Practical Solutions for Format Preserving Encryption

Authors: Mor Weiss, Boris Rozenberg, and Muhammad Barham
Presented by Boris Rozenberg (borisr@il.ibm.com)

May 21, 2013
Talk Outline

• Motivating example
• Encryption: background
• Format Preserving Encryption (FPE):
  – Simple constructions
  – Better constructions:
    • Representing general formats
    • Encrypting general formats
  – Dealing with large formats
  – Evaluation
• Concurrent Work
• Conclusion
Motivating Example

Age
Former and present illnesses
Prescribed medication
Encryption
(keeping data private)
Encryption Schemes

• A triplet $\Pi = (KeyGen, Enc, Dec)$ of algorithms
  – $\Pi$ associated with 3 sets:
    • $\mathcal{K}$: domain of valid keys
    • $\mathcal{M}$: message domain
    • $\mathcal{C}$: ciphertext domain.
  – $KeyGen$ generates random key from $\mathcal{K}$
  – $Enc$ on message (plaintext) $m \in \mathcal{M}$ and key $k \in \mathcal{K}$ outputs ciphertext $c \in \mathcal{C}$
  – $Dec$ on ciphertext $c \in \mathcal{C}$ and key $k \in \mathcal{K}$ outputs message $m \in \mathcal{M}$

• Deterministic encryption: only $KeyGen$ is randomized
  – Everything deterministic once key is chosen

• Assumed adversary knows everything but key

© 2015 IBM Corporation
Encryption Schemes: Required Properties

• A triplet \( \Pi = (\text{KeyGen}, \text{Enc}, \text{Dec}) \) of algorithms

• Correctness: for every \( k \in \mathcal{K} \) and every \( m \in \mathcal{M} \)

\[
\text{Dec}(k, \text{Enc}(k, m)) = m
\]

• Security:
  – Many security notions
  – Intuitively, ciphertext \( c \) reveals (almost) no information on message \( m \)
    • Even if adversary has prior knowledge
  – Achieved by random 1:1 functions

• For usability, all algorithms must be efficient
Security-Efficiency Tradeoffs

\[ \text{Enc}(k, m) = m \]
for every key \( k \)

\[ \text{Enc}(k, \cdot) \] applies a random 1:1 function
Format Preserving Encryption
(encrypting to “acceptable” formats)
Format Preserving Encryption (FPE)

- Standard encryption maps messages to “garbage”
  - May be impossible to store ciphertext in same tables
  - Applications using data may crash
- Need some plaintext properties to be preserved
- FPE: *Deterministic* encryption scheme $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$
- with additional property $M = C$
- Ciphertexts have the same format as plaintexts!
  - Social security number (ssn) mapped to legal ssn
  - Credit card number (ccn) mapped to legal ccn
  - Address mapped to legal address
  - Etc...
Example: The DES Encryption

DES is format-preserving!
FPE Schemes For General Formats: Simple Solution

• Known encryption schemes are FP for *fixed, specific* formats
  – Usually, bit strings of fixed length

• What about other formats?
  – For CCNs, message space $\subseteq \{0,1, ..., 9\}^{16}$
  – No known encryption for this message space!

• Can use *cycle-walking* [Black-Rogaway’02]
  “if at first you don’t succeed, you pick yourself up and try again”
  – Use “standard” encryption with $\{0,1, ..., 9\}^{16} \subseteq \mathcal{M}$
  – Repeat until ciphertext in $\{0,1, ..., 9\}^{16}$
Cycle-Walking

Message space $\mathcal{M} = 2^{128}$

Valid CCNs $\{0, 1, \ldots, 9\}^{16}$
Cycle-Walking: Pros and Cons

• Pros:
  – Use “off-the-shelf” encryption schemes
    • One design for all formats
  – Known encryption schemes are provably secure

• Cons:
  – Average efficiency depends on ratio between format-size and message domain size
    • Need to repeat \( \frac{\text{format size}}{|\mathcal{M}|} \) times on average
  – No bound on actual efficiency
Improved FPEs for Numeric Domains

• Several known schemes for numeric domains
  – Considered due to (in)efficiency of cycle walking

• [Bellare et al. ’09] construct integer-FPE: FPE with $\mathcal{M} = \{0,1, \ldots, M - 1\}$

What about non-numeric domains?
From Integer-FPE to General-Format FPE

• Can base general-format FPE on integer-FPE using Rank-then-Encipher (RtE): [Bellare et al. ’09]
  – Message space $\mathcal{M}$ arbitrarily ordered: rank: $\mathcal{M} \to \{0,1,\ldots,M\}$
Warm-Up Example

<table>
<thead>
<tr>
<th>X</th>
<th>y</th>
<th>7</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper case</td>
<td>lower case</td>
<td>digit</td>
<td>upper case</td>
</tr>
</tbody>
</table>

idea: compute location in lexicographic order

index each character

23 24 7 0

rank calculated by scaling and summing the indices

23 \cdot 26 \cdot 10^{26} + 24 \cdot 10^{26} + 7 \cdot 26 + 0

gives the \text{Sum-and-Scale} method

1234 = 1 \cdot 10 \cdot 10 \cdot 10 + 2 \cdot 10 \cdot 10 + 3 \cdot 10 + 4

© 2015 IBM Corporation
Ranking General Formats: Simple Solution

• Want: efficient rank: $\mathcal{M} \rightarrow \{0, 1, \ldots, M - 1\}$

• Can rank every format $\mathcal{F}$ defined by
  – Length $\ell$
  – Sets $\Sigma_1, \ldots, \Sigma_\ell$ of “legal” characters in locations 1, $\ldots$, $\ell$.

• Simple solution:
  – Divide $\mathcal{M}$ to subsets $\mathcal{M}_1, \ldots, \mathcal{M}_k$
  – $\mathcal{M}_i$ defined by $\ell_i, \Sigma_1^i, \ldots, \Sigma_\ell^i$ How to define efficiently?!
  – Rank and encryption of $m \in \mathcal{M}_i$ computed in relation to $\mathcal{M}_i$
Simple Solution: Security Analysis

Simple solution:
- Divide $\mathcal{M}$ to subsets $\mathcal{M}_1, \ldots, \mathcal{M}_k$
- $\mathcal{M}_i$ defined by $\ell_i \Sigma_1^i, \ldots, \Sigma_\ell^i$
- Rank and encryption of $m \in \mathcal{M}_i$ computed in relation to $\mathcal{M}_i$

Security is compromised:
- Ranking computed in every $\mathcal{M}_i$ separately
- So $m \in \mathcal{M}_i$ always encrypted to ciphertext in $\mathcal{M}_i$
- Rarely the case for random 1:1 functions $f: \mathcal{M} \rightarrow \mathcal{M}$, especially for large $k$
Simple Solution: **Practical** Security

Simple solution:
- Divide $\mathcal{M}$ to subsets $\mathcal{M}_1, \ldots, \mathcal{M}_k$
- $\mathcal{M}_i$ defined by $\ell_i \sum^i_1, \ldots, \Sigma^i_\ell$
- Rank and encryption of $m \in \mathcal{M}_i$ computed in relation to $\mathcal{M}_i$

- $\mathcal{M} =$ names format:
  - 2-4 words
  - Every word upper-case followed by 1-10 lower-case
- $\mathcal{M}_i$ defines number of words + number of letters in each word
- “John Smith” can encrypt to “Angm Ojkri” but not to “Bar Refaeli”
- If only one of them is possible, adversary knows plaintext for sure

© 2015 IBM Corporation
Optimizing Security-Efficiency Tradeoff

- Cycle walking inefficient since ignores format properties
- Simple solution insecure since preserves “cosmetic” message properties
- Want a “balanced” encryption scheme
  - Take into consider format properties...
  - ...and preserve only them!
  - Need:
    - Framework of representing general formats
    - Method of ranking general formats

© 2015 IBM Corporation
Representing General Formats: Framework

- Define building-blocks and operations
- Building blocks are called “primitives”
  - SSNs
  - CCNs
  - Dates (between minDate and maxDate)
  - Fixed-length strings with index-specific character-sets
- Usually represent “rigid” formats
  - e.g., fixed length
- Can also represent “less rigid” formats
  - Variable-length strings over some alphabet

© 2015 IBM Corporation
Representing General Formats: Framework (2)

- Define building-blocks and operations
- Operations allow constructing compound (and complex) formats from primitives
  - Operations preserve the parsing property: compound format can parse string to ingredients
- Compound formats are called “fields”
- Can construct format $\mathcal{F}$ from “smaller” formats $\mathcal{F}_1, \ldots, \mathcal{F}_k$ by:
  - Union
  - Concatenation:
    - $\mathcal{F} = \mathcal{F}_1 \cdot d_1 \cdot \mathcal{F}_2 \cdot \ldots \cdot d_{n-1} \cdot \mathcal{F}_n, d_1, \ldots, d_{n-1}$ are delimiter characters
    - $\mathcal{F} = \mathcal{F}_1 \cdot \ldots \cdot \mathcal{F}_k$ in some cases
  - Range: $\mathcal{F} = (\mathcal{F}_1 \cdot d)^k, \min \leq k \leq \max$
Constructing Compound Formats: Example

- \( F_1 = \{A, B, \ldots, Z\} \)
- \( F_2 = \) length-\( k \) strings of lower-case letters, \( 1 \leq k \leq 10 \)
- \( F_3 = \) SSNs

- Concatenation:
  - \( F_{\text{word}} = F_1 \cdot F_2 \) gives words
  - \( F = F_2 \cdot \ldots \cdot F_2 \), e.g., “abc-def” or “aaaaa-bb”

- Union: \( F = F_1 \cup F_3 \), e.g., “1112233333” or “A”

- Range: \( F_{\text{name}} = (F_{\text{word}} \cdot space)^k \) for \( 2 \leq k \leq 4 \) gives names, e.g. “Bar Refaeli” or “Louisa May Alcott”
Ranking General Formats

• Define ranking for building-blocks
• Define ranking for operations
• Automatically gives ranking for compound formats:
  – Parse string to ingredients
  – Delegate ranking of substrings to ingredients
  – Use ranking for operations to “glue” ranks together
Ranking Primitives

- Ranking usually fairly simple:
  - **SSNs:** “basically” 9-digit numbers, remove illegal-SSNs smaller that given SSN
  - **CCN:** first 15 digits are the rank
  - **Dates:** count seconds since minDate
  - **Fixed-length strings:** Sum-and-Scale
  - **Variable-length strings:** Sum-and-Scale with same-length strings + offset by number of shorter strings

- Unranking more complex

© 2015 IBM Corporation
Ranking Operations: Union

\[ \mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2 \]
Ranking Operations: Concatenation

\[ \mathcal{F} = \mathcal{F}_1 \cdot d \cdot \mathcal{F}_2 \]
\[ m = m_1 \cdot d \cdot m_2 \]

Sum-and-Scale:

\[ r = r_1 \cdot \mathcal{F}_2 \cdot \text{size()} + r_2 \]

“smaller” formats interpreted as character-sets
Ranking Operations: \textit{Range}

\[ \mathcal{F} = (\mathcal{F}_1 \cdot d)^k, \quad 1 \leq k \leq 4 \]
\[ m = m_1 \cdot d \cdot m_2 \cdot d \cdot m_3 \cdot d \]

Add contribution of shorter strings:

\[ r'' = \mathcal{F}_1.\text{size}()^2 + \mathcal{F}_1.\text{size}() \]

Sum-and-scale:

\[ r' = r_1 \cdot (\mathcal{F}_1 \cdot d) \]
\[ r = r' + r'' \]

lexicographic order!

© 2015 IBM Corporation
Our FPE: Analysis

• **Security:**
  – Only format properties preserved ⇒ security reduces to security of integer-FPE
  – Best security guarantee possible!

• **Efficiency:**
  – Ranking and unranking unavoidable in the Rank-then-Encipher method
  – Efficiency reduces to efficiency of integer-FPE
  – Medium-sized domains:
  – Large domains: only provably secure scheme [Bellare et al. ‘09] for range \( \{0,1, \ldots, M - 1\} \) first factors \( M \)
Improving Efficiency For Large Formats

- Efficiency-security tradeoff for large formats:
  - 1st solution: use FFX for integer FPE
    - Has no rigorous security analysis
  - 2nd solution: keep formats small ⇒ reduce format size
    - As we will see, this compromises security
    - We try to compromise as little as possible
- Partition message-space $\mathcal{M}$: $\mathcal{M} = \mathcal{M}_1 \cup \cdots \cup \mathcal{M}_n$
- But try to “hide” message-specific properties when possible
- Intuitively, try to increase the $\mathcal{M}_i$’s
  - Knowing $m \in \mathcal{M}_i$ still leaves “many unknowns”
The “Large Formats” Problem: Closer Look

- Inefficiency due to integer-FPE factoring domain size $M$
- Need to restrict domain size when calling integer-FPE
- Ranking and unranking is calculated in relation to $M$
- How do we rank in large formats?

**Our solution** combines:
- Delegating to sub-formats
- Parsing message to substrings $m = m_1 \ldots m_n$ and applying Rank-the-Encipher separately to every $m_i$

**Main challenge:** parsing $m$ while hiding message-specific properties
- Obtained by keeping sub-formats as large as possible
\[ F = F_1 \cup F_2 \]

$\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$

 Parsing and Ranking Union

$\cup$

© 2015 IBM Corporation
Parsing and Ranking Concatenation (1)

\[ m = m_1 \cdot d \cdot m_2 \]

\[ \mathcal{F} = \mathcal{F}_1 \cdot d \cdot \mathcal{F}_2 \]

- Ranking outputs a list
  \[ r_1 \rightarrow r_2 \]

- Each rank encrypted separately:
  \[ c_i = \text{unrank}(\text{intEnc}(r_i)) \]

- Encryption of \( m \) is concatenation:
  \[ c = c_1 \cdot c_2 \]
Parsing and Ranking Concatenation (2)

\[ m = m_1 \cdot d_1 \cdot m_2 \cdot d_2 \cdot m_3 \cdot d_3 \cdot m_4 \cdot d_4 \cdot m_5 \]

**Ranking outputs a list**

\[ r' \rightarrow r'' \rightarrow r''' \]

**Each rank encrypted separately:**

\[ c' = \text{unrank}(\text{intEnc}(r')) \]
\[ c'' = \text{unrank}(\text{intEnc}(c'')) \]
\[ c''' = \text{unrank}(\text{intEnc}(r''')) \]

**Encryption of** \( m \) **is concatenation:**

\[ c = c' \cdot c'' \cdot c''' \]

© 2015 IBM Corporation
Parsing and Ranking Range

\[ \mathcal{F} = (\mathcal{F}_1 \cdot d)^k, \ 1 \leq k \leq 4 \]

\[ m = m_1 \cdot d \cdot m_2 \cdot d \cdot m_3 \cdot d \]

ranking outputs a list

\[ r' \rightarrow r'' \]

each rank encrypted separately:

\[ c' = \text{unrank}(\text{intEnc}(r')) \]
\[ c'' = \text{unrank}(\text{intEnc}(r'')) \]

Encryption of \( m \) is concatenation:

\[ c = c' \cdot c'' \]
Security Of Our FPE

- Format sub-dividing preserve *some* message-specific properties
- The larger the sub-format, the smaller the probability of reversing encryption
- Choosing parameters “correctly” ⇒ “reasonable” tradeoff

© 2015 IBM Corporation
Our FPE: Evaluation

- Federal Election Commission (FEC) reports:
  - Name, home address, employer, job title
  - Format size $\sim 2^{856}$

- FFX achieves better performance
- Splitting significantly improves the FE1 running time
  - Setting maxSize $< 2^{256}$ has no efficiency gain
Concurrent Work

• libFTE [Luchaup et al. ’14]
  – Also employ RtE
  – Format represented by regexp
    • Regexp->DFA/NFA
    • Rank/Unrank using DFA/NFA

• Limitations:
  – Designed for developers:
    • Defining new format (regexp) requires a developer’s involvement
    • outputs several possible schemes out of which developer choses the most appropriate one
    • resultant scheme could have poor performance and there is no way to know whether a different regex would give better performance
Concurrent Work (Cont.)

• Performance of our scheme compared to libFTE:

<table>
<thead>
<tr>
<th>Type</th>
<th>#Messages</th>
<th>Initialization</th>
<th>Rank</th>
<th>Unrank</th>
<th>FFX</th>
<th>Overall</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>libFTE (DFA)</td>
<td>100000</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>110</td>
<td>121</td>
<td>113 MB</td>
</tr>
<tr>
<td>libFTE (NFA)</td>
<td>100000</td>
<td>3</td>
<td>1697</td>
<td>15</td>
<td>100</td>
<td>1814</td>
<td>865 MB</td>
</tr>
<tr>
<td>Our Scheme</td>
<td>108238</td>
<td>-</td>
<td>27</td>
<td>80</td>
<td>84</td>
<td>213</td>
<td>34 MB</td>
</tr>
</tbody>
</table>

• Running Time: libFTE is ~ twice as fast as our approach
• Memory Usage: libFTE uses ~ 3 time more memory
Our FPE: Practical Summary

- We provide an FPE for general formats
  - First framework for efficiently representing general formats
  - First scheme to eliminate cycle-walking
    - Efficiency can be measured!
  - Optimal security guarantee
  - Support of large formats
    - With best security guarantee under size limitation

- Ingredients:
  - Framework for defining general formats
  - Efficient ranking and unranking methods for general formats
  - Support of large format
    - Through user-defined upper-bound on permissible format sizes

© 2015 IBM Corporation
Thanks For Listening!