6^{th} International Conference on Public Key Infrastructure and its Applications (PKIA 2025)

Energy-Efficient Modular Exponential Techniques for Public-Key Cryptography

Presented By
Dr. Satyanarayana Vollala
Head & Associate Professor

Department of Computer Science and Engineering
International Institute of Information Technology, Naya Raipur

Outline of Presentation

Introduction

Energy Efficient ME Techniques

CCT

Introduction

- **Cryptography**
  - Indispensable tool to prevent unauthorized access to data
  - Essential part of human life for protecting sensitive data
- **Types of Cryptosystems**
  - Private-Key Cryptography (Symmetric Key Cryptography)
  - Public-Key Cryptography (Asymmetric key Cryptography)
- **Public-Key Cryptography (PKC)**
  - Authentication, confidentiality, data integrity & non-repudiation
  - Effective solution to the key operations
  - Minimizing the secure channel to exchange key information
- **Most popular PKC**
  - Diffie-Hellman Key Exchange Algorithm
  - RSA Public-key cryptography
  - ElGamal Public-key Cryptography
  - Rabin Public-key Cryptography
  - Elliptic Curve Cryptography

Public-Key Cryptography

- **Diffie-Hellman Key Exchange Algorithm**
  - Key generation at User A : $K = (Y_B)^{X_A} mod\ q$
  - Key generation at User B : $K = (Y_A)^{X_B} mod\ q$
- **Rivest Shamir Adleman (RSA)**
  - Encryption : $C = M^E mod\ N$
  - Decryption : $M = C^D mod\ N$
- **ElGamal Public-key Cryptography**
  - Public-Key : $\{p, \alpha, \alpha^a mod\ p\}$
  - Encryption : C=(γ, δ), $\gamma = \alpha^k mod\ p$, $\delta = m.(\alpha^a)^k mod\ p$
  - Decryption : $\gamma^{p-1-a} mod\ p$
- **Rabin Public-key Cryptography**
  - Encryption : $C = M^2 mod\ N$
  - Decryption : $M = \sqrt{C} mod\ N$

Public-Key Cryptography ... Contd.

- Modular Exponentiation is the crucial Operation for every PKC
- Modular Exponentiation is composed of repeated modular multiplications
- Hence, the performance of PKC $\Leftarrow$ efficiency of ME and MM
- Modular multiplication is the time consuming process
- Montgomery multiplication method avoids trial divisions
- It substitutes the trial divisions with shift operations

Multi-Core Architectures

- Performance of a system
  - Power consumption
  - Heat dissipation
  - Clock rate
  - The number of active cores
- Uni-core system
  - One Encryption/Decryption at a time
  - Performance Uni-core
- Multi-core system
  - Organized with two or more independent cores
  - Identical core (Homogeneous system)
  - Dissimilar cores (heterogeneous system)

Multi-Core Architectures ... Contd.

- Multi-core system ... Contd.
  - The performance can described by Amdhals law
  - Runs at low frequency but with better performance
  - Work Scheduled on different cores
  - Multiple requests received for Encryption/Decryption
  - Throughput $\propto$ Number of active cores
- Scheduler
  - Designing of a scheduler in the multi-core environment is a challenging task
  - Hardware scheduler is better than software scheduler

The major phases in PKC include ME as a key operation

- The time taken to evaluate ME is influenced by:
  - The no. of MMS
  - The time consumed by each operation
- Minimizing the time taken to perform the above key operations will give a great impact

Optimization the key arithmetic operations in terms of energy and throughput

Exploiting multi-core potential to cryptographic transformations is a real issue to be addressed

Modular Exponentiation

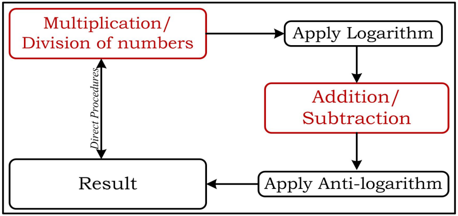
- The Central Tool of PKC
  - It is composed of repetition of sequence of modular multiplications
- Methods for ME
  - Right-to-left binary modular exponentiation
    - *Exponent scanned from LSB to MSB*
  - Left-to-right binary modular exponentiation (SM)
    - *Exponent scanned from MSB to LSB*
  - Left-to-right k-ary modular exponentiation
    - *Exponent scanned from LSB to MSB*
  - Sliding-window exponentiation
    - *Exponent scanned from LSB to MSB*
- The Most Used method
  - Left-to-Right binary exponential method

Montgomery Multiplication

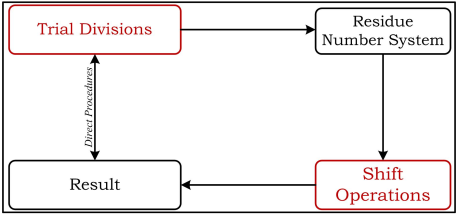
- Modular Multiplication
  - It involves trial divisions
    - Which are considerably time consuming operations
  - Direct Hardware implementation of MM is not possible
  - The traditional way is sequence of subtractions
- Hardware Implementations
  - Crypto Techniques can be implemented in H/w & S/w
  - Very hard to implement in hardware because of MMs
  - Hardware will be the ultimate choice
- Montgomery Method
  - Trial divisions are replaced by add/sub & shift operations
  - The basic Idea is like Logarithm in Maths

Montgomery Multiplication ... Contd.

- Conventional method of finding multiplication/division



- Montgomery method of finding modular multiplication



- Montgomery Value
  - For an integer A, the Montgomery value of A is $A^1 = AR^{-1} mod\ N$, where $R = 2^n mod\ N$
  - The Montgomery value of two integers A, B is $A.B.R^{-1} mod\ N$, where $R = 2^n mod\ N$

- Montgomery method to calculate MM value of two integers :
  - Algorithm Montgomery(P,Q,N)

    1: $P = A.B$;
    2: $Q = P.N^1 mod\ R$
    3: $Z = (P + Q.N)/R$
    4: **if** $(Z \geq N)$ **then**
    5: $\quad Z = Z - N$;
    6: **end if**
    7: $Return\ Z$

Table: Existing hardware designs for modular multiplications

Sl.No.	Authors and Year	Work
1	Shiann-Rong Kuang et al.[1], 2013	This architecture is capable of by passing the superfluous carry-save addition and register write operations
2	Shiann-Rong Kuang et al.[2], 2014	A simple and high-performance Montgomery multiplier, where multiplier uses only one level carry-save adder
3	Miyamoto, Atsushi and Homma et al.[3], 2011	A systematic design of RSA processor with the help of high-radix Montgomery multipliers
4	Huang, Miaoqing et al.[4], 2011	an optimized hardware design for MWR2MM and MWR4MM algorithms to minimize the # CC for computing n-bit MM
5	Sutter, Gustavo D et al.[5], 2011	Framed an architecture by using digital serial method and used carry-skip addition to convert the intermediate product
6	Yao, Gavin Xiaoxu et al.[6], 2014	Presented RNS parameter selection process for computational efficiency
7	Batina, Lejla et al.[7], 2001	A detailed survey of HA of diferent MM
8	Shieh, Ming-Der et al.[8], 2008	Have avoided the data dependency in multiplication process for conventional Montgomery Multiplication algorithm
9	Shieh, Ming-Der et al.[9], 2009	Introduced a new modular exponentiation hardware design with unified multiplication
10	Montgomery, Peter L [10], 1985	Introduced new technique to avoid trial divisions.
11	Montgomery, Peter L[11], 1994	A survey of modern integer factorization algorithms
12	McIvor, Ciaran [12], 2004	Introduced two new versions of Montgomery multiplication algorithms for evaluating RSA exponentiation using 4 to 2 CSA instead of 5 to 2 CSA

Sl.No.	Authors and Year	Work
13	N. Nedjah et al.[17], 2013	A massively parallel scheme aiming at performing all IMMs concurrently
14	Néto, João Carlos et al.[18], 2014	A way to speed up the Montgomery Multiplication by distributing the multiplier operand bits into partitions
15	Xiaofeng Chen et al.[19], 2014	a new secure outsourcing algorithm for (V-E, V-B) exponentiation modulo a prime in the two untrusted program model
16	Dimitrios Schinianakis et al.[20], 2014	A design methodology for incorporating RNS and Polynomial RNS in GF Montgomery modular multiplication in or respectively, as well as a VLSI architecture of a dual-field residue arithmetic Montgomery multiplier
17	Abdalhossein Rezai et al.[21], 2015	A new and efficient Montgomery modular multiplication architecture based on a new digit serial computation (Multibit-Scan–Multibit-Shift Technique).
18	Masahiro Kaminaga et al.[22], 2015	A new fault attack, double counting attack (DCA), on the precomputation of 2^t-ary ME for a classical RSA digital signature is proposed
19	Xinming Huang et al.[23], 2015	a novel and efficient design for RSA cryptosystem with a very large key size. A new modular multiplier architecture is proposed by combining the fast Fourier transform-based Strassen multiplication algorithm and Montgomery reduction,which is different from the interleaved version of Montgomery multiplications used in traditional RSA designs.
20	Hari Krishna Garg et al.[24], 2016	A new computational techniques for RNSs-based Barrett algorithm
21	Mehdi Tibouchi et al.[25], 2016	Improves upon Farashahi et al.′s [26] character sum estimates for point arithmetic
22	Joppe W. Bos [27], 2015	Analysis of point arithmetic for PKC such as ECC.

[28, 29, 30]

Energy Efficient Modular Exponential Algorithms based on Bit Forwarding Techniques

Algorithms for improving the efficiency of PKC

- Bit Forwarding 1-bit Algorithm (BFW1)
- Bit Forwarding 2-bits Algorithm (BFW2)
- Bit Forwarding 3-bits Algorithm (BFW3)
- Adoptable Montgomery Method (AMM)
- Methods to evaluate $A mod\ N$ and $(A.B) mod\ N$

Bit Forwarding Techniques

Bit Forwarding Techniques

- Scan the digits of Exponent from left-to-right
- For every bit of Exponent, square the result
- If there are c consecutive ones in the exponent, forward $c-1$ number of bits
- Then multiply the result with $M_{2^c-1} = M^{2^c-1}\ mod\ N$
- Pre-compute the values of $M_{2^c-1}\ for\ c = 1, 2, 3, \ldots$

Adoptable Montgomery Multiplication

- For computing MM involved in ME, Montgomery method is tuned according to the needs of BFTs, and named as AMM
- It is adaptable in the sense, that it can be used for any bit forwarding k-bit algorithm
- It can also used to compute $M^Z mod\ N$ for all $+$ve integers $Z \leq E$

Theorem

Theorem (Multiplication property of modular arithmetic)

$$(\alpha.\beta)\ mod\ \gamma = (\alpha\ mod\ \gamma\ .\ \beta\ mod\ \gamma)\ mod\ \gamma$$

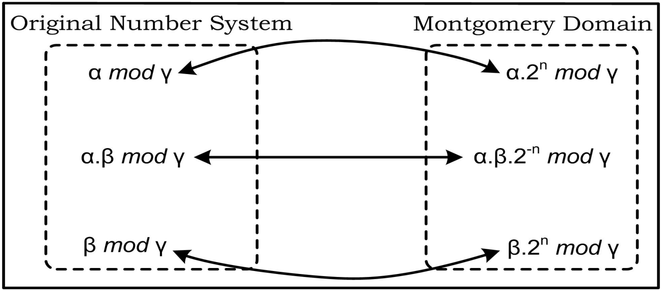
Algorithm AMM(A, B, N)

Require: A, B, N

Ensure: $R = A.B.2^{-n} \ mod \ N$

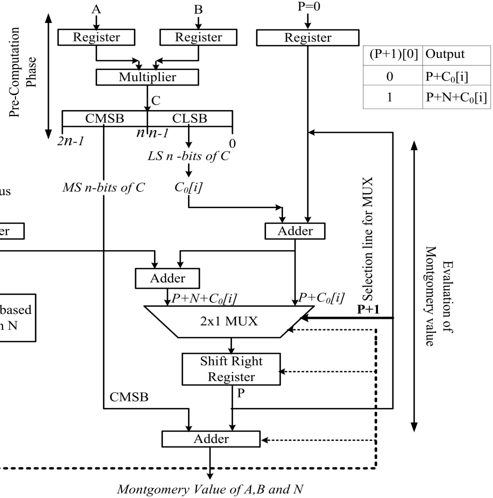
```
    Phase-I : Pre-computation
C = A.B;
C_1 = C[2n − 1, n];
C_0 = C[n − 1, 0];
P = 0;
    Phase-II : Evaluation of Montgomery Value
for i = 0 to n − 1 do
    if ((P + 1)[0] ≠ 0) then
        P = (P + N + C_0[i]) >> 1;
    else
        P = (P + C_0[i]) >> 1;
    end if
end for
R = P + C_1;
return R;
```

Mapping between general number system and Montgomery Domain



Let $2^n = R$, then the basic idea here is :
Multiplication modulo N ⇔ A division by R and a reduction modulo R
R is an exact power of 2, so multiplications and divisions are replaced by addition, subtractions and shift operations

Architecture Diagram of AMM



(P+1)[0]	Output
0	$P+C_0[i]$
1	$P+N+C_0[i]$

Modified Square and Multiply Method

Algorithm MSM(M, E, N)

Require: M, E, N and $PC(proposed\ constant) = 2^{2n} mod\ N$
Ensure: $R = M^E\ mod\ N$

$M_1 = AMM(M, PC, N)$; //Pre-Processing the message
$R[k-1] = M_1$;
for $i = K - 2$ Down to 0 **do**
  $R[i] = AMM(R[i+1], R[i+1], N)$;
  if $(e_i \neq 0)$ **then**
    $R[i] = AMM(R[i], M_1, N)$;
  end if
end for
$R = AMM(R[0], 1, N)$; //Post-Processing the message
return R

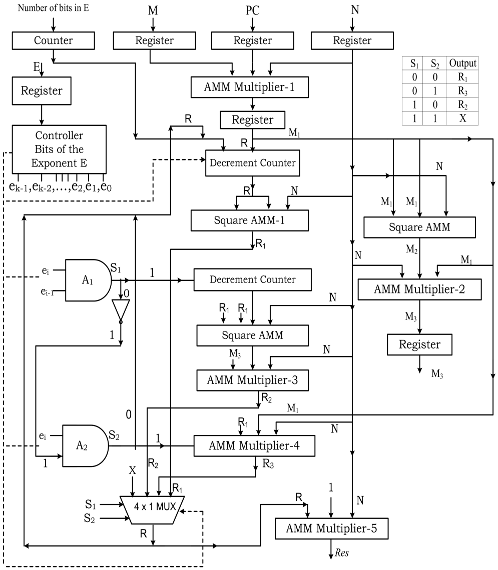
Bit Forwarding 1-bit Algorithm

Algorithm BFW1(M, E, N)

Require: M, E, N and $PC(proposed\ \ constant)$=$2^{2n} mod\ \ N$

Ensure: $R = M^E\ \ mod\ \ N$

$M_1 = AMM(M, PC, N)$; //Pre-Processing the message
$M_2 = AMM(M_1, M_1, N)$;
$M_3 = AMM(M_2, M_1, N)$;
$R[k-1] = M_1$;
for $i = k-2$ Down to 0 **do**
    $R[i] = AMM(R[i+1], R[i+1], N)$;
    if $((e_i \neq 0) \&\& (e_{i-1} \neq 0))$ **then**
        $i = i-1$; //Forwarding 1-bit
        $R[i] = AMM(R[i+1], R[i+1], N)$;
        $R[i] = AMM(R[i], M_3, N)$;
    else if $(e_i \neq 0)$ **then**
        $R[i] = AMM(R[i], M_1, N)$;
    end if
end for

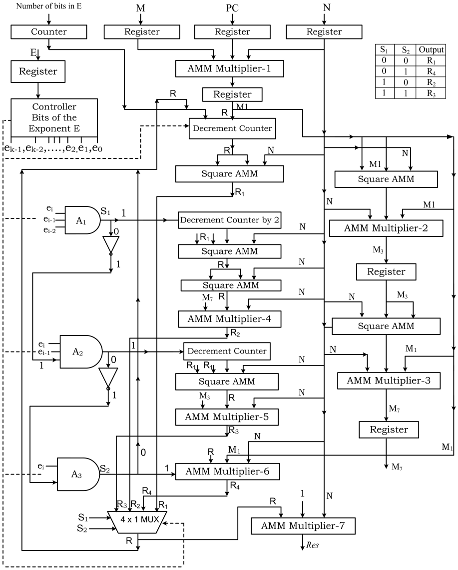


S_1	S_2	Output
0	0	R_1
0	1	R_3
1	0	R_2
1	1	X

Bit Forwarding 2-bits Algorithm

Algorithm BFW2(M, E, N)

$M_1 = AMM(M, PC, N);$
$M_2 = AMM(M_1, M_1, N);$
$M_3 = AMM(M_2, M_1, N);$
$M_6 = AMM(M_3, M_3, N);$
$M_7 = AMM(M_6, M_1, N);$
$R[k-1] = M_1;$
for $i = k-2$ Down to 0 **do**
    $R[i] = AMM(R[i+1], R[i+1], N);$
    if $((e_i \neq 0)\&\&(e_{i-1} \neq 0)\&\&(e_{i-2} \neq 0))$ **then**
        $i = i - 1;$ //Forwarding 1-bit
        $R[i] = AMM(R[i+1], R[i+1], N);$
        $i = i - 1;$ //Forwarding 1-bit
        $R[i] = AMM(R[i+1], R[i+1], N);$
        $R[i] = AMM(R[i], M_7, N);$
    else if $((e_i \neq 0)\&\&(e_{i-1} \neq 0))$ **then**
        $i = i - 1;$ //Forwarding 1-bit
        $R[i] = AMM(R[i+1], R[i+1], N);$
        $R[i] = AMM(R[i], M_3, N);$
    else if $(e_i \neq 0)$ **then**
        $R[i] = AMM(R[i], M_1, N);$
    end if
end for
$R = AMM(R[0], 1, N);$
return R

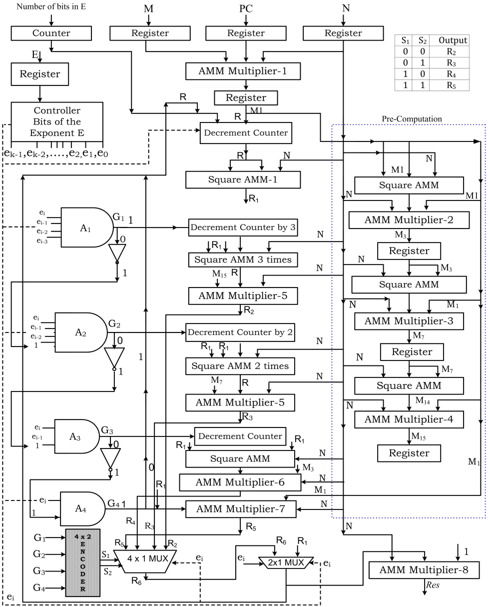


S_1	S_2	Output
0	0	R_1
0	1	R_4
1	0	R_2
1	1	R_3

Bit Forwading 3-bits Algorithm

Algorithm BFW3(M, E, N)

$M_1 = AMM(M, PC, N)$; $M_2 = AMM(M_1, M_1, N)$;
$M_3 = AMM(M_2, M_1, N)$; $M_6 = AMM(M_3, M_3, N)$;
$M_7 = AMM(M_6, M_1, N)$; $M_{14} = AMM(M_7, M_7, N)$;
$M_{15} = AMM(M_{14}, M_1, N)$; $R[k-1] = M_1$;
for $i = k - 2$ Down to 0 **do**
  $R[i] = AMM(R[i+1], R[i+1], N)$;
  if $((e_i)\&\&(e_{i-1})\&\&(e_{i-2})\&\&(e_{i-3}))$ **then**
    $i = i - 1$; //Forwarding one bit//
    $R[i] = AMM(R[i+1], R[i+1], N)$; $i = i - 1$; //Forwarding one bit//
    $R[i] = AMM(R[i+1], R[i+1], N)$; $i = i - 1$; //Forwarding one bit//
    $R[i] = AMM(R[i+1], R[i+1], N)$;
    $R[i] = AMM(R[i], M_{15}, N)$;
  else if $((e_i)\&\&(e_{i-1})\&\&(e_{i-2}))$ **then**
    $i = i - 1$; // Forwarding one bit //
    $R[i] = AMM(R[i+1], R[i+1], N)$; $i = i - 1$; //Forwarding one bit//
    $R[i] = AMM(R[i+1], R[i+1], N)$;
    $R[i] = AMM(R[i], M_7, N)$;
  else if $((e_i \neq 0)\&\&(e_{i-1} \neq 0))$ **then**
    $i = i - 1$; // Forwarding one bit //
    $R[i] = AMM(R[i+1], R[i+1], N)$;
    $R[i] = AMM(R[i], M_3, N)$;
  else if $(e_i \neq 0)$ **then**
    $R[i] = AMM(R[i], M_1, N)$;
  end if
end for
$Res = AMM(R[0], 1, N)$;
return Res



S_1	S_2	Output
0	0	R_2
0	1	R_3
1	0	R_4
1	1	R_5

Algorithms to evaluate $Amod\ N$ and $(A.B)mod\ N$

- Algorithms to evaluate $A\ mod\ N$

Require: A, N and PC
Ensure: $R = A\ mod\ N$

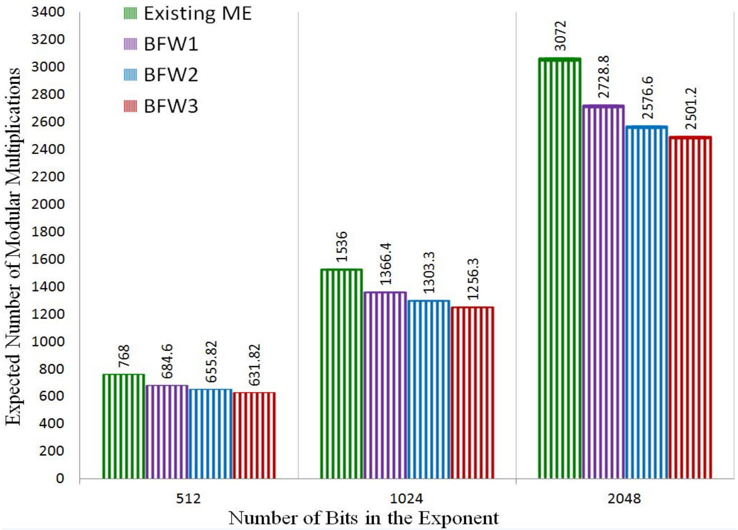
1: $A^1 = AMM(A, PC, N)$;
2: $R = AMM(A^1, 1, N)$;
3: **return** R;

- Algorithms to evaluate $(A.B)mod\ N$

Require: A, B, N and PC
Ensure: $R = (A.B)\ mod\ N$

1: $A^1 = AMM(A, PC, N)$;
2: $B^1 = AMM(B, PC, N)$;
3: $M^1 = AMM(A^1, B^1, N)$;
4: $R = AMM(M^1, 1, N)$;
5: **return** R;

Comparison in terms of Number of MMs



Performance in terms of Clock Cycles

Table: Complexity of different algorithms in terms of clock cycles

Sl. No.	Proposed Algorithms	Clock cycles consumed with proposed MMM	Clock cycles consumed with existing modified Montgomery multiplication	Clock cycles consumed with traditional Montgomery multiplication
1.	MMEC2_42 [1]	$(n+3)(k+N_1+1)$	$(n+5)(k+N_1+1)$	$2n(k+N_1+1)$
2.	BFW1	$(n+3)(k+N_1-N_2+2)$	$(n+5)(k+N_1-N_2+2)$	$2n(k+N_1-N_2+2)$
3.	BFW2	$(n+3)(k+N_1-N_2-2.N_3+4)$	$(n+5)(k+N_1-N_2-2.N_3+4)$	$2n(k+N_1-N_2-2.N_3+4)$
4.	BFW3	$(n+3)(k+N_1-N_2-2.N_3-3.N_4+6)$	$(n+5)(k+N_1-N_2-2.N_3-3.N_4+6)$	$2n(k+N_1-N_2-2.N_3-3.N_4+6)$

Where,

n : Number of bits in the modulus

k : Number of bits in the Exponent

N_1 : Number of 1's in the exponent

N_2 : Number of two independent consecutive 1's in the exponent

N_3 : Number of three independent consecutive 1's in the exponent

N_4 : Number of four independent consecutive 1's in the exponent

Performance Evaluation

- Throughput
- Let Processor Frequency be F and # Clock cycles per CT is X, then Time for CT is : $\frac{X}{F}$
- $\Rightarrow$ Throughput $= \frac{F}{X}$

For hardware devices :

- $Throughput = \frac{Frequency}{Number\ of\ Clock\ Cycles}$

for fixed frequency

- $Throughput \propto \frac{1}{Number\ of\ Clock\ Cycles}$

- The proposed algorithms ↑ the speed by ↓ the number of clock cycles
- The power consumption of proposed designs will also ↓

Performance of existing & proposed designs

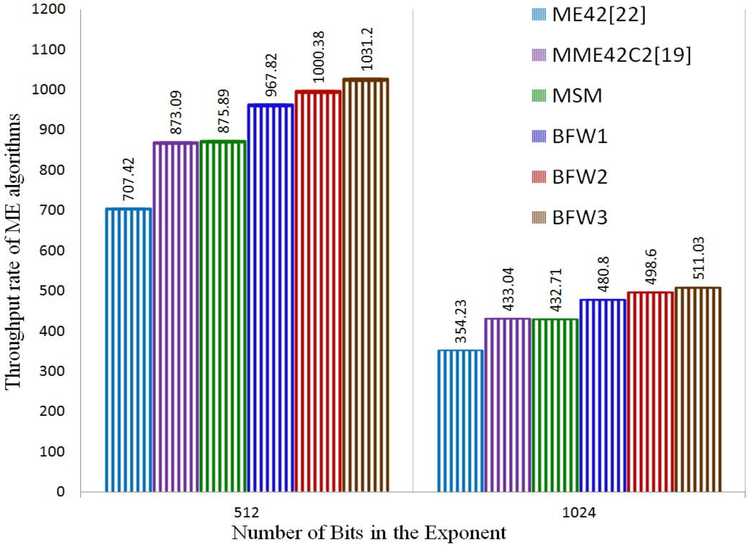
Table: E. Performance of existing designs

Key Length	ME Design	Power (μw)	Avg No. of Modular Multiplications	Area (μm^2)	Throughput Rate(kbps)
512	ME42[12]	41.10	768	498633	707.42
	MME42_C2[1]	19.30	768	351881	873.09
1024	ME42[12]	70.60	1536	852899	354.23
	MME42_C2[1]	40.30	1536	714676	433.04

Table: P. Performance of proposed designs

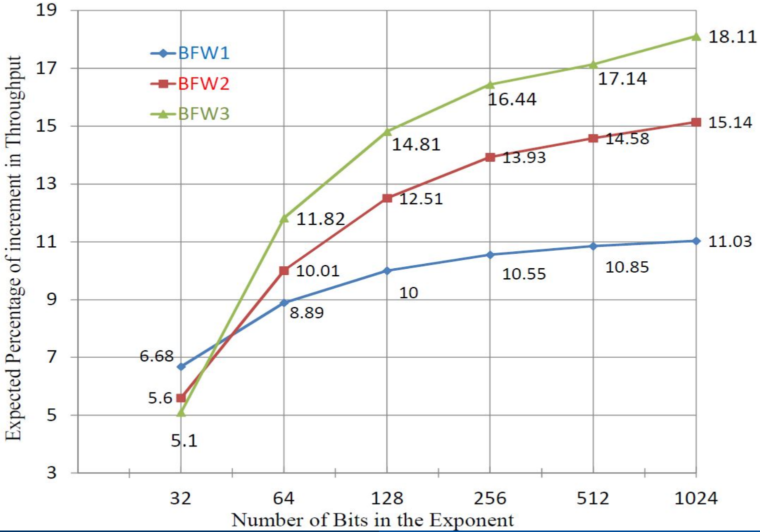
Key Length	ME Design	Power (μw)	Frequency (MHz)	Avg No. of Modular Multiplications	Area (μm^2)	Throughput Rate(kbps)
512	MSM	20.82	672.68	768	352021	875.89
	BFW1	19.29	662.56	685	353053	967.82
	BFW2	18.62	656.20	657	354488	1000.38
	BFW3	18.24	652.26	631	356121	1031.20
1024	MSM	43.30	664.64	1536	714976	432.71
	BFW1	39.52	656.97	1367	717621	480.80
	BFW2	37.74	649.83	1304	720123	498.60
	BFW3	36.86	644.53	1256	722524	511.03

Throughput Comparison between various designs



Throughput Improvement by proposed designs

With Respect to the design MME42_C2 [1]



Energy calculation

The energy consumed is given by $E = T.C_t$, where

T : Time needed for encryption/decryption

C_t : The power consumed

Energy $\propto$ The power consumed and is calculated as follows :

$$\begin{aligned} Energy &= Power \times Execution\ Time \\ &= Power \times Clock\ Period \times Number\ of\ Clock\ Cycles \end{aligned}$$

where *clock period* is the reciprocal of the clock frequency

The proposed algorithms taking less energy in comparison with the state-of-the-art

Energy comparison

Table: Energy comparison between existing and proposed ME designs

Key Length	ME Design		Energy (μJ)
512	Existing Designs	ME42[12]	109.38
		MME42_C2[1]	41.65
	Proposed Designs	MSM	44.93
		BFW1	41.62
		BFW2	40.18
		BFW3	38.96
1024	Existing Designs	ME42[12]	748.26
		MME42_C2[1]	344.52
	Proposed Designs	MSM	370.16
		BFW1	337.85
		BFW2	322.63
		BFW3	309.72

Selection Criteria for BFWk

- The proposed BFW1, BFW2 and BFW3 can be $\rightarrow\rightarrow$ ... $\rightarrow$ to BFW4, BFW5, up to BFWk

The choice among the various BFWn, (for $n = 1, 2, 3, ..., k$) algorithms will be based on the requirement of the application :

- Applications which focus on higher throughput: Can prefer a particular BFWj with an appropriate saving in the number of clock cycles
- Higher throughput with physical area constraints: Settle down with appropriate *time-area* trade of that can compromise the requirements.
- If the area constraints are relaxed: The choice of Bit Forwarding j-bits algorithm can be narrowed down based on the maximum value of *NoM*
- For example smart card applications that have memory restrictions: BFW1

NoM : Number of Multiplications reduced by BFWj :

- Let f_i be the frequency of i consecutive ones, for $i = 2, 3, 4, ..., j, j + 1$ and $f_i \geq 0$ in the exponent, then
- $NoM = \sum_{i=2}^{j+1} f_i.(i - 1)$ represents the number of multiplications reduced by *BFWj*

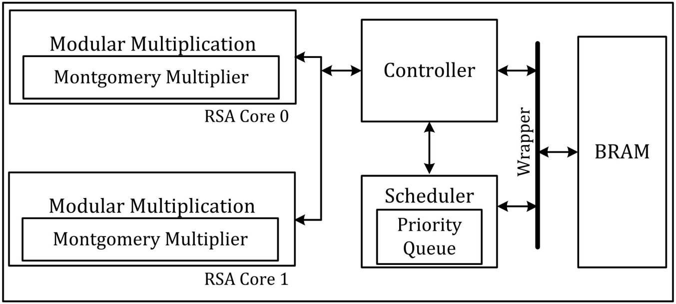
Concurrent Cryptographic Transformations

A Dual-core RSA processor(DCRSAP)

- To carry out concurrent cryptographic transformations
- To improve the throughput without changing the frequency
- MSM based DCRSAP and BFW1 based DCRSAP

Dual-core RSA processor

- Block Schematic diagram of Dual-core RSA Processor



- The prime modules are
  - Controller
  - Scheduler
  - Block BRAM
  - RSA Core

Dual-core RSA processor ... Contd.

Controller

- Checks for a FREE RSA core and assign the task
- No RSA core is FREE, it enques the task in priority queue
- Priority of the task has been customized based on the application
- If any RSA core becomes FREE, it deques the task assigned to that particular RSA core
- It ensures balance load factor
- Once the RSA core completes its processing, controller write the results in BRAM
- The task stored in the queue has the following structure

0 15

Block Starting Address	
Process ID(16-23)	Priority(24-31)

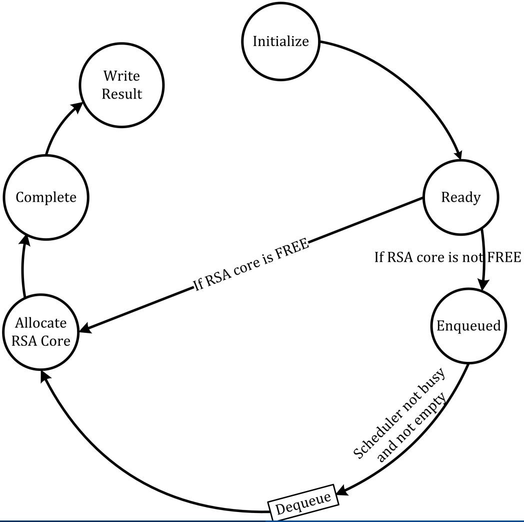
Dual-core RSA processor ... Contd.

- The sequence of steps executed by the controller

```
Initialize the RAM and Scheduler;
Initialize RSA core;
Create Registers of M,N,E,PC; which will be useful for latching values of particular task;
if (Any one of the RSA core is FREE) then
    Assign task to that core;
else
    while ((!Scheduler is busy) & (!Scheduler is empty)) do
        de-queue value from the scheduler(Priority queue);
        parse output of scheduler and store into the RAM;
        Load values from RAM, poited by the starting address;
        Check if any RSA core is free or not;
        if (FREE) then
            Take a process from the Scheduler and schedule it;
        else
            wait for any RSA core to become IDLE;
            Assign task to RSA core;
        end if
    end while
end if
Write the result of RSA to BRAM;
```

Dual-core RSA processor ... Contd.

- State Diagram of Controller



Dual-core RSA processor ... Contd.

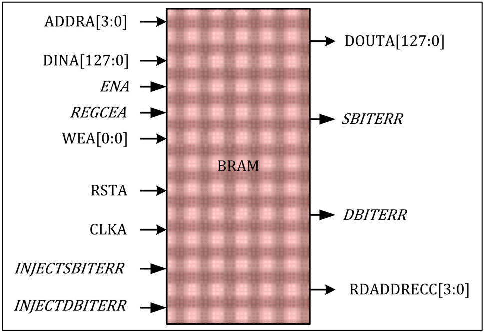
Hardware Scheduler

- Scheduler can work in parallel
- Heap based priority queue is implemented
- It consists of 32 registers, each is of 32 bits in length (Scheduler is scalable up to 32 cores)
- In linear aray form $i/2$ is parent, $2i$ and $2i + 1$ childs
- Binary heap is implemented in the hardware
- Performs enqueue operation in $O(1)$ time complexity and dequeue in $O(log\ n)$ time
- The scheduler takes the process- ID, priority of the process and the starting address of the process from BRAM as input.
- The scheduler uses binary heap as its data structure with MAX-HEAP property
- The two operations used in this hardware scheduler are :
  - Enqueue operation
  - Dequeue operation

Dual-core RSA processor ... Contd.

BRAM Controller

- Block RAM
  - It provides interface for reading and writing of data from and into BRAM
- Structure of BRAM



- BRAM in FPGA
- BRAM in ASIC

Dual-core RSA processor ... Contd.

➤ Power and Area of Proposed modules

Key Length	Module	Power(μw)	Area(μm^2)
512	MSM based RSA core	19.47	350021
	BFW1 based RSA core	19.29	352013
	Scheduler	0.41	1367
	BRAM	0.13	912
	MSM based DCRSAP	39.48	702321
	BFW1 based DCRSAP	39.02	706305
1024	MSM based RSA core	40.72	713976
	BFW1 based RSA core	39.52	715621
	Scheduler	0.76	1367
	BRAM	0.23	2176
	MSM based DCRSAP	82.33	1431495
	BFW1 based DCRSAP	80.03	1434785

Security Analysis

References

[1] Shiann-Rong Kuang, Jiun-Ping Wang, Kai-Cheng Chang, and Huan-Wei Hsu. Energy-efficient high-throughput montgomery modular multipliers for rsa cryptosystems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 21(11):1999–2009, 2013.

[2] Shiann-Rong Kuang, Kun-Yi Wu, and Ren-Yao Lu. Low-cost high-performance vlsi architecture for montgomery modular multiplication.

[3] Atsushi Miyamoto, Naofumi Homma, Takafumi Aoki, and Akashi Satoh. Systematic design of rsa processors based on high-radix montgomery multipliers. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 19(7):1136–1146, 2011.

[4] Miaoqing Huang, Kris Gaj, and Tarek El-Ghazawi. New hardware architectures for montgomery modular multiplication algorithm. *Computers, IEEE Transactions on*, 60(7):923–936, 2011.

[5] Gustavo D Sutter, Jean-Pierre Deschamps, and José Luis Imaña. Modular multiplication and exponentiation architectures for fast rsa cryptosystem based on digit serial computation. *Industrial Electronics, IEEE Transactions on*, 58(7):3101–3109, 2011.

[6] Gavin Xiaoxu Yao, Junfeng Fan, Ray CC Cheung, and Ingrid Verbauwhede. Novel rns parameter selection for fast modular multiplication. *Computers, IEEE Transactions on*, 63(8):2099–2105, 2014.

[7] Lejla Batina, Sıddıka Berna Örs, Bart Preneel, and Joos Vandewalle. Hardware architectures for public key cryptography. *Integration, the VLSI journal*, 34(1):1–64, 2003.

[8] Ming-Der Shieh, Jun-Hong Chen, Hao-Hsuan Wu, and Wen-Ching Lin. A new modular exponentiation architecture for efficient design of rsa cryptosystem. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(9):1151–1161, 2008.

[9] Ming-Der Shieh, Jun-Hong Chen, Wen-Ching Lin, and Hao-Hsuan Wu. A new algorithm for high-speed modular multiplication design. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 56(9), 2009.

[10] Peter L Montgomery. Modular multiplication without trial division. *Mathematics of computation*, 44(170):519–521, 1985.

[11] Peter L Montgomery. A survey of modern integer factorization algorithms. *CWI quarterly*, 7(4):337–366, 1994.

[12] Ciaran McIvor, Maire McLoone, and John V McCanny. Modified montgomery modular multiplication and rsa exponentiation techniques. *IEE Proceedings-Computers and Digital Techniques*, 151(6):402–408, 2004.

[13] Wen-Ching Lin, Jheng-Hao Ye, and Ming-Der Shieh. Scalable montgomery modular multiplication architecture with low-latency and low-memory bandwidth requirement. *Computers, IEEE Transactions on*, 63(2):475–483, 2014.

[14] Chia-Long Wu. An efficient common-multiplicand-multiplication method to the montgomery algorithm for speeding up exponentiation. *Information sciences*, 179(4):410–421, 2009.

[15] Atef Ibrahim, Fayez Gebali, Hamed Elsimary, and Amin Nassar. Processor array architectures for scalable radix 4 montgomery modular multiplication algorithm. *Parallel and Distributed Systems, IEEE Transactions on*, 22(7):1142–1149, 2011.

[16] Daesung Lim, Nam Su Chang, Sung Yeon Ji, Chang Han Kim, Sangjin Lee, and Young-Ho Park. An efficient signed digit montgomery multiplication for rsa. *Journal of Systems Architecture*, 55(7):355–362, 2009.

[17] Nadia Nedjah, LM Mourelle, M Santana, and S Raposo. Massively parallel modular exponentiation method and its implementation in software and hardware for high-performance cryptographic systems. *IET computers & digital techniques*, 6(5):290–301, 2012.

[18] João Carlos Néto, Alexandre Ferreira Tenca, and Wilson Vicente Ruggiero. A parallel and uniform-partition method for montgomery multiplication. *IEEE Transactions on Computers*, 63(9):2122–2133, 2014.

[19] Xiaofeng Chen, Jin Li, Jianfeng Ma, Qiang Tang, and Wenjing Lou. New algorithms for secure outsourcing of modular exponentiations. *IEEE Transactions on Parallel and Distributed Systems*, 25(9):2386–2396, 2014.

[20] Dimitrios Schinianakis and Thanos Stouraitis. Multifunction residue architectures for cryptography. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(4):1156–1169, 2014.

[21] Abdalhossein Rezai and Parviz Keshavarzi. High-throughput modular multiplication and exponentiation algorithms using multibit-scan–multibit-shift technique. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(9):1710–1719, 2015.

[22] Masahiro Kaminaga, Hideki Yoshikawa, and Toshinori Suzuki. Double counting in-ary rsa precomputation reveals the secret exponent. *IEEE Transactions on Information Forensics and Security*, 10(7):1394–1401, 2015.

[23] Xinming Huang and Wei Wang. A novel and efficient design for an rsa cryptosystem with a very large key size. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(10):972–976, 2015.

[24] Hari Krishna Garg and Hanshen Xiao. New residue arithmetic based barrett algorithms: Modular integer computations. *IEEE Access*, 4:4882–4890, 2016.

[25] Mehdi Tibouchi and Taechan Kim. Improved elliptic curve hashing and point representation. *Designs, Codes and Cryptography*, pages 1–17, 2016.

[26] Reza R Farashahi, Pierre-Alain Fouque, Igor Shparlinski, Mehdi Tibouchi, and J Voloch. Indifferentiable deterministic hashing to elliptic and hyperelliptic curves. *Mathematics of Computation*, 82(281):491–512, 2013.

[27] Joppe W Bos, Craig Costello, Patrick Longa, and Michael Naehrig. Selecting elliptic curves for cryptography: An efficiency and security analysis. *Journal of Cryptographic Engineering*, pages 1–28, 2015.

[28] Steven D Galbraith. *Mathematics of public key cryptography*. Cambridge University Press, 2012.

[29] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[30] William Stallings. *Cryptography and network security: principles and practices*. Pearson Education India, 2006.

Thank You