Hardware implementation of ECC

The University of Electro-Communications

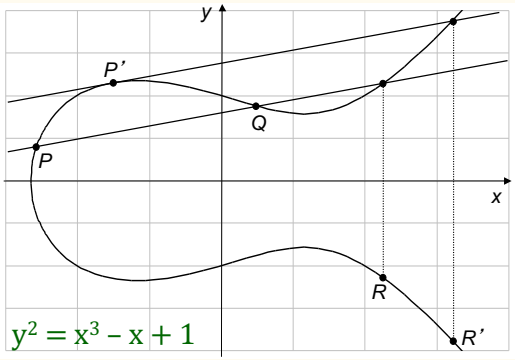
Kazuo SAKIYAMA

Introduction

- Public-key Cryptography (PKC)
  - The most famous PKC is RSA and ECC
  - Used for key agreement (Diffie-Hellman), digital signatures, public-key encryption
- Implementation of public-key cryptosystem
  - Slower encryption/decryption (~Mbps) than symmetric key cryptography (AES: ~Gbps)
  - RSA: modular exponentiation, $c^d \bmod n$
  - ECC: point multiplication, kP on elliptic curve
  - Expensive!!

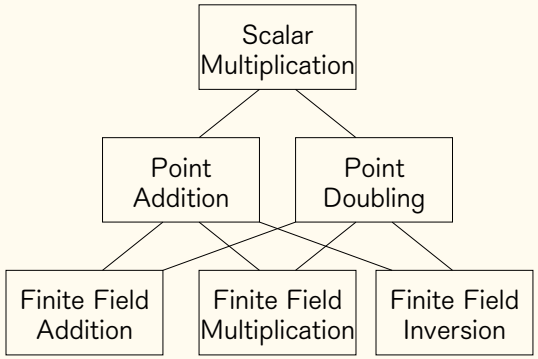
Point operations on elliptic curve

- Point Addition
  R = P + Q
- Point Doubling
  R' = 2P'
  Tangent line
- Scalar Multiplication
  kP
  Binary algorithm



Basic hierarchy for ECC

- Optimization is possible in multi-layer



Series of modular operations

Type of Curve?
GF(p), GF(2^m)?
Security Level?
Coordinate?
SW/HW?
FPGA/ASIC
Technology?
Size, Latency, Power?

(AB + C) mod p

Require: $P = (X_1, Y_1, Z_1)$, $Q = (X_2, Y_2, Z_2)$.
Ensure: $Q = Q + P$.

1: $t_1 = Z_1 Z_1$;
2: $t_2 = X_2 t_1$;
3: $t_3 = Z_2 Z_2$;
4: $t_4 = X_1 t_3 + t_2$;
5: $t_2 = X_1 t_3 - t_2$;
6: $t_5 = t_1 Z_1$;
7: $t_6 = Y_2 t_5$;
8: $t_1 = t_3 Z_2$;
9: $t_3 = t_1 Y_1 + Y_2$;
10: $Y_2 = t_1 Y_1 - Y_2$;
11: $t_5 = t_2 t_2$;
12: $t_1 = t_4 t_5$;
13: $X_2 = Y_2 Y_2 - t_1$;
14: $t_4 = -2X_2 + t_1$;
15: $t_1 = t_5 t_2$;
16: $t_1 = t_3 t_1$;
17: $t_3 = t_4 Y_2 - t_1$;
18: $Y_2 = t_3/2$;
19: $Z_2 = t_2 Z_2$;
20: $Z_1 = Z_1 Z_2$;
21: Return Q;

Require: $P = (X_1, Y_1, 1)$, $Q = (X_2, Y_2, Z_2)$.
Ensure: $Q = Q + P$.

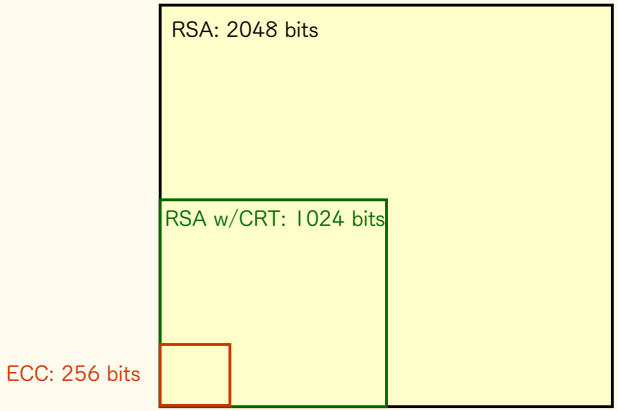
1: $t_3 = Z_2 Z_2$;
2: $t_4 = X_1 t_3 + X_2$;
3: $t_2 = X_1 t_3 - X_2$;
4: $t_1 = t_3 Z_2$;
5: $t_3 = t_1 Y_1 + Y_2$;
6: $Y_2 = t_1 Y_1 - Y_2$;
7: $t_5 = t_2 t_2$;
8: $t_1 = t_4 t_5$;
9: $X_2 = Y_2 Y_2 - t_1$;
10: $t_4 = -2X_2 + t_1$;
11: $t_1 = t_5 t_2$;
12: $t_1 = t_3 t_1$;
13: $t_3 = t_4 Y_2 - t_1$;
14: $Y_2 = t_3/2$;
15: $Z_2 = t_2 Z_2$;
16: Return Q;

Require: $Q = (X_2, Y_2, Z_2)$.
Ensure: $Q = 2Q$.

1: $t_1 = X_2 X_2$;
2: $t_1 = 3t_1$;
3: $t_2 = Z_2 Z_2$;
4: $t_2 = t_2 t_2$;
5: $t_2 = at_2 + t_1$;
6: $t_1 = 2Y_2$;
7: $Z_2 = Z_2 t_1$;
8: $t_3 = t_1 t_1$;
9: $t_4 = X_2 t_3$;
10: $X_2 = 2t_4$;
11: $X_2 = t_2 t_2 - X_2$;
12: $t_1 = t_1 t_3$;
13: $t_1 = t_1 Y_2$;
14: $t_3 = t_4 - X_2$;
15: $Y_2 = t_2 t_3 - t_1$;
16: Return Q;

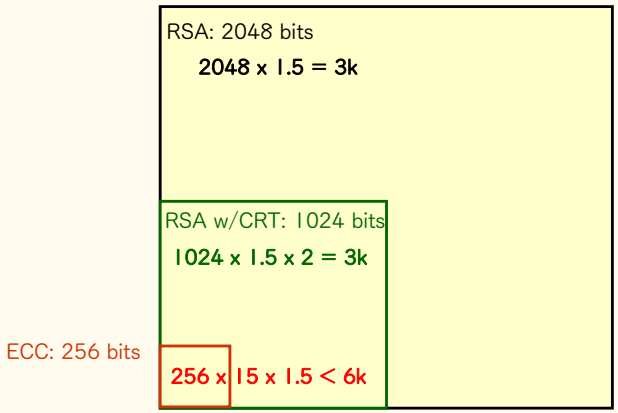
Data Size in Modular Operations

- Modular multiplication

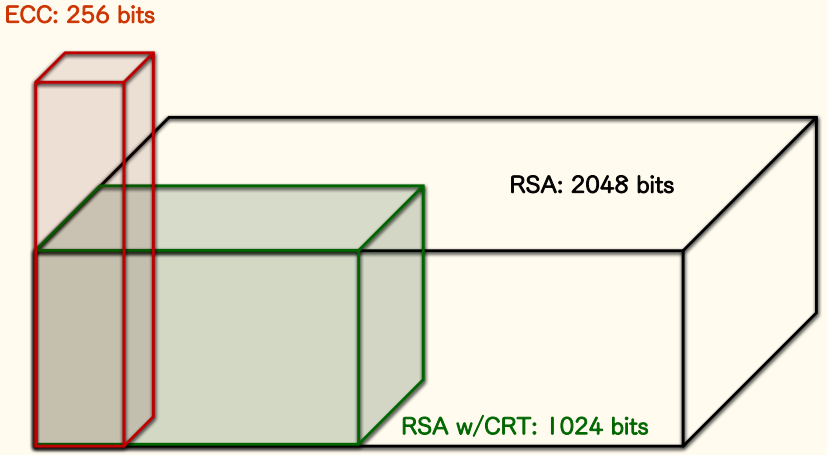


of Modular Multiplications

- RSA: c^d , ECC: kP



Total computational complexity



Right-to-left & Left-to-right

Left-to-right Binary Algorithm

Input: non-negative integers c, $d = (d_{t-1}d_{t-2}\cdots d_1d_0)_2$.
Output: $c^d \bmod n$.

```
1: T ← 1
2: for i from t − 1 down to 0 do
3:     T ← T^2 mod n
4:     if d_i = 1 then
5:         T ← cT mod n
6:     end if
7: end for
8: Return T
```

Dependency

Right-to-left Binary Algorithm

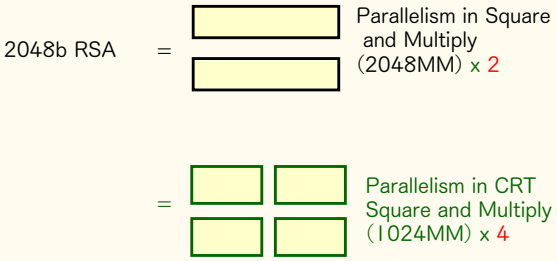
Input: non-negative integers c, $d = (d_{t-1}d_{t-2}\cdots d_1d_0)_2$.
Output: $c^d \bmod n$.

```
1: S ← 1, T ← c
2: for i from 0 to t − 1 do
3:     if d_i = 1 then
4:         S ← ST mod n
5:     end if
6:     T ← T^2 mod n
7: end for
8: Return S
```

No Dependency

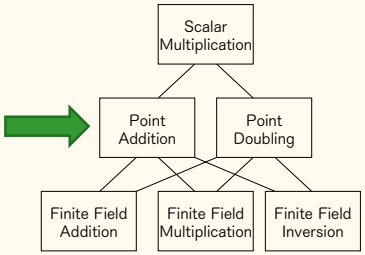
RSA: parallelism for high speed

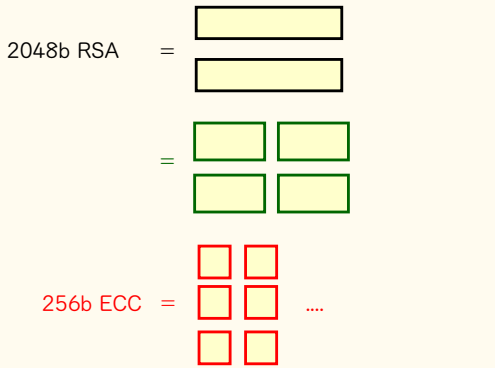
- CRT
- Binary algorithm



ECC: parallelism for high speed

- Point operations





ECC Scalar Multiplication

(256MM) x ??

Exercise 1

- Q: Latency of 2048b CRT-RSA?
  - Latency of 1024b MM = 100ns
- A1: 100ns x 1024 x 1.5 x 2 = 300us
  - when using one MM
- A2: 100ns x 1024 x 1.5 = 150us
  - when using two MMs
- Squaring = MM

Exercise 2

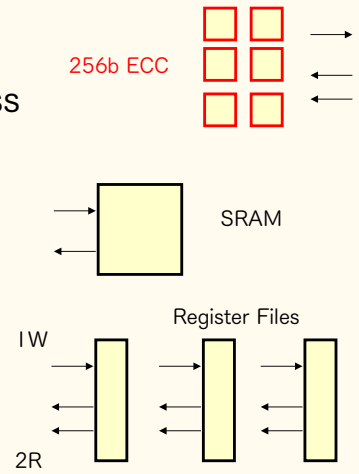
- Q: Latency of 256b ECC?
  - Latency of 256b MM = 5ns
  - Point addition needs 12 MM
  - Point doubling needs 10 MM
- A1: 5ns x 256 x 16 = 160us
  - when using one MM
- A2: 160/2 = 80us
  - when using two MMs & deg. parallelism = 2

Faster implementation

- Latency of 256 ECC can be less than 1us?
  - Short latency of 256b MM : e.g. 4x
  - More parallelism : e.g. 4x
  - Efficient point operation sequence : e.g. 4x
- Limitation of
  - current CMOS technology
  - current PKC

Yet Another Bottleneck

- Data coherence
  - Memory (register) access
    - SRAM
    - 1W2R register
- Scheduling
  - Dynamic/Static



Modular Multiplication

- Multiplication + Reduction
  - Barrett Reduction (1986): Focus on MSBs
  - Montgomery Reduction (1985): Focus on LSBs
- Based on n-bit multiplier (n = 8, 16, 32, 64)
  - E.g., CIOS [Koç, Acar, Kaliski Jr. 1996]
- Based on bit-serial multiplier
  - Multiplicand x n-bit [Großschädl 2001]

Final reduction for prime field p

- $XYR^{-1} \bmod p$
- $T = (T + x_i Y + mp)/2 < 2p$

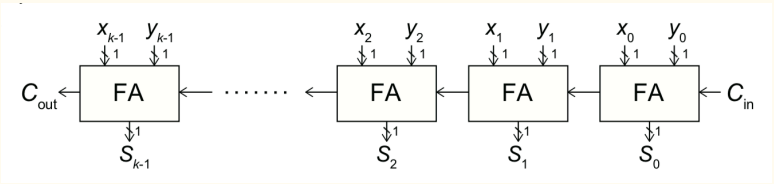
Input: k-bit integers X, Y with $0 \le X, Y < p$, $R = 2^k$.
Output: $XYR^{-1} \bmod p$.
1: $T = (t_k, \ldots t_0)_2 \leftarrow 0$
2: **for** i from 0 to $k-1$ **do**
3: $\quad m \leftarrow t_0 \oplus x_i \cdot y_0$
4: $\quad T \leftarrow (T + x_i Y + mp)/2$
5: **end for**
6: **if** $T \ge p$ **then**
7: $\quad T \leftarrow T - p$
8: **end if**
9: return T

Final Reduction
(Unnecessary fo Binary fields)

Final reduction: $T = T - p < p$

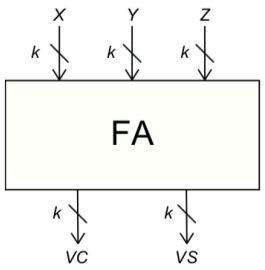
Ripple Carry Adder (RCA)

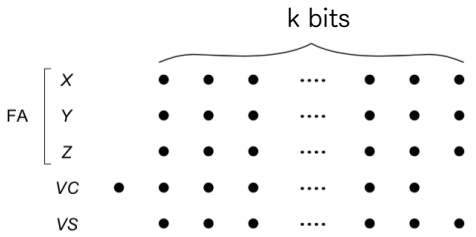
- Long carry chain
  - $s_i = x_i \oplus y_i \oplus z_i$
  - $c_{i+1} = MAJ(x_i, y_i, z_i)$
    $= (x_i \wedge y_i) \vee (y_i \wedge z_i) \vee (z_i \wedge x_i)$



Carry Save Adder (CSA)

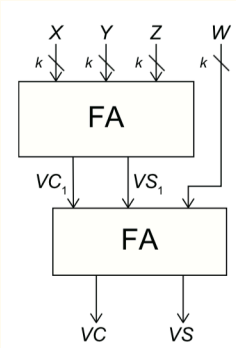
- 3-to-2 compression
  - $X + Y + Z = 2VC + VS$

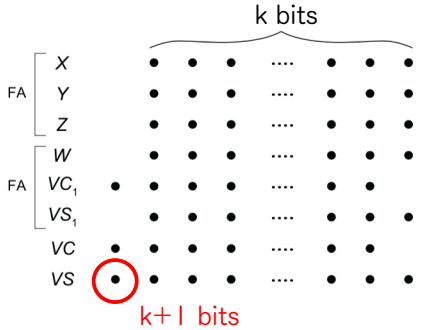




4-2 CSA

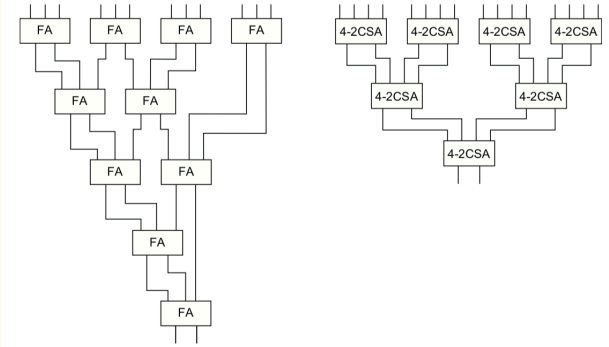
- Two FAs w/o carry chain
  - $X + Y + Z + W = 2VC + VS$





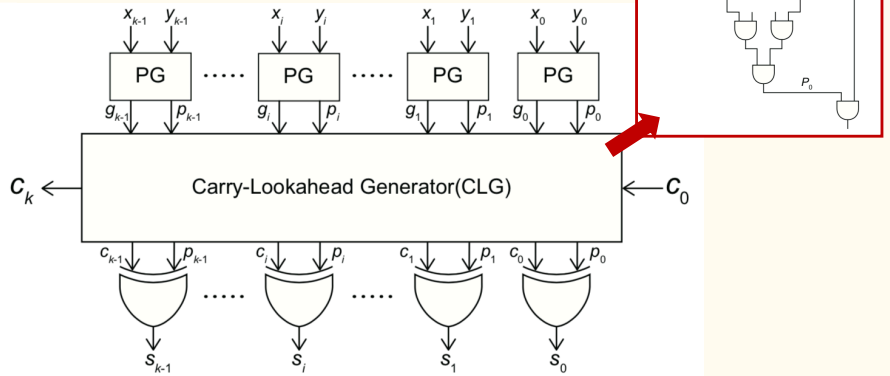
Adder Tree

- 16 operands
  - Latency ~ 5 or 6 FAs (RCA = 16FAs)



Carry Lookahead Adder (CLA)

- Generate and Propagate
  - $g_i = x_i \wedge y_i$, $p_i = x_i \oplus y_i$



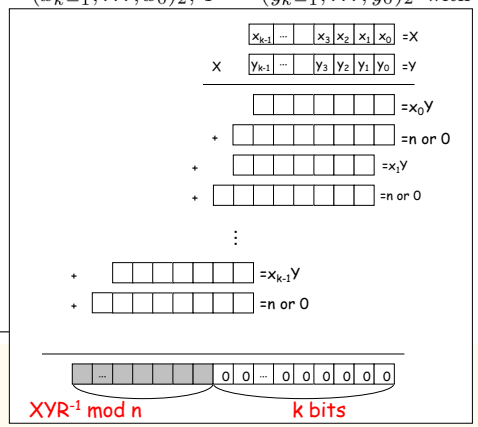
Bit-serial Montgomery MM

Bit-serial Montgomery Modular Multiplication

Input: integers $n = (n_{k-1}, \ldots, 1)_2$, $X = (x_{k-1}, \ldots, x_0)_2$, $Y = (y_{k-1}, \ldots, y_0)_2$ with $0 \leqq X, Y < n$.
Output: $XYR^{-1} \bmod n$.

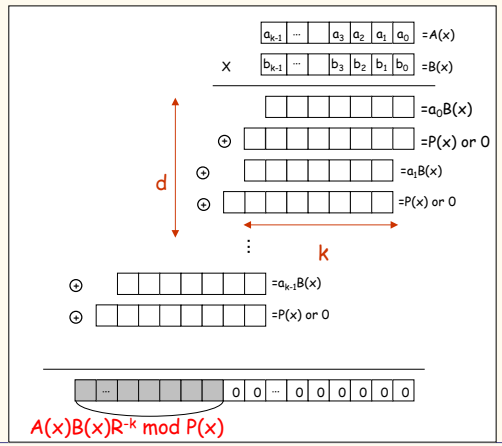
1: $T_0 \leftarrow 0$
2: **for** i from 0 to $k-1$ **do**
3: $\quad q_i \leftarrow t_0 \oplus (x_0 \wedge y_i)$
4: $\quad T \leftarrow (T + q_i n + y_i X)/2$
5: **end for**
6: **if** $T \geqq n$ **then**
7: $\quad T \leftarrow T - n$
8: **end if**
9: return T

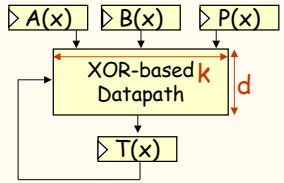
4-2 CSA!



Modular Reduction from LSB

- LSB-first bit-serial multiplier (binary fields)



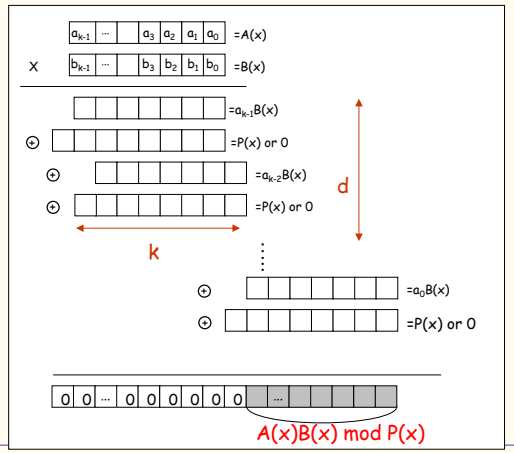


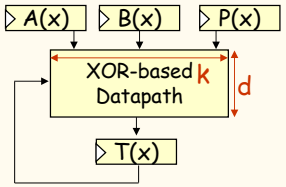
Montgomery Multiplier:
$A(x)B(x)R^{-1} \bmod P(x)$ $(R = x^k)$

- Delay is determined by **d**
- Loop : k/d

Modular Reduction from MSB

- MSB-first bit-serial multiplier (binary field)





A(x)B(x) mod P(x)

- Delay is determined by **d**
- Loop : k/d

Towards no final reduction

- $XYR^{-1} \bmod p$
- $T = (T + x_i Y + mp)/2 < 3p$

Input: $(k+1)$-bit integers, X, Y with $0 \le X, Y < 2p$, $R = 2^{k+2}$.
Output: $XYR^{-1} \bmod p$.
1: $T = (t_{k+1}, \ldots t_0)_2 \leftarrow 0$
2: **for** i from 0 to k **do** — Loop + 1
3: $\quad m \leftarrow t_0 \oplus x_i \cdot y_0$
4: $\quad T \leftarrow (T + x_i Y + mp)/2$
5: **end for**
6: $T \leftarrow (T + mp)/2$
7: **return** T

$T = (T + mp)/2 < 2p$

Final reduction for prime field p

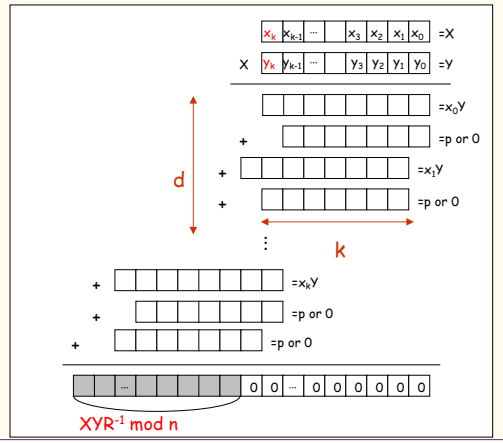
- $XYR^{-1} \bmod p$
- $T = (T + x_i Y + mp)/2 < 2p$

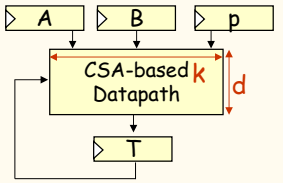
Input: k-bit integers X, Y with $0 \le X, Y < p$, $R = 2^k$.
Output: $XYR^{-1} \bmod p$.
1: $T = (t_k, \ldots t_0)_2 \leftarrow 0$
2: **for** i from 0 to $k-1$ **do**
3: $\quad m \leftarrow t_0 \oplus x_i \cdot y_0$
4: $\quad T \leftarrow (T + x_i Y + mp)/2$
5: **end for**
6: **if** $T \ge p$ **then**
7: $\quad T \leftarrow T - p$ — Final Reduction
8: **end if**
9: return T

Final reduction: $T = T - p < p$

Modular Reduction from LSB

- LSB-first bit-serial multiplier (prime field)



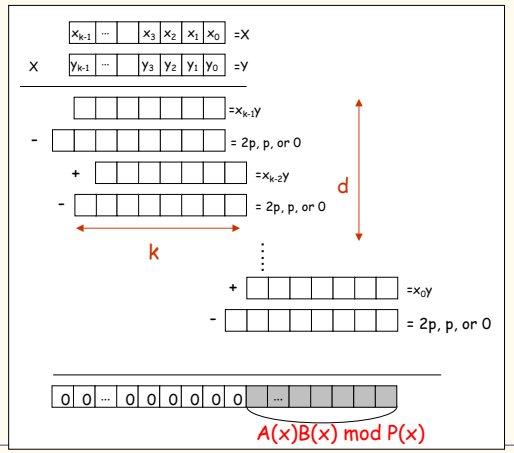


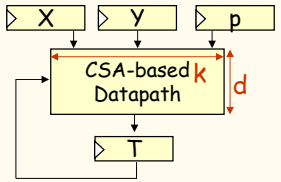
Montgomery Multiplier:
$XYR^{-1} \bmod p$ ($R = 2^{k+2}$)

- Delay is determined by **d**
- Loop : (k+2)/d

Modular Reduction from MSB

- MSB-first bit-serial multiplier (prime field)





X·Y mod p

-Delay is determined by **d**
-Loop : (k+2)/d

Division in MSB-first

- XY mod p
- Intermediate value, $T = (2T + x_i Y) < 3p$

Input: k-bit integers X, Y with $0 \leq X, Y < p$.
Output: $XY \bmod p$.
1: $T = (t_{k+1}, \ldots t_0)_2 \leftarrow 0$
2: **for** i from $k-1$ to 0 **do**
3: $\quad T \leftarrow (2T + x_i Y)$
4: $\quad q \leftarrow \lfloor T/p \rfloor$
5: $\quad T \leftarrow (T - qp)$
6: **end for**
7: return T

- Division is escaped with precomputation

[Knezevic, Vercauteren, Verbauwhede, 2010]

Barrett Reduction

- $T/p = T/2^{k-1} * 2^{k-1}/p$
- S4 : $q = T/2^{k-1}$ shift operation (wiring)
- S5 : $T = (T - q * 2^{k-1}/p)$ precomputation

Input: k-bit integers X, Y with $0 \leq X, Y < p$.
Output: $XY \bmod p$.
1: $T = (t_{k+1}, \ldots t_0)_2 \leftarrow 0$
2: **for** i from $k-1$ to 0 **do**
3: $\quad T \leftarrow (2T + x_i Y)$
4: $\quad q \leftarrow \lfloor T/p \rfloor$
5: $\quad T \leftarrow (T - qp)$
6: **end for**
7: return T

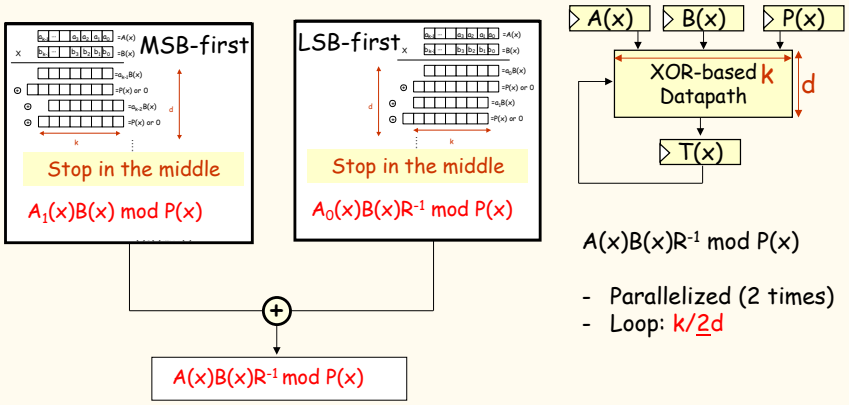
- But q = 0, 1, 2 ⋯

Parallelism in Modular Multiplication

- Both MSB/LSB Modular Multiplication
  [Batina et. al., 2004; Kaihara and Takagi, 2005]
- Combine MSB-first and LSB-first MM
- $A(x)B(x)R^{-1} \bmod P(x)$
  $= (A_1(x)R + A_0)B(x)R^{-1} \bmod P(x)$
  $= A_1(x)B(x) \bmod P(x)$ MSB-first
  $+ A_0(x)B(x)R^{-1} \bmod P(x)$ LSB-first
- $R = x^{k/2}$

Bipartite Modular Multiplication

- Bipartite bit-serial multiplier (binary field)



$A(x)B(x)R^{-1} \bmod P(x)$

- Parallelized (2 times)
- Loop: k/2d

Merits in Bipartite Modular Multiplication

- Parallelism in Modular Multiplication
- Applicable to prime field (original proposal)
  - Need to take care (final) reductions
- Almost double performance, less than double area (F/Fs are shared)

Fast Multiplier

- Booth recoding (radix-4)
  - Partial products $x_i Y$
  - Escape 3Y nor -3Y
  - Half # of operands (at worst)
  - CSA + CLA
- Also derived by 2X – X (=X)
  - Ex) $X = (00\ 1111\ 0011\ 1001)_2 = (10\bar{1}\ 10\bar{2}1)_4$

x2i+1	x2i	x2i-1	zi	
0	0	0	0	
0	0	1	1	
0	1	0	1	
0	1	1	2	
1	0	0	−2	$(=\bar{2})$
1	0	1	−1	$(=\bar{1})$
1	1	0	−1	$(=\bar{1})$
1	1	1	0	

2X =	0	0	1	1	1	1	0	0	1	1	1	1	0	0	1	0
− X =		0	0	1	1	1	1	0	0	1	1	1	1	0	0	1
Z =		1		0		$\bar{1}$		1		0		$\bar{2}$		1	

Multiplier-Based MMM

Multiplier-Based Montgomery Modular Multiplication

Input: integers $n = (n_{r-1}, \ldots, n_0)_{2^m}$, $X = (x_{r-1}, \ldots, x_0)_{2^m}$, $Y = (y_{r-1}, \ldots, y_0)_{2^m}$ with $m = \lceil k/r \rceil$, $0 \leq X, Y < n$, $\gcd(n_0, 2) = 1$.
Output: $XYR^{-1} \bmod n$.

$RR^{-1} - nn' = 1$, $R = 2^k$, $0 < R^{-1} < n$, $0 < n' < R$.

1: $T \leftarrow 0$
2: **for** i from 0 to $m-1$ **do**
3: $\quad q_i \leftarrow (T + x_0 y_i) n' \mod 2^k$
4: $\quad T \leftarrow (T + q_i n + y_i X)/2^k$
5: **end for**
6: **if** $T \leftarrow n$ **then**
7: $\quad T \leftarrow T - n$
8: **end if**
9: Return T

Multiplications

X = (f3 d7 2f 88), Y = (f6 c6 ba 98), n = (f9 81 89 2b)

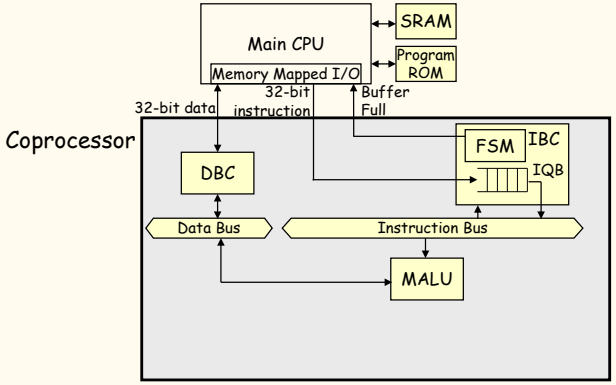
i	q_i	T	←	$T + q_i n + y_i X$
Initial	–	(00 00 00 00 00)		
0	c0	(01 3e 3a ba 01)	←	(01 3e 3a ba 01 00)
1	c5	(00 ee 8d 65 8a)	←	(00 ee 8d 65 8a 00)
2	6a	(01 37 7f 16 e7)	←	(01 37 7f 16 e7 00)
3	f3	(01 d8 4c 17 69)	←	(01 d8 4c 17 69 00)
–	–	(00 de ca 8e 3e)		–

Architecture for ECC coprocessor

- HW/SW co-design
  - CPU + coprocessor for ECC
- Speed performance depends on architecture
  - Trade-off between cost and performance
  - Local dedicated memory for high performance
- Instruction-set extension, ASIC, etc.

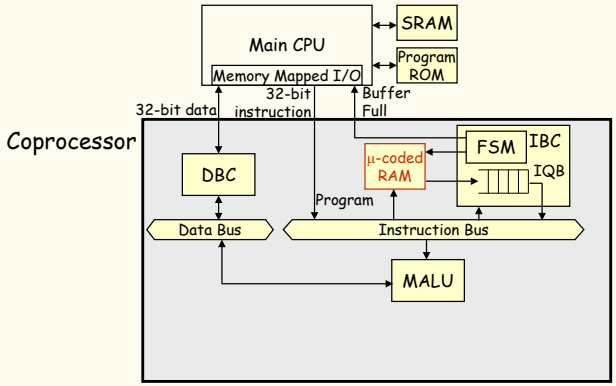
ECC coprocessor (I)

TYPE I: Smallest implementation (baseline)



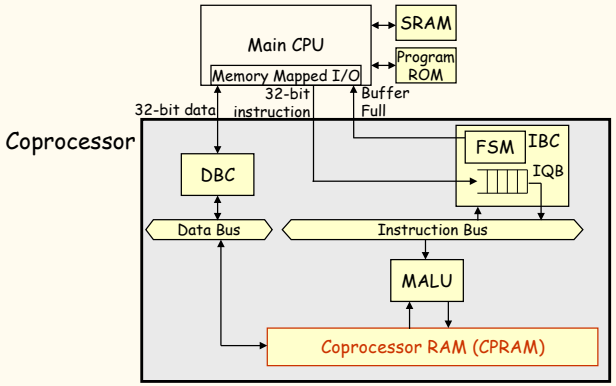
ECC coprocessor (II)

TYPE II: TYPE I + m-coded RAM



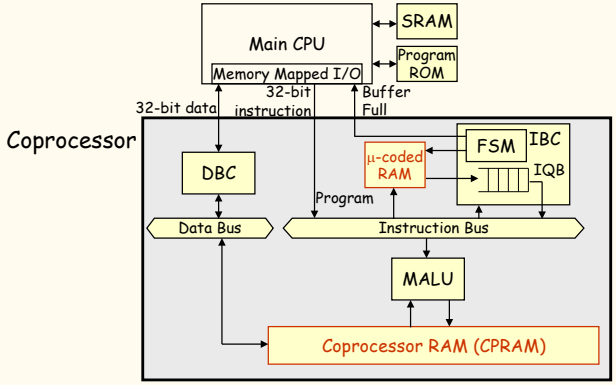
ECC coprocessor (III)

TYPE III: TYPE I + CPRAM



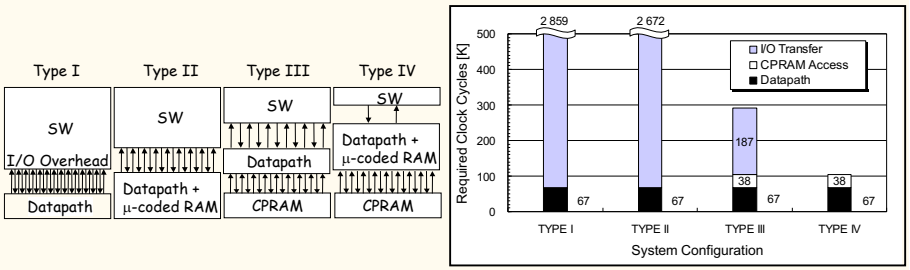
ECC coprocessor (IV)

TYPE IV: TYPE I + CPRAM & m-code RAM



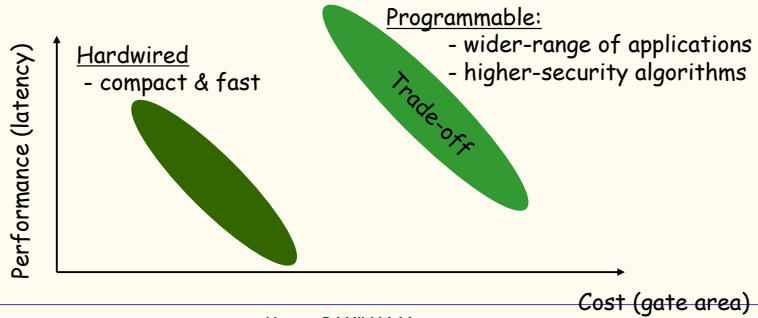
Profiling ECC coprocessor

- Performance for HECC over GF(2^{83})
  - The number of cycles is reduced by CPRAM
  - Micro-coded RAM can avoid instruction stalls



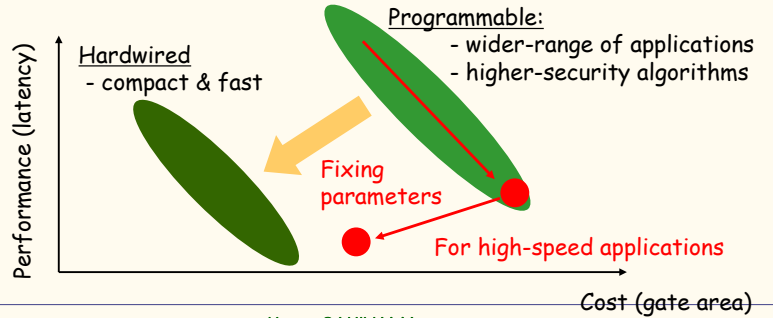
Cost, Performance and Security Trade-offs

- Programmable VS hardwired parameters
  - Curve parameters
  - Computational sequence: Coordinates, operation form.
  - Prime p, irreducible polynomial P(x)
  - Field sizes



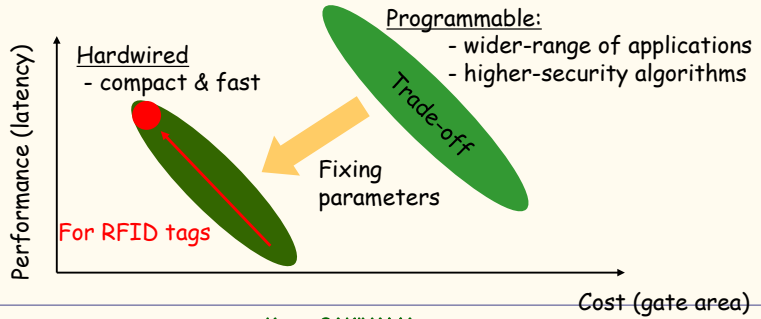
Cost, Performance and Security Trade-offs

- Programmable VS hardwired parameters
  - Curve parameters
  - Computational sequence: Coordinates, operation form.
  - Prime p, irreducible polynomial P(x)
  - Field sizes



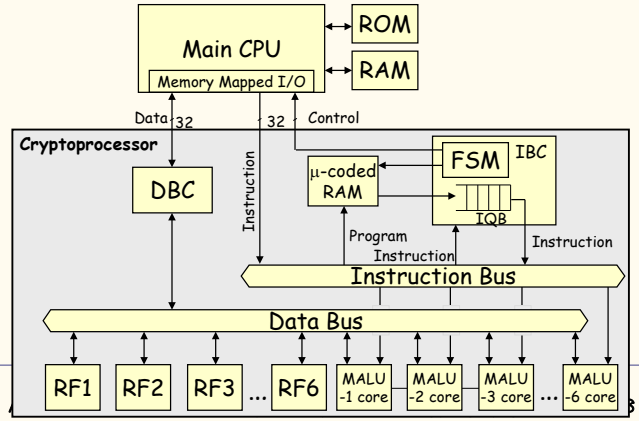
Cost, Performance and Security Trade-offs

- Programmable VS hardwired parameters
  - Curve parameters
  - Computational sequence: Coordinates, operation form.
  - Prime p, irreducible polynomial P(x)
  - Field sizes



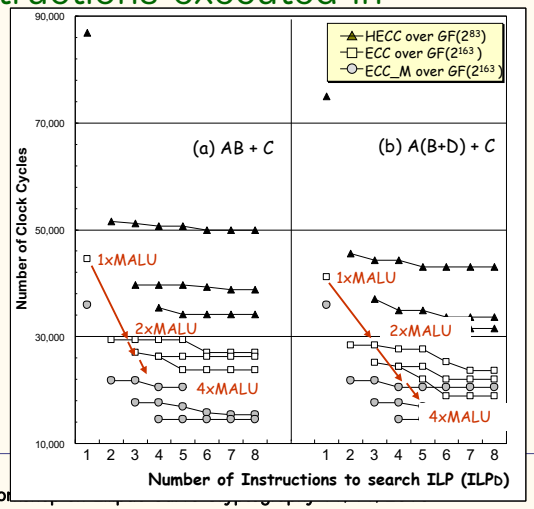
High-speed ECC & HECC

- Programmable high-speed coprocessor
  - ECC / HECC over GF(2^m)
  - Six MALU cores (MALU)



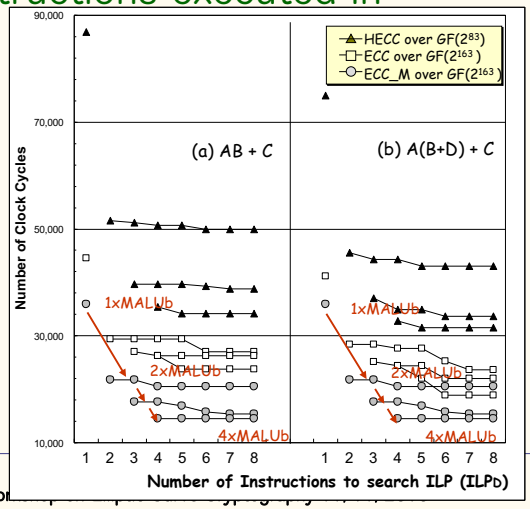
Parallelism in scalar multiplication

- How many instructions executed in parallel?
  - Operation form
    AB + C and
    A(B+C) + D
  - Check the data dependencies for ECC (and HECC)



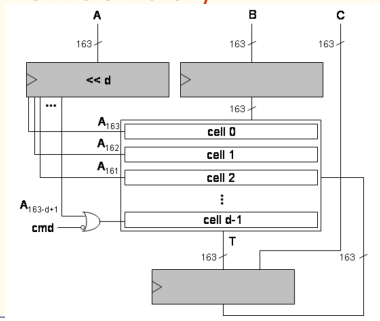
Parallelism in scalar multiplication

- How many instructions executed in parallel?
  - Operation form
    AB + C and
    A(B+C) + D
  - Check the data dependencies for ECC (and HECC)



Lightweight ECC For RFID

- Low-power MALU for RFID tags
  - Based on MALU
  - Cost and performance for ECC and HECC
  - Power estimation with 0.13-um CMOS library
- Resource sharing
  - Modular addition and multiplication use the same hardware (cell)



Conclusions

- Trade-offs in ECC hardware
  - Parallel processing
  - Operation forms: AB + C or A(B+D) + C
  - Cost, performance, security