Multi-Exponentiation Algorithm

Chien-Ning Chen

Email: chienning@ntu.edu.sg

Feb 15, 2012

Coding and Cryptography Research Group

Outline

- Review of multi-exponentiation algorithms
- Double/Multi-exponentiation based on binary GCD algorithm
- Side-channel analysis and countermeasures

Double-/Multi-Exponentiation

- Evaluating the product of several exponentiations
  - *Example*: double-exponentiation, $x^a y^b$, in many digital signature verification primitives
  - *Example*: multi-exponentiation, ${g_1}^{e_1} \cdots {g_h}^{e_h}$, in bilinear ring signature and batch verification of signatures

- Existing better algorithms than "multiplying the results of individual exponentiations"
  - Most are developed based on Shamir's simultaneous squaring algorithm (1985) (Also proposed by Straus in 1964)

Square and Multiply Algorithm

INPUT: base number g,
exponent $e = (e_{k-1} \cdots e_0)$
OUTPUT: g^e

$\mathsf{R} = 1$	$\Leftarrow$ Accumulator
for $i = k-1$ to 0 step -1	$\Leftarrow$ From MSB to LSB
$\mathsf{R} = \mathsf{R}^2$	$\Leftarrow$ Square
if $e_i = 1$ then $\mathsf{R} = \mathsf{R} \times g$	$\Leftarrow$ Multiply (optional)
return R	

- Left-to-right algorithm, complexity $= 1.5k$ multiplications

Shamir's Simultaneous Squaring Multi-Exponentiation Algorithm

INPUT: base numbers $g_1 \sim g_h$,
exponents $e_1 \sim e_h$ where $e_j = (e_{j,k-1} \cdots e_{j,0})$
OUTPUT: ${g_1}^{e_1} \times \cdots \times {g_h}^{e_h}$

$\mathsf{R} = 1$
for $i = k-1$ to 0 step -1
  $\mathsf{R} = \mathsf{R}^2$ $\Leftarrow$ Simultaneous squaring
  $\mathsf{R} = \mathsf{R} \times ({g_1}^{e_{1,i}} \times \cdots \times {g_h}^{e_{h,i}})$ $\Leftarrow$ How to compute it?
return R

- If $g_1 \times g_2$ is pre-computed Table $= \{g_1, g_2, g_1 \times g_2\}$
  Complexity of double-exponentiation $= 1.75k$

Interleaving / Simultaneous Multi-Exponentiation[1]

- How to compute $\boxed{\mathsf{R} \times (g_1{}^{e_{1,i}} \times \cdots \times g_h{}^{e_{h,i}})}$
  - Interleaving: compute each of multiplications
    - $hk/2$ multiplications on average (h terms, k-bit exponents)
  - Simultaneous: prepare a table $\{g_1{}^{\epsilon_1} \times \cdots \times g_h{}^{\epsilon_h}\}$
    - $k(1 - 2^{-h}) \approx k$ multiplications,
      where 2^{-h} is the prob. of $e_{1,i} = \cdots = e_{h,i} = 0$
    - Table size $= (2^h - h - 1)$

- Exponent recoding can further improve performance
  - Table size grows faster than in single-exponentiation

[1] Named by Bodo Möller

Interleaving Double-Exp. with Window Method

- Separately recode exponents by sliding/fractional window method

Recoding Method	Avg. HW	Double-Exp. with $w = 2$
w-bit sliding window	$\frac{k}{w+1}$	$(1 + \frac{2}{w+1})k = 1.66k$
w-bit signed sliding window	$\frac{k}{w+2}$	$(1 + \frac{2}{w+2})k = 1.5k$

- *Example*: 2-bit signed sliding window, digit set $\{0, \pm 1, \pm 3\}$

$$b = 334 = 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0 \Rightarrow \text{binary}$$
$$0\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 3\ 0 \Rightarrow \text{2-bit sliding window}$$
$$0\ 0\ 3\ 0\ 0\ \bar{3}\ 0\ 0\ \bar{1}\ 0 \Rightarrow \text{2-bit signed sliding window}$$

- Prepare separate tables: $\{x, x^3, x^{-1}, x^{-3}\}$, $\{y, y^3, y^{-1}, y^{-3}\}$

Simultaneous Double-Exp. with Recodings

- Recode exponents by NAF or Joint Sparse Form

- *Example*: double-exp. $x^a y^b$, Average complexity

$$\left.\begin{array}{r} a = 403 = 0\,1\,1\,0\,0\,1\,0\,0\,1\,1 \\ b = 334 = 0\,1\,0\,1\,0\,0\,1\,1\,1\,0 \end{array}\right\} \Rightarrow$$ binary, average j-HW $= 0.75k$ — $1.75k$

$$\left.\begin{array}{r} 1\,0\,\bar{1}\,0\,0\,1\,0\,1\,0\,\bar{1} \\ 0\,1\,0\,1\,0\,1\,0\,0\,\bar{1}\,0 \end{array}\right\} \Rightarrow$$ NAF recoding separately, average j-HW $= 0.56k$ — $1.56k$

$$\left.\begin{array}{r} 1\,0\,\bar{1}\,0\,0\,1\,0\,0\,1\,1 \\ 1\,0\,\bar{1}\,\bar{1}\,0\,1\,0\,0\,\bar{1}\,0 \end{array}\right\} \Rightarrow$$ JSF recoding, average j-HW $= 0.5k$ — $1.5k$

- Prepare joint table: $\{x, x^{-1}, y, y^{-1}, xy, (xy)^{-1}, xy^{-1}, x^{-1}y\}$

Binary GCD Multi-Exponentiation Algorithm

Euclidean Double-Exponentiation Algorithm[2]

- Find GCD of a and b when evaluating $x^a y^b$
  - $\gcd(a,b) = \gcd(b, a \bmod b) = \gcd(b,r)$ where $a = bq + r$
  - $x^a y^b = x^{(bq+r)} y^b = x^r (x^{bq} y^b) = (yx^q)^b x^r = z^b x^r = \cdots$

- Evaluate the double-exponentiation $x^a y^b$ by
  1. Initialize: $(\mathsf{A}_{\langle 0\rangle}, \mathsf{B}_{\langle 0\rangle}, \mathsf{X}_{\langle 0\rangle}, \mathsf{Y}_{\langle 0\rangle}) = (a, b, x, y)$
  2. $(\mathsf{A}_{\langle i+1\rangle}, \mathsf{B}_{\langle i+1\rangle}, \mathsf{X}_{\langle i+1\rangle}, \mathsf{Y}_{\langle i+1\rangle}) =$
     $(\mathsf{B}_{\langle i\rangle}, \mathsf{A}_{\langle i\rangle} \bmod \mathsf{B}_{\langle i\rangle}, \mathsf{Y}_{\langle i\rangle} \times {\mathsf{X}_{\langle i\rangle}}^{\lfloor \mathsf{A}_{\langle i\rangle}/\mathsf{B}_{\langle i\rangle} \rfloor}, \mathsf{X}_{\langle i\rangle})$
  3. Terminate: $x^a y^b = {\mathsf{X}_{\langle i\rangle}}^{\mathsf{A}_{\langle i\rangle}}$ when $\mathsf{B}_{\langle i\rangle} = 0$

---

[2] In 1989, Bergeron *et al.* firstly employed Euclidean algorithm to construct continued fractions for evaluating double-exponentiation.

Binary GCD Algorithm

- Alternate method to find greatest common divisor, base on
  1. $\gcd(a, b) = 2\gcd(a/2, b/2)$, when both a and b are even
  2. $\gcd(a, b) = \gcd(a/2, b)$, when a is even and b is odd
  3. $\gcd(a, b) = \gcd(a - b, b)$, when $a \geq b$
- Recursively perform $\gcd(a, b) = \begin{cases} \gcd((a-b)/2^k, b) \\ \gcd(a, (b-a)/2^k) \end{cases}$
- More efficient when handling long integers
  - No long-integer modular operation
  - Only subtraction and right shifting (divided by 2)

Binary GCD Double-Exponentiation Algorithm

- Compute GCD of exponents by binary GCD algorithm
  - $x^a y^b = (x^2)^{a/2} y^b$ (if a is even) or $= x^{a-b}(xy)^b$ (if $a \geq b$)

$$(\mathsf{A}_{\langle i+1\rangle}, \mathsf{B}_{\langle i+1\rangle}, \mathsf{X}_{\langle i+1\rangle}, \mathsf{Y}_{\langle i+1\rangle}) = \begin{cases} (\mathsf{A}_{\langle i\rangle}/2, \mathsf{B}_{\langle i\rangle}, {\mathsf{X}_{\langle i\rangle}}^2, \mathsf{Y}_{\langle i\rangle}) & \text{if } \mathsf{A}_{\langle i\rangle} \text{ is even} \\ (\mathsf{A}_{\langle i\rangle}, \mathsf{B}_{\langle i\rangle}/2, \mathsf{X}_{\langle i\rangle}, {\mathsf{Y}_{\langle i\rangle}}^2) & \text{if } \mathsf{B}_{\langle i\rangle} \text{ is even} \\ (\mathsf{A}_{\langle i\rangle} - \mathsf{B}_{\langle i\rangle}, \mathsf{B}_{\langle i\rangle}, \mathsf{X}_{\langle i\rangle}, \mathsf{X}_{\langle i\rangle}\mathsf{Y}_{\langle i\rangle}) & \text{if } \mathsf{A}_{\langle i\rangle} \geq \mathsf{B}_{\langle i\rangle} \\ (\mathsf{A}_{\langle i\rangle}, \mathsf{B}_{\langle i\rangle} - \mathsf{A}_{\langle i\rangle}, \mathsf{X}_{\langle i\rangle}\mathsf{Y}_{\langle i\rangle}, \mathsf{Y}_{\langle i\rangle}) & \text{if } \mathsf{B}_{\langle i\rangle} > \mathsf{A}_{\langle i\rangle} \end{cases}$$

- Require about $1.4 \log_2 a$ squarings and $0.7 \log_2 a$ multiplications when evaluating $x^a y^b$ when $a \approx b$ Complexity $= 2.1k$

Analysis of bGCD Double-Exp. Alg.

- Evaluate performance[3] by $\log_2(\mathsf{A}_{\langle i\rangle}\mathsf{B}_{\langle i\rangle})$ (i.e., length of $\mathsf{A}_{\langle i\rangle}\mathsf{B}_{\langle i\rangle}$)
- Halving: $(\mathsf{A}_{\langle i+1\rangle}, \mathsf{B}_{\langle i+1\rangle}, \mathsf{X}_{\langle i+1\rangle}, \mathsf{Y}_{\langle i+1\rangle}) = (\mathsf{A}_{\langle i\rangle}/2, \mathsf{B}_{\langle i\rangle}, {\mathsf{X}_{\langle i\rangle}}^2, \mathsf{Y}_{\langle i\rangle})$
  - $\log_2(\mathsf{A}_{\langle i\rangle}) - \log_2(\mathsf{A}_{\langle i\rangle}/2) = 1$, always reduce 1 bit
- Subtraction: $(\cdots) = (\mathsf{A}_{\langle i\rangle} - \mathsf{B}_{\langle i\rangle}, \mathsf{B}_{\langle i\rangle}, \mathsf{X}_{\langle i\rangle}, \mathsf{X}_{\langle i\rangle}\mathsf{Y}_{\langle i\rangle})$
  - $\log_2(\mathsf{A}_{\langle i\rangle}) - \log_2(\mathsf{A}_{\langle i\rangle} - \mathsf{B}_{\langle i\rangle})$ depends on $\mathsf{A}_{\langle i\rangle}/\mathsf{B}_{\langle i\rangle}$
  - Reduce more bits when $\mathsf{A}_{\langle i\rangle} \approx \mathsf{B}_{\langle i\rangle}$
  - Reduce almost nothing when $\mathsf{A}_{\langle i\rangle} \gg \mathsf{B}_{\langle i\rangle}$

[3] Referring to the analysis of Brent in 1976.

Improvement to bGCD Double-Exp. Alg.

- **Strategy 1**: Always perform subtraction when $\mathsf{A}_{\langle i\rangle} \approx \mathsf{B}_{\langle i\rangle}$
  - Subtraction has better performance than halving if $\mathsf{A}_{\langle i\rangle} \approx \mathsf{B}_{\langle i\rangle}$
  - Determine $\mathsf{A}_{\langle i\rangle} \approx \mathsf{B}_{\langle i\rangle}$ by length, $\lfloor \log_2 \mathsf{A}_{\langle i\rangle} \rfloor - \lfloor \log_2 \mathsf{B}_{\langle i\rangle} \rfloor \leq 1$
- **Strategy 2**: Append 1^1 to be a triple-exp. $x^a y^b 1^1$
  - Solve the worst case, $\mathsf{A}_{\langle i\rangle}$ is odd, $\mathsf{B}_{\langle i\rangle}$ is even, $\mathsf{A}_{\langle i\rangle} \gg \mathsf{B}_{\langle i\rangle}$
  - $\mathsf{A}_{\langle i+1+k\rangle} = (\mathsf{A}_{\langle i\rangle} - 1)/2^k$,
    until $\mathsf{A}_{\langle i+1+k\rangle}$ is odd, or $\mathsf{A}_{\langle i+1+k\rangle} \approx \mathsf{B}_{\langle i\rangle}$
  - Require 1 additional variable,
    ${\mathsf{X}_{\langle i\rangle}}^{\mathsf{A}_{\langle i\rangle}} {\mathsf{Y}_{\langle i\rangle}}^{\mathsf{B}_{\langle i\rangle}} \mathsf{Z}_{\langle i\rangle} = {\mathsf{X}_{\langle i\rangle}}^{(\mathsf{A}_{\langle i\rangle} - 1)} {\mathsf{Y}_{\langle i\rangle}}^{\mathsf{B}_{\langle i\rangle}} (\mathsf{Z}_{\langle i\rangle} \times \mathsf{X}_{\langle i\rangle})$

Comparison of Double-Exp. Alg.

- Performance comparison of 1024-bit double-exponentiation

Algorithm	Avg. # of Operations			Avg. Comp.	Variables	
	Square	Mul.	Sum		Base	Exp
Euclidean	749.7	896.8	1646.5	1.6079	3	3
Binary GCD	1445.3	723.3	2168.5	2.1177	2	2
Strategy 1	724.8	1048.1	1772.9	1.7314	2	2
Strategy 1&2	503.5	1106.8	1610.3	1.5726	3	2
Simult. binary	1024	768.0	1792.0	1.75	4	2
Simult. JSF	1024	512.0	1536.0	1.5	5 / 9	2
Inter. binary	1024	1024.0	2048.0	2.0	3	2
Inter. 2-uSW	1024	682.7	1706.7	1.6667	5	2
Inter. 2-sSW	1024	512.0	1536.0	1.5	5 / 9	2

binary GCD Multi-Exp. Algorithm

- Follow the same strategies of binary GCD double-exponentiation to reduce the largest exponent as efficient as possible

- No pre-computation table, memory efficiency

- Scalable from single exp. $g^e 1^1$ to multi-exp.
  Good performance for any bit length of exponents

Performance of High-Dimensional

- Performance comparison of 1024-bit multi-exponentiation

Term	Algorithm	Avg. # of		Avg. Comp.	Variables	
		Square	Mul.		Base	Exp.
3	bGCD	284.3	1503.5	1.746	4	3
	Simult. binary	1024.0	4+ 896.0	0.004+1.875	8	3
	Inter. binary	1024.0	1536.0	2.500	4	3
4	bGCD	173.7	1822.4	1.949	5	4
	Simult. binary	1024.0	11+ 960.0	0.010+1.938	16	4
	Inter. binary	1024.0	2048.0	3.000	5	4
10	bGCD	20.8	3301.1	3.244	10	10
	Simult. binary	1024.0	1013+1023.0	0.989+1.999	1024	10
	Inter. binary	1024.0	5120.0	6.000	11	10

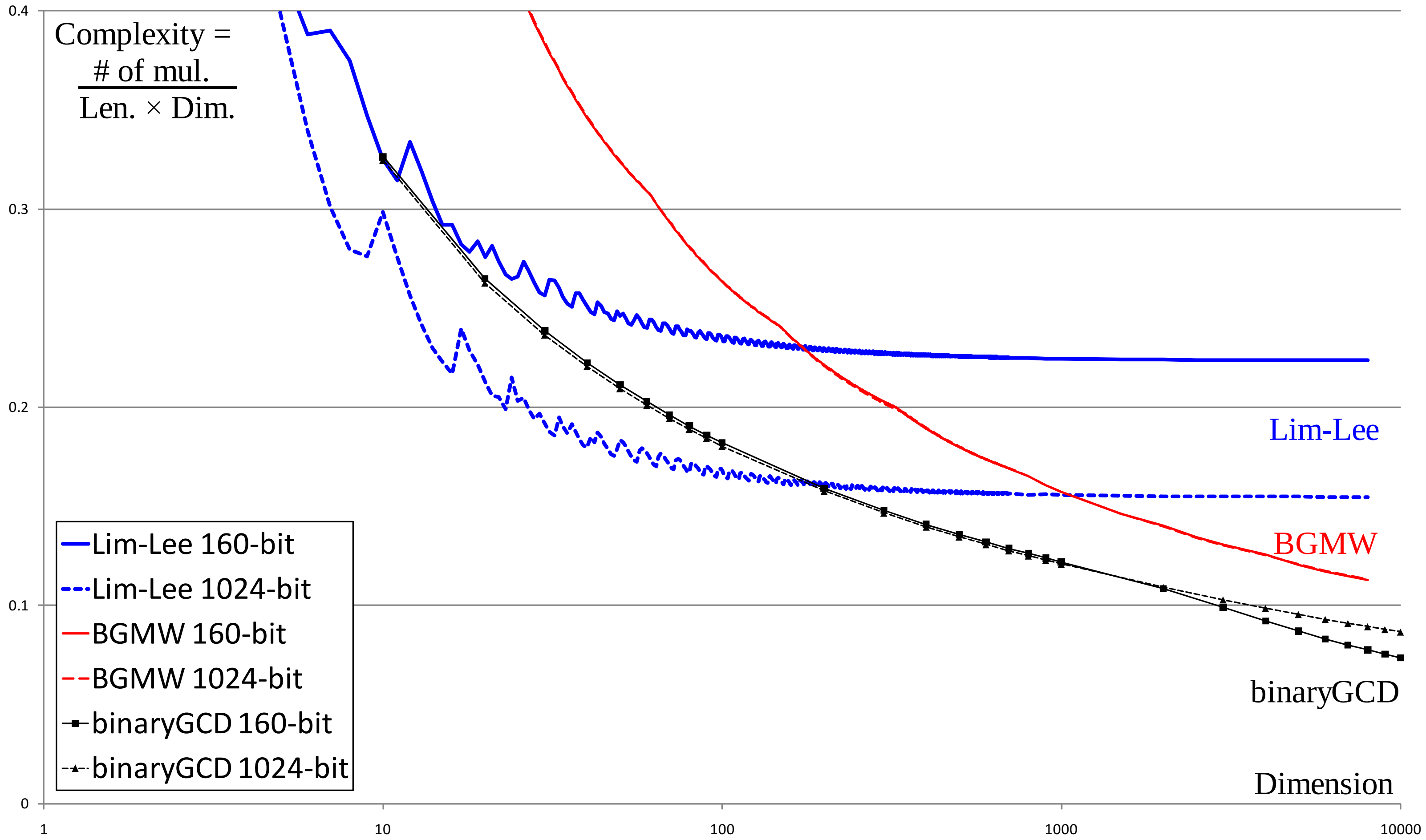
Lim-Lee Algorithm and BGMW Method

- Lim-Lee: simultaneous exponentiation with multiple smaller tables
  - Split h base numbers into l set, construct table on each set
  - Table size reduced from $O(2^h)$ to $O(l2^{h/l})$, l-fold multiplications

03	$R = R^2$
04	$R = R \times ({g_1}^{e_{1,i}} \times \cdots \times {g_w}^{e_{w,i}}) \times ({g_{w+1}}^{e_{w+1}} \times \cdots \times {g_{2w}}^{e_h}) \times \cdots$

- BGMW: w-bit fixed window with a special comp. sequence
  - *Example*: $R^8 \times ({g_1}^3 \times g_2 \times {g_3}^3 \times {g_4}^4) \rightarrow$
    $R^8 \times (g_4) \times (g_4 g_1 g_3) \times (g_4 g_1 g_3) \times (g_4 g_1 g_3 g_2)$

Comparison with Lim-Lee and BGMW



Side-Channel Analysis and Countermeasures

Simple Power Analysis

- Attacker can distinguish squaring and multiplication
  - How much info can be retrieved from S and M sequence?

- Left-to-right binary square-and-multiply algorithm

```
R = R^2                          ⇐ Squaring always happens
if e_i = 1 then R = R × g        ⇐ Mul. indicates a nonzero bit
```

  - Fully recover private exponent when retrieving one sequence
  - *Example*: . . . S M S S S M S M . . . indicates . . . 10011 . . .

Immunity Against Simple Power Analysis

- bGCD multi-exp. alg. is natively with immunity against SPA, because both base numbers are updated
  - When squaring occurs, $(\mathsf{X}_{\langle i+1\rangle}, \mathsf{Y}_{\langle i+1\rangle}) = \begin{cases} ({\mathsf{X}_{\langle i\rangle}}^2, \mathsf{Y}_{\langle i\rangle}) \\ (\mathsf{X}_{\langle i\rangle}, {\mathsf{Y}_{\langle i\rangle}}^2) \end{cases}$, can not distinguish which variable is squared
  - When multiplication occurs, $(\mathsf{X}_{\langle i+1\rangle}, \mathsf{Y}_{\langle i+1\rangle}) = \begin{cases} (\mathsf{X}_{\langle i\rangle}\mathsf{Y}_{\langle i\rangle}, \mathsf{Y}_{\langle i\rangle}) \\ (\mathsf{X}_{\langle i\rangle}, \mathsf{X}_{\langle i\rangle}\mathsf{Y}_{\langle i\rangle}) \end{cases}$, can not distinguish which variable is overwritten

- More than $1.5k$ indistinguishable operations in k-bit double-exp.

Differential Power Analysis

- Statistical methods to test whether expected values appear
  - Power consumption depends on operand

- In left-to-right square-and-multiply algorithm, if attacker has retrieved MSBs of exponent, $\mathsf{E}_{\langle i+1 \rangle} = (e_{k-1} \cdots e_{i+1})$

1. Calculate $\mathsf{R}_{\langle i+1 \rangle} = g^{\mathsf{E}_{\langle i+1 \rangle}}$
2. $\mathsf{E}_{\langle i \rangle} = (e_{k-1} \cdots e_{i+1} e_i)$, guess $e_i = 0$ or 1
3. Either $\mathsf{R}_{\langle i \rangle} = \mathsf{R}_{\langle i+1 \rangle}{}^2$ or $\mathsf{R}_{\langle i \rangle} = \mathsf{R}_{\langle i+1 \rangle}{}^2 \times g$
4. Test whether $\left(\mathsf{R}_{\langle i \rangle}\right)^2$ or $\left(\mathsf{R}_{\langle i \rangle}\right)^2$ appears by DPA

Immunity Against Differential Power Analysis

- To prevent DPA by appending r^{ϕ}, where r is a random number and ϕ is the order of group
  - Single exp. $g^e \Longrightarrow g^e r^{\phi} 1^1$, Complexity $= 1.5726k$
  - Double exp. $x^a y^b \Longrightarrow x^a y^b r^{\phi} 1^1$, Complexity $= 1.7461k$
- The intermediate values will be of the form: $g^{\alpha} r^{\beta}$
  - Cannot guess them because r is unknown $\Rightarrow$ NO DPA
  - After computation, we have $\left(g^{\alpha} r^{\beta}\right)^0 \left(g^{\alpha'} r^{\beta'}\right)^0 \left(g^e\right)^1$
    - $*$ Either $g^{\alpha} r^{\beta}$ or $g^{\alpha'} r^{\beta'}$ will be the next random number

Summary of bGCD Multi-exp. Alg.

- Comparable performance, scalable from single exp. to multi-exp.
- No pre-computation table, no inversion computation
- Side-channel immunity

- No explicit proof of complexity, only simulation
- All variables will be overwritten during computation

Thank You