EMBEDDED END-TO-END WIRELESS SECURITY WITH ECDH KEY EXCHANGE

Sandeep Kumar[1]*,Marco Girimondo*[1]*,André Weimerskirch*[1]*,Christof Paar*[1]*,Arun Patel*[2]*,Arvinderpal S. Wander*[3]

[1]Communication Security Group,Ruhr-Universität Bochum, Germany
{kumar, girimondo, weika, paar}@crypto.rub.de
[2]Sun Microsystems Laboratories, California, USA
arun.patel@sun.com
[3]University of Michigan, USA
awander@umich.edu

ABSTRACT

Sensor networks offer tremendous benefits for the future as they have the potential to make life more convenient and safer. For instance, sensors can be used for climate control to reduce power consumption, for structures such as bridges to monitor the maintenance status, or for company badges to locate employees in order to increase productivity. However, the introduction of such ubiquitous computing to everyday life also raises privacy concerns. In this work we present a public-key cryptography implementation for secure key exchange on low-end wireless devices using elliptic curves. Our implementation is based on optimal extension fields (OEF) that are a special type of finite fields $GF(p^m)$. As our platform we chose a Chipcon CC1010 chip which is based on the 8051 architecture and that is especially suited for secure wireless applications as it has a built-in radio transceiver as well as a hardware DES engine [2]. We are able to establish a secure end-to-end connection between the sensor and a base station in an acceptable time of 3 seconds without requiring a cryptographic coprocessors.

1. INTRODUCTION

Recently, there has been increasing interest in low-cost radio-enabled devices to be used as sensors and for pervasive computing. With the rising number of modern appliances becoming networked with each other, there is a growing demand for cost-effective wireless networking. The main disadvantage of using wireless networking is that an eavesdropper can easily intercept communications. Hence, security is a major concern in such devices.

Low cost and power constraints have limited cryptographic implementations on these devices to only symmetric key algorithms. While symmetric key algorithms provide privacy of communication, they require both parties to know a shared secret a priori. Protocols based on public-key algorithms such as RSA and DSA [3] exist which can be used to set up these keys. However, RSA and DSA are based on modular arithmetic using large operands, and therefore are very resource intensive both in terms of time and memory. Consequently, these particular algorithms are poorly suited for low-end processors such as the 8051. Public key techniques also facilitate the use of digital signatures, which provide the security goal of non-repudiation. The scalability of key distribution is also made possible by using public key cryptography.

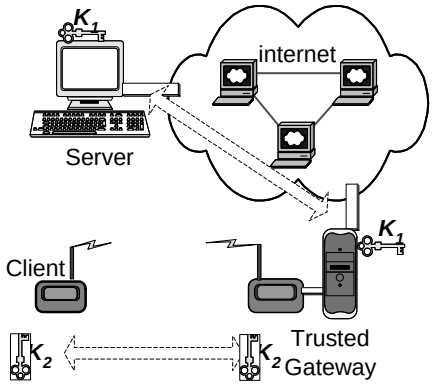

Figure 1. Network security based on a trusted gateway.

The main aim of this work is to prove that public-key cryptography can indeed be used on low-end 8-bit processors such that it provides adequate security for establishing the secret keys required for secure wireless connections. A further objective was to provide end-to-end security between communicating devices. These goals were met by leveraging the computational savings provided by Elliptic Curve Cryptography (ECC). Figure 1 displays a possible application scenario. There is a client that establishes a secure communication channel to a gateway over a wireless channel. The gateway itself establishes a secure channel to

a server through an insecure Internet connection such that finally there is end-to-end security established between the client and the server by means of the secure gateway.

This paper is structured as follows. In Section 2 we give an introduction to ECC, and Section 3 describes the Elliptic Curve Diffie-Hellman (ECDH) key exchange. Section 4 describes the communication protocol based on the ECDH key exchange. Section 5 describes our demonstration application, and finally we conclude with results and future work in Section 6.

2. ELLIPTIC CURVE CRYPTOGRAPHY

Elliptic curve cryptography is a promising branch of public key cryptography which offers similar security to other public key algorithms in use today such as RSA but with smaller key sizes and memory requirements. ECC was first proposed independently by Koblitz [6] and Miller [8] in 1985. At present, ECC has been commercially accepted, and has also been adopted by many standardizing bodies such as ANSI, IEEE, ISO, and NIST.

ECC operates over points on an elliptic curve defined over a finite field. It relies on the assumed difficulty of the elliptic curve discrete logarithm problem (ECDLP), which is given points P and $Q = k \cdot P$ (where $\cdot$ is a *scalar point multiplication*) it is computationally intractable to determine k. The elliptic curve point multiplication (which is the main operation in ECC) can be expressed in terms of arithmetic operations over the finite field. Thus the efficiency of the finite field arithmetic, especially the field multiplication, determines the overall efficiency of the elliptic curve cryptosystem.

2.1 Finite Field Choice

A finite field is denoted as $GF(p^m)$ for p prime and m a positive integer. There exist finite fields for all primes p and positive integers m. $GF(p^m)$ is isomorphic to $GF(p)[x]/(P(x))$, where $P(x) = x^m + \sum_{i=0}^{m-1} p_i x^i$, $p_i \in GF(p)$, is a monic irreducible polynomial of degree m.

Various finite fields allow the use of different algorithms for field arithmetic, thus the choices of p, m, and $P(x)$ can have a dramatic impact on the cryptosystem performance. In finite fields of a particular form, there are specialized algorithms which give better performance than generic algorithms. We choose Optimal Extension Fields (OEF) [1] as they are computationally more efficient than other field types offering roughly the same level of security. Table 1 [12] displays three fields of similar field order which implies a similar security strength of the cryptosystem based on these fields. The third column denotes the number of cycles for one field multiplication which is the crucial operation. One can see that the OEF displayed in the third row performs more efficiently than the binary field displayed in the first row.

Table 1. ECC field performance

Field	Field order	♯ Cycles for multiply
$GF(2^{135})$	2^{135}	19,600
$GF((2^8)^{17})$	2^{136}	7,479
$GF((2^8-17)^{17})$	2^{134}	5,084

2.2. Optimal Extension Fields

In OEF's, p is of the form $2^n \pm c$, for n, c positive integers and $\log_2 c \leq \lfloor \frac{1}{2} n \rfloor$. One chooses p of appropriate size to exploit the multiply instruction available on the 8051 processor, i.e., $p \leq 2^8$. m is chosen such that an irreducible binomial $P(x) = x^m - \omega$ exists which speeds up the extension field modular reduction. The goal is to select parameters which provide adequate security without incurring excessive computation time. Based on this analysis, we chose $p = 2^8 - 17$ and the irreducible polynomial $P(x) = x^{17} - 2$, i.e., $m = 17$.

Lenstra and Verheul showed that under certain assumptions, 952-bit RSA and DSA systems may be considered equivalent in security to a 132-bit ECC system [7]. The field $GF((2^8-17)^{17})$ provides a security level of 134 bits such that the proposed system is far more secure than 512-bit RSA system which has been popular for smart card applications. Therefore, our selection of the field order provides a security level which is appropriate to protect data for a medium time interval, say one year. This should be sufficient for most embedded applications.

2.3. Field Arithmetic

The key performance advantage of OEFs is due to fast modular reduction in the subfield. Given prime $p = 2^8 - 17$, reduction of a double-sized operand c is performed by dividing it into two 8-bit words, $c = c_1\ 2^8 + c_0$, where $c_0, c_1 < 2^8$. The upper bits of c are "folded" into the lower ones, $c = 17c_1 + c_0 \bmod (2^8 - 17)$, leading to a very efficient reduction which requires one multiplication by 17, one addition and no division or inversions.

For field multiplication, we observe that $x^{17} = 2 \bmod (x^{17} - 2)$. Thus we can express $C(x) = A(x) \cdot B(x) = \hat{c}_{16}x^{16} + (2\hat{c}_{32} + \hat{c}_{15})x^{15} + \ldots + (2\hat{c}_{18} + \hat{c}_1)x + (2\hat{c}_{17} + \hat{c}_0) \bmod (x^{17} - 2)$ where $\hat{c}_i = \sum_{j+k=i} a_j b_k$.

Squaring is similar to multiplication with the advantage that the two operands are same. Therefore the $\hat{c}_i$'s are faster to calculate. Inversion is implemented using the Itoh-Tsujii algorithm [4] in the sub-field $GF(p^{17})$ as $A^{-1} = (A^r)^{-1} A^{r-1}$ where $r = (p^{17} - 1)/(p - 1)$. Table 2 presents the execution time and code size of the field operations.

Table 2. Field arithmetic performance

Description	Operation	Time (μsec)	Code size (bytes)
Multiplication	$A(x)B(x)$	5093	5212
Squaring	$A^2(x)$	3142	3400
Inversion	$A^{-1}(x)$	24672	neg.

2.3.1 Point Arithmetic

There are two methods for representing points on an elliptic curve: affine coordinates, and projective point coordinates. Projective coordinates are used to save field inversions at the cost of further multiplications. We use affine point coordinates since the ratio for multiplication time to inversion time is 1:4.8. If $P = (x_1, y_1) \in GF(p^m)$, then $-P = (x_1, -y_1)$. If $Q = (x_2, y_2) \in GF(p^m)$, $Q \neq -P$, then $P + Q = (x_3, y_3)$, where

$$x_3 = \lambda^2 - x_1 - x_2 \tag{1}$$

$$y_3 = \lambda(x_1 - x_3) - y_1 \tag{2}$$

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1}, \text{ if } P \neq Q \\ \frac{3x_1^2 + a}{2y_1}, \text{ if } P = Q \end{cases} \tag{3}$$

If $P = Q$ we call the operation a point doubling, otherwise a point addition. For point multiplication we use the binary double-and-add method as described in [5].

3. Elliptic Curve Diffie-Hellman

In the elliptic curve Diffie-Hellman (ECDH) key exchange, the two communicating parties server S and client C agree beforehand to use the same curve parameters and base point G. They each generate their private keys Pr_S and Pr_C, respectively, and the corresponding public keys $Pu_S = Pr_S \cdot G$ and $Pu_C = Pr_C \cdot G$.

Both the client and server exchange their public keys, and each multiplies its private key with the other party's public key to derive a common shared secret $Pr_C \cdot Pu_S = Pr_S \cdot Pu_C = Pr_S \cdot Pr_C \cdot G$. An attacker cannot determine this shared secret from the curve parameters, G or the public keys of the parties as described in Section 2.

4. Communication Protocol

Our communication protocol consists of two phases. The first one is the *key establishment phase* which is done initially to exchange the keys. Thereafter in *normal mode* application data is transmitted. Figure 2 displays the protocol which is described below.

4.1. Key Establishment Phase

The client initiates a connection by relaying a *Client Hello* with its pre-computed public key Pu_C on the wireless channel. The gateway, which is in receive mode, on receiving the public key passes it to the server through the wired interface and waits for the server's public key. The server daemon, upon receiving the connection request sends a *Server Hello* with its public key Pu_S to the gateway, and then starts computing the shared secret from the received public key Pu_C and the server's secret key Pr_S.

$$MSecret = Pr_S \cdot Pu_C \tag{4}$$

The gateway transmits the server's public key to the client and waits for the data transmission to begin. The client, on receiving the server's public key (Pu_{ser}) from the gateway, computes the *master secret*.

$$MSecret = Pr_C \cdot Pu_S \tag{5}$$

The client and server now have the same shared master secret *MSecret* of 134 bits using the Elliptic Curve Diffie-Hellman key exchange. The key for the symmetric DES operation is then derived from this 134-bit secret.

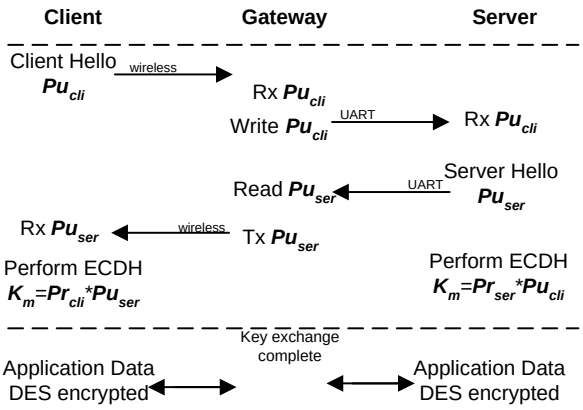

Figure 2. Key exchange protocol.

4.2. Normal Mode

Once the keys are set up, the client and the server can transmit application data encrypted with Triple-DES in CFB mode. To close the connection securely, we use a *close connection* control message which deletes the previously generated keys.

5. Demonstration Implementation

The hardware platform chosen for the implementation is the Chipcon CC1010 chip [2], an 8-bit 8051 processor core with a built-in radio transceiver and hardware DES engine. The CC1010 has been optimized to execute one instruction cycle every four clock cycles, which offers roughly 2.5 times the performance of the original Intel 8051. It is a

very power efficient device which allows for use in mobile devices. We use two CC1010 evaluation modules for our setup. One of them is used as a *client*, which is connected to a portable magnetic stripe reader, similar to a Point-of-Sale terminal. The reader presents the encoded data through an RS-232 serial link to the CC1010. The other evaluation module is used as a *gateway*. It listens to the wireless channel and transmits the data to a PC connected through a RS-232 serial link. The client device operates on three 1.5V AA batteries, and communicates with the server through the gateway on a radio frequency of 868Mhz.

The CC1010 contains 32 kilobytes of flash memory for storing programs, 2048 bytes of SRAM external to the 8051 core, and 128 bytes of internal SRAM. Due to the extreme limits in terms of memory capacity and processing power, ECC was implemented in assembly for efficiency. The wireless communication protocol was written in C and cross-compiled using Small-Devices C Compiler (SDCC) [10] tools. On the server side, ECC algorithms were implemented with OpenSSL [9] and the Number Theory Library (NTL) [11].

An exchange begins when a user swipes a card with a magnetic stripe, such as a credit card, on the client device. The client first saves the data encoded on the magnetic stripe and then initiates the key exchange protocol described in Section 4. After the 134-bit shared secret is established, we use the first 112 bits as the key for the Triple-DES engine. The card data is then encrypted with Triple-DES in CFB mode on the client, and decrypted and displayed on the server side. The wireless range of this demonstration exceeds 100ft indoors. The client can operate on the same set of batteries for at least 10 hours continuously, and longer battery life can be achieved with additional power management software.

6. Results and Future Work

In this paper we have shown that security protocols based on public key cryptographic algorithms are possible on low-end wireless devices without the extra cost of additional hardware. The code size of the elliptic curve point multiplication library is 13.5 kilobytes, while the overall demonstration program occupies 22.5 kilobytes. The total RAM used also includes variables for the field arithmetic operations, storage of temporary points and the secret key (an integer coefficient), and buffers for the communication protocol. The breakdown of memory usage is described in Table 3.

It takes 2.99 seconds to complete an elliptic curve point multiplication on the Chipcon platform. The overall session setup time for the secure connection takes 3.15 sec. Table 4 furthermore displays the execution time for a point doubling and a point addition operation.

Table 3. Memory map for CC1010

Type	Size (bytes)	Function
Code	13.5k	ECC
	9k	RF protocol
Internal RAM	128	finite field arithmetic
External RAM	406	temporary points
	34	coefficients

Table 4. ECC point arithmetic performance

Operation	Time (msec)
Point Addition	14.049
Point Doubling	15.395
Point Multiplication	2999.8

The time taken for setting up the connection is acceptable, considering the additional security it enables. A working model of a wireless card reader was implemented to prove the concept in practice.

In future versions we will add a hash function to change the keys regularly, or to derive session keys of a master secret key to allow a rapid reconnection. Certified public keys are also required in future implementations to avoid man-in-the-middle attacks. Furthermore, we plan to implement a digital signature scheme based on our elliptic curve library.

References

[1] D. Bailey and C. Paar. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. *Journal of Cryptology*, 14, 2001.

[2] Chipcon. *SmartRF. CC1010 PRELIMINARY Datasheet (rev. 1.2)*, 2003.

[3] A. O. Freier, P. Karlton, and P. C. Kocher. *The SSL Protocol Version 3.0*. Transport Layer Security Working Group INTERNET-DRAFT, November 1996.

[4] T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases. *Information and Computation*, 78:171–177, 1988.

[5] D. E. Knuth. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms*. Addison-Wesley, Reading, Massachusetts, USA, 2nd edition, 1981.

[6] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, vol. 48:203–209, 1987.

[7] A. Lenstra and E. Verheul. Selecting cryptographic key sizes. In H. Imai and Y. Zheng, editors, *PKC 2000*, volume LNCS 1751, Berlin, 2000. Springer-Verlag.

[8] V. S. Miller. Use of elliptic curves in cryptography. *CRYPTO '85*, pages 417–426, 1986.

[9] Openssl. available at http://www.openssl.org/.

[10] Sdcc - small device c compiler. available at http://sdcc.sourceforge.net/.

[11] V. Shoup. Ntl. available at http://www.shoup.net/ntl/.

[12] A. Woodbury, D. V. Bailey, and C. Paar. Elliptic curve cryptography on smart cards without coprocessors. In *CARDIS 2000*, Bristol, UK, September 20–22 2000. Kluwer.