Elliptic Curves over Prime and Binary Fields in Cryptography

Authors

Dana Neustadter (danan@ellipticsemi.com)

Tom St Denis (tstdenis@ellipticsemi.com)

Elliptic Curve Cryptography (ECC)

- Public key (asymmetric) cryptosystem
- Based upon a hard number theoretic problem: Elliptic Curve Discrete Logarithms (ECDL)
- At the base of ECC operations is finite field (Galois Field) algebra with focus on prime Galois Fields (GF(p)) and binary extension Galois Fields ($GF(2^m)$)
- Standardized by NIST, ANSI and IEEE: NIST, NSA Suite B, ANSI X9.62, IEEE P1363, etc.

Elliptic Curve Discrete Logarithms

- ECDL is a so called “trap-door” or “one-way” function
- Given an elliptic curve and points P and Q on the curve, find integer k such that Q = k * P
- Relatively easy to use to transform data one-way, but having the result and the transformation key does not easily give the input:
  - encryption - is easy to compute
  - decryption - much more complicated if not impossible to compute without knowing the trap-door
- The hardness of ECDL defines the security level of all ECC protocols

ECC Systems

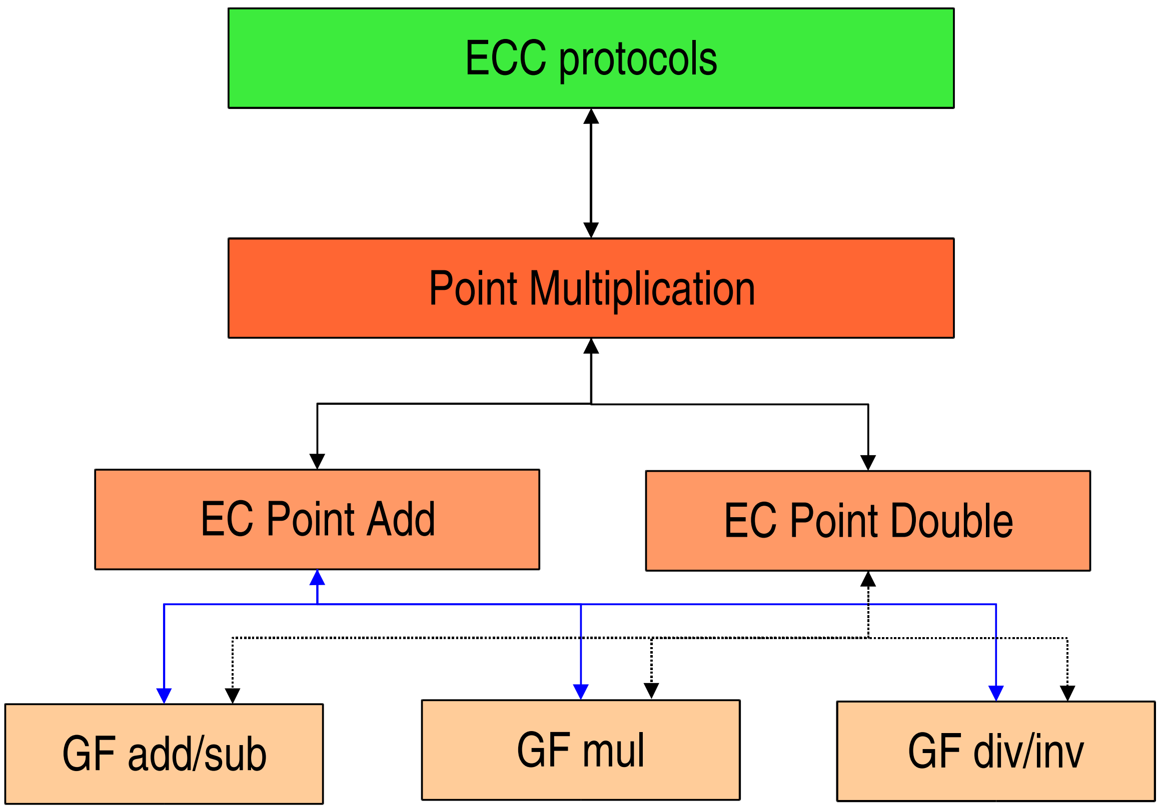
- Performance, security, size and versatility of ECC systems are a function of:
  - finite field selection
  - elliptic curve type
  - point representation type
  - algorithms used
  - protocol
  - key size
  - hardware only, software only or mixed hardware-software implementations
  - memory available (table lookups)
  - code and area

ECC Operations Hierarchy

- First level: basic Galois Field operations
  - GF addition
  - GF multiplication
  - GF inversion
- Second level: Elliptic Curve point operations
  - Point Add
  - Point Double
- Third Level: Elliptic Curve point operation
  - Point Multiplication – the fundamental and most time consuming operation in ECC
- Fourth Level: ECC protocol
  - ECDSA, ECDH, ECMQV, El-Gamal, ...

Elliptic

ECC Operations Hierarchy



Finite (Galois) Fields

- Finite Field = A finite group of prime characteristic (with defined ring structure, and multiplicative structure)
- The number of units in the finite field is determined by the "field order" which is based on a prime number or the power of a prime number
- Allow for fields to be practically manipulated with full accuracy

Galois Fields

- Galois Field algebra is at the base of ECC operations and protocols
- Best suited for cryptographic applications and primarily used:
  - Prime fields GF(p)
    - operations are done modulo prime number p
  - Binary extension fields $GF(2^m)$
    - operations are done modulo an irreducible polynomial F(t)
  - Binary composite fields $GF((2^m)^n)$
  - Prime extension fields $GF(p^m)$
    - Edward Curves (Bernstein et al.)

Prime Galois Fields

- GF(p) = prime field of order *p*
- GF (p) contains *p* elements, p – 1 units
- Field elements are residue classes *modulo p*
- At the basis of GF(p) related operations is integer modular arithmetic
- Basic operations
  - addition (GF add) : a + b mod p
  - subtraction (GF sub) : a – b mod p
  - multiplication (GF mul) : a x b mod p
  - division (GF div) : a / b mod p
  - inversion (GF inv) : 1 / b mod p

Prime Galois Fields

- Algorithms
  - Reduction techniques
    - Reduced Radix (NIST curves)
    - Montgomery (more practical)
  - Multiplication techniques
    - Comba multipliers
    - Karatsuba (less so)
  - Inversion (dominant last step)
    - Euclids
    - Almost Inverse

Prime Galois Fields

- Commonly used for software implementations because the integer arithmetic is more optimized in today's microprocessors
- Desktops: favour fast multipliers
- Embedded: varies based on processor architecture
- Hardware implementations benefit from the full size operands but the area impact may be significant
- Hardware implementations carry chain timing challenges

Prime Galois Fields

- Integer Multiply and Accumulate
  - Multiply and accumulate is the inner dominant step for multiplication and squaring
  - With Comba it requires a 3x wide accumulator and a 2x wide product
  - Examples:

```
x86_32
movl   %6,%%eax
mull   %7
addl   %%eax,%0
adcl   %%edx,%1
adcl   $0,%2
```

```
ARM_V5
UMULL  r0,r1,%6,%7
ADDS   %0,%0,r0
ADCS   %1,%1,r1
ADC    %2,%2,#0
```

Prime Galois Fields

- Integer Multiply and Accumulate
  - Examples:

```
PPC32
mullw  16,%6,%7
addc   %0,%0,16
mulhwu 16,%6,%7
adde   %1,%1,16
addze  %2,%2
```

```
MIPS32
multu  %6,%7
mflo   $12
mfhi   $13
addu    %0,%0,$12
sltu   $12,%0,$12
addu    %1,%1,$13
sltu   $13,%1,$13
addu    %1,%1,$12
sltu   $12,%1,$12
addu    %2,%2,$13
addu    %2,%2,$12
```

Prime Galois Fields

- Large field order is more challenging for standard computers
  - The elements of the field have to be represented by multiple words
  - Carries between words have to be propagated
    - Comba technique pays off, reduces carry chain to small three-register chain
  - The reduction operation has to be performed across multiple words
    - NIST's “reduced radix” form is generally impractical in software
    - Montgomery reduction used predominantly

Prime Extension Fields

- Fields of form GF(p^q) for some prime p
  - p is usually either very small (large q) or relatively moderate (smaller q)
- Can lead to "Optimal Extension Fields" where p fits in a machine register (larger q)
- Removes the requirement to propagate carries
- Fast inversion algorithms exist
- Reduction *can* be more complicated than straightforward integer Montgomery

Binary Extension Fields GF (2^m)

- Finite field with 2^m elements: GF(2^m) = GF(2)[x] / F(x)
  - GF(2)[x] is a set of polynomials over GF(2)
  - $F(x) = x^m + f_{m-1}x^{m-1} + ... + f_2x^2 + f_1x + 1$ is the irreducible polynomial (trinomial and pentanomial primarily used)
  - f_i are GF(2) elements
- Basic operations
  - addition (GF add) : A(x) + B(x)
  - subtraction (GF sub) : A(x) – B(x)
  - multiplication (GF mul) : A(x) x B(x) mod F(x)
  - division (GF div) : A(x) / B(x) mod F(x)
  - inversion (GF inv) : 1 / B(x) mod F(x)

Binary Extension Fields

- Two main advantages regarding the Binary Finite Field math GF(2):
  - the bit additions are performed *mod 2* and hence represented in hardware by simple XOR gates => no carry chain is required
  - the bit multiplications are represented in hardware by AND gates
  - “1” is its own inverse => (1 = -1)
- The GF(2^m) elements can be viewed as vectors of dimension *m* where each bit can take values “0” or “1”
- All GF(2^m) field operations require *m*-bit operations which are more efficiently implemented in hardware because of GF(2) algebra properties (XORs, ANDs, no carry)

Elliptic

Binary Extension Fields

- Algorithms
  - Almost Inverse
    - Simple way to compute inverse with compact FSM with compact registers
  - Squaring
    - Free
  - Reduction can be accomplished in O(log n) time
    - Same is true for GF(p) but at a much higher size cost
  - Multiplication
    - Bit serial, digit serial, bit parallel

Binary Extension fields

- Not as efficient in SW implementations compared to prime fields where large multipliers are available
  - Integer multipliers can deal with word size data
  - Not true for smaller processors with inefficient integer multipliers
- Even more challenging for custom SW implementations if *m* is a large value
  - Challenging for SW implementations with reduced register space
- Usually use a sliding window dbl/add to speed up multiplication

Elliptic Curves

- An elliptic curve over a finite field has a finite number of points with coordinates in that finite field
- Given a finite field, an elliptic curve is defined to be a group of points (x,y) with $x,y \in GF$, that satisfy the following generalized Weierstrass equation:
  - $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$, where $a_i \in GF$
- Nonsupersingular EC over the finite binary field $GF(2^m)$
  - $y^2 + xy = x^3 + ax^2 + b \quad a, b \in GF(2^m)$
- EC over prime field GF(p)
  - $y^2 = x^3 + ax + b \quad a,b \in GF(p),\ 4a^3 + 27b^2 \neq 0,\ a = -3$ typically

Elliptic

Embedded security you can trust

Elliptic Curves

- Basic Point Operations
  - Point add: P(x,y) + Q(x,y)
  - Point double: 2 * P(x,y)
  - Point (scalar) multiplication: k * P(x,y), where $k \in [1, n-1]$ and n is the order of the EC base point
    - k * P(x,y) = P + P + ... + P (k summands)
    - Dominates the execution time in ECC
    - Requires multiple operations of point add and point double
    - Various algorithms available which are field type and coordinate representation dependent

Elliptic Curves

- Algorithms
  - EC over binary extension fields
    - Double and add
    - Montgomery scalar multiplication
    - Using Frobenius expansion, etc
  - EC over prime fields
    - Double and add
    - Fixed point
    - Shamir, etc

NIST Standard Elliptic Curves

- Pseudo-random curves over $GF(2^m)$
  - B-163, B-233, B-283, B-409, B-571
- Koblitz curves (special curves over $GF(2^m)$)
  - K-163, K-233, K-283, K-409, K-571
- Curves over prime fields GF(p)
  - P-192
  - P-224
  - P-256
  - P-384
  - P-521

Point Multiplication Performance

- Based on Elliptic's hardware and software solutions for B-233 and P-224 NIST Elliptic Curves
- Hardware IP
  - B-233: 4500 cyc/pmult (250k gates)
  - B-233: 800000 cyc/pmult (60k gates)
  - P-224: 900000 cyc/pmult (50k gates + memories)
- Software IP (on Power PC)
  - B-233: 5300000 cyc/pmult
  - P-224: 3500000 cyc/pmult

Conclusions

- Both prime and binary extension fields are finding uses in real world ECC applications
- The implementation of ECC solutions is highly dependent on the problem being solved, the implementation platform and the level of security intended to be achieved
- New finite field and elliptic curve types may emerge in ECC applications in the future

About Elliptic

- Incorporated August 2001
- Largest portfolio of volume proven security cores
  - 1st to market in several application spaces (MACsec, DTCP, others)
- Software and IP cores shipping in volume
- Security solutions spanning cores and middleware
- Customers in the U.S., Canada, China, Japan, Malaysia, Taiwan, Korea, Israel and Europe
- Partnerships with leading industry players including ARM, MIPS, RSA, Impinj, Lattice, Faraday
- NIST Certified – cores and software
- 20 Patents in process, 1 issued
- Investors: