LibFTE: A Toolkit for Constructing Practical, Format-Abiding Encryption Schemes

Daniel Luchaup, University of Wisconsin-Madison
Kevin P. Dyer, Portland State University
Somesh Jha, University of Wisconsin-Madison
Thomas Ristenpart, University of Wisconsin-Madison
Thomas Shrimpton, Portland State University
In-place encryption in database

Plaintext

1234 5678 9876 5432

FPE

(Format-preserving encryption)
(Bellare et al. SAC’09)

Ciphertext

4417 1234 5678 9112

Voltage security

PCI
Censorship circumvention

Plaintext → FTE → Ciphertext

(Format-transforming encryption)
(Dyer et al. CCS’13)

or SMTP, SIP, RSTP, SSH,….
What about other applications?

Expert knowledge required, substantial implementation and performance challenges.
What about other applications?

Expert knowledge required, substantial implementation and performance challenges.
Our contribution: LibFTE

- New algorithms to support this framework that solve open problem
- A general framework for building FTE (and FPE)
- A library (python/C++), and a toolkit to support development

https://libfte.org/
Our contribution: LibFTE

- New algorithms to support this framework that solve open problem
- A general framework for building FTE (and FPE)
- A library (python/C++), and a toolkit to support development

https://libfte.org/
The Previous FTE scheme (Dyer et al. 2013)

key
plaintext
regex R

Encrypt
regex-to-DFA

Unrank

ciphertext in L(R)

Goldberg/Sipser (1985)
Bellare et al. (2009)
The Previous FTE scheme (Dyer et al. 2013)

key
plaintext
regex R

Encrypt
regex-to-DFA

Unrank

ciphertext in L(R)

Goldberg/Sipser (1985)
Bellare et al. (2009)

L(R)

rank(x_i)=i
unrank(2)=x_2

regex R
plaintext
plaintext
plaintext

0 1 2 i |L(R)|-1
The Previous FTE scheme (Dyer et al. 2013)


efficient implementation of (un)rank requires DFA representation of language
The Previous FTE scheme (Dyer et al. 2013)

- Key
- Plaintext
- Regex R

- Encrypt
- Regex-to-DFA

Unrank

Ciphertext in L(R)

Unranking requires space linear in the size of the DFA, and the length of the longest plaintext.
The Previous FTE scheme (Dyer et al. 2013)

For some regular expressions, this works out just fine…

unranking requires space linear in the size of the DFA, and the length of the longest plaintext

regex \rightarrow NFA \rightarrow DFA
The Previous FTE scheme (Dyer et al. 2013)

For others, you can have an exponential space blow-up.

A DFA with 13K states requires ~200MB of memory for (un)ranking. (Dyer et al., CCS’13)
A new, NFA-based, FTE scheme

Based on new concept, relaxed ranking:

1. Encrypt
2. valid?
3. Unrank
4. cipher text in L(R)

key
plaintext
regex R
A new, NFA-based, FTE scheme

Based on new concept, \textit{relaxed ranking}: 

\begin{align*}
\text{rank}(x_i) &= i \\
\text{unrank}(2) &= x_2 \\
\text{unrank}(3) &= x_2
\end{align*}
LibFTE framework is more general

- key
- plaintext in \( L(R_1) \)
- regex \( R_1 \)
- regex \( R_2 \)
- ciphertext in \( L(R_2) \)
LibFTE framework is more general

key
plaintext in L(R1)
regex R1
regex R2

Rank
Encrypt
valid?
Unrank

Y
N
ciphertext in L(R2)
LibFTE framework is more general

key plaintext in L(R1)
regex R1
regex R2

plaintext in L(R1)

Rank Encrypt valid? Unrank

Y N

ciphertext in L(R2)
LibFTE framework is more general
LibFTE framework is more general

NFA or DFA ranking for R1
LibFTE framework is more general

NFA or DFA ranking for R1
NFA or DFA ranking for R2
LibFTE framework is more general

NFA or DFA ranking for R1
NFA or DFA ranking for R2
Deterministic or randomized encryption
LibFTE framework is more general

NFA or DFA ranking for R1
NFA or DFA ranking for R2
Deterministic or randomized encryption
Choice of regular expressions R1, R2
LibFTE framework is more general

NFA or DFA ranking for R1
NFA or DFA ranking for R2
Deterministic or randomized encryption
Choice of regular expressions R1, R2
LibFTE framework is more general

NFA or DFA ranking for R1
NFA or DFA ranking for R2
Deterministic or randomized encryption
Choice of regular expressions R1, R2
LibFTE Configuration Assistant

**input:** input format, output format, and optional restrictions (e.g., encryption must be randomized/deterministic)

**output:** an error OR a list of schemes that satisfy the user-specified constraints, with statistics (no. cycle walks, etc.)

```bash
$ ./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 ...
T-NN 0.39ms 0.38ms ...

$ OR

error

success
```
**LibFTE Configuration Assistant**

**input:** input format, output format, and optional restrictions (e.g., encryption must be randomized/deterministic)

**output:** an error OR a list of schemes that satisfy the user-specified constraints, with statistics (no. cycle walks, etc.)

```bash
$ ./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

```bash
$ ./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

$ 
```

error  OR  success
Case Studies
LibFTE: Snort IDS Regexs

- Used a set of ~3.5K regexs from the Snort corpus. A good stress test.
- Reduced avg. memory by 30%. In some cases by orders of magnitude.
- Handled extreme cases (~3%) where DFA ranking fails. All 3.5K regexs could be used with <150MB of memory using NFA ranking.
LibFTE: Snort IDS Regexs

- Used a set of ~3.5K regexs from the Snort corpus. A good stress test.
- Reduced avg. memory by 30%. In some cases by orders of magnitude.
- Handled extreme cases (~3%) where DFA ranking fails. All 3.5K regexs could be used with <150MB of memory using NFA ranking.
LibFTE: Database encryption+compression

• Compare against PostgreSQL encryption library.
  Because there does not exist any public implementation of FPE/FFX.

• Created a table with credit card numbers.
  Then tested under various configurations.

<table>
<thead>
<tr>
<th>Table Size</th>
<th>Database Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PSQL</td>
</tr>
<tr>
<td>50MB</td>
<td>65MB</td>
</tr>
<tr>
<td>Query Avg.</td>
<td>74ms</td>
</tr>
</tbody>
</table>

Query Avg. = time to retrieve 1000 CC nums
LibFTE: Database encryption+compression

- **Compare against PostgreSQL encryption library.**
  Because there does not exist any public implementation of FPE/FFX.

- **Created a table with credit card numbers.**
  Then tested under various configurations.

Compared to AES, LibFTE saves ~35% on disk.

<table>
<thead>
<tr>
<th>Database Configuration</th>
<th>PSQL</th>
<th>+AES</th>
<th>+AE</th>
<th>+FPE</th>
<th>+FTE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Table Size</strong></td>
<td>50MB</td>
<td>65MB</td>
<td>112MB</td>
<td>50MB</td>
<td>42MB</td>
</tr>
<tr>
<td><strong>Query Avg.</strong></td>
<td>74ms</td>
<td>92ms</td>
<td>112ms</td>
<td>125ms</td>
<td>110ms</td>
</tr>
</tbody>
</table>

Query Avg. = time to retrieve 1000 CC nums
LibFTE: Firefox extension

- A pure-JavaScript LibFTE interface.
- We created a mapping between fields and FPE/FTE schemes. Simple, using CSS id/class attrs.
- Using libfte: only 20 lines of code
LibFTE: Firefox extension

- A pure-Javascript LibFTE interface.

- We created a mapping between fields and FPE/FTE schemes. Simple, using CSS id/class attrs.

- Using libfte: only 20 lines of code
Conclusion: LibFTE

• **A general framework for building FTE (and FPE) schemes.**
  No one-off, per-deployment solutions.

• **New algorithmic advances.**
  Abstracts away clunky design choices.

• **Surfaced new use cases.**
  Compression+encryption, in-browser encryption.

• **High-performance, publicly available.**
  APIs for C++ and Python.

https://libfte.org/