Privacy-Preserving Email Forensics (PPEF)

Frederik Armknecht (University of Mannheim), Andreas Dewald (Friedrich-Alexander University)

Sven Kälber (Speaker, Friedrich-Alexander University)
Agenda

- Idea, motivation & contributions
- The big picture / overall scheme design
- Own implementation details
  - Protection mechanism
  - Extraction mechanism
- Cryptographic building blocks
- Practical implementation and evaluation
- Summary & conclusion
- Limitations & future work
Idea of PPEF

- Privacy protection of employees in (large-scale) digital forensic investigations
- Revealing of only case relevant information
  - Achieved through strong cryptographic standards
- Operation principle:
  1. Extraction of mailboxes
  2. Encryption of all emails by applying our introduced scheme
  3. Hand over of only encrypted mailboxes to third-party investigators
  4. Decryption of individual emails only possible on $t$ matching keywords
Motivation

- Private use of corporate e-mail accounts
  - Private e-mails typically contain private and very sensitive data
  - This information is often highly protected by local data protection laws
  - Typically case irrelevant information in private e-mails
- Today's approaches and tools are often limited to filtering, which is not enforced
  - Investigators might read private e-mails by accident or on purpose
- Problems of leaving the e-mails at the company's IT
  - Leaks search queries of investigators
  - Is costly and time consuming because of the high degree of interaction needed
  - Trust issues
Contribution

- Novel approach for privacy-preserving email forensics allowing for **non-interactive** threshold keyword search on **encrypted** emails
- Proof-of-concept implementation in Python and as a Autopsy v3 plug-in
- An evaluation of the practical applicability in terms of:
  - en- / decryption runtime performance
  - introduced storage overhead
  - brute-force / dictionary attack vulnerability
Encryption of email contains:
• The encrypted text
• A compact representation of the mapping

```
Encryption

@ → Encrypt → @

Encryption of email contains:
• The encrypted text
• A compact representation of the mapping

Keyword 1 → Mapping → Key Share 1
Keyword 2 → Mapping → Key Share 2
... → Mapping → ...
Keyword n → Mapping → Key Share n
```
Decryption success

Searches for...

Keyword' 1
Keyword' 2
...
Keyword' t

Mapping

Key Share' 1
Key Share' 2
...
Key Share' n

Decrypt

Secret Sharing
Decryption fail

@ → Decrypt → @

Searches for...

- Correct kw 1
- Correct kw 2
- ... Wrong kw t

Correct kw 1 → Key Share’ 1
Correct kw 2 → Key Share’ 2
... → ... Key Share’ n

Mapping

Secret Sharing

Sven Kälber - DFRWS USA 2015
Details

• Encryption of e-mails (protection mechanism):
  • Each e-mail plaintext $P$ is encrypted to a cyphertext $C$ with an individual secret key $k$.
  • $k$ gets split up in shares and might later be reconstructed
  • Support for blacklisting of commonly used words (e.g. “the“)
  • Support for whitelisting of investigation keywords (e.g. “fraud“)

• Decryption of e-mails (extraction mechanism):
  • Only possible when the e-mail in question contains at least $t$ keywords.
  • Investigator learns nothing about the secret key of other e-mails upon successfully decrypting one e-mail.
  • Investigator learns nothing about the content of the mail if $t-1$ or less keywords match the content of the e-mail.
1. Encryption function:
   • AES-128 in CBC mode used for the encryption of individual e-mails
   • Add characteristic padding $p$ as the first block to be decrypted (e.g. $[0, \ldots, 0]$)

2. Shamir‘s Secret Sharing
   • Used for splitting the secret key $k$ into shares
   • Details follow on the next slide

3. Mapping
   • Hash function: SHA-256 part of the mapping function
   • Further tweaks for efficiency reasons
Shamir‘s Secret Sharing

• Functionality:
  • Input: Two integers $t \leq n$, a secret $k$
  • Output: $n$ shares $k_{\downarrow 1}, \ldots, k_{\downarrow n}$

• Security
  • Given at least $t$ shares, one can reconstruct the secret
  • If less than $t$ shares are known, reconstruction not possible

• Realization
  • Polynomial interpolation of a polynomial of degree $t-1$
Working Principle

- Input: Two integers \( t \leq n \), Secret \( k \)
- Choose polynomial \( p(x) \) of degree \( t-1 \)
- Compute shares: \((x_{\downarrow 1}, y_{\downarrow 1}), \ldots, (x_{\downarrow n}, y_{\downarrow n})\) (here: \( n=12 \)) with \( y_{\downarrow i} = p(x_{\downarrow i}) \)
- Reconstruction from \( t \) shares:
  - Interpolate \( p(x) \)
  - Compute \( k = p(x_{\downarrow 0}) \)
The Mapping Function

Text

Keyword 1
Keyword 2
...
Keyword n

Mapping

256-bit Strings

Key Share 1
Key Share 2
...
Key Share n

SHA 256 Hashfunction

256-bit Strings

Hash value 1
Hash value 2
...
Hash value t

Function G
Function G

- **Task:** Map 256-bit hash values to 256-bit shares \((x_{\downarrow i}, y_{\downarrow i})\)

- **Approach:**
  - Interpret hash values as \((x_{\downarrow i}, z_{\downarrow i})\) (128-bit + 128-bit)
  - Use the values \(x_{\downarrow i}\) to compute shares \((x_{\downarrow i}, y_{\downarrow i}) = (x_{\downarrow i}, p(x_{\downarrow i}))\)
  - Find mapping \(g(x)\) such that \(g(x_{\downarrow i}) = y_{\downarrow i} \oplus z_{\downarrow i}\)
  - Function \(G(h_{\downarrow i}) = G(x_{\downarrow i}, z_{\downarrow i}) = (x_{\downarrow i}, g(x_{\downarrow i}) \oplus z_{\downarrow i}) \rightarrow (x_{\downarrow i}, y_{\downarrow i})\)

- **Getting the mapping:**
  - Core idea: compute polynomial \(g(x)\) such that \(g(x_{\downarrow i}) = y_{\downarrow i} \oplus z_{\downarrow i}\)
  - Problem: requires to interpolate polynomial of degree \(n\) \(\rightarrow\) effort is \(O(n^{\uparrow 3})\), too slow
  - Idea: Split range of \(x\) into \(l\) subsets, e.g. determined by the \(l\) last bits
  - Interpolate polynomials \(g_{\downarrow j}(x)\) for each subset
  - Effort: interpolate \(l\) polynomials, each of degree \(\approx n/l\)
  - Overall effort: \(l \cdot (n / l)^{\uparrow 3} = n^{\uparrow 3} / l^{\uparrow 2}\)
1. Python en- / decryption of mailboxes
   Supported mailbox formats:
   - mbox
   - pst
   - MH (RFC 822)
   - Maildir

2. PPEF plugin for Autopsy v3
PPEF Autopsy plug-in

Enter Data Source Information:

Select source type to add: Logical Files

Add local files and folders:
C:\Mailarchive.ppef

Please select the input timezone:

- Ignore orphan files in FAT file systems
  (faster results, although some data will not be searched)

Press 'Next' to analyze the input data, extract volume and file system data, and populate a local database.
PPEF Autopsy plug-in
PPEF Autopsy plug-in
PPEF Autopsy plug-in
PPEF Autopsy plug-in
The data set used in our evaluation consists of 5 different mailboxes:

- *Apache* – httpd user mailing list (75724 e-mails)
- *Work* – personal work e-mails (1590 e-mails)
- *A, B, C* – private e-mail accounts (511, 349, 83 e-mails)

**Evaluations:**

1. Encryption runtime performance
2. Encryption storage overhead
3. Search / decryption runtime performance
4. Brute-force attack performance
Encryption Performance

- Time (in seconds) to encrypt the corresponding emails of each account
- Average encryption rate: 13.5 emails/sec
  - Encryption of large mailboxes might take several hours (< 2h for 75724 e-mails), but only needs to be done once!

<table>
<thead>
<tr>
<th></th>
<th>Apache [s]</th>
<th>Work</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>0.004</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
</tr>
<tr>
<td>Max</td>
<td>31.745</td>
<td>1.403</td>
<td>1.932</td>
<td>1.117</td>
<td>0.460</td>
</tr>
<tr>
<td>Avg</td>
<td>0.082</td>
<td>0.136</td>
<td>0.122</td>
<td>0.110</td>
<td>0.173</td>
</tr>
<tr>
<td>Med</td>
<td>0.072</td>
<td>0.115</td>
<td>0.101</td>
<td>0.074</td>
<td>0.150</td>
</tr>
<tr>
<td>σ</td>
<td>0.133</td>
<td>0.120</td>
<td>0.132</td>
<td>0.153</td>
<td>0.071</td>
</tr>
<tr>
<td>Σ</td>
<td>6243.511</td>
<td>217.242</td>
<td>62.842</td>
<td>38.535</td>
<td>14.367</td>
</tr>
</tbody>
</table>
Encryption storage overhead

- Encryption with AES does not add much storage overhead (33 – 48 bytes per mail)
- Main storage overhead factor is the mapping function (on average 582.4 bytes per mail)
- Average storage overhead: 5.2 %

<table>
<thead>
<tr>
<th></th>
<th>Apache</th>
<th>Work</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size Raw [KB]</td>
<td>376,551</td>
<td>418,680</td>
<td>16,386</td>
<td>47,486</td>
<td>6,676</td>
</tr>
<tr>
<td>Size PPEF [KB]</td>
<td>418,870</td>
<td>420,418</td>
<td>16,885</td>
<td>47,821</td>
<td>6,806</td>
</tr>
<tr>
<td>Overhead</td>
<td>11.2 %</td>
<td>0.4 %</td>
<td>3.0 %</td>
<td>0.7 %</td>
<td>1.9 %</td>
</tr>
</tbody>
</table>
Search / decryption performance

- Time (in seconds) to search each e-mail for 3 keywords and decrypt matching e-mails
- Average search and decryption rate: 98 mails/sec
  - Searches on large mailboxes take time (< 15min) but are still feasible

<table>
<thead>
<tr>
<th></th>
<th>Apache [s]</th>
<th>Work</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>0.0090</td>
<td>0.0096</td>
<td>0.0098</td>
<td>0.0097</td>
<td>0.0098</td>
</tr>
<tr>
<td>Max</td>
<td>0.0598</td>
<td>0.1645</td>
<td>0.0139</td>
<td>0.1508</td>
<td>0.0148</td>
</tr>
<tr>
<td>Avg</td>
<td>0.0115</td>
<td>0.0137</td>
<td>0.0114</td>
<td>0.0123</td>
<td>0.0117</td>
</tr>
<tr>
<td>Med</td>
<td>0.0115</td>
<td>0.0117</td>
<td>0.0113</td>
<td>0.0113</td>
<td>0.0116</td>
</tr>
<tr>
<td>σ</td>
<td>0.0007</td>
<td>0.0103</td>
<td>0.0007</td>
<td>0.0086</td>
<td>0.0009</td>
</tr>
<tr>
<td>Σ</td>
<td>876.8591</td>
<td>21.7977</td>
<td>5.8650</td>
<td>4.2982</td>
<td>0.9750</td>
</tr>
</tbody>
</table>
Brute-force attacks to decrypt the whole mailbox ($\pi=0.99$) or a random half of the mailbox ($\pi=0.5$)

Using 4 different vocabularies

- Oxford English Dictionary (171,476 words)
- 50 % of the Oxford English Dictionary (85,738 words)
- Vocabulary in daily speech edu. person (20,000 words)
- Vocabulary of uneducated person (10,000 words)

<table>
<thead>
<tr>
<th>$\pi$</th>
<th>$N$</th>
<th>Apache</th>
<th>Work</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.99</td>
<td>171,476</td>
<td>$1.15 \cdot 10^8$</td>
<td>$3.26 \cdot 10^5$</td>
<td>$1.23 \cdot 10^5$</td>
<td>$1.17 \cdot 10^5$</td>
<td>5,373.15</td>
</tr>
<tr>
<td>0.50</td>
<td>171,476</td>
<td>$1.73 \cdot 10^7$</td>
<td>49,072.84</td>
<td>18,565.34</td>
<td>17,638.94</td>
<td>808.74</td>
</tr>
<tr>
<td>0.99</td>
<td>85,738</td>
<td>$1.44 \cdot 10^7$</td>
<td>40,753.36</td>
<td>15,418.00</td>
<td>14,648.51</td>
<td>671.63</td>
</tr>
<tr>
<td>0.50</td>
<td>85,738</td>
<td>$2.17 \cdot 10^6$</td>
<td>6,133.99</td>
<td>2,320.64</td>
<td>2,204.82</td>
<td>101.09</td>
</tr>
<tr>
<td>0.99</td>
<td>20,000</td>
<td>$1.83 \cdot 10^5$</td>
<td>517.23</td>
<td>195.68</td>
<td>185.91</td>
<td>8.52</td>
</tr>
<tr>
<td>0.50</td>
<td>20,000</td>
<td>27,510.37</td>
<td>77.85</td>
<td>29.45</td>
<td>27.98</td>
<td>1.28</td>
</tr>
<tr>
<td>0.99</td>
<td>10,000</td>
<td>22,843.44</td>
<td>64.64</td>
<td>24.46</td>
<td>23.24</td>
<td>1.07</td>
</tr>
<tr>
<td>0.50</td>
<td>10,000</td>
<td>3,438.28</td>
<td>9.73</td>
<td>3.68</td>
<td>3.50</td>
<td>0.16</td>
</tr>
</tbody>
</table>
Summary / Conclusion

- We proposed a novel approach for privacy-preserving email forensics allowing for non-interactive threshold keyword search on encrypted e-mails.
- We developed a prototype-implementation in Python and an Autopsy plug-in that supports multiple well-known mailbox formats.
- We evaluated the practical applicability in terms of en-/ decryption performance, storage overhead and brute-force vulnerability.
  - Sufficiently large mailboxes are well protected against dictionary (brute-force) attacks.
Limitations:

- Scheme based on keyword searches, therefore prone to spelling errors
- No wildcard operator or regular expression possible that allows for more advanced search queries
- Brute-force / dictionary attacks possible

Future work:

- Support for wildcard usage within the search keywords
Thank you for your attention!

Speaker:
Sven Kälber - sven.kaelber@cs.fau.de

Authors:
Frederik Armknecht – armknecht@uni-mannheim.de
Andreas Dewald – andreas.dewald@fau.de
Function G

• Task: Map 256-bit hash values to 256-bit key shares

• Approach
  • Interpret hash values as \((128 \text{ bit} + 128 \text{ bit})\)
  • Use the values to compute key shares
  • Find a mapping such that
  • Function

• Getting the Mapping
  • Core idea: compute polynomial such that
  • Problem: requires to interpolate polynomial of degree \(n\) effort is in \(2^{128}\) too slow
  • Idea: Split range of into subsets, e.g., determined by the first bits
  • Interpolate polynomials for each subset
  • Effort: interpolate polynomials, each of degree
  • Overall effort: