Format-Preserving Encryption
Part I: Introduction and Definitions

Mor Weiss
Technion

Winter School on Cryptography in the Cloud:
Verifiable Computation and Special Encryption
Bar-Ilan University
Why Format Preserving Encryption?
Why Format Preserving Encryption?
Why Format Preserving Encryption?

Problem (1): encrypted entry incompatible with database entry structure

Non-solution (1): generate new tables
Why Format Preserving Encryption?
Why Format Preserving Encryption?
Why Format Preserving Encryption?

Problem (2): encrypted entry incompatible with applications using data

Non-solution (2): re-write applications
Session I: Outline

• Tweakable ciphers: motivation and definition
• Format Preserving Encryption (FPE):
  – Security definitions
  – Relations between definitions
• FPE constructions
Tweakable Encryption: Introduction

• In these sessions: all encryption schemes are deterministic and private-key

• **Deterministic Encryption Scheme** $\Pi$:
  - Message space $\mathcal{M}$
  - Randomized $\text{KeyGen}: \mathbb{N} \to \mathcal{K}$
  - Deterministic $E: \mathcal{K} \times \mathcal{M} \to \mathcal{C}$
  - Deterministic $D: \mathcal{K} \times \mathcal{C} \to \mathcal{M}$

• **Semantics**: correctness and secrecy

• **Notation**:
  - $E_K = E(K, \cdot)$
  - $D_K = D(K, \cdot)$

• “Unpredictability”: only due to encryption key $K$
  - For random $K$, $E_K$ “similar” to random permutation on $\mathcal{M}$
Tweakable Encryption: Introduction (2)

• Key-provided “unpredictability” insufficient for small $\mathcal{M}$
  – Example: credit card numbers

  4385822056110982
Tweakable Encryption: Introduction (2)

• Key-provided “unpredictability” insufficient for small $\mathcal{M}$
  – Example: credit card numbers

4 38582 205611 0982
Issuer & bank number Account number Check digits
Tweakable Encryption: Introduction (2)

• Key-provided “unpredictability” insufficient for small $\mathcal{M}$
  – Example: credit card numbers

\[ E_k(205611) \]

4 38582  \hspace{1cm} 0982

Issuer & bank number  \hspace{1cm} Account number  \hspace{1cm} Check digits
Tweakable Encryption: Introduction (2)

• Key-provided “unpredictability” insufficient for small $\mathcal{M}$
  – Example: credit card numbers

  $4\ 38582\ \ E_k(205611)\ \ 0982$
Tweakable Encryption: Introduction (2)

- Key-provided “unpredictability” insufficient for small \( \mathcal{M} \)
  - Example: credit card numbers

\[
\begin{array}{c|c|c}
438582 & E_k(205611) & 0982 \\
448539 & 205611 & 2836
\end{array}
\]
Tweakable Encryption: Introduction (2)

- Key-provided “unpredictability” insufficient for small $\mathcal{M}$
  - Example: credit card numbers

\[
\begin{align*}
438582 & \quad E_k(205611) & 0982 \\
448539 & \quad E_k(205611) & 2836
\end{align*}
\]

- **Problem:** dictionary attacks allow decrypting unknown ciphertexts!

- **Want:** different plaintexts $\Rightarrow$ encryption uses different pseudorandom permutations

- **Solution:** “tweak” encryption using public info!
Tweakable Encryption: Definition

• Deterministic **Tweakable Encryption Scheme** \( \Pi [\text{LRW`02}] \):
  – Message space \( \mathcal{M} \)
  – **Tweak space** \( \mathcal{T} \)
  – Randomized \( \text{KeyGen}: \mathbb{N} \rightarrow \mathcal{K} \)
  – Deterministic \( E: \mathcal{K} \times \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{C} \)
  – Deterministic \( D: \mathcal{K} \times \mathcal{T} \times \mathcal{C} \rightarrow \mathcal{M} \)

• **Notation:**
  – \( E^T_K = E(K, T, \cdot) \)
  – \( D^T_K = D(K, T, \cdot) \)

• “Unpredictability”: **still** only due to encryption key \( K \), but...
• ... for random \( K \), \( E^T_1(K, \cdot) \), \( E^T_2(K, \cdot) \) “similar” to **independent** random permutations
• Tweaks give **family** of pseudorandom permutations
  – Different pseudorandom permutation for every plaintext
• Tweak fundamentally different than key
  – Provides **variability**, NOT **unpredictability**
### Tweakable Encryption: Example

- Deterministic encryption is problematic in small domains
  - E.g., credit card numbers
- Before:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4 38582</td>
<td>205611</td>
<td>0982</td>
</tr>
<tr>
<td>4 48539</td>
<td>205611</td>
<td>2836</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4 38582</td>
<td>(E_k(205611))</td>
<td>0982</td>
</tr>
<tr>
<td>4 48539</td>
<td>(E_k(205611))</td>
<td>2836</td>
</tr>
</tbody>
</table>
Tweakable Encryption: Example

• Deterministic encryption is problematic in small domains
  – E.g., credit card numbers

• Before:

  | 4 38582 | 205611 | 0982 |
  | 4 48539 | 205611 | 2836 |
  | 4 38582 | 849682 | 0982 |
  | 4 48539 | 849682 | 2836 |
Tweakable Encryption: Example

• Deterministic encryption is problematic in small domains
  – E.g., credit card numbers

• Before:
  4 38582  205611  0982
  4 48539  205611  2836
  4 38582  849682  0982
  4 48539  849682  2836

• Tweaks solve the problem
  – All available public info used as tweak

• Now:
  4 38582 \( E_K^{4385820982} (205611) \)  0982
  4 48539 \( E_K^{4485392836} (205611) \)  2836
Tweakable Encryption: Example

- Deterministic encryption is problematic in small domains
  - E.g., credit card numbers

- Before:
  - 4 38582 205611 0982
  - 4 48539 205611 2836

- Tweaks solve the problem
  - All available public info used as tweak

- Now:
  - $\alpha \rightarrow 4\ 38582$
  - $E_K^{(\alpha,\beta)}(205611)$
  - 0982
  - $\beta \leftarrow \alpha' \rightarrow 4\ 48539$
  - $E_K^{(\alpha',\beta')}(205611)$
  - 2836
  - $\beta' \leftarrow$
Tweakable Encryption: Example

- Deterministic encryption is problematic in small domains
  - E.g., credit card numbers
- Before:
  - 4 38582 205611 0982
  - 4 48539 205611 2836
  - 4 38582 849682 0982
  - 4 48539 849682 2836
- Tweaks solve the problem
  - All available public info used as tweak
- Now:
  - $\alpha \rightarrow 4 38582$ $E_K^{\alpha,\beta}(205611)$ $0982 \leftarrow \beta$
  - $\alpha' \rightarrow 4 48539$ $E_K^{\alpha',\beta'}(205611)$ $2836 \leftarrow \beta'$
Tweakable Encryption: Example

- Deterministic encryption is problematic in small domains
  - E.g., credit card numbers

- Before:
  - 4 38582  205611  0982
  - 4 48539  205611  2836
  - 4 38582  849682  0982
  - 4 48539  849682  2836

- Tweaks solve the problem
  - All available public info used as tweak

- Now:
  - 4 38582  237849  0982
  - 4 48539  967395  2836
Tweakable Encryption: History

• Tweakable block ciphers [LRW`02] use tweak to
  – Design better “modes of operation”
    • Instead of a fixed IV
  – Improve efficiency
    • Instead of replacing encryption key

• In small domains: tweaks are **essential**!

• Many formats for which format preserving encryption is needed are small
  – Social security numbers (SSNs), credit card numbers (CCNs),…
Format-Preserving Encryption
Format-Preserving Encryption (FPE): Introduction

• Standard encryption maps messages to “garbage”, causing
  – Applications using data to crash
  – Tables designed to store data unsuitable for storing encrypted data

• Sometimes plaintext properties should be preserved

• Want: $\mathcal{M} = \mathcal{C}$
  – i.e., $E^K_T$ is a permutation over $\mathcal{M}$

• $\mathcal{M}$ is union of messages over all supported formats
  – Supported formats are called “slices”

• Examples:
  – $\mathcal{M} = \text{SSNs} \cup \text{CCNs} \cup \text{Dates} \cup \{1, \ldots, N\}$
  – $\mathcal{M} = \bigcup_{n \in \mathbb{N}} \{0,1\}^n$
  – $\mathcal{M} = \bigcup_{n \in \mathbb{N}} \mathbb{Z}_n$
FPE: Syntactic Definition

- **Format-Preserving Encryption (FPE) \( \Pi [\text{BRRS`09}] \):**
  - Format space \( \mathcal{N} \)
  - Message space \( \mathcal{M} = \bigcup_{N \in \mathcal{N}} \mathcal{M}_N \)
    - All \( \mathcal{M}_N \)'s are finite
  - Tweak space \( \mathcal{T} \)
  - Randomized \( \text{KeyGen} : \mathbb{N} \rightarrow \mathcal{K} \)
  - **Deterministic** \( E : \mathcal{K} \times \mathcal{T} \times \mathcal{N} \times \mathcal{M} \rightarrow \mathcal{M} \cup \{ \perp \} \)
    - \( \perp \notin \mathcal{M} \)
    - \( E(K,T,N,m) = \perp \) denotes encryption error \( (m \notin \mathcal{M}_N) \)
      - Failure depends only on \( N, m \) and **not** on \( K, T \)
    - \( E(K,T,N,\cdot) \) is a permutation **over** \( \mathcal{M}_N \)
  - **Deterministic** \( D : \mathcal{K} \times \mathcal{T} \times \mathcal{N} \times \mathcal{M} \rightarrow \mathcal{M} \cup \{ \perp \} \)

- **Notation:**
  - \( E_{K}^{T,N} = E(K,T,N,\cdot) \)
  - \( D_{K}^{T,N} = D(K,T,N,\cdot) \)
FPE: Semantic Definition

- **Correctness:** for every $K \in \mathcal{K}$, every $T \in \mathcal{T}$, every $N \in \mathcal{N}$ and every $m \in \mathcal{M}_N$
  
  \[ D_K^{T,N} \left( E_K^{T,N}(m) \right) = m \]

- **Security:**
  
  - Hierarchy of security notions [BRRS`09]
  
  - Strongest: **Pseudo-Random Permutation (PRP)** security
    
    - $K$ random $\Rightarrow E_K^{T,N}$ close to pseudorandom permutation on $\mathcal{M}_N$
Pseudo-Random Permutation (PRP) security

\[ \forall T \in T \quad \forall N \in \mathcal{N} \quad \pi^{T,N} \leftarrow \text{Perm}(\mathcal{M}_N) \]

\[ b = 0 \quad b = 1 \]

\[ b' \]

\[ K \leftarrow \text{KeyGen}(1^n) \]

\[ (T, N, m) \quad E_k^{T,N}(m) \]
Pseudo-Random Permutation (PRP) security

\[ \forall T \in \mathcal{T}, \forall N \in \mathcal{N}, \pi_{T,N} \leftarrow \text{Perm}(\mathcal{M}_N) \]

\[ K \leftarrow \text{KeyGen}(1^n) \]

\[ \text{Adv}^{PRP}_\mathcal{A} = 2 \Pr[b = b'] - 1 \]
FPE: Security Definitions (2)

• Hierarchy of security notions [BRRS`09]
• Strongest: **Pseudo-Random Permutation (PRP)** security
  – $K$ random $\Rightarrow E_{K}^{T,N}$ close to pseudorandom permutation on $\mathcal{M}_N$
  – Guaranteed security against (improbable) attacks incurs expensive overhead
  – “Overkill” for typical applications

• **Single Point Indistinguishability (SPI)** security
  – Adversary cannot distinguish encryption of **single** point of its choice from random
  – Analogous to PRF and PRP security notions [GGM`84, DM`00, MRS`09]
Single Point Indistinguishability (SPI) Security

ideal

challenger

\[ K \leftarrow \text{KeyGen}(1^n) \]

\[ c^* \leftarrow M_{N^*} \]

real

adversary

\[ b = 0 \]

\[ b = 1 \]

challenger

\[ K \leftarrow \text{KeyGen}(1^n) \]

\[ c^* \leftarrow E_{K^*,N^*}^T(m^*) \]

\[ c^* \leftarrow E_{K}^{T,N}(m) \]

unique

\[ (T, N, m) \]

\[ (T, N, m) \]

b

b'

b

b'
Single Point Indistinguishability (SPI) Security

$$Adv^S_{\mathcal{A}} = 2 \Pr[b = b'] - 1$$
Why SPI?

- **Pseudo-Random Permutation (PRP)**
  - Adversary cannot distinguish encryption oracle from random permutations
  - \( \text{Adv}^{\text{PRP}}_{\mathcal{A}} = 2 \Pr[b = b'] - 1 \)

- **Single Point Indistinguishability (SPI)**
  - Adversary cannot distinguish encryption of single point of its choice from random
    - Even given encryption oracle
  - \( \text{Adv}^{\text{SPI}}_{\mathcal{A}} = 2 \Pr[b = b'] - 1 \)

- Equivalent notions, SPI easier to work with

- **PRP ⇒ SPI**: \( \text{Adv}^{\text{SPI}}_{\mathcal{A}} \leq 2 \cdot \text{Adv}^{\text{PRP}}_{\mathcal{A}}' + \frac{q}{M} \)
  - \( q \) = number of queries of PRP adversary
  - \( M \) = minimal size of supported format

- **SPI ⇒ PRP**: \( \text{Adv}^{\text{PRP}}_{\mathcal{A}} \leq q \cdot \text{Adv}^{\text{SPI}}_{\mathcal{A}}' + \frac{q^2}{M} \)
FPE: Security Definitions (3)

• Hierarchy of security notions [BRRS`09]
  - Strongest: **Pseudo-Random Permutation (PRP)** security
    - $K$ random $\Rightarrow E_{K}^{T,N}$ close to pseudorandom permutation on $\mathcal{M}_{N}$

• **Single Point Indistinguishability (SPI)** security
  - Adversary cannot distinguish encryption of single point of its choice from random

• **Message Privacy (MP)** security
  - “Format-preserving” analog of semantic security
  - Challenge ciphertext $c^*$ **practically** no help in computing $f(m^*)$
  - **Randomized** encryption: “practically” = no help
  - **Deterministic** encryption: “practically” = encryption oracle equivalent to equality oracle
Message Privacy (MP) Security

**Ideal**

- **Challenger**:
  - $D$ 
  - $(T^*, N^*, m^*) \leftarrow D$

- **Adversary**:
  - $T^*, N^*$
  - $m$?
  - $q$ \(\leftarrow\) Yes\No

  - Same as in real world

**Real**

- **Challenger**:
  - $K \leftarrow \text{KeyGen}(1^n)$
  - $(T^*, N^*, m^*) \leftarrow D$
  - $c^* \leftarrow E_{K, N^*}^{T^*}(m^*)$

- **Adversary**:
  - $T^*, N^*, c^*$
  - $q$ \(\leftarrow\) $E_{K}^{T, N}(m)$
  - $q$ \(\leftarrow\) $E_{K}^{T, N}(m)$

  - Guess for $f(m^*)$
Message Privacy (MP) Security

- **Ideal Scenario**
  - Challenger: $(T^*, N^*, m^*) \leftarrow \mathcal{D}$
  - Adversary: $T^*, N^*$
  - Same as in real world

- **Real Scenario**
  - Challenger: $K \leftarrow \text{KeyGen}(1^n)$
  - $K \leftarrow \text{KeyGen}(1^n)$
  - $c^* \leftarrow E_K^{T^*, N^*}(m^*)$
  - $T^*, N^*, c^*$

- **Guess for $f(m^*)$**
  - Same as in real world

**Advantage**

$$\text{Adv}_{\mathcal{A}}^{MP} = \Pr_{\text{real}} \left[ z = f(m^*) \right] - \max_{\text{ideal Lucy}} \Pr_{\text{ideal}} \left[ z = f(m^*) \right]$$
FPE: Security Definitions (4)

- Hierarchy of security notions [BRRS`09]
- Strongest: **Pseudo-Random Permutation (PRP)** security
  - $K$ random $\Rightarrow E_K^{T,N}$ close to pseudorandom permutation on $\mathcal{M}_N$
- **Single Point Indistinguishability (SPI)** security
  - Adversary cannot distinguish encryption of single point of its choice from random
- **Message Privacy (MP)** security
  - “Format-preserving” analog of semantic security
  - Challenge ciphertext $c^*$ practically no help in computing $f(m^*)$
- **Weakest:** **Message Recovery (MR)** security
  - Adversary cannot **completely** recover challenge plaintext
Message Recovery (MR) Security

\[ (T^*, N^*, m^*) \leftarrow D \]

ideal

challenger

adversary

Same as in real world

\[ K \leftarrow \text{KeyGen}(1^n) \]

\[ (T^*, N^*, m^*) \leftarrow D \]

\[ c^* \leftarrow E_{K,T^*,N^*}(m^*) \]

real

challenger

adversary

\[ (T, N, m) \leftarrow D \]

\[ E_{K,T,N}(m) \leftarrow D \]
Message Recovery (MR) Security

\[ Adv_A^{MR} = \Pr_{\text{real}}[m' = m^*] - \max_{\text{ideal Lucy}} \Pr[m' = m^*] \]
FPE: Security Definitions (5)

• Hierarchy of security notions [BRRS`09]
  – Pseudo-Random Permutation (PRP)
  – Single Point Indistinguishability (SPI)
  – Message Privacy (MP)
  – Message Recovery (MR)

• Similar to IND-Distinct\textbf{CPA} security

• Extends to stronger IND-Distinct\textbf{CCA} security:
  – \textbf{Strong-PRP}:
    • Real world: adversary given $D_{K}^{T,N}$
    • Ideal world: adversary given $(\pi^{T,N})^{-1}$
  – \textbf{Strong SPI, MP, MR}:
    • Real world: adversary given $D_{K}^{T,N}$
    • Ideal world: simulator given no additional oracle

• We will work with IND-CPA notions (no decryption oracle)
Relations Between Security Definitions

\[ PRP \iff SPI \implies MP \implies MR \]

**PRP**: Pseudo Random Permutation  
**SPI**: Single Point Indistinguishability  
**MP**: Message Privacy  
**MR**: Message Recovery

- \( PRP \implies SPI \): \( \text{Adv}_{\mathcal{A}}^{SPI} \leq 2 \cdot \text{Adv}_{\mathcal{A}'}^{PRP} + \frac{q}{M} \)

- \( SPI \implies PRP \): \( \text{Adv}_{\mathcal{A}}^{PRP} \leq q \cdot \text{Adv}_{\mathcal{A}'}^{SPI} + \frac{q^2}{M} \)
Relations Between Security Definitions (2)

\[ PRP \iff SPI \implies MP \implies MR \]

- **PRP**: Pseudo Random Permutation
- **SPI**: Single Point Indistinguishability
- **MP**: Message Privacy
- **MR**: Message Recovery

- \( MP \implies MR \): \( \text{Adv}_{\mathcal{A}}^{MR} \leq \text{Adv}_{\mathcal{A}'}^{MP} \)
  - MR special case of MP: \( \mathcal{A}' \) chooses \( f = \text{identity function} \)

### PRP

**Ideal**

\[ (T', N', m') \leftarrow \mathcal{D} \]

**Real**

\[ K \leftarrow \text{KeyGen}(1^n) \]
\[ (T', N', m') \leftarrow \mathcal{D} \]
\[ c^* \leftarrow E_{K}^{T', N'}(m') \]

**Challenger**

\[ q \rightarrow m? \]
\[ q \rightarrow c^* \]

**Adversary**

\[ \mathcal{D} \]

\[ \text{Same as in real world} \]

\[ m' \]

\[ m^* \]

- **MR Special Case**

**Ideal**

\[ (T', N', m') \leftarrow \mathcal{D} \]

**Real**

\[ K \leftarrow \text{KeyGen}(1^n) \]
\[ (T', N', m') \leftarrow \mathcal{D} \]
\[ c^* \leftarrow E_{K}^{T', N'}(m') \]

**Challenger**

\[ q \rightarrow m? \]
\[ q \rightarrow c^* \]

**Adversary**

\[ \mathcal{D} \]

\[ \text{Same as in real world} \]

\[ m' \]

\[ m^* \]

- **MR**

**Ideal**

\[ (T', N', m') \leftarrow \mathcal{D} \]

**Real**

\[ K \leftarrow \text{KeyGen}(1^n) \]
\[ (T', N', m') \leftarrow \mathcal{D} \]
\[ c^* \leftarrow E_{K}^{T', N'}(m') \]

**Challenger**

\[ q \rightarrow m? \]
\[ q \rightarrow c^* \]

**Adversary**

\[ \mathcal{D} \]

\[ \text{Guess for } f(m') \]

**Same as in real world**

\[ f \]

\[ m' \]
Relations Between Security Definitions (3)

\[ PRP \iff SPI \Rightarrow MP \Rightarrow MR \]

PRP: Pseudo Random Permutation
SPI: Single Point Indistinguishability
MP: Message Privacy
MR: Message Recovery

- \( SPI \Rightarrow MP \): \( \text{Adv}_{\mathcal{A}}^{MP} \leq \text{Adv}_{\mathcal{A}'}^{SPI} \)
Relations Between Security Definitions (4)

\[ PRP \iff SPI \iff MP \iff MR \]

**PRP:** Pseudo Random Permutation

**SPI:** Single Point Indistinguishability

**MP:** Message Privacy

**MR:** Message Recovery

- \( MR \not\iff MP \):
  - E.g., \( \mathcal{M}_N = \{0,1\}^N \) for all \( N \in \mathbb{N} \), and encryption is:
    - Identity on first plaintext bit
    - Pseudorandom on other plaintext bits
  - \( \text{Adv}^{MP}(\mathcal{A}) = \frac{1}{2} \) Compare \( \mathcal{A} \) to “best possible” ideal \( \mathcal{A} \)
  - \( \text{Adv}^{MR}(\mathcal{A}) \approx 2^{-(N-1)} \)
Relations Between Security Definitions (5)

\[ \text{PRP} \iff \text{SPI} \not\iff \text{MP} \not\iff \text{MR} \]

**PRP:** Pseudo Random Permutation  
**SPI:** Single Point Indistinguishability  
**MP:** Message Privacy  
**MR:** Message Recovery

- **MP \not\iff SPI:**
  - e.g., encryption has “fixed point” \( m_N \in \mathcal{M}_N \) for every \( N \) and every \( K, T \):
    - \( \pi_{K,T}^{T,N} \) is pseudorandom permutation over \( \mathcal{M}_N \setminus \{m_N\} \)
    - \( m \neq m_N \Rightarrow E_{K,T}^{T,N}(m) = \pi_{K,T}^{T,N}(m) \)
    - \( E_{K,T}^{T,N}(m_N) = m_N \)
Relations Between Security Definitions (6)

\[ \text{PRP} \iff \text{SPI} \iff \text{MP} \iff \text{MR} \]

**PRP**: Pseudo Random Permutation  
**SPI**: Single Point Indistinguishability  
**MP**: Message Privacy  
**MR**: Message Recovery

- **MP \not\iff SPI**:
  - e.g., encryption has “fixed point” \( m_N \in \mathcal{M}_N \) for every \( N \) and every \( K, T \)
  - \( A^{\text{SPI}} \) chooses challenge plaintext \( (T, N, m_N) \) for maximal \( |\mathcal{M}_N| \)
    - Has advantage \( 1 - \frac{1}{|\mathcal{M}_N|} \)
    - “Best” \( A^{\text{MP}} \): choose “easy to guess” \( f \) or \( m \)

**SPI**
- Also “easy to guess” in ideal world

**MP**
- \( K \leftarrow \text{KeyGen}(1^n) \)
- \( c^* \leftarrow \mathcal{M}_N^* \)
- \( c^* \leftarrow E_K^{T, N^*}(m^*) \)
- \( T^*, N^*, m^* \leftarrow D \)
- \( T^*, N^*, c^* \leftarrow D \)
- \( f, z \)

---

** ideal**

- \( b = 0 \)
- \( b = 1 \)
- \( T, N, m \)
- \( E_K^{T, N}(m) \)
- \( E_K^{T, N^*}(m^*) \)
- \( E_K^{T, N}(m) \)
- \( E_K^{T, N^*}(m^*) \)
- \( T^*, N^*, m^* \)

---

** real**

- \( b' \)
- \( b' \)
- \( T, N, m \)
- \( E_K^{T, N}(m) \)
- \( E_K^{T, N^*}(m^*) \)
- \( T^*, N^*, c^* \)
- \( f, z \)

---

**Same as in real world**

- \( m? \)
- \( q \)
- \( q \)

---

**Same as in real world**

- \( D \)
- \( T^*, N^*, c^* \)

---

**Same as in real world**

- \( E_K^{T, N}(m) \)
- \( E_K^{T, N^*}(m^*) \)
- \( f, z \)
Recap

• **Tweakable ciphers:** parameterized by key $K$ and tweak $T$
  – Tweak “equivalent” to using pseudorandom permutation family
  – Essential when encrypting small domains

• **Format Preserving Encryption:** preserves message format
  – Hierarchy of security definitions: $PRP \Leftrightarrow SPI \Rightarrow MP \Rightarrow MR$

  **PRP**: Pseudo Random Permutation
  **SPI**: Single Point Indistinguishability

  • $PRP \Rightarrow SPI$: $Adv_{A}^{SPI} \leq 2 \cdot Adv_{A'}^{PRP} + \frac{q}{M}$
  • $SPI \Rightarrow PRP$: $Adv_{A}^{PRP} \leq q \cdot Adv_{A'}^{SPI} + \frac{q^2}{M}$
  • $SPI \Rightarrow MP \Rightarrow MR$ with tight bounds
  • $MR \not\Rightarrow MP \not\Rightarrow SPI$
Constructions
What We Know About FPE

• First* FPE

• AES
• Term coined by Terence Spies, Voltage Security’s CTO
• First formal definitions due to [BRRS`09]
• Constructions for specific formats
  – Social Security Numbers (SSNs) [Hoo`11]
  – Credit Card Numbers (CCNs)
  – Dates [LJLC`10]
  – ...

• Drawbacks:
  – Designed for specific formats
  – New encryption techniques, little (if any) security analysis
  – Often inconsistent with syntactic definition

• Interested in schemes for general formats
  – Starting point: schemes for integral domains
Format-Preserving Encryption
Part II: Integral Domains
Session II: Outline

- Integral and “almost-integral” domains
- Feistel Networks and Generalized Feistel Network
- Integer-FPE constructions from Feistel Networks
- Integer-FPE standards
Integer-FPEs

- In many cases, interested in encrypting integral domains
  - E.g., credit-card numbers
- FPEs for integral (and “almost integral”) domains useful for encrypting general formats
  - Stay tuned...
- **Int-FPE**: FPE for integral domain $\mathbb{Z}_M$ \cite{BR02,BRRS09}
- Also interested in FPEs for “almost integral” domains $\mathcal{M} = \{0,1,\ldots,m-1\}^n$ for $n,m \in \mathbb{N}$
  - Methods described as early as 1981
  - FFX \cite{BRS10}, BPS \cite{BPS10} under NIST consideration
- We will refer to both as “int-FPE”
- Many constructions based on Feistel Networks
Integer-FPE: Constructions

• “Tiny” domains $\mathbb{Z}_M$: spending $O(M)$ time/space is feasible
  – Using card shuffles [Dur`98,FY`38,Knu`69,MO`63,San`98]
  – Using block ciphers [BR`02]

• “Small” domains $\mathbb{Z}_M$ or $\{0, 1, \ldots, m-1\}^n$:
  
  $M, m^n \leq$ domain of underlying block cipher
  
  – Based on Feistel networks for
    • $\mathbb{Z}_{ab}$ [BR`02,BRRs`09]
    • $\{0,1\}^n$ [Fei`74,AB`96,Luc`96,SK`96]
    • $\{0,1, \ldots, m-1\}^n$ [BRS`10,BPS`10] (with some size restrictions)

  – Based on card shuffling for $\mathbb{Z}_M$
    • Obtained as special case of Feistel network, or inefficient [Tho`73,GP`07]

• “Huge” domains $\{0, 1\}^n$:

  $2^n >$ domain of underlying block cipher

  – Constructions based on block ciphers, e.g.,
    [ZMI`89,Hal`04,HR`04,MF`07,CS`08,Sar`08,SAR`11]
Feistel-Based Integer FPEs
Feistel Networks [Smi`71,Fei`74,FNS`75]
Feistel Networks [Smi`71,Fei`74,FNS`75]

Balanced:
\[ |L_i| = |R_i| \]

Unbalanced:
\[ |L_i| \neq |R_i| \]

\[ R_1 = F_{k_1}(R_0) \oplus L_0 \]

\[ L_1 = R_0 \]
Feistel Networks (2)

Balanced: $|L_i| = |R_i|$  

Unbalanced: $|L_i| \neq |R_i|$  

Alternating: $|L_i| \neq |R_i|$
Generalized Feistel Networks

- Classic Feistel networks defined over bit strings
- First generalized to integral domains $\mathbb{Z}_{ab}$ by [BR`02]
  - Used alternating Feistel
- Tweakable Feistel for $\mathbb{Z}_{ab}$ described in [BRRS`09] (FE1 and FE2)
  - Tweakable round function $F$
  - Tweak of $F$ includes all public info (round #, provided tweak, format)
  - Use either alternating or unbalanced Feistel
- Operations computed modulo $b$

Example: $a = 500$, $b = 200$ (generally, $b \leq a$), input $61956$

\[
123 \cdot a + 456 = 61956
\]
Generalized Feistel Networks

- Classic Feistel networks defined over bit strings
- First generalized to integral domains $\mathbb{Z}_{ab}$ by [BR`02]
  - Used alternating Feistel
- Tweakable Feistel for $\mathbb{Z}_{ab}$ described in [BRRS`09] (FE1 and FE2)
  - Tweakable round function $F$
  - Tweak of $F$ includes all public info (round #, provided tweak, format)
  - Use either alternating or unbalanced Feistel
- Operations computed modulo $b$

- Example: $a = 500$, $b = 200$ (generally, $b \leq a$), input $61956$

\[
\begin{align*}
X' &= 456 \cdot b + 95 \\
   &= 182 \cdot a + 295
\end{align*}
\]
Generalized Feistel Networks (2)

- Feistel for $\mathbb{Z}_M$, $M = ab$: given format size $M$, requires factoring $M$
  - Highly inefficient for large $M$!
- Can we avoid factoring?
- Feistel for $\{0, 1, \ldots, m - 1\}^n$ for $n, m \in \mathbb{N}$ [BRS`10,BPS`10]
  - Operations computed coordinate-wise or block-wise (mod $m^{|R|}$)
- **Example**: $m = n = 10$, $|L| = |R| = 5$, input $1234567890$

coordinate-wise: mod 10

block-wise: mod $10^5$
Generalized Feistel Networks (3)

- Feistel for $\mathbb{Z}_M, M = ab$: given format size $M$, requires factoring $M$
  - Highly inefficient for large $M$!
- Can we avoid factoring?
- Feistel for $\{0,1, \ldots, m-1\}^n$ for $n, m \in \mathbb{N}$ [BRS`10,BPS`10]
  - Operations computed coordinate-wise of block-wise (mod $m^{|R|}$)
- **Efficiency**: no factoring
- Generalized Feistel networks $\Rightarrow$ int-FPE for domains $\mathbb{Z}_M, M = ab$
  and $\{0,1, \ldots, m-1\}^n$
  - Main issue: choosing network parameters
    - Round function, # rounds, operation and network type...
Security of Feistel Networks

• Main approach for block cipher constructions
• Intensively studied for over 3 decades
  – Security proofs (e.g., [LR`88, Mau`92, NR`97, Vau`98, Pat`98, MP`03, Pat`03, MRS`09, Pat`10, LP`12])
  – Attacks (e.g., [Pat`01, Pat`04, PNB`06, PNB`07])

• **Security measure**: PRP or strong-PRP security (random round functions)
  – Also: attacks exploiting round function structure, or allowing adversary oracle access to round functions

• **Parameters of interest**: 
  – # queries
  – Running time

• Parameters of interest influence choice of round number

• Huge gap between security guarantees and known attacks
  – In part due to *highly* inefficient information theoretic attacks
  – Major open problem!
Security of Generalized Feistel Networks

• Generalized Feistel (almost) as secure as standard Feistel
  – But not as well studied

• **Standard Feistel:** security follows from pseudo-randomness of $F$

• **Generalized Feistel:**
  – Output $z$ of $F$ pseudorandom in $2^n > |\mathbb{Z}_a|$
  – Output used in generalized Feistel is $z \mod a$

• Mod operation preserves pseudo-randomness [BRRS`09]:
  
  \[(z \mod a) \text{ is } \frac{a}{2^{n-2}}\text{-statistically close to random } z' \in_R \mathbb{Z}_a\]
### Security of Generalized Feistel Networks (2)

<table>
<thead>
<tr>
<th>Domain</th>
<th>Network type</th>
<th>Number queries $q$</th>
<th>Number rounds $r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{Z}_M, M = ab$</td>
<td>unbalanced, contracting</td>
<td>$q \approx a^{1-\epsilon}$</td>
<td>$r = O\left(\frac{[\log_b a]}{\epsilon}\right)$</td>
</tr>
<tr>
<td>$a &gt; b$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mathbb{Z}_M, M = ab$</td>
<td>unbalanced, expanding</td>
<td>$q \approx a^{1-\epsilon}$</td>
<td>$r = O\left(\frac{[\log_a b]}{\epsilon}\right)$</td>
</tr>
<tr>
<td>$a \leq b$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>${0, 1, \ldots, m-1}^N$, $N = 2n$</td>
<td>balanced</td>
<td>$q \approx m^{n(1-\epsilon)}$</td>
<td>$r = O\left(\frac{1}{\epsilon}\right)$</td>
</tr>
<tr>
<td>${0, 1, \ldots, m-1}^N$, $N = n+k$, $n &gt; k$</td>
<td>unbalanced, contracting*</td>
<td>$q \approx m^{n(1-\epsilon)}$</td>
<td>$r = O\left(\frac{n}{\epsilon k}\right)$</td>
</tr>
<tr>
<td>${0, 1, \ldots, m-1}^N$, $N = n+k$, $n \leq k$</td>
<td>unbalanced, expanding**</td>
<td>$q \approx m^{n(1-\epsilon)}$</td>
<td>$r = O\left(\frac{n}{\epsilon k}\right)$</td>
</tr>
</tbody>
</table>

Security bounds from [HR`10]
*bound improves with imbalance
**bound deteriorates with imbalance
Int-FPE (Soon To Be*) Standards

• **Recall:** generalized Feistel networks ⇒ int-FPE for domains $\mathbb{Z}_M, M = ab$ and $\{0,1, ..., m - 1\}^n$
  – Main issue: choosing network parameters
    • Round function, # rounds, operation and network type...

• Two Feistel-based int-FPE schemes for $\{0,1, ..., m - 1\}^n$ currently under NIST consideration:
  – FFX [BRS`10]
  – BPS [BPS`10]
FFX [BRS`10]

• Highly parameterized:
  – Format structure: $m, n$ ($100 \leq m^n \leq 2^{128}$)
    • No mode of operation
  – Round function $F$ (and key space)
    • E.g., CBC-MAC, CMAC, HMAC
  – # rounds, tweak space
    • Tweak should include all public info
  – Network structure: alternating/unbalanced; block/coordinate-wise operation; imbalance factor

• Security goal: strong-PRP against $m^n - 2$ queries in time < exhaustive key search
  – “Suggested” (conservative) # rounds based on known results
  – Shorter input $\Rightarrow$ more rounds

• Variants for useful domains:
  – FFX-A2: bit strings, lengths 8-128 (12-36 rounds)
  – FFX-A10: decimal strings, lengths 4-36 (12-24 rounds)
• **Construction parameters:**
  – **Format structure:** $m, n$ for any $m, n \in \mathbb{N}$
    • Mode of operation for long messages ($\#$ blocks $\leq 2^{16}$)
  – **Round function** $F$ (and key space)
    • E.g., AES, TDES, SHA-2
  – **$\#$ rounds** (even $\geq 8$)

• **Construction constants:**
  – **Tweak space:** $\{0,1\}^{64}$
    • Tweak should include all public info (long tweaks hashed)
  – **Network structure:**
    • Alternating, maximally balanced
    • Coordinate-wise operation in Feistel, block-wise in mode of operation
  – **Mode of operation:** CBC (block size $= 2 \times \log_m \left[2^{\text{input length to } F \text{ minus 32}}\right]$)

• **Security goal:** PRP-security against $m^n$ queries (no time bound)
  – “Suggested” $\#$ rounds based on known attacks and security analysis
  – $\#$ rounds fixed to 8 for all input lengths
## Int-FPE (Soon To Be*) Standards (2)

<table>
<thead>
<tr>
<th></th>
<th>FFX [BRS`10]</th>
<th>BPS [BPS`10]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Domain size</strong></td>
<td>( M = m^n )</td>
<td>arbitrary</td>
</tr>
<tr>
<td><strong>Security goal</strong></td>
<td>strong-PRP ( q = m^n - 2 )</td>
<td>PRP ( q = m^n )</td>
</tr>
<tr>
<td></td>
<td>( T &lt; \text{exhaustive key search} )</td>
<td>( \text{no time bound} )</td>
</tr>
<tr>
<td><strong>Efficiency</strong></td>
<td>more rounds ( \text{(conservative bounds)} )</td>
<td>8 rounds ( \text{(less conservative)} )</td>
</tr>
<tr>
<td></td>
<td>more calls to ( F )</td>
<td>less calls to ( F )</td>
</tr>
<tr>
<td></td>
<td>( \text{(defeat strong attacks)} )</td>
<td>( \text{(strong attacks outside of security goal)} )</td>
</tr>
<tr>
<td>**Suggested ( F )</td>
<td>CBC-MAC, CMAC, HMAC</td>
<td>AES, TDES, SHA-2</td>
</tr>
<tr>
<td><strong>Flexibility</strong></td>
<td><img src="https://example.com" alt="User Defined" /></td>
<td><img src="https://example.com" alt="Fixed" /></td>
</tr>
<tr>
<td>(tweak space and</td>
<td>user defined</td>
<td>fixed</td>
</tr>
<tr>
<td>network structure)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Format-Preserving Encryption
Part III: General Formats
Post-Lunch Recap

• Format Preserving Encryption (FPE):
  – Preserves message format
  – Tweakable, deterministic, private key
  – Useful for:
    • Storing data at remote servers
    • Running applications for (unencrypted data) on encrypted data

• Hierarchy of security notions: $PRP \Leftrightarrow SPI \Rightarrow MP \Rightarrow MR$

• Int-FPE based on generalized Feistel networks
  – For $\mathbb{Z}_M, M = ab$
  – For $\{0,1, \ldots, m-1\}^n$
Session III: Outline

• Techniques for general-format FPE
• Natural FPE construction: analysis and insecurities
• FPE constructions for general formats:
  – From regular expressions and relaxed ranking
  – From bottom-up framework and (standard) ranking
Techniques for General-Format FPE (Part 1)

• How to encrypt social security numbers (SSNs)?
  – Subset of \(\{0,1,\ldots,9\}^9\)
  – Additional constraints

• We have FPE for \(\mathcal{M} = \{0,1,\ldots,9\}^9\)

• Can get FPE for SSNs from FPE for \(\mathcal{M}\):
  • Use cycle walking [SO`98,BR`02]
    “if at first you don’t succeed, pick yourself up and try again”
    – Use “standard” FPE for \(\mathcal{M} = \{0,1,\ldots,9\}^9\)
    – Repeat until ciphertext is valid SSN
Cycle Walking

Message space $\mathcal{M} = \{0, 1, \ldots, 9\}^9$

Valid SSNs
Cycle Walking: Security Analysis

- **Want:** FPE for $\mathcal{M}$
  - Encryption “looks like” random permutation on $\mathcal{M}$
- **Have:** ideal FPE for $\mathcal{M}'$, $\mathcal{M} \subseteq \mathcal{M}'$ with encryption $E'_K$
  - Ideal FPE: each permutation on $\mathcal{M}'$ induced by single key
- $E^{CW}_K :=$ apply cycle walking to $E'_K$ until ciphertext in $\mathcal{M}$
- For random $K$, $E^{CW}_K$ is random permutation on $\mathcal{M}$ [BR`02]
  - Enough to show all permutations $\pi$ on $\mathcal{M}$ obtained by same number of keys $K$
  - Adding one element $x \in \mathcal{M}' \setminus \mathcal{M}$ to $\pi$: $\Sigma_{i=1}^r \ell_i + 1$ options
    - $|\mathcal{M}'| + 1$ options of adding $x$
  - General case follows by induction on $k = |\mathcal{M}'| - |\mathcal{M}|$

![Diagram](Image)
Cycle Walking: Efficiency Analysis

- \( \mathcal{M} \subseteq \mathcal{M}' \)

- \( E_{K}^{CW} \) for \( \mathcal{M} \) obtained from cycle walking on \( E_{K}' \) for \( \mathcal{M}' \)

- Single \( E_{K}^{CW} \) call requires \textit{on average} \( \frac{|\mathcal{M}|}{|\mathcal{M}'|} \) calls to \( E_{K}' \)
  - No timing attacks due to repeated encryption [BRRS`09] (cycle length independent of plaintext)

- No bound on actual efficiency
  - But... for “good” FPE (=like PRP) on \( \mathcal{M}' \): bound close to average
Cycle Walking: Summary

• $\mathcal{M} \subseteq \mathcal{M}'$

• $E_K^{cw}$ for $\mathcal{M}$ obtained from cycle walking on $E_K'$ for $\mathcal{M}'$

• **Cons:** Efficiency loss
  – single $E_K^{cw}$ call = multiple $E_K'$ calls
  – Average, not worst case, bound
  – Even average bound (typically $\approx 2$) sometimes too expensive

• **Pros:** can use known schemes (e.g., Feistel)
  – Inherit security

But... can also be obtained *without* cycle walking

• Would like to avoid when possible
  – E.g., design dedicated int-FPE schemes
Techniques for General-Format FPE (Part 2)

- Rank-then-Encipher (RtE) [BRRS`09]: general-format FPEs from int-FPE
  - Order $\mathcal{M}$ arbitrarily: $\text{rank}: \mathcal{M} \rightarrow \{1, \ldots, M\}$
Techniques for General-Format FPE (Part 2)

- Rank-then-Encipher (RtE) [BRRS`09]: general-format FPEs from int-FPE
  - Order $\mathcal{M}$ arbitrarily: \textbf{rank}: $\mathcal{M} \rightarrow \{1, \ldots, M\}$
Techniques for General-Format FPE (Part 2)

• **Rank-then-Encipher (RtE) [BRRS`09]: general-format FPEs from int-FPE**
  
  – Order $\mathcal{M}$ arbitrarily: $\textbf{rank}: \mathcal{M} \rightarrow \{1, \ldots, M\}$
  
  – To encrypt message $m$:
    
    • **Rank $m$:** $i = \text{rank}(m)$
    • **Encipher $i$:** $j = \text{intE}(K, i)$
    • **Unrank $j$:** $c = \text{rank}^{-1}(j)$
Techniques for General-Format FPE (Part 2)

- **Rank-then-Encipher (RtE) [BRRS`09]:** general-format FPEs from **int-FPE**
  - Order $\mathcal{M}$ arbitrarily: $\text{rank}: \mathcal{M} \rightarrow \{1, \ldots, M\}$
  - To encrypt message $m$:
    - **Rank $m$:** $i = \text{rank}(m)$
    - **Encipher $i$:** $j = \text{intE}(K, i)$
    - **Unrank $j$:** $c = \text{rank}^{-1}(j)$
Techniques for General-Format FPE (Part 2)

• **Rank-then-Encipher (RtE)** [BRRS`09]: general-format FPEs from int-FPE
  - Order $\mathcal{M}$ arbitrarily: $\textbf{rank}: \mathcal{M} \rightarrow \{1, \ldots, M\}$
  - To encrypt message $m$:
    • **Rank $m$**: $i = \text{rank}(m)$
    • **Encipher $i$**: $j = \text{intE}(K, i)$
    • **Unrank $j$**: $c = \text{rank}^{-1}(j)$

• **Security**: from security of int-FPE
  - rank not meant to, and does not, add security

• **Efficiency**: only if rank, unrank are efficient

• **Main challenge**: design efficient ranking procedures
  - “Meta” technique for regular languages [BRRS`09]
Constructing General-Format FPE

- **Goal:** design FPE supporting general formats
  - E.g., SSNs, CCNS, dates, names, addresses...
- **Main tool:** RtE Technique
- **Main challenges:**
  - Designing efficient ranking procedures
  - Representing formats
Simplification-Based FPE [MYHC`11,MSP`11]

• Represent formats as union of simpler sub-formats
  – Messages interpreted as strings
  – \( \mathcal{M} \) divided into subsets \( \mathcal{M}_1, \ldots, \mathcal{M}_k \) defined by
    • Length
    • Index-specific character sets

• Encrypt each \( \mathcal{M}_i \) separately using Rank-then-Encipher
  – Ranking computed using generalizes decimal counting method

\( \mathcal{F}_{name} \): format of valid names
**Name:** 1-4 space-separated words
**Word:** upper case letter followed by 1-15 lower case letters

**Subsets:**
- \( \mathcal{M}_1 \) contains Al
- \( \mathcal{M}_2 \) contains Tal
- \( \mathcal{M}_{15} \) contains Muthuramakrishna
- \( \mathcal{M}_{16} \) contains El Al
  \( \mathcal{M}_5 \) contains Migel
Simplification-Based FPE: Security Concerns

• **The problem:** encryption preserves *message-specific* properties
  – Length and character type at each location
  – John Doe can encrypt Jane Lee but not Johnnie Smith

• Scheme insecure both in theory and practice [WRB`15]
  – **Practice:** experimental results
    • Ciphertext usually completely reveals plaintext
    • Worse than not encrypting at all...
  – **Theory:** scheme is MR (message recovery) insecure
    • Implies insecurity according to all FPE security notions
Message Recovery (MR)

\begin{align*}
&T^*, N^*, m^* \leftarrow D \\
&T^*, N^* \\
&m? \\
&q \quad \text{Yes/No} \\
&\text{Same as in real world} \\
\end{align*}

\begin{align*}
&K \leftarrow \text{KeyGen}(1^n) \\
&(T^*, N^*, m^*) \leftarrow D \\
&c^* \leftarrow E_{K, T^*, N^*}^N(m^*) \\
&T^*, N^*, c^* \\
&q \\
&(T, N, m) \leftarrow E_{K, T^*, N^*}^N(m) \\
&m' \\
\end{align*}
Message Recovery (MR)

\[ \text{Adv}^{\text{MR}}_A = \Pr_{\text{real}} [m' = m^*] - \max_{\text{ideal Lucy}} \Pr_{\text{ideal}} [m' = m^*] \]
Simplification-Based FPE: MR-insecurity

Warm-up example: attacking sparse formats

- $\mathcal{M} = \{m_1, ..., m_n\}$, $|m_i|$’s are unique
- Ciphertext reveals message length ($\Rightarrow$ reveals message)
  - In this case: $E_{K,T}^{T,N}(m) = m$
- The adversary $\mathcal{A}$:
  - Picks $\mathcal{D}$ = uniform distribution over $\mathcal{M}$
  - Given $c^*$, guesses $c^*$
  - Makes no queries!
  - $\Pr[\mathcal{A} \text{ wins}] = 1$
- Best ideal-world adversarial strategy: random guess
  - $\Pr[\text{ideal } \mathcal{A} \text{ wins}] = \frac{1}{|\mathcal{M}|}$
- Adversarial advantage: $1 - \frac{1}{|\mathcal{M}|} \rightarrow |\mathcal{M}| \rightarrow \infty 1$

General case: “sparsify” the format

- For every possible length, $\mathcal{A}$ selects a single message
- Picks uniforms distribution over these messages
Simplification-Based FPE: Take-Home Message

• “Natural” method of representing formats is insecure

• **Reason:** encryption preserves *message-specific* properties

    **FPE “wish list”**

• **Functionality (and efficiency):**
  – *Simple* method of representing formats
  – *Efficient* rank, unrank procedures
    • In particular: minimize cycle walking

• **Security:** preserve *only format-specific* properties
  – Hide *all message-specific* properties
RtE-Based FPE for General Formats

- Two concurrent works [LDJRS‘14,WRB‘15], differ in focus and design
- **Focus:** Developer- or user-oriented
- **Design:** representing formats, ranking methods
  - Both schemes based on RtE (Rank-then-Encipher)
  - **libFTE [LDJRS‘14]:**
    - Developer-oriented
    - Represent formats using regular expressions
    - Extend RtE method to allow efficient ranking
  - **GFPE [WRB‘15]:**
    - User-oriented
    - Represent formats using bottom-up framework
    - Use standard RtE
libFTE [LDJRS`14]

• Library for format-preserving and format transforming encryption

• Developer-oriented: developer needed to...
  – Choose “right” scheme to use (using “Configuration assistant”)
    • Several schemes (with different parameters) available
  – Define new formats

• **Structure:**
  – Represent formats with Regular Expressions (Regexes)
    • Expressions limited to lengths in range \( \{n_{\text{min}}, n_{\text{max}}\} \)
  – Ranking from automatons
  – Int-FPE using FFX-A2 (FFX over bit strings)

• **Main challenge:** efficient rank, unrank algorithms
Ranking in libFTE

• Format represented as regular expressions (regexes) ⇒ need a method of ranking regexes
  – **Ranking**: bijection from $\mathcal{M}$ to $\{1, \ldots, |\mathcal{M}|\}$
    • Only useful when bijection easy to compute
  – (Exact) ranking may be an overkill
  – Suffices to achieve a relaxed ranking notion
  – **Relaxed ranking** [LDJRS`14]:
    map $\mathcal{M}$ to $\{1, \ldots, M'\}$, $|\mathcal{M}| < M'$
    • $\text{r}rank: \mathcal{M} \rightarrow \{1, \ldots, M'\}$ injective
    • $\text{unr}rank: \{1, \ldots, M'\} \rightarrow \mathcal{M}$ surjective
    • For all $m \in \mathcal{M}$, $\text{unr}rank(\text{r}rank(m)) = m$
Ranking in libFTE (2)

- Format represented as regular expressions (regexes) ⇒ need a method of ranking regexes

(Highly Informal) Automata Theory Crash Course

- Automatons, and regexes, used to represent sets \( \mathcal{M} \)
- Automatons are graphs: \( m \in \mathcal{M} \) represented through paths in graph
  - **Deterministic** finite automaton (DFA)
  - **Nondeterministic** finite automaton (NFA)
- Regexes equivalent to automatons:
  - Regex-to-NFA transformation in **linear** time
  - Regex-to-DFA transformation in **exponential** time (this is tight!)

- “Meta” ranking technique from DFA [BRRS`09]
  - Order paths in DFA, map \( m \in \mathcal{M} \) to index of corresponding path
  - **Too inefficient!**
- Relaxed ranking from NFA in polynomial time [LDJRS`14]
  - In NFA, (possibly) more than one path for \( m \in \mathcal{M} \)
  - Find one such path efficiently through implicit graph representation
- libFTE supports DFA-based ranking and NFA-based relaxed ranking
libFTE: Tools and Algorithms

• Configuration assistant helps developer choose appropriate scheme
  – Randomized\deterministic, DFA-based\NFA-based ranking…

• “Appropriate” schemes chosen according to input parameters
  – Format, memory threshold for encryption\ranking…
  – Assistant runs tests to evaluate time and memory performance

• Developer chooses preferred scheme from list

```
$ ./configuration-assistant \
> --input-format "(a|b)*a(a|b){16}" 0 32 \

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

==== Evaluating valid schemes ====
SCHEME ENCRYPT DECRYPT ... MEMORY
P-ND  0.32ms 0.31ms ... 77KB
P-NN  0.39ms 0.38ms ... 79KB
...
```
libFTE: Tools and Algorithms

• Configuration assistant helps developer choose appropriate scheme
  – Randomized\deterministic, DFA-based\NFA-based ranking...

• “Appropriate” schemes chosen according to input parameters
  – Format, memory threshold for encryption\ranking...
  – Assistant runs tests to evaluate time and memory performance

• Developer chooses preferred scheme from list

```bash
$ ./configuration-assistant \
> --input-format "(a|b)*a(a|b){16}"

Format

min, max len

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

NFA-based ranking

DFA-based unranking

randomized

Evaluating valid schemes ====
SCHEME ENCRYPT DECRYPT ... MEMORY
P-ND  0.32ms  0.31ms ...  77KB
P-NN  0.39ms  0.38ms ...  79KB
...
libFTE: Implementation Notes

- Encryption\decryption performance (runtime and memory consumption) determined by:
  - Chosen scheme (DFA or NFA-based ranking)
  - Chosen representation of format (!)

- Unclear how to find scheme + format representation optimizing performance
  - Even given performance estimate of assistant
  - Bad performance due to bad regex or bad format?

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Input/Output Format</th>
<th>(R)</th>
<th>(\alpha)</th>
<th>(\beta)</th>
<th>DFA/NFA States</th>
<th>Memory Required</th>
<th>Encrypt (ms)</th>
<th>Decrypt (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-DD</td>
<td>((a</td>
<td>b)*)</td>
<td>0</td>
<td>32</td>
<td>2</td>
<td>2</td>
<td>4KB</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>((a</td>
<td>b)*{16}{a</td>
<td>b})</td>
<td>16</td>
<td>32</td>
<td>131,073</td>
<td>266MB</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>((a</td>
<td>b){16}{a</td>
<td>b})</td>
<td>16</td>
<td>32</td>
<td>18</td>
<td>36KB</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td>((a</td>
<td>b){1024})</td>
<td>1,024</td>
<td>1,024</td>
<td>1,026</td>
<td>34MB</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>P-NN</td>
<td>((a</td>
<td>b)*)</td>
<td>0</td>
<td>32</td>
<td>3</td>
<td>6KB</td>
<td>0.36</td>
<td>0.35</td>
</tr>
<tr>
<td></td>
<td>((a</td>
<td>b)*{16}{a</td>
<td>b})</td>
<td>16</td>
<td>32</td>
<td>36</td>
<td>73KB</td>
<td>0.61</td>
</tr>
<tr>
<td></td>
<td>((a</td>
<td>a</td>
<td>b){16}{a</td>
<td>b})</td>
<td>16</td>
<td>32</td>
<td>51</td>
<td>103KB</td>
</tr>
<tr>
<td></td>
<td>((a</td>
<td>b){1024})</td>
<td>1,024</td>
<td>1,024</td>
<td>2,049</td>
<td>68MB</td>
<td>6.6</td>
<td>6.6</td>
</tr>
</tbody>
</table>
libFTE: Implementation Notes

- Encryption/decryption performance (runtime and memory consumption) determined by:
  - Chosen scheme (DFA or NFA-based ranking)
  - Chosen representation of format (!)

- Unclear how to find scheme + format representation optimizing performance
  - Even given performance estimate of assistant
  - Bad performance due to bad regex or bad format?

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Input/Output Format</th>
<th>DFA/NFA States</th>
<th>Memory Required</th>
<th>Encrypt (ms)</th>
<th>Decrypt (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-DD</td>
<td>(a|b)*</td>
<td>0</td>
<td>2</td>
<td>4KB</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>(a|b)<em>a(a|b)</em>{16}</td>
<td>16</td>
<td>131,073</td>
<td>266MB</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>(a|a|b)<em>{16}(a|b)</em></td>
<td>16</td>
<td>18</td>
<td>36KB</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td>(a|b)*{1024}</td>
<td>1,024</td>
<td>1,026</td>
<td>34MB</td>
<td>1.2</td>
</tr>
<tr>
<td>P-NN</td>
<td>(a|b)*</td>
<td>0</td>
<td>3</td>
<td>6KB</td>
<td>0.36</td>
</tr>
<tr>
<td></td>
<td>(a|b)<em>a(a|b)</em>{16}</td>
<td>16</td>
<td>36</td>
<td>73KB</td>
<td>0.61</td>
</tr>
<tr>
<td></td>
<td>(a|a|b)<em>{16}(a|b)</em></td>
<td>16</td>
<td>51</td>
<td>103KB</td>
<td>1,340</td>
</tr>
<tr>
<td></td>
<td>(a|b)*{1024}</td>
<td>1,024</td>
<td>2,049</td>
<td>68MB</td>
<td>6.6</td>
</tr>
</tbody>
</table>
GFPE [WRB`15]

• User-oriented
  – Part of a larger system used by the end-user
• Encryption\decryption using RtE, supporting int-FPE for:
  – $\mathbb{Z}_M$ (proven security, no cycle walking, inefficient for large formats)
  – $\{0,1,...,m-1\}^n$ (no security proofs, requires cycles walking, efficient for large formats)

• **Main challenge:** user-friendly format representation

• **Structure:** formats represented using bottom-up framework
  – “Basic” building-blocks (primitives)
    • Usually “rigid” formats
      – SSNs, CCNs, dates, set of valid strings, fixed-length strings...
    • Also “less rigid” formats (e.g., variable-length strings)
  – **Operations** used to construct complex formats
    • Operations preserve the “parsing property”
GFPE: Representing Formats

• “Basic” building-blocks (primitives):
  – \( F_{\text{upper}} = \{\text{A,B,...,Z}\} \)
  – \( F_{\text{lower}} = \text{length-}k \text{ lower-case letter strings}, 1 \leq k \leq 15 \)
  – \( F_{\text{ssn}} = \text{SSNs} \)

• Operations:
  – **Concatenation:**
    • \( F = F_1 \cdot ... \cdot F_k \)
      – Words: \( F_{\text{word}} = F_{\text{upper}} \cdot F_{\text{lower}} \)
    • \( F = F_1 \cdot d_1 \cdot F_2 \cdot ... \cdot d_{n-1} \cdot F_n \) (\( d_1, ..., d_{n-1} \) are delimiters)
  – **Range:** \( F = (F_1 \cdot d)^k, \ min \leq k \leq max \)
    • Names: \( F_{\text{name}} = (F_{\text{word}} \cdot \text{space})^k \) for \( 1 \leq k \leq 4 \)
  – **Union:** \( F = F_1 \cup ... \cup F_k \)
    • “Names or SSNs”: \( F = F_{\text{name}} \cup F_{\text{ssn}} \)
Example: Representing Addresses

<table>
<thead>
<tr>
<th>name</th>
<th>house #</th>
<th>street</th>
<th>city</th>
<th>zip</th>
<th>state</th>
</tr>
</thead>
</table>

- $F_{name} = (F_{word} \cdot space)^k$ for $1 \leq k \leq 4$ (range)
- $F_{num} = \{1, \ldots, 100\}$ (integral domain)
- $F_{zip} = \{0,1, \ldots, 9\}^5$ (fixed length string)
- $F_{state} = \text{set of valid state abbreviations}$
- Valid addresses obtained through concatenation:
  \[ F_{add} = F_{name} \cdot F_{num} \cdot F_{name} \cdot F_{name} \cdot F_{zip} \cdot F_{state} \]
GFPE: Ranking

• Define ranking for primitives and operations
• Rank of compound formats computed top-down:
  – Parse string to components
  – Delegate substring ranking to format components
  – “Glue” ranks together using ranking for operations
Example: Ranking Concatenation

\[ F = F_1 \cdot d \cdot F_2 \]

\[ m = m_1 \cdot d \cdot m_2 \]
Example: Ranking Concatenation

$$\mathcal{F} = \mathcal{F}_1 \cdot d \cdot \mathcal{F}_2$$

Scale-and-Sum:

$$r = r_1 + r_2 \cdot \mathcal{F}_1 \cdot \text{size()}$$

Scale by size of sub-formats
GFPE: Supporting Large Formats

• Scheme supports int-FPEs for $\mathbb{Z}_M$ [BR`02,BRRS`09]

• Requires factoring $M \Rightarrow$ inefficient for large $M$’s!

• Supporting large formats: keep formats small
  – Divide large formats
  – Minimize security loss by “hiding” message-specific properties:
    • Division according to format structure
    • Maximizing sub-format size
  – $maxSize$ determined by user-defined performance constraints
Example: Dividing Address Format

<table>
<thead>
<tr>
<th>Name</th>
<th>house #</th>
<th>street</th>
<th>city</th>
<th>zip</th>
<th>state</th>
</tr>
</thead>
</table>

- Valid addresses obtained through concatenation:
  \[ F_{add} = F_{name} \cdot F_{num} \cdot F_{name} \cdot F_{name} \cdot F_{zip} \cdot F_{state} \]

- Jane Doe 23 Delaford New York 12345 NY
- Jane Doe 23 Delaford Berkeley 12345 CA
- Smaller maxSize \( \Rightarrow \) further division
  - E.g., \( F_{name} \) divided according to number of words in name
GFPE: Supporting Large Formats (2)

• Scheme supports int-FPEs for $\mathbb{Z}_M$ [BR`02,BRRS`09]
• requires factoring $M \Rightarrow$ inefficient for large $M$’s!
• **Supporting large formats:** keep formats small
  – Divide large formats
  – Minimize security loss by “hiding” *message-specific* properties:
    • Division according to *format structure* ← Main challenge!
    • Maximizing sub-format size
  – $maxSize$ determined by user-defined performance constraints
• Introduces complications in ranking and unranking
  – Generalize rank, unrank to lists of ranks
• Minimal security loss (according to experimental results)
# FPEs for General Formats: Summary

<table>
<thead>
<tr>
<th></th>
<th>libFTE [LDJRS’14]</th>
<th>GFPE [WRB’15]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Underlying int-FPE</strong></td>
<td>FFX (any FPE for ({0,1,\ldots, m-1}^n))</td>
<td>FFX or FE1 (any FPE for (\mathbb{Z}_M) or ({0,1,\ldots, m-1}^n))</td>
</tr>
<tr>
<td><strong>Designed for</strong></td>
<td>developers</td>
<td>end users</td>
</tr>
<tr>
<td><strong>Encryption type</strong></td>
<td>deterministic\ random</td>
<td>deterministic</td>
</tr>
<tr>
<td><strong>Format representation</strong></td>
<td>regular expressions</td>
<td>bottom-up framework</td>
</tr>
<tr>
<td><strong>Security guarantee</strong></td>
<td>same as underlying int-FPE</td>
<td>same as underlying int-FPE</td>
</tr>
<tr>
<td><strong>Encryption type</strong></td>
<td>FPE + format transforming</td>
<td>FPE</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td>depends on scheme and format representation</td>
<td>uniform</td>
</tr>
<tr>
<td><strong>Expressiveness</strong></td>
<td>not clear how to efficiently represent though computations</td>
<td>representation thorough computation is possible</td>
</tr>
<tr>
<td><strong>Open source?</strong></td>
<td>Yes: Python, C++, JavaScript</td>
<td>No</td>
</tr>
</tbody>
</table>
Format Preserving Encryption (FPE): Summary

- FPE preserves plaintext format under encryption
- Useful when adding encryption layer to existing schemes
- Int-FPEs based on generalized Feistel networks
  - Two constructions under NIST consideration for standardization
- Techniques for general-format FPE:
  - Cycle walking
  - Rank-then-Encipher (RtE)
- FPE for general formats constructed from int-FPE
  - Using RtE on top of int-FPE
  - Comparable security and performance
THANK YOU