Implementation Options for Finite Field Arithmetic for Elliptic Curve Cryptosystems

ECC '99

Christof Paar

Electrical & Computer Engineering Dept.
and
Computer Science Dept.
Worcester Polytechnic Institute
Worcester, MA, USA
http://www.ece.wpi.edu/Research/crypt

Contents

Why Public-Key Algorithms?

Traditional tool for data security: Private-key (or symmetric) cryptography

Main applications:

- Encryption
- Message Authentication

Traditional shortcomings:

1. Key distribution, especially with large, dynamic user population (Internet)
2. How to assure sender authenticity and non-repudiation?

Solution: Public-key schemes, e.g., Diffie-Hellman key exchange or digital signatures.

Practical Public-Key Algorithms

There are three families of PK algorithms of practical relevance:

Integer Factorization Schemes

Exp: RSA, Rabin, etc.
required operand length: 1024–2048 bits
arithmetic type: Integer ring Z_m

Discrete Logarithm Schemes

Exp: Diffie-Hellman, DSA, ElGamal, etc.
required operand length: 1024–2048 bits
arithmetic type: Finite field

Elliptic Curve Schemes

Exp: EC Diffie-Hellman, ECDSA, etc.
required operand length: 160–256 bits
arithmetic type: Finite field

Practical Aspects of PK Algorithms

Major problem in practice: All PK algorithms are relatively slow.

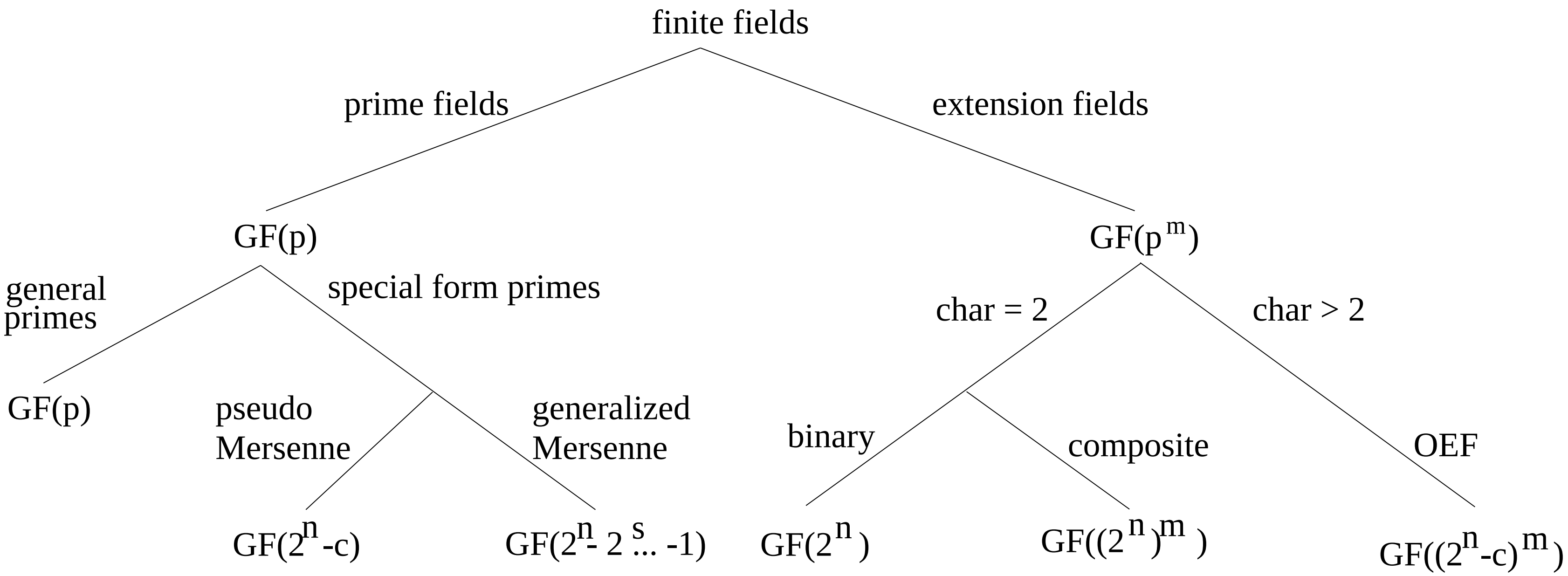
Observation: Algorithm speed is heavily dependent on arithmetic performance in HW and SW:

fast arithmetic $\Rightarrow$ fast PK algorithm

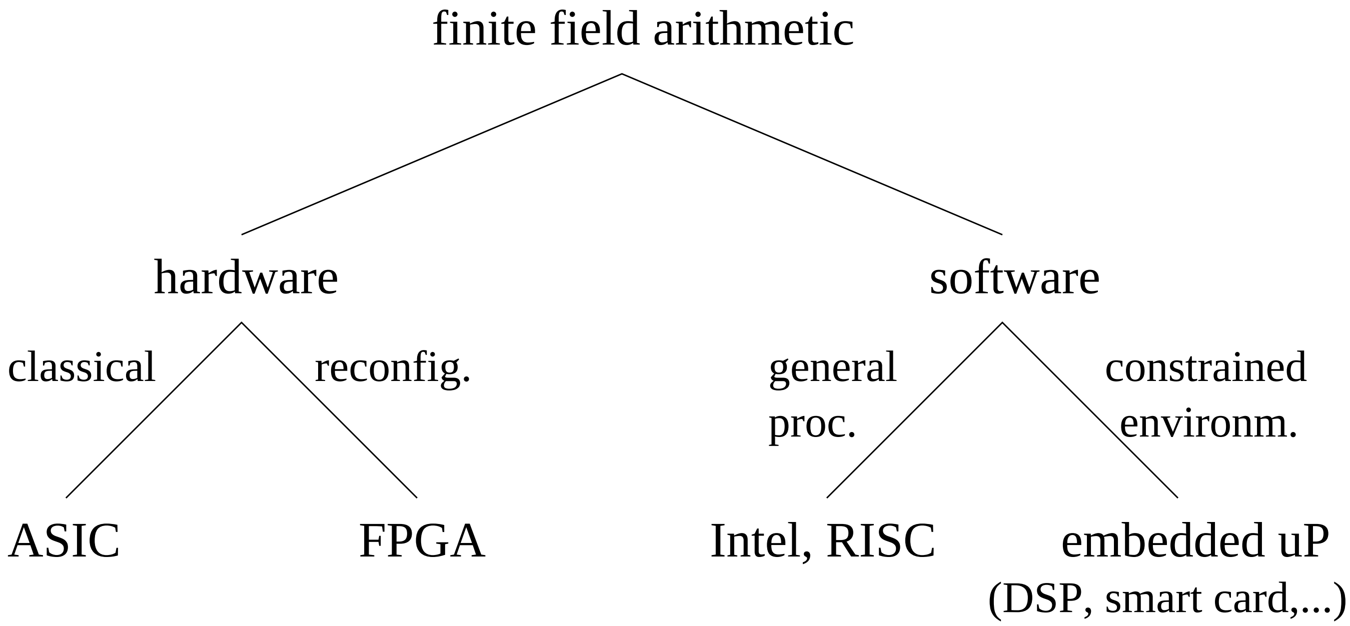
$\Rightarrow$ **Interdisciplinary Research area** (Computer Science, Electrical Engineering, Mathematics):

Efficient finite field arithmetic for discrete logarithm (DL) and elliptic curve cryptosystems (ECC)

Finite Fields Proposed for Use in PK Schemes



Platform Options



Arithmetic performance and area/cost greatly depends on:

1. Platform
2. Finite field type

with strong interaction:

platform choice $\Leftrightarrow$ finite field type

Prime Fields $GF(p)$

General remarks:

- preferred for DL systems
- also popular for ECC
- addition is cheap
- inversion is much slower than multiplication
  $\Rightarrow$ use of projective coord. for ECC
- "Remaining" problem: Efficient multiplication algorithms

Problem definition: Multiplication with long numbers (160–2048 bits) on processors with short word length (8–64 bits).

General Prime Fields $GF(p)$: Software

Exp: $A, B \in GF(p)$, $p < 2^{1024}$, word size $w = 16$ bit

element representation:

$$\begin{aligned} A &= a_{63}2^{63\cdot 16} + \cdots + a_1 2^{16} + a_0 \ , \ \ a_i \in \{0, 1, \ldots, 2^{16} - 1\} \\ B &= b_{63}2^{63\cdot 16} + \cdots + b_1 2^{16} + b_0 \ , \ \ b_i \in \{0, 1, \ldots, 2^{16} - 1\} \end{aligned}$$

1. Step: Multi-precision Multiplication

$$C' = A \cdot B = c'_{126}2^{126\cdot 16} + \cdots + c'_1 2^{16} + c'_0$$

where

$$\begin{aligned} c'_0 &= a_0 b_0 \\ c'_1 &= a_0 b_1 + a_1 b_0 + \text{carry} \\ &\vdots \end{aligned}$$

Complexity: $(n/w)^2$ inner products (integer mult), where $n = \lceil \log_2 p \rceil$.

Rem: Quadratic complexity can be reduced to $(n/w)^{1.58}$ using Karatsuba algorithm.

Further reading: [Menezes/van Oorschot/Vanstone 97]

General Prime Fields $GF(p)$: Software

2. Step: Modular reduction

$$C \equiv A \cdot B \bmod p \equiv C' \bmod p$$

1. (naïve) approach: long division of C' by p

2. (better) approach: fast modulo reduction techniques which avoid division:

   2.1. Montgomery

   2.2. Barrett

   2.3. Sedlack

   2.4. ... (see, e.g., [Naccache/M'Raïhi 96])

Complexity: $\approx (n/w)^2$ inner products + precomputations

Rem: Multi-precision mult (Step 1) and modular reduction (Step 2) can be interleaved.

further reading for Montgomery in SW: [Koç et al. 96]

General Prime Fields $GF(p)$: Hardware

recall: $n = \lceil \log_2 p \rceil$

Idea: Compute n inner products in parallel

Best studied architecture: Montgomery multiplication

Input: A, B, where $A = \sum_{i=0}^{n+2} a_i 2^i$, $B = \sum_{i=0}^{n+1} b_i 2^i$
Output: $A \cdot B \bmod N$

1. $R_0 = 0$
2. for $i = 0$ to $n + 2$ do
3. $q_i = R_i(0)$
4. $R_{i+1} = (R_i + a_i \cdot B + q_i \cdot N)/2 \qquad (\star)$

time complexity (radix 2): $\approx n$ clock cycles
time complexity (radix r): $\approx n/r$ clock cycles
area complexity: $k \cdot n$ gates, k constant

Rem: $(\star)$ is performance critical operation

General Prime Fields $GF(p)$: Hardware

Remarks

1. is $\mathcal{O}(n)$ times faster than software
2. modular reduction is reduced to addition of long numbers:
$$R_{i+1} = (R_i + a_i \cdot B + q_i \cdot N)/2$$
3. $\Rightarrow$ use systolic array or redundant representation to avoid long carry chains
4. further reading:
[Eldridge/Walter 93] for general HW,
[Blum 99] for FPGA

Mersenne Prime Fields $GF(2^n - 1)$

Idea: Reduce modular reduction to addition.

Central relation: $2^n \equiv 1 \bmod p$

Algorithm: let $A, B \in GF(2^n - 1)$

$$\begin{aligned} A \cdot B &= c_h 2^n + c_l \quad \text{where } c_h, c_l \leq 2^n - 1 \\ A \cdot B &\equiv c_h + c_l \bmod p \end{aligned}$$

Complexity: Modular reduction requires 1 add (as opposed to $(n/w)^2$ mult in the case of general primes).

Remarks:

- Modular mult complexity is $\approx (n/w)^2$ inner products
- Roughly twice as fast as mult with general prime.
- $GF(2^n - c)$, c small, was proposed for ECC in [Crandall 92]

Generalized Mersenne Prime Fields

see [NIST 99]

Idea: Generalize modulo reduction "trick" from $2^n - 1$ to primes

$$p = 2^{n_l w} \pm 2^{n_{l-1} w} \pm \cdots \pm 2^{n_1 w} \pm 1$$

where $n_l > n_{l-1} > \cdots > n_1 > 0$
and $w = 2^i$, often $i = 16, 32, 64$.

Let $A, B \in GF(p)$, and write $A \cdot B$ as:

$$A \cdot B = c_{2n_l - 1} 2^{(2n_l - 1)w} + c_{2n_l - 2} 2^{(2n_l - 2)w} + \cdots + c_1 2^w + c_0$$

Coefficients $c_i 2^{iw}$, $i > n_l$, can be reduced recursively:

$$2^{n_l w} \equiv \mp 2^{n_{l-1} w} \mp \cdots \mp 2^{n_1 w} \mp 1 \bmod p$$

For instance:

$$2^{(2n_l - 1)w} \equiv \mp 2^{(n_l + n_{l-1} - 1)w} \mp \cdots \mp 2^{(n_l + n_1 - 1)w} \mp 1 \bmod p$$

Gener. Mersenne Primes: Example

$p = 2^{192} - 2^{64} - 1 = 2^{3 \cdot 64} - 2^{64} - 1 \quad , \quad w = 64$

$$A \cdot B = c_5 2^{320} + c_4 2^{256} + c_3 2^{192} + c_2 2^{128} + c_1 2^{64} + c_0$$

Reduction equations:

$$\begin{array}{lclclclcll} 2^{320} & \equiv & 2^{192} & + & 2^{128} & & & & & \bmod p \\ 2^{256} & \equiv & & & 2^{128} & + & 2^{64} & & & \bmod p \\ 2^{192} & \equiv & & & & & 2^{64} & + & 1 & \bmod p \end{array}$$

$$\begin{aligned} A \cdot B &\equiv c_4 2^{256} + [c_5 + c_3] 2^{192} + [c_5 + c_2] 2^{128} + c_1 2^{64} \\ &\quad + c_0 \bmod p \\ A \cdot B &\equiv [c_5 + c_3] 2^{192} + [c_5 + c_4 + c_2] 2^{128} + [c_4 + c_1] 2^{64} \\ &\quad + c_0 \bmod p \\ A \cdot B &\equiv [c_5 + c_4 + c_2] 2^{128} + [c_5 + c_4 + c_3 + c_1] 2^{64} \\ &\quad + [c_5 + c_3 + c_0] \bmod p \end{aligned}$$

- Reduction requires no multiplication
- Modular mult complexity is $\approx (n/w)^2$ inner products
- Roughly twice as fast as mult with general primes
- Specific primes are recommended by NIST for ECC

Extension Fields $GF(2^m)$

- applicable to DL and ECC
- extremely well studied (compared to other characteristics) since 1960s due to applications in coding
- choice of char $= 2$ was traditionally driven by hardware implementations
- arithmetic is greatly influenced by choice of *basis*
- bases proposed for applications:
    1. standard (or polynomial) basis
    2. normal basis
    3. other (dual basis, triangular basis, ...)

here: focus on polynomial basis.

$GF(2^m)$ Multiplication in Hardware

- active research area, many proposed architectures
- classification according to time-area trade-off

arch. type	m	#clocks (time)	#gates (area)	Remarks
bit parallel	any	1	$\mathcal{O}(m^2)$	often "too big"
digit serial	any	m/D	$\mathcal{O}(mD)$	$D < m$
hybrid	$D \vert m$	m/D	$\mathcal{O}(mD)$	$D < m$
bit serial	any	m	$\mathcal{O}(m)$	classical arch.
super serial	any	ms	$\mathcal{O}(m/s)$	new, mainly for FPGA [O/P 99]

main relevance for cryptography: bit serial, digit serial, and hybrid multipliers

Bit Serial Multiplication

Standard basis GF multiplication:

$$\begin{aligned} A \cdot B &= (a_{m-1}x^{m-1} + \cdots a_1 x + a_0) \\ &\quad (b_{m-1}x^{m-1} + \cdots b_1 x + b_0) \text{ mod } P(x) \end{aligned}$$

where $a_i, b_i \in GF(2)$.

Often: $P(x)$ is trinomial or pentanomial

Two traditional architectures

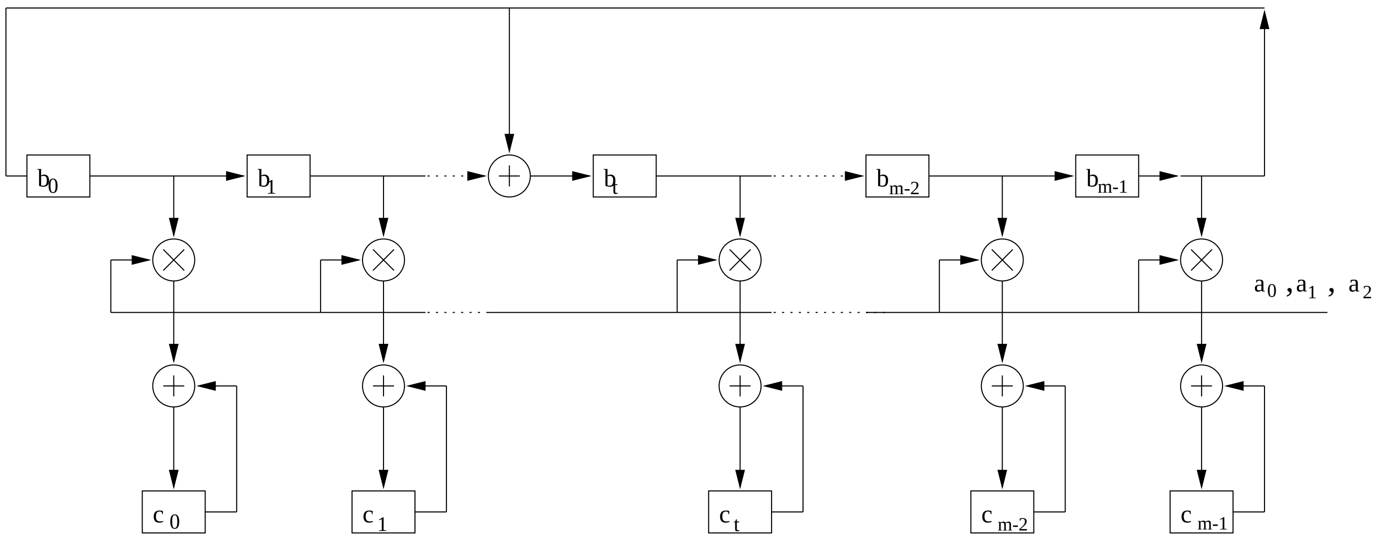
- least significant bit-first (LSB) multiplier
- most significant bit-first (MSB) multiplier

(see, e.g., [Beth/Gollmann 89])

Least Significant Bit-First Architecture

$$\begin{aligned} A \cdot B &= a_0 B(x) \\ &+ a_1[xB(x) \bmod P(x)] \\ &+ \cdots \\ &+ a_{m-1}[x(x^{m-2}B(x)) \bmod P(x)] \end{aligned}$$

Architecture if $P(x)$ is trinomial:



In every clock cycle compute:

1. mult by x and mod red.: $x \times (x^{i-1}B(x)) \bmod P(x)$
2. scalar mult by a_i and add: $+\, a_i \times [x^i B(x)]$

time complexity: m clock cycles
area complexity: $c\,m$ gates, c small

Hybrid Multipliers

- work for composite fields $GF((2^n)^m)$ (see [P/S 97])
- $\Rightarrow$ total extension degree (nm) can't be prime
- trades space for speed (faster but larger than LSB)
- least significant and most significant architectures are possible
- architectures analogous to bit serial mult (LSB, MSB)
- fundamental idea: process n subfield bits in parallel

Recall: Element representation in binary fields $A \in GF(2^{nm})$

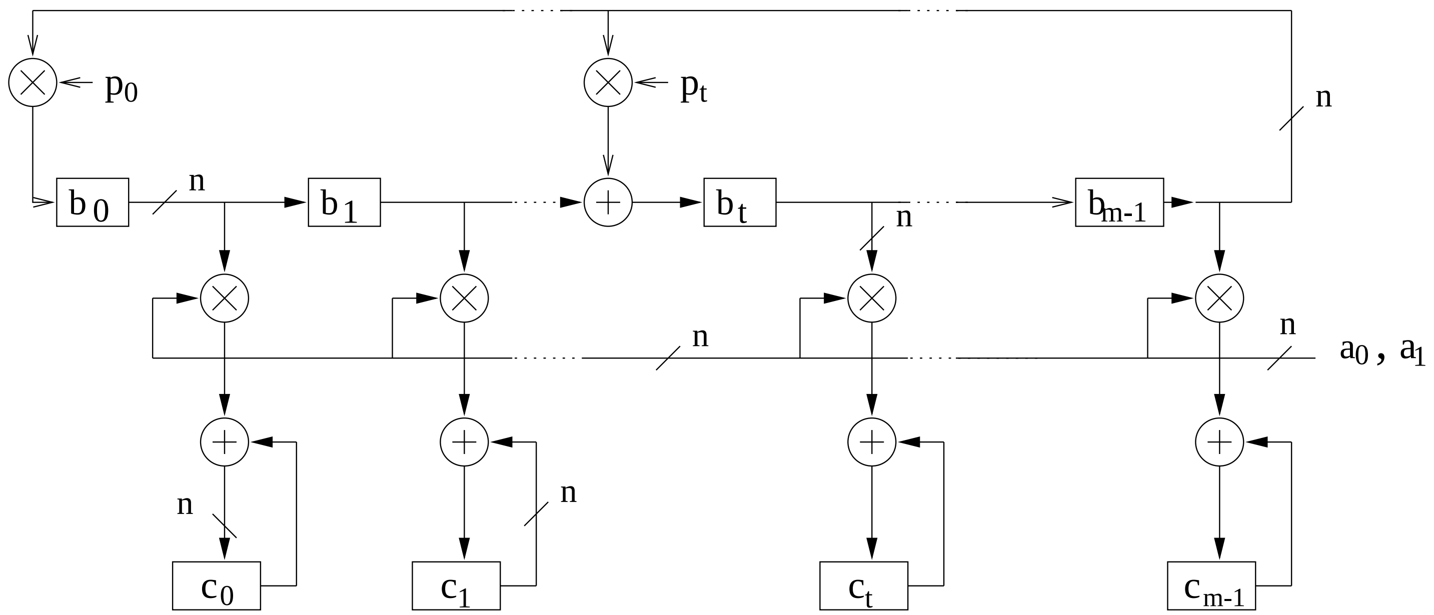
$$A(x) = a_{nm-1}x^{nm-1} + \cdots + a_1 x + a_0 \;\; , \;\; a_i \in GF(2)$$

Element representation in composite fields $A \in GF((2^n)^m)$

$$A(x) = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0 \;\; , \;\; a_i \in GF(2^n)$$

$$
\begin{aligned}
A \cdot B &= a_0 B(x) \\
&+ a_1[xB(x) \bmod P(x)] \\
&+ \cdots \\
&+ a_{m-1}[x(x^{m-2}B(x)) \bmod P(x)]
\end{aligned}
$$

Architecture if $P(x)$ is trinomial:



- gate costs occur in $GF(2^n)$ bit parallel multipliers
- **area compl.**: $\approx m\,n^2$ AND + $\approx m\,n^2$ XOR
- **time compl.**: $m \Rightarrow n$ times faster than LSB

Digit Multipliers

- relatively new [Song/Parhi 96]
- trades space for speed (faster but larger than LSB)
- time and area complexity similar to hybrid multipliers
- works for any m
- LSD and MSD are possible
- fundamental idea: Process $D > 1$ bit at a time.

Least Significant Digit Architecture

1. Step: Break $A(x)$ down into s digit polynomials, where $s = \lceil m/D \rceil$.

$$\begin{aligned} A(x) &= a_{m-1}x^{m-1} + \cdots + a_1 + a_0 \ , \ \ a_i \in GF(2) \\ A(x) &= \tilde{a}_{s-1}(x)\, x^{(s-1)D} + \cdots + \tilde{a}_1(x)\, x^D + \tilde{a}_0(x) \end{aligned}$$

where

$$\tilde{a}_i(x) = a_{i,D-1}x^{D-1} + \cdots + a_{i,1}x + a_{i,0} \ , \ \ a_{i,j} \in GF(2)$$

2. Step: Digit wise multiplication

$$\begin{aligned} AB &= \tilde{a}_0(x)B(x) \bmod P(x) \\ &+ \tilde{a}_1(x)[(x^D B(x)) \bmod P(x)] \bmod P(x) \\ &+ \tilde{a}_2(x)[x^D(x^D B(x)) \bmod P(x)] \bmod P(x) + \cdots \\ &+ \tilde{a}_{s-1}(x)[x^D(x^{D(s-2)} B(x)) \bmod P(x)] \bmod P(x) \end{aligned}$$

Operations per clock cycle:

1. multiplication by x^D and modular reduction:

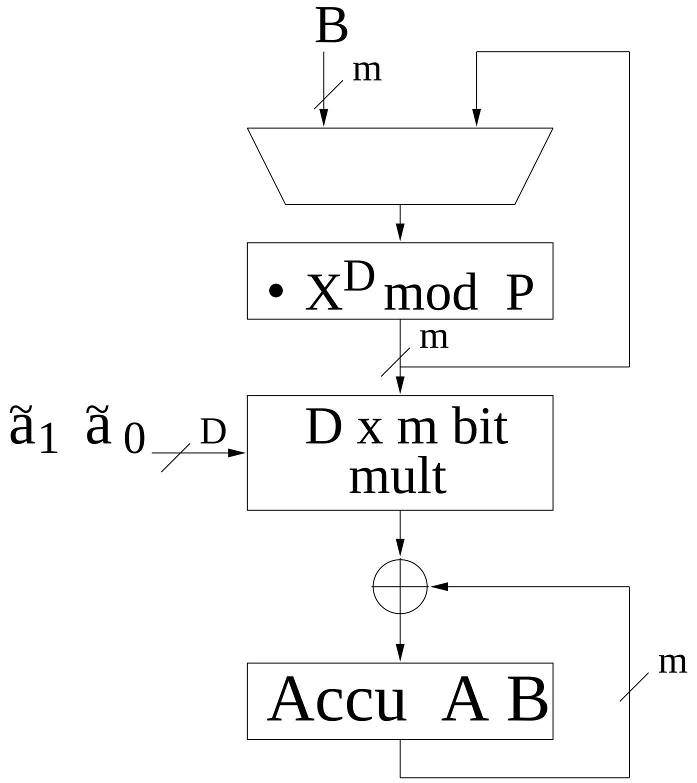
$$x^D \cdot [x^{(i-1)D} B(x) \bmod P(x)]$$

2. bit parallel multiplication of $D \times m$ bit polynomials:

$$\tilde{a}_i(x) \cdot [x^{iD} B(x) \bmod P(x)]$$

2. Step:

$$\begin{aligned} AB &= \tilde{a}_0(x)B(x) \bmod P(x) \\ &+ \tilde{a}_1(x)[(x^D B(x)) \bmod P(x)] \bmod P(x) \\ &+ \cdots \\ &+ \tilde{a}_{s-1}(x)[x^D(x^{D(s-2)}B(x)) \bmod P] \bmod P \end{aligned}$$



- mult by x^D is mainly a bit permutation
- gate costs occur in $D \times m$ bit parallel mult
- **area compl.**: $\approx m\,D$ AND $+ \approx m\,D$ XOR
- **time compl.**: $m/D \Rightarrow D$ times faster than LSB

Optimal Extension Fields $GF(p^m)$

- relatively new (see [B/P 98])
- main applications in ECC
- small extension degrees of $m \approx 3 \ldots 8$ are common
- very fast arithmetic on 64 bit processors

Optimal Extension Fields $GF(p^m)$

Idea: Fully exploit the fast integer arithmetic available in modern microprocessors

Design Principles

1. Choose subfield $GF(p)$ to be close to the processor's word size
   $\rightarrow$ fast subfield multiplication

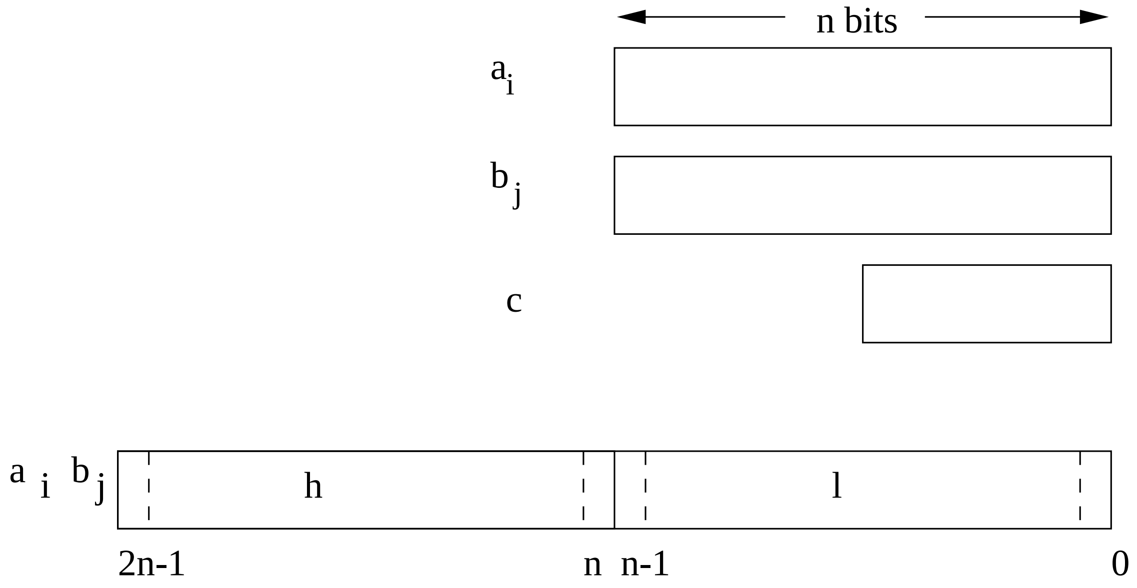
2. Choose subfield $GF(p)$ to be a pseudo-Mersenne prime, that is, p $= 2^n \pm c$, for "small" c
   $\rightarrow$ fast subfield modular reduction

3. Choose m so that an irreducible binomial $P(x) = x^m - \omega$ exists
   $\rightarrow$ fast extension field modular reduction

Subfield Multiplication: $a_i \cdot b_j \bmod p$

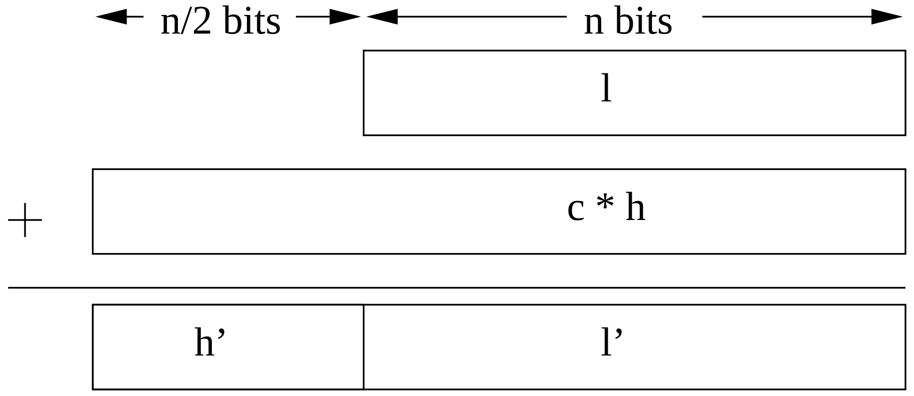
Note: Subfield mult is time critical operation

Important: $p = 2^n - c$, where $c \leq 2^{n/2}$.
$\Rightarrow 2^n \equiv c \pmod{(2^n - c)}$

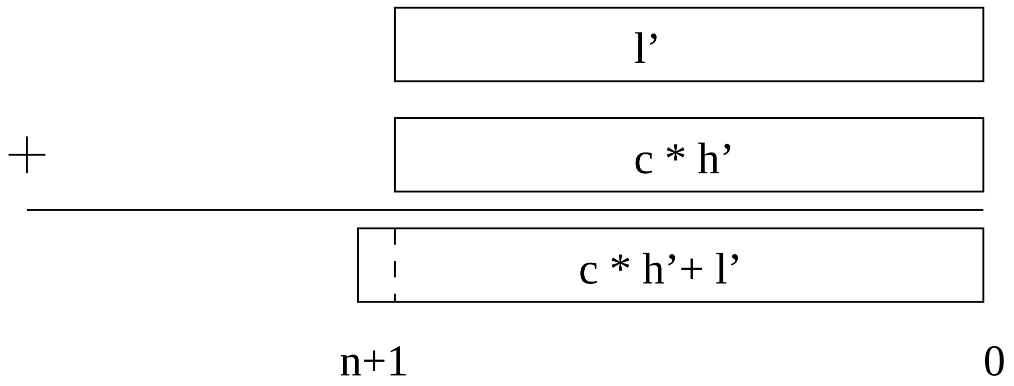


$$
\begin{aligned}
h, l &\leq 2^n - 1 \\
a_i\, b_j &= 2^n h + l \\
a_i\, b_j &\equiv ch + l \bmod p
\end{aligned}
$$

Subfield Multiplication: $a_i \cdot b_j \bmod p$



$$a_i b_j \equiv ch + l \bmod p = 2^n h' + l' \equiv ch' + l' \bmod p$$



Subfield mult complexity: 3 mults by c + adds, shifts
OEF mult complexity: $3(m^2 + m - 1)$ int mult (very low for small m)
Rem: Major speed-up if $c = 1$, i.e., p is Mersenne prime

Some Research Problems

- Fast Galois field arithmetic in software for general field polynomials?

- Hardware arithmetic architectures for some "new" field types, such as generalized Mersenne prime fields and OEFs?

- Other $GF(2^m)$ bases which lead to faster arithmetic?

- Thorough comparison of standard basis vs. normal basis vs. ..., especially in software?

- Faster inversion in $GF(p)$?

References

[1] D. Bailey and C. Paar. Optimal extension fields for fast arithmetic in public-key algorithms. In H. Krawczyk, editor, *Advances in Cryptography — CRYPTO '98*, volume LNCS 1462, pages 472–485. Springer-Verlag, 1998.

[2] T. Beth and D. Gollmann. Algorithm engineering for public key algorithms. *IEEE Journal on Selected Areas in Communications*, 7(4):458–466, 1989.

[3] T. Blum. Modular exponentiation on reconfigurable hardware. Master's thesis, ECE Dept., Worcester Polytechnic Institute, Worcester, USA, May 1999.

[4] R. Crandall. Method and apparatus for public key exchange in a cryptographic system. United States Patent, Patent Number 5159632, October 27 1992.

[5] S. E. Eldridge and C. D. Walter. Hardware implementation of Montgomery's modular multiplication algorithm. *IEEE Transactions on Computers*, 42(6):693–699, July 1993.

[6] Ç. Koç, T. Acar, and B. Kaliski. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16:26–33, 1996.

[7] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[8] D. Naccache and D. M'Raïhi. Cryptographic smart cards. *IEEE Micro*, 16:14–23, 1996.

[9] National Institute of Standard and Technology. Recommended elliptic curves for federal government use. available at http://csrc.nist.gov/encryption, May 1999.

[10] G. Orlando and C. Paar. A super-serial Galois fields multiplier for FPGAs and its application to public-key algorithms. In *Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM '99*, Napa Valley, USA, April 12–23 1997.

[11] C. Paar and P. Soria Rodriguez. Fast arithmetic architectures for public-key algorithms over Galois fields $GF((2^n)^m)$. In W. Fumy, editor, *Advances in Cryptography — EUROCRYPT '97*, volume LNCS 1233, pages 363–378. Springer-Verlag, 1997.

[12] L. Song and K. K. Parhi. Low energy digit-serial/parallel finite field multipliers. *Journal of VLSI Signal Processing*, 19(2):149–166, June 1998.