POLITECNICO DI MILANO

Dipartimento di
Elettronica, Informazione
e Bioingegneria

Format Transforming Encryption:

More than Meets the DPI

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

15th Semptember 2017

Credits & Outline

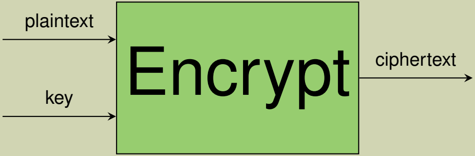
This talk is mainly based on the seminars of Tom Shrimtpon (Florida Institute for Cybersecurity Research) at Real World Crypto and Privacy summer school (Sibenik 5-9th June 2017)

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

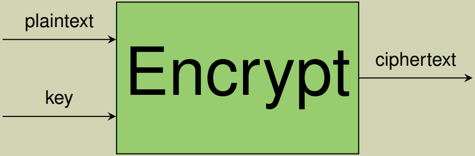
Traditional Encryption



Plaintexts and ciphertexts are bit strings!

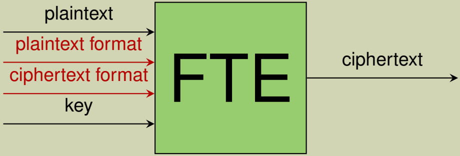
Traditional Encryption vs FTE

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

Format Definition

How Do We Define a Format?

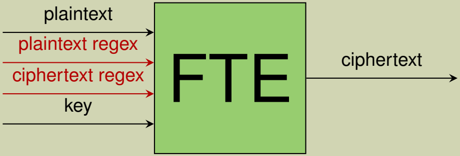
- A format is a set of rules which defines the structure of the plain/ciphertexts
- Set of rules $\rightarrow$ a grammar which defines the syntax of the messages
- That is, a format basically defines a language
- A rather intuitive and user friendly way of specifying a language is a regular expression

Format Definition

How Do We Define a Format?

- A format is a set of rules which defines the structure of the plain/ciphertexts
- Set of rules $\rightarrow$ 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

Format Transforming Encryption



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

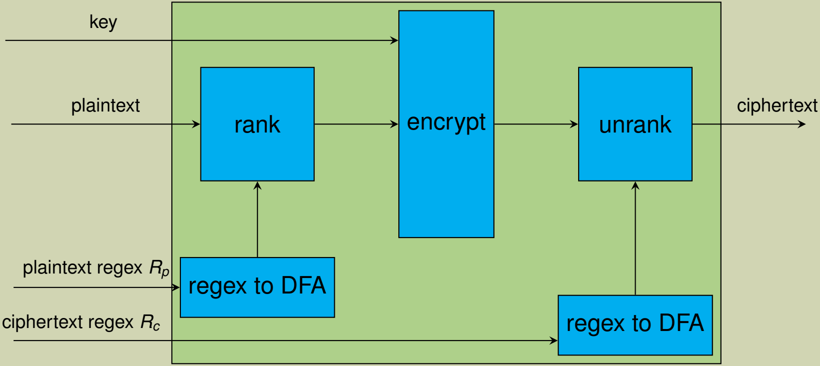
Formal definition:

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

Once we have a map to $\mathbb{Z}_{|L|}$, we can use whatever cipher to encrypt the plaintext!

Ranking Based FTE

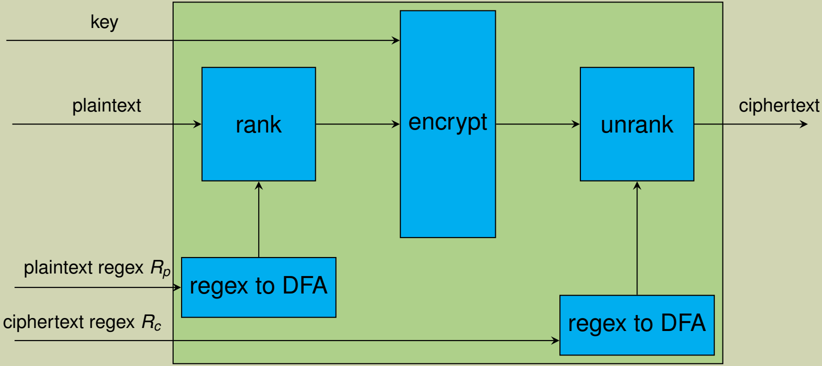
Inside the FTE Box



- 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?

Ranking Based FTE

Inside the FTE Box



- 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!

Ranking and Unranking

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 $|\mathbb{Q}| \cdot max_{x \in L}(|X|)$ is employed
- With this precomputed table, ranking and unranking are $O(|X|)$

Precomputation

DFA Notation

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

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

Precomputation

DFA Notation

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

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

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

Precomputation

The function can be computed in a recursive fashion:

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

Precomputation

The function can be computed in a recursive fashion:

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

We can write a simple procedure to compute this function:

Table Precomputation

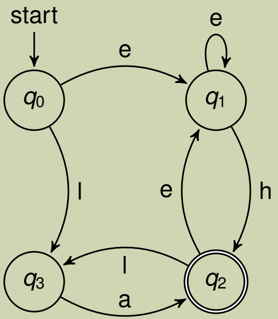
BUILD-TABLE(n)

for $q \in \mathbb{Q}$
  do if $q \in F$
    then $T(q,0) \leftarrow 1$
for $i \leftarrow 1$ **to** n
  do for $q \in \mathbb{Q}$
    do for $a \in \Sigma$
      do $T(q,i) \xleftarrow{+} T(\delta(q,a), i-1)$

Precomputation Example

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

DFA for L



Code

BUILD-TABLE(n)

for $q \in \mathbb{Q}$
  do if $q \in F$
    then $T(q, 0) \leftarrow 1$
for $i \leftarrow 1$ **to** n
  do for $q \in \mathbb{Q}$
    do for $a \in \Sigma$
      do $T(q, i) \xleftarrow{+} T(\delta(q, a), i - 1)$

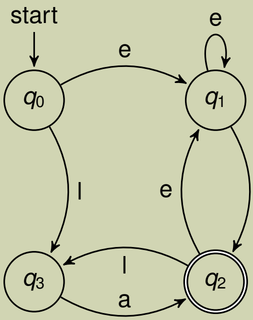
Tabulated Function $T(q, n)$

States	n				
	0	1	2	3	4
q_0					
q_1					
q_2					
q_3					

Precomputation Example

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

DFA for L



Code

BUILD-TABLE(n)

for $q \in \mathbb{Q}$
  do if $q \in F$
    then $T(q, 0) \leftarrow 1$
for $i \leftarrow 1$ **to** n
  do for $q \in \mathbb{Q}$
    do for $a \in \Sigma$
      do $T(q, i) \overset{+}{\leftarrow} T(\delta(q, a), i - 1)$

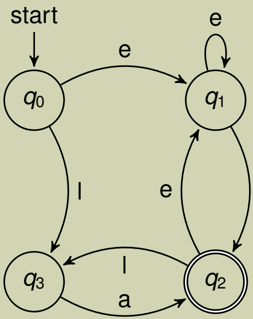
Tabulated Function $T(q, n)$

States	n				
	0	1	2	3	4
q_0	0				
q_1	0				
q_2	1				
q_3	0				

Precomputation Example

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

DFA for L



Code

BUILD-TABLE(n)

$$
\begin{array}{l}
\textbf{for } q \in \mathbb{Q} \\
\quad \textbf{do if } q \in F \\
\qquad \textbf{then } T(q,0) \leftarrow 1 \\
\textbf{for } i \leftarrow 1 \textbf{ to } n \\
\quad \textbf{do for } q \in \mathbb{Q} \\
\qquad \textbf{do for } a \in \Sigma \\
\qquad\quad \textbf{do } T(q,i) \xleftarrow{+} T(\delta(q,a), i-1)
\end{array}
$$

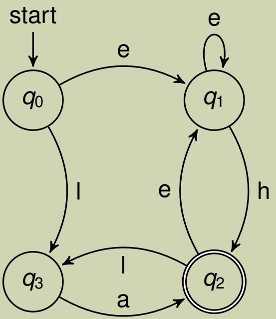
Tabulated Function $T(q,n)$

States	n = 0	1	2	3	4
q_0	0	0			
q_1	0	1			
q_2	1	0			
q_3	0	1			

Precomputation Example

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

DFA for L



Code

BUILD-TABLE(n)

for $q \in \mathbb{Q}$
  do if $q \in F$
    then $T(q, 0) \leftarrow 1$
for $i \leftarrow 1$ **to** n
  do for $q \in \mathbb{Q}$
    do for $a \in \Sigma$
      do $T(q, i) \xleftarrow{+} T(\delta(q, a), i - 1)$

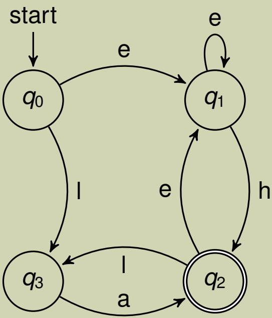
Tabulated Function $T(q, n)$

States	n = 0	1	2	3	4
q_0	0	0	2		
q_1	0	1	1		
q_2	1	0	2		
q_3	0	1	0		

Precomputation Example

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

DFA for L



Code

BUILD-TABLE(n)

for $q \in \mathbb{Q}$
  do if $q \in F$
    then $T(q, 0) \leftarrow 1$
for $i \leftarrow 1$ **to** n
  do for $q \in \mathbb{Q}$
    do for $a \in \Sigma$
      do $T(q, i) \xleftarrow{+} T(\delta(q, a), i - 1)$

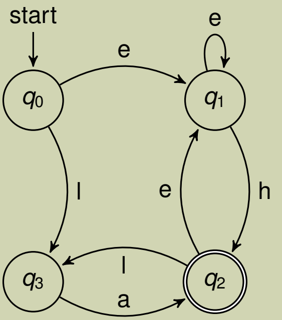
Tabulated Function $T(q, n)$

States	n = 0	1	2	3	4
q_0	0	0	2	1	
q_1	0	1	1	3	
q_2	1	0	2	1	
q_3	0	1	0	2	

Precomputation Example

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

DFA for L



Code

BUILD-TABLE(n)

for $q \in \mathbb{Q}$
    do if $q \in F$
        then $T(q,0) \leftarrow 1$
for $i \leftarrow 1$ **to** n
    do for $q \in \mathbb{Q}$
        do for $a \in \Sigma$
            do $T(q,i) \xleftarrow{+} T(\delta(q,a), i-1)$

Tabulated Function $T(q,n)$

States	n = 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

Rank Algorithm

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

Rank Algorithm

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 $T(q, n)$:

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

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

Rank Algorithm

Rank Algorithm

RANK(x)

$q \leftarrow q_0$
$rank \leftarrow 0$
for $i \leftarrow 0$ **to** $|x| - 1$
    do for $a \in \Sigma$
        do if $a \geq x[i]$
            then continue
          $rank \stackrel{+}{\leftarrow} T(\delta(q, a), |x| - i - 1)$
      $q \leftarrow \delta(q, x[i])$
      $rank \stackrel{+}{\leftarrow} T(q_0, i)$

Rank Algorithm

Rank Algorithm

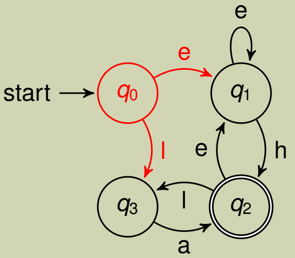
RANK(x)

```
q ← q_0
rank ← 0
for i ← 0 to |x| − 1
    do for a ∈ Σ
        do if a ≥ x[i]
            then continue
            rank +← T(δ(q, a), |x| − i − 1)
    q ← δ(q, x[i])
    rank +← T(q_0, i)
```

Example for $x = ehla$:

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

DFA for L



Tabulated Function $T(q, n)$

States	n = 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

Rank Algorithm

Rank Algorithm

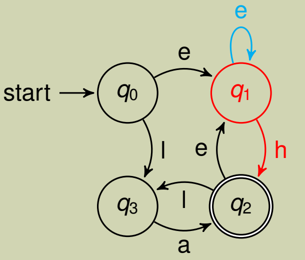
RANK(x)

$q \leftarrow q_0$
$rank \leftarrow 0$
for $i \leftarrow 0$ **to** $|x| - 1$
    do for $a \in \Sigma$
        do if $a \geq x[i]$
            then continue
          $rank \xleftarrow{+} T(\delta(q, a), |x| - i - 1)$
      $q \leftarrow \delta(q, x[i])$
      $rank \xleftarrow{+} T(q_0, i)$

Example for $x = ehla$:

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

DFA for L



Tabulated Function $T(q, n)$

States	n = 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

Rank Algorithm

Rank Algorithm

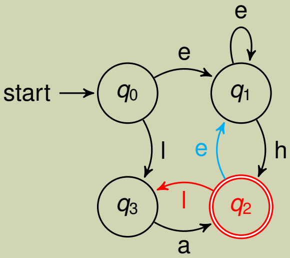
RANK(x)

$q \leftarrow q_0$
$rank \leftarrow 0$
for $i \leftarrow 0$ **to** $|x| - 1$
  do for $a \in \Sigma$
    do if $a \geq x[i]$
      then continue
      $rank \overset{+}{\leftarrow} T(\delta(q, a), |x| - i - 1)$
  $q \leftarrow \delta(q, x[i])$
  $rank \overset{+}{\leftarrow} T(q_0, i)$

Example for $x = ehla$:

- $i = 2$
- $x[i] = l$
- $rank = 4$

DFA for L



Tabulated Function $T(q, n)$

States	n: 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

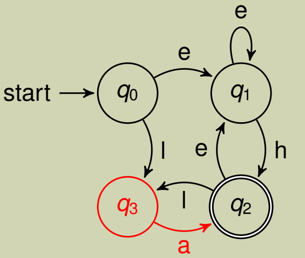
Rank Algorithm

Rank Algorithm

RANK(x)

$q \leftarrow q_0$
$rank \leftarrow 0$
for $i \leftarrow 0$ **to** $|x| - 1$
  do for $a \in \Sigma$
    do if $a \geq x[i]$
      then continue
      $rank \xleftarrow{+} T(\delta(q, a), |x| - i - 1)$
  $q \leftarrow \delta(q, x[i])$
  $rank \xleftarrow{+} T(q_0, i)$

DFA for L



Example for $x = ehla$:

- $i = 3$
- $x[i] = a$
- $rank = 5$

Tabulated Function $T(q, n)$

States	n 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

Rank Algorithm

Rank Algorithm

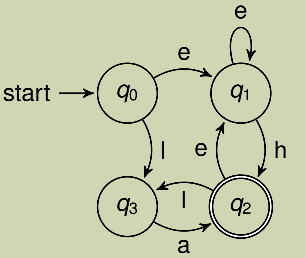
RANK(x)

```
q ← q_0
rank ← 0
for i ← 0 to |x| − 1
      do for a ∈ Σ
            do if a ≥ x[i]
                  then continue
                rank +← T(δ(q, a), |x| − i − 1)
         q ← δ(q, x[i])
         rank +← T(q_0, i)
```

Example for $x = ehla$:

$rank = 5 \rightarrow$ There are 5 strings less than x: *eh*, *la*, *eeh*, *eeeh*, *eheh*

DFA for L



Tabulated Function $T(q, n)$

States	n = 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

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 $n = |x|$):

$$
\begin{aligned}
r = {} & |\{s : |s| = 0\}| + |\{s : |s| = 1\}| + \cdots + |\{s : |s| = n-1\}| \\
& + |\{s : |s| = n \wedge s[0] < x[0]\}| + |\{s : |s| = n \wedge s[1] < x[1] \wedge s[0] = x[0]\}| \\
& + |\{s : |s| = n \wedge s[2] < x[2] \wedge s[0 \ldots 1] = x[0 \ldots 1]\}| + \cdots + \\
& + |\{s : |s| = n \wedge s[n-1] < x[n-1] \wedge s[0 \ldots n-2] = x[0 \ldots n-2]\}|
\end{aligned}
$$

- 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 $\mathbf{n} = |\mathbf{x}|$

- $|\{s : |s| = 0\}| = T(q_0, 0) \wedge |\{s : |s| = 1\}| = t(q_0, 1) \ldots$
- 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

Unrank Algorithm: Exploiting Partitions

Find $\mathbf{n} = |\mathbf{x}|$

- $|\{s : |s| = 0\}| = T(q_0, 0) \wedge |\{s : |s| = 1\}| = t(q_0, 1) \ldots$
- 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 $r = r - \sum_{i=0}^{n-1} T(q_0, i)$
- $|\{s : |s| = n \wedge s[0] < x[0]\}| = \sum_{\substack{a \in \Sigma \\ a < x[0]}} T(\delta(q_0, a), n - 1)$
- 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 $n = |x|$

- $|\{s : |s| = 0\}| = T(q_0, 0) \wedge |\{s : |s| = 1\}| = t(q_0, 1) \ldots$
- 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 $r = r - \sum_{i=0}^{n-1} T(q_0, i)$
- $|\{s : |s| = n \wedge s[0] < x[0]\}| = \sum_{\substack{a \in \Sigma \\ a < x[0]}} T(\delta(q_0, a), n-1)$
- 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 $r = r - \sum_{\substack{a \in \Sigma \\ a < x[0]}} T(\delta(q_0, a), n-1)$
- 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)

```
q ← q_0
n ← 0
while r ≥ T(q_0, n)
      do r ←(−) T(q_0, n)
         n ←(+) 1
x ← ε
for i ← 0 to n − 1
      do j ← 1
         while r ≥ T(δ(q, a_j), n − i − 1)
               do r ←(−) T(δ(q, a_j), n − i − 1)
                  j ←(+) 1
         x[i] ← a_j
         q ← δ(q, x[i])
```

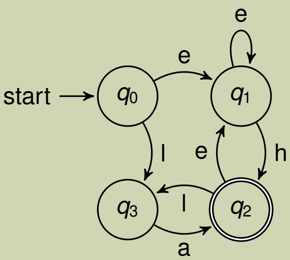
Unrank Algorithm

Code

UNRANK(r)

$q \leftarrow q_0$
$n \leftarrow 0$
while $r \geq T(q_0, n)$
    do $r \xleftarrow{-} T(q_0, n)$
        $n \xleftarrow{+} 1$
$x \leftarrow \epsilon$
for $i \leftarrow 0$ **to** $n - 1$
    do $j \leftarrow 1$
        while $r \geq T(\delta(q, a_j), n - i - 1)$
            do $r \xleftarrow{-} T(\delta(q, a_j), n - i - 1)$
                $j \xleftarrow{+} 1$
        $x[i] \leftarrow a_j$
        $q \leftarrow \delta(q, x[i])$

DFA for L



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

- $r = 5$
- $n = 0$
- $x = \epsilon$

Tabulated Function $T(q, n)$

States	n = 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

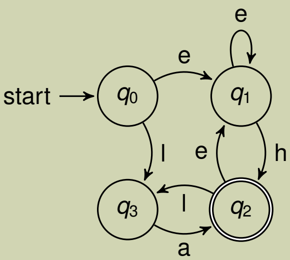
Unrank Algorithm

Code

UNRANK(r)

```
q ← q_0
n ← 0
while r ≥ T(q_0, n)
      do r ←(−) T(q_0, n)
         n ←(+) 1
x ← ε
for i ← 0 to n − 1
      do j ← 1
         while r ≥ T(δ(q, a_j), n − i − 1)
               do r ←(−) T(δ(q, a_j), n − i − 1)
                  j ←(+) 1
         x[i] ← a_j
         q ← δ(q, x[i])
```

DFA for L



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

- $r = 5$
- $n = 0$
- $x = \epsilon$

Tabulated Function $T(q, n)$

States	n = 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

Unrank Algorithm

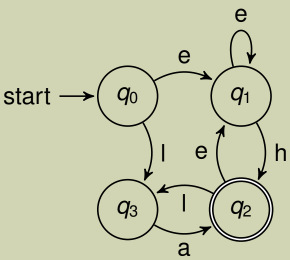
Code

```
UNRANK(r)
q ← q0
n ← 0
while r ≥ T(q0, n)
      do r −← T(q0, n)
         n +← 1
x ← ε
for i ← 0 to n − 1
      do j ← 1
         while r ≥ T(δ(q, a_j), n − i − 1)
               do r −← T(δ(q, a_j), n − i − 1)
                  j +← 1
         x[i] ← a_j
         q ← δ(q, x[i])
```

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

- $r = 5$
- $n = 1$
- $x = \epsilon$

DFA for L



Tabulated Function $T(q, n)$

States	n = 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

Unrank Algorithm

Code

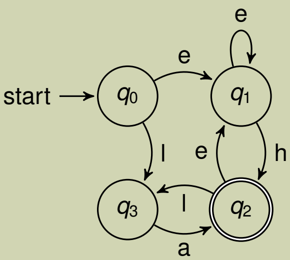
$\textsc{Unrank}(r)$

$q \leftarrow q_0$
$n \leftarrow 0$
while $r \geq T(q_0, n)$
  do $r \overset{-}{\leftarrow} T(q_0, n)$
    $n \overset{+}{\leftarrow} 1$
$x \leftarrow \epsilon$
for $i \leftarrow 0$ **to** $n - 1$
  do $j \leftarrow 1$
    while $r \geq T(\delta(q, a_j), n - i - 1)$
      do $r \overset{-}{\leftarrow} T(\delta(q, a_j), n - i - 1)$
        $j \overset{+}{\leftarrow} 1$
    $x[i] \leftarrow a_j$
    $q \leftarrow \delta(q, x[i])$

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

- $r = 3$
- $n = 2$
- $x = \epsilon$

DFA for L



Tabulated Function $T(q, n)$

States	n = 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

Unrank Algorithm

Code

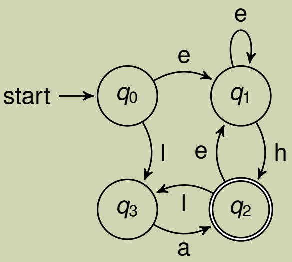
UNRANK(r)

$q \leftarrow q_0$
$n \leftarrow 0$
while $r \geq T(q_0, n)$
    do $r \overset{-}{\leftarrow} T(q_0, n)$
        $n \overset{+}{\leftarrow} 1$
$x \leftarrow \epsilon$
for $i \leftarrow 0$ **to** $n - 1$
    do $j \leftarrow 1$
        while $r \geq T(\delta(q, a_j), n - i - 1)$
            do $r \overset{-}{\leftarrow} T(\delta(q, a_j), n - i - 1)$
                $j \overset{+}{\leftarrow} 1$
        $x[i] \leftarrow a_j$
        $q \leftarrow \delta(q, x[i])$

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

- $r = 2$
- $n = 3$
- $x = \epsilon$

DFA for L



Tabulated Function $T(q, n)$

States	n				
	0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

Unrank Algorithm

Code

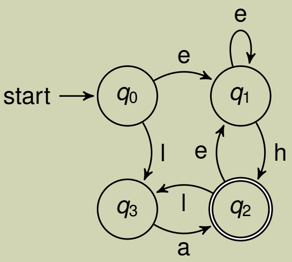
UNRANK(r)

$q \leftarrow q_0$
$n \leftarrow 0$
while $r \geq T(q_0, n)$
    do $r \overset{-}{\leftarrow} T(q_0, n)$
        $n \overset{+}{\leftarrow} 1$
$x \leftarrow \epsilon$
for $i \leftarrow 0$ **to** $n - 1$
    do $j \leftarrow 1$
        while $r \geq T(\delta(q, a_j), n - i - 1)$
            do $r \overset{-}{\leftarrow} T(\delta(q, a_j), n - i - 1)$
                $j \overset{+}{\leftarrow} 1$
        $x[i] \leftarrow a_j$
        $q \leftarrow \delta(q, x[i])$

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

- $r = 2$
- $n = 4$
- $x = \epsilon$

DFA for L



Tabulated Function $T(q, n)$

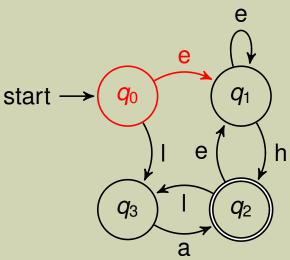
States	n = 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

Unrank Algorithm

Code

```
UNRANK(r)
q ← q0
n ← 0
while r ≥ T(q0, n)
      do r -← T(q0, n)
         n +← 1
x ← ϵ
for i ← 0 to n − 1
      do j ← 1
         while r ≥ T(δ(q, aj), n − i − 1)
               do r -← T(δ(q, aj), n − i − 1)
                  j +← 1
         x[i] ← aj
         q ← δ(q, x[i])
```

DFA for L



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

- $r = 2$
- $n = 4$
- $x = \epsilon$

Tabulated Function $T(q, n)$

States	n = 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

Unrank Algorithm

Code

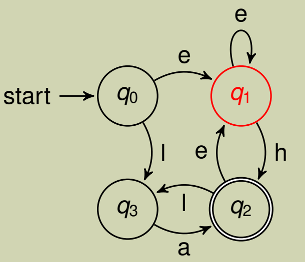
UNRANK(r)

$q \leftarrow q_0$
$n \leftarrow 0$
while $r \geq T(q_0, n)$
  do $r \stackrel{-}{\leftarrow} T(q_0, n)$
    $n \stackrel{+}{\leftarrow} 1$
$x \leftarrow \epsilon$
for $i \leftarrow 0$ **to** $n - 1$
  do $j \leftarrow 1$
    while $r \geq T(\delta(q, a_j), n - i - 1)$
      do $r \stackrel{-}{\leftarrow} T(\delta(q, a_j), n - i - 1)$
        $j \stackrel{+}{\leftarrow} 1$
    $x[i] \leftarrow a_j$
    $q \leftarrow \delta(q, x[i])$

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

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

DFA for L



Tabulated Function $T(q, n)$

States	n = 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

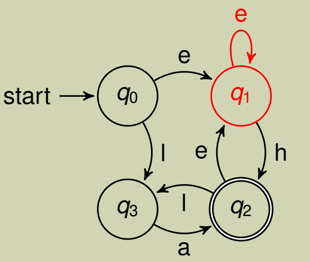
Unrank Algorithm

Code

UNRANK(r)

$q \leftarrow q_0$
$n \leftarrow 0$
while $r \geq T(q_0, n)$
  do $r \stackrel{-}{\leftarrow} T(q_0, n)$
    $n \stackrel{+}{\leftarrow} 1$
$x \leftarrow \epsilon$
for $i \leftarrow 0$ **to** $n - 1$
  do $j \leftarrow 1$
    while $r \geq T(\delta(q, a_j), n - i - 1)$
      do $r \stackrel{-}{\leftarrow} T(\delta(q, a_j), n - i - 1)$
        $j \stackrel{+}{\leftarrow} 1$
    $x[i] \leftarrow a_j$
    $q \leftarrow \delta(q, x[i])$

DFA for L



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

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

Tabulated Function $T(q, n)$

States	n = 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

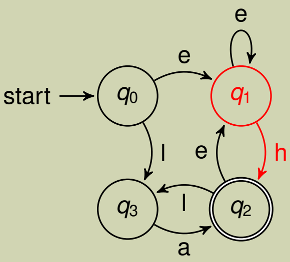
Unrank Algorithm

Code

UNRANK(r)

```
q ← q_0
n ← 0
while r ≥ T(q_0, n)
      do r ←⁻ T(q_0, n)
         n ←⁺ 1
x ← ε
for i ← 0 to n − 1
      do j ← 1
         while r ≥ T(δ(q, a_j), n − i − 1)
               do r ←⁻ T(δ(q, a_j), n − i − 1)
                  j ←⁺ 1
         x[i] ← a_j
         q ← δ(q, x[i])
```

DFA for L



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

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

Tabulated Function $T(q, n)$

States	n = 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

Unrank Algorithm

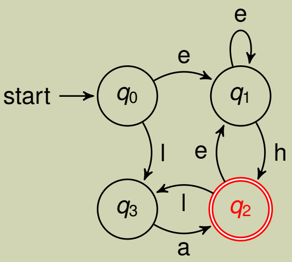
Code

```
UNRANK(r)
q ← q0
n ← 0
while r ≥ T(q0, n)
      do r ←− T(q0, n)
         n ←+ 1
x ← ϵ
for i ← 0 to n − 1
      do j ← 1
         while r ≥ T(δ(q, aj), n − i − 1)
               do r ←− T(δ(q, aj), n − i − 1)
                  j ←+ 1
         x[i] ← aj
         q ← δ(q, x[i])
```

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

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

DFA for L



Tabulated Function $T(q, n)$

States	n = 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

Unrank Algorithm

Code

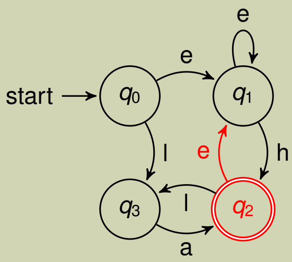
UNRANK(r)

$q \leftarrow q_0$
$n \leftarrow 0$
while $r \geq T(q_0, n)$
  do $r \overset{-}{\leftarrow} T(q_0, n)$
    $n \overset{+}{\leftarrow} 1$
$x \leftarrow \epsilon$
for $i \leftarrow 0$ **to** $n - 1$
  do $j \leftarrow 1$
    while $r \geq T(\delta(q, a_j), n - i - 1)$
      do $r \overset{-}{\leftarrow} T(\delta(q, a_j), n - i - 1)$
        $j \overset{+}{\leftarrow} 1$
    $x[i] \leftarrow a_j$
    $q \leftarrow \delta(q, x[i])$

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

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

DFA for L



Tabulated Function $T(q, n)$

States	n				
	0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

Unrank Algorithm

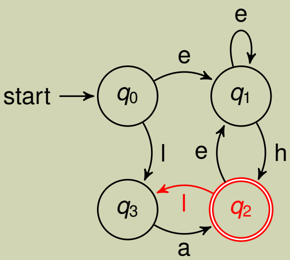
Code

```
UNRANK(r)
q ← q0
n ← 0
while r ≥ T(q0, n)
      do r ←⁻ T(q0, n)
         n ←⁺ 1
x ← ϵ
for i ← 0 to n − 1
      do j ← 1
         while r ≥ T(δ(q, a_j), n − i − 1)
               do r ←⁻ T(δ(q, a_j), n − i − 1)
                  j ←⁺ 1
         x[i] ← a_j
         q ← δ(q, x[i])
```

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

- $r = 0$
- $n = 4$
- $x =$ *eh*

DFA for L



Tabulated Function $T(q, n)$

States	n = 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

Unrank Algorithm

Code

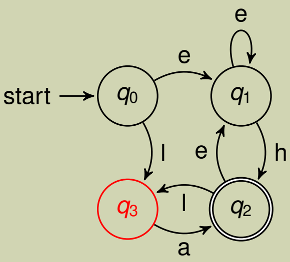
UNRANK(r)

$q \leftarrow q_0$
$n \leftarrow 0$
while $r \geq T(q_0, n)$
  do $r \overset{-}{\leftarrow} T(q_0, n)$
    $n \overset{+}{\leftarrow} 1$
$x \leftarrow \epsilon$
for $i \leftarrow 0$ **to** $n - 1$
  do $j \leftarrow 1$
    while $r \geq T(\delta(q, a_j), n - i - 1)$
      do $r \overset{-}{\leftarrow} T(\delta(q, a_j), n - i - 1)$
        $j \overset{+}{\leftarrow} 1$
    $x[i] \leftarrow a_j$
    $q \leftarrow \delta(q, x[i])$

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

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

DFA for L



Tabulated Function $T(q, n)$

States	n = 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

Unrank Algorithm

Code

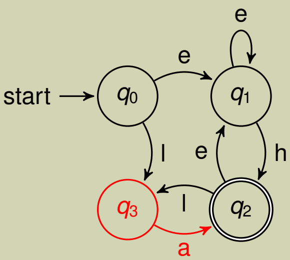
UNRANK(r)

```
q ← q_0
n ← 0
while r ≥ T(q_0, n)
      do r ←⁻ T(q_0, n)
         n ←⁺ 1
x ← ε
for i ← 0 to n − 1
      do j ← 1
         while r ≥ T(δ(q, a_j), n − i − 1)
               do r ←⁻ T(δ(q, a_j), n − i − 1)
                  j ←⁺ 1
         x[i] ← a_j
         q ← δ(q, x[i])
```

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

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

DFA for L



Tabulated Function $T(q, n)$

States	n = 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

Unrank Algorithm

Code

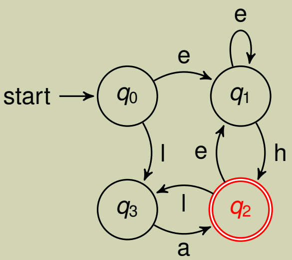
UNRANK(r)

$q \leftarrow q_0$
$n \leftarrow 0$
while $r \geq T(q_0, n)$
    do $r \overset{-}{\leftarrow} T(q_0, n)$
        $n \overset{+}{\leftarrow} 1$
$x \leftarrow \epsilon$
for $i \leftarrow 0$ **to** $n - 1$
    do $j \leftarrow 1$
        while $r \geq T(\delta(q, a_j), n - i - 1)$
            do $r \overset{-}{\leftarrow} T(\delta(q, a_j), n - i - 1)$
                $j \overset{+}{\leftarrow} 1$
        $x[i] \leftarrow a_j$
        $q \leftarrow \delta(q, x[i])$

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

- $r = 0$
- $n = 4$
- $x =$ *ehla*

DFA for L



Tabulated Function $T(q, n)$

States	n = 0	1	2	3	4
q_0	0	0	2	1	5
q_1	0	1	1	3	4
q_2	1	0	2	1	5
q_3	0	1	0	2	1

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?

$$ba^*(c|d)a^+ \xrightarrow{\text{Thompson algorithm}} \text{N-DFA} \xrightarrow{\text{determinization}} \text{DFA}$$

$$(a|b)^+a(a|b)(a|b) \xrightarrow{\text{Thompson algorithm}} \text{N-DFA} \xrightarrow{\text{determinization}} \textbf{DFA}$$

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?

$$ba^*(c|d)a^+ \xrightarrow{\text{Thompson algorithm}} \text{N-DFA} \xrightarrow{\text{determinization}} \text{DFA}$$

$$(a|b)^+a(a|b)(a|b) \xrightarrow{\text{Thompson algorithm}} \text{N-DFA} \xrightarrow{\text{determinization}} \textbf{DFA}$$

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

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?

$\Downarrow$

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:

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

Ranking is a particular case of relaxed ranking where $i = |L|$, thus *rank* and *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 $\mathbb{Z}_i \to i = |\mathcal{P}|$
- 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

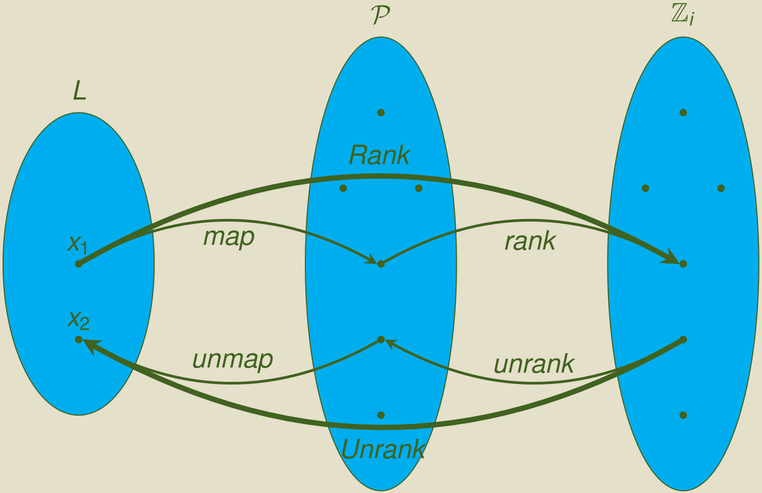
- Since there is a bijection between the set of accepting paths $\mathcal{P}$ and $\mathbb{Z}_i \rightarrow i = |\mathcal{P}|$
- In N-DFA, more than one accepting path for each string
- Therefore, we need to map a string to one of its accepting paths

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



Map Function

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

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 $\mathbb{Q}$)

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 $\mathbb{Q}$)

Building the Graph

- In a N-DFA, the transition function is defined as $\delta : \mathbb{Q} \times \Sigma \mapsto \mathcal{P}(\mathbb{Q})$
- When we parse the k-th character of string x, we define a frontier $\mathcal{F}_k \subseteq \mathbb{Q} = \bigcup_{q \in \mathcal{F}_{k-1}} \delta(q, x[k-1])$, with the base case $\mathcal{F}_0 = \{q_0\}$
- $x \in L \Leftrightarrow \mathcal{F}_{|x|} \cap \mathbb{F} \neq \emptyset$
- 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 $\mathcal{B}_k \subseteq \mathbb{Q} = \{q \mid \exists q' \in \mathcal{B}_{k+1}(\delta(q, x[k]) = q')\}$, with the base case $\mathcal{B}_{|x|} = \mathbb{F}$
- A frontier $\mathcal{B}_k$ represents the states which belongs to an accepting path for the substring $x[k \dots n-1]$, starting from any state of $\mathcal{B}_k$
- Now, define the frontier $\mathcal{S}_k = \mathcal{F}_k \cap \mathcal{B}_k \rightarrow$ 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 $x[k \dots n-1]$

Map Function: Graph Building

Pruning Non-Accepting Paths

- We can enrich the information provided by the frontiers with backward frontiers
- A backward frontier $\mathcal{B}_k \subseteq \mathbb{Q} = \{q \mid \exists q' \in \mathcal{B}_{k+1}(\delta(q, x[k]) = q')\}$, with the base case $\mathcal{B}_{|x|} = \mathbb{F}$
- A frontier $\mathcal{B}_k$ represents the states which belongs to an accepting path for the substring $x[k \ldots n-1]$, starting from any state of $\mathcal{B}_k$
- Now, define the frontier $\mathcal{S}_k = \mathcal{F}_k \cap \mathcal{B}_k \rightarrow$ 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 $x[k \ldots n-1]$

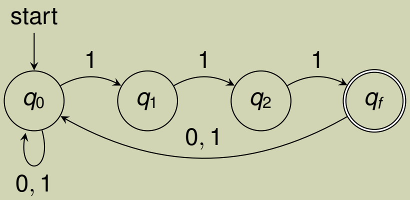
Building the Graph

- Each frontier $\mathcal{S}_k$ becomes a layer of the graph
- Each layer is made of the states $q \in \mathcal{S}_k$
- Each state q is connected to the states of the subsequent layer $q' \in \delta(q, x[k])$

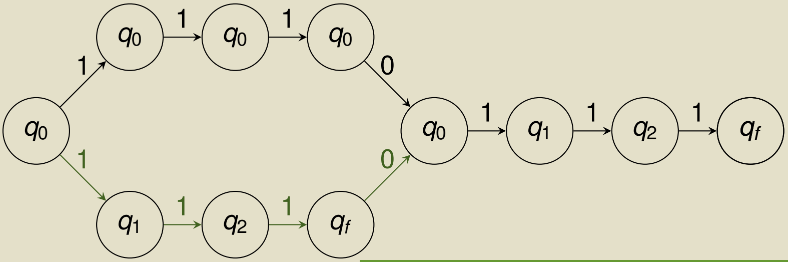
Graph Building

Graph building for $\mathbf{x} = \mathbf{1110111}$

N-DFA for $\mathbf{L} = (\mathbf{0} \mid \mathbf{1})^*\mathbf{111}$



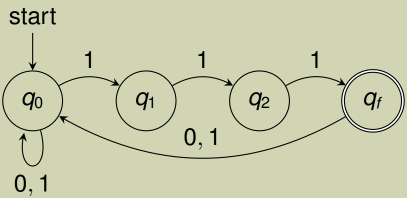
k	$\mathcal{F}_k$	$\mathcal{B}_k$	$\mathcal{S}_k$
0	q_0	q_0	q_0
1	q_0, q_1	q_0, q_1	q_0, q_1
2	q_0, q_1, q_2	q_0, q_2	q_0, q_2
3	q_0, q_1, q_2, q_f	q_0, q_f	q_0, q_f
4	q_0	q_0	q_0
5	q_0, q_1	q_1	q_1
6	q_0, q_1, q_2	q_2	q_2
7	q_0, q_1, q_2, q_f	q_f	q_f



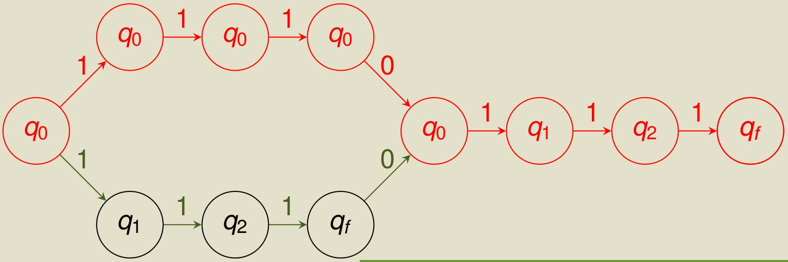
Graph Building

Graph building for x = 1110111

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



k	$\mathcal{F}_k$	$\mathcal{B}_k$	$\mathcal{S}_k$
0	q_0	q_0	q_0
1	q_0, q_1	q_0, q_1	q_0, q_1
2	q_0, q_1, q_2	q_0, q_2	q_0, q_2
3	q_0, q_1, q_2, q_f	q_0, q_f	q_0, q_f
4	q_0	q_0	q_0
5	q_0, q_1	q_1	q_1
6	q_0, q_1, q_2	q_2	q_2
7	q_0, q_1, q_2, q_f	q_f	q_f



Completing Relaxed Ranking

Unmap

- Performing *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
- *unmap* is trivially linear in $|x|$

Completing Relaxed Ranking

Unmap

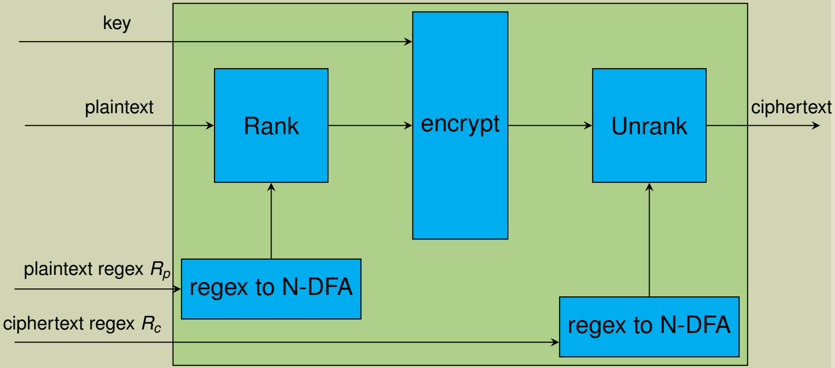
- Performing *unmap* means deriving the string $x \in L$ from an accepting path $\pi \in \mathcal{P}$
- This is straightforward: just follow the transitions of the path and concatenate their characters
- *unmap* is trivially linear in $|x|$

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

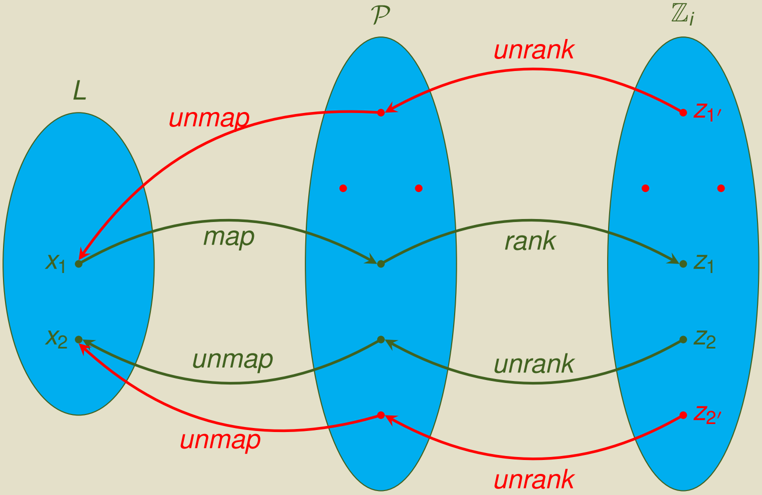
FTE On a Generic Regular Language

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



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 $\mathcal{I}(L) = \{z \in \mathbb{Z}_i \mid Rank_L(Unrank_L(z)) = z\}$
- Since *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 $\mathcal{I}(L) = \{z \in \mathbb{Z}_i \mid Rank_L(Unrank_L(z)) = z\}$
- Since *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 $z \in \mathcal{I}(L) \Leftrightarrow Rank_L(Unrank_L(z)) = z$
- More efficient way: $map_L(Unrank_L(z)) = unrank_L(z)$

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 $= [0-9]\{16\}$

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 $\rightarrow$ 64 bits required
- However, to represent all possible integers of 16 digits, 54 bits are sufficient $\rightarrow$ 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 $\rightarrow$ 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 $\rightarrow$ 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

	PSQL	AES	AE	FPE	FTE
Table size(MB)	50	65	112	50	42
Query avg. Time(ms)	74	92	112	125	110

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

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

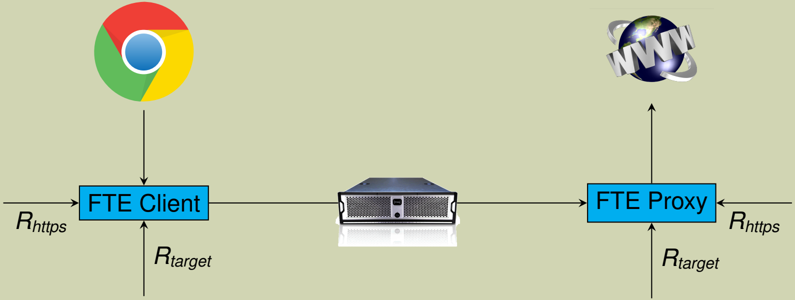
Existing Inspectors

Name	Classifier	Price
appid	regex + port filtering	Free
L7-filter	regex	Free
YAF	regex	Free
Bro	regex + C code to check false positives	Free
Nprobe	regex + flow tracking	300 €
DPI-X	??	10k €

- 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

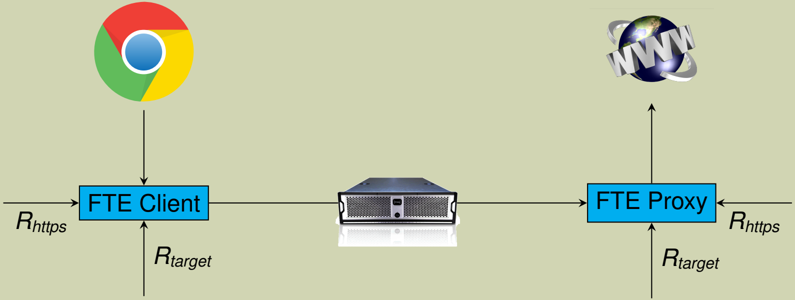
Attack Scenario



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 $\approx$ 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

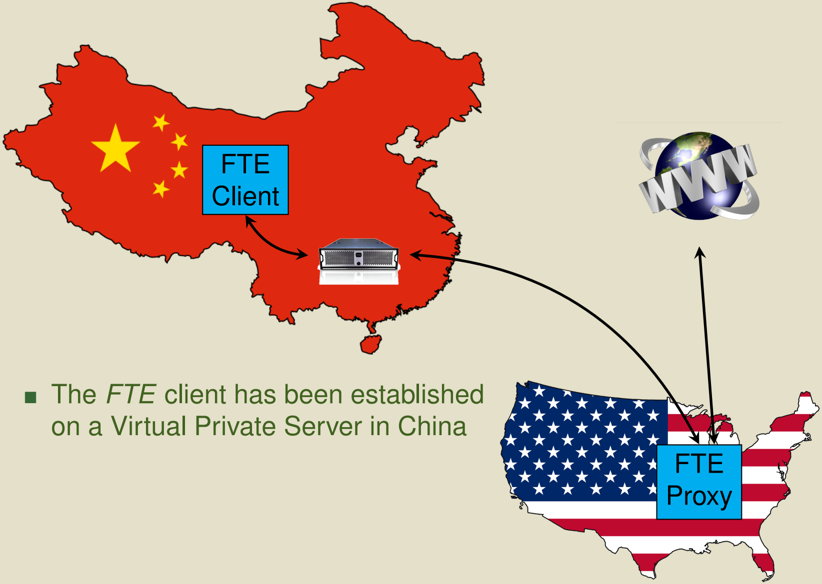
Defeating Censorship

- 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

$\Downarrow$

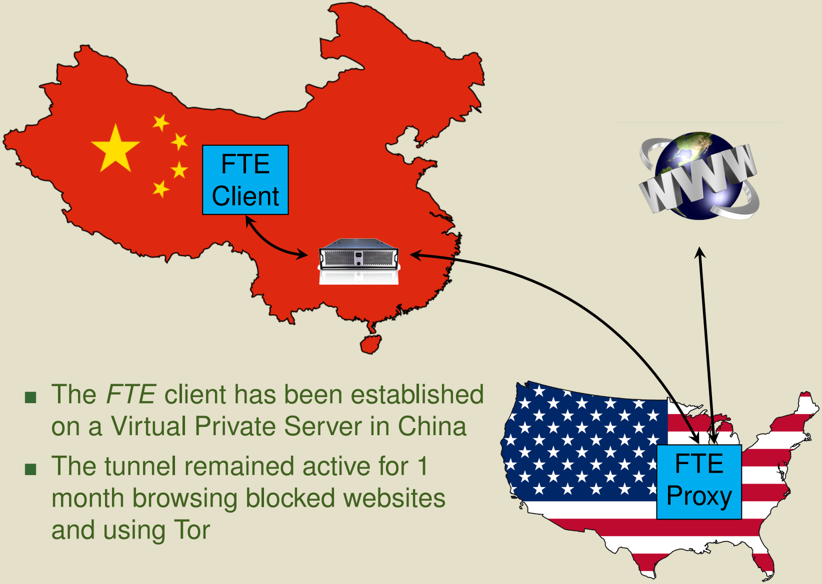
We can use FTE to communicate with the Tor OP!

On the Field Testing



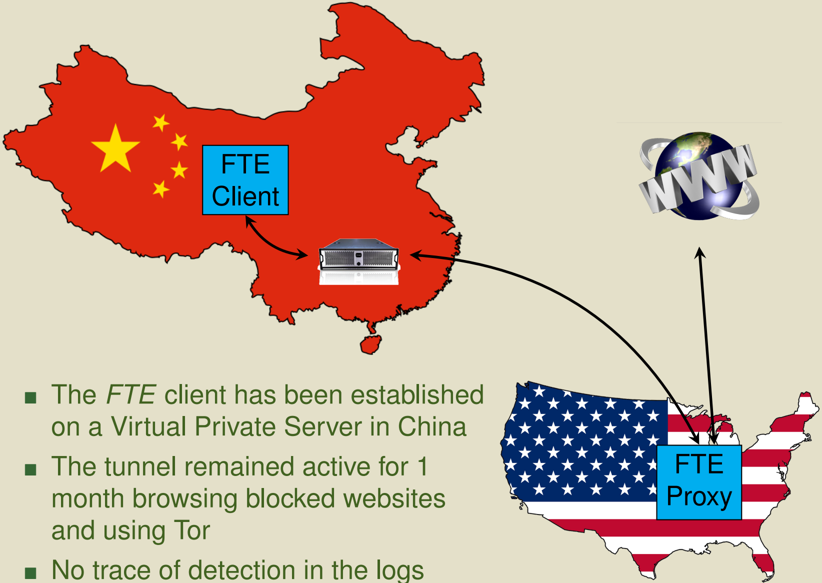
- The *FTE* client has been established on a Virtual Private Server in China

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

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 $\rightarrow$ high complexity or negligible probability of being caught

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 $\rightarrow$ 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

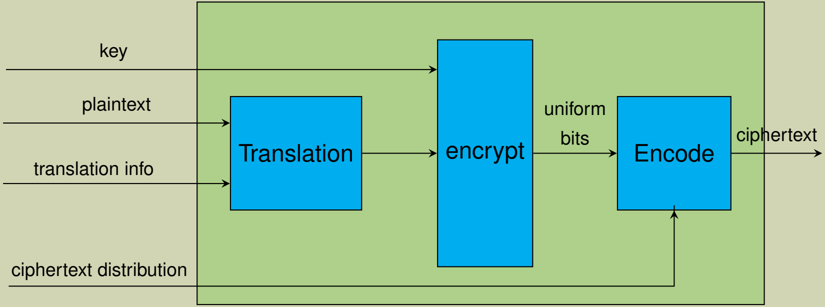
Further Developments

- 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?

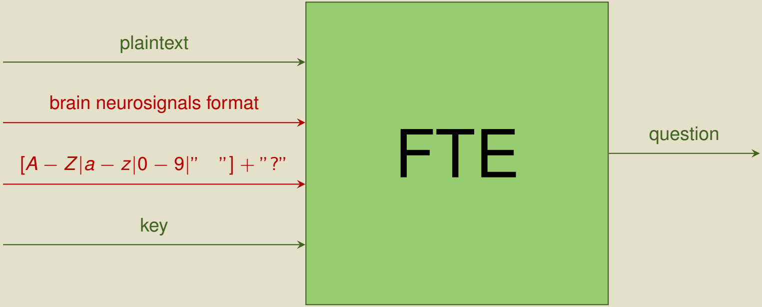
Conclusions

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

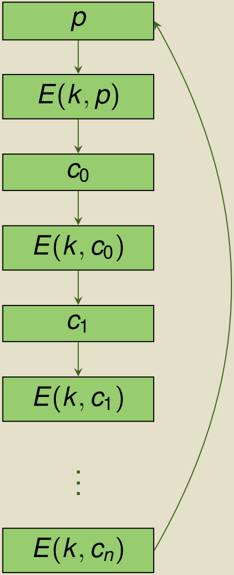


plaintext
brain neurosignals format
$[A-Z|a-z|0-9|"\quad"]+"?"$
key
FTE
question

Invalid Encryptions: Cycle Walking

Workaround for Unrankable Ciphertexts

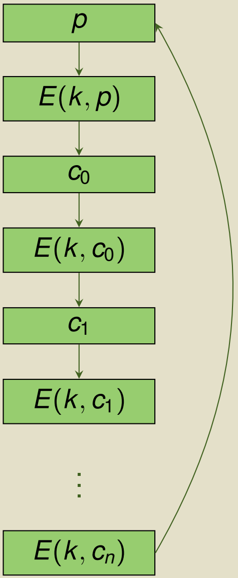
- In a deterministic cipher $c = E(k, p)$, c is constant for the same k, p values
- If c is not a valid ciphertext, we can generate a new ciphertext by computing $c_1 = E(k, c)$
- Iterating this process, we get either back to p or to a valid ciphertext
- This procedure is called cycle walking



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 $p_i \in \mathcal{I}(L(R_p))$
- Unfortunately, if there is a $p_k \in \mathcal{I}(L(R_p)) \mid k > i$, 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 $c_i \in \mathcal{I}(L(R_p)) \wedge c_i \notin \mathcal{I}(L(R_c)) \rightarrow$ decryption will fail, thus encryption is impossible!



LibFTE Configuration Assistant

```
$ ./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
...
$
```