocols. In this paper, we report the implementation results of the wireless authentication protocol described in [8]. We implemented the protocol on the 80MHz, 32-bit ARM7TDMI microprocessor using the ARM software development toolkit. The ARM7TDMI is a commonly used low-power processor for wireless communication platforms; for example, see [9, 10] and the web locations:

http://www.dspg.com/prodtech/core/article/18.htm

http://www.lucent.com/micro/NEWS/PRESS1999/022399c.html

http://www.mobilinktel.com/Press/

http://www.oki.co.jp/OKI/DBG/english/arm7tdmi.htm

http://www.sirius.be/satcom_integr.htm

In our implementation, we obtained the timings of 46.4ms ECDSA signature generation and 92.4ms ECDSA signature verification for the 160-bit ECC over the field $GF(p)$. We also obtained the total protocol execution timings, memory and bandwith requirements, which are given in this paper.

2 Elliptic curve operations

The speed of the elliptic curve operations, e.g. the point addition and point multiplication, depends on the arithmetic of the underlying finite field. The drafted IEEE standard [11] proposes the use of the fields $GF(p)$ and $GF(2^k)$. The use of the field $GF(p)$ requires that we implement modular arithmetic with respect to the prime modulus p. Due to the security requirements, the size of p is at least 100 bits, and usually around 160 bits. The large number arithmetic has been extensively studied in the context of the RSA algorithm, and efficient algorithms for field multiplication have been designed [15]. An efficient method for performing the field multiplication is the Montgomery method [13, 14], which effectively performs modulo 2^k multiplication instead of modulo p multiplication, where $2^k > p > 2k^{-1}$.

In the following we summarise several different coordinate systems used to represent elliptic curve points. This is important because for each system the total number of field multiplications is different, resulting in different speed values for elliptic curve point additions and doublings. The number of expensive field operations (multiplication, squaring and inversion) required by the elliptic curve point addition and doubling operations is summarised in Table 1 for each coordinate system.

Table 1: Field operations required in each coordinate system

	Affine	Projective	Modified Jacobian
EC addition	1 Inv + 3 Mul	16 Mul	13 Mul + 6 Squ
EC doubling	1 Inv + 4 Mul	10 Mul	4 Mul + 4 Squ

2.1 Arithmetic using affine coordinates

An elliptic curve over the finite field $GF(p)$ is defined as the set of points (x, y), satisfying the elliptic curve equation

$$y^2 = x^3 + ax + b$$

where x, y, a and b are the elements of the field. Note that the condition $4a^3 + 27b^2 \neq 0$ should be met. The addition formulae in the affine coordinates are given below. Let $P = (x_1, y_1)$, $Q = (x_2, y_2)$, and $K = P + Q = (x_3, y_3)$ be points on the elliptic curve E over the finite field $GF(p)$. The formulae for obtaining K are given below.

- Addition formulae when $P \neq \pm Q$

$U_1 = y_1 - y_2$, $U_2 = x_1 - x_2$, $U_3 = U_1 U_2^{-1}$ then

$x_3 = U_3^2 - x_1 - x_2$ and $y_3 = U_3(x_1 - x_3) - x_1$.

- Doubling formulae when $P = Q$

$U_1 = 3x_1^2 + a$, $U_2 = 2y_1$, $U_3 = U_1 U_2^{-1}$ then

$x_3 = U_3^2 - 2x_1$ and $y_3 = U_3(x_1 - x_3) - y_1$.

2.2 Arithmetic using projective coordinates

The inversion operation within the field $GF(p)$ is a time consuming operation. The projective coordinates are used to reduce the number of modular inversions [6]. Given the affine coordinates x and y, the projective coordinates X, Y and Z are obtained as

$$X = x, \quad Y = y, \quad Z = 1$$

Actually, there is more than one type of projective coordinates, although the one mentioned here provides the fastest arithmetic [11]. The equations given above are used for converting a point from the affine coordinates to the projective coordinates. The formulae for converting it back to the affine coordinates are given as

$$x = ZX^{-2} \text{ and } y = YZ^{-3}$$

The addition formulae in the projective coordinates are given in [6, 11]. Let $P = (X_1, Y_1, Z_1)$, $Q = (X_2, Y_2, Z_2)$, and $K = P + Q = (X_3, Y_3, Z_3)$ be points on the elliptic curve E over the field $GF(p)$. The formulae for obtaining K are given below.

- Addition formulae when $P \neq \pm Q$

$U_1 = X_1 Z_2^2$, $S_1 = Y_1 Z_2^3$, $U_2 = X_2 Z_1^2$, $S_2 = Y_2 Z_1^3$, $W = U_1 - U_2$, $R = S_1 - S_2$, $T = U_1 + U_2$, $M = S_1 + S_2$, $Z_3 = Z_1 Z_2 W$ then

$X_3 = R^2 - TW^2$ and $Y_3 = 2^{-1}(VR - MW^3)$, where $V = TW^2 - 2X_3$.

- Doubling formulae when $P = Q$

$M = 3X_1^2 + aZ_1^4$, $Z_3 = 2Y_1 Z_1$, $S = 4X_1 Y_1^2$ then

$X_3 = M^2 - 2S$ and $Y_3 = M(S - X^3) - T$, where $T = 8Y_1^4$.

2.3 Arithmetic using modified Jacobian coordinates

The Jacobian coordinates of the affine coordinates (x, y) are defined as (X, Y, Z), such that $x = XZ^{-2}$ and $y = YZ^{-3}$. The new elliptic curve equation then takes the form

$$Y^2 = X^3 + aXZ^4 + dZ^6$$

over the field $GF(p)$. When the Jacobian coordinates are represented as a quadruple (X, Y, Z, aZ^4), we obtain the modified Jacobian coordinates which seem to provide the fastest possible doubling formulae. The addition formulae for the Jacobian and the modified Jacobian coordinates are given in [15]. Here, we only give the equations for the latter one, since it is the one that we decided to use in our software implementation. Let $P = (X_1, Y_1, Z_1, aZ_1^4)$, $Q = (X_2, Y_2, Z_2, aZ_2^4)$, and $K = P + Q = (X_3, Y_3, Z_3, aZ_3^4)$ be points on elliptic curve E over the field $GF(p)$. The formulae for obtaining K are given below.

- Addition formulae when $P \neq \pm Q$

$U_1 = X_1 Z_2^2$, $S_1 = Y_1 Z_2^3$, $U_2 = X_2 Z_1^2$, $S_2 = Y_2 Z_1^3$, $S_2 = Y_2 Z_1^3$, $H = U_1 - U_2$, $r = S_1 - S_2$ then

$X_3 = -H^3 - 2U_1H^2 + r^2$, $Y_3 = -S_1H^3 + r(U_1H^2 - X_3)$, $Z_3 = Z_1Z_2H$ and $Z_3^4 = aZ_1^4$.

- Doubling formulae when $P = Q$

$S = 4X_1Y_1^2$, $U = 8Y_1^4$, $M = 3X_1^2 + (aZ_1^4)$, $T = -2S + M^2$ then $X_3 = T$, $Y_3 = M(S - T) - U$, $Z_3 = 2Y_1Z_1$ and $aZ_3^4 = 2U(aZ_1^4)$.

3 Elliptic curve digital signature algorithm

The operations in the elliptic curve analogue of the digital signature algorithm utilise the arithmetic of points which are elements of the set of solutions of an elliptic curve equation defined over a finite field. The security of the protocol depends on the intractability of the elliptic curve analogue of the discrete logarithm problem. First, an elliptic curve E defined over $GF(p)$ with large group of order n and a point P of large order is selected and made public to all users. Then, the following key generation primitive is used by each party to generate the individual public and private key pairs. Furthermore, for each transaction the signature and verification primitives are used. We briefly outline the elliptic curve digital signature algorithm (ECDSA) below, details of which can be found in [11].

ECDSA key generation: The user A follows these steps:

Step 1. Select a random integer $d \in [2, n - 2]$.

Step 2. Compute $Q = d \times P$.

Step 3. The public and private keys of the user A are (E, P, n, Q) and d, respectively.

ECDSA signature generation: The user A signs the message m using the following steps.

Step 1. Select a random integer $k \in [2, n - 2]$.

Step 2. Compute $k \times P = (x_1, y_1)$ and $r = x_1 \bmod n$.
If $x_1 \in GF(2k)$, it is assumed that x_1 is represented as a binary number.
If $r = 0$ then go to step 1.

Step 3. Compute $k^{-1} \bmod n$.

Step 4. Compute $s = k^{-1}(H(m) + d \cdot r) \bmod n$.
Here H is the secure hash algorithm SHA.
If $s = 0$ go to Step 1.

Step 5. The signature for the message m is the pair of integers (r, s).

ECDSA signature verification: The user B verifies A's signature (r, s) on the message m by applying the following steps:

Step 1. Compute $c = s^{-1} \bmod n$ and $H(m)$.

Step 2. Compute $u_1 = H(m) \cdot c \bmod n$ and $n2 = r \cdot c \bmod n$.

Step 3. Compute $u_1 \times P + u_2 \times Q = (x_0, y_0)$ and $v = x_0 \bmod n$.

Step 4. Accept the signature if $v = r$.

4 ECC-based wireless authentication protocol

The authentication protocol given in [8] was originally intended for mobile phones. However, it is also suitable for handheld devices and smartcards. This makes the protocol a very strong security algorithm candidate to be deployed in the next generation cellular phones and smartcards. The 160-bit key length is considered secure enough for now and the immediate future. However, the algorithms were implemented in such a way that the key length can easily be increased to any integer multiple of 16 between 176 and 256. This scalability makes our implementation unique. We briefly describe the protocol details of which are found in [8]. The protocol goals can be stated as follows:

- mutual authentication of the server and the user;
- establishing a secret authentication key to protect the data used in mutual authentication;
- non-repudiation of origin by the user and the server for relevant data sent from the user to the server and *vice versa*;
- agreement on a secret session key, which will be used to encrypt voice or data communication.

Additional features can easily be added to the protocol. These include user identity confidentiality that is hiding the identity of the portable device from an eavesdropper on the communication channel, and interoperability that is allowing the negotiation of the symmetric key algorithm between the communicating parties. The first feature can be provided by sending a new encrypted temporary ID from the server to the user after the authentication process. The latter can be supported in the protocol by changing the exchanged message format and implementing several well-known encryption algorithms at both server and user terminals.

4.1 Terminal and server initialisations

In order to receive a certificate, the terminal sends its public key Q_s together with its user identity, through a secure and authenticated channel to the CA. The CA uses its private key to sign the hashed value of the concatenation of the public key, the temporary identity I_s, and the certification expiration date t_s. The CA then sends the signed message through the secure and authenticated channel to the terminal as shown in Fig. 1.

By repeating the very same process the user acquires its certificate as shown in Fig. 2. The certificate consists of a pair of integers which is denoted as (r_s, s_s) for the server and (r_u, s_u) for the user. Here r_u and r_s are the x-coordinates of the (distinct) elliptic curve points R_u and R_s, respectively. As mentioned earlier, the proposed protocol is based on the ECDSA.

4.2 Mutual authentication between terminal and server

The protocols shown in Figs. 1 and 2 are executed off-line. The mutual authentication and key agreement protocols between the terminal (user) and the server need to be executed in real-time. We give the combined protocol in Fig. 3. The protocol steps and its resistance to several attacks have been elaborated in [8]. The number of exchanged messages of this protocol over the air is equal to four. It is important to minimise this number, since combined with the propagation delay it increases the call setup time. The transmission time will be the dominant factor for low-bit transmission channels. On the other hand, the bottleneck will be the encryption and decryption operations for high-rate transmission channels.

The protocol consists of exchanging public keys, generating random challenge numbers, exchanging encrypted certificates and the other necessary data using the special key, and then verifying the certificates in order to complete the mutual authentication process. The computational cost until this point on the user side is just a point multiplication on the curve (eP operation), generating a random number, a secret key encryption and a secret key decryption (DES, 3DES, RC5, or IDEA), and finally an ECC signature verification operation. The timing figures of these operations will increase as we increase the ECC key length from 160 bits. The scalability protects the long term investments: as

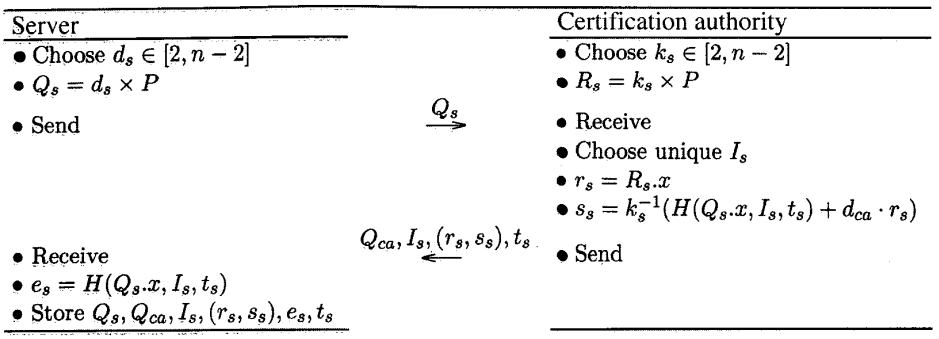

Fig. 1 *Network server initialisation*

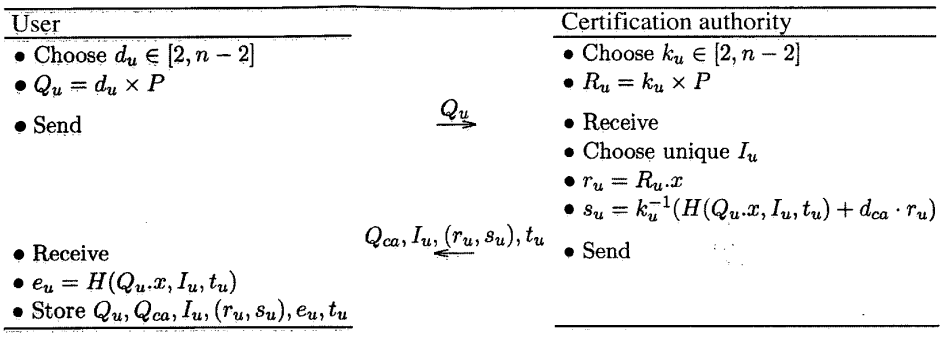

Fig. 2 *User terminal initialisation*

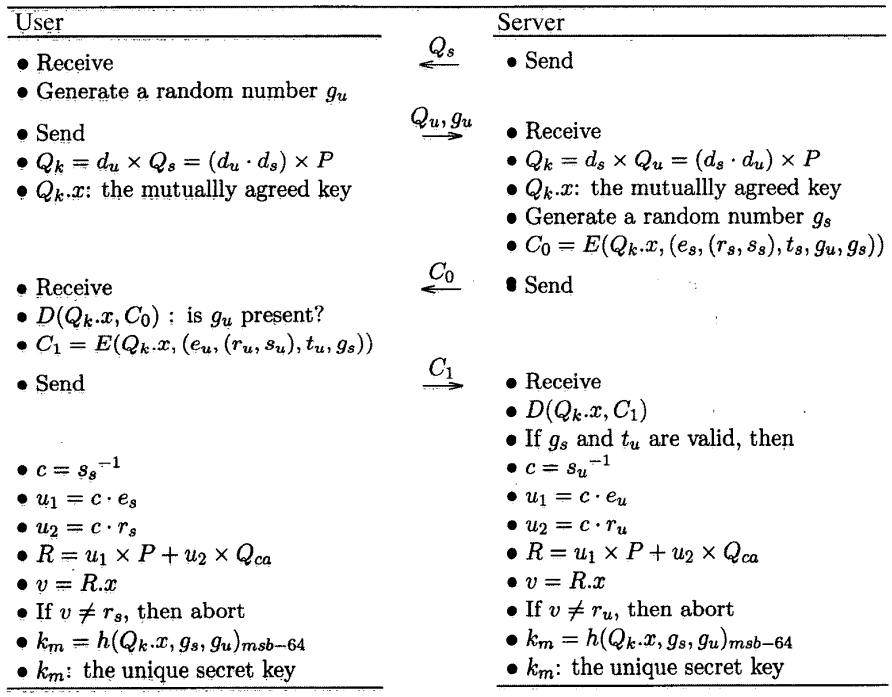

Fig. 3 *Mutual authentication and key agreement*

the key length is increased, the hardware or the software need not be modified.

The last part of the protocol establishes a session key between the user and the server. The one-time unique key is obtained by hashing several previously obtained data blocks. This key will be used to encrypt the data sent through the channel.

There are several advantages of the protocol. While roaming, a visited network cannot know the session key until a visiting user makes a request. In addition, the protocol is secure unless the attacker can compromise either the user or the server, and at the same time break the public key algorithm. Furthermore, unlike many other protocols, certificates in this protocol are kept secret at all times to prevent spoofing attacks. Finally, the unique session key is generated by performing a one-way function on the previously obtained data. Both parties contribute the same amount in generating this key.

5 Comparisons with other existing protocols

The parameter lengths (for 160–256 bits implementation) and the bandwidth and storage requirements of the protocol are summarised in Table 2. We then compare this protocol to the Beller–Chang–Yacobi protocol [16] and Aziz–Diffie protocol [17].

Table 2: Parameter lengths, bandwith and storage requirements in bits

	160-bit ECC	176-bit ECC	192-bit ECC	208-bit ECC	256-bit ECC
$Q_{u,s}$	161	177	193	209	257
$e_{u,s}$	160	160	160	160	160
$(r_{u,s}, s_{u,s})$	320	352	384	416	512
$t_{u,s}, g_{u,s}$	64	64	64	64	64
Bandwidth	1730	1826	1922	2018	2306
Storage	1408	1520	1632	1744	2080

- The protocol requires less bandwidth. The total number of bits exchanged in the real-time portion of the protocols is given as follows:

Beller–Chang–Yacobi: 8320 bits (1024-bit key)
Aziz–Diffie: 8680 bits (1024-bit key)
This protocol: 1730 bits (160-bit key)

- The protocol has low storage requirements for the user side, which makes it suitable for smartcards and other handheld computing devices. Here we refer to the space required to store public and private keys, the certificates, or any extra data required throughout the protocol:

Beller–Chang–Yacobi: 5120 bits (1024-bit key)
Aziz–Diffie: 2176 bits (1024-bit key)
This protocol: 1408 bits (160-bit key)

- The protocol has modest computational load on the user side for real-time execution:

Beller–Chang–Yacobi: 2 PKE (1024-bit) + 1 PKD (1024-bit) + Precomputation
Aziz-Diffie: 3 PKE (1024-bit) + 2 PKD (1024-bit)
This protocol: 1 eP (160-bit) + 1 ECDSAV (160-bit) + 2 SKE (672-bit data) + 1 SHA (288-bit data)

The meanings of the above symbols are as follows: PKE: public key encryption, PKD: public key decryption, eP : point multiplication, ECDSAV: elliptic curve digital signature algorithm verification, SKE: secret key encryption or decryption

6 32-bit ARM microprocessor and development toolkit

ARM Incorporated offers several microprocessor cores, and the 32-bit RISC processor, ARM7TDMI, is one of them. It is of interest to us because the processor is optimised for the best combination of die size, performance and power consumption. The processor uses a three-stage pipeline: fetch, decode and execute [18]. A pure RISC processor executes each instruction in a single cycle. However, none of the nonsuperscalar commercial RISC processors actually achieves this goal. The ARM7 processor takes one cycle to perform most data processing operations, which account for 50% of all instructions in a typical code. Single data loads take three cycles, and stores require two cycles. Load and store multiples can take up to 18 cycles. Overall, the ARM7 achieves an average CPI (clock cycles per instruction) of around 1.8 [19]. The ARM7 processor has 31 32-bit registers. At any time, 16 are visible. The other registers are used to speed up exception processing. All register specifiers in ARM instructions can address any of the 16 registers.

The ARM7TDMI is a very simple RISC processor. The core is fully 32-bit, including a 32-bit ALU, a barrel shifter, data and address buses. Although the 4 Gb of address range is rarely used in wireless applications, it does have the advantage of simplifying the decode logic by using the upper address lines as chip select signals [20]. Certain features of the processor are summarised as follows.

- Shortest instruction execution time:
  800 ns (at $f = 80$ MHz)
- Registers:
  30 general purpose registers
  6 status registers
  program counter
- Instruction sets: 48 instructions
  load and store instructions
  data processing instructions
  multiply instructions
  coprocessor instructions
  branch instructions

Portable and handheld products require processors that consume less power than those in desktop and other powered applications. RISC processors such as ARM7TDMI have some extra strengths as far as the power is concerned. A modern 32-bit RISC architecture can provide software compatibility between a range of products. This kind of modern microcontroller family is also very easy to implement. These microprocessors are available as small cores which are easy to integrate. Another advantage is on-chip debug support. These advantages make this family a good fit for embedded applications.

Another advantage of the ARM7TDMI is the fact that it has two instruction sets: The ARM7TDMI implements both the traditional 32-bit wide ARM instruction set and the new Thumb instruction set, which is only 16 bits wide. The Thumb instruction set was added to remove the limitations of code density and performance from narrow memory. Effectively, the traditional 32-bit ARM instruction set was compressed into the Thumb 16-bit instruction set. The Thumb instructions are then decompressed at execution time to produce a traditional 32-bit wide ARM instruction, which is then executed on the core as normal. As the ARM decoding is relatively simple, it is possible to do the Thumb decompression on the fly without taking any additional cycles. The special use of ARM thumb instructions enables ARM to evaluate the real GSM, DECT and D-AMPS code from the leading wireless players. There are three main issues for benchmarking the code [10]:

- Code density: This shows how much memory is required for a given high level C code. The smaller size will result in reduced cost.
- Performance: The processor's clock speed is an important factor. The smaller the clock rate to execute given algorithms, the less the power consumed. This will also lead to simpler designs. The 32-bit RISC controllers will spend most of its time in an idle mode resulting in saving power.
- Power consumption: This is one of the most important factors in wireless technology. The lower power consump-

tion will make the batteries last longer, the size smaller and the price cheaper. The ARM7TDMI consumes only 1.85mW per MHz, while the StrongARM runs up to 233MHz but only consumes 900mW [10].

ARM7TDMI is widely accepted and used in the cellular phone and smart phone technology due to its cost and power efficiencies. The future prospects show that ARM9TDMI will probably replace ARM7TDMI. Integrating the DSP module with the ARM7 family will produce the new ARM9 family [9].

7 Software architecture

A practical cryptographic library implementation of the ECC over $GF(p)$ was designed to perform the ECDSA signature generation and signature verification, which is being standardised in the ANSI X9F1 and IEEE P1363 standards committees. The IEEE-P1363 describes the algorithms in detail for elliptic point addition, doubling, multiplication, etc.

In the creation of our library, we did not make any assumption as to the elliptic curve parameters to be used. Elliptic curves can be generated randomly. Note that some ECDSA implementations fix the constant term a of the curve equation to $p - 3$ to speed up the elliptic doubling. In our case, the curve parameters and the base point (P_x, P_y) are generated randomly. Our library allows users to choose different curves with different key lengths, and therefore our library is scalable. The machine word size is 32-bit on the ARM microprocessor. The library is implemented in 27kb of code size. The modified Jacobian coordinates are used to represent the points on the curves since this gives the fastest point doubling timings.

The important features of the software library can be listed as follows:

- It supports digital signature generation, signature verification and key generation.
- It supports a superset of all standard ECC fields, basis representations, curves and key lengths, enabling compatibility with current standards and future advances.
- It supports long key lengths providing security for high-value or very long-term applications.
- It provides two optional levels of curve-based precomputation that speed up repeated operations on the same curve. Level 1 uses a small amount of additional memory and provides moderate speedup and level 2 uses a large amount of memory and provides much more speedup.

Short definitions of the modules are given as follows.

Modulo p integer library: This module contains modular operations such as modular addition, subtraction, multiplication and inversion operations modulo p. In the ECDSA signature generation operation, these routines consume the largest amount of time. In particular, the modular multiplication operation dominates the timing performance of an EC signature. To improve the performance, we use an improved version of the Montgomery multiplication algorithm.

General integer library: This library contains general operation routines. These routines accept variable length inputs.

EC point arithmetic library: This library consists of point addition, point doubling and point multiplication routines. The point addition and doubling routines are performed using the modified Jacobian coordinate system.

ECDSA key and signature generation/verification: This is the root module of our software architecture. The elliptic curve parameters and key generation are performed here. Upon creating these parameters, this top module can interact with other modules to generate signatures or to verify signatures. Note that our library does not contain a digest algorithm such as SHA-1 or MD5. We use randomly generated 160-bit message values, which is assumed to be the output of a hash function algorithm, to test the modules.

8 Implementation results

In this Section, we present our implementation results. The elliptic curve signature generation and verification timings are listed for variable key lengths to give an idea of how fast these operations could be done in today's technology. Table 3 shows the timings of the operations for variable ECC key lengths.

Table 3: Performance timings

	160-bit ECC, ms	176-bit ECC, ms	192-bit ECC, ms	208-bit ECC, ms	256-bit ECC, ms
DES	0.25	0.25	0.25	0.25	0.25
SHA	2	2	2	2	2
Point Mul	44.8	63.4	69.2	93.6	150.2
Sign Gen	46.4	65.4	71.3	96.2	153.5
Sign Ver	92.4	131.3	148.3	194.3	313.4
Protocol	139.7	197.2	220	290.4	466.1

Note that our library does not have a random number generator (RNG). Generating a random number is very fast and therefore its timing value is negligible compared to the other operations such as point multiplication and signature generation. Similarly, SHA operations can be executed very fast. According to the implementation in [6], the SHA-1 requires approximately 2 ms to digest one block (512 bits) of data. It is a hardware implementation on a 16-bit Mitsubishi microprocessor (M16C). In our protocol the input size to the SHA-1 is given as $k + 128$, where k is the implemented elliptic curve key length. The largest k value shown in the table is 256 bits for which the input size for SHA-1 is 384-bits. Therefore, for each key length given in the Table 3, the SHA-1 input length in our protocol should be padded to reach 512-bit block size. We assume that in the worst case scenario we will obtain 2ms timing value for processing a block of data using SHA-1.

The protocol's timings on signature generation and verification is better than that of [6]. Hasegawa, Nakajima and Matsui reported signature generation and verification timings as 150ms and 630ms on a 16-bit microcomputer, respectively, in our implementation, the signature generation and verification timings are 46ms and 92ms on a 32-bit ARM microprocessor.

9 Possible enhancements

Possible enhancements for further speeding up and/or reducing code size are: (i) the scalar multiplication of the base point can be performed in a more efficient way by having a precomputed look-up table in the ROM area; (ii) the finite field multiplication operations dominate the performance of signature generation and verification. Even a small improvements on the existing multiplication routine improves the overall ECDSA performance; and (iii) the 16-bit wide Thumb instruction set of ARM7TDMI can be used to reduce the code size.

10 Conclusions

In this paper, we presented a practical implementation of the ECC over the field $GF(p)$. The field and elliptic curve operation algorithms in the library were written in such a way that the implemented design will permit the use of increased key lengths. Recently, it was claimed [4] that 1024-bit RSA and 139-bit ECC offer computationally equivalent security. This is better than the generally believed security comparison, in which 1024-bit RSA and 160-bit ECC offer similar security.

In our implementation we created an ECC library that is capable of performing the ECDSA signature generation and verification operations. More importantly, the implementation permits users to select different elliptic curves with longer key sizes. This scalable architecture of the design enables the ECC to be used in restricted platforms as well as high-end servers. With this implementation, we obtained timing results of 46 ms and 92 ms for the ECC-160 signature generation and verification on a 32-bit ARM processor, respectively. In addition, the timing results were obtained for a recently proposed wireless authentication and key agreement protocol [8]. This protocol can be used in third generation wireless communication as a security protocol due to its bandwidth and storage efficiency and fast execution timing performance. The protocol execution timing is 140 ms on a ARM7TDMI processor.

11 Acknowledgments

This research was supported by rTrust Technologies.

12 References

1 MENEZES, A.J.: 'Elliptic curve public key cryptosystems' (Kluwer Academic Publishers, Boston, MA, 1993)
2 KOBLITZ, N.: 'A course in number theory and cryptography' (Springer, Berlin, Germany, 1994, 2nd edn.)
3 BLAKE, I., SEROUSSI, G., and SMART, N.: 'Elliptic curves in cryptography' (Cambridge University Press, New York, 1999)
4 LENSTRA, A.K., and VERHEUL, E.R.: 'Selecting cryptographic key sizes'. Proceedings of 3rd Workshop on *Elliptic curve cryptography* (ECC 99), Waterloo, Canada, 1999, pp. 1–3
5 CERTICOM.: SigGen Smart Card. 1997, http://205.150.149.57/ce2/embed.htm
6 HASEGAWA, T., NAKAJIMA, J., and MATSUI, M.: 'A practical implementation of elliptic curve cryptosystems over $GF(p)$ on a 16-bit microcomputer' *in* IMAI, H., and ZHENG, Y. (Eds): 'First international workshop on Practice and theory in public key cryptography, Lecture Notes in Computer Science, No.1431' (Springer, Berlin, Germany, 1998), pp. 182–194
7 ITOH, K., TAKENAKA, M., TORII, N., TEMMA, S., and KURIHARA, Y.: 'Fast implementation of public-key cryptography on a dsp tms320c6201' *in* KOÇ, Ç.K, and PAAR, C. (Eds): 'Cryptographic hardware and embedded systems, Lecture Notes in Computer Science, No. 1717' (Springer, Berlin, Germany, 1999), pp. 61–72
8 AYDOS, M., SUNAR, B., and KOÇ, Ç.K.: 'An elliptic curve cryptography based authentication and key agreement protocol for wireless communication'. Proceedings of 2nd International workshop on *Discrete algorithms and methods for mobile computing* and Communications symposium on *Information theory*, Dallas, Texas, 1998
9 GUNASEKARA, O.: 'Smart phone challenges'. 1997, http://www.arm.com/Documentation/WhitePapers/SmartPhone
10 GUNASEKARA, O.: 'Developing a digital cellular phone using a 32-bit microcontroller'. 1998, http://www.arm.com/Documentation/WhitePapers/CellPhone
11 IEEE.: 'P1363: Standard specifications for public-key cryptography'. Draft Version 13, 1999
12 KOÇ, Ç.K.: 'High-speed RSA implementation'. Technical Report TR 201, 1994, RSA Laboratories
13 MONTGOMERY, P.L.: 'Modular multiplication without trial division', *Math. Comput.*, 1985, **44**, (170), pp. 519–521
14 KOÇ, Ç.K., ACAR, T., and KALISKI, B.S.: 'Analyzing and comparing Montgomery multiplication algorithms', *IEEE Micro*, 1996, **16**, (3), pp. 26–33
15 COHEN, H., MIYAJI, A., and ONO, T.: 'Efficient elliptic curve exponentiation using mixed coordinates' *in* OHTA, K., and PEI, D. (Eds): 'Advances in cryptology - ASIACRYPT 98, Lecture Notes in Computer Science, No.1514' (Springer, Berlin, Germany, 1998) pp. 51–65
16 BELLER, M.J., CHANG, L.F., and YACOBI, J.: 'Privacy and authentication on a portable communications systems', *IEEE J. Sel. Areas Commun.*, 1993, **11**, (6), pp. 821–829
17 AZIZ, A., and DIFFIE, W.: 'A secure communications protocol to prevent unauthorized access: Privacy and authentication for wireless local area networks', *IEEE Pers. Commun.*, 1994, pp. 25–31
18 JAGGAR, D.: 'ARM architecture and systems', *IEEE Micro*, 1997, pp. 9–11
19 SEGARS, S.: 'ARM7TDMI power consumption', *IEEE Micro*, 1997, pp. 12–19
20 ARM Incorporated.: 'Advanced RISC machines architectural reference manual' (Prentice-Hall, New York, 1998)