A look at the PGP keyserver data

Hanno Böck
The PGP Ecosystem

- The OpenPGP standard (RFC 4880)
- Software packages (Original PGP, GnuPG, endtoend, ...)
- Key servers (today mostly sks)
- When I say PGP I mean the ”PGP ecosystem” (software, standards etc.), not the PGP software product itself
Should we care?

- 'Why is GPG ”damn near unusable”?’ (31C3)
- "In the 1990s, I was excited about the future, and I dreamed of a world where everyone would install GPG. Now I’m still excited about the future, but I dream of a world where I can uninstall it.” (Moxie Marlinspike)
- ”Please throw some money to the GPG guy. Even though PGP sucks, it’s the best we’ve got.” (Matthew Green)
PGP problems

- Crypto is outdated, some of that is not fixable within the current model (Forward secrecy)
- PGP is and has always been ”damn near unusable”
- Lots of backwards compatibility cruft, complex format, limited software options (no library)
- No subject encryption
- Two competing mail formats (PGP/MIME and PGP/Inline) each with its own advantages and disadvantages
- The trust model (web-of-trust, key signing) is incomprehensible for everyone outside the geek cosmos
Google and Yahoo work on PGP-based solutions (endtoend)

Nothing currently seeks to replace it in the E-Mail space

Systems like Textsecure and Pond are technically superior, but they’re not built to replace E-Mail
I hate PGP, but I still try to make it better

- Fuzzing GnuPG found various vulnerabilities (CVE-2014-9087, CVE-2015-1606, CVE-2015-1607)
- Made proposal for subject encryption (a variant of it developed by Daniel Kahn Gillmor may land in Enigmail)
- I looked at the keyserver data to find crypto attacks (this talk)
The Idea

- PGP key servers store all keys ever sent to them on an add only basis
- You can’t delete keys from key servers, you can just revoke them
- This leads to all kinds of potential problems (key servers can be flooded with bogus data, privacy issues, ...)
- Crypto researchers perspective: Great, lots of data to investigate.
Inspiration

- EFF SSL Observatory (2010)
- Mining Your Ps and Qs (Nadia Heninger et al, 2012)
Look at keyserver data

- Large scale analysis of Internet wide scans for TLS certificate found crypto vulnerabilities
- For PGP we don’t have to scan the Internet - we can get the data from the keyservers
- Let’s put the crypto values in a database and analyze it
Parser challenges

- Lack of software: There is no low-level library to parse PGP key data
- pgpdump: Command line tool, doesn’t give us all the data we want
- I wrote my own parser in python (warning: I’m not a good coder, the code looks horrible, but it works)
- keyr (abbr for key parser) will take keyserver data and output MySQL statements
Database challenges

- Large database (84 GB), careful adjustments of parameters (e.g. indexes)
- Used MyISAM, MySQL 5.6 and tcmalloc (improved memory allocator from Google)
- Increased values for max_allowed_packet, key_buffer_size, wait_timeout, interactive_timeout
- (Warning: My MySQL knowledge is limited)
How does it work?

- Download keyserver dump, unpack if necessary
- Create database and tables from keyr-tables.sql
- Run keyr on keyserver dump files, pipe output to MySQL
### How does it look like?

<table>
<thead>
<tr>
<th>id</th>
<th>keyid_short</th>
<th>rsa_e</th>
<th>rsa_bits</th>
<th>ver</th>
<th>sub</th>
<th>errors</th>
<th>file</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3b6c249e</td>
<td>4096</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td>sks-dump-0000.pgp</td>
</tr>
<tr>
<td>2</td>
<td>3b6c249e</td>
<td>4096</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td>sks-dump-0000.pgp</td>
</tr>
<tr>
<td>3</td>
<td>f9a3f1dd</td>
<td>1024</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td>sks-dump-0000.pgp</td>
</tr>
<tr>
<td>4</td>
<td>cce5c3caa9995bde7</td>
<td>2048</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td>sks-dump-0000.pgp</td>
</tr>
<tr>
<td>5</td>
<td>cce5c3caa9995bde7</td>
<td>2048</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td>sks-dump-0000.pgp</td>
</tr>
<tr>
<td>6</td>
<td>bdb86311ea5de89</td>
<td>11</td>
<td>1024</td>
<td>3</td>
<td></td>
<td>0</td>
<td>sks-dump-0000.pgp</td>
</tr>
<tr>
<td>7</td>
<td>bdb86311ea5de89</td>
<td>11</td>
<td>2048</td>
<td>4</td>
<td></td>
<td></td>
<td>sks-dump-0000.pgp</td>
</tr>
<tr>
<td>8</td>
<td>d0f0188a5da5ea5e3c9</td>
<td>11</td>
<td>1024</td>
<td>2</td>
<td></td>
<td></td>
<td>sks-dump-0000.pgp</td>
</tr>
<tr>
<td>9</td>
<td>d0f0188a5da5ea5e3c9</td>
<td>11</td>
<td>2048</td>
<td>3</td>
<td></td>
<td></td>
<td>sks-dump-0000.pgp</td>
</tr>
<tr>
<td>10</td>
<td>6ebee426f9061ab</td>
<td>11</td>
<td>2048</td>
<td>3</td>
<td></td>
<td></td>
<td>sks-dump-0000.pgp</td>
</tr>
<tr>
<td>11</td>
<td>34455afc616e901</td>
<td>11</td>
<td>2048</td>
<td>3</td>
<td></td>
<td>0</td>
<td>sks-dump-0000.pgp</td>
</tr>
<tr>
<td>12</td>
<td>33a321e3343428ae5f</td>
<td>11</td>
<td>2048</td>
<td>4</td>
<td></td>
<td>0</td>
<td>sks-dump-0000.pgp</td>
</tr>
<tr>
<td>13</td>
<td>a7d48cd9ba57e19c</td>
<td>11</td>
<td>2048</td>
<td>4</td>
<td></td>
<td>0</td>
<td>sks-dump-0000.pgp</td>
</tr>
<tr>
<td>14</td>
<td>a7d48cd9ba57e19c</td>
<td>11</td>
<td>2048</td>
<td>4</td>
<td></td>
<td>1</td>
<td>sks-dump-0000.pgp</td>
</tr>
<tr>
<td>15</td>
<td>54119a80a8e5fe8e</td>
<td>11</td>
<td>2048</td>
<td>4</td>
<td></td>
<td>0</td>
<td>sks-dump-0000.pgp</td>
</tr>
<tr>
<td>16</td>
<td>2d5c76a9e2d1897</td>
<td>11</td>
<td>2048</td>
<td>4</td>
<td></td>
<td>1</td>
<td>sks-dump-0000.pgp</td>
</tr>
</tbody>
</table>
What data?

- Keys and signatures split into their hex encoded crypto values
- Hashes for signatures
- Remember: Crypto keys and signatures are just numbers
- Ignored: User ID strings etc. - everything that’s not crypto/math
Attack idea: RSA

- RSA public key: Modulus N (product of primes p, q) and exponent e
- If we know p and q we can break the key
- If due to a bad random number generator two RSA keys share one factor of N (p*q1, p*q2) we can efficiently break the keys by calculating the greatest common divisor (GCD)
- Same attack as Heninger et al and Lenstra et al (2012)
Batch GCD

- We can replicate the attack with the code from Nadia Heninger, but no new insights
- Leads to two valid looking breakable keys, reason unknown
- Various obviously broken keys (small factors, no user ids etc.)
  - the key servers are full of invalid data, likely due to data transmission errors
**DSA is common**

- GnuPG by default created primary DSA keys with 1024 bit for a long time
- 1024 bit is considered bad, it can be broken by attackers with a large budget
- I don’t have millions of euros and no degree in advanced number theroy
- But: DSA has a weakness when it comes to random numbers
DSA duplicate k

- When creating a DSA signature one has to create a temporary, random and unique value $k$
- If two signatures were created with the same $k$ it leads to the same $r$, so we can easily find these signatures
- If due to bad random numbers we have two different signatures with a shared $k/r$ value we can break the private key
- This is a real problem: Attack on Playstation 3 and Bitcoin stealing
Lots of DSA keys and signatures

- We have lots of DSA keys and signatures - if there ever was a PGP DSA implementation with a flawed random number generator we will probably find it.
- A look at the code of original PGP and GnuPG shows that the developers knew of this problem and did a lot of things to prevent it from happening.
Let MySQL do the work:

```
SELECT a.keyid, a.dsa_r, a.dsa_s, b.dsa_s, a.hash, b.hash, c.dsa_p, c.dsa_q, c.dsa_g, c.dsa_y FROM sigs_dsa a JOIN sigs_dsa b JOIN keys_dsa c ON a.dsa_r = b.dsa_r AND a.dsa_s = b.dsa_s AND a.keyid = c.keyid GROUP BY a.dsa_r;
```

We get around 350 duplicates, but most don’t lead to working keys - again lots of invalid data

- One key fails
- Checking ECDSA (same vuln) gives no results
The broken key

- The key belongs to a developer of the company PrimeFactors
- Answer from PrimeFactors: Test keys created during development.
- "our shipping product versions use the Blum-Blum-Shub generator which does not suffer from the problem you mention."
- This doesn’t completely make sense.
- Request for NDA prevented further analysis.
What could be done next?

- We only checked the signatures on the key servers - mailing list archives could be scanned for DSA signatures on mails (non-trivial)
- Other crypto attacks that work on large scale data sets? Ideas welcome.
Thanks! Questions?

Code and background paper will be released:
https://github.com/hannob/pgpecosystem/