Breaking RSA Generically is Equivalent to Factoring

Divesh Aggarwal and Ueli Maurer

ETH Zurich

April 27, 2009

RSA public key cryptosystem

Public Key: (n, e) $\quad n = pq \quad$ $\gcd(e, \phi(n)) = 1$.

Encryption: $m \mapsto c = m^e \mod n$.

RSA public key cryptosystem

Public Key: (n, e) $\quad n = pq \quad$ $\gcd(e, \phi(n)) = 1$.

Encryption: $m \mapsto c = m^e \mod n$.

Breaking RSA $\iff$ $c \mapsto c^{1/e} \mod n$.

RSA public key cryptosystem

Public Key: (n, e) $\quad n = pq \quad$ $\gcd(e, \phi(n)) = 1$.

Encryption: $m \mapsto c = m^e \mod n$.

Breaking RSA $\iff$ $c \mapsto c^{1/e} \mod n$.

Factoring n $\quad$ Breaking RSA

RSA public key cryptosystem

Public Key: (n, e) $\quad n = pq \quad$ $\gcd(e, \phi(n)) = 1$.

Encryption: $m \mapsto c = m^e \mod n$.

Breaking RSA $\iff$ $c \mapsto c^{1/e} \mod n$.

Factoring n $\quad$ Breaking RSA

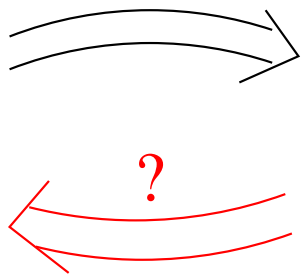
Is it necessary to factor n to break RSA?

RSA public key cryptosystem

Public Key: (n, e) $\quad n = pq \quad$ $\gcd(e, \phi(n)) = 1$.

Encryption: $m \mapsto c = m^e \mod n$.

Breaking RSA $\iff$ $c \mapsto c^{1/e} \mod n$.

Factoring n  Breaking RSA

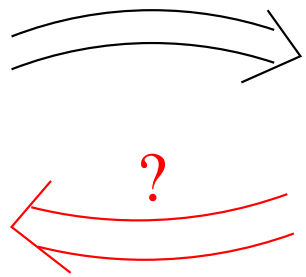
- General models of computation: Unknown

RSA public key cryptosystem

Public Key: (n, e) $\quad n = pq \quad$ $\gcd(e, \phi(n)) = 1$.

Encryption: $m \mapsto c = m^e \mod n$.

Breaking RSA $\iff$ $c \mapsto c^{1/e} \mod n$.

Factoring n  Breaking RSA

- General models of computation: Unknown
- The generic model of computation: This paper!!

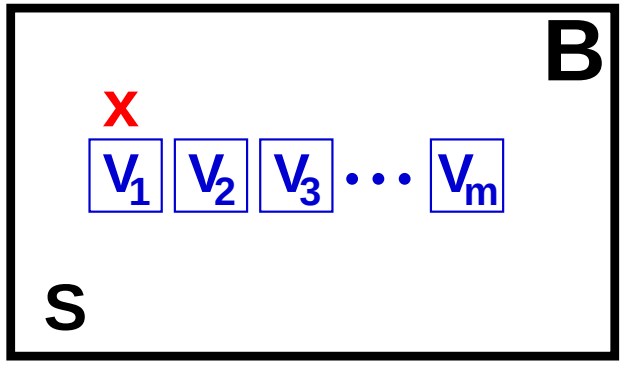
The generic model of computation

Generic Algorithms:

- Introduced by Shoup in ’97
- Representation independent
- Used to prove lower bounds
  (DL, CDH, DDH, etc)

The generic model of computation

The model [Maurer05]



Generic Algorithms:

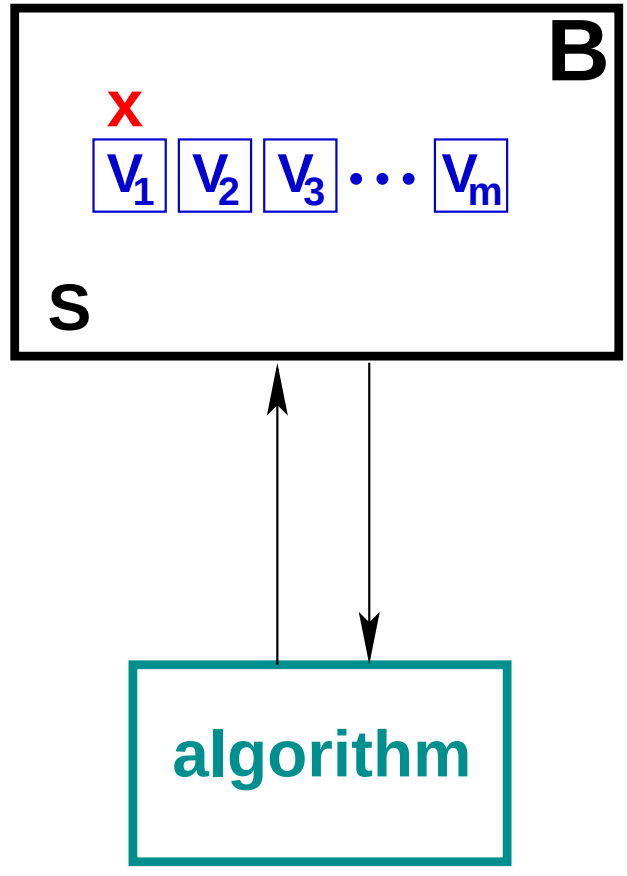
- Introduced by Shoup in ’97
- Representation independent
- Used to prove lower bounds (DL, CDH, DDH, etc)

The generic model of computation

Generic Algorithms:

- Introduced by Shoup in ’97
- Representation independent
- Used to prove lower bounds (DL, CDH, DDH, etc)

The model [Maurer05]

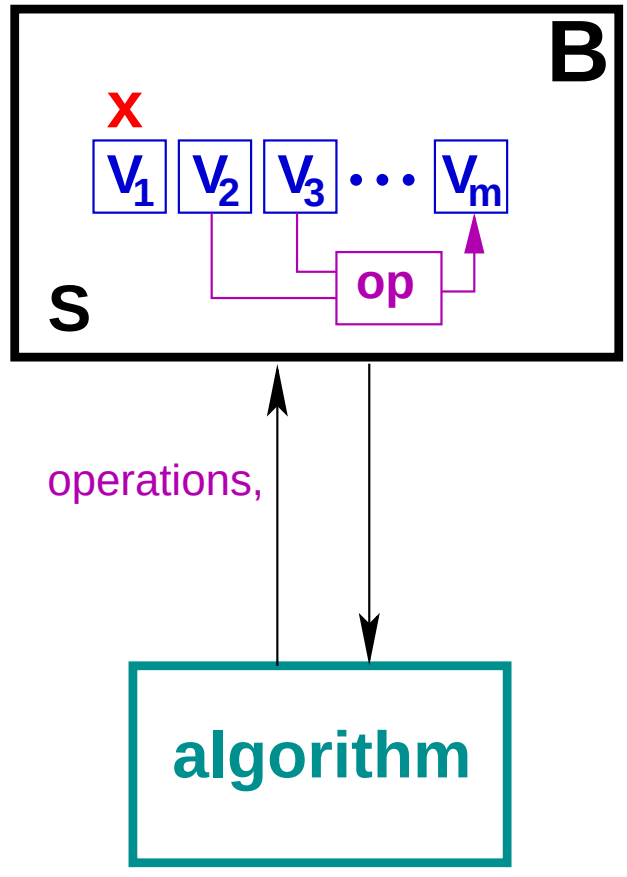


The generic model of computation

Generic Algorithms:

- Introduced by Shoup in ’97
- Representation independent
- Used to prove lower bounds (DL, CDH, DDH, etc)

The model [Maurer05]

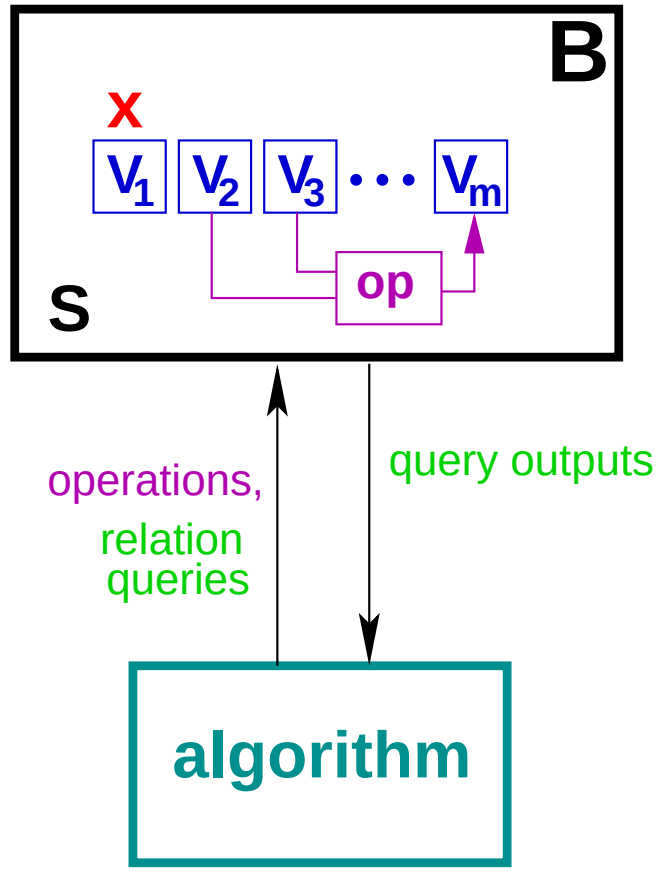


The generic model of computation

Generic Algorithms:

- Introduced by Shoup in ’97
- Representation independent
- Used to prove lower bounds (DL, CDH, DDH, etc)

The model [Maurer05]

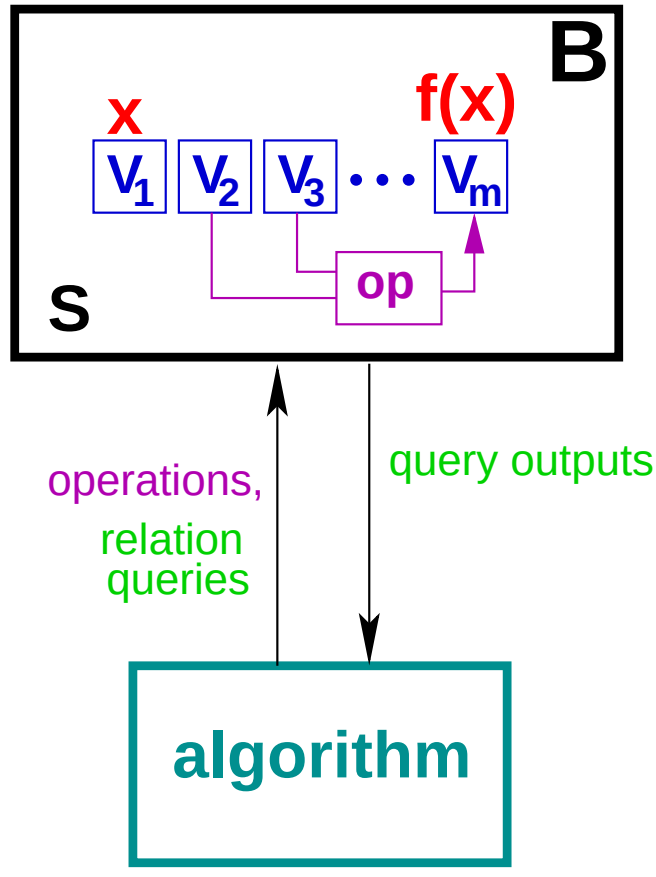


The generic model of computation

Generic Algorithms:

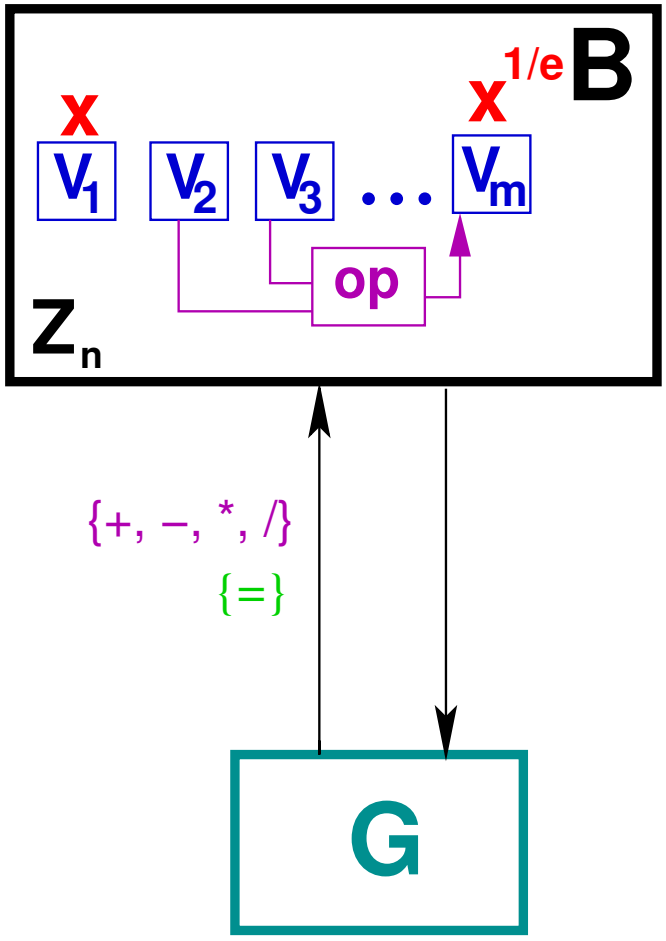
- Introduced by Shoup in ’97
- Representation independent
- Used to prove lower bounds (DL, CDH, DDH, etc)

The model [Maurer05]



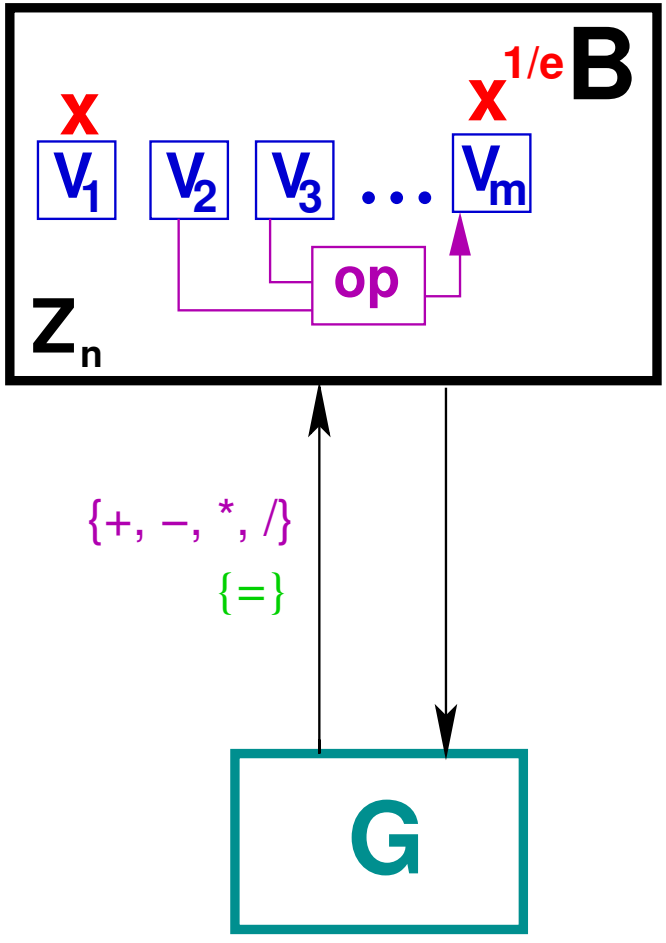
Generic ring algorithms(GRAs) to compute e-th roots

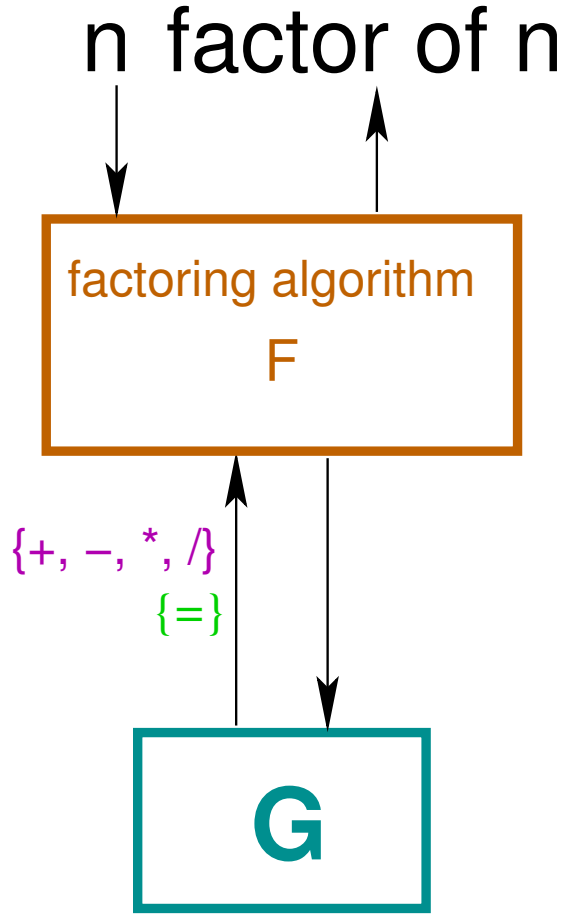
- Only allowed $\{+, -, \cdot, /\}$, $\{=\}$
- Task of G: Compute $x^{1/e}$ modulo n.



The main result

$G : x \mapsto x^{1/e} \quad \Longrightarrow \quad F^G : n \mapsto$ factor of n





Comparison to previous results

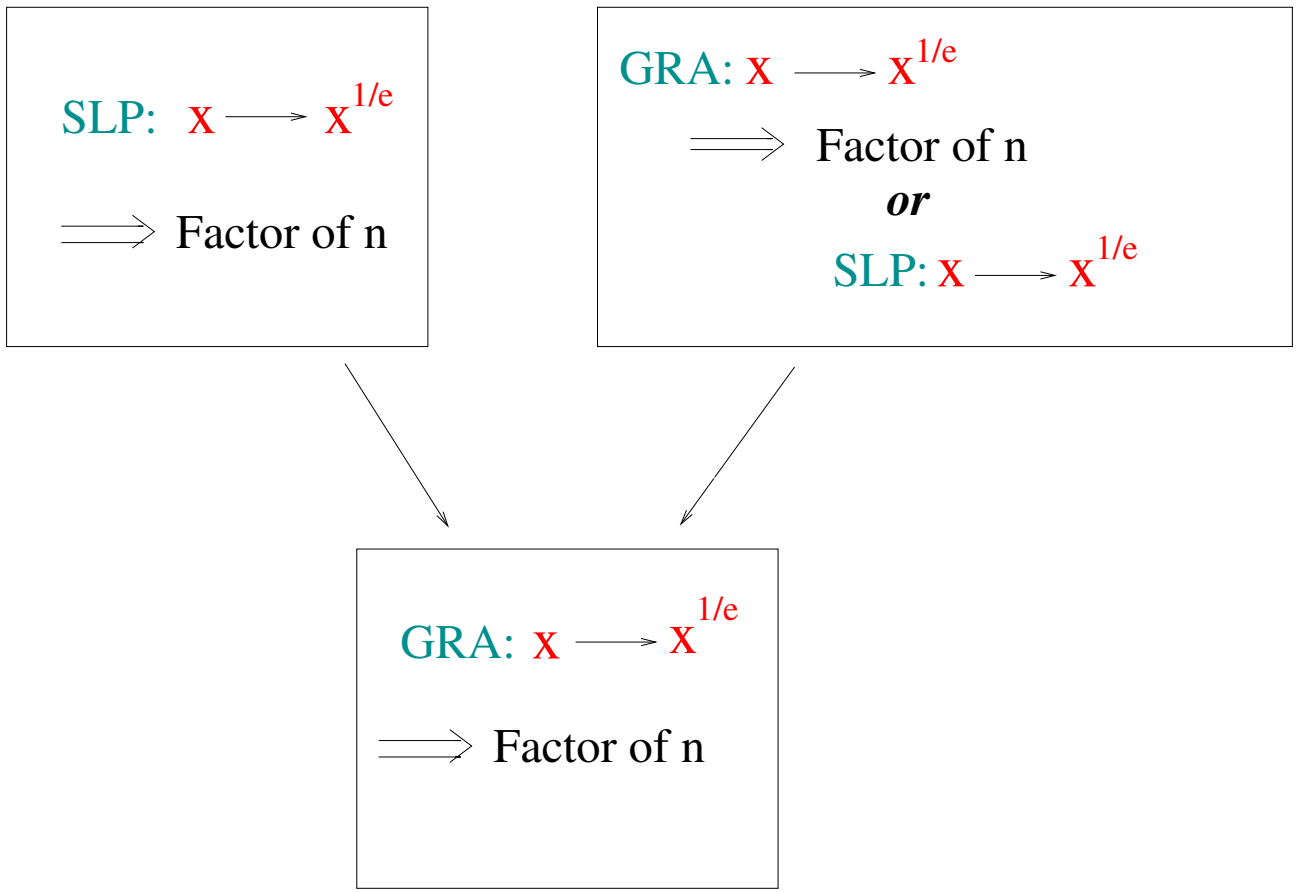
	operations	queries	GRAs	e	Result
DK02	$\{\cdot\}$	$\{=\}$	-	any	e-th root hard
Brown06	$\{+,-,\cdot\}$	$\{\}$	deterministic	small	e-th root $\Rightarrow$ Factoring
LR06	$\{+,-,\cdot\}$	$\{=\}$	deterministic	small	e-th root $\Rightarrow$ Factoring
This paper	$\{+,-,\cdot,/\}$	$\{=\}$	**randomized**	**any**	e-th root $\Rightarrow$ Factoring

Outline

- Straight line programs (SLPs): $\{+, -, \cdot, /\}$, $\{\}$.
- Generic ring algorithms (GRAs): $\{+, -, \cdot, /\}$, $\{=\}$.

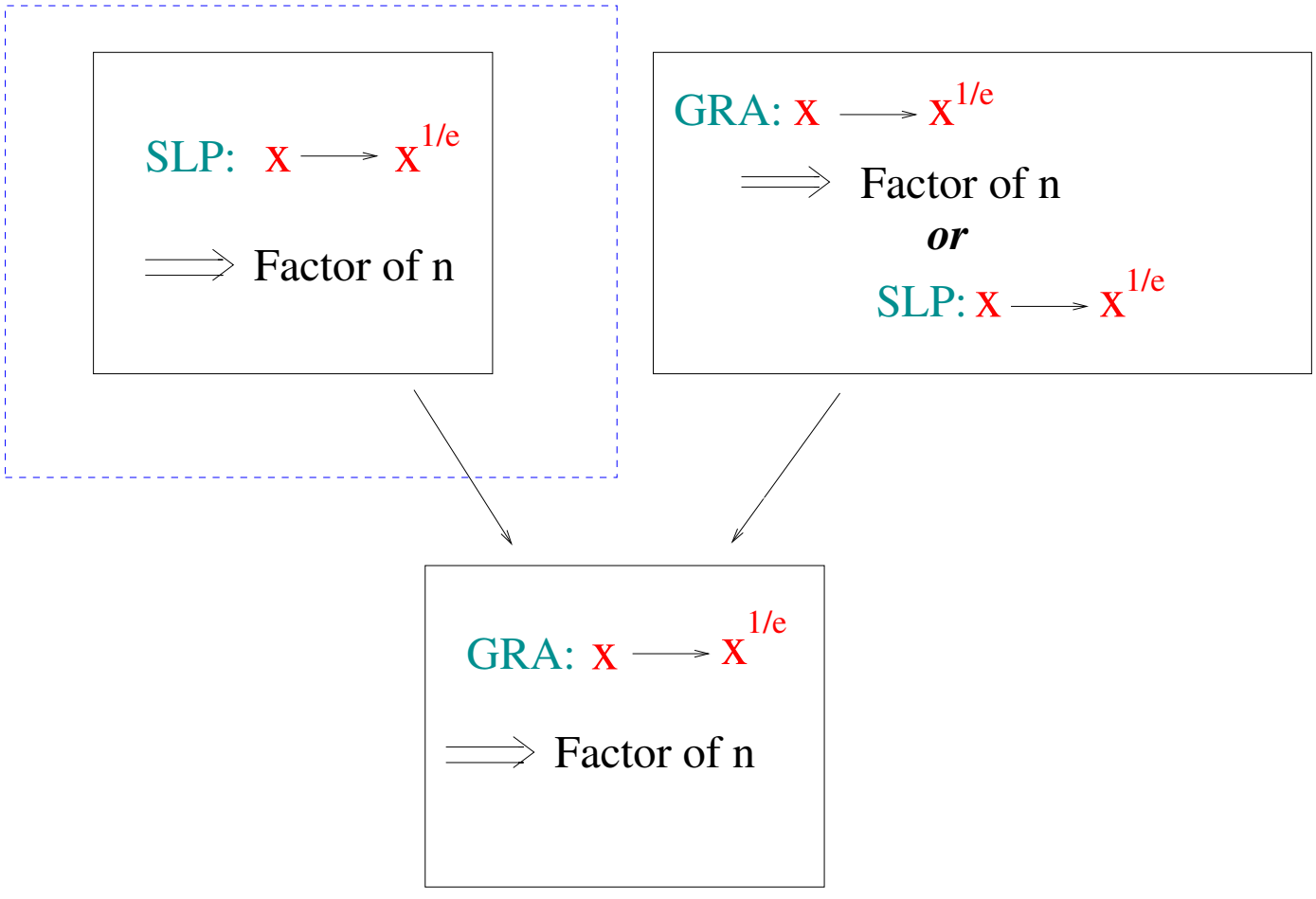
Outline

- Straight line programs (SLPs): $\{+, -, \cdot, /\}$, $\{\}$.
- Generic ring algorithms (GRAs): $\{+, -, \cdot, /\}$, $\{=\}$.



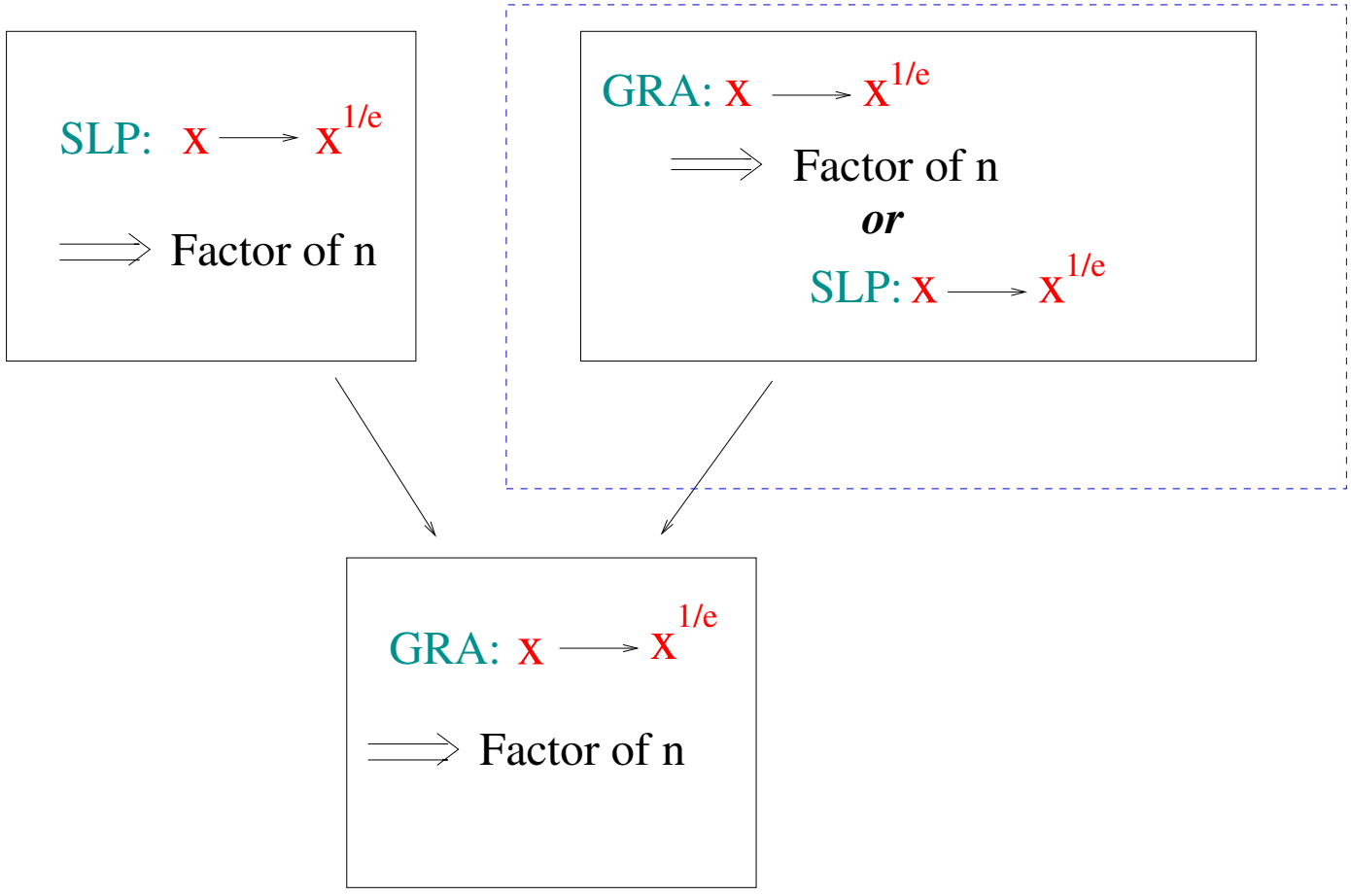
Outline

- Straight line programs (SLPs): $\{+, -, \cdot, /\}$, $\{\}$.
- Generic ring algorithms (GRAs): $\{+, -, \cdot, /\}$, $\{=\}$.



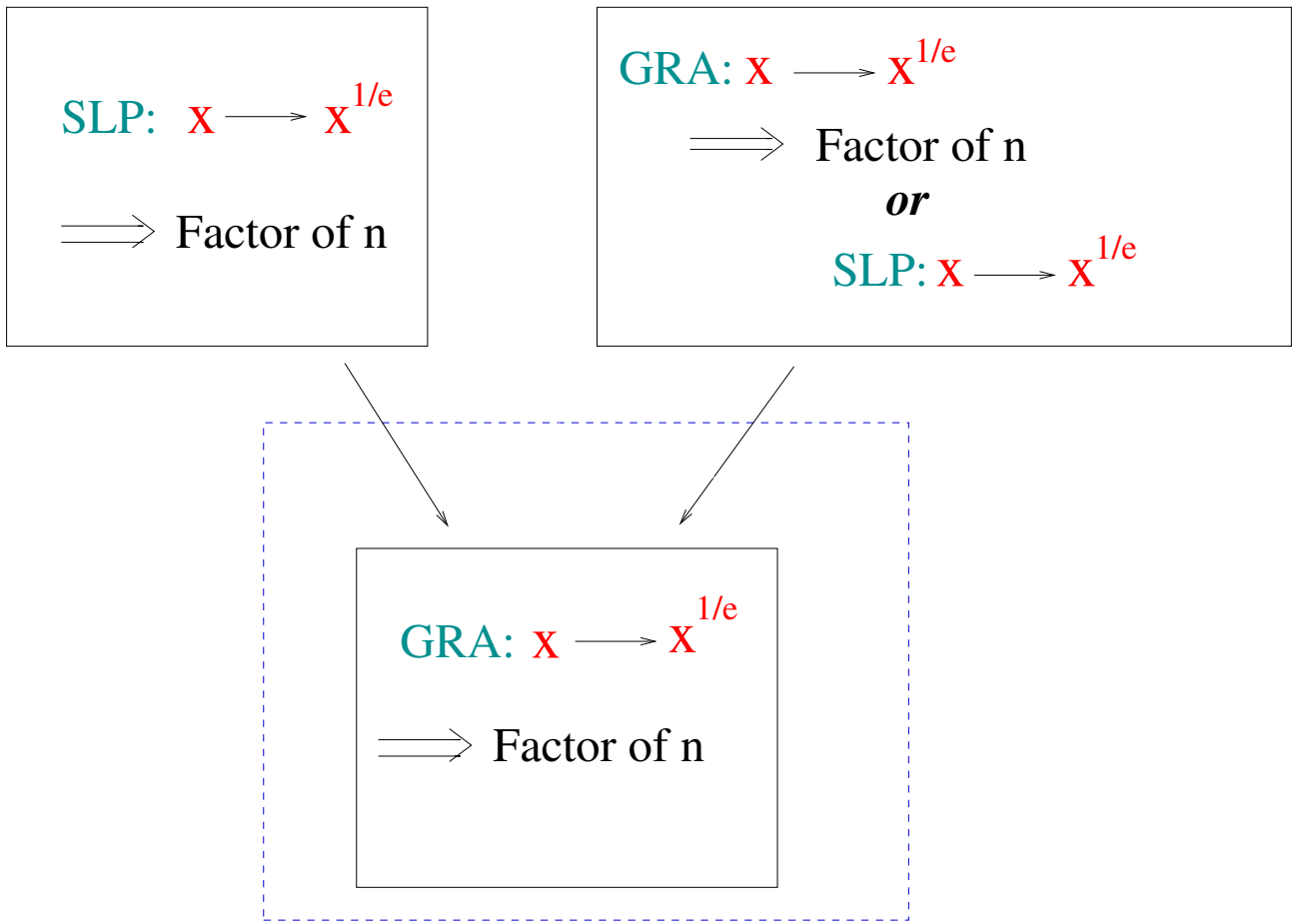
Outline

- Straight line programs (SLPs): $\{+, -, \cdot, /\}$, $\{\}$.
- Generic ring algorithms (GRAs): $\{+, -, \cdot, /\}$, $\{=\}$.



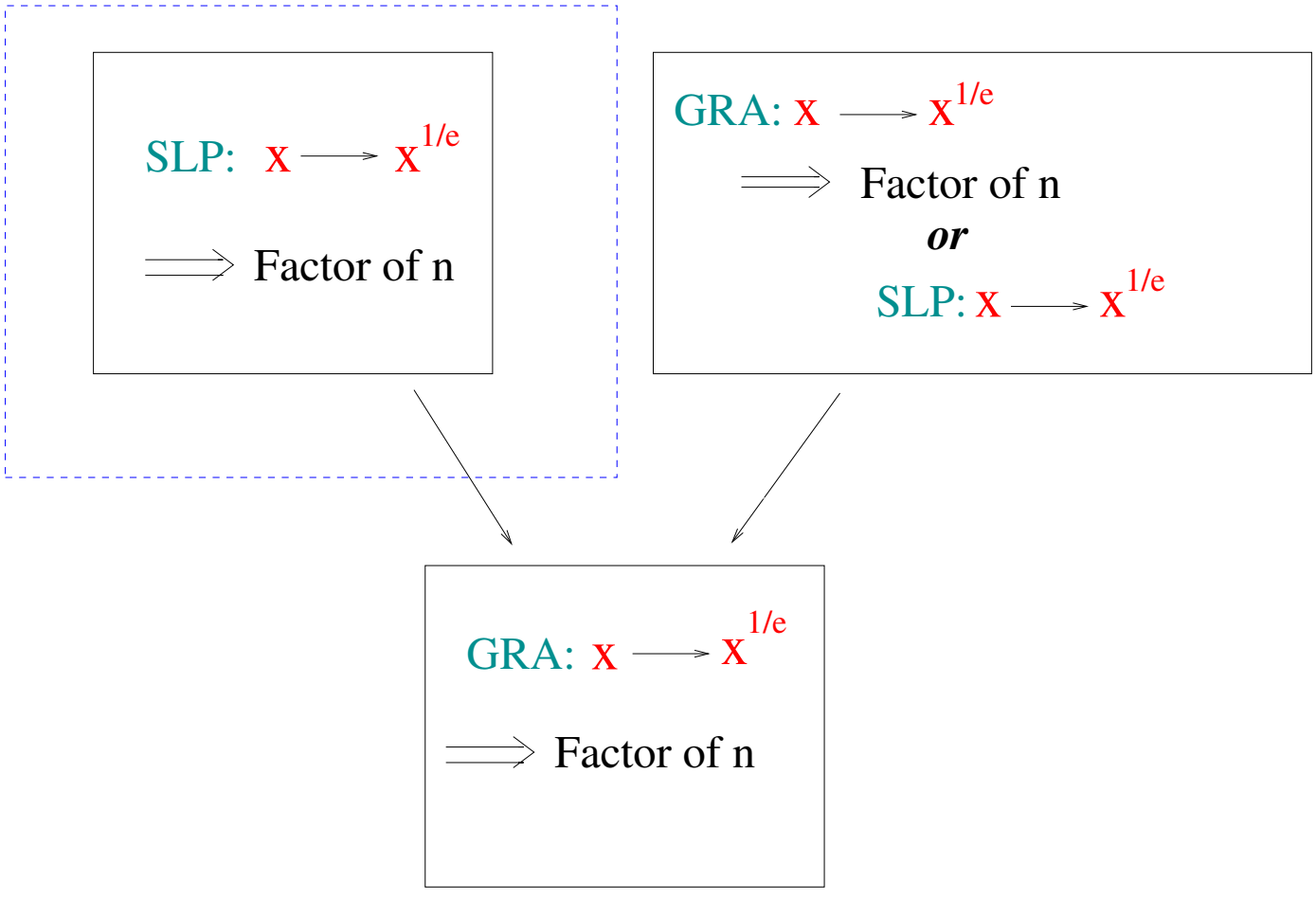
Outline

- Straight line programs (SLPs): $\{+, -, \cdot, /\}$, $\{\}$.
- Generic ring algorithms (GRAs): $\{+, -, \cdot, /\}$, $\{=\}$.

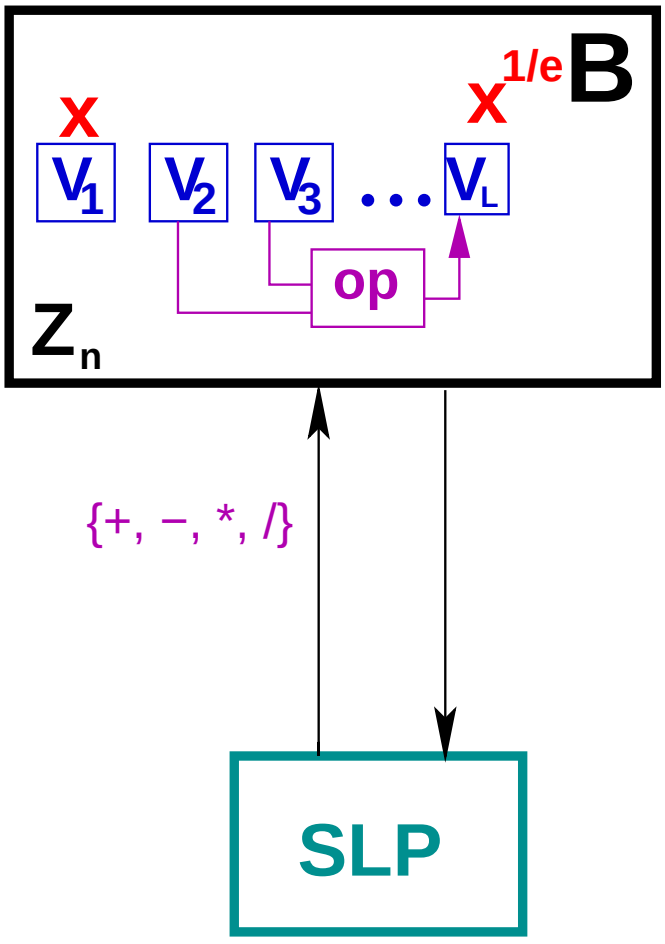


Outline

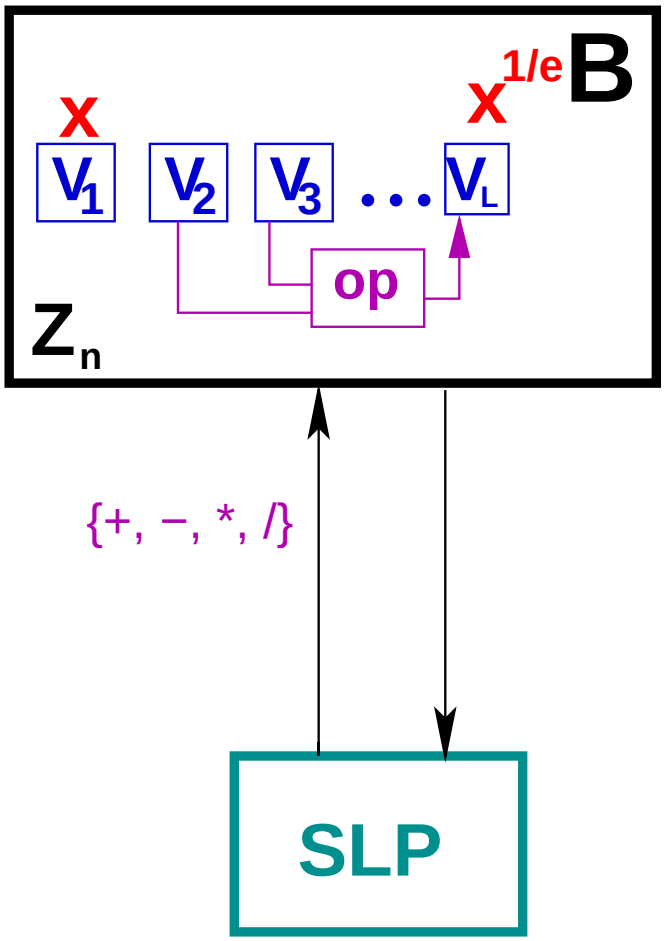
- Straight line programs (SLPs): $\{+, -, \cdot, /\}$, $\{\}$.
- Generic ring algorithms (GRAs): $\{+, -, \cdot, /\}$, $\{=\}$.



Straight line programs



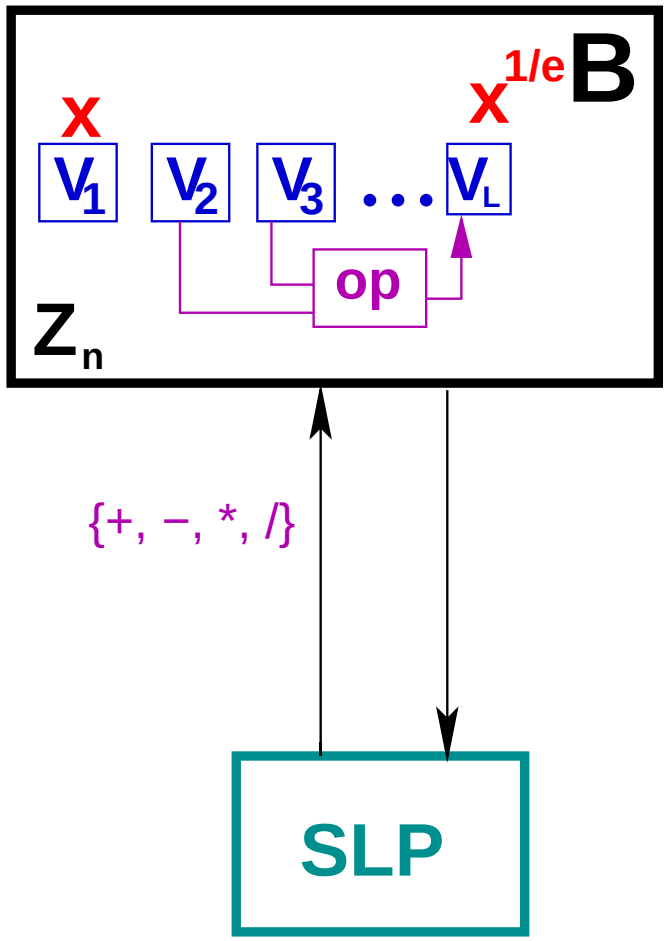
Straight line programs



SLP of length L: Sequence of triples $(i_2, j_2, \circ_2), \ldots, (i_L, j_L, \circ_L)$

- $\circ_k \in \{+, -, \cdot, /\}$.

Straight line programs



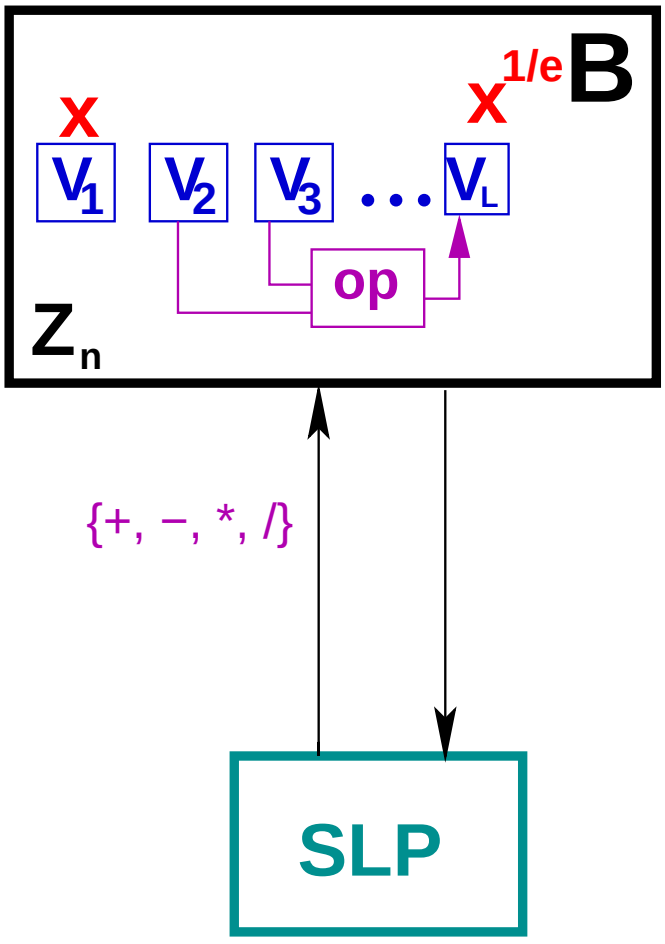
SLP of length L: Sequence of triples $(i_2, j_2, \circ_2), \ldots, (i_L, j_L, \circ_L)$

- $\circ_k \in \{+, -, \cdot, /\}$.

In **B**

- $V_k = V_{i_k} \circ_k V_{j_k}$

Straight line programs



SLP of length L: Sequence of triples $(i_2, j_2, \circ_2), \ldots, (i_L, j_L, \circ_L)$

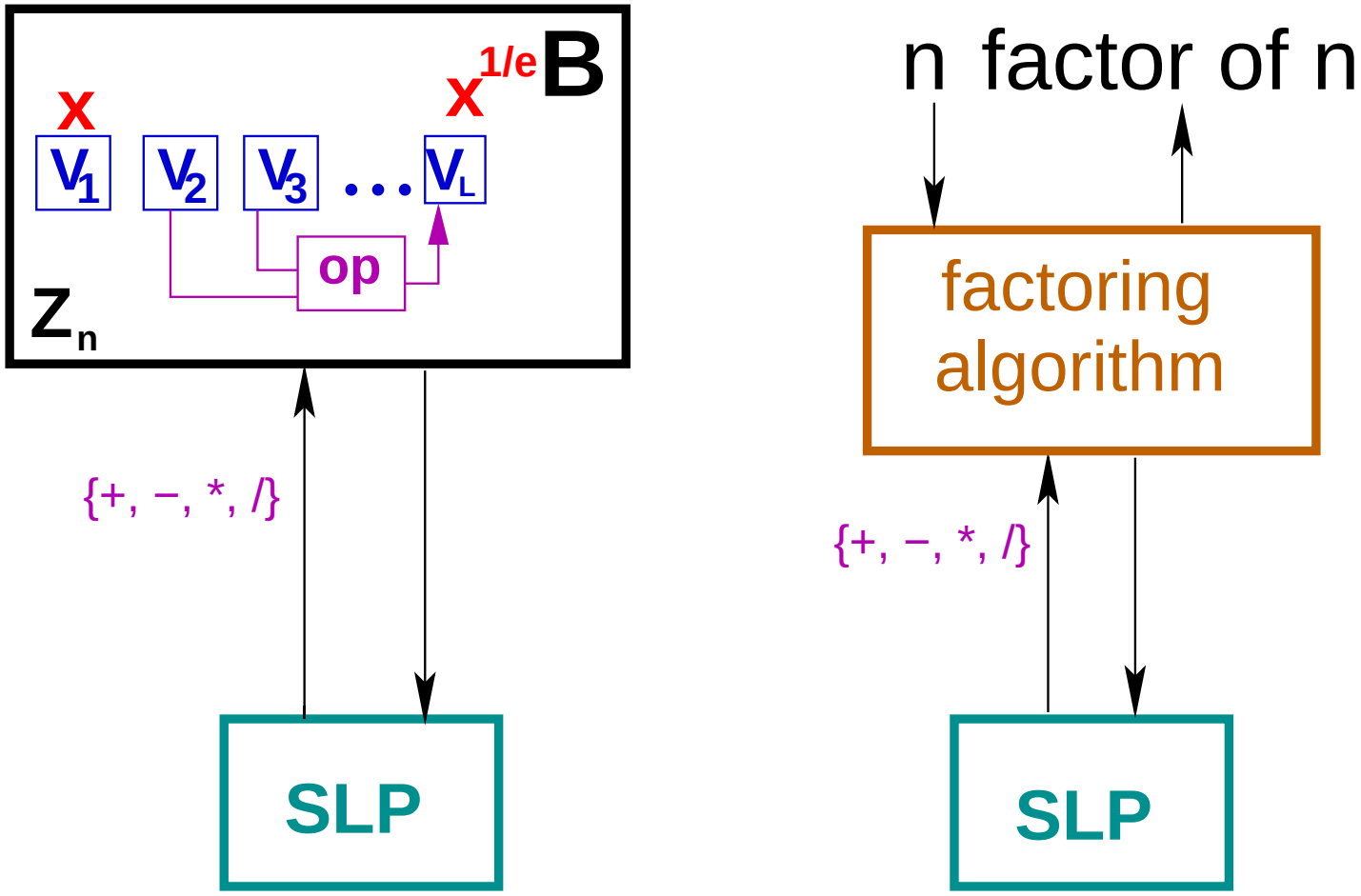
- $\circ_k \in \{+, -, \cdot, /\}$.

In **B**

- $V_k = V_{i_k} \circ_k V_{j_k} = \frac{P_k(x)}{Q_k(x)}$.

SLP for computing e-th roots $\Rightarrow$ Factoring

$SLP : x \mapsto x^{1/e} \quad \Longrightarrow \quad F_1{}^{SLP} : n \mapsto$ factor of n



More formally...

Lemma

$\exists$ *efficient* F_1, *given an L-step SLP* $S : x \mapsto \frac{P(x)}{Q(x)}$ *s.t.,*

$$\Pr_{x \in_R \mathbb{Z}_n} \left(\left(\frac{P(x)}{Q(x)} \right)^e \equiv_n x \right) \geq \mu$$

$\Rightarrow {F_1}^S : n \mapsto$ *factor of* n

More formally...

Lemma

$\exists$ *efficient* F_1, *given an L-step SLP* $S : x \mapsto \frac{P(x)}{Q(x)}$ *s.t.,*

$$\Pr_{x \in_R \mathbb{Z}_n} \left(\left(\frac{P(x)}{Q(x)} \right)^e \equiv_n x \right) \geq \mu$$

$\Rightarrow {F_1}^S : n \mapsto$ *factor of* n

Proof: Note that,

$$\left(\frac{P(x)}{Q(x)} \right)^e \equiv_n x \quad \Longrightarrow \quad P(x)^e - x \cdot Q(x)^e \equiv_n 0 \quad \wedge \quad Q(x) \neq 0$$

More formally...

Lemma

$\exists$ *efficient* F_1, *given an L-step SLP* $S : x \mapsto \frac{P(x)}{Q(x)}$ *s.t.,*

$$\Pr_{x \in_R \mathbb{Z}_n} \left(\left(\frac{P(x)}{Q(x)} \right)^e \equiv_n x \right) \geq \mu$$

$\Rightarrow {F_1}^S : n \mapsto$ *factor of* n

Proof: Note that,

$$\left(\frac{P(x)}{Q(x)} \right)^e \equiv_n x \quad \Longrightarrow \quad P(x)^e - x \cdot Q(x)^e \equiv_n 0 \quad \wedge \quad Q(x) \neq 0$$

A few facts:

- $\deg(P(x)^e - x \cdot Q(x)^e) \quad \leq \quad e \cdot 2^L + 1$
- $\exists$ $(4L + 4 \log e + 2)$-step SLP

$$S_1 \ : \ x \mapsto P(x)^e - x \cdot Q(x)^e.$$

Factoring algorithm

Let $f(x) = P(x)^e - x \cdot Q(x)^e$ $\qquad$ $S_1 \ : \ x \mapsto f(x)$.

The Algorithm

1. Choose random $h(x) \in \mathbb{Z}_n[x]$ of degree $d = L + log(e)$.
2. Choose random $r(x) \in \mathbb{Z}_n[x]/h(x)$.
3. Compute $z(x) = f(r(x))$ in $\mathbb{Z}_n[x]/h(x)$.
4. (i) Run Euclid$\big(h(x), z(x)\big)$ in $\mathbb{Z}_n[x]$.
   (ii) Return the factor of n if the Euclid's algorithm fails.

Factoring algorithm

Let $f(x) = P(x)^e - x \cdot Q(x)^e$ $\qquad$ $S_1 \ : \ x \mapsto f(x)$.

The Algorithm

1. Choose random $h(x) \in \mathbb{Z}_n[x]$ of degree $d = L + log(e)$.
2. Choose random $r(x) \in \mathbb{Z}_n[x]/h(x)$.
3. Compute $z(x) = f(r(x))$ in $\mathbb{Z}_n[x]/h(x)$.
4. (i) Run Euclid$(h(x), z(x))$ in $\mathbb{Z}_n[x]$.
   (ii) Return the factor of n if the Euclid's algorithm fails.

Proof Sketch

$\Pr\left(h(x) \text{ has a root } s \text{ in } \mathbb{Z}_p[x] \ \wedge \ \text{ is irreducible in } \mathbb{Z}_q[x]\right) \geq \frac{1}{4d}$.

Factoring algorithm

Let $f(x) = P(x)^e - x \cdot Q(x)^e$ $\qquad S_1 \ : \ x \mapsto f(x).$

The Algorithm

1. Choose random $h(x) \in \mathbb{Z}_n[x]$ of degree $d = L + log(e)$.
2. Choose random $r(x) \in \mathbb{Z}_n[x]/h(x)$.
3. Compute $z(x) = f(r(x))$ in $\mathbb{Z}_n[x]/h(x)$.
4. (i) Run Euclid$\big(h(x), z(x)\big)$ in $\mathbb{Z}_n[x]$.
   (ii) Return the factor of n if the Euclid's algorithm fails.

Proof Sketch

$\Pr\Big(h(x)$ has a root s in $\mathbb{Z}_p[x] \ \wedge \ $ is irreducible in $\mathbb{Z}_q[x]\Big) \geq \frac{1}{4d}$.

$\mathbb{Z}_n[x]/h(x) \ \cong \ \mathbb{Z}_p[x]/(x-s) \ \times \ \mathbb{Z}_p[x]/h_1(x) \ \times \ \mathbb{F}_{q^d}$

Factoring algorithm

Let $f(x) = P(x)^e - x \cdot Q(x)^e$ $\qquad S_1 \ : \ x \mapsto f(x).$

The Algorithm

1. Choose random $h(x) \in \mathbb{Z}_n[x]$ of degree $d = L + log(e)$.
2. Choose random $r(x) \in \mathbb{Z}_n[x]/h(x)$.
3. Compute $z(x) = f(r(x))$ in $\mathbb{Z}_n[x]/h(x)$.
4. (i) Run Euclid$\big(h(x), z(x)\big)$ in $\mathbb{Z}_n[x]$.
   (ii) Return the factor of n if the Euclid's algorithm fails.

Proof Sketch

$\Pr\Big(h(x)$ has a root s in $\mathbb{Z}_p[x] \ \wedge \ $ is irreducible in $\mathbb{Z}_q[x]\Big) \geq \frac{1}{4d}$.

$\mathbb{Z}_n[x]/h(x) \ \cong \ \mathbb{Z}_p[x]/(x-s) \ \times \ \mathbb{Z}_p[x]/h_1(x) \ \times \ \mathbb{F}_{q^d}$

- $\Pr\left(z(s) \equiv_p 0\right) \ \geq \ \mu$.
- $\Pr(z(x) \neq_q 0) \ \geq \ \frac{1}{2}$.

Factoring algorithm

Let $f(x) = P(x)^e - x \cdot Q(x)^e$ $\qquad S_1 \ : \ x \mapsto f(x).$

The Algorithm

1. Choose random $h(x) \in \mathbb{Z}_n[x]$ of degree $d = L + log(e)$.
2. Choose random $r(x) \in \mathbb{Z}_n[x]/h(x)$.
3. Compute $z(x) = f(r(x))$ in $\mathbb{Z}_n[x]/h(x)$.
4. (i) Run Euclid$\big(h(x), z(x)\big)$ in $\mathbb{Z}_n[x]$.
   (ii) Return the factor of n if the Euclid's algorithm fails.

P

P

$\mathbb{Z}$

$\bar{d}$.

Observation

- On a non-Euclidean domain,
  Euclid's algorithm fails $\Rightarrow$ it gives a zero-divisor.
- $\deg\Big(gcd_p\big(h(x), z(x)\big)\Big) \neq \deg\Big(gcd_q\big(h(x), z(x)\big)\Big)$
  $\Rightarrow$ Euclid$\big(h(x), z(x)\big)$ fails.

Factoring algorithm

Let $f(x) = P(x)^e - x \cdot Q(x)^e$ $\qquad S_1 \ : \ x \mapsto f(x).$

The Algorithm

1. Choose random $h(x) \in \mathbb{Z}_n[x]$ of degree $d = L + log(e)$.
2. Choose random $r(x) \in \mathbb{Z}_n[x]/h(x)$.
3. Compute $z(x) = f(r(x))$ in $\mathbb{Z}_n[x]/h(x)$.
4. (i) Run Euclid$\big(h(x), z(x)\big)$ in $\mathbb{Z}_n[x]$.
   (ii) Return the factor of n if the Euclid's algorithm fails.

Proof Sketch

$\Pr\Big(h(x)$ has a root s in $\mathbb{Z}_p[x] \ \wedge \ $ is irreducible in $\mathbb{Z}_q[x]\Big) \geq \frac{1}{4d}$.

$\mathbb{Z}_n[x]/h(x) \ \cong \ \mathbb{Z}_p[x]/(x-s) \ \times \ \mathbb{Z}_p[x]/h_1(x) \ \times \ \mathbb{F}_{q^d}$

- $\Pr\left(z(s) \equiv_p 0\right) \ \geq \ \mu$.
- $\Pr(z(x) \neq_q 0) \ \geq \ \frac{1}{2}$.

Factoring algorithm

Let $f(x) = P(x)^e - x \cdot Q(x)^e$ $\qquad S_1 \ : \ x \mapsto f(x).$

The Algorithm

1. Choose random $h(x) \in \mathbb{Z}_n[x]$ of degree $d = L + log(e)$.
2. Choose random $r(x) \in \mathbb{Z}_n[x]/h(x)$.
3. Compute $z(x) = f(r(x))$ in $\mathbb{Z}_n[x]/h(x)$.
4. (i) Run Euclid$\big(h(x), z(x)\big)$ in $\mathbb{Z}_n[x]$.
   (ii) Return the factor of n if the Euclid's algorithm fails.

Proof Sketch

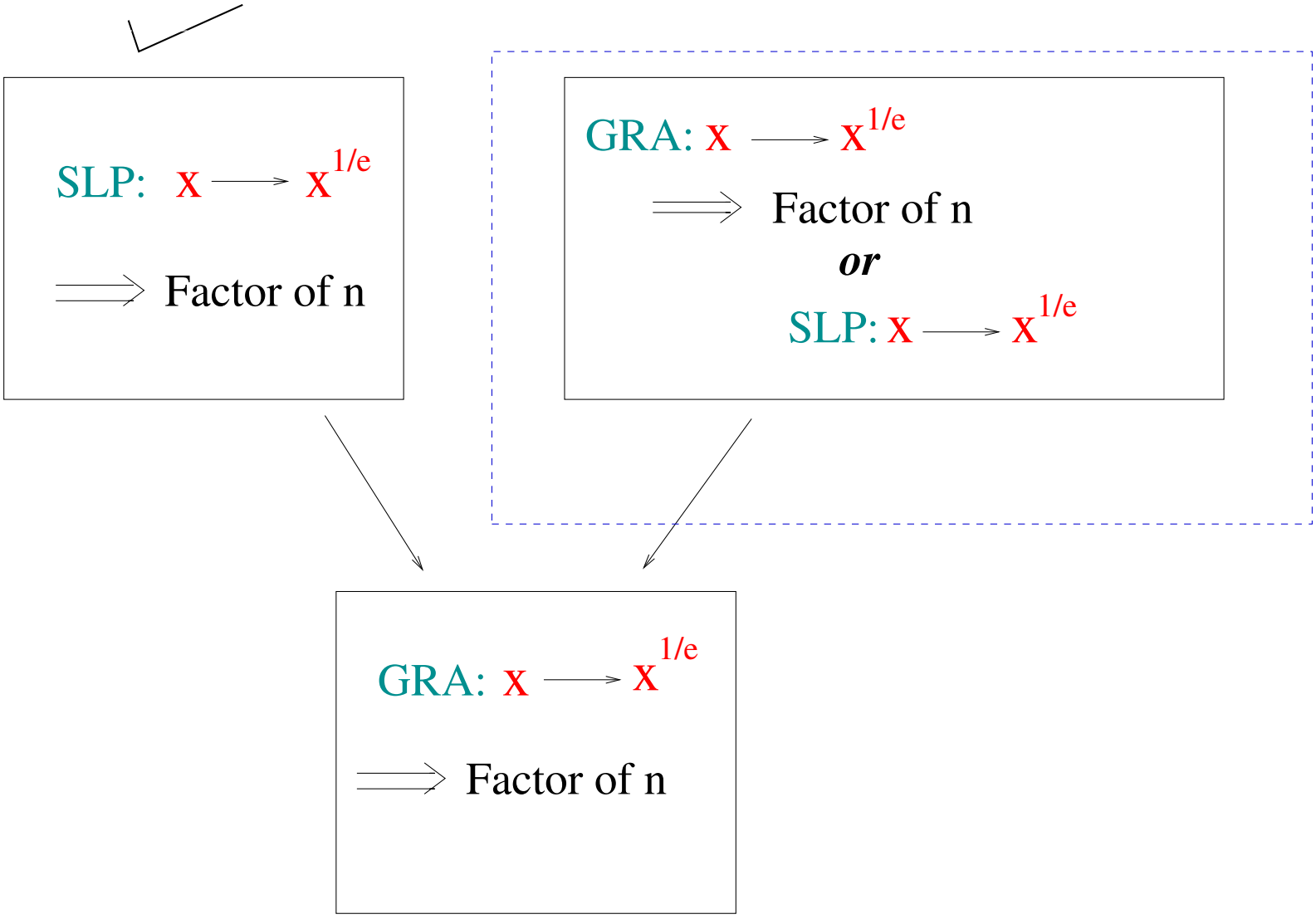
$\Pr\Big(h(x)$ has a root s in $\mathbb{Z}_p[x] \ \wedge \ $ is irreducible in $\mathbb{Z}_q[x]\Big) \geq \frac{1}{4d}$.

$\mathbb{Z}_n[x]/h(x) \ \cong \ \mathbb{Z}_p[x]/(x-s) \ \times \ \mathbb{Z}_p[x]/h_1(x) \ \times \ \mathbb{F}_{q^d}$

- $\Pr\left(z(s) \equiv_p 0\right) \ \geq \ \mu.$
- $\Pr(z(x) \neq_q 0) \ \geq \ \frac{1}{2}.$

$$\Longrightarrow \Pr\Big((x-s) \mid gcd_p\big(h(x), z(x)\big) \ \wedge \ gcd_q\big(h(x), z(x)\big) = 1\Big)$$
$$\geq \frac{1}{4d} \cdot \mu \cdot \frac{1}{2} = \frac{\mu}{8(L + \log e)}$$

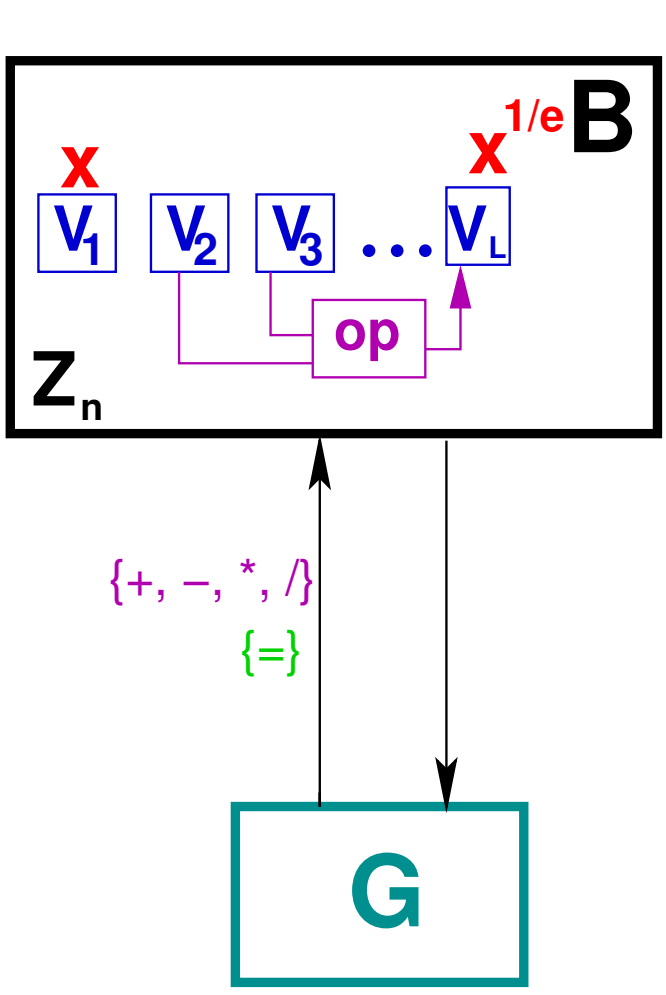
Outline

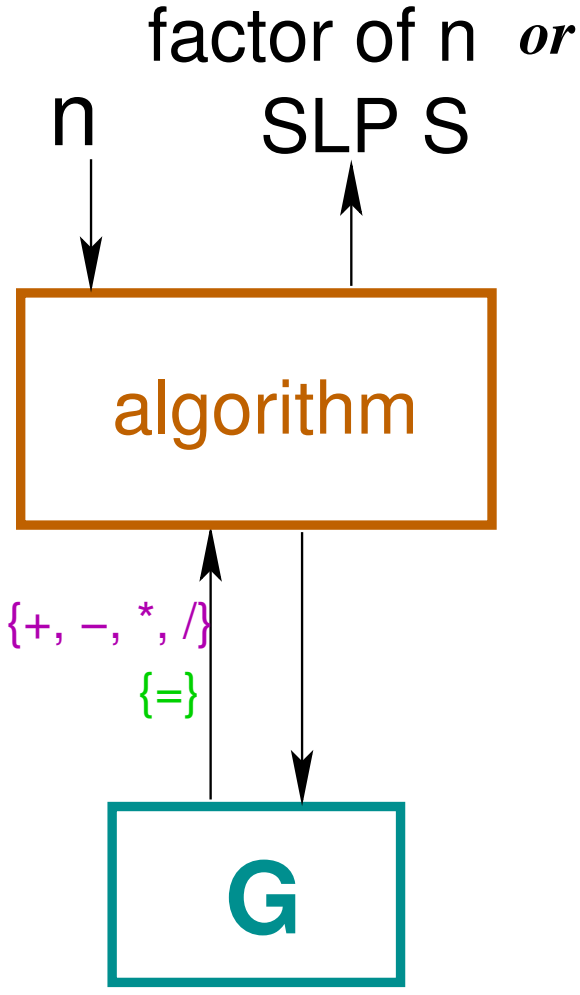

SLP: $x \longrightarrow x^{1/e}$
$\Longrightarrow$ Factor of n
GRA: $x \longrightarrow x^{1/e}$
$\Longrightarrow$ Factor of n
or
SLP: $x \longrightarrow x^{1/e}$
GRA: $x \longrightarrow x^{1/e}$
$\Longrightarrow$ Factor of n

From SLPs to deterministic GRAs

Lemma

$G : x \mapsto x^{1/e} \implies A^G : n \mapsto$ *factor of n or S*





A key lemma

$f(x) \in \mathbb{Z}_n[x]$ with "not too many" and "not too few" roots

$\Longrightarrow$ Factor of n.

A key lemma

$f(x) \in \mathbb{Z}_n[x]$ with "not too many" and "not too few" roots

$\Longrightarrow$ Factor of n.

A few definitions:

- $$\nu_n(f) = \frac{|\{x \in \mathbb{Z}_n \ : \ f(x) \equiv_n 0\}|}{n} \ .$$
- $$\eta_n(f) = \frac{|\{x \in \mathbb{Z}_n \ : \ \gcd(f(x), n) \notin \{1, n\}\}|}{n} \ .$$

A key lemma

$f(x) \in \mathbb{Z}_n[x]$ with "not too many" and "not too few" roots

$\Longrightarrow$ Factor of n.

A few definitions:

- $$\nu_n(f) = \frac{|\{x \in \mathbb{Z}_n \ : \ f(x) \equiv_n 0\}|}{n} \ .$$
- $$\eta_n(f) = \frac{|\{x \in \mathbb{Z}_n \ : \ \gcd(f(x), n) \notin \{1, n\}\}|}{n} \ .$$

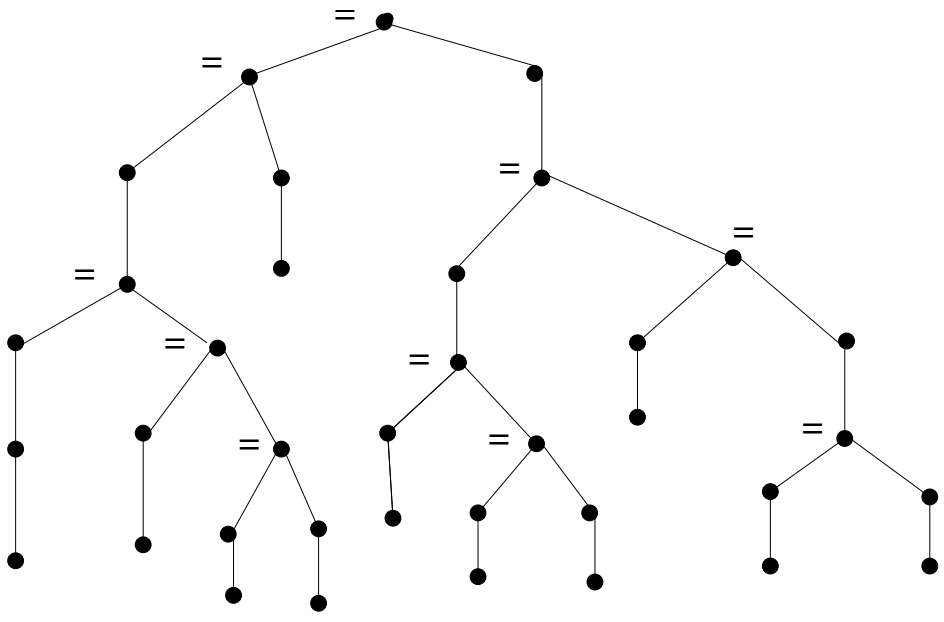
Lemma

$\forall \ f(x) \in \mathbb{Z}_n[x], \qquad \delta \leq \nu_n(f) \leq 1 - \delta \implies \eta_n(f) \geq \delta^{\frac{3}{2}}.$

From SLPs to deterministic GRAs: A proof sketch

GRA can be seen as a binary tree!

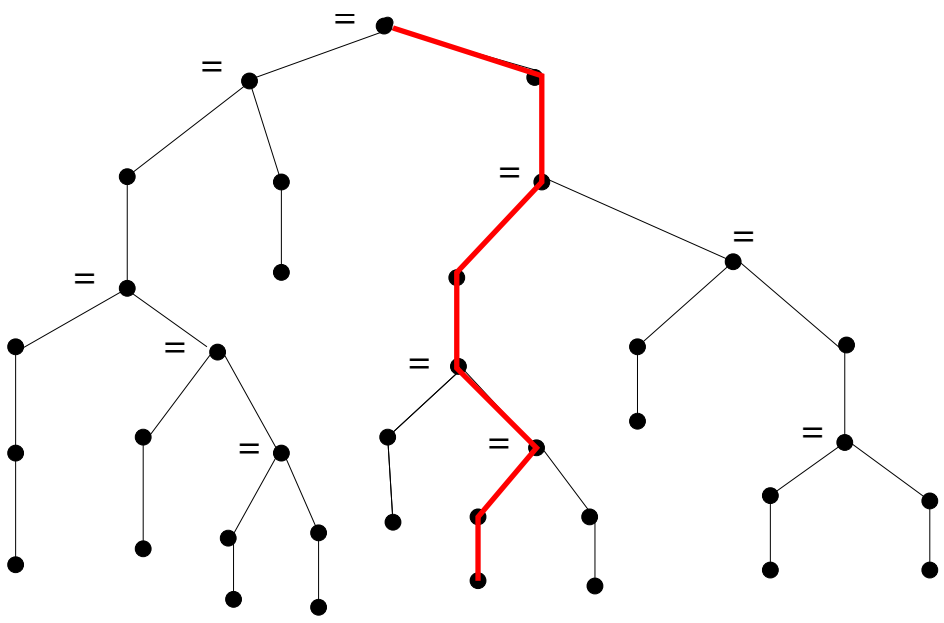
Every path from the root to a leaf corresponds to an SLP.



From SLPs to deterministic GRAs: A proof sketch

GRA can be seen as a binary tree!

Every path from the root to a leaf corresponds to an SLP.

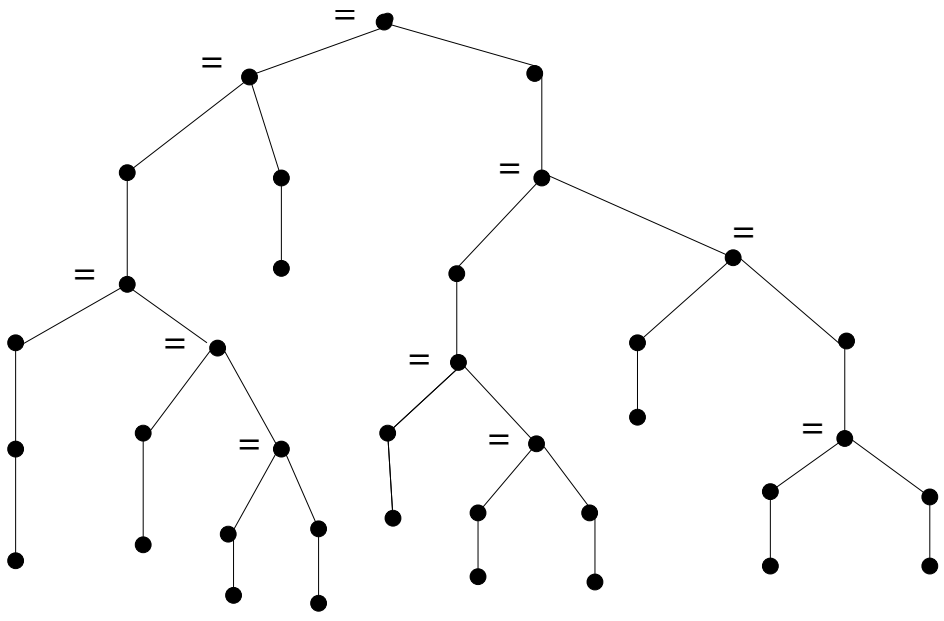


From SLPs to deterministic GRAs: A proof sketch

GRA can be seen as a binary tree!

Every path from the root to a leaf corresponds to an SLP.

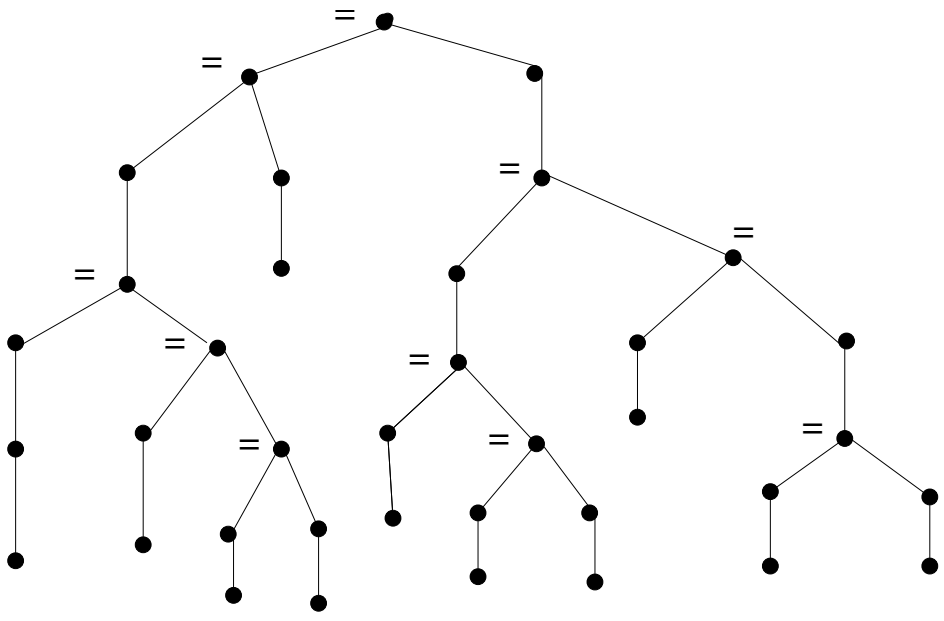
Equality query: $\frac{P_i}{Q_i} \stackrel{?}{=} \frac{P_j}{Q_j}$



From SLPs to deterministic GRAs: A proof sketch

GRA can be seen as a binary tree!

Every path from the root to a leaf corresponds to an SLP.



Equality query: $\frac{P_i}{Q_i} \stackrel{?}{=} \frac{P_j}{Q_j}$

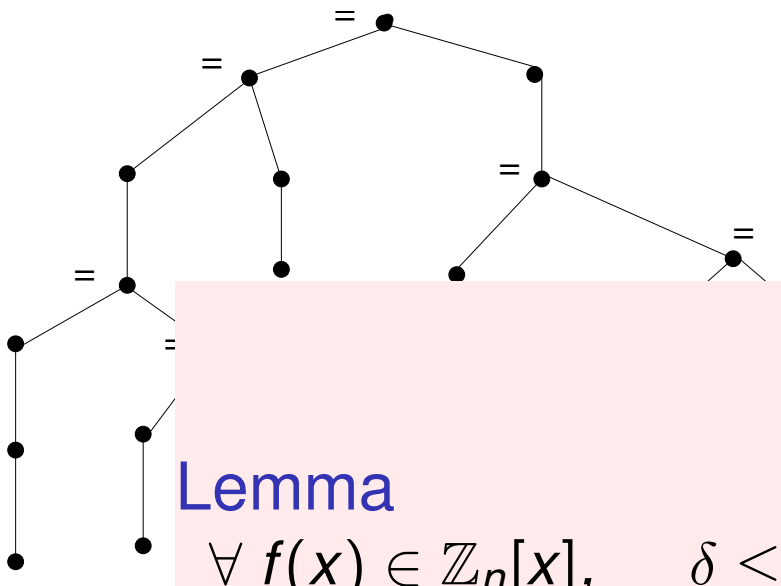
Two cases:

- An equality query with $\delta \leq \nu_n(P_iQ_j - Q_iP_j) \leq 1-\delta$.

From SLPs to deterministic GRAs: A proof sketch

GRA can be seen as a binary tree!

Every path from the root to a leaf corresponds to an SLP.



Equality query: $\frac{P_i}{Q_i} \stackrel{?}{=} \frac{P_j}{Q_j}$

Two cases:

- An equality query with $\delta \leq \nu_n(P_iQ_j - Q_iP_j) \leq 1-\delta$.

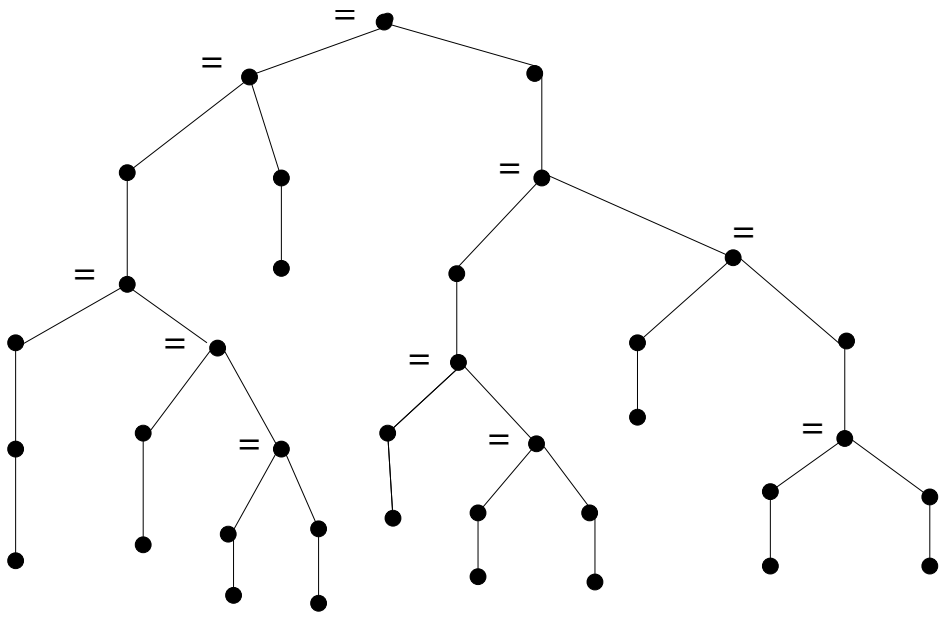
Lemma

$\forall\, f(x) \in \mathbb{Z}_n[x], \quad \delta \leq \nu_n(f) \leq 1-\delta \implies$ *Factor of n.*

From SLPs to deterministic GRAs: A proof sketch

GRA can be seen as a binary tree!

Every path from the root to a leaf corresponds to an SLP.



Equality query: $\frac{P_i}{Q_i} \stackrel{?}{=} \frac{P_j}{Q_j}$

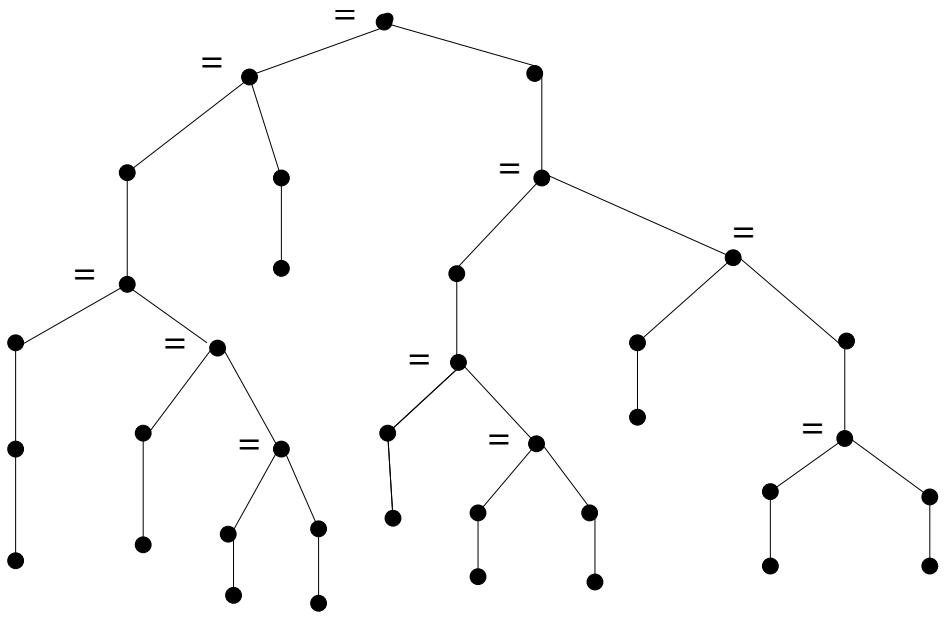
Two cases:

- An equality query with $\delta \leq \nu_n(P_iQ_j - Q_iP_j) \leq 1-\delta$.
  - Can be used to factor n.

From SLPs to deterministic GRAs: A proof sketch

GRA can be seen as a binary tree!

Every path from the root to a leaf corresponds to an SLP.



Equality query: $\frac{P_i}{Q_i} \stackrel{?}{=} \frac{P_j}{Q_j}$

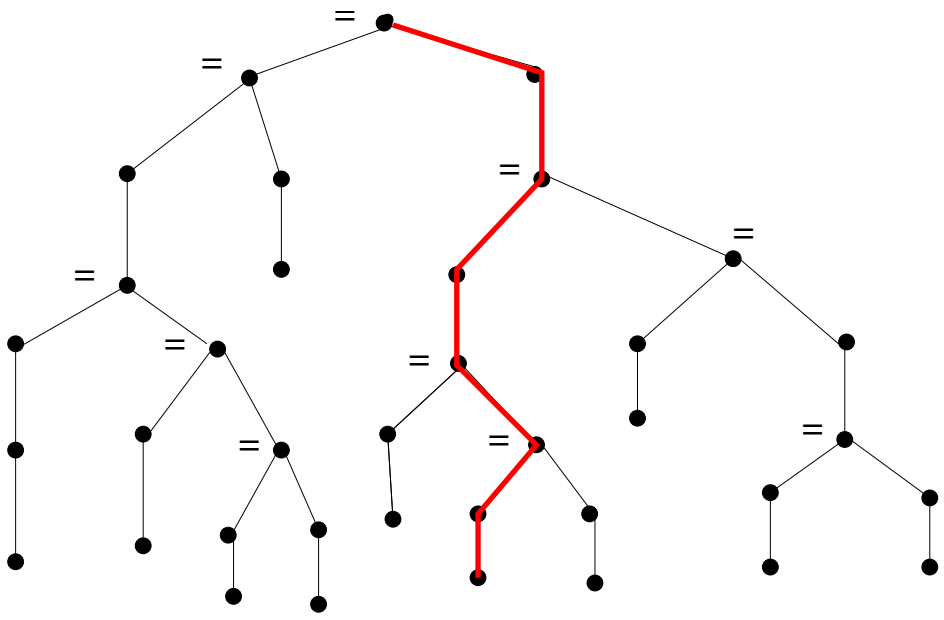
Two cases:

- An equality query with $\delta \leq \nu_n(P_iQ_j - Q_iP_j) \leq 1-\delta$.
  - Can be used to factor n.
- All equality queries have $\nu_n(P_iQ_j - Q_iP_j) > 1-\delta$ or $\nu_n(P_iQ_j - Q_iP_j) < \delta$.
  -

From SLPs to deterministic GRAs: A proof sketch

GRA can be seen as a binary tree!

Every path from the root to a leaf corresponds to an SLP.

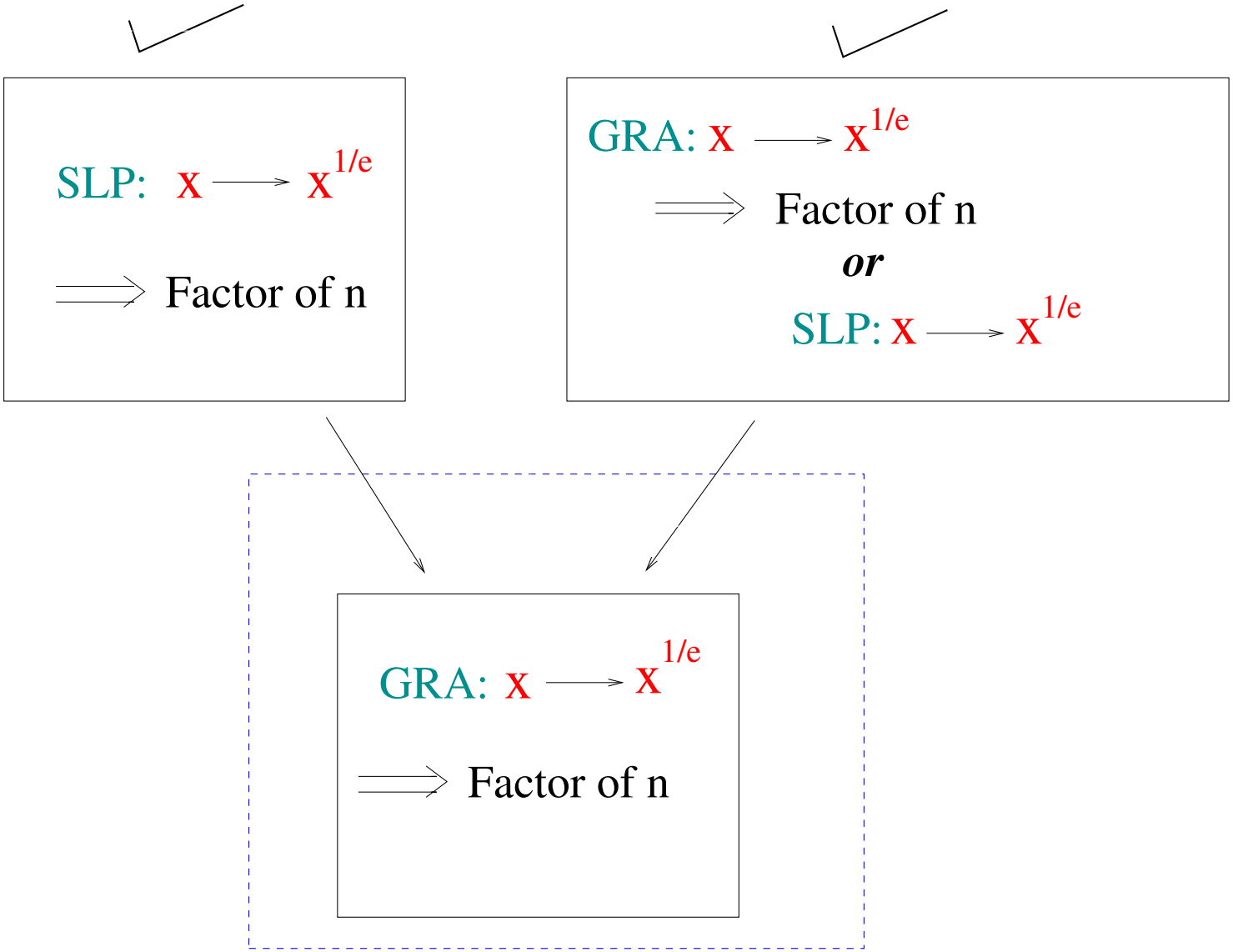


Equality query: $\frac{P_i}{Q_i} \stackrel{?}{=} \frac{P_j}{Q_j}$

Two cases:

- An equality query with $\delta \leq \nu_n(P_iQ_j - Q_iP_j) \leq 1-\delta$.
  - Can be used to factor n.
- All equality queries have $\nu_n(P_iQ_j - Q_iP_j) > 1-\delta$ or $\nu_n(P_iQ_j - Q_iP_j) < \delta$.
  - Gives an SLP.

Outline


SLP: $x \longrightarrow x^{1/e}$
$\Longrightarrow$ Factor of n
GRA: $x \longrightarrow x^{1/e}$
$\Longrightarrow$ Factor of n
or
SLP: $x \longrightarrow x^{1/e}$
GRA: $x \longrightarrow x^{1/e}$
$\Longrightarrow$ Factor of n

Generalize to randomized GRAs

Definition

A randomized GRA $\mathcal{G}$ is a GRA where the choice of the operation at each step is randomized.

Theorem

$\exists\ F$ *s.t.*, $\mathcal{G} : x \mapsto x^{1/e} \implies F^{\mathcal{G}} : n \mapsto$ *factor of* n

Proof: Less trivial than it may appear. Refer paper.

Conclusions

Breaking RSA $\Longrightarrow$ Factor n *or* Exploit bit representation.

Conclusions

Breaking RSA $\Longrightarrow$ Factor n *or* Exploit bit representation.

New evidence: Breaking RSA may be equivalent to factoring.

Thank You