Recovery document for key 83224

Fri Feb 1 16:46:11 UTC 2019

This document contains the recovery procedure for the *private* key whose public key has the fingerprint shown below: '

```
/Volumes/tmp.83224/pubring.kbx
------------------------------
pub   ed25519 2019-02-01 [SC] [expires: 2021-01-31]
      1DBF E0AF 9276 E1B8 C3B2  6A3E 993D 670E 6F8D E438
uid           [ultimate] key-83224
sub   cv25519 2019-02-01 [E]
      26CD 9E12 0058 6738 82B5  00E9 F1C0 22B5 E4D2 D4D8
```

Process

Prior to opening the sealed envelope with the private key details we suggest that you:

1. Consider having a second person acting as an observer; and document the steps taken during recovery.
2. Ensure that you have a clean machine; and that your pgp, gpg or gnupg supports "ECDSA" as a pubkey (Use "gpg –version" to check).
3. Have acces to a QR decoder; or have a library such as https://github.com/dlbeer/quirc.git compiled and tested. (The alternative is OCR or entering half a page (500 numbers) by hand).

Decode

Unfortunately most QR decoders are somewhat careless about binary data and loose the ‘top bit’ Code known to correctly extract QR binary code is shown below. Needs to be linked against:

> https://github.com/dlbeer/quirc.git (– v1.00, commit 307473d on 2018-02-01).

```
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include "quirc_internal.h"
#include "dbgutil.h"

int main(int argc, char **argv)
{
        struct quirc *q;

        if (argc < 2) {
                fprintf(stderr, "Usage:␣%s␣<testfile.jpg|testfile.png>\n", argv[0]);
                return -1;
        }

        q = quirc_new();
        if (!q) {
                perror("can't␣create␣quirc␣object");
                return -1;
        }

        int status = -1;
        if (check_if_png(argv[1])) {
                status = load_png(q, argv[1]);
        } else {
                status = load_jpeg(q, argv[1]);
        }
        if (status < 0) {
                fprintf(stderr,"loading␣of␣png/jpeg␣image␣failed:␣%s\n", quirc_strerror(status));
                quirc_destroy(q);
                return -1;
        }
        quirc_end(q);

  int count = quirc_count(q);
  if (count != 1) {
    fprintf(stderr,"Expected␣exactly␣one␣image.␣Got␣%d\n", count);
                quirc_destroy(q);
                return -1;
  };

        struct quirc_code code;
        struct quirc_data data;

        quirc_extract(q, 0, &code);
        int err = quirc_decode(&code, &data);
  if (err) {
                fprintf(stderr,"quirc␣decode␣failed:␣%s\n", quirc_strerror(err));
                quirc_destroy(q);
                return -1;
  };

        for(int i = 0; i < data.payload_len; i++)
                printf("%c", data.payload[i]);

        quirc_destroy(q);
        return 0;
}
```

Compile and extract raw key

A typical sequence of steps to compile the decoder and import thee key into a GPG is shown below. The 'XXXX' refers to the path to the C code listed on the previous page. The 'generation' script on the next page does much the same thing in the section *'Decode the key and verify ...'*.

```
#!/bin/sh
#
set -x
set -e

mkdir extract.$$ && cd extract.$$

git clone https://github.com/dlbeer/quirc.git
( cd quirc && make )

gcc -I quirc/tests -I quirc/lib  -o decode \
  ../decode.c  \
  quirc/lib/*.o quirc/tests/dbgutil.o \
  -ljpeg -lpng -lSDL

# Decode QR code.
./decode ~/Downloads/qr.png  > key.raw

# Check checksum
openssl sha256 key.raw

# Import key.
gpg --homedir . --import key.raw
```

Script used to generate this key

```
#!/bin/sh -e
umask 077
set -e
set -x

SCRIPT=$(realpath $0)
CODE=$(realpath $0/../decode.c)
EXTRACT=$(realpath $0/../extract.sh)
DIR=$(dirname $CODE)
EXPDIR=`pwd`
PASSWD=${PASSWD:-}
PIN=$(openssl rand -base64 128 | tr -dc 0-9 | cut -c 1-4)

# create an ephemeral disk that disappears once we're done.
#
openssl rand -base64 128 | hdiutil create -attach -stdinpass \
  -encryption -size 1M -fs HFS+ -volname tmp.$$ tmp.$$
rm tmp.$$.dmg
OUTDIR=/Volumes/tmp.$$

# Switch off spotlight indexing; and then disable it.
mdutil -d -i off $OUTDIR

# create and from hereon work on that ephemeral disk; that should
# disappear once we're done.
#
test -d $OUTDIR
chmod 700 $OUTDIR
cd $OUTDIR

# Disable the agent - as it will prevent us from unmounting the
# disk post generation due to it claiming a socket.
#
echo no-use-agent > gpg.conf

# generate a private/public keypair; and export it. We use ED25519
# as to allow it ti fit in a QR code (hence we keep the name short too).
#
gpg2 --homedir . --batch --passphrase "$PASSWD" --quick-generate-key key-$$ ed25519
FPR=$(gpg --homedir . --list-keys --with-colons key-$$| grep fpr | head -1 | cut -f 10 -d:)
gpg2 --homedir . --batch --passphrase "$PASSWD" --quick-add-key $FPR cv25519

# test that we can encrupt
echo All working ok | gpg2 --homedir . --encrypt  --batch --passphrase "$PASSWD" -r key-$$ > test.gpg
gpg2 --homedir . --decrypt < test.gpg

gpg2 --homedir . --batch --passphrase "$PASSWD" --export-options export-minimal --export-secret-keys key-$$  > gpg.
      priv.raw
gpg2 --homedir . --batch --passphrase "$PASSWD" --export-options export-minimal -a --export-secret-keys key-$$  >
      gpg.priv.raw.asc

# Encode the private key to both a hexdump and a QR code.
#
qrencode -l H -8 -s 8 -r gpg.priv.raw -o qr.png
hexdump gpg.priv.raw  > priv.txt

# Decode the key and check that it is identical
#
qrdecode qr.png > gpg.priv.raw.dec # quirc v1.00
diff gpg.priv.raw.dec gpg.priv.raw

# Import the key into a fresh keyring - to verify that that works.
#
mkdir test.$$
gpg2 --homedir test.$$ --batch --passphrase "$PASSWD" --import gpg.priv.raw.dec
gpg2 --homedir test.$$ --batch --passphrase "$PASSWD" --decrypt test.gpg

gpg2 --homedir . --batch --passphrase "$PASSWD" --list-keys  --fingerprint --with-subkey-fingerprint > gpg.txt
rm -rf test.$$

(
  uname -a
  echo
  gpg --version
  echo
  openssl version
  echo
  qrencode --version
  echo
  echo quirc
  echo "Copyright␣(C)␣2010-2012␣Daniel␣Beer␣<dlbeer@gmail.com>"
  echo Library version: 1.0
) > info.txt
# Generare the various pages.
#
DATE=`date -u`
SHA256=`openssl sha256 gpg.priv.raw |sed -e "s/.*=␣//" -e 's/\(........\)/ \1/g'`
SHA256A=`openssl sha256 gpg.priv.raw.asc |sed -e "s/.*=␣//" -e 's/\(........\)/ \1/g'`

KEYNAME=$$

SPECIMEN=""
if [ "x$1" = 'x-d' ]; then
  SPECIMEN="SPECIMEN"
fi

export CODE SHA256 SHA256A KEYNAME DATE SCRIPT EXTRACT SPECIMEN
cat $DIR/doc.tex | envsubst > privkey.tex

/usr/local/texlive/2018/bin/x86_64-darwin/pdflatex privkey.tex
/usr/local/texlive/2018/bin/x86_64-darwin/pdflatex privkey.tex

if [ "x$1" = 'x-d' ]; then
  open privkey.pdf
else
lpr -\# 2 \
  -o collate=true \
  -o com.brother.print.PrintSettings.secureprint=ON \
  -o com.brother.print.PrintSettings.cfjobname=key-$$ \
  -o com.brother.print.PrintSettings.cfusername=${LOGNAME:-key-$$} \
  -o com.brother.print.PrintSettings.password=$PIN \
  \
  `perl -e '($p, $f,$v)=@ARGV;$i=0; map { print "-o␣$p.$f..a.$i..n.=$_␣"; $i++; } unpack("c*",$v);; print "-o␣$p.".
       $f."cnt=$i␣-o␣$p.cf$f=$v";' \
    com.brother.print.PrintSettings jobname key-$$` \
  `perl -e '($p, $f,$v)=@ARGV;$i=0; map { print "-o␣$p.$f..a.$i..n.=$_␣"; $i++; } unpack("c*",$v);; print "-o␣$p.".
       $f."cnt=$i␣-o␣$p.cf$f=$v";' \
    com.brother.print.PrintSettings username ${LOGNAME:-key-$$}` \
  \
  privkey.pdf
fi

# Export the public key - for actual use. The private key will fromhereon
# only exist on paper.
#
gpg2 --homedir . --export key-$$ >  $EXPDIR/public-key-$$.gpg

# Ejecting the ephemeral disk will wipe all private key material.
# We use 'force' to foil MDS.
#
hdiutil eject -force $OUTDIR

echo
echo
echo Printer PIN is $PIN

exit 0
```

GPG listing

```
/Volumes/tmp.83224/pubring.kbx
------------------------------
pub   ed25519 2019-02-01 [SC] [expires: 2021-01-31]
      1DBF E0AF 9276 E1B8 C3B2  6A3E 993D 670E 6F8D E438
uid           [ultimate] key-83224
sub   cv25519 2019-02-01 [E]
      26CD 9E12 0058 6738 82B5  00E9 F1C0 22B5 E4D2 D4D8
```

Auxilary info

```
Darwin beeb-5.local 16.7.0 Darwin Kernel Version 16.7.0: Thu Dec 13 21:36:55 PST 2018; root:xnu-3789.73.29~1/
     RELEASE_X86_64 x86_64

gpg (GnuPG/MacGPG2) 2.2.10
libgcrypt 1.8.3
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Home: /Users/dirkx/.gnupg
Supported algorithms:
Pubkey: RSA, ELG, DSA, ECDH, ECDSA, EDDSA
Cipher: IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH,
        CAMELLIA128, CAMELLIA192, CAMELLIA256
Hash: SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
Compression: Uncompressed, ZIP, ZLIB, BZIP2

OpenSSL 1.0.2p  14 Aug 2018

qrencode version 4.0.0
Copyright (C) 2006-2017 Kentaro Fukuchi

quirc
Copyright (C) 2010-2012 Daniel Beer <dlbeer@gmail.com>
Library version: 1.0
```

Private key 83224

Fri Feb 1 16:46:11 UTC 2019

This document allows for the reconstruction of a private key (secret key).
The path is as follows:

1. Ensure that you have a clean machine; and that your pgp, gpg or gnupg supports "ECDSA" as a pubkey (Use "gpg –version" to check).
2. Scan in the private key on page 8.
3. Using a QR decoder; extract the binary data from it[1].

   If that does not work - enter the keys manaully (provided on the next pages).
4. (optional) Check if the entered/extracted key is correct using the SHA256 listed on each page.
5. Import the extracted secret key with "gpg2 –homedir . –import qr-code-extrracted.binary".

You can now use the keyring created in the directory specified with "–homedir" to decrypt files.

Note that this secret key is *not* protected by a password.

[1]Note that a fair number of qrdecoders on the internet are 'broken' due to careless handling of UTF8 and binary data. See page 2 for a tested alternative.

Private key 83224 - Fri Feb 1 16:46:11 UTC 2019

Sha 256 of private key

```
1a417ab1 d13c3d7d cb41523e 0ecab749 329e5381 f540f533 6f013842 7487b9c0
```

Private key 83224 - Fri Feb 1 16:46:11 UTC 2019

```
-----BEGIN PGP PRIVATE KEY BLOCK-----

lFgEXFR30hYJKwYBBAHaRw8BAQdAc9m+6lvt7oz0MrKdPpG05vki/ciM8UfqQRkT
cxDftHQAAP4+UTqSB89RYAIA7nsIec88laI6n3q4ai2JJQuUriTqCQ3CtAlrZXkt
ODMyMjSIlgQTFggAPhYhBB2/4K+SduG4w7JqPpk9Zw5vjeQ4BQJcVHfSAhsDBQkD
wmcABQsJCAcCBhUKCQgLAgQWAgMBAh4BAheAAAoJEJk9Zw5vjeQ4JlEA/1Tyjevn
pW/2a2GmLw/risTVNj5ZgyR6zPsb8kGZVHBMAP9Mw8ExHB0jSTnsw11KTJzgkKfS
9xUjq+4dveitdDCDDJxdBFxUd9ISCisGAQQBl1UBBQEBB0BozRtDfCZcIoKIfHuY
YXmljiXePaKKaf7CbhmWdWIrLgMBCAcAAP9bWYYVjfbW+RQE0enoY60Py8URu1HX
+B/0xC3xtZTjSBM4iHgEGBYIACAWIQQdv+CvknbhuM0yaj6ZPWc0b43k0AUCXFR3
0gIbDAAKCRCZPWc0b43k0A0lAQDRcL540U2SY1bsw07KXH8sGbX10QS7nU6oDfMu
HLNnhgEA2lawgCarR40V6L+4JTcQV6xD1c0XasyDvdnXqSz4awc=
=YWmG
-----END PGP PRIVATE KEY BLOCK-----
```

Sha 256 of the ASCII armoured private key including headers

```
ff520069 1574bbe5 d3ee4715 da33adac 4341f7e5 31951f19 9212e0cd 3a5c1337
```

Sha 256 of private key

```
1a417ab1 d13c3d7d cb41523e 0ecab749 329e5381 f540f533 6f013842 7487b9c0
```

Private key 83224 - Fri Feb 1 16:46:11 UTC 2019

```
0000000 94 58 04 5c 54 77 d2 16 09 2b 06 01 04 01 da 47
0000010 0f 01 01 07 40 73 d9 be ea 5b ed ee 8c f4 32 b2
0000020 9d 3e 91 b4 e6 f9 22 fd c8 8c f1 47 ea 41 19 13
0000030 73 10 df b4 74 00 00 fe 3e 51 3a 92 07 cf 51 60
0000040 02 00 ee 7b 08 79 cf 3c 95 a2 3a 9f 7a b8 6a 2d
0000050 89 25 0b 94 ae 24 ea 09 0d c2 b4 09 6b 65 79 2d
0000060 38 33 32 32 34 88 96 04 13 16 08 00 3e 16 21 04
0000070 1d bf e0 af 92 76 e1 b8 c3 b2 6a 3e 99 3d 67 0e
0000080 6f 8d e4 38 05 02 5c 54 77 d2 02 1b 03 05 09 03
0000090 c2 67 00 05 0b 09 08 07 02 06 15 0a 09 08 0b 02
00000a0 04 16 02 03 01 02 1e 01 02 17 80 00 0a 09 10 99
00000b0 3d 67 0e 6f 8d e4 38 26 51 00 ff 54 f2 8d eb e7
00000c0 a5 6f f6 6b 61 a6 2f 0f eb 8a c4 d5 36 3e 59 83
00000d0 24 7a cc fb 1b f2 41 99 54 70 4c 00 ff 4c c3 c1
00000e0 31 1c 13 a3 49 39 ec c3 5d 4a 4c 9c e0 90 a7 d2
00000f0 f7 15 23 ab ee 1d bd e8 ad 74 30 83 0c 9c 5d 04
0000100 5c 54 77 d2 12 0a 2b 06 01 04 01 97 55 01 05 01
0000110 01 07 40 68 cd 1b 43 7c 26 5c 22 82 88 7c 7b 98
0000120 61 79 a5 8e 25 de 3d a2 8a 69 fe c2 6e 19 96 75
0000130 62 2b 2e 03 01 08 07 00 00 ff 5b 59 86 15 8d f6
0000140 d6 f9 14 04 d1 e9 e8 63 ad 0f cb c5 11 bb 51 d7
0000150 f8 1f ce c4 2d f1 b5 94 e3 48 13 38 88 78 04 18
0000160 16 08 00 20 16 21 04 1d bf e0 af 92 76 e1 b8 c3
0000170 b2 6a 3e 99 3d 67 0e 6f 8d e4 38 05 02 5c 54 77
0000180 d2 02 1b 0c 00 0a 09 10 99 3d 67 0e 6f 8d e4 38
0000190 0d 25 01 00 d1 70 be 78 d1 4d 92 63 56 ec c3 4e
00001a0 ca 5c 7f 2c 19 b5 f5 39 04 bb 9d 4e a8 0d f3 2e
00001b0 1c b3 67 86 01 00 da 56 b0 80 26 ab 47 8d 15 e8
00001c0 bf b8 25 37 10 57 ac 43 d5 c3 97 6a cc 83 bd d9
00001d0 d7 a9 2c f8 6b 07
00001d6
```

Sha 256 of private key

```
1a417ab1 d13c3d7d cb41523e 0ecab749 329e5381 f540f533 6f013842 7487b9c0
```