Attack on Private Signature Keys of the OpenPGP format, PGP™ programs and other applications compatible with OpenPGP

Vlastimil Klíma [1] and Tomáš Rosa [2]

[1] Decros spol. s r.o., a member of the group ICZ a.s., Prague, v.klima@decros.cz
[2] Decros spol. s r.o., a member of the group ICZ a.s., Prague and CTU - FEE Prague, t.rosa@decros.cz

Abstract. The article describes an attack on OpenPGP format, which leads to disclosure of the private signature keys of the DSA and RSA algorithms. The OpenPGP format is used in a number of applications including PGP, GNU Privacy Guard and other programs specified on the list of products compatible with OpenPGP, which is available at http://www.pgpi.org/products. Therefore all these applications must undergo the same revision as the actual program PGP™. The success of the attack was practically verified and demonstrated on the PGP™ program, version 7.0.3 with a combination of AES and DH/DSS algorithms. As the private signature key is the basic information of the whole system which is kept secret, it is encrypted using the strong cipher. However, it shows that this protection is illusory, as the attacker has neither to attack this cipher nor user´s secret passphrase. A modification of the private key file in a certain manner and subsequent capturing of one signed message is sufficient for successful attack. Insufficient protection of the integrity of the public as well as private parts of signature keys in the OpenPGP format is analyzed in DSA and RSA algorithms and on the basis of this, a procedure of attacks is shown on both private signature keys. The attacks apply to all lengths of parameters (modules, keys) of RSA and DSA. In the end the cryptographic measures for correction of the OpenPGP format as well as PGP™ format are proposed.

1 Introduction

The OpenPGP format is defined in the document [1] from the year of 1998. Its objective was to publish all necessary information about the OpenPGP format so that various interoperable applications could be created on its basis. It describes the massage formats (data structures) and a manner of how they are to be created. In this contribution we point at a serious mistake of OpenPGP format, which consists in insufficient security of integrity of both public and private parts of signature keys of DSA and RSA algorithms. We show that this mistake may be used to reveal the private signature key. In order to protect the private key, its value is stored in the private keyring (file) secring.skr in the encrypted form. A strong symmetric cipher is selected for this by the user (AES, CAST5, IDEA) with sufficiently long key, which is derived from the secret access password (password, passphrase), known only by the user. In the contribution we try to show that if the intruder has access to this file (or a record), it may obtain the user´s private signature key under certain circumstances without the need to know its passphrase or without having to attack on it. The attack consists in a special modification of parameters of the signature algorithm and obtaining of a signature of any file (e-mail message) by such a modified signature key. We show that the intruder is able to compute the private signature key based on this procedure. As the corrupted record or file (secring.skr) can be restored back to its original condition), this intrusion is very dangerous. Private keys transferred in encrypted form on floppy disks or over the network are threatened the same way. The attack was practically tested on the latest version of PGP™ 7.0.3 program with a combination of AES and DH/DSS algorithms. Revealed private signature key of the DSA algorithm was the result of the attack. In the end we propose certain security and cryptographic measures for correction of the OpenPGP format and changes in the PGP™ program. All other applications which are compatible with OpenPGP format must undertake a

detailed revision. This applies for instance to the GNU Privacy Guard and other applications specified on the list of applications compatible with OpenPGP, which is available at http://www.pgpi.org/products.
The following text is organized as follows: first we remind of a definition of the signature scheme of DSA and the storage format of the private keys according to OpenPGP [1] . Then we describe an idea of the attack on the private signature DSA key and the specific procedure of the attack as we performed it in the PGP™ program. Further we describe an idea of the attack on the signature key RSA, stored according to the format OpenPGP. Then we specify the basic temporary measures for protection of the private keys in the PGP™ program and proposals for revision of the OpenPGP format. In appendices we present the technical details of the attacks.

2 DSA Signature Algorithm

For the needs of this article we will recap the procedure of creation of the key pair and the signature using 1024bit DSA algorithm (see for instance [2], page 452) and the signature verification procedure and introduce the necessary variables.

2.1 Creation of the Key Pair

Let us refer to SHA-1 function as to h. Every user will generate the private and public key as follows:

1. Select 160bit prime number q , $2^{159} < q < 2^{160}$.
2. Select 1024bit prime number p in such a way that q divides (p-1) and $2^{1023} < p < 2^{1024}$.
3. Select generator g of the cyclic subgroup of order q in $\mathbf{Z}_p^*$ (that means element $\alpha \in \mathbf{Z}_p^*$ will be selected so that $g = \alpha^{(p-1)/q} \bmod p$ and $g \neq 1$, otherwise different α is selected).
4. Select random number x in such a way, that $1 \leq x \leq q-1$.
5. Compute $y = g^x \bmod p$.
6. The public key is y, the public parameters are (p, q, g), the private key is x.

2.2 Creation of a Digital Signature

When creating the signature of the message m (or its hash value h(m)) the user uses its private key x and public parameters according to the following procedure:

1. Select the random secret number k, $0 < k < q$.
2. Calculate $r = (g^k \bmod p) \bmod q$.
3. Calculate $kInv = k^{-1} \bmod q$.
4. Calculate $s = [kInv * (h(m) + x*r)] \bmod q$.
5. Digital signature of the message m is the pair (r, s).

Let us note that r, s, q are generally 160bit numbers, whereas p, g, y are 1024bit numbers.

2.3 Verification of the Digital Signature

Upon verification of the digital signature of the message m we use signer´s public key y and public parameters (p, q, g) according to the following procedure.

1. Verify that $0 < r, s < q$. In the opposite case the signature is invalid.
2. Calculate $sInv = s^{-1} \bmod q$ and hash value h(m).
3. Calculate $u1 = sInv * h(m) \bmod q$, $u2 = sInv * r \bmod q$.
4. Calculate $v = (g^{u1} * y^{u2} \bmod p) \bmod q$.
5. The signature is valid iff v = r.

3 Description of the Secret Key Packet Data Structure for Storage of the Private Signature Key according to OpenPGP

Here we describe the data structure (Tag) "Secret Key Packet", in which the primary signature key is stored (RSA, DSA). There are two versions of this format. Version 3 is valid for RSA keys only, version 4 may include DSA and RSA keys. We will attack on the RSA signature key using both format versions and on DSA signature key using version 4 of the format. The version 4 of this format is used also by the program PGP™ 7.0.3, which prefers DSA to RSA. In practice we will therefore meet in particular with the RSA keys in 3 format and with DSA keys in the 4 format. Let us note that the versions 3 and 4 are different in a encryption method of the private data and therefore also the attacks on both algorithms are different. In both versions of the format, the structure of the Secret Key Packet contains in the beginning only the data from the structure of the Public Key Packet, concerning the public key, and then the data concerning the private key. Description and the content of the individual items are illustrated in Table 1 for the DSA algorithm and Table 2 for the RSA algorithm. Regarding the content of the table let us repeat that MPI format (multi-precision integer) contains prefix and then the actual (big) integer number in the entry BIG ENDIAN. The prefix forms two octets in BIG ENDIAN and indicates the number of valid *bits* of the subsequent number [1] .

<table>
<tr><td rowspan="8">Public Key Packet</td><td>1 octet indicating the version number</td><td rowspan="11">Area of publicly available data</td></tr>
<tr><td>4 octets indicating time, when the key was created</td></tr>
<tr><td>1 octet indicating algorithm for creation of a digital signature. Then the fields (numbers in MPI format) follow containing public parameters and public key for 1024bit DSA algorithm:</td></tr>
<tr><td>prime number p (2 + 128 octets in figure 1)</td></tr>
<tr><td>prime number q (2 + 20 octets in figure 1)</td></tr>
<tr><td>number g (2 + 128 octets in figure 1)</td></tr>
<tr><td>user`s public key y (2 + 128 octets in figure 1)</td></tr>
<tr><td colspan="2">1 octet (string-to-key usage), indicating whether and how the private key is encrypted. The value 0xFF is preferred indicating that the following three optional items are completed.</td></tr>
<tr><td colspan="2">[Optional] In case that string -to-key usage is 0xFF, there is 1 octet here, identifying a symmetrical encryption algorithm for protection of the private key.</td></tr>
<tr><td colspan="2">[Optional] If the string-to-key usage is 0xFF, there is a "string-to-key specifier", which says, how the password of the user is processed for a symmetrical key. The value 0x03 is preferred, indicating the so-called iterated and salted string to key specifier. Typically the following data is stored here with the following meaning:
1 octet: 0x03 (iterated and salted string-to-key identifier)
1 octet: identifier of the hash algorithm (for SHA-1 it is 0x02)
8 octets: salt (random data, which are hashed together with the user´s passphrase and diversifies thus derived symmetrical key)
1 octet: the number of hashed octets of the data (the so-called "count").</td></tr>
<tr><td colspan="2">[Optional] If the private key is encrypted, initialization vector (IV) is stored here. This is random data in the length of the block cipher (8 octets for 64bit block ciphers, 16 octets for AES algorithm).</td></tr>
<tr><td colspan="2">Algorithmically dependent numbers in the format MPI. For DSA only private exponent x is here:</td><td rowspan="4">Area of Sensitive Data</td></tr>
<tr><td colspan="2">2 octets, prefix of the x number (in version 4 encrypted, in version 3 not encrypted)</td></tr>
<tr><td colspan="2">20 octets, x number (in version 3 and 4 encrypted)</td></tr>
<tr><td colspan="2">2 octets, checksum, arithmetic sum of 22 previous octets as plaintext, modulo 65536 (in version 4 encrypted, in version 3 not encrypted).</td></tr>
</table>

Table 1: Content of the Secret Key Packet structure for 1024bit algorithm DSA

Now let us stop at files of secring.skr type in programs of PGP™, where the PGP™ program saves the Secret Key Packet structure. This file stores apart from the Secret Key Packet field usually several other records such as

- UserID Packet (contains an identifier of the user, i.e. its name and e-mail address),
- Signature Packet (contains time of signature, key expiration and so on),
- Secret Subkey Packet (contains similar data such as Secret Key Packet, but this time about asymmetric key and algorithm for encryption of the data),
- another Signature Packet (contains the data such as for instance a time of signature of this key by the signature key, amount/size of trust to it and so on).

These other records however do not contain any check of integrity of the Secret Key Packet record. This provides for successful attack on the Secret Key Packet.
Now let us describe the attack separately on RSA and DSA separately.

4 Attack on DSA Signature Algorithm

Let us notice that the integrity of the "Public Key Packet" field is not visibly secured anywhere in the format of OpenPGP, and as it became apparent by effecting a practical attack, not even in PGP™ programs. Nevertheless, when creating the digital signature it is public parameters of this field that are just utilized (in the event of PGP™ program, the Secret Key Packet is stored specifically in secring.skr file). These parameters could be read from the record of the public key (the file pubring.pkr), but it is logical that if the record of the private key is open, they will be read from here. In the record of Secret Key Packet the value of the private signature key is protected, but the mistake is that here the value of public parameters or public key is not protected anyhow. Specifically in the event of DSA values p, q, g, y are at issue, of which we will use only p, g for specific attack.

The main idea of the attack on DSA consists in the following steps. The attacker:

1. will prepare special numbers (constants) of PGPrime and PGGenerator
2. will obtain the structure of Secret Key Packet of the given user and replace p, g values stored in the structure "Public Key Packet" inside Secret Key Packet by values p´= PGPrime and g´= PGGenerator
3. will capture the first not enciphered message or the file which the user signed with such false parameters and will keep its signature
4. on the basis of the obtained message and its signature it will calculate the private key of the user (x value)
5. will return the p, g values to the original condition

The procedure of the attack on the DSA algorithm will be now described in detail and specifically as we carried it out utilizing the PGP™ 7.0.3. program. Examples and the procedure are specified for 1024 bit DSA. In the text below, we will always mark the foisted values and values computed on the basis of these false values with an apostrophe.

1.step

We selected prime number p´ (= PGPrime, constant) in such a way, that

1. p´ have 159 bits and the condition p´ < q be surely fulfilled.
2. upon entry of p´ in the format $p´ = t*2^s + 1$, 2^s was as big a number as possible and t a small prime number.

Specifically we selected s = 151 and t = 167, i.e. p´ has a binary format 10100111000....(150 zeros)....0001 and hexadecimally it is recorded as 0x5380 0000 0000 0000 0000 0000 0000 0000 0000 0001.

Then we selected the number g´ (=PGGenerator, constant) in such a way that

1. $1 < g´ < p´ - 1$.
2. g´ was a generator of the multiplicative group $\mathbf{Z}_{p´}^{*}$.

Specifically we chose g´ = 0x31AC8529 1FF814E6 25E4B88C 8C5047A7 DB2F0E45 and verified that $(g´)^{(p´ - 1)/2} \bmod p´ \neq 1$ and $(g´)^{(p´ - 1)/t} \bmod p´ \neq 1$.

2.step

Now we have obtained the file secring.skr and in its record of Secret Key Packet we exchanged the values (p,g) with values (p´, g´). Then we adjusted lengths of these numbers in the MPI format and cut short the overall length of the Secret Key Packet (values in the beginning of the record) in such a way, that it correspond to shorter false values p´ and g´. The situation is illustrated in figure 1 and 2.

3.step
With such foisted values, we waited till the user signs any file known to us (message m) and captured its signature - values (r´, s´). Now let us denote k a randomly chosen number, unknown to us, which the user program chose upon this signature (see the description of DSA above).

4.step
In this step, we calculated the value of the private key x of the given user on the basis of values p´, g´, m, r´ and s´. From the definition of the signature value (r´, s´) it results that

(1) $r' = (g')^k \bmod p' \bmod q$, which gives with reference to the choice $p' < q$

(1a) $r' = (g')^k \bmod p'$

and

(2) $s' = \{[k^{-1} \bmod q] * [h(m) + \mathbf{x}*r'] \} \bmod q$, thus

(2a) $\mathbf{x} = \{ [s'* k - h(m)] * [(r')^{-1} \bmod q] \} \bmod q$.

The key issue is now that we are able to calculate the unknown randomly chosen number thanks to the choice of PGPrime and PGGenerator. The prime number p´= PGPrime was selected in such a way, that the equation (1a), i.e. the task of the discrete logarithm in $Z_{p'}^*$ be easy to solve. The specific procedure of calculation of the discrete logarithm in this special group is specified separately in appendix 1. On the basis of this procedure we then calculated value k from the equation (1a) and additionally computed value **x** from the equation (2a).

We checked the correctness of **x** according to the relationship $y = g^{\mathbf{x}} \bmod p$ with original values y, g, p. The x value is therefore calculated and its validity is verified against the value of the public key.

5.step
To the user, we returned its original file secring.skr.

4.1 Practical Implementation of the Attack

The attack was applied to the latest version of PGP™ program v. 7.0.3 for Windows 95/98/NT/2000. From the nature of the attack and the used data formats (Public Key Packet versions 3 and 4, Secret Key Packet versions 3 and 4) it results that this attack should be successful also on other platforms. The procedure of the attack was selected precisely according to the aforesaid description. The adjustment of parameters was carried out in the field Secret Key Packet in the secring.skr file, where the private signature key is stored as we can see in figures 1 and 2. On the first figure, original values are indicated and the second one shows false values.

```
95 01D7 043A8D29DF11 0400 F2E02A396A14E137085DA859B3569AF4027EA379682F46780920B
72127C88787DDC1BFF9FDB59E564B741FD5FC98856679F1C041CB71895CB6975E7FE6E15A6D
4B70514560E11A25637F3FBA35E89E5F1FA272A2707F4865EA106EE402973D4969A276DA491
1005B968B81548621CEBBB5771A35C5A785F7F480E47277D2BAB500A0FF04303152BD2A9AD9
63E063A3FE34A8A5534F3F 0400 CD8580F20AA821A6D2FF5255DFD02E4F4C8D8DA3731517476
BEE096F7B104B01B6CE1C4DE586BAEA30D82B50DCB3F0D20B0F0D07D8384C09F12CBF079887
CEB696E822D753A48584F2BC573C84E8490AB310FDBCC40EAEBCD05973B3F2A1A479FFE0E4B
63026E066B6E936F1B2B7F1C91C65CBA0F27B4C0D22254BBC852DEDE10400AC756BB6EB8231
3A0FE91F47A36D1425D89FB124CD0ACBA082E8B2C2B048BE92C5CE7A5FAA5AF317DCC086150
B98AB504C0DA6BF1D87FAB73C8F8D0FC821BD8902CA6927338CF0D682E7C9E3E8D89A3D00D5
3224C301E6C932ADA7562FA15E9027E105F803043D4CBC08807A8FB71FEF9B27EE6A0722C4B
F601D032CC59F6FE4FF09030213CF38106B7BCA3F603F59437C3860B98DA3A1A3F02A4D2754
075B494CAC156E38E1282705FB0BBD68940A1653457E161AB00187B428566C617374696D696
C204B6C696D612C20445353203C762E6B6C696D6140646563726F732E637A3EB00303FFFF89
```

```
005A04101102001A05023A8D29DF050901E13380040B0908070219010 51B03000000000A091
09B89D5F084A0EAD7A35C009F5B643D5D2C37F4B2CFC9F399873B747CB3FBFE6800A0A8FE0A
2498E332586F6BED3BB88F278B0C5CF079B001679D014E043A8D2A78100400C904D0246F862
2352D6A60F67F1B4AAA4E94562BB00595A67DDB853AEF3F421CEE2D5FCD5AF18180872FE502
960093815901046096792 74CC92770C6DEBCF391A39B92270D71E7C5EBEC66B3A3BF1BDF521
7E9F609F5D011B9D648A930998C61CD462F3BAAAFD916FDBFFFAA01FCBD2E42F1BC5C406BB0
763B3D4830240841390002020 3FE2B39B802893DE670D745D2AE4DF802BDA707E829B7B0FF7
438FFB88EAB76189AA90A143FB11C1DCC5149046C913AB114D9775563BA0103E65C951D6AC9
199D52818BD2B8A8BF07A6E9F8C242811FF9522EE168207F1EC5D49B441C63D473F7C83D89C
3B6F43A3D80B1B38F7195DE45A55807207159A70CA883493532CD4D8802FF09030221D3B8EC
1276C0D3601B745E982D01201DA87DB47FD3B9C8CDD8F6BC857F56B6F4370AB8A94C7A4E528
D209A80B365A416AF80E1198BAC1AE4175A0F90B00187890052041811020012050 23A8D2A78
050901E13380051B0C000000000A09109B89D5F084A0EAD7CD04009F7056BA18F5907E36E4E
D9A79B4160AE8C6338D98009B043443B6665E860719768B49382DEC95FD7F96BBB00167
```

Figure 1: the file secring.skr hexadecimally, original values p,g are indicated with their lengths and total length of the record

```
95 00FF 043A8D29DF11 009F 538000000000000000000000000000000000000100A0FF0430315
2BD2A9AD963E063A3FE34A8A5534F3F 009E 31AC85291FF814E625E4B88C8C5047A7DB2F0E45
0400AC756BB6EB82313A0FE91F47A36D1425D89FB124CD0ACBA082E8B2C2B048BE92C5CE7A5
FAA5AF317DCC086150B98AB504C0DA6BF1D87FAB73C8F8D0FC821BD8902CA6927338CF0D682
E7C9E3E8D89A3D00D53224C301E6C932ADA7562FA15E9027E105F803043D4CBC08807A8FB71
FEF9B27EE6A0722C4BF601D032CC59F6FE4FF09030213CF38106B7BCA3F603F59437C3860B9
8DA3A1A3F02A4D2754075B494CAC156E38E1282705FB0BBD68940A1653457E161AB00187B42
8566C617374696D696C204B6C696D612C20445353203C762E6B6C696D6140646563726F732E
637A3EB00303FFFF89005A04101102001A05023A8D29DF050901E13380040B0908070219010
51B03000000000A09109B89D5F084A0EAD7A35C009F5B643D5D2C37F4B2CFC9F399873B747C
B3FBFE6800A0A8FE0A2498E332586F6BED3BB88F278B0C5CF079B001679D014E043A8D2A781
00400C904D0246F8622352D6A60F67F1B4AAA4E94562BB00595A67DDB853AEF3F421CEE2D5F
CD5AF18180872FE5029600938159010460967927 4CC92770C6DEBCF391A39B92270D71E7C5E
BEC66B3A3BF1BDF5217E9F609F5D011B9D648A930998C61CD462F3BAAAFD916FDBFFFAA01FC
BD2E42F1BC5C406BB0763B3D4830240841390002020 3FE2B39B802893DE670D745D2AE4DF80
2BDA707E829B7B0FF7438FFB88EAB76189AA90A143FB11C1DCC5149046C913AB114D9775563
BA0103E65C951D6AC9199D52818BD2B8A8BF07A6E9F8C242811FF9522EE168207F1EC5D49B4
41C63D473F7C83D89C3B6F43A3D80B1B38F7195DE45A55807207159A70CA883493532CD4D88
02FF09030221D3B8EC1276C0D3601B745E982D01201DA87DB47FD3B9C8CDD8F6BC857F56B6F
4370AB8A94C7A4E528D209A80B365A416AF80E1198BAC1AE4175A0F90B00187890052041811
02001205023A8D2A78050901E13380051B0C000000000A09109B89D5F084A0EAD7CD04009F7
056BA18F5907E36E4ED9A79B4160AE8C6338D98009B043443B6665E860719768B49382DEC95
FD7F96BBB00167
```

Figure 2: file secring.skr after intrusion, modified values p´,g´ are indicated with their lengths and total length of the record

5 Attack on Signature Algorithm RSA in OpenPGP

5.1 Brief Description of RSA

Here we briefly recap the definition of RSA algorithm (see for instance [6]) and introduce identification of the variables. Let us refer to RSA modulus as to n and let n = p*q, where p, q are prime numbers. Let us denote public exponent e and private exponent d. Then let us define pInv = p^{-1} mod q. We mark the pair (n,e) with public key. For the needs of OpenPGP format private key is considered to be represented by the quadruple (d, p, q, pInv). Signature of the m message is created as a value s = m^d mod n, while it is anticipated that the message m has already been formatted in advance in a certain manner. The signature of s message is valid if m = s^e mod n.

5.2 Description of Attack on the RSA Signature Key

We anticipate that the following thoughts are applied to the signature key of RSA algorithm and OpenPGP format, not for PGP™ programs. PGP™ programs have in-built mechanisms for integrity check of this key prior to its use for signature. In this case, we therefore attack only on the OpenPGP format.

As with DSA, even here there is a private key stored in the Secret Key Packet structure. At present, versions 3 and 4 of this format are used which differ in how the private data is encrypted. Both versions of Secret Key Packet contain in the beginning the data on the public key (in the Public Key Packet structure) and then the data of the private key follows. The structure Public Key Packet has also two versions of the format (3 and 4), but these differ only by one time piece of data. Therefore let us state only the content of the newer version 4 (see Table 2).

The Public Key Packet structure contains:

1. 1 octet version number
2. 4 octet number, indicating time, when the key was created
3. 1 octet identifying asymmetric algorithm (RSA here), belonging to this key
4. A sequence of MPI integers containing a public key. Here it is :

- n number (RSA modulus) in MPI format,
- e number (RSA public exponent) in MPI format.

After the Public Key Packet other data of Secret Key Packet follows, while the data of the private key is already encrypted as Table 2 shows. In RSA the following data is concerned:

- private exponent d
- prime number p
- prime number q ($p < q$)
- pInv ($= p^{-1} \bmod q$)
- two octets of checksum.

<table>
<tr><td rowspan="6">Public Key Packet</td><td>1 octet indicating the version number</td><td rowspan="11" colspan="2">Area of Publicly Available Data</td></tr>
<tr><td>4 octets indicating the time when the key was created</td></tr>
<tr><td>(in version 3 Public Key Packet 2 octets more indicating the number of days of the key validity)</td></tr>
<tr><td>1 octet indicating RSA algorithm</td></tr>
<tr><td>n number (RSA modulus) in MPI format</td></tr>
<tr><td>e number (public RSA exponent) in MPI format</td></tr>
<tr><td colspan="2">1 octet (string-to-key usage)</td></tr>
<tr><td colspan="2">[Optional] If the string-to-key usage is 0xFF, there 1 octet here identifying symmetric encryption algorithm for the private key protection (see Table 1).</td></tr>
<tr><td colspan="2">[Optional] If the string-to-key usage is 0xFF, there is "string-to-key specifier" (see Table 1).</td></tr>
<tr><td colspan="2">[Optional] If the private key is encrypted, initialization vector (IV) is saved here. This is random data in the length of the block of the used block cipher (8 octets for 64-bit block ciphers, 16 octets for AES algorithm).</td></tr>
<tr><td colspan="2">Algorithmically dependent numbers in MPI format. The content for RSA (modulus 1024 bits):</td><td>version 3</td><td>version 4</td></tr>
<tr><td colspan="2">2 octets, prefix of d number</td><td>plaintext</td><td rowspan="10">encrypted</td></tr>
<tr><td colspan="2">128 octets, d number</td><td>encrypted</td></tr>
<tr><td colspan="2">2 octets, prefix of p number</td><td>plaintext</td></tr>
<tr><td colspan="2">64 octets, p number</td><td>encrypted</td></tr>
<tr><td colspan="2">2 octets, prefix of q number</td><td>plaintext</td></tr>
<tr><td colspan="2">64 octets, p number</td><td>encrypted</td></tr>
<tr><td colspan="2">2 octets, prefix of pInv number</td><td>plaintext</td></tr>
<tr><td colspan="2">64 octets, pInv number</td><td>encrypted</td></tr>
<tr><td colspan="2">2 octets (H_{Sum} , L_{Sum}) checksum, a sum of previous items (MPI numbers) in open form (mod 65536)</td><td>openly</td></tr>
</table>

Table 2: Content of the structure Secret Key Packet version 3 and 4 for RSA algorithm (modulus 1024 bits)

Table 2 specifies how the private data encryption differs in versions 3 and 4 of Secret Key Packet. Both formats will be observed separately. A common element of both formats is a calculation of the checksum as a simple arithmetic sum of the individual bytes of the private data modulo 65536: checksum = $(d_1 + d_2 + ...+ d_n)$ mod 65536. In both versions the user uses for encryption the chosen symmetric block cipher in the so-called specific (PGP™) mode CFB.

Special feature of the version 3 of Secret key Packet format is that the prefixes of MPI numbers forming the private key and the checksum are not enciphered. What is more in the beginning of every MPI the condition of CFB is resynchronized in such a way, that the new block starts only from the new MPI value.

In the Secret Key Packet format – version 4 all private MPI including prefixes, checksums are encrypted without resynchronization.

5.3 Attack on Version 3 of Secret Key Packet (RSA) Format

The attack which may be launched on version 3 of the format consists in the possibility of changing the length of the individual MPIs, i.e. the MPI prefixes, as they are not encrypted and neither is checksum. For instance it is possible to reduce the prefix of MPI number pInv by one and to reduce also the checksum by one at the same time. The validity of the checksum of the record Secret Key Packet (with new pInv´) will be kept (all octets are summed up regardless of whether 1 or 8 bits are in them), but the value of the private information pInv is changed, as this number is cut short by 1 bit. Likewise, the values of other private numbers d, p, q can be changed. As soon as the attacker gets the message and its signature, which will be created using a private key (d, p, q, pInv´), it will be capable of computing the whole RSA private key. A detailed procedure is specified in Appendix 2. In this sense, the situation is entirely analogical to the attack against DSA.

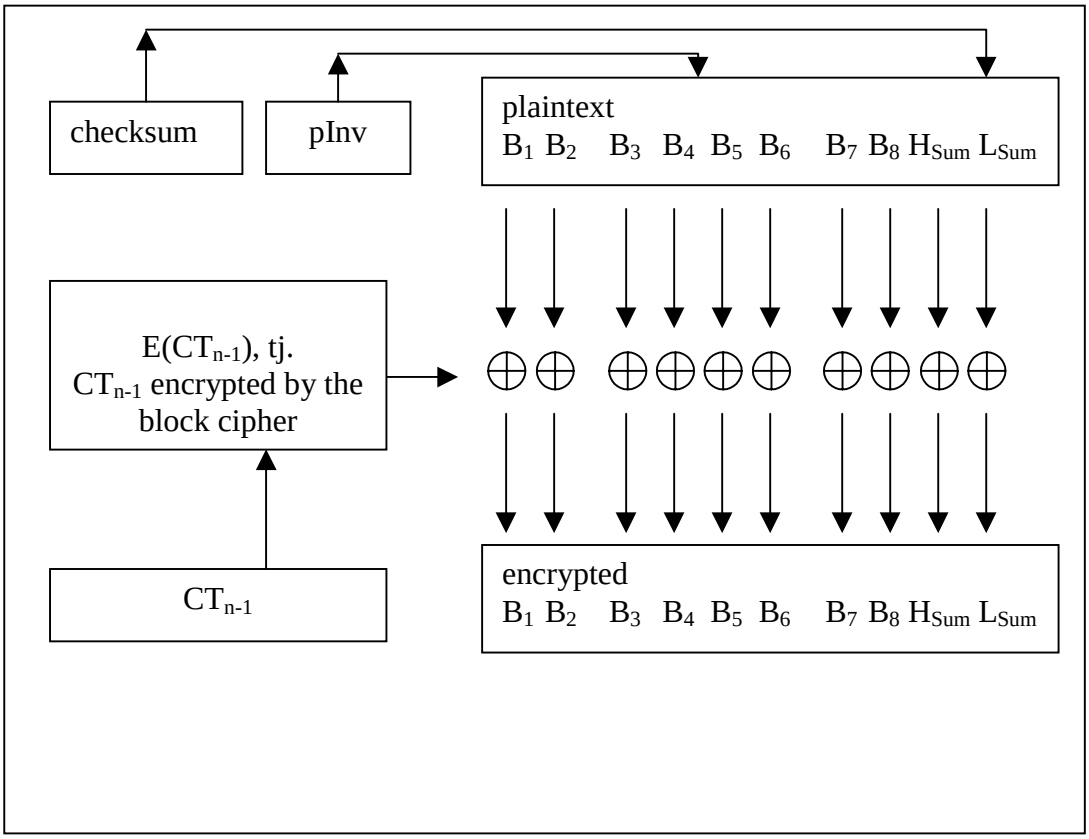

Figure 1: Encryption of the last data block of the private key in the Secret Key Packet, containing the private value pInv.

5.4 Attack on the Version 4 of Secret Key Packet (RSA) Format

Upon attack on the version 4 of the Secret Key Packet format it is not possible to utilize the previous procedure directly as the prefixes as well as the checksum are encrypted, but they may be modified. The modification consists in using the CFB properties in encryption the last block of plaintext. For instance if we use a block cipher with the block length of 16 octets (i.e. in the event of AES) and RSA 1024 bits modulus, the last incomplete block of the ciphertext will contain eight last octets of the pInv number (let us denote them B_1, B_2,..., B_8) and two octets of the checksum (let us denote them H_{Sum}, L_{Sum}). This plaintext will be encrypted by a simple XOR operation with the key material (encrypted previous cipher block). If we carry

out the change of “XOR CONST” type in the last block of the ciphertext, it will appear precisely as “XOR CONST” after decryption in the plain text. The objective is to carry out the change in pInv (to be more accurate in the last eight octets of pInv number) and simultaneously in checksum (in L_{Sum} octet) for pInv´ and checksum´ so that the check of integrity agree after deciphering. The result will again be an intruded value pInv, while the Secret Key Packet format will have the correct checksum. The use of this false pInv´ is the same as in the previous case. We will obtain a signature of some message with this false key and from here we will compute the value of the RSA private key. Now we will show how it is possible to change the individual bits of B_1 to B_8 octets and L_{Sum} in such a way, so that the check of integrity of the private key agree after their modification. The changes of the individual bits of the specified octets will be carried out on ciphertext and as we have already stated, with regards to the CFB mode, these changes will project in the same way into the corresponding bits of octets of the plaintext. Now let us anticipate that in the open format of some octet B_i from the set of $\{B_1, B_2,..., B_8\}$ some j-th bit is set (where j may be 0 to 7) *in the same way as* j-th bit in L_{Sum} octet. Then it only suffices to change this j-th bit in the enciphered octet B_i and simultaneously in the enciphered octet L_{Sum} and the check of integrity will agree. If this bit were 1, B_i octet will change to B_i - 2**j octet and L_{Sum} octet will change to L_{Sum} - 2**j. The new checksum will thus be valid. Likewise if j-th bit of B_i and L_{Sum} octets were 0, the octet B_i will change to B_i + 2**j and L_{Sum} octet to L_{Sum} + 2**j octet. The new checksum will again be valid.! As we do not know whether j-th bit of the selected octet B_i is and/or isn´t the same as j-th bit L_{Sum}, we will try it. We can change a total of 64 bits and the likelihood of us not succeeding is very low (2^{-64}). Let us note that we will learn of the success of the specified change only after the user tries to sign a message with this false key. Nevertheless on average two attempts should suffice. Another method is to change the j-th bit simultaneously always in two octets of any choice from the set $\{B_1,B_2,..., B_8\}$, whereas L_{Sum} will be left unchanged. This time we wait for a situation, when these bits are different in the plaintext. Their current change will annul their influence in the checksum. Likewise we can carry out variations with four or eight j-th bits. The result of this change is an intrusion of pInv with the same consequences as in previous cases, i.e. finding out of the signing RSA private key.

6 Attack on the Private Keys after their Export

Further it is necessary to note that apart from the private keyring secring.skr it is possible to use the same method to attack also on the private key, which is exported to the file of the type "ASCII Key File" and then transferred through the network or on a floppy disk. This file has apart from the additional encoding the same content as the secring.skr file and therefore the same attack can be applied to it as to the secring.skr file. This means that the transfer of the private key through this file over the network or on a floppy disk is not safe.

7 Countermeasures

7.1 Basic Temporary Countermeasures

The main cause of the just presented attacks is insufficient control of integrity of the public as well as private data in the file, containing the private key of the user. As a logical countermeasure a necessity results from this condition to introduce a better control of integrity of the saved records. We emphasize that this control must secure also the integrity of the public values, which must not be necessarily enciphered.

The requirement for introduction of a quality integrity check does not have to be easy to implement in a short-time horizon. Until the adjustment of the OpenPGP [1] format occurs for records of the private keys (Secret Key Packet), it is possible to use temporarily at least the following control tests in the PGP™ programs and others which implement the OpenPGP format. These are proposed in such a way, that the keys for DSA and RSA algorithms, which fulfill the below-mentioned relations, defeat the attack described by us. It is presumed, that this test will be carried out as an additional check of integrity after reading the respective parameters from the file with the private key. For the operation of the actual signature only such a key may be used the values of which pass this test. We stress out that the aforesaid test is not to be a substitute for the missing check of integrity of the file with the private key but it is to serve as a temporary measure which prevents the herein specified attack.

7.2 Temporary Test for DSA

We propose the following temporary test for DSA. The following relations should be verified:

1. p, q, g, x, y > 0
2. p is odd, q is odd
3. $2^{159} < q < 2^{160}$
4. $1 < g < p$
5. $1 < y < p$
6. $x < q$
7. $q \mid (p-1)$
8. $g^q \bmod p = 1$
9. $g^x \bmod p = y$

Let us also note that whereas such type of tests as we already mentioned is trusted a lot, for instance in RSA, in case of DSA we have to be careful. Unlike RSA there is only one value here (private key x), unknown by the attacker. Other parameters are known by the attacker and it may change them at its own discretion. This is a reason, why this test is considered only a temporary solution, which must be replaced as soon as possible with another type of check of integrity of the discussed records, as hereafter specified.

7.3 Temporary Test for RSA

We propose this temporary test for RSA. The following relations should be verified:

1. e*d mod (p - 1) = 1
2. e*d mod (q - 1) = 1
3. pInv * p (mod q) = 1
4. n (from the record of the public key) = p*q
5. $e \in E$, where E is a set of possible values planned for e, i.e. for PGP™ for instance {17, 65537,...}

Let us note that in the program PGP™ the inspections 1 to 4 and other checks are implemented. In OpenPGP format these controls are not however anticipated, which is a cardinal mistake.

7.4 Other Topics for OpenPGP format

Here we state some other topics which came to our mind after first familiarization with the OpenPGP format (the record Secret Key Packet) and PGP™ program. These topics would definitely contribute to increased safety of the format as well as PGP™ program. However, it

is necessary to perceive them as recommended ideas only. Before specific adjustments are implemented, they should undergo at least a basic independent analysis. The analysis of the whole OpenPGP format is however much more complex and reaches beyond the frame of this paper. The proposed measures are:

1. Enciphering modus CFB should be replaced with CBC modus
   - will make the described attack on the last block of enciphered private data more difficult
2. Replace the checksum (sum of bytes mod 65536 with HMAC, based on SHA-1 or on another safe hash function (for instance SHA-256, 384, 512 and so on)
   - will make the attacks on protected data and simultaneously on the security code more difficult
3. New checksum (HMAC):
   a) save in the length of at least 160 bits,
   - will make the attacks using the birthday paradox more difficult,
   b) compute it from all data of the record Secret Key Packet (not only from the private but also from the public values),
   - will make the integrity attacks on public as well as private parts of keys in Secret Key Packet more difficult
   c) derive the key used in HMAC from passphrase in another manner than the key for symmetric cipher
   - will make the attack on HMAC more difficult
   d) encipher the resulting HMAC together with the private data by a symmetric cipher similarly as it is in case of checksum in Verion 4 of the Secret Key Packet format
   - will make the integrity attacks on public as well as private parts of the key in Secret Key Packet more difficult
4. Use the format of EMSA-PSS type specified in [6] for the RSA signature scheme
   - will make a number of attacks including attack described in Appendix 2 more difficult.

8 Impacts

The demonstrated types of attacks show a considerable impact to security of the programs based on the format OpenPGP (e.g. PGP™ program itself). Anybody, who can change a file with private key, can get the private key value in algorithms DSA and RSA based on a single incorrect signature. And this change need not occur only in the workstation of attacked user at all. Sensitive point of the system can be also seen in the files with exported private keys, which are used by the user for a transfer of his private keys between various stations. The fact that private key is stored in an encrypted form may provide the user with some feeling of security, which is false, however. If attacker approaches to such flexible disk during its transport, a security of user´s private key is endangered considerably.

Another scenario, which is very efficient in case of the attack described above, can be used in situation, when the file with private key is stored on the shared device. In such a case, the attacker can be e.g. server administrator, who foists a modified version of this file upon the user for certain time period. Further on, he waits until the user uses it for signing (time period can be determined by monitoring of user´s station network activity with relatively high accuracy), and finally he returns the original content of the file. He can identify the private key value for the signature having been generated. When having sufficient control over the entire system, the attacker can even cover the tracks of the attack, when he sends a message with valid signature instead the original message furnished with false signature. This can be

done easily, because at this moment the attacker knows both the message itself and the proper private key.

User of the programs based on OpenPGP has to face to a very difficult situation, when he finds out that faulty signature value was generated. Definitely, he may wonder, whether he is facing to an impact of intentional attack, or it is "only" some technical failure. Of course, it is clear that proper care has to be paid to every file with false signature, as if it was a file containing private key in open form! Subsequent treatment includes, first of all, irreversible wiping of the file from the station or even the server in question.

9 Conclusion

The above mentioned attacks leading to revealing of the most sensitive system information (RSA and DSA algorithms of signature private keys), show clearly the importance of protection of private keys and public parameters of asymmetric algorithms in security systems. Let us mention also the fact that it was right the research study aimed to general problems and principles of protection, in the framework of which we saw "how the PGP™ program makes it", without a primary intention to target to it.
Our analysis was based on general documentation of OpenPGP [1]. We have revealed serious insufficiencies in it, which can lead to easy vulnerability of the applications based on it. As a practical example, we can mention the PGP™ program, which shows resistance to attacks to RSA thanks to sufficient protection beyond the scope of OpenPGP, but which is easily vulnerable by the attack to DSA signature algorithm.

Note that although we limited our study to RSA and DSA algorithms in relation to OpenPGP, similar vulnerabilities can be expected also in other asymmetric cryptosystems, including the systems based on elliptic curves, if sufficient protection of private keys and public parameters is not ensured. And the OpenPGP format as well as the PGP™ program are probably not the only cases, when attacking a given system is enabled by improper protection of the respective parameters. As a whole, this document calls for attention, which must be paid to designing of application of the above mentioned values and storing them within a given system.

Acknowledgement

We would like to thank Pavel Rydlo for technical co-operation in practical implementation of the attack under description.

References

[1] RFC 2440: *OpenPGP Message Format*, J. Callas, Network Associates, L. Donnerhacke, IN-Root-CA Individual Network e.V., H. Finney, Network Associates, R. Thayer, EIS Corporation, November 1998

[2] Menezes A.J., Oorschot P.C., Vanstone S.A.: *Handbook of Applied Cryptography*, CRC Press, 1997

[3] Lenstra, A. K.: *Memo on RSA signature generation in the presence of faults*, manuscript, Sept. 28, 1996. Available from the author.

[4] Rosa, T.: *Future Cryptography: Standards Are Not Enough*, sent to the conference CATE 2001, 2001.

[5] Rosen, K. H., Michels, J. G., Gross, J. L., Grossman, J. W., Shier, D. R.: *Handbook of Discrete and Combinatorial Mathematics,* CRC Press, 2000.
[6] PKCS#1 verze 2.1: RSA Cryptography Standard, RSA Laboratories, Draft 1, September 17, 1999

Appendices

Appendix 1: Determination of a private key value by changing the DSA public parameters.

Assumption P1: ***Method of DSA signature calculation***
Let us assume the DSA parameters (p, q, g, y, x), where p, q are prime numbers, $g \in Z_p^*$, ord(g) = q, $y = g^x \bmod p$, $0 < x < q$, x is signer´s private key. Signature for the message with hashing code h can be calculated as follows:

1. *select random k, $0 < k < q$*
2. *calculate $r = (g^k \bmod p) \bmod q$*
3. *calculate $s = k^{-1}(h + xr) \bmod q$, where $k*k^{-1} \equiv 1 \pmod q$*
4. *the signature is the pair (r, s)*

The attack described below assumes that the attacker changes the DSA parameters (p, q, g, y, x) for (p', q, g', y, x), where p' is 159 bit prime number, $2^{158} < p' < 2^{159}$ and g' is generator of group $Z_{p'}^*$ so that he could for any $r \in Z_{p'}^*$ easily determine the value w, $0 \le w < (p-1)$, for which it holds that $(g')^w \equiv r \pmod p$, i.e. $w = \log_{(g')} r$. So he can solve a problem of discrete logarithm in $Z_{p'}^*$ easily.

In the following two steps we demonstrate that the value of private key x can be easily determined from the known hashing code h and signature (r, s,), which was acquired by algorithm DSA with spurious parameters (p', q, g', y, x).

Step 1: determination of the set K
In this step we are working with the value r. It follows from equation *(P1.2)*, that $r = (g')^k \bmod p' \bmod q = (g')^k \bmod p'$, because $p' < q$.

According to the above given assumption, the attacker is capable to determine the value of $w = \log_{(g')} r$ easily for any r. For the unknown value of k we have $k = w + b(p'-1)$, where $b \in Z$, $b \ge 0$. The condition from *(P1.1)* must be added, assuming that $0 < k < q$. As p' is 159 bits and q is160 bits, so we obtain a set of the values admissible for k, i.e. $K = \{ w + b(p'-1): b(p'-1) < q-w, 0 \le b \le 3 \}$. This way we get a set of at most four possible k_i values, and the value k under search is surely among them.

Step 2: determination of the value x
Now, we will test $k_i \in K$ one after another and compute the values of x_i, from *(P1.3)* as $x_i = r^{-1} (k_i*s - h) \bmod q$, where $r*r^{-1} \equiv 1 \pmod q$. Note that $\gcd(r,q) = 1$, so the value r^{-1} exists and is unique. This way we obtain a set of the values $X = \{x_i: k_i \in K\}$, which must include the private key value x being searched for.

All we have to do now is to choose the required value x from the set X. It can be done easily with the help of relation $y = (g^{xi} \bmod p)$, where p and g are the original public parameters of DSA and y is the public key. By testing four different values (as the maximum) of x_i, we can eliminate remaining uncertainty introduced by a low value of p' and obtain the value x under search. Note that the element g has the order q within the group Z_p^*. As a result, we can say that only one value of $0 < x_i < q$ exists, for which it is true that $y = (g^{xi} \bmod p)$. That is why the x value determined by this procedure is unique.

Note. *The main principle of the entire attack is obvious from the above given description. It is based on such modification of DSA public parameters, thanks to which very weak instances of the problem of discrete logarithm occur in signature calculations. Instead of solving this problem in multiplicative cyclic subgroup of the group Z_p^* having the 160-bit order, this problem can be solved sufficiently in the cyclic group $Z_{p'}^*$, where p' is a 159-bit prime number with a suitably chosen structure.*

The complexity of such instances of the discrete logarithm problem is considered as absolutely insufficient from the cryptology point of view. Moreover, for practical implementation of the above described attack, a generally applicable group $Z_{p'}^*$ has been found. This group has a special structure, which enables very quick solution of the instances generated in the above given problem, even on a PC of common office type - see the description of algorithm A1 given below.

Algorithm A1: Calculation of a value of $w = \log_g r$ for special type of Z_p^*.

In the following part, we describe an efficient algorithm for calculation of a value of discrete logarithm, which can be used for multiplicative group Z_p^* that have a certain special structure (here p equals to the spurious value of p'). It is assumed that a generally selected group with this structure will be used for practical implementation of the above described type of attack against DSA. A structure of the above mentioned group will be given in a form of the following proposition:

Proposition P2. Let us have *a multiplicative group Z_p^*, where p is a prime number with a structure of $p = t*2^s + 1$, and t is a prime number. Further on, g is generator of Z_p^*. The following procedure shows a method of calculation of the w value, which is efficient for low t (for practical implementation of the foregoing attack, the generally applicable group with the parameters of **$t = 167, s = 151$** was found).*

Before we start describing single steps of the algorithm itself, we should mention some useful formalisms to be used later on for explaining particular operations.

Definition D1. [see 5, page 277] If p and k are positive integers *and b is an integer relatively prime to p, then b is kth power residue of p if the congruence $x^k \equiv b \pmod p$ has a solution. (In case of k=2 we often use an expression of quadratic residue modulo p.)*

Fact F1. [see 5, page. 279] *Let p be a prime number, k an integer positive number, and b an integer number fulfilling the condition of gcd(b, p) = 1. Then b is kth power residue modulo p, iff $b^{(p-1)/d} \equiv 1 \pmod p$, where d = gcd(k, p-1).*

Lemma L1. *Let p be a prime number and g the generator of group Z_p^*. Then the value of y, $y = g^w \bmod p$, is kth power residue modulo p, where $k|(p-1)$, iff $k|w$.*

Proof. If y is *kth power residue modulo p,* , then from the fact F1 we have $y^{(p-1)/k} \equiv 1 \pmod p$. From the assumption $y = g^w \bmod p$, we get $(g^w)^{(p-1)/k} \equiv 1 \pmod p$. As g is the generator of Z_p^*, is must be valid that $w*(p-1)/k \equiv 0 \pmod{(p-1)}$. From this expression, we can directly determine that k|w.

Verification of implication in the reverse order is easy. Let k|w, i.e. w=k*b, where b is positive integer. Then it applies that $y = g^w \bmod p = (g^b)^k \bmod p$ and it results directly from the definition D1 that y is *kth power residue modulo p.* °

Further on, we describe three steps of algorithm - one after another - in which a calculation of the discrete logarithm $w = \log_{(g)} r$ under search takes place. It is based on a modified version of Pohling-Hellman algorithm (see [2]), which would be also very efficient in the given type of multiplicative group. In our effort for the most specific structure of the group employed, we derived the following procedure.

Step 1: determination of the value $^s w = w \bmod 2^s$

Let us assume that $w = w_n*2^{n-1} + w_{n-1}*2^{n-2} + ...+ w_1$, where n is a number of bits of binary expansion w and $w_i \in \{0,1\}$, for $1 \le i \le n$.

Let us concentrate our attention to determining of the bit w_1. If this bit is zero, it is valid that w = 2*b, for some integer b. For the value $r = g^w \bmod p$, we can derive that $r \equiv (g^b)^2 \pmod p$, so that r is the quadratic remainder of modulo p. On the other hand, if the value of bit w_1 is one, then the value of w is odd, and r is not a quadratic residue modulo p according to the lemma L1. Based on this analysis and the fact F1, we can define the following conditions for w_1:

- $r^{(p-1)/2} \equiv 1 \pmod p \Rightarrow w_1 = 0$
- $r^{(p-1)/2} \not\equiv 1 \pmod p \Rightarrow w_1 = 1$

Let us continue with determination of w_2. First we adjust r for $r_2 = (r*g^{-w1}) \bmod p$ based on the known w_1. By this adjustment we obtain the value $r_2 = g^{w'} \bmod p$, where $w' = w_n*2^{n-1} + w_{n-1}*2^{n-2} + ...+ w_2*2$. If it is true that $w_2 = 0$, we obtain a equation for the value r_2, saying that $r_2 \equiv (g^b)^4 \pmod p$, where b is an integer. This means that the value r_2 is a 4th *power residue modulo p* in this case.

If it is true that $w_2 = 1$, w' is not divisible by four and the value r_2 is not a 4th *power residue modulo p* according to the lemma L1. By this, we can derive the following conditions for w_2:

- $r_2^{(p-1)/4} \equiv 1 \pmod p \Rightarrow w_2 = 0$
- $r_2^{(p-1)/4} \not\equiv 1 \pmod p \Rightarrow w_2 = 1$

To determine w_2 we adjust r_2 again for r_3 as $r_3 = (r_2*g^{-2*w2} \bmod p)$ and we continue in determining the values of w_i while it is true that $2^i \mid (p-1)$.

Procedure of calculation of ${}^s w = \log_g r \bmod 2^s$.

1. Let us assume that:
   a. binary notation ${}^s w$ is ${}^s w = w_s w_{s-1} \ldots w_1$
   b. Z_p^*, where p is prime number, $p = t*2^s + 1$
2. $i = 1$; $f = g^{p-2} \bmod p$; $v = p-1$
3. $v = v/2$; $y = r^v \bmod p$
4. **if** $(y = 1)$ **then** $w_i = 0$ **else** $w_i = 1$; $r = r*f \bmod p$
5. $f = f^2 \bmod p$
6. $i = i+1$
7. **if** $(i \le s)$ **then goto** 3
8. **return** ${}^s w = w_s w_{s-1} \ldots w_1$

Fig. 3: Step 1 of algorithm A1.

As $(p-1)/2^s = t$, where t is an odd prime number, we can determine the values of w_i for $1 \le i \le s$. By this, we obtain the binary notation of the value ${}^s w = w_s 2^{s-1} + w_{s-1} 2^{s-2} + \ldots + w_1$, where ${}^s w = w \bmod 2^s$. It is the value, we wished to identify in this step. The entire procedure is illustrated in Fig. 3.

Step 2: determination of the value ${}^t w = w \bmod t$

It is easy to demonstrate that for the integer j fulfilling the condition of $r^{(p-1)/t} \equiv (g^{(p-1)/t})^j \pmod p$, it holds that ${}^t w \equiv j \pmod t$. When $j \le t-1$, it applies directly that ${}^t w = j$. We can identify the value ${}^t w$ in this step by testing the numbers of j, $0 \le j \le t-1$, until we identify the j value fulfilling the congruence $r^{(p-1)/t} \equiv (g^{(p-1)/t})^j \pmod p$. Such j is also the value of ${}^t w$ under search.

Step 3: determination of the value $w = \log_g r$.

In the preceding steps, we have obtained a set of the following congruencies:

- $w \equiv {}^s w \pmod{2^s}$
- $w \equiv {}^t w \pmod t$

It is also true that $\gcd(t, 2^s) = 1$, and so, according to the Chinese Remainder Theorem (CRT), an unique value $0 \le w < t*2^s$ exists, which fulfils both congruencies. As the value $t*2^s$ is also the order of group Z_p^* for $p = t*2^s+1$, the value of w is also the discrete logarithm of the value r under search. Direct procedure leading to determination of w follows:

1. calculate $\gamma \equiv (2^s)^{-1} \pmod t$, where this value exists and is unique, because $\gcd(t, 2^s) = 1$
2. calculate $v = ({}^t w - {}^s w)*\gamma \bmod t$
3. $w = {}^s w + v*2^s$

Proof (of correctness of foregoing procedure):

For factor 2^s, it is obvious from the expression for w that $w \equiv {}^s w \pmod{2^s}$. For factor t, we obtain that $w \equiv {}^s w + ({}^t w - {}^s w)(2^s)^{-1}*2^s \pmod t$, and so $w \equiv {}^t w \pmod t$. Moreover, $w = {}^s w + v*2^s < 2^s + (t-1)*2^s = t*2^s$. By this we have verified that the above given procedure actually corresponds to application of CRT in the above given set of congruencies. °

Results of the experiment

The procedure described in steps 1 through 3 was applied in various configurations of office PCs. Table 3 shows average time of calculations for randomly chosen values of r. We can see that the entire calculation takes some hundreds of milliseconds in general.

Configuration	Time of calculation of a single discrete logarithm given in milliseconds
Pentium III/ 500MHz 128 MB RAM Windows NT 4.0 SP 6a	96
Celeron 400 MHz 128 MB RAM Windows NT 4.0 SP 6a	113
Pentium II 400 MHz 128 MB RAM Windows 2000 Advanced Server	116
Pentium II 300 MHz 128 MB RAM Windows NT 4.0 SP 6a	150
Pentium 166 MHz 96 MB RAM Windows NT 4.0 SP 6a	535
Pentium 75 MHz 46 MB RAM Windows NT 4.0 SP 4	1020

Table 3: Times of calculation of the value $w = \log_g r$ for special type of Z_p^*.

Appendix 2: Attack to a private RSA key

In this appendix, we describe briefly how the value of a private RSA key can be obtained from the value of a faulty signature, which was calculated using the affected private key. This attack is based on an analysis of OpenPGP format. This attack could not be applied on the tested PGP™ directly, because the PGP™ program performs additional check of the private key integrity beyond the scope of OpenPGP definition. However, such attack is possible in case of applications implemented strictly according to OpenPGP, and it has the same effects as the above presented attack to DSA.

By OpenPGP, the private RSA key consists of the following set of six values (n, p, q, pInv, e, d), where p, q are prime numbers, $n = p*q$ is public modulus, $p*pInv \equiv 1 \pmod q$, e is public exponent and d is private exponent, i.e. $e*d \equiv 1 \pmod{\text{lcm}(q-1,p-1)}$. Based on this structure, one can assume that the RSA signature transformation is calculated for the specific value of formatted message m by the following procedure:

1. $s_1 = m^d \bmod p$
2. $s_2 = m^d \bmod q$
3. $h = pInv*(s_2 - s_1) \bmod q$
4. $s = s_1 + p*h$

5. s is the result of signature transformation; it can be derived that for the value s calculated by this procedure, it applies that $s = m^d \bmod n$

This procedure corresponds to application of the Chinese Remainder Theorem on the signing transformation, and it makes it possible to calculate this transformation efficiently. As it was shown in [3] for the first time, application of this technique is quite susceptible to attacking, making use of the error in calculation of signature. And these errors can be implemented not only by affecting of the attacked device during calculation of signature, but also e.g. by affecting of certain values forming private key. This topic is described in detail in [4]. Here we draw our attention to one specific type of attack, which should be considered in OpenPGP.

Let us assume that the attacker affects the parameters RSA (n, p, q, pInv, e, d) by substituting pInv with $pInv' \in Z$, $pInv' \not\equiv pInv \pmod q$. Let us mention that random change of pInv will fulfill this condition with high probability. Other parameters are left without changes.

Now, let us have a pair of values (m, s'), where the value s' was obtained as a result of above described signature transformation, in which the affected value pInv´ was used..
It is valid that

1. $s_1 = m^d \bmod p$
2. $s_2 = m^d \bmod q$
3. $h' = pInv' * (s_2 - s_1) \bmod q$
4. $s' = s_1 + p * h'$
5. s´ is the result of signing transformation

Regarding the factor p for this value as it results from the equation given in point 4 it is true that $s' \equiv m^d \pmod p$. However, it is very likely (probability close to $1 - q^{-1}$) that for factor q $s' \not\equiv m^d \pmod q$ is true. A pair of numbers (m, y), where $y = (s')^e \bmod n$, then fulfils the following conditions: $m \equiv y \pmod p$, $m \not\equiv y \pmod q$. It results from it that $p|(m-y)$, but q does not divide (m-y) at the same time. That is why it is valid for the factor p that $p = \gcd((m-y), n)$.

By the above given procedure we have obtained the value of factor p from a single faulty signature, and the remaining secret values of private key can be determined from it easily.

Let us mention that the above given procedure is based on a condition that the attacker knows the value of formatted message m, which directly enters the RSA signature transformation. And this condition need not be fulfilled for all types of formats, but the format RFC 2313 (alias PKCS#1, version 1.5) is recommended in OpenPGP, where this condition is met.

Similarly as in case of attack to DSA we recommend to introduce more powerful check of data integrity in the private key files into the format OpenPGP. No correction is necessary directly in the program PGP™ 7.0.3, as subsequent checks of algebraic relations between the values of private key are employed, which defeat the attempts for attack of this type.