An High-Speed ECC-based Wireless Authentication Protocol on an ARM Microprocessor

M. Aydos, T. Yanık, and Ç. K. Koç
Electrical & Computer Engineering
Oregon State University
Corvallis, Oregon 97331, USA
{aydos,yanik,koc}@ece.orst.edu

Abstract

In this paper, we present the results of our implementation of elliptic curve cryptography (ECC) over the field $GF(p)$ on an 80-MHz, 32-bit ARM microprocessor. We have produced a practical software library which supports variable length implementation of the elliptic curve digital signature algorithm (ECDSA). We implemented the ECDSA and a recently proposed ECC-based wireless authentication protocol using the library. Our timing results show that the 160-bit ECDSA signature generation and verification operations take around 46 ms and 94 ms, respectively. With these timings, the execution of the ECC-based wireless authentication protocol takes around 140 ms 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, however, 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 brings 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 optimized 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 [18, 14, 4] offers secure and efficient solutions for the new communication technologies. It requires fewer bits than the RSA for similar amount of security. For example, recently, it was claimed [17] 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. While the ECC provides shorter key sizes, the time and code size requirements may still be excessive. Thus, efficient and optimized implementations are required for the restricted platforms particularly found in wireless communication.

Certicom's SigGen smartcard [5] is an example 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 Multiflex 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 600 ms while using

1063-9527/00 $10.00 © 2000 IEEE

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 microcomputer was introduced in [9]. They have designed a practical cryptographic library, 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 10-MHz, 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 was here to support not only the ECC but also the RSA. They have reported a speed of 150 ms for generating a 160-bit ECDSA signature and 630 ms for verifying the signature. Total code size was 4 kilobytes, including the SHA-1. There are much faster implementations of the ECC [12], however, these implementations are obtained on high-end microprocessors.

Our goal has been 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 [1]. We implemented the protocol on the 80-MHz, 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 the references [7, 8] 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 46.4 ms ECDSA signature generation and 92.4 ms 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. We first summarize methods to perform efficient elliptic curve arithmetic in §2. We then give a brief description of the ECDSA algorithm in §3 and the recently proposed wireless authentication protocol in §4. The ECC-based wireless authentication protocol is compared to the other existing protocols in §5. We briefly describe the ARM microprocessor and its development environment in §6. A brief description of the software architecture of our ECC implementation is given in §7. Finally, the timing results of our implementation are given in §8 and the conclusions of the study are given in §9.

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 [10] 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, 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 [19, 16], which effectively performs modulo 2^k multiplication instead of modulo p multiplication, where $2^k > p > 2^{k-1}$.

In the following we summarize 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 are summarized in Figure 1 for the considered coordinate systems.

Figure 1: The field operations for coordinate systems.

EC Op.	Affine	Project.	Modif. Jacobi
Add	1 Inv + 3 Mul	16 Mul	13 Mul + 6 Squ
Double	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 $P \neq \pm Q$**

$$\begin{aligned} U_1 &= y_1 - y_2 \\ U_2 &= x_1 - x_2 \\ U_3 &= U_1 U_2^{-1} \\ x_3 &= U_3^2 - x_1 - x_2 \\ y_3 &= U_3(x_1 - x_3) - x_1 \end{aligned}$$

- **Doubling formulae when $P = Q$**

$$\begin{aligned} U_1 &= 3x_1^2 + a \\ U_2 &= 2y_1 \\ U_3 &= U_1 U_2^{-1} \\ x_3 &= U_3^2 - 2x_1 \\ y_3 &= U_3(x_1 - x_3) - y_1 \end{aligned}$$

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 [9]. Given the affine coordinates x and y, the *projective coordinates* X, Y, and Z are obtained as

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

Actually, there are more than one kind of projective coordinates, however, the one mentioned here provides the fastest arithmetic [10]. 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

$$x = XZ^{-2} \text{ and } y = YZ^{-3} \, .$$

The addition formulae in the projective coordinates are given in [9, 10]. 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 $P \neq \pm Q$**

$$\begin{aligned} U_1 &= X_1 Z_2^2 \\ S_1 &= Y_1 Z_2^3 \\ U_2 &= X_2 Z_1^2 \\ S_2 &= Y_2 Z_1^3 \\ W &= U_1 - U_2 \\ R &= S_1 - S_2 \\ T &= U_1 + U_2 \\ M &= S_1 + S_2 \\ Z_3 &= Z_1 Z_2 W \\ X_3 &= R^2 - TW^2 \\ V &= TW^2 - 2X_3 \\ Y_3 &= 2^{-1}(VR - MW^3) \end{aligned}$$

- **Doubling formulae when $P = Q$**

$$\begin{aligned} M &= 3X_1^2 + aZ_1^4 \\ Z_3 &= 2Y_1 Z_1 \\ S &= 4X_1 Y_1^2 \\ X_3 &= M^2 - 2S \\ T &= 8Y_1^4 \\ Y_3 &= M(S - X_3) - T \end{aligned}$$

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

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

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 [6]. 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 $P \neq \pm Q$**

$$\begin{aligned} U_1 &= X_1 Z_2^2 \\ S_1 &= Y_1 Z_2^3 \\ U_2 &= X_2 Z_1^2 \\ S_2 &= Y_2 Z_1^3 \\ H &= U_1 - U_2 \\ r &= S_1 - S_2 \\ X_3 &= -H^3 - 2U_1 H^2 + r^2 \\ Y_3 &= -S_1 H^3 + r(U_1 H^2 - X_3) \\ Z_3 &= Z_1 Z_2 H \\ aZ_3^4 &= aZ_1^4 \end{aligned}$$

- **Doubling formulae when $P = Q$**

$$\begin{aligned} S &= 4X_1Y_1^2 \\ U &= 8Y_1^4 \\ M &= 3X_1^2 + (aZ_1^4) \\ T &= -2S + M^2 \\ X_3 &= T \\ Y_3 &= M(S-T) - U \\ Z_3 &= 2Y_1Z_1 \\ aZ_3^4 &= 2U(aZ_1^4) \end{aligned}$$

3. Elliptic Curve Digital Signature Algorithm

The operations in the elliptic curve analogue of the Digital Signature Algorithm utilize 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 [10].

ECDSA Key Generation The user A follows these steps:

1. Select a random integer $d \in [2, n-2]$.
2. Compute $Q = d \times P$.
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.

1. Select a random integer $k \in [2, n-2]$.
2. Compute $k \times P = (x_1, y_1)$ and $r = x_1 \bmod n$.
   If $x_1 \in GF(2^k)$, it is assumed that x_1 is represented as a binary number.
   If $r = 0$ then go to Step 1.
3. Compute $k^{-1} \bmod n$.
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.
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:

1. Compute $c = s^{-1} \bmod n$ and $H(m)$.
2. Compute $u_1 = H(m) \cdot c \bmod n$ and $u_2 = r \cdot c \bmod n$.
3. Compute $u_1 \times P + u_2 \times Q = (x_0, y_0)$ and $v = x_0 \bmod n$.
4. Accept the signature if $v = r$.

4. An ECC-based Wireless Authentication Protocol

The authentication protocol given in [1] 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 immediate future. However, the algorithms were implemented in 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. Below, we briefly describe the protocol, details of which are found in [1].

4.1. Terminal and Server Initializations

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 Figure 2.

By repeating the very same process the user acquires its certificate as shown in Figure 3. 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 in Figures 2 and 3 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 Figure 4.

Figure 2: Network Server Initialization.

SERVER		CERTIFICATION AUTHORITY
• Choose $d_s \in [2, n-2]$		• Choose $k_s \in [2, n-2]$
• $Q_s = d_s \times P$		• $R_s = k_s \times P$
• Send	$\xrightarrow{Q_s}$	• Receive
		• Choose unique I_s
		• $r_s = R_s.x$
		• $s_s = k_s^{-1}(H(Q_s.x, I_s, t_s) + d_{ca} \cdot r_s)$
	$\xleftarrow{Q_{ca}, I_s, (r_s, s_s), t_s}$	• Send
• Receive		
• $e_s = H(Q_s.x, I_s, t_s)$		
• Store $Q_s, Q_{ca}, I_s, (r_s, s_s), e_s, t_s$		

Figure 3: User Terminal Initialization.

USER		CERTIFICATION AUTHORITY
• Choose $d_u \in [2, n-2]$		• Choose $k_u \in [2, n-2]$
• $Q_u = d_u \times P$		• $R_u = k_u \times P$
• Send	$\xrightarrow{Q_u}$	• Receive
		• Choose unique I_u
		• $r_u = R_u.x$
		• $s_u = k_u^{-1}(H(Q_u.x, I_u, t_u) + d_{ca} \cdot r_u)$
	$\xleftarrow{Q_{ca}, I_u, (r_u, s_u), t_u}$	• Send
• Receive		
• $e_u = H(Q_u.x, I_u, t_u)$		
• Store $Q_u, Q_{ca}, I_u, (r_u, s_u), e_u, t_u$		

Figure 4: Mutual Authentication and Key Agreement.

USER		SERVER
	$\xleftarrow{Q_s}$	• Send
• Receive		
• Generate a random number g_u		
	$\xrightarrow{Q_u, g_u}$	
• Send		• Receive
• $Q_k = d_u \times Q_s = (d_u \cdot d_s) \times P$		• $Q_k = d_s \times Q_u = (d_s \cdot d_u) \times P$
• $Q_k.x$: The mutually agreed key		• $Q_k.x$: The mutuallly agreed key
		• Generate a random number g_s
		• $C_0 = E(Q_k.x, (e_s, (r_s, s_s), t_s, g_u, g_s))$
	$\xleftarrow{C_0}$	
• Receive		• Send
• $D(Q_k.x, C_0)$: Is g_u present?		
• $C_1 = E(Q_k.x, (e_u, (r_u, s_u), t_u, g_s))$		
	$\xrightarrow{C_1}$	
• Send		• Receive
		• $D(Q_k.x, C_1)$
		• If g_s and t_u are valid, then
• $c = {s_s}^{-1}$		• $c = {s_u}^{-1}$
• $u_1 = c \cdot e_s$		• $u_1 = c \cdot e_u$
• $u_2 = c \cdot r_s$		• $u_2 = c \cdot r_u$
• $R = u_1 \times P + u_2 \times Q_{ca}$		• $R = u_1 \times P + u_2 \times Q_{ca}$
• $v = R.x$		• $v = R.x$
• If $v \neq r_s$, then abort		• If $v \neq r_u$, then abort
• $k_m = h(Q_k.x, g_s, g_u)_{msb-64}$		• $k_m = h(Q_k.x, g_s, g_u)_{msb-64}$
• k_m: The unique secret key		• k_m: The unique secret key

The protocol steps and its resistance to several attacks have been elaborated in [1]. The number of exchanged messages of this protocol over the air is equal to 4. It is important to minimize 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 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 to higher. The scalability protects the long term investments: as 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.

5. Comparisons to other Existing Protocols

The parameter lengths (for 160-256 bits implementation) and the bandwidth and storage requirements of the protocol are summarized in Figure 5. We compare this protocol to the Beller-Chang-Yacobi protocol [3] and Aziz-Diffie protocol [2] below.

- 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)

  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

Figure 5: The parameter lengths, bandwith, and storage.

ECC →	160	176	192	208	256
$Q_{u,s}$	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

6. The 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 optimized for the best combination of die size, performance and power consumption. The processor uses a three-stage pipeline: fetch, decode and execute [13]. 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 [20]. 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 busses. Although the 4GB 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 [11]. Certain features of the processor are summarized below.

- **Shortest instruction execution time**:
  - 800 ns (at f = 80 MHz)
- **Registers**:
  - 30 general purpose registers
  - 6 status registers
  - A 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 are suitable platforms for these applications due to their low power requirements. Furthermore, a 32-bit RISC architecture makes it easy to port many different applications. This kind of microcontrollers are also very easy to implement. They are available as small cores which are easy to integrate. Another advantage is on-chip debug support. These advantages make this family a good match for embedded, wireless 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. 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 Thumb 16-bit instruction set. 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 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 [8]:

- **Code Densitys** shows how much memory is required for a given high level C code. The smaller size will result in a reduced cost.
- **Performance** relates to he processor's clock speed which is an important factor. The smaller the clock rate to execute given algorithms, the less the power consumed. This will also lead to easier designs. The 32-bit RISC controllers will spend most of its time in an idle mode resulting in saving power.
- **Power Consumption** is one of the most important factors in wireless technology. The lower power consumption will make the batteries life longer, the device size smaller, and the price cheaper. The ARM7TDMI consumes about 1.85 mW per MHz. On the other hand, the StrongARM runs up to 233MHz and consumes a total of 900 mW [8].

ARM7TDMI is widely accepted and used in the cellular phone and smart phone technology due to its low cost and power efficiency. The future prospects show that ARM9TDMI will probably replace ARM7TDMI. Integrating the DSP module with ARM7 family will produce the new ARM9 family [7].

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 standardized 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 creation of our library, we did not make any assumption on 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, therefore our library is scalable. The machine word size is 32-bit on the ARM microprocessor. The library is implemented in 27 kilobytes of code size. The modified Jacobian coordinates are used to represent the points on the curves since it gives the fastest point doubling timings. 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. Particularly, 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 about how fast these operations could be done in today's technology. Figure 6 shows the timings of the operations for variable ECC key lengths.

Figure 6: The timings in milliseconds.

ECC →	160	176	192	208	256
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 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 [9], the SHA-1 requires approximately 2 ms digesting 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 being 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 Figure 6, 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 2 msec timing value for processing a block of data using SHA-1.

9. 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 a way that the implemented design will permit the use of increased key lengths.

In our implementation, we created an ECC library, which 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 being used in restricted platforms as well as high-end servers. With this implementation, we obtained timing results less than 100 ms for both the ECC-160 signature generation and verification on a 32-bit ARM processor. In addition, the timing results were obtained for a recently proposed wireless authentication and key agreement protocol [1]. 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.

Possible enhancements for further speeding up and/or reducing the code size are:

- The scalar multiplication of the base point can be performed in more efficient way by having a precomputed look-up table in ROM area.
- 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.
- The 16-bit wide *Thumb* instruction set of ARM7TDMI can be used to reduce the code size.

10. Acknowledgements

This research was supported by Secured Information Technology, Inc.

References

[1] M. Aydos, B. Sunar, and Ç.. K. Koç. An elliptic curve cryptography based authentication and key agreement protocol for wireless communication. In *2nd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications Symposium on Information Theory*, Dallas, Texas, October 30, 1998.

[2] A. Aziz and W. Diffie. A secure communications protocol to prevent unauthorized access: Privacy and authentication for wireless local area networks. *IEEE Personal Communications*, pages 25–31, First Quarter 1994.

[3] M. J. Beller, L.-F. Chang, and J. Yacobi. Privacy and authentication on a portable communications systems. *IEEE Journal on Selected Areas in Communications*, 11(6):821–829, Aug. 1993.

[4] I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography*. Cambridge University Press, New York, NY, 1999.

[5] Certicom. SigGen Smart Card. `http://205.150.149.57/ce2/embed.htm`, 1997.

[6] H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates. In K. Ohta and D. Pei, editors, *Advances in Cryptology – ASIACRYPT 98*, Lecture Notes in Computer Science, No. 1514, pages 51–65. Springer, Berlin, Germany, 1998.

[7] O. Gunasekara. Smart phone challenges. `http://www.arm.com/Documentation/White Papers/SmartPhone`, 1997.

[8] O. Gunasekara. Developing a digital cellular phone using a 32-bit microcontroller. `http://www.arm.com/Documentation/White Papers/CellPhone`, 1998.

[9] T. Hasegawa, J. Nakajima, and M. Matsui. A practical implementation of elliptic curve cryptosystems over $GF(p)$ on a 16-bit microcomputer. In H. Imai and Y. Zheng, editors, *First International Workshop on Practice and Theory in Public Key Cryptography*, Lecture Notes in Computer Science, No. 1431, pages 182–194. Springer, Berlin, Germany, 1998.

[10] IEEE. P1363: Standard specifications for public-key cryptography. Draft Version 13, November 12, 1999.

[11] ARM Incorporated. *Advanced RISC Machines Architectural Reference Manual*. Prentice-Hall, New York, NY, 1996.

[12] K. Itoh, M. Takenaka, N. Torii, S. Temma, and Y. Kurihara. Fast implementation of public-key cryptography on a dsp tms320c6201. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems*, Lecture Notes in Computer Science, No. 1717, pages 61–72. Springer, Berlin, Germany, 1999.

[13] D. Jaggar. ARM architecture and systems. *IEEE Micro*, pages 9–11, July/August 1997.

[14] N. Koblitz. *A Course in Number Theory and Cryptography*. Springer, Berlin, Germany, Second edition, 1994.

[15] Ç. K. Koç. High-Speed RSA Implementation. Technical Report TR 201, RSA Laboratories, 73 pages, November 1994.

[16] Ç. K. Koç, T. Acar, and B. S. Kaliski Jr. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, June 1996.

[17] A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. In *The 3rd Workshop on Elliptic Curve Cryptography (ECC 99)*, Waterloo, Canada, November 1–3 1999.

[18] A. J. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, Boston, MA, 1993.

[19] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, Apr. 1985.

[20] S. Segars. ARM7TDMI power consumption. *IEEE Micro*, pages 12–19, July/August 1997.

— Notes —