Format Transforming Encryption:
More than Meets the DPI

Nicholas Mainardi
Email: nicholas.mainardi@polimi.it
home.deib.polimi.it/nmainardi

15th September 2017
This talk is mainly based on the seminars of Tom Shrimpton (Florida Institute for Cybersecurity Research) at Real World Crypto and Privacy summer school (Sibenik 5-9th June 2017)

Outline

1. FTE Definition
2. Ranking/unranking algorithms
3. Relaxed ranking
4. Build FTE framework
5. Application Scenarios: In-Place Encryption and Great Firewall of China
6. Further developments
Traditional Encryption vs FTE

Traditional Encryption

Plaintexts and ciphertexts are bit strings!
Traditional Encryption vs FTE

**Traditional Encryption**

- Plaintexts and ciphertexts are bit strings!

**Format Transforming Encryption**

- Ciphertexts must abide the specified format!
- Plaintexts and ciphertexts formats can be changed online
How Do We Define a Format?

- A format is a set of rules which defines the structure of the plain/ciphertexts.
- Set of rules → a grammar which defines the syntax of the messages.
- That is, a format basically defines a language.
- A rather intuitive and user friendly way of specifying a language is a regular expression.
How Do We Define a Format?

- A format is a set of rules which defines the structure of the plain/ciphertexts
- Set of rules → a grammar which defines the syntax of the messages
- That is, a format basically defines a language
- A rather intuitive and user friendly way of specifying a language is a regular expression
Building FTE

2 approaches:

1. We use a bijection to map words of the language to a domain which is suitable for encryption algorithms (bit strings)
Building FTE

2 approaches:

1. We use a bijection to map words of the language to a domain which is suitable for encryption algorithms (bit strings)
2. We design a cipher able to output formatted ciphertexts by itself
Building FTE

2 approaches:
1. We use a bijection to map words of the language to a domain which is suitable for encryption algorithms (bit strings)
2. We design a cipher able to output formatted ciphertexts by itself

Ranking

- An invertible, bijective mapping between words of a language and integers, according to an established lexicographical ordering
- Goldberg and Sipser proposed an algorithm to efficiently rank and unrank regular languages, starting from their Deterministic Finite Automaton (DFA) in 1985

Formal definition:

\[ \text{rank} : L \mapsto \mathbb{Z}_{|L|}, \quad \text{unrank} : \mathbb{Z}_{|L|} \mapsto L \quad \text{s.t.} \quad \forall l \in L, \quad \text{unrank}(\text{rank}(l)) = l \]

Once we have a map to \( \mathbb{Z}_{|L|} \), we can use whatever cipher to encrypt the plaintext!
This construction is really flexible: change the regex to change the format
Not all the formats can be used: $|L(R_p)| \leq |L(R_c)|$. Why?
This construction is really flexible: change the regex to change the format
Not all the formats can be used: $|L(R_p)| \leq |L(R_c)|$. Why?
Multiple plaintexts would map to the same ciphertext!
The basic brick of the framework: How to efficiently rank/unrank?

**Ranking Idea**

- Ranking requires a DFA representation of the language
- An ordering on the alphabet of the language is defined
- The ranking procedure proposed by Goldberg and Sipser can be conceptually split in 2 parts:
  1. The string \( x \in L \) is mapped to its corresponding accepting path in the DFA (parsing)
  2. Accepting paths are ranked according to a lexicographical order
- A precomputed table of size \( |Q| \cdot \max_{x \in L}(|X|) \) is employed
- With this precomputed table, ranking and unranking are \( O(|X|) \)
## DFA Notation

A deterministic finite automaton (DFA) is a 5-tuple \( (Q, \Sigma, \delta, q_0, F) \), where:

- \( Q \) is the set of states
- \( \Sigma \) is the alphabet of the language described by the DFA
- \( \delta : Q \times \Sigma \rightarrow Q \) is the transition function
- \( q_0 \) is the initial state
- \( F \) is the set of final states

The rank and unrank procedures employ the images of the function 

\[
T(q, n) = \text{the number of accepting paths starting from state } q \text{ of length } n
\]

Thus, \( n \) can be at most \( \max_{x \in L} |x| \). This function can be precomputed and stored in tabular form.
DFA Notation

A deterministic finite automaton (DFA) is a 5-tuple \( (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{F}) \), where:

- \( \mathcal{Q} \) is the set of states
- \( \Sigma \) is the alphabet of the language described by the DFA
- \( \delta : \mathcal{Q} \times \Sigma \mapsto \mathcal{Q} \) is the transition function
- \( q_0 \) is the initial state
- \( \mathcal{F} \) is the set of final states

- Rank and unrank procedures employ the images of function \( T : \mathcal{Q} \times \mathbb{N} \mapsto \mathbb{N} \)
  - \( T(q, n) \) is the number of accepting paths starting from state \( q \) of length \( n \)
  - Thus, \( n \) can be at most \( \max_{x \in \mathcal{L}}(|x|) \)
  - This function can be precomputed and stored in tabular form
The function can be computed in a recursive fashion:

\[
T(q, n) = \begin{cases}
1 & n = 0 \land q \in F \\
0 & n = 0 \land q \notin F \\
\sum_a T(\delta(q, a), n - 1) & \text{otherwise}
\end{cases}
\]

We can write a simple procedure to compute this function:

```
Table Precomputation
BUILD-TABLE(n)
for q ∈ Q do if q ∈ F then T(q, 0) ← 1
for i ← 1 to n do for q ∈ Q do for a ∈ Σ do T(q, i) + ← T(δ(q, a), i - 1)
```
The function can be computed in a recursive fashion:

\[
T(q, n) = \begin{cases} 
1 & n = 0 \land q \in F \\
0 & n = 0 \land q \notin F \\
\sum_a T(\delta(q, a), n - 1) & \text{otherwise}
\end{cases}
\]

We can write a simple procedure to compute this function:

**Table Precomputation**

```plaintext
BUILD-TABLE(n)
1 for q ∈ Q
2 do if q ∈ F
3 then T(q, 0) ← 1
4 for i ← 1 to n
5 do for q ∈ Q
6 do for a ∈ Σ
7 do T(q, i) ← T(δ(q, a), i - 1)
```
Precomputation Example

Example format: \( L = \{ (e^+ h | la)^+ \} \)

DFA for \( L \)

```
Code
BUILD-TABLE(n)
1  for q ∈ Q
2      do if q ∈ F
3      then T(q, 0) ← 1
4  for i ← 1 to n
5      do for q ∈ Q
6          do for a ∈ Σ
7              do T(q, i) ← T(δ(q, a), i − 1)
```

Tabulated Function \( T(q, n) \)

<table>
<thead>
<tr>
<th>States</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₀</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example format: \( L = \{(e^+ h|la)^+\} \)

**Code**

```plaintext
BUILD-TABLE(n)
1 for q ∈ Q
2       do if q ∈ F
3         then T(q, 0) ← 1
4 for i ← 1 to n
5       do for q ∈ Q
6          do for a ∈ Σ
7           do T(q, i) ← T(δ(q, a), i − 1)
```

**Tabulated Function** \( T(q, n) \)

<table>
<thead>
<tr>
<th>States</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_1 )</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_2 )</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_3 )</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Precomputation Example

Example format: \( L = \{(e^+ h|a)^+\} \)

**DFA for \( L \)**

**Code**

```plaintext
BUILD-TABLE(n)
1 for q ∈ Q
2   do if q ∈ F
3   then T(q, 0) ← 1
4 for i ← 1 to n
5   do for q ∈ Q
6       do for a ∈ Σ
7           do T(q, i) ← T(δ(q, a), i − 1)
```

**Tabulated Function \( T(q, n) \)**

<table>
<thead>
<tr>
<th>States</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_1 )</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_2 )</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_3 )</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example format: \( L = \{(e^+ h|la)^+\} \)

### DFA for \( L \)

- **States:** \( q_0, q_1, q_2, q_3 \)
- **Transitions:**
  - \( q_0 \) to \( q_1 \) on \( e \)
  - \( q_1 \) to \( q_0 \) on \( e \)
  - \( q_2 \) to \( q_3 \) on \( h \)
  - \( q_3 \) to \( q_2 \) on \( l \)

### Code

```plaintext
BUILD-TABLE(n)
1 for q ∈ Q
2   do if q ∈ F
3   then T(q, 0) ← 1
4 for i ← 1 to n
5   do for q ∈ Q
6     do for a ∈ Σ
7       do T(q, i) ← T(δ(q, a), i − 1)
```

### Tabulated Function \( T(q, n) \)

<table>
<thead>
<tr>
<th>States</th>
<th>( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 1 2 3 4</td>
</tr>
<tr>
<td>( q_0 )</td>
<td>0 0 2</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>0 1 1</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>1 0 2</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>0 1 0</td>
</tr>
</tbody>
</table>
Precomputation Example

Example format: \( L = \{(e^+ h | la)^+\} \)

DFA for \( L \)

Code

```
BUILD-TABLE(n)
1    for q ∈ Q
2        do if q ∈ F
3            then T(q, 0) ← 1
4    for i ← 1 to n
5        do for q ∈ Q
6            do for a ∈ Σ
7                do T(q, i) ← T(δ(q, a), i − 1)
```

Tabulated Function \( T(q, n) \)

<table>
<thead>
<tr>
<th>States</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>( q_1 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( q_2 )</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>( q_3 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Example format: $L = \{(e^+ h|la)^+\}$

**DFA for $L$**

**Code**

```latex
\text{BUILD-TABLE}(n)\\
1 \quad \text{for } q \in \mathbb{Q}\\
2 \quad \text{do if } q \in F\\
3 \quad \text{then } T(q, 0) \leftarrow 1\\
4 \quad \text{for } i \leftarrow 1 \text{ to } n\\
5 \quad \text{do for } q \in \mathbb{Q}\\
6 \quad \text{do for } a \in \Sigma\\
7 \quad \text{do } T(q, i) \leftarrow T(\delta(q, a), i - 1)
```

**Tabulated Function $T(q, n)$**

<table>
<thead>
<tr>
<th>States</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>$q_1$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$q_2$</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>$q_3$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Exploiting $T(q, n)$ for Ranking

To rank a string $x \in L$:

- All the strings of the language whose length is less than $|x|$ can be easily counted by $\sum_{i=0}^{\lfloor |x| - 1 \rfloor} T(q_0, i)$
- We need an algorithm to compute the strings whose length is $|x|$ which are lexicographically less than $x$
Exploiting $T(q, n)$ for Ranking

To rank a string $x \in L$:

- All the strings of the language whose length is less than $|x|$ can be easily counted by $\sum_{i=0}^{|x|-1} T(q_0, i)$
- We need an algorithm to compute the strings whose length is $|x|$ which are lexicographically less than $x$

Again, rank can be recursively computed using $T(q, n)$:

$$rank(x[0 \ldots n]) = \begin{cases} 
0 & x = \epsilon \\
rank(x[1 \ldots n]) + \sum_{a \in \Sigma \ a < x[0]} T(\delta(q, a), n - 1) & \text{otherwise}
\end{cases}$$

with $q$ being updated according to the parsing of $x \rightarrow q = \delta(q, x[0])$ at each step.
Rank Algorithm

\[
\text{RANK}(x) \\
1 \quad q \leftarrow q_0 \\
2 \quad \text{rank} \leftarrow 0 \\
3 \quad \text{for } i \leftarrow 0 \text{ to } |x| - 1 \\
4 \quad \quad \text{do for } a \in \Sigma \\
5 \quad \quad \quad \text{do if } a \geq x[i] \\
6 \quad \quad \quad \quad \text{then continue} \\
7 \quad \quad \quad \text{rank} \leftarrow T(\delta(q, a), |x| - i - 1) \\
8 \quad q \leftarrow \delta(q, x[i]) \\
9 \quad \text{rank} \leftarrow T(q_0, i)
\]
**Rank Algorithm**

```
RANK(x)
1 q ← q₀
2 rank ← 0
3 for i ← 0 to |x| − 1
4   do for a ∈ Σ
5     do if a ≥ x[i]
6       then continue
7     rank ← T(δ(q, a), |x| − i − 1)
8   q ← δ(q, x[i])
9 rank ← T(q₀, i)
```

Example for $x = ehla$:

- $i = 0$
- $x[i] = e$
- $rank = 0$

**DFA for $L$**

**Tabulated Function $T(q, n)$**

<table>
<thead>
<tr>
<th>States</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₀</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>q₁</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>q₂</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>q₃</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Rank Algorithm

RANK(x)
1. q ← q₀
2. rank ← 0
3. for i ← 0 to |x| − 1
4. do for a ∈ Σ
5. if a ≥ x[i]
6. then continue
7. rank ← T(δ(q, a), |x| − i − 1)
8. q ← δ(q, x[i])
9. rank ← T(q₀, i)

Example for x = ehla:

- i = 1
- x[i] = h
- rank = 1

DFA for L

Tabulated Function T(q, n)

<table>
<thead>
<tr>
<th>States</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>q₀</td>
<td>0</td>
</tr>
<tr>
<td>q₁</td>
<td>0</td>
</tr>
<tr>
<td>q₂</td>
<td>1</td>
</tr>
<tr>
<td>q₃</td>
<td>0</td>
</tr>
</tbody>
</table>
Rank Algorithm

**Example for** $x = \text{ehla}:

- $i = 2$
- $x[i] = l$
- $\text{rank} = 4$
**Rank Algorithm**

**RANK** (x)

1. \( q \leftarrow q_0 \)
2. \( \text{rank} \leftarrow 0 \)
3. **for** \( i \leftarrow 0 \) **to** \( |x| - 1 \)
4. **do** **for** \( a \in \Sigma \)
5. **do if** \( a \geq x[i] \)
6. **then continue**
7. \( \text{rank} \leftarrow T(\delta(q, a), |x| - i - 1) \)
8. \( q \leftarrow \delta(q, x[i]) \)
9. \( \text{rank} \leftarrow T(q_0, i) \)

**Example for** \( x = ehla: \)

- \( i = 3 \)
- \( x[i] = a \)
- \( \text{rank} = 5 \)

**DFA for** \( L \)

**Tabulated Function** \( T(q, n) \)

<table>
<thead>
<tr>
<th>States</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Rank Algorithm

RANK(x)
1  \( q \leftarrow q_0 \)
2  \( \text{rank} \leftarrow 0 \)
3  \text{for } i \leftarrow 0 \text{ to } |x| - 1 
4    \text{do for } a \in \Sigma 
5      \text{do if } a \geq x[i] 
6        \text{then continue} 
7      \text{rank} \leftarrow T(\delta(q, a), |x| - i - 1) 
8  q \leftarrow \delta(q, x[i]) 
9  \text{rank} \leftarrow T(q_0, i)

Example for \( x = ehla \): 

\( \text{rank} = 5 \rightarrow \text{There are 5 strings less than } x: \text{ eh, la, eeh, eeh, eeh } \)

Tabulated Function \( T(q, n) \)

<table>
<thead>
<tr>
<th>States</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Unrank Algorithm

Unranking Idea: We can get back the string due to the "structure" of the rank

Rank Structure

- The rank of $x$ is the number of strings lexicographically less than $x$
- Thus, we can consider the rank as a sum of all these strings
- We can partition the elements of this sum in order to use them to build $x$ one character at a time
Unrank Algorithm

Unranking Idea: We can get back the string due to the "structure" of the rank

Rank Structure

- The rank of $x$ is the number of strings lexicographically less than $x$
- Thus, we can consider the rank as a sum of all these strings
- We can partition the elements of this sum in order to use them to build $x$ one character at a time
- Rank partitioning (given $n = |x|$):

$$r = |\{s : |s| = 0\}| + |\{s : |s| = 1\}| + \cdots + |\{s : |s| = n - 1\}|$$

$$+ |\{s : |s| = n \land s[0] < x[0]\}| + |\{s : |s| = n \land s[1] < x[1] \land s[0] = x[0]\}|$$

$$+ |\{s : |s| = n \land s[2] < x[2] \land s[0 \ldots 1] = x[0 \ldots 1]\}| + \cdots +$$

$$+ |\{s : |s| = n \land s[n - 1] < x[n - 1] \land s[0 \ldots n - 2] = x[0 \ldots n - 2]\}|$$

- The size of these sets can be computed using $T(q, n)$ table, and used together with $r$ to build $x$
Unrank Algorithm: Exploiting Partitions

Find \( n = |x| \)

- \( \{s : |s| = 0\} = T(q_0, 0) \land \{s : |s| = 1\} = t(q_0, 1) \ldots \)
- We sum up all \( T(q_0, i) \), starting from \( i = 0 \), until when \( \text{sum} > r \)
- The \( i \) when we stop is equal to \( n \)
Unrank Algorithm: Exploiting Partitions

**Find $n = |x|$**

- $|\{s : |s| = 0\}| = T(q_0, 0) \land |\{s : |s| = 1\}| = t(q_0, 1) \ldots$
- We sum up all $T(q_0, i)$, starting from $i = 0$, until when $sum > r$
- The $i$ when we stop is equal to $n$

**Find $x[0]$**

- Compute $r = r - \sum_{i=0}^{n-1} T(q_0, i)$
- $|\{s : |s| = n \land s[0] < x[0]\}| = \sum_{\begin{array}{c}a \in \Sigma \\text{a} < x[0]\end{array}} T(\delta(q_0, a), n - 1)$
- We sum up all $T(\delta(q_0, a), n - 1)$, following alphabetical order to select the character $a$, until when $sum > r$
- The $a$ when we stop is $x[0]$
Unrank Algorithm: Exploiting Partitions

**Find n = |x|**

- \(|\{ s : |s| = 0\}| = T(q_0, 0) \land |\{ s : |s| = 1\}| = t(q_0, 1) \ldots\)
- We sum up all \(T(q_0, i)\), starting from \(i = 0\), until when \(sum > r\)
- The \(i\) when we stop is equal to \(n\)

**Find x[0]**

- Compute \(r = r - \sum_{i=0}^{n-1} T(q_0, i)\)
- \(|\{ s : |s| = n \land s[0] < x[0]\}| = \sum_{a \in \Sigma} T(\delta(q_0, a), n - 1)\)
- We sum up all \(T(\delta(q_0, a), n - 1)\), following alphabetical order to select the character \(a\), until when \(sum > r\)
- The \(a\) when we stop is \(x[0]\)

**Finding next character**

- Compute \(r = r - \sum_{a \in \Sigma} T(\delta(q_0, a), n - 1)\)
- Repeat the process used for \(x[0]\) replacing \(T(\delta(q', a), n - 2)\), where \(q' = \delta(q_0, x[0])\), with \(T(\delta(q_0, a), n - 1)\) in the summation
Unrank Algorithm

Code

```
UNRANK(r)

1. q ← q₀
2. n ← 0
3. while r ≥ T(q₀, n)
   4. do r ← T(q₀, n)
   5. n ← 1
6. x ← ϵ
7. for i ← 0 to n − 1
   8. do j ← 1
   9. while r ≥ T(δ(q, aᵢ), n − i − 1)
   10. do r ← T(δ(q, aᵢ), n − i − 1)
   11. j ← 1
12. x[i] ← aᵢ
13. q ← δ(q, x[i])
```
Unrank Algorithm

Code

UNRANK(r)
1 \( q \leftarrow q_0 \)
2 \( n \leftarrow 0 \)
3 \( \textbf{while } r \geq T(q_0, n) \)
4 \( \quad \textbf{do } r \leftarrow T(q_0, n) \)
5 \( \quad n \leftarrow 1 \)
6 \( x \leftarrow \epsilon \)
7 \( \textbf{for } i \leftarrow 0 \textbf{ to } n - 1 \)
8 \( \quad \textbf{do } j \leftarrow 1 \)
9 \( \quad \textbf{while } r \geq T(\delta(q, a_j), n - i - 1) \)
10 \( \quad \quad \textbf{do } r \leftarrow T(\delta(q, a_j), n - i - 1) \)
11 \( \quad j \leftarrow 1 \)
12 \( x[i] \leftarrow a_j \)
13 \( q \leftarrow \delta(q, x[i]) \)

DFA for \( L \)

Example: unranking \( r = 5 \) for
\( L = (eh^+|la)^+ \)

- \( r = 5 \)
- \( n = 0 \)
- \( x = \epsilon \)

Tabulated Function \( T(q, n) \)

<table>
<thead>
<tr>
<th>States</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Unrank Algorithm

<table>
<thead>
<tr>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNRANK(r)</td>
</tr>
<tr>
<td>1 ( q \leftarrow q_0 )</td>
</tr>
<tr>
<td>2 ( n \leftarrow 0 )</td>
</tr>
<tr>
<td>3 while ( r \geq T(q_0, n) )</td>
</tr>
<tr>
<td>4 do ( r \leftarrow T(q_0, n) )</td>
</tr>
<tr>
<td>5 ( n \leftarrow 1 )</td>
</tr>
<tr>
<td>6 ( x \leftarrow \epsilon )</td>
</tr>
<tr>
<td>7 for ( i \leftarrow 0 ) to ( n - 1 )</td>
</tr>
<tr>
<td>8 do ( j \leftarrow 1 )</td>
</tr>
<tr>
<td>9 while ( r \geq T(\delta(q, a_j), n - i - 1) )</td>
</tr>
<tr>
<td>10 do ( r \leftarrow T(\delta(q, a_j), n - i - 1) )</td>
</tr>
<tr>
<td>11 ( j \leftarrow 1 )</td>
</tr>
<tr>
<td>12 ( x[i] \leftarrow a_j )</td>
</tr>
<tr>
<td>13 ( q \leftarrow \delta(q, x[i]) )</td>
</tr>
</tbody>
</table>

Example: unranking \( r = 5 \) for 
\( L = (eh^+ | la)^+ \)

- \( r = 5 \)
- \( n = 0 \)
- \( x = \epsilon \)

<table>
<thead>
<tr>
<th>DFA for ( L )</th>
</tr>
</thead>
<tbody>
<tr>
<td>start ( \rightarrow ) q0 ( \rightarrow ) q1 ( \rightarrow ) q3 ( \rightarrow ) q2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tabulated Function ( T(q, n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>States</td>
</tr>
<tr>
<td>( q_0 )</td>
</tr>
<tr>
<td>( q_1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
</tr>
<tr>
<td>( q_3 )</td>
</tr>
</tbody>
</table>
Unrank Algorithm

**Code**

UNRANK\( (r) \)
1. \( q \leftarrow q_0 \)
2. \( n \leftarrow 0 \)
3. while \( r \geq T(q_0, n) \)
   4. do \( r \leftarrow T(q_0, n) \)
   5. \( n \leftarrow 1 \)
6. \( x \leftarrow \epsilon \)
7. for \( i \leftarrow 0 \) to \( n - 1 \)
   8. do \( j \leftarrow 1 \)
   9. while \( r \geq T(\delta(q, a_j), n - i - 1) \)
   10. do \( r \leftarrow T(\delta(q, a_j), n - i - 1) \)
   11. \( j \leftarrow 1 \)
12. \( x[i] \leftarrow a_j \)
13. \( q \leftarrow \delta(q, x[i]) \)

**Example:** unranking \( r = 5 \) for 
\( L = (eh^+ | la)^+ \)
- \( r = 5 \)
- \( n = 1 \)
- \( x = \epsilon \)

**DFA for \( L \)**

**Tabulated Function \( T(q, n) \)**

<table>
<thead>
<tr>
<th>States</th>
<th>( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 1 2 3 4</td>
</tr>
<tr>
<td>( q_0 )</td>
<td>0 0 2 1 5</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>0 1 1 3 4</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>1 0 2 1 5</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>0 1 0 2 1</td>
</tr>
</tbody>
</table>
### Unrank Algorithm

#### Code

UNRANK(r)

1. \( q \leftarrow q_0 \)
2. \( n \leftarrow 0 \)
3. while \( r \geq T(q_0, n) \)
   do \( r \leftarrow T(q_0, n) \)
4. \( n \leftarrow 1 \)
5. \( x \leftarrow \epsilon \)
6. for \( i \leftarrow 0 \) to \( n - 1 \)
7.   do \( j \leftarrow 1 \)
8.     while \( r \geq T(\delta(q, a_j), n - i - 1) \)
9.       do \( r \leftarrow T(\delta(q, a_j), n - i - 1) \)
10. \( j \leftarrow 1 \)
11. \( x[i] \leftarrow a_j \)
12. \( q \leftarrow \delta(q, x[i]) \)

#### Example: unranking \( r = 5 \) for
\[ L = (eh^+|la)^+ \]

- \( r = 3 \)
- \( n = 2 \)
- \( x = \epsilon \)

### DFA for \( L \)

#### Tabulated Function \( T(q, n) \)

<table>
<thead>
<tr>
<th>States</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Unrank Algorithm

**Code**

```plaintext
UNRANK(r)
1  q ← q₀
2  n ← 0
3  while r ≥ T(q₀, n)
4    do r ← T(q₀, n)
5      n ← n + 1
6  x ← ε
7  for i ← 0 to n − 1
8    do j ← 1
9      while r ≥ T(δ(q, a_j), n – i – 1)
10     do r ← T(δ(q, a_j), n – i – 1)
11    j ← j + 1
12  x[i] ← a_j
13  q ← δ(q, x[i])
```

Example: unranking \( r = 5 \) for 
\[
L = (eh^+|la)^+
\]

- \( r = 2 \)
- \( n = 3 \)
- \( x = ε \)

**DFA for \( L \)**

![DFA diagram](image)

**Tabulated Function \( T(q, n) \)**

<table>
<thead>
<tr>
<th>States</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q₀ )</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( q₁ )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>( q₂ )</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( q₃ )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Unrank Algorithm

**Code**

\text{UNRANK}(r)

1. \( q \leftarrow q_0 \)
2. \( n \leftarrow 0 \)
3. \textbf{while} \( r \geq T(q_0, n) \)
4. \quad \textbf{do} \( r \leftarrow T(q_0, n) \)
5. \quad \( n \leftarrow 1 \)
6. \( x \leftarrow \epsilon \)
7. \textbf{for} \( i \leftarrow 0 \) \textbf{to} \( n - 1 \)
8. \quad \textbf{do} \( j \leftarrow 1 \)
9. \quad \textbf{while} \( r \geq T(\delta(q, a_j), n - i - 1) \)
10. \quad \textbf{do} \( r \leftarrow T(\delta(q, a_j), n - i - 1) \)
11. \quad \( j \leftarrow 1 \)
12. \( x[i] \leftarrow a_j \)
13. \( q \leftarrow \delta(q, x[i]) \)

**Example:** unranking \( r = 5 \) for 

\( L = (eh^+ | la)^+ \)

- \( r = 2 \)
- \( n = 4 \)
- \( x = \epsilon \)

**DFA for \( L \)**

**Tabulated Function** \( T(q, n) \)

<table>
<thead>
<tr>
<th>States</th>
<th>( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>0 0 2 1 5</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>0 1 1 3 4</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>1 0 2 1 5</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>0 1 0 2 1</td>
</tr>
</tbody>
</table>
Unrank Algorithm

**Code**

```
UNRANK(r)
1  q ← q0
2  n ← 0
3  while r ≥ T(q0, n)
4      do r ← T(q0, n)
5          n ← 1
6  x ← ε
7  for i ← 0 to n − 1
8      do j ← 1
9          while r ≥ T(δ(q, a_j), n − i − 1)
10         do r ← T(δ(q, a_j), n − i − 1)
11     j ← 1
12    x[i] ← a_j
13  q ← δ(q, x[i])
```

**Example**: unranking \( r = 5 \) for \( L = (eh^+|la)^+ \)

- \( r = 2 \)
- \( n = 4 \)
- \( x = \epsilon \)

**DFA for \( L \)**

**Tabulated Function \( T(q, n) \)**

<table>
<thead>
<tr>
<th>States</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Unrank Algorithm

**Code**

UNRANK($r$)

1. $q \leftarrow q_0$
2. $n \leftarrow 0$
3. **while** $r \geq T(q_0, n)$
4.     **do** $r \leftarrow T(q_0, n)$
5.     $n \leftarrow 1$
6. $x \leftarrow \epsilon$
7. **for** $i \leftarrow 0$ **to** $n - 1$
8.     **do** $j \leftarrow 1$
9.     **while** $r \geq T(\delta(q, a_j), n - i - 1)$
10.       **do** $r \leftarrow T(\delta(q, a_j), n - i - 1)$
11.         $j \leftarrow 1$
12. $x[i] \leftarrow a_j$
13. $q \leftarrow \delta(q, x[i])$

**Example:** unranking $r = 5$ for $L = (eh^+|la)^+$

- $r = 2$
- $n = 4$
- $x = e$

**DFA for $L$**

![DFA Diagram]

**Tabulated Function $T(q, n)$**

<table>
<thead>
<tr>
<th>States</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>$q_0$</td>
<td>0</td>
</tr>
<tr>
<td>$q_1$</td>
<td>0</td>
</tr>
<tr>
<td>$q_2$</td>
<td>1</td>
</tr>
<tr>
<td>$q_3$</td>
<td>0</td>
</tr>
</tbody>
</table>
Unrank Algorithm

**Code**

```plaintext
UNRANK(r)
    1 q ← q₀
    2 n ← 0
    3 while r ≥ T(q₀, n)
        4 do r ← T(q₀, n)
        5 n ← 1
    6 x ← ε
    7 for i ← 0 to n − 1
        8 do j ← 1
        9 while r ≥ T(δ(q, aᵢ), n − i − 1)
        10 do r ← T(δ(q, aᵢ), n − i − 1)
        11 j ← 1
        12 x[i] ← aᵢ
        13 q ← δ(q, x[i])
```

**Example: unranking r = 5 for**

\[ L = (eh^+ | la)^+ \]

- \( r = 2 \)
- \( n = 4 \)
- \( x = e \)

**DFA for L**

**Tabulated Function** \( T(q, n) \)

<table>
<thead>
<tr>
<th>States</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q₀ )</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( q₁ )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>( q₂ )</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( q₃ )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Unrank Algorithm

**Code**

```plaintext
UNRANK(r)
1  q ← q_0
2  n ← 0
3  while r ≥ T(q_0, n)
4    do r ← T(q_0, n)
5      n ← n + 1
6  x ← ε
7  for i ← 0 to n - 1
8    do j ← 1
9    while r ≥ T(δ(q, a_j), n - i - 1)
10   do r ← T(δ(q, a_j), n - i - 1)
11    j ← j + 1
12  x[i] ← a_j
13  q ← δ(q, x[i])
```

**Example:** unranking \( r = 5 \) for 
\( L = (eh^+|la)^+ \)

- \( r = 1 \)
- \( n = 4 \)
- \( x = ε \)

**DFA for \( L \)**

![DFA Diagram](image)

**Tabulated Function \( T(q, n) \)**

<table>
<thead>
<tr>
<th>States</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Unrank Algorithm

**Code**

```
UNRANK(r)
1 q ← q₀
2 n ← 0
3 while r ≥ T(q₀, n)
4 do r ← T(q₀, n)
5 n ← n + 1
6 x ← ϵ
7 for i ← 0 to n – 1
8 do j ← 1
9 while r ≥ T(δ(q, a_j), n – i – 1)
10 do r ← T(δ(q, a_j), n – i – 1)
11 j ← j + 1
12 x[i] ← a_j
13 q ← δ(q, x[i])
```

**Example:** unranking $r = 5$ for $L = (eh^+ | la)^+$

- $r = 1$
- $n = 4$
- $x = eh$

**DFA for $L$**

```
start → q₀
q₁
q₂
q₃
```

**Tabulated Function $T(q, n)$**

<table>
<thead>
<tr>
<th>States</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>q₀</td>
<td>0</td>
</tr>
<tr>
<td>q₁</td>
<td>0</td>
</tr>
<tr>
<td>q₂</td>
<td>1</td>
</tr>
<tr>
<td>q₃</td>
<td>0</td>
</tr>
</tbody>
</table>
Unrank Algorithm

Code

**UNRANK(r)**

1. $q \leftarrow q_0$
2. $n \leftarrow 0$
3. while $r \geq T(q_0, n)$
   4. do $r \leftarrow T(q_0, n)$
   5. $n \leftarrow 1$
6. $x \leftarrow \epsilon$
7. for $i \leftarrow 0$ to $n - 1$
   8. do $j \leftarrow 1$
   9. while $r \geq T(\delta(q, a_j), n - i - 1)$
   10. do $r \leftarrow T(\delta(q, a_j), n - i - 1)$
   11. $j \leftarrow 1$
12. $x[i] \leftarrow a_j$
13. $q \leftarrow \delta(q, x[i])$

Example: unranking $r = 5$ for $L = (eh^+|la)^+$

- $r = 1$
- $n = 4$
- $x = eh$

### DFA for $L$

![DFA diagram]

### Tabulated Function $T(q, n)$

<table>
<thead>
<tr>
<th>States</th>
<th>$0$</th>
<th>$1$</th>
<th>$2$</th>
<th>$3$</th>
<th>$4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>$q_1$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$q_2$</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>$q_3$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Unrank Algorithm

**Code**

UNRANK(r)

1. \( q \leftarrow q_0 \)
2. \( n \leftarrow 0 \)
3. while \( r \geq T(q_0, n) \)
   do \( r \leftarrow T(q_0, n) \)
   \( n \leftarrow 1 \)
4. \( x \leftarrow \epsilon \)
5. for \( i \leftarrow 0 \) to \( n - 1 \)
   do \( j \leftarrow 1 \)
   while \( r \geq T(\delta(q, a_j), n - i - 1) \)
   do \( r \leftarrow T(\delta(q, a_j), n - i - 1) \)
   \( j \leftarrow 1 \)
6. \( x[i] \leftarrow a_j \)
7. \( q \leftarrow \delta(q, x[i]) \)

**Example:** unranking \( r = 5 \) for \( L = (eh^+ | la)^+ \)

- \( r = 0 \)
- \( n = 4 \)
- \( x = eh \)

**DFA for L**

**Tabulated Function** \( T(q, n) \)

<table>
<thead>
<tr>
<th>States</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Unrank Algorithm

**Code**

```plaintext
UNRANK(r)
1 q ← q₀
2 n ← 0
3 while r ≥ T(q₀, n)
4 do r ← T(q₀, n)
5 n ← n + 1
6 x ← ε
7 for i ← 0 to n − 1
8 do j ← 1
9 while r ≥ T(δ(q, aᵢ), n − i − 1)
10 do r ← T(δ(q, aᵢ), n − i − 1)
11 j ← j + 1
12 x[i] ← aᵢ
13 q ← δ(q, x[i])
```

**Example:** unranking $r = 5$ for $L = (eh^+ | la)^+$

- $r = 0$
- $n = 4$
- $x = ehl$

**DFA for $L$**

**Tabulated Function $T(q, n)$**

<table>
<thead>
<tr>
<th>States</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₀</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>q₁</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>q₂</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>q₃</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Unrank Algorithm

**Code**

UNRANK(\(r\))

1. \(q \leftarrow q_0\)
2. \(n \leftarrow 0\)
3. **while** \(r \geq T(q_0, n)\)
   - **do** \(r \leftarrow T(q_0, n)\)
   - \(n \leftarrow n + 1\)
4. \(x \leftarrow \epsilon\)
5. **for** \(i \leftarrow 0\) **to** \(n - 1\)
   - **do** \(j \leftarrow 1\)
   - **while** \(r \geq T(\delta(q, a_j), n - i - 1)\)
     - **do** \(r \leftarrow T(\delta(q, a_j), n - i - 1)\)
     - \(j \leftarrow j + 1\)
   - \(x[i] \leftarrow a_j\)
6. \(q \leftarrow \delta(q, x[i])\)

**Example:** unranking \(r = 5\) for \(L = (eh^+|la)^+\)

- \(r = 0\)
- \(n = 4\)
- \(x = ehl\)

**DFA for L**

**Tabulated Function** \(T(q, n)\)

<table>
<thead>
<tr>
<th>States</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>(q_1)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>(q_2)</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>(q_3)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Unrank Algorithm

**Code**

**UNRANK**(\(r\))

1. \(q \leftarrow q_0\)
2. \(n \leftarrow 0\)
3. **while** \(r \geq T(q_0, n)\)
   
   **do**
   
   \(r \leftarrow T(q_0, n)\)
   
   \(n \leftarrow n + 1\)

4. \(x \leftarrow \epsilon\)
5. **for** \(i \leftarrow 0\) to \(n - 1\)
   
   **do** \(j \leftarrow 1\)

6. **while** \(r \geq T(\delta(q, a_j), n - i - 1)\)
   
   **do**
   
   \(r \leftarrow T(\delta(q, a_j), n - i - 1)\)

7. \(j \leftarrow j + 1\)
8. \(x[i] \leftarrow a_j\)
9. \(q \leftarrow \delta(q, x[i])\)

**Example:** unranking \(r = 5\) for

\[L = (eh^+|la)^+\]

- \(r = 0\)
- \(n = 4\)
- \(x = ehla\)

**DFA for \(L\)**

**Tabulated Function** \(T(q, n)\)

<table>
<thead>
<tr>
<th>States</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>(q_1)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>(q_2)</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>(q_3)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
From Regex To DFA

- Plaintext and ciphertext formats are regular expressions
- The ranking/unranking algorithms require a DFA
- How do we move from regular expressions to DFA?
From Regex To DFA

- Plaintext and ciphertext formats are regular expressions
- The ranking/unranking algorithms require a DFA
- How do we move from regular expressions to DFA?

\[ ba^*(c|d)a^+ \]  
Thompson algorithm  
N-DFA  
determinization  
DFA

\[ (a|b)^+ a(a|b)(a|b) \]  
Thompson algorithm  
N-DFA  
determinization  
DFA

For some languages, determinization of the N-DFA leads to an exponential blowup of the number of states.

To perform rank/unrank we need to precompute a table which is proportional to \(|Q|\).

For some languages, ranking is extremely expensive in terms of memory consumption!
From Regex To DFA

- Plaintext and ciphertext formats are regular expressions
- The ranking/unranking algorithms require a DFA
- How do we move from regular expressions to DFA?

\[ ba^*(c|d)a^+ \rightarrow \begin{array}{c} \text{Thompson algorithm} \\ \text{N-DFA} \end{array} \rightarrow \begin{array}{c} \text{determinization} \\ \text{DFA} \end{array} \]

\[ (a|b)^+ a(a|b)(a|b) \rightarrow \begin{array}{c} \text{Thompson algorithm} \\ \text{N-DFA} \end{array} \rightarrow \begin{array}{c} \text{determinization} \\ \text{DFA} \end{array} \]

- For some languages, determinization of the N- DFA leads to an exponential blowup of the number of states \(|\mathcal{Q}|\) of the DFA
- To perform rank/unrank we need to precompute a table which is proportional to \(|\mathcal{Q}|\)

For some languages, ranking is extremely expensive in terms of memory consumption!
Relaxed Ranking

What if the ranking is performed from the N-DFA representation?
Relaxed Ranking

What if the ranking is performed from the N-DFA representation?

\[ \Downarrow \]

Ranking from an N-DFA is PSPACE-hard! \(\equiv\) Each NP problem can be reduced to this problem!
Relaxed Ranking

What if the ranking is performed from the N-DFA representation?

⇓

Ranking from an N-DFA is PSPACE-hard! \( \equiv \) Each NP problem can be reduced to this problem!

Relaxed Ranking

We can still achieve our target by relaxing the ranking definition:

**Formal Definition:**

- \( \text{Rank} : L \mapsto \mathbb{Z}_i, i > |L| \) is injective
- \( \text{Unrank} : \mathbb{Z}_i \mapsto L \) is surjective
- \( \forall x \in L \quad \text{Unrank}(\text{Rank}(x)) = x \)

Ranking is a particular case of relaxed ranking where \( i = |L| \), thus \( \text{rank} \) and \( \text{unrank} \) are bijections.
Relaxed Ranking Design

Relaxed Ranking Idea

As with ranking, we can conceive the relaxed ranking as a 2 steps process

1. Map each string in the language to an accepting path in the corresponding N-DFA (parsing)

2. Rank (not relaxed) the set of all accepting paths
Relaxed Ranking Design

Relaxed Ranking Idea
As with ranking, we can conceive the relaxed ranking as a 2 steps process

1. Map each string in the language to an accepting path in the corresponding N-DFA (parsing)
2. Rank (not relaxed) the set of all accepting paths

- Since there is a bijection between the set of accepting paths $\mathcal{P}$ and $\mathbb{Z}_i \rightarrow i = |\mathcal{P}|$
- In N-DFA, more than one accepting path for each string
- Therefore, we need to map a string to one of its accepting paths
Relaxed Ranking Design

Relaxed Ranking Idea

As with ranking, we can conceive the relaxed ranking as a 2 steps process

1. Map each string in the language to an accepting path in the corresponding N-DFA (parsing)

2. Rank (not relaxed) the set of all accepting paths

Since there is a bijection between the set of accepting paths $\mathcal{P}$ and $\mathbb{Z}_i \rightarrow i = |\mathcal{P}|$

In N-DFA, more than one accepting path for each string

Therefore, we need to map a string to one of its accepting paths

Map function

The relaxed ranking requires 2 functions

- $map : L \mapsto \mathcal{P}$, which is injective
- $unmap : \mathcal{P} \mapsto L$, which is surjective

Moreover, $\forall x \in L \quad unmap(map(x)) = x$
Relaxed Ranking Overview

$P$

$\mathbb{Z}_i$

$L$

$x_1$

$x_2$

map

unmap

rank

unrank

Rank

Unrank
$map \equiv$ deterministically parse the string $\rightarrow$ possibly exponential time!
Map Function

\( \text{map} \equiv \) deterministically parse the string \( \rightarrow \) possibly exponential time!

Linear Time Parsing

Idea: 2 phases parsing

1. Build a graph which represents all the accepting paths for a string \( x \in L \)
2. Traverse this graph by choosing the lexicographically least one (fixing an order for the set \( Q \))
**Map Function**

\( map \equiv \) deterministically parse the string \( \rightarrow \) possibly exponential time!

**Linear Time Parsing**

**Idea:** 2 phases parsing

1. Build a graph which represents all the accepting paths for a string \( x \in L \)
2. Traverse this graph by choosing the lexicographically least one (fixing an order for the set \( Q \))

**Building the Graph**

- In a N-DFA, the transition function is defined as \( \delta : Q \times \Sigma \mapsto P(Q) \)
- When we parse the \( k \)-th character of string \( x \), we define a frontier \( \mathcal{F}_k \subseteq Q = \bigcup_{q \in \mathcal{F}_{k-1}} \delta(q, x[k-1]) \), with the base case \( \mathcal{F}_0 = \{q_0\} \)
- \( x \in L \iff \mathcal{F}_{|x|} \cap \mathcal{F} \neq \emptyset \)
- The frontiers \( \mathcal{F}_k \) contains all the possible states reachable from the initial one after parsing of \( k \) characters of \( x \)

But we need only the states belonging to accepting paths!
Map Function: Graph Building

Pruning Non-Accepting Paths

- We can enrich the information provided by the frontiers with backward frontiers.
- A backward frontier $B_k \subseteq Q = \{ q \mid \exists q' \in B_{k+1}(\delta(q, x[k]) = q') \}$, with the base case $B_{|x|} = \emptyset$.
- A frontier $B_k$ represents the states which belongs to an accepting path for the substring $x[k \ldots n-1]$, starting from any state of $B_k$.
- Now, define the frontier $S_k = F_k \cap B_k \rightarrow$ these are the states which are reachable from the initial one after parsing the first $k$ characters of $x$ which leads to an accepting path for the substring $x[k \ldots n-1]$.
Pruning Non-Acceptor Paths

- We can enrich the information provided by the frontiers with backward frontiers.

- A backward frontier $B_k \subseteq Q = \{ q \mid \exists q' \in B_{k+1}(\delta(q, x[k]) = q') \}$, with the base case $B_{|x|} = \emptyset$.

- A frontier $B_k$ represents the states which belong to an accepting path for the substring $x[k \ldots n - 1]$, starting from any state of $B_k$.

- Now, define the frontier $S_k = F_k \cap B_k \rightarrow$ these are the states which are reachable from the initial one after parsing the first $k$ characters of $x$ which leads to an accepting path for the substring $x[k \ldots n - 1]$.

Building the Graph

- Each frontier $S_k$ becomes a layer of the graph.

- Each layer is made of the states $q \in S_k$.

- Each state $q$ is connected to the states of the subsequent layer $q' \in \delta(q, x[k])$. 

Graph building for \( x = 1110111 \)

N-DFA for \( L = (0 \mid 1)^*111 \)

\[
\begin{array}{c|c}
 k & F_k \\ \hline
 0 & q_0 \\ 1 & q_0, q_1 \\ 2 & q_0, q_1, q_2 \\ 3 & q_0, q_1, q_2, q_f \\ 4 & q_0 \\ 5 & q_0, q_1 \\ 6 & q_0, q_1, q_2 \\ 7 & q_0, q_1, q_2, q_f \\
\end{array}
\]

\[
\begin{array}{c|c|c}
 k & B_k & S_k \\ \hline
 0 & q_0 & q_0 \\ 1 & q_0, q_1 & q_0, q_1 \\ 2 & q_0, q_2 & q_0, q_2 \\ 3 & q_0, q_f & q_0, q_f \\ 4 & q_0 & q_0 \\ 5 & q_1 & q_1 \\ 6 & q_2 & q_2 \\ 7 & q_f & q_f \\
\end{array}
\]
Graph Building

Graph building for \( x = 1110111 \)

**N-DFA for \( L = (0 \ | \ 1)^*111 \)**

```
<table>
<thead>
<tr>
<th>k</th>
<th>( \mathcal{F}_k )</th>
<th>( \mathcal{B}_k )</th>
<th>( S_k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( q_0 )</td>
<td>( q_0 )</td>
<td>( q_0 )</td>
</tr>
<tr>
<td>1</td>
<td>( q_0, q_1 )</td>
<td>( q_0, q_1 )</td>
<td>( q_0, q_1 )</td>
</tr>
<tr>
<td>2</td>
<td>( q_0, q_1, q_2 )</td>
<td>( q_0, q_2 )</td>
<td>( q_0, q_2 )</td>
</tr>
<tr>
<td>3</td>
<td>( q_0, q_1, q_2, q_f )</td>
<td>( q_0, q_f )</td>
<td>( q_0, q_f )</td>
</tr>
<tr>
<td>4</td>
<td>( q_0 )</td>
<td>( q_0 )</td>
<td>( q_0 )</td>
</tr>
<tr>
<td>5</td>
<td>( q_0, q_1 )</td>
<td>( q_1 )</td>
<td>( q_1 )</td>
</tr>
<tr>
<td>6</td>
<td>( q_0, q_1, q_2 )</td>
<td>( q_2 )</td>
<td>( q_2 )</td>
</tr>
<tr>
<td>7</td>
<td>( q_0, q_1, q_2, q_f )</td>
<td>( q_f )</td>
<td>( q_f )</td>
</tr>
</tbody>
</table>
```
Completing Relaxed Ranking

Unmap

- Performing \textit{unmap} means deriving the string $x \in L$ from an accepting path $\pi \in \mathcal{P}$
- This is straightforward: just follow the transitions of the path and concatenate their characters
- \textit{unmap} is trivially linear in $|x|$
Completing Relaxed Ranking

Unmap

- Performing \textit{unmap} means deriving the string $x \in L$ from an accepting path $\pi \in \mathcal{P}$
- This is straightforward: just follow the transitions of the path and concatenate their characters
- \textit{unmap} is trivially linear in $|x|$

Ranking on Accepting Paths

- In classical ranking, the 2 steps of parsing and computing the rank are done at the same time
- However, the actual ranking is already computed over accepting paths
- Therefore, we can employ the same ranking/unranking procedures, with slight modifications, to rank directly on N-DFA accepting paths
Inside the FTE Box

- N-DFA have usually less states than DFA
- Conversion from regex is also faster, since no determinization is involved
- Rank and Unrank are a bit more complex
Invalid Encryptions

\[ L \rightarrow \mathcal{P} \]

\[ \mathcal{P} \rightarrow \mathbb{Z}_i \]

\[ \text{map} \]

\[ \text{unmap} \]

\[ \text{rank} \]

\[ \text{unrank} \]

\[ x_1 \rightarrow \mathcal{P} \rightarrow \mathbb{Z}_i \]

\[ x_2 \rightarrow \mathcal{P} \rightarrow \mathbb{Z}_i \]

15th Semptember 2017
Nicholas Mainardi
Invalid Encryptions

A Further Issue

- Generally, it is unlikely that $\mathbb{Z}_i$ is the domain of the encryption algorithm.
- It is expected that the ciphertext domain is a superset of $\mathbb{Z}_i$.
- Example: if you use AES-128, the ciphertext domain is $\mathbb{Z}_{2^{128}}$, but likely $i < 2^{128}$.
- Therefore, an encryption $c \notin \mathbb{Z}_i$ is an invalid encryption.

How to deal with invalid encryptions? We can repeat the encryption until you get a valid ciphertext!
Determine Invalid Encryptions

**Ranking Image**

- In case of relaxed ranking, the ranking image for a language $L$ is the set $\mathcal{I}(L) = \{ z \in \mathbb{Z}_i | \text{Rank}_L(\text{Unrank}_L(z)) = z \}$
- Since $\text{Rank}$ is injective, $|L| = |\mathcal{I}(L)|$
- Note that these images are likely disjoint intervals of integers in $\mathbb{Z}_i$
Determine Invalid Encryptions

**Ranking Image**
- In case of relaxed ranking, the ranking image for a language $L$ is the set $\mathcal{I}(L) = \{ z \in \mathbb{Z}_i \mid \text{Rank}_L(\text{Unrank}_L(z)) = z \}$
- Since $\text{Rank}$ is injective, $|L| = |\mathcal{I}(L)|$
- Note that these images are likely disjoint intervals of integers in $\mathbb{Z}_i$

**Testing Membership to Ranking Images**
- Testing if $z \in \mathbb{Z}_i$ is trivial
- How do we test that the encryption output $z \in \mathbb{Z}_i$ is in a ranking image $\mathcal{I}(L)$?
- Recall that $z \in \mathcal{I}(L) \iff \text{Rank}_L(\text{Unrank}_L(z)) = z$
- More efficient way: $\text{map}_L(\text{Unrank}_L(z)) = \text{unrank}_L(z)$
Credit Card Numbers

Suppose we have a database with a table containing a column storing credit card numbers.

- We want to store them encrypted, but without requiring additional memory
- In-place encryption can be achieved with Format Preserving Encryption (FPE)
- Input format = Output format = [0 – 9]^{16}
Applications: In-Place Encryption

Credit Card Numbers

Suppose we have a database with a table containing a column storing credit card numbers.

- We want to store them encrypted, but without requiring additional memory
- In-place encryption can be achieved with Format Preserving Encryption (FPE)
- Input format = Output format = \([0 − 9]\{16\}\)

Compression?

- To represent a decimal digits, the DBMS employ at least 4 bits
- For 16 digits → 64 bits required
- However, to represent all possible integers of 16 digits, 54 bits are sufficient → 7 bytes are enough
- Therefore, we can encrypt using FTE where the input format = \([0 − 9]\{16\}\) and output format = \([0 − 255]\{7\}\)
In-Place Encryption: Performance

Testing Environment

Testing Platform: PostgreSQL 9.1 on Ubuntu 12.04 equipped with Intel Xeon E3 – 1220 v3 @ 3.10GHz and 32GB RAM

For performance comparison, 3 different configurations (apart from FPE and FTE) are employed

1. **PSQL**: The default PostgreSQL configuration → no encryption is involved
2. **AES**: The data is encrypted using AES-128 in ECB mode
3. **AE**: The data is encrypted with an authenticated encryption scheme (recommended scenario for PostgreSQL)
In-Place Encryption: Performance

Testing Environment

Testing Platform: PostgreSQL 9.1 on Ubuntu 12.04 equipped with Intel Xeon E3 – 1220 v3 @ 3.10GHz and 32GB RAM

For performance comparison, 3 different configurations (apart from FPE and FTE) are employed

1. **PSQL**: The default PostgreSQL configuration → no encryption is involved
2. **AES**: The data is encrypted using AES-128 in ECB mode
3. **AE**: The data is encrypted with an authenticated encryption scheme (recommended scenario for PostgreSQL)

Performances

<table>
<thead>
<tr>
<th></th>
<th>PSQL</th>
<th>AES</th>
<th>AE</th>
<th>FPE</th>
<th>FTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table size(MB)</td>
<td>50</td>
<td>65</td>
<td>112</td>
<td>50</td>
<td>42</td>
</tr>
<tr>
<td>Query avg. Time(ms)</td>
<td>74</td>
<td>92</td>
<td>112</td>
<td>125</td>
<td>110</td>
</tr>
</tbody>
</table>

- Reducing size but comparable performances with AE scheme
- Approximately 20% performance overhead against 35% size reduction w.r.t. AES

15th September 2017 Nicholas Mainardi

POLITECNICO DI MILANO Dipartimento di Elettronica, Informazione e Bioingegneria
Deep Packet Inspection

- A deep packet inspector (DPI) filters the network traffic based on the protocol or on the content of the packets
- The protocols allowed by the DPI are called target protocols
Deep Packet Inspection

- A deep packet inspector (DPI) filters the network traffic based on the protocol or on the content of the packets.
- The protocols allowed by the DPI are called target protocols.

### Existing Inspectors

<table>
<thead>
<tr>
<th>Name</th>
<th>Classifier</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>appid</td>
<td>regex + port filtering</td>
<td>Free</td>
</tr>
<tr>
<td>L7-filter</td>
<td>regex</td>
<td>Free</td>
</tr>
<tr>
<td>YAF</td>
<td>regex</td>
<td>Free</td>
</tr>
<tr>
<td>Bro</td>
<td>regex + C code to check false positives</td>
<td>Free</td>
</tr>
<tr>
<td>Nprobe</td>
<td>regex + flow tracking</td>
<td>300 €</td>
</tr>
<tr>
<td>DPI-X</td>
<td>??</td>
<td>10k €</td>
</tr>
</tbody>
</table>

- DPI-X is a proprietary system, but the company did not authorize publication of the name.
- It achieves high throughput inspection (1.5 Gbps), but the classifier is unknown.
- However, it has been identified as the device employed by Iran for censorship purposes, thus it is a good device to be tested.
Protocol Misclassification Attacks

**Attack Scenario**

The attack is successful if the DPI is fooled misclassifying the https as the target protocol.
Protocol Misclassification Attacks

**Attack Scenario**

The attack is successful if the DPI is fooled misclassifying the https as the target protocol.

2 main results:
- Attack succeeds against all the tested DPI (even DPI-X)
- Performance: using FTE tunnel ≈ using SSH tunnel
Defeating the Great Firewall of China (GFC)

Chinese government uses DPI for censorship purposes!

Censorship

Connections to Facebook or Youtube are usually dropped. What about Tor?

- It is possible to use Tor, but the GFC is able to actively probe the Tor Onion Proxy (OP) and understand that it is a Tor node
- Then, the GFC drops the connection by sending RST TCP packets
Defeating the Great Firewall of China (GFC)

Chinese government uses DPI for censorship purposes!

**Censorship**

Connections to Facebook or Youtube are usually dropped. What about Tor?

- It is possible to use Tor, but the GFC is able to actively probe the Tor Onion Proxy (OP) and understand that it is a Tor node
- Then, the GFC drops the connection by sending RST TCP packets

**Defeating Censorship**

- **Defeating website censorships**: With FTE, we can map HTTPS packets for Facebook to random HTTP packet
- **Defeating Tor censorship**: The main issue which triggers actively probing by the GFC is the usage of HTTPS to communicate with Tor OP

⇒ We can use FTE to communicate with the Tor OP!
On the Field Testing

- The *FTE* client has been established on a Virtual Private Server in China

- The tunnel remained active for 1 month browsing blocked websites and using Tor

- No trace of detection in the logs

15th September 2017
Nicholas Mainardi
POLITECNICO DI MILANO
Dipartimento di Elettronica, Informazione e Bioingegneria
The FTE client has been established on a Virtual Private Server in China.

The tunnel remained active for 1 month browsing blocked websites and using Tor.
On the Field Testing

- The *FTE* client has been established on a Virtual Private Server in China
- The tunnel remained active for 1 month browsing blocked websites and using Tor
- No trace of detection in the logs
Countermeasures

Active Probing

- The GFC may actively probe the FTE proxy
- However, differently from the Tor OP case, plain HTTP is used
- Therefore, the GFC should actively probe on HTTP connections too in order to detect FTE → high complexity or negligible probability of being caught
Active Probing

- The GFC may actively probe the FTE proxy
- However, differently from the Tor OP case, plain HTTP is used
- Therefore, the GFC should actively probe on HTTP connections too in order to detect FTE → high complexity or negligible probability of being caught

Employing Distribution of HTTP packets

- FTE outputs a random HTTP packet
- Therefore, the distributions of HTTP packets on a tunneled FTE flow or in a normal flow are actually distinguishable
- The DPI can thus set up a distinguisher which is able to spot FTE tunnel
- However, it is still an open problem performing this detection in a scalable way and with negligible overhead for the DPI settings
Further Developments

- Countermeasures based on distinguishing packets distributions is a serious threat to FTE usage against DPI!
- Can’t we extend FTE such that the format is no longer a regex but a probability distribution?
Further Developments

- Countermeasures based on distinguishing packets distributions is a serious threat to FTE usage against DPI!
- Can’t we extend FTE such that the format is no longer a regex but a probability distribution?

An Improved FTE Box

Main challenge: How to invertibly sample from a generic distribution using uniform bits $\equiv$ Encode box?
Conclusions

Take Home Messages

- Format Transforming Encryption (FTE) allows to get ciphertexts abiding to a specified format, being able to recover the plaintext in its format.
- Format Preserving Encryption (FPE) is a particular case of FTEw, where the ciphertext format is the same as the plaintext one.
- FTE is based on ranking/unranking algorithms for regular languages.
- To enlarge the set of language which can be efficiently ranked, the relaxed ranking primitive has been introduced.
- Relaxed ranking employs the first linear time parsing algorithm for N-DFA.
- FPE/FTE are suitable for in-place encryption scenario, achieving a good tradeoff between performance and size reduction.
- FTE can successfully be used for protocol misclassification attacks, even against enterprise grade DPI.
- A FTE tunnel can be used to circumvent Chinese censorship performed by the Great Firewall of China.

Want to try it?: https://github.com/kpdyer/libfte
Questions

plaintext → brain neurosignals format → [A – Z|a – z|0 – 9]" " ] + "?

key → FTE

question
Invalid Encryptions: Cycle Walking

Workaround for Unrankable Ciphertexts

- In a deterministic cipher \( c = E(k, p) \), \( c \) is constant for the same \( k, p \) values.
- If \( c \) is not a valid ciphertext, we can generate a new ciphertext by computing \( c_1 = E(k, c) \).
- Iterating this process, we get either back to \( p \) or to a valid ciphertext.
- This procedure is called \textit{cycle walking}.
Invalid Encryptions: Cycle Walking

**Cycle Walking Decryption**

- When we decrypt, we perform reverse cycle walking, getting values $p_i$, to retrieve the original plaintext $p$.
- We retrieve $p$ when we get a $p_i \in I(L(R_p))$.
- Unfortunately, if there is a $p_k \in I(L(R_p))$ such that $k > i$, then $p = p_k \rightarrow$ decryption is wrong.
- Since this situation is not detectable during decryption, we need to modify encryption.
- Hence, during encryption, if $c_i \in I(L(R_p)) \land c_i \notin I(L(R_c)) \rightarrow$ decryption will fail, thus encryption is impossible!
$ ./configuration-assistant \
> --input-format "(a|b)*a(a|b){16}" 0 64 \
> --output-format "[0-9a-f]{16}" 0 16

==== Identifying valid schemes ====
No valid schemes.
ERROR: Input language size greater than output language size.
$

OR

$ ./configuration-assistant \
> --input-format "(a|b)*a(a|b){16}" 0 32 \
> --output-format "[0-9a-f]{16}" 0 16

==== Identifying valid schemes ====
WARNING: Memory threshold exceeded when building DFA for input format
VALID SCHEMES: T-ND, T-NN,
               T-ND-$, T-NN-$

==== Evaluating valid schemes ====
SCHEME  ENCRYPT  DECRYPT  ...  MEMORY
T-ND    0.32ms  0.31ms  ...  77KB
T-NN    0.39ms  0.38ms  ...  79KB
...
$