Reducing Metadata Leakage from Encrypted Files and Communication with PURBs

Kirill Nikitin*, Ludovic Barman*, Wouter Lueks, Matthew Underwood, Jean-Pierre Hubaux, and Bryan Ford

École polytechnique fédérale de Lausanne (EPFL)

*Shared first authorship
[Dog video]
Ciphertexts Expose Metadata in Clear

To whom
Message size
Algorithms used
...

Ciphertext
Metadata
Encrypted Payload
OpenPGP Packet Format

Packet Type | Format version | Recipient Key ID | Public-Key Algorithm
--- | --- | --- | ---
85 02 0c 01 | 9497 608d d051 8f79 | 01 0f ff 46 bd7f 1821 27a9 42c4 01b4 7ec4 433e 7f90 |
1 | 74b8 139c a802 6678 ba0d 1abd |

Session Key Part

Data Part

Credit for the picture of the attacker here and graphics afterwards is to Vecteezy.com
OpenPGP Packet Format

Packet Type | Format version | Recipient Key ID | Public-Key Algorithm
---|---|---|---
8502 0c01 9497 608d d051 8f79 0f0f ff46
bd7f 1821 27a9 42c4 01b4 7ecd 433e 7f90
... ... 74b8 139c a802 6678 ba0d 1abd

d2 64 014b 6a5a f586 e3fa b98e 92d1 6759
7186 2ccc ac50 3db7 fa03 4f31 dcd7 fa40
... ... 4b09 d9b4 1654 972d 5c22 47db

A message to the King of Sweden encrypted with RSA-512 using an outdated OpenPGP format??

Small key? Outdated format? I might crack it!

Credit for the picture of the attacker here and graphics afterwards is to Vecteezy.com
OpenPGP Packet Format

<table>
<thead>
<tr>
<th>Packet Type</th>
<th>Format version</th>
<th>Recipient Key ID</th>
<th>Public-Key Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>85 0c 01</td>
<td>9497 608d d051 8f79</td>
<td>01 0f ff46</td>
<td>bd7f 1821 27a9 42c4 01b4 7ecd 433e 7f90</td>
</tr>
<tr>
<td></td>
<td>74b8 139c a802 6678 ba0d 1abd</td>
<td>d2 64 014b 6a5a f586 e3fa b98e 92d1 6759</td>
<td>7186 2ccc ac50 3db7 fa03 4f31 dcd7 fa40</td>
</tr>
<tr>
<td></td>
<td>4b09 d9b4 1654 972d 5c22 47db</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A message to the King of Sweden encrypted with RSA-512 using an outdated OpenPGP format??

Small key? Outdated format? I might crack it!

Credit for the picture of the attacker here and graphics afterwards is to Vecteezy.com
What If We Stripped Off All the Metadata?

“Black Square”, 1915, by Kazimir Malevich
It Is Possible But Challenging

1. Efficient decoding
2. When addressing multiple recipients
3. Using different cryptographic algorithms
Padded Uniform Random Blobs (PURBs)

- A novel format for encrypted data without any metadata in clear.
- The properties (informally):
  - Indistinguishability from random bits
  - Minimized length leakage
  - Content and metadata protection
Padded Uniform Random Blobs (PURBs)

- Two core components
  - Encoding scheme (*Multi-Suite* PURB or MsPURB)
  - Padding scheme (Padmé)
Encoding scheme (MsPURB)
Roadmap to MsPURB

- Single Recipient
- Multiple Recipients
- Multiple Suites
- Non-malleability
Single Recipient: Model

Honest Sender → PURB → Honest Recipient

Insecure channel

Is it a PURB or a random bit string?!

Active Adversary
Single Recipient

Recipient – public key $G_y$

Similar to the Integrated Encryption Scheme (IES) [ABR01]
Single Recipient

Recipient – public key $G^y$

Similar to the Integrated Encryption Scheme (IES) [ABR01]

Sender:
1. Generates an ephemeral key pair $x$, $G^x$;
Single Recipient

Recipient – public key $G^y$

Similar to the Integrated Encryption Scheme (IES) [ABR01]

Sender:
1. Generates an ephemeral key pair $x, G^x$;
2. Computes a shared secret $G^{yx}$;
Single Recipient

Recipient – public key $G^y$

Similar to the Integrated Encryption Scheme (IES) [ABR01]

Sender:
1. Generates an ephemeral key pair $x, G^x$;
2. Computes a shared secret $G^{yx}$;
3. Encrypts the data with one-time session key $K$;
Single Recipient

Recipient – public key $G^y$

Similar to the Integrated Encryption Scheme (IES) [ABR01]

Sender:
1. Generates an ephemeral key pair $x$, $G^x$;
2. Computes a shared secret $G^{yx}$;
3. Encrypts the data with one-time session key $K$;
4. Creates an entry point with $K$ and other metadata, encrypted with $G^{yx}$;

<table>
<thead>
<tr>
<th>Entry point</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>$AE_{G^y}(K | meta)$</td>
<td>$Enc_K(data)$</td>
</tr>
</tbody>
</table>
Reducing Metadata Leakage with PURBs @ PETS 2019

Single Recipient

Recipient – public key $G_y$

Similar to the Integrated Encryption Scheme (IES) [ABR01]

Sender:
1. Generates an ephemeral key pair $x, G^x$;
2. Computes a shared secret $G^{yx}$;
3. Encrypts the data with one-time session key $K$;
4. Creates an entry point with $K$ and other metadata, encrypted with $G^y$;
5. Encodes $G^x$ as a uniform bit string, e.g., with Elligator [BHKL13].
Multiple Recipients
Multiple Recipients

Recipients – public keys $G^y_1$, $G^y_2$, $G^y_3$.

We create an entry point per recipient, each with $K$ and metadata but encrypted with $G^y_1x$, $G^y_2x$, $G^y_3x$ respectively.

- $AE_{G^y_1x}(K||meta)$
- $AE_{G^y_2x}(K||meta)$
- $AE_{G^y_3x}(K||meta)$
Multiple Recipients

Recipients – public keys $G^y_1$, $G^y_2$, $G^y_3$.

We create an entry point per recipient, each with $K$ and metadata but encrypted with $G^{y_1x}$, $G^{y_2x}$, $G^{y_3x}$ respectively.

But how do we organize these entry points in the PURB?
Linear Approach Strawman

Recipients – public keys $G^y_1$, $G^y_2$, $G^y_3$.

We create an entry point per recipient, each with $K$ and metadata but encrypted with $G^{y_1x}$, $G^{y_2x}$, $G^{y_3x}$ respectively.
Linear Approach Strawman

Recipients – public keys $G^y_1$, $G^y_2$, $G^y_3$.

We create an entry point per recipient, each with $K$ and metadata but encrypted with $G^y_1x$, $G^y_2x$, $G^y_3x$ respectively.
Linear Approach Strawman

Recipients – public keys $G^y_1, G^y_2, G^y_3$.

$$\begin{align*}
\text{Inefficient to decode} \\
\text{Hide}(G^x) & \quad \text{AE}_{G^y_1}(K||\text{meta}) & \quad \text{AE}_{G^y_2}(K||\text{meta}) & \quad \text{AE}_{G^y_3}(K||\text{meta}) & \quad \text{Enc}_K(\text{data})
\end{align*}$$

We create an entry point per recipient, each with $K$ and metadata but encrypted with $G^y_1, G^y_2, G^y_3$ respectively.
Single Hash-Table Strawman

Recipients – public keys $G^y_1$, $G^y_2$, $G^y_3$.

Entry points are placed in a hash table, indexed by $G^y_x$. 

$\text{Hide}(G^x)$

$\text{Hash Table}$

$\text{Enc}_K(\text{data})$
Single Hash-Table Strawman

Recipients – public keys $G^y_1$, $G^y_2$, $G^y_3$.

Entry points are placed in a hash table, indexed by $G^y_x$.
Single Hash-Table Strawman

Recipients – public keys $G^{y_1}, G^{y_2}, G^{y_3}$.

Entry points are placed in a hash table, indexed by $G^{yx}$.
Single Hash-Table Strawman

Recipients – public keys $G^y_1, G^y_2, G^y_3$.

Entry points are placed in a hash table, indexed by $G^y_x$.

1. Space waste
2. Bound on $N$ of recipients
Multiple Recipients: Our Solution

Recipients – public keys $G^y_1$, $G^y_2$, $G^y_3$.

Entry points are placed in a series of growing hash-tables!
Multiple Recipients: Our Solution

Recipients – public keys $G^y_1$, $G^y_2$, $G^y_3$.

$\text{HT0}$

$\text{Hide}(G^x) \quad \text{AE}_{G^y_1}(K||\text{meta}) \quad \text{Enc}_K(\text{data})$

Entry points are placed in a series of growing hash-tables!
Multiple Recipients: Our Solution

Recipients – public keys $G^{y_1}$, $G^{y_2}$, $G^{y_3}$.

Hide($G^x$) $\rightarrow$ $\text{AE}_{G^{y_1}}(K||\text{meta})$ $\rightarrow$ $\text{AE}_{G^{y_2}}(K||\text{meta})$ $\rightarrow$ $\text{Enc}_K(\text{data})$

Entry points are placed in a series of growing hash-tables!
Multiple Recipients: Our Solution

Recipients – public keys $G^y_1$, $G^y_2$, $G^y_3$.

- $\text{Hide}(G^x)$
- $\text{AE}_{G^y_1 x}(K||\text{meta})$
- $\text{AE}_{G^y_2 x}(K||\text{meta})$
- $\text{AE}_{G^y_3 x}(K||\text{meta})$
- $\text{Enc}_K(\text{data})$

Entry points are placed in a series of growing hash-tables!
Multiple Recipients: Our Solution

Recipients – public keys $G^y_1$, $G^y_2$, $G^y_3$.

Entry points are placed in a series of growing hash-tables!
Multiple Recipients: Our Solution

Recipients – public keys $G^y_1$, $G^y_2$, $G^y_3$.

Entry points are placed in a series of growing hash-tables!
Multiple Suites

- PURB -

Single Recipient

Multiple Suites

Non-malleability

Multiple Recipients

Multiple Suites
Multiple Suites

- Recipients use several distinct suites, based on public-key group (e.g., Curve25519 or Curve448) or entry point length.

- Each suite (an encoded public key and hash tables) becomes a distinct logical layer in a PURB, and these layers overlap!
Multiple Suites: Layout

Suite A

Suite B

PURB bytes
Multiple Suites: Layout

Suite A

Hide(A)  

Enc(data)

PURB bytes

Suite B

Hide(B)
Reducing Metadata Leakage with PURBs @ PETS 2019

Multiple Suites: Layout

Suite A

PURB bytes

Hide(A)

Enc(data)

Suite B

Hide(B)

Non-malleability

Single Recipient

Multiple Recipients

Multiple Suites

Kirill Nikitin
Multiple Suites: Layout

- **Suite A**
  - `Hide(A)`
  - `Enc(data)`

- **Suite B**
  - `Hide(B)`

**PURB bytes**

---

Kirill Nikitin
Multiple Suites: Layout

Suite A

Hide(A) → \text{AE}_G^{y_1x}(K) → \text{AE}_G^{y_1x}(K) → \text{AE}_G^{y_1x}(K) → \text{Enc(data)}

Suite B

Hide(B)
Multiple Suites: Layout

<table>
<thead>
<tr>
<th>Suite A</th>
<th>Suite B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hide(A)</td>
<td>Hide(B)</td>
</tr>
<tr>
<td>$AE_G^{ylx}(K)$</td>
<td>$AE_G^{ylx}(K)$</td>
</tr>
<tr>
<td>$AE_G^{ylx}(K)$</td>
<td>$AE_G^{ylx}(K)$</td>
</tr>
</tbody>
</table>

**PURB bytes**

- **Single Recipient**
- **Multiple Recipients**
- **Multiple Suites**
- **Non-malleability**
Multiple Suites: Layout

Suite A

Hide(A)  \(\text{Enc(data)}\)

\[\text{AE}_G^y(k)\]

\[\text{AE}_G^y(k)\]

\[\text{AE}_G^y(k)\]

Suite B

Hide(B)

\[\text{Hide(A)}\]

\[\text{AE}_G^y(k)\]

\[\text{AE}_G^y(k)\]

PURB bytes

Non-malleability

Single Recipient  Multiple Recipients  Multiple Suites  Non-malleability
Multiple Suites: Layout

**Suite A**

- **Hide(A)**
- **$AE_G^{y_1x}(K)$**
- **$AE_G^{y_1x}(K)$**
- **$AE_G^{y_1x}(K)$**

**PURB bytes**

**Suite B**

- **Hide(B)**
- **$AE_Q^{y_4h}(K)$**

**Enc(data)**

**random**

---

Hide(A) and Hide(B) are protocols for hiding information. The **$AE_G^{y_1x}(K)$** and **$AE_Q^{y_4h}(K)$** represent encryption functions. Enc(data) denotes the encryption of data. The figure illustrates how metadata leakage can be reduced using PURBs (Packed Unpackable Recipients) in different suites.
Multiple Suites: Layout

Suite A

- Hide(A)
- $\text{AE}_G^{y_{1x}(K)}$
- $\text{AE}_G^{y_{1x}(K)}$
- $\text{AE}_G^{y_{1x}(K)}$

- $\text{Enc(data)}$

Suite B

- Hide(B)
- $\text{AE}_Q^{y_{4h}(K)}$

Random

- Single Recipient
- Multiple Recipients
- Multiple Suites
- Non-malleability
Non-malleability

Suite A

Hide(A)  \[ AE_{Q^y_1x}(K) \]
\[ AE_{Q^y_1x}(K) \]
\[ AE_{Q^y_1x}(K) \]

Enc(data)

PURB bytes

Suite B

Hide(B)  \[ AE_{Q^{y_4h}}(K) \]

random

MAC

Single Recipient  Multiple Recipients  Multiple Suites  Non-malleability
Non-malleability

**Suite A**
- **Hide(A)**
- $\text{AE}_G^y\text{lx}(K)$
- $\text{AE}_G^y\text{lx}(K)$
- $\text{AE}_G^y\text{lx}(K)$
- **Enc(data)**

**PURB bytes**

**Suite B**
- **Hide(B)**
- random
- $\text{AE}_Q^y\text{hx}(K)$

**IND$^+$-CCA2**
Finding Public Keys Efficiently

Suite A

Possible position I
Possible position II
Possible position III

PURB bytes

Hide(A)

See the paper for the details
Encoding and Decoding of PURBs

![Graph showing CPU time vs. number of recipients with different suites](image)

- EncHeader
- KeyGen
- SharedSecrets
- Total time

Number of Recipients: 1, 3, 10, 100

CPU time [ms]

- EncHeader
- KeyGen
- SharedSecrets
- Total time

- 1 suite
- 3 suites
- 10 suites

Kirill Nikitin
Encoding and Decoding of PURBs

![Graph showing CPU time vs. number of recipients for different operations and numbers of suites.]
Padmé: reducing leakage about the size
Padmé

- The total size is an **important metadata**, used in many attacks:
  - Website Fingerprinting
  - Traffic-Analysis
  - Attacks against HTTPS

- Design a padding function to improve “size privacy”
Naïve approach: (constant) block-padding

e.g. 150B => 256B
Naïve approach: (constant) block-padding

Problem: no good value for block size

Example: $b = 1 \text{ MB}$
Naïve approach: (constant) block-padding

Problem: no good value for block size

Example: $b = 1 \text{ MB}$
Padding relative to the object size

Variable block size:

small objects: small overhead

Diagram showing variable block size with small objects having small overhead.
Padding relative to the object size

Variable block size:

large objects: larger privacy
Padding relative to the object size

Variable block size:

Padding to the "Next Power of 2"
Quantifying leakage of a padding function

- Let \( f: \mathbb{N} \rightarrow \mathbb{N} \) be the padding function. Let \( C \) be the image set of \( f \).

\[
\text{Leakage [bits]} = \log_2(|C|)
\]

Leakage: \( \log_2(1) = 0 \text{ bit} \)

Leakage: \( \log_2(2) = 1 \text{ bit} \)
Padding to the nearest power of 2

- Leakage: $O(\log \log(M))$, where $M$ is the biggest plaintext possible size of the image set

- Much better than with constant block-size, which leaks $O(\log(M))$

- Interestingly, not padding at all also leaks $O(\log(M))$

- Max overhead: +100%
  - e.g., 16.1 GB => 32 GB
Reducing the overhead of padding

Next power of 2: Blocks have the form $2^0$, $2^1$, $2^2$, etc. Represent them like floating points:

- Exponent
- Mantissa

\[ \log_2(M) \text{ bits} \quad \text{all 0's} \]

Padmé: Instead of 0's, allow some values in the mantissa:

=> Smaller blocks => Smaller overhead

46
Padmé

Pad to the next length $L$ which respects:

- Exponent: $\log_2(L)$ bits
- Mantissa: $\leq \log_2(\log_2(L))$ bits

Intuition: the exponent can be anything, but the mantissa cannot be "too precise"

Doubles leakage $\Rightarrow$ still in $O(\log \log(M))$
Padmé's overhead

\[
\text{max overhead} = \frac{1}{2 \log_2 L} \%
\]

Slowly decreases with L

Max 12% ∀ L

Max ~6% for L ≥ ~1 MB

Max ~3% for L ≥ ~1 GB
Padmé's "size privacy"

57,000 objects collected from apt lists

Mean overhead:

- Next power of 2: +44%
- Padmé: +2.3%
Conclusion

- Padded Uniform Random Blobs (PURBs): ciphertext format with no metadata leakage except length, which is minimized.

- Encoding + Padding schemes.

- Applications: Email, Group Chat, Disk Encryption, Initiation of Protocols.

- To the best of our knowledge, the first video with pets @ PETS.

https://purbs.net

https://github.com/dedis/purb

kirill.nikitin@epfl.ch  @ni_kirill

ludovic.barman@epfl.ch  @lbarman_ch