Dual Exponentiation Schemes

Colin D. Walter

Information Security Group
Royal Holloway University of London

Colin.Walter@rhul.ac.uk

29 Feb 2012

RSA CONFERENCE 2012

The Problem

- *Motivation*: New algorithms are always useful as there are always so many different optimisations and conflicting pressures on resource-constrained platforms.
- *Aim*: Better exponentiation on space-limited chip. (Fast memory is expensive.)
- *Setting*: Mixed base representation for the exponent.
- *Solution*: Define a *dual* for the associated addition chain.
- *Benefits*: Derive new algorithms from existing ones; Better understanding of exponentiation.

Information Security Group

Outline

1. Background
2. The Transposition Method
3. Space Duality
4. Extra Requirements
5. New Algorithms
6. Conclusion

Background

1. Background
2. The Transposition Method
3. Space Duality
4. Extra Requirements
5. New Algorithms
6. Conclusion

Information Security Group

r-ary Exponentiation — L2R (Brauer, 1939)

Inputs: $g \in G$,
$D = ((d_{n-1}r + d_{n-2})r + \ldots + d_1)r + d_0 \in \mathbb{N}$ where $0 \le d_i < r$.

Output: $g^D \in G$

Initialise table: $T[d] \leftarrow g^d$ **for all** d, $0 < d < r$.
$P \leftarrow 1_G$
for $i \leftarrow n-1$ **downto** 0 **do** {
  if $i \neq n-1$ **then** $P \leftarrow P^r$
  if $d_i \neq 0$ **then** $P \leftarrow P \times T[d_i]$ }
return P

Information Security Group

r-ary Exponentiation — R2L (Yao, 1976)

Inputs: $g \in G$,
$D = d_{n-1}r^{n-1}+d_{n-2}r^{n-2}+\ldots+d_1r^1+d_0$ where $0 \le d_i < r$.

Output: $g^D \in G$

Initialise table: $T[d] \leftarrow 1_G$ **for all** d, $0<d<r$.
$P \leftarrow g$
for $i \leftarrow 0$ **to** $n-1$ **do** $\{$
  if $d_i \neq 0$ **then** $T[d_i] \leftarrow T[d_i] \times P$
  if $i \neq n-1$ **then** $P \leftarrow P^r$ $\}$
return $\prod_{d:0<d<r} T[d]^d$

Information Security Group

Sliding Window — L2R

Inputs: $g \in G$,
$D = ((d_{n-1}r_{n-2}+d_{n-2})r_{n-3}+\ldots+d_1)r_0+d_0 \in \mathbb{N}$, where
$d_i \in \{0, \pm 1, \pm 3, \ldots, \pm\frac{1}{2}(r-1)\}$, $r_i \in \{2, 2^w\}$ and $d_i=0$ if $r_i=2$.

Output: $g^D \in G$

Initialise table: $T[d] \leftarrow g^d$ **for all** $d \neq 0$.
$P \leftarrow 1_G$
for $i \leftarrow n-1$ **downto** 0 **do** {
  if $i \neq n-1$ **then** $P \leftarrow P^{r_i}$
  if $d_i \neq 0$ **then** $P \leftarrow P \times T[d_i]$ }
return P

Information Security Group

Sliding Window — R2L

Inputs: $g \in G$,
$D = ((d_{n-1}r_{n-2}+d_{n-2})r_{n-3}+\ldots+d_1)r_0+d_0 \in \mathbb{N}$, where
$d_i \in \{0, \pm1, \pm3, \ldots, \pm\frac{1}{2}(r-1)\}$, $r_i \in \{2, 2^w\}$ and $d_i=0$ if $r_i=2$.

Output: $g^D \in G$

Initialise table: $T[d] \leftarrow 1_G$ **for all** $d \neq 0$.
$P \leftarrow g$
for $i \leftarrow 0$ **to** $n-1$ **do** {
  if $d_i \neq 0$ **then** $T[d_i] \leftarrow T[d_i] \times P$
  if $i \neq n-1$ **then** $P \leftarrow P^{r_i}$ }
return $\prod_{d \neq 0} T[d]^d$

Information Security Group

Mixed Base Exponentiation — L2R

Inputs: $g \in G$,
$D = ((d_{n-1}r_{n-2}+d_{n-2})r_{n-3}+\ldots+d_1)r_0+d_0 \in \mathbb{N}$,
where $(r_i, d_i) \in \mathcal{R}\times\mathcal{D}$.

Output: $g^D \in G$

Initialise table: $T[d] \leftarrow g^d$ **for all** $d \in \mathcal{D} \setminus \{0\}$.
$P \leftarrow 1_G$
for $i \leftarrow n-1$ **downto** 0 **do** {
  if $i \neq n-1$ **then** $P \leftarrow P^{r_i}$
  if $d_i \neq 0$ **then** $P \leftarrow P \times T[d_i]$ }
return P

Mixed Base Exponentiation — R2L

Inputs: $g \in G$,
$D = ((d_{n-1}r_{n-2}+d_{n-2})r_{n-3}+\ldots+d_1)r_0+d_0 \in \mathbb{N}$,
where $(r_i, d_i) \in \mathcal{R}\times\mathcal{D}$.

Output: $g^D \in G$

Initialise table: $T[d] \leftarrow 1_G$ **for all** $d \in \mathcal{D} \setminus \{0\}$.
$P \leftarrow g$
for $i \leftarrow 0$ **to** $n-1$ **do** {
  if $d_i \neq 0$ **then** $T[d_i] \leftarrow T[d_i]\times P$
  if $i \neq n-1$ **then** $P \leftarrow P^{r_i}$ }
return $\prod_{d\in\mathcal{D}\setminus\{0\}} T[d]^d$

Information Security Group

A Compact Right-to-Left Algorithm (Arith13, 1997)

Inputs: $g \in G$,
$D = ((d_{n-1}r_{n-2}+d_{n-2})r_{n-3}+\ldots+d_1)r_0+d_0 \in \mathbb{N}$,
where $(r_i, d_i) \in \mathcal{R}\times\mathcal{D}$.

Output: $g^D \in G$

$T \leftarrow 1_G$
$P \leftarrow g$
for $i \leftarrow 0$ **to** $n-1$ **do** $\{$
  if $d_i \neq 0$ **then** $T \leftarrow T \times P^{d_i}$
  if $i \neq n-1$ **then** $P \leftarrow P^{r_i}$ $\}$
return T

The loop body involves computing P^{d_i} en route to P^{r_i}.

The Transposition Method

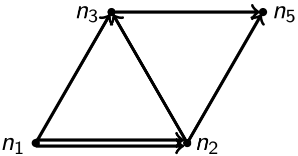
Information Security Group

The Computational Di-Graph

An addition chain for D yields a computational, acyclic *di-graph*:

Here is that for

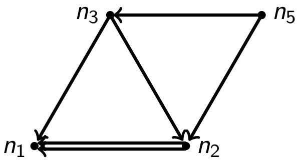
1+1=2; 1+2=3; 2+3=5.



For convenience, nodes are numbered so n_d represents g^d.

- Addition $i+j=k$ gives directed edges $n_i n_k$ and $n_j n_k$.
- It is *acyclic*, with a single root n_1 and a single leaf n_5.
- All nodes except root n_1 have input degree 2 as all op^s are binary.
- $\#\text{Ops} = \#\text{Nodes}-1 = \frac{1}{2}\#\text{Edges}$.
- By induction, $D = \#\text{paths from } n_1 \text{ to } n_D$.

Di-Graph for the Transpose Method



- Reverse the edges for the "*transposition*" method. Node inputs are again multiplied together.
- Path count is D, as before. So it again computes g^D.
- Nodes may need *merging* or *expanding* to restore in-degree 2. The #binary operations is not changed: $\frac{1}{2}$#edges.
- This reverses the addition chain in some sense.
- It *doesn't* preserve space requirements and without care, sq^{g} & mult^{n} counts may change.

Space Duality

Information Security Group

Royal Holloway University of London

RSACONFERENCE2012

Space-Aware Addition Chains

Definition. For a given set of registers, take five classes of “atomic” op^s:

- *Copying* one register to another;
- *Copying* one register to another *& initialising* source register to 1_G;
- In-place *squaring* of the contents of one register;
- *Multiplying* two different registers into one of the input registers;
- *Multiplying* two different registers into one of the input registers, *& initialising* the other input to 1_G.

A **space-aware addition chain** is a sequence of such operations in which the registers are named.

Every addition chain can be written as a space-aware addition chain.

Matrix Representation — Space

For a device with two locations, matrix examples of each class are:

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}, \quad \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \quad \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad \text{and} \quad \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}.$$

They act on a column vector containing the values in each register.

By omitting more general op^{ns}, this set is *closed under transposition*.

- Copy (without initialise) becomes multiplication with initialise, and *vice versa*. (The *red* matrices.)
- Other operations stay in their class under transposition.

Definition. The *dual* of a space-aware chain is its transpose.
(The transposed operations are applied in reverse order.)

The dual uses the same space but may not have the same mult^{n} count.

Matrix Representation — Space

For a device with two locations, matrix examples of each class are:

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}, \quad \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \quad \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad \text{and} \quad \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}.$$

They act on a column vector containing the values in each register.

By omitting more general op^{ns}, this set is *closed under transposition*.

- Copy (without initialise) becomes multiplication with initialise, and *vice versa*. (The *red* matrices.)
- Other operations stay in their class under transposition.

Definition. The *dual* of a space-aware chain is its transpose.
(The transposed operations are applied in reverse order.)

The dual uses the same space but may not have the same mult^n count.

Matrix Representation — Space

For a device with two locations, matrix examples of each class are:

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}, \quad \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \quad \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad \text{and} \quad \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}.$$

They act on a column vector containing the values in each register.

By omitting more general op^{ns}, this set is *closed under transposition*.

- Copy (without initialise) becomes multiplication with initialise, and *vice versa*. (The *red* matrices.)
- Other operations stay in their class under transposition.

Definition. The *dual* of a space-aware chain is its transpose.
(The transposed operations are applied in reverse order.)

The dual uses the same space but may not have the same mult^{n} count.

The Dual Chain — An Example

$R3 \leftarrow R2;\ \ R3 \leftarrow R2{+}R3;\ \ R1 \leftarrow_I R2;\ \ R2 \leftarrow_I R3;\ \ R2 \leftarrow_I R1{+}R2$

In matrices acting on a col^{mn} vector:

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The dual (the transpose) is:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

i.e. $R1 \leftarrow R2;\ \ R3 \leftarrow_I R2;\ \ R2 \leftarrow_I R1;\ \ R2 \leftarrow R2{+}R3;\ \ R2 \leftarrow_I R2{+}R3$

- Both have two multiplications and no squarings.
- Both compute g^3 from $g \in G$ with R_2 for I/O.

Extra Requirements

Information Security Group

The Main Problems

1. #Mults may not be preserved in the dual
   as copying becomes mult^n with initialisation.
2. The dual chain may not compute the same value
   unless the matrix product is symmetric.

To overcome the first of these, extra conditions are required:

- Select the initialising op^n when possible.
- Eliminate 1_G as an operand.
- Remove operations whose output is not used.
- Fix a subset of registers for I/O.
  (An I/O register *must* read input *and* write non-trivial output.)

Definition. A space-aware chain is *normalised* if the above hold.

The Main Problems

1. #Mults may not be preserved in the dual
   as copying becomes mult^n with initialisation.
2. The dual chain may not compute the same value
   unless the matrix product is symmetric.

To overcome the first of these, extra conditions are required:

- Select the initialising op^n when possible.
- Eliminate 1_G as an operand.
- Remove operations whose output is not used.
- Fix a subset of registers for I/O.
  (An I/O register *must* read input *and* write non-trivial output.)

Definition. A space-aware chain is *normalised* if the above hold.

The Main Problems

1. #Mults may not be preserved in the dual
   as copying becomes mult^n with initialisation.
2. The dual chain may not compute the same value
   unless the matrix product is symmetric.

To overcome the first of these, extra conditions are required:

- Select the initialising op^n when possible.
- Eliminate 1_G as an operand.
- Remove operations whose output is not used.
- Fix a subset of registers for I/O.
  (An I/O register *must* read input *and* write non-trivial output.)

Definition. A space-aware chain is *normalised* if the above hold.

Counting Ones

Instances of 1_G or $\perp$ arise from:

a) Initial value of a non-input register.

b) Initialised by copy or mult^{n} op^{n}.

Instances of 1_G or $\perp$ finish their lives as:

c) Final value in a non-output register.

d) Overwritten by a copy op^{n}.

Since $\#\text{a} = \#\text{c}$, we conclude $\#\text{b} = \#\text{d}$.
Subtracting the $\#\{\text{copies with init}^{\text{n}}\}$ from #b and #d, we have

$$\#\text{Mult}^{\text{ns}} \text{ with init}^{\text{n}} = \#\text{Copies without init}^{\text{n}}$$

These op^{n} types are swapped in the dual & others stay as they are. So:

- **Theorem.** For a normalised space-aware chain, $\#\text{Mult}^{\text{ns}}$ & $\#\text{Sq}^{\text{res}}$ are the same for the dual.

Counting Ones

Instances of 1_G or $\perp$ arise from:

a) Initial value of a non-input register.

b) Initialised by copy or mult^n op^n.

Instances of 1_G or $\perp$ finish their lives as:

c) Final value in a non-output register.

d) Overwritten by a copy op^n.

Since $\#a = \#c$, we conclude $\#b = \#d$.
Subtracting the $\#\{\text{copies with init}^n\}$ from $\#b$ and $\#d$, we have

$$\#\text{Mult}^{ns} \text{ with init}^n = \#\text{Copies without init}^n$$

These op^n types are swapped in the dual & others stay as they are. So:

- **Theorem.** For a normalised space-aware chain, $\#\text{Mult}^{ns}$ & $\#\text{Sq}^{res}$ are the same for the dual.

Counting Ones

Instances of 1_G or $\perp$ arise from:

a) Initial value of a non-input register.

b) Initialised by copy or mult^n op^n.

Instances of 1_G or $\perp$ finish their lives as:

c) Final value in a non-output register.

d) Overwritten by a copy op^n.

Since #a = #c, we conclude #b = #d.
Subtracting the #{copies with init^n} from #b and #d, we have

$$\#\text{Mult}^{ns} \text{ with init}^n = \#\text{Copies without init}^n$$

These op^n types are swapped in the dual & others stay as they are. So:

- **Theorem.** For a normalised space-aware chain, #Mult^{ns} & #Sq^{res} are the same for the dual.

Counting Ones

Instances of 1_G or $\perp$ arise from:

a) Initial value of a non-input register.

b) Initialised by copy or mult^n op^n.

Instances of 1_G or $\perp$ finish their lives as:

c) Final value in a non-output register.

d) Overwritten by a copy op^n.

Since $\#a = \#c$, we conclude $\#b = \#d$.
Subtracting the $\#\{$copies with init$^n\}$ from $\#b$ and $\#d$, we have

$$\#\text{Mult}^{ns} \text{ with init}^n = \#\text{Copies without init}^n$$

These op^n types are swapped in the dual & others stay as they are. So:

- **Theorem.** For a normalised space-aware chain, $\#\text{Mult}^{ns}$ & $\#\text{Sq}^{res}$ are the same for the dual.

Symmetric Cases

If the action of a (multi-) exponentiation function f on registers is described by matrix M then a dual f^* is described by the transpose M^T.

Theorem a) f^* computes the same values as f iff its matrix is symmetric.
b) In particular, it uses the same registers for output as input.

- In the normalised case, unused registers give columns of zeros.
- Used, non-output registers are over-written with 1_G: more zeros.
- Used, non-input registers are initialised to 1_G: more zeros.
- So only the sub-matrix M_{IO} on I/O registers need be symmetric.

Theorem A normalised space-aware chain for a *single* exponentiation and its dual compute the same values.

(Duals become unique only when written in atomic operations.)

Symmetric Cases

If the action of a (multi-) exponentiation function f on registers is described by matrix M then a dual f^* is described by the transpose M^{T}.

Theorem a) f^* computes the same values as f iff its matrix is symmetric.
b) In particular, it uses the same registers for output as input.

- In the normalised case, unused registers give columns of zeros.
- Used, non-output registers are over-written with 1_G: more zeros.
- Used, non-input registers are initialised to 1_G: more zeros.
- So only the sub-matrix M_{IO} on I/O registers need be symmetric.

Theorem A normalised space-aware chain for a *single* exponentiation and its dual compute the same values.

(Duals become unique only when written in atomic operations.)

Symmetric Cases

If the action of a (multi-) exponentiation function f on registers is described by matrix M then a dual f^* is described by the transpose M^T.

Theorem a) f^* computes the same values as f iff its matrix is symmetric.
b) In particular, it uses the same registers for output as input.

- In the normalised case, unused registers give columns of zeros.
- Used, non-output registers are over-written with 1_G: more zeros.
- Used, non-input registers are initialised to 1_G: more zeros.
- So only the sub-matrix M_{IO} on I/O registers need be symmetric.

Theorem A normalised space-aware chain for a *single* exponentiation and its dual compute the same values.

(Duals become unique only when written in atomic operations.)

New Algorithms

Information Security Group

High Level Algorithms

Question: When is an algorithm *dualisable* if its steps are more complex than the atomic operations?

We want to be able to decompose steps independently into atomic op^{ns} yet obtain the normalised property when all steps are concatenated.

Solution: For each step the values initially in its non-input registers must not be used and its used non-output registers must be reset to 1_G.

The output registers for one step must be the input registers for the next. (Include unused registers in the I/O set for convenience here.)

These are only requirements on how steps are realised as space-aware chains. So not a restriction on algorithm formulation.

Definition The dual of a high level exponentiation algorithm is that given by transposing its steps and reversing their order.

High Level Algorithms

Question: When is an algorithm *dualisable* if its steps are more complex than the atomic operations?

We want to be able to decompose steps independently into atomic op^{ns} yet obtain the normalised property when all steps are concatenated.

Solution: For each step the values initially in its non-input registers must not be used and its used non-output registers must be reset to 1_G.

The output registers for one step must be the input registers for the next. (Include unused registers in the I/O set for convenience here.)

These are only requirements on how steps are realised as space-aware chains. So not a restriction on algorithm formulation.

Definition The dual of a high level exponentiation algorithm is that given by transposing its steps and reversing their order.

High Level Algorithms

Question: When is an algorithm *dualisable* if its steps are more complex than the atomic operations?

We want to be able to decompose steps independently into atomic op^{ns} yet obtain the normalised property when all steps are concatenated.

Solution: For each step the values initially in its non-input registers must not be used and its used non-output registers must be reset to 1_G.

The output registers for one step must be the input registers for the next. (Include unused registers in the I/O set for convenience here.)

These are only requirements on how steps are realised as space-aware chains. So not a restriction on algorithm formulation.

Definition The dual of a high level exponentiation algorithm is that given by transposing its steps and reversing their order.

An "Old" Algorithm (Arith13, 1997)

Inputs: $g \in G$, $\quad D = ((d_{n-1}r_{n-2}+d_{n-2})r_{n-3}+\ldots+d_1)r_0+d_0 \in \mathbb{N}$
Output: $g^D \in G$

$T \leftarrow 1_G$
$P \leftarrow g$
for $i \leftarrow 0$ **to** $n-1$ **do** $\{$
  if $d_i \neq 0$ **then** $T \leftarrow T \times P^{d_i}$
  if $i \neq n-1$ **then** $P \leftarrow P^{r_i}$ $\}$
return T

The loop body involves computing P^{d_i} *en route* to P^{r_i}.

Information Security Group

One Iteration

Base/digit pairs (r, d) are chosen for compact, fast performance.
Specifically at most one register in addition to P and T.

e.g. $r = 2^i \pm 1, d = 2^j$ will involve i squarings & 2 mult^s.

It avoids a table entry for each d.

There is now a dual algorithm using the same space – only three registers.

The step $T \leftarrow TP^d$, $P \leftarrow P^r$ is achieved by $\begin{bmatrix} r & 0 \\ d & 1 \end{bmatrix} = \begin{bmatrix} r & d \\ 0 & 1 \end{bmatrix}^{\mathsf{T}}$.

So the transpose performs the dual op^n $P \leftarrow P^r T^d$.

The sequence of op^s is easily determined via the dual.

One Iteration

Base/digit pairs (r, d) are chosen for compact, fast performance.
Specifically at most one register in addition to P and T.

e.g. $r = 2^i \pm 1, d = 2^j$ will involve i squarings & 2 mult^s.

It avoids a table entry for each d.

There is now a dual algorithm using the same space – only three registers.

The step $T \leftarrow TP^d$, $P \leftarrow P^r$ is achieved by $\begin{bmatrix} r & 0 \\ d & 1 \end{bmatrix} = \begin{bmatrix} r & d \\ 0 & 1 \end{bmatrix}^{\mathsf{T}}$.

So the transpose performs the dual op^n $P \leftarrow P^r T^d$.

The sequence of op^s is easily determined via the dual.

A New Compact Left-to-Right Algorithm

Inputs: $g \in G, \quad D = ((d_{n-1}r_{n-2}+d_{n-2})r_{n-3}+\ldots+d_1)r_0+d_0 \in \mathbb{N}$
Output: $g^D \in G$

$T \leftarrow g$
$P \leftarrow 1_G$
for $i \leftarrow n-1$ **downto** 0 **do**
    $P \leftarrow P^{r_i} \times T^{d_i}$
return P

Loop iterations are computed as described on last slide.

It is the dual of the previous R2L algorithm, as just derived.

Information Security Group

Royal Holloway University of London

RSACONFERENCE2012

The Value of the Algorithm

- "Table-less" exponentiation – useful in constrained environments.
- If space for only three registers and division has the same cost as mult$^{\text{n}}$, the compact algorithms are faster.
- A left-to-right version allows better use of composite op$^{\text{s}}$, e.g. double-and-add, triple-and-add, quintuple-and-add.
- Recoding is done on-the-fly for R2L exp$^{\mathrm{n}}$; in advance for L2R exp$^{\mathrm{n}}$. The recoding typically needs up to 3 times the storage space of D.

Information Security Group

Conclusion

Information Security Group

Summary & Final Remarks

- A general setting enabling most exp^{n} algorithms to be described naturally, namely a mixed base recoding.
- A new space- and time-preserving duality between left-to-right and right-to-left exp^{n} algorithms.
- A new tableless exp^{n} algorithm. It enables new speed records to be set in certain environments.
- New understanding of exp^{n} is possible, e.g. a comparison of R2L initialisation with L2R finalisation steps.
- . . .

Information Security Group

Summary & Final Remarks

- A general setting enabling most exp^{n} algorithms to be described naturally, namely a mixed base recoding.
- A new space- and time-preserving duality between left-to-right and right-to-left exp^{n} algorithms.
- A new tableless exp^{n} algorithm. It enables new speed records to be set in certain environments.
- New understanding of exp^{n} is possible, e.g. a comparison of R2L initialisation with L2R finalisation steps.
- . . .

Information Security Group

Summary & Final Remarks

- A general setting enabling most $\exp^n$ algorithms to be described naturally, namely a mixed base recoding.
- A new space- and time-preserving duality between left-to-right and right-to-left $\exp^n$ algorithms.
- A new tableless $\exp^n$ algorithm. It enables new speed records to be set in certain environments.
- New understanding of $\exp^n$ is possible, e.g. a comparison of R2L initialisation with L2R finalisation steps.
- ...

Summary & Final Remarks

- A general setting enabling most exp^{n} algorithms to be described naturally, namely a mixed base recoding.
- A new space- and time-preserving duality between left-to-right and right-to-left exp^{n} algorithms.
- A new tableless exp^{n} algorithm.
  It enables new speed records to be set in certain environments.
- New understanding of exp^{n} is possible,
  e.g. a comparison of R2L initialisation with L2R finalisation steps.
- . . .

Information Security Group

Summary & Final Remarks

- A general setting enabling most exp^{n} algorithms to be described naturally, namely a mixed base recoding.
- A new space- and time-preserving duality between left-to-right and right-to-left exp^{n} algorithms.
- A new tableless exp^{n} algorithm. It enables new speed records to be set in certain environments.
- New understanding of exp^{n} is possible, e.g. a comparison of R2L initialisation with L2R finalisation steps.
- ...

Information Security Group