Keywords: ECDSA, MAA, MAXQ1103, Elliptic Curve, ECC

APPLICATION NOTE 4016

Using the Elliptic Curve Digital Signature Algorithm (ECDSA) with the DeepCover Secure Microcontroller's (MAXQ1103) Modular Arithmetic Accelerator (MAA)

Jun 17, 2008

Abstract: This application note describes the use and implementation of elliptic curve digital signature algorithm (ECDSA) on the DeepCover® Secure Microcontroller's (MAXQ1103) modular arithmetic accelerator (MAA) module. Performance figures are given for a standard 160-bit curve specified in Standards for Efficient Cryptography published by Certicom®.

Overview

Modern electronic transactions have increased productivity and decreased cost in many financial and legal applications, all the while ensuring that the underlying trust inherent in the transaction is not violated. With the advent of ever-increasing personal computational resources and easy access to information through the Internet, potential interlopers are more capable of intercepting and altering these transactions. In this article we focus on the use of the elliptic curve digital signature algorithm (ECDSA), a more advanced variant of the digital signature algorithm (DSA), to detect transactional tampering. There is a notable benefit to ECDSA over DSA: the resulting signature size is much smaller. The 160-bit version of the ECDSA described below is roughly equivalent to a 1024-bit DSA.

Introduction

Let us outline a typical scenario. Imagine that two parties, Alice and Bob, wish to execute a contract created by Alice electronically. Alice transmits a copy of this contract to Bob. Bob computes a message-specific hash value, and then uses ECDSA to create a signature based on this hash value. He sends this signature back to Alice. Alice runs the ECDSA verify routine over the signature, along with the hash value that she computed for her local copy of the contract. If the algorithm indicates success, Alice can now assert with confidence that Bob received an unaltered copy of the contract, and that he cannot claim not to have signed the contract. Meanwhile, Bob is assured that Alice could not have modified the content of the contract nor signed the contract herself without his private key information. Bob, of course, wants Alice to sign the contract as well in order to gain the same assurances.

Terminology

We will use these typographic conventions in the following sections:

- F_p is the finite field of order p
- G(x,y) is the function G, taking two variables x and y
- * means scalar multiplication carried out by repeated addition
- [and] function as normal arithmetical grouping operators

The Inner Workings of ECDSA

For every transaction involving ECDSA, all parties must agree on a set of values called the elliptic curve domain parameters. These parameters specify which curve will be used, and allow all parties to successfully sign and verify messages. These parameters are usually given as a sextuple T = (p, a, b, G, n, h). However, we will only reference p, G, and n in this document, as they are the parameters actually used for signatures and verification. Picking curve parameters is a complex and tedious job, fraught with pitfalls. Therefore, we usually use well-known published curves[1], which have been tested for known weaknesses.

Note: Many of the underlying operations used in ECDSA are explained in the *Online Elliptic Curve Cryptography Tutorial*.[2] This tutorial will guide the reader on elliptic curve point multiplication and addition.

We now return to Bob. For Bob to generate his electronic signature, it must randomly pick a private key (d) from a field of numbers ranging from [1 ... n-1], where n is a prime number agreed upon by all parties. In addition to n, several other numbers are agreed upon in advance and are called the curve parameters. The other two known parameters are G(x,y) and p. G(x,y) is called the Base Point; p is the order of the finite field containing G(x,y). The point G(x,y) represents a point on the chosen elliptic curve in affine space.

Bob then takes his private key (d) and multiplies it by the base point (G) in the field F_p, obtaining a public key Q(x,y). Equation 1 represents the generation of Bob's public key.

$$Q(x,y) = [G(x,y) * d] \bmod p \qquad \text{(Eq. 1)}$$

The public key can be distributed by any convenient means, even by insecure email, as it can only be used to verify that the person holding the private key signed a message. The public key and private key are typically generated only once, and form the permanent credentials of the principal (Bob, in this case). Key management is beyond the scope of this document, and more information can be found elsewhere.[3]

When Bob receives his copy of Alice's contract, he feeds the copy to the Secure Hash Algorithm 1 (SHA-1) routine to generate a unique 160-bit value. SHA-1 is currently used for ECDSA signatures, but may be supplanted in the near future by newer, more secure hash algorithms.

Bob is now ready to compute the actual signature. This signature has two components, named r and s. Only the s component depends on the message hash H(m), so r can be precomputed. To sign this message, Bob picks a new, per-signature random number (k) in the range of [1 ... n-1], and multiplies this number by G within F_p. The result is an affine coordinate point (x_1,y_1). The nonmessage portion of

the signature is r = x_1 mod n, and y_1 is discarded. For ECDSA, F_n is a superset of F_p. Thus, Bob does not actually need to run the modulo operation over x_1 to obtain r. Equation 2 shows the generation of r.

$$(x_1,y_1) = [G(x,y) * k] \bmod p$$
$$r = x_1 \bmod n \qquad \text{(Eq. 2)}$$

Bob can compute the message-dependant portion of the signature(s) by first finding the modulo inverse of k, which we shall call k^{-1}. Then, he multiplies his secret key (d) by the r value found in the previous equation, adds the result to the message hash value H(m), and multiplies the entire result by k^{-1}. All operations for s are performed in the field F_n. Equation 3 shows the generation of s.

$$s = [k^{-1} * (H(m) + d * r)] \bmod n \qquad \text{(Eq. 3)}$$

If either s or r results in a zero value, Bob must pick a new random (k) and start the signature over again.

Bob can now transmit his ECDSA digital signature pair (r,s) to Alice as proof that he signed the document with the hash H(m).

To verify that Bob signed her contract, Alice needs: her local copy of the contract, Bob's public key, and the signature pair (r,s) sent by Bob for this transaction. First, Alice performs a "sanity check" over (r,s) to ensure that both components exist in the field F_n, excluding the element 0 (i.e., [1 ... n-1]). Should this check fail, the signature is immediately declared invalid. Next, several intermediate products are computed in Equations 4, 5, and 6 for signature verification.

$$w = s^{-1} \bmod n \qquad \text{(Eq. 4)}$$
$$u1 = [H(m) * w] \bmod n \qquad \text{(Eq. 5)}$$
$$u2 = [r * w] \bmod n \qquad \text{(Eq. 6)}$$

Now, Alice can perform a final calculation (Equation 7) using the intermediate products and can check the validity of Bob's signature.

$$(x_2,y_2) = [u1 * G(x,y) + u2 * Q(x,y)] \bmod n \qquad \text{(Eq. 7)}$$

Alice compares the x_2 as a scalar against the r coordinate of Bob's signature. If they match exactly, then the signature is authentic. Otherwise, the signature is invalid.

If Alice determines that Bob's signature is valid, she knows that Bob signed an exact copy of her contract. She also knows that Bob cannot deny signing without admitting that he lost control of his secret key. This property is called nonrepudiation. If Alice determines that the signature is invalid, she only knows that the signature or message (and, thus, the hash value) was altered after it left her control.

Theory into Practice: DeepCover Secure Microcontroller (MAXQ1103) ECDSA Performance

As implemented for the DeepCover® Secure Microcontroller (MAXQ1103), the ECDSA signature and

verify routines are very efficient for a microcontroller of its class. The implementation utilizes the MAXQ1103's integrated MAA running at approximately 55MHz. As the MAA is clocked from a free-running ring oscillator, the exact speed and timing cannot be known precisely, which adds a layer of defense against differential power-analysis cryptographic attacks.

Table 1. Timing for ECDSA Operations over the p160r1 Curve

Operation	160-Bit Normal	160-Bit Accelerated
Key Generation	77ms	18ms
Sign	75ms	19ms
Verify	148ms	79ms

Table 1 shows the nominal timings for ECDSA routines for a 55MHz MAA clock. The 160-bit elliptic curve p160r1 was selected from Certicom Research's *Recommended elliptic curve domain parameters*.[1] Of course, some of the nonmodular computational work is done by the MAXQ1103 CPU, which will affect the timings to a small degree, depending on the core frequency selected.

Conclusion

The MAXQ1103 microcontroller provides a secure, efficient platform for computation and verification of ECDSA signatures. By using the MAA module, the microcontroller's CPU can continue to process other tasks while modular math computations progress. Differential power analysis and timing attacks are thwarted by a free-running ring oscillator used to clock the MAA, which is itself not synchronized to any other clock on the die.

Note: The MAXQ1103 software used to generate this application note is available under NDA. Please contact us through our Maxim Support Center for more information.

References

[1] Certicom Research. *SEC 2: Recommended elliptic curve domain parameters*, September, 2000 (www.secg.org/secg_docs.htm).
[2] www.certicom.com/index.php?action=ecc_tutorial,home.
[3] A. Menezes, et al., *Handbook of Applied Cryptography*, 1st ed. (CRC Press, 2001).

Related Parts	
MAXQ1103	DeepCover Secure Microcontroller with Rapid Zeroization Technology and Cryptography

More Information
For Technical Support: http://www.maximintegrated.com/support
For Samples: http://www.maximintegrated.com/samples
Other Questions and Comments: http://www.maximintegrated.com/contact

---

Application Note 4016: http://www.maximintegrated.com/an4016
APPLICATION NOTE 4016, AN4016, AN 4016, APP4016, Appnote4016, Appnote 4016