Trolling the Web of Trust

Micah Lee
Staff Technologist
Electronic Frontier Foundation

```
micah@spock:~$ gpg --fingerprint 99999697
pub   4096R/99999697 2011-06-24
      Key fingerprint = 5C17 6163 61BD 9F92 422A C08B B4D2 5A1E 9999 9697
uid                  Micah Lee <micahflee@riseup.net>
uid                  Micah Lee <micah@eff.org>
uid                  Micah Lee <micahflee@gmail.com>
uid                  Micah Lee <micah@pressfreedomfoundation.org>
sub   4096R/E8839F99 2011-06-24
```

Twitter: @micahflee

Terminology

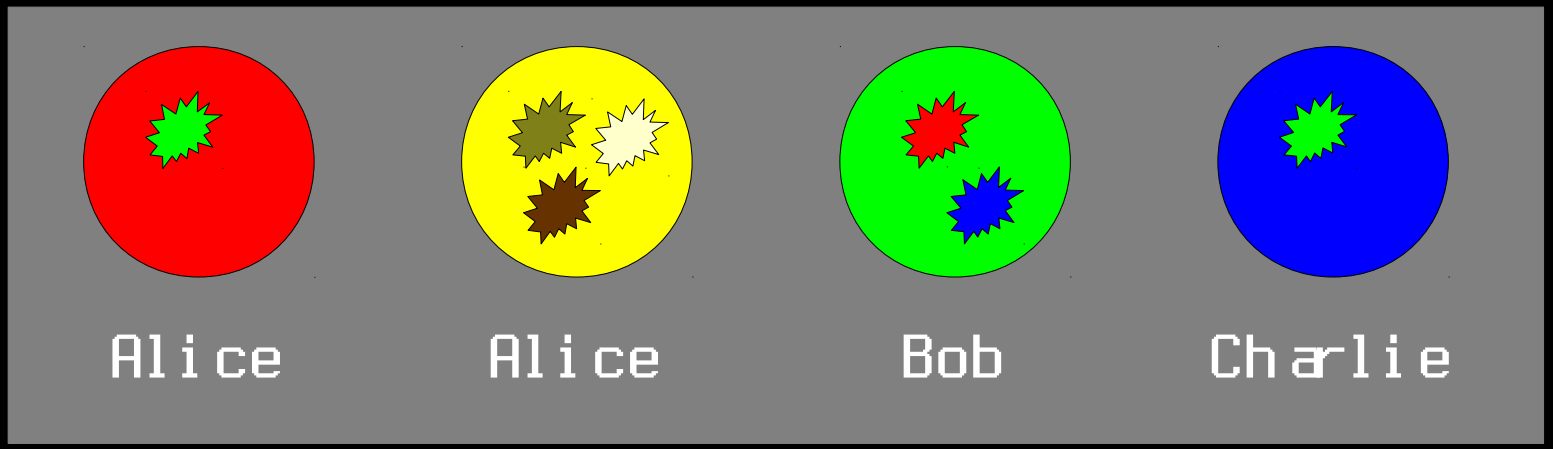
- **OpenPGP**: open standard for email encryption that various pieces of software implement
- **PGP**: proprietary software that implements the OpenPGP standard
- **GnuPG (GPG)**: free software (and much more popular) that implements the OpenPGP standard
- **OpenPGP key, PGP key, GPG key**: used interchangably to describe an OpenPGP keypair, or just a public key

A brief introduction to OpenPGP

- Each OpenPGP user needs their own key pair, split into public and secret keys
- Use someone's public key to:
  - Encrypt a message to them
  - Verify a signature that their secret key generated
- Use your secret key to:
  - Decrypt messages that were encrypted with your public key
  - Digitally sign messages and keys

A brief introduction to key servers and the Web of Trust

KEY SERVER



Alice and Bob have signed keys
Bob and Charlie have signed keys

Charlie needs to talk to Alice, but two keys are called 'Alice'! Which key is correct?

The Web of Trust is Crawling With Lies

Here, let me show you...

The Web of Trust is Crawling With Lies

Some of the things that you can't trust:

- Name and email address part of the key (e.g.: Micah Lee <micahflee@riseup.net>)
- Signatures on a key (if you haven't manually confirmed the signer's fingerprint)
  - If one email address has two keys, and one key is signed by 50 people, the other by 5, which is more trustworthy?
- Timestamps

What can you trust?

If:

- You have manually verified someone's fingerprint
  - In person, over the phone if you recognize their voice, or through a very trusted 3rd party that you are verified with
- And you're pretty sure you did it right

Then:

- That public key belongs to the person you think it belongs to
- Other keys they have signed probably were actually signed by them

What can you trust?

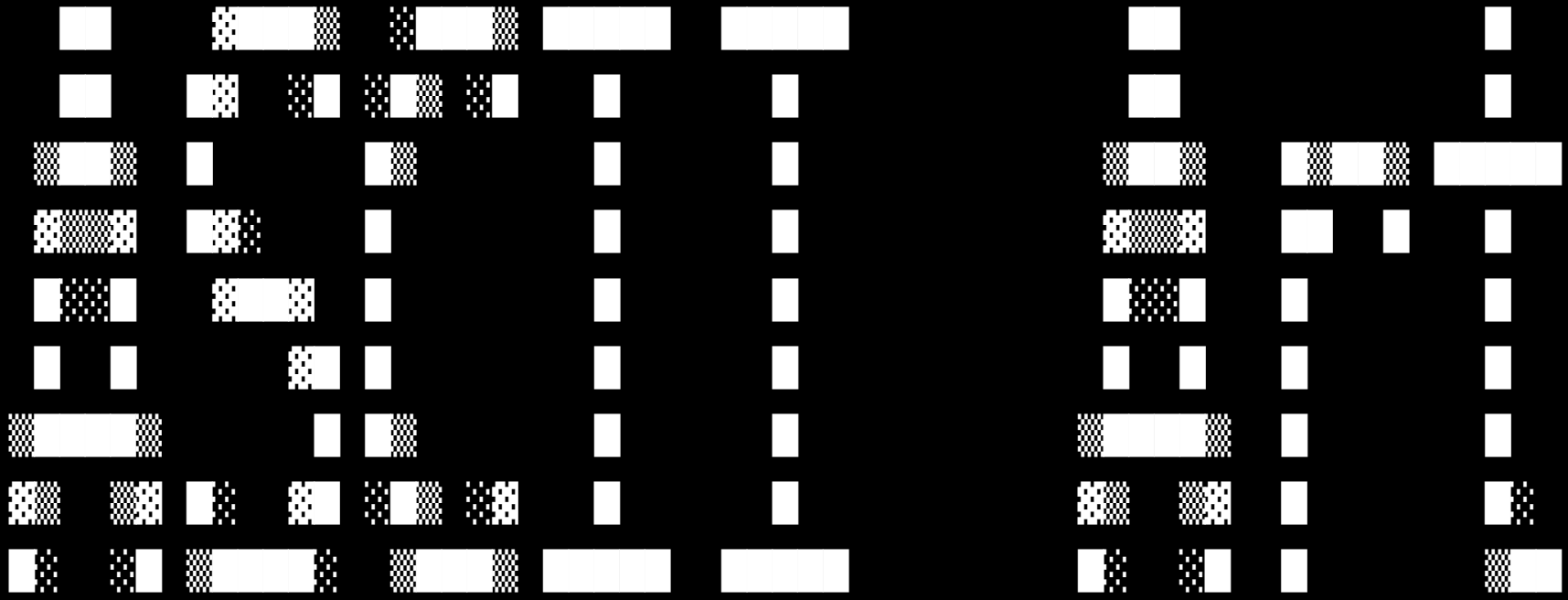
However:

- You can't be certain their key hasn't been compromised
- You can't trust other signatures on *their* key

Some Code to Play With

https://github.com/micahflee/trollwot

Bringing Art to Key Servers
(pleæse paint responsibly)



Introducing ascii_sign

https://github.com/micahflee/trollwot/

Usage:

```
./ascii_sign [ASCII_ART_FILENAME]
[KEYID]
```

ASCII Signing

Inspired by ASCII Goatse

I thought better of including it in these slides. But if you'd really like to see it, I made a convenient short URL:

http://bit.ly/ascii-goatse

Search results for '0x5c17616361bd9f92422ac08bb4d25a1e99999697'

Type bits/keyID cr. time exp time key expir

---

pub 4096R/99999697 2011-06-24

uid Micah Lee <micahflee@riseup.net>
sig sig3 99999697 2012-02-01 _____ _____ [selfsig]
sig sig 135EA668 2012-03-20 _____ _____ Richard Stallman (Chief GNUisance) <rms@gnu.org>
sig sig 9C7DD150 2012-03-30 _____ _____ Seth David Schoen <schoen@loyalty.org>
sig sig3 5C5C7343 2012-03-30 _____ _____ David Grant <starchy@eff.org>
sig sig3 1FC237AF 2012-04-26 _____ _____ Cory Doctorow (Canonical key as of Thu, April 17 2008) <doctorow@craphound.com>
sig sig F4CEF0A5 2012-05-18 _____ _____ mark burdett <mark@indymedia.org>
sig sig3 A12DB343 2012-06-12 _____ _____ cooperq <cooperq@gmail.com>
sig sig3 F0AFE2CA 2012-06-12 _____ _____ Cooper Quintin <cooperq@garlic.is>
sig sig3 D3366755 2012-06-13 _____ _____ William Budington <bill@inputoutput.io>
sig sig3 8ED38319 2012-06-20 _____ _____ Jillian C. York (Jillian @ EFF) <jillian@eff.org>
sig sig 5F2E4935 2012-07-14 _____ _____ Jamie McClelland <jamie@mayfirst.org>
sig sig CB2D0500 2012-07-16 _____ _____ Nat Meysenburg <nat@openflows.com>
sig sig3 2F37ACED 2012-07-20 _____ _____ Maira Sutton <maira@eff.org>
sig sig3 9CA55A17 2012-07-23 _____ _____ Adi Kamdar <adi@eff.org>
sig sig 21DBEFD4 2012-09-25 _____ _____ Rejo Zenger <rejo@zenger.nl>
sig sig 6B6B5983 2012-11-11 _____ _____ William Gillis <rechelon@riseup.net>
sig sig3 4FC5A760 2012-11-27 _____ _____ Trevor Ellermann <trevor@ellermann.net>
sig sig D67C4D21 2012-11-28 _____ _____ Peter Eckersley <pde@eff.org>
sig sig A3FDE45E 2012-11-28 _____ _____ Danny O'Brien <danny@eff.org>
sig sig3 2CDB8B35 2012-11-28 _____ _____ Isis! <isis@patternsinthevoid.net>
sig sig3 F2D52E48 2012-11-28 _____ _____ Sacha van Geffen <sacha@greenhost.nl>
sig sig3 D8CF8028 2012-11-28 _____ _____ Bahaa Nasr <bahaan@iwpr.net>
sig sig 64E9FFD3 2012-12-21 _____ _____ Michael Schade <michael@stripe.com>
sig sig3 80AF07D3 2013-02-02 _____ _____ Flamsmark <flamsmark@gmail.com>
sig sig2 C59578AD 2013-02-03 2017-02-03 _____ Patrick Ball <pball@hrdag.org>
sig sig C11F6276 2013-04-13 _____ _____ David Fifield <david@bamsoftware.com>
sig sig 63FEE659 2013-05-27 _____ _____ Erinn Clark <erinn@debian.org>
sig sig3 BF306743 2013-06-14 2017-06-14 _____ Richard Esguerra <richard@eff.org>
sig sig C9A30854 2013-07-21 _____ _____
sig sig F7F253D8 2013-07-21 _____ _____
sig sig 5FFDBF8A 2013-07-21 _____ _____
sig sig 17A750F2 2013-07-21 _____ _____
sig sig 524FB9A7 2013-07-21 _____ _____
sig sig 0C5A841A 2013-07-21 _____ _____

uid Micah Lee <micah@eff.org>
sig sig3 99999697 2011-06-24 _____ _____ [selfsig]
sig sig2 4CCD75E3 2011-07-09 _____ _____ Garrett Robinson <garrett.f.robinson@gmail.com>
sig sig3 8BF8DD0B 2011-07-29 _____ _____ Leez Wright <leez@eff.org>

Search results for '0x9956d611'

Type bits/keyID cr. time exp time key expir

---

pub 1024R/9956D611 2013-07-21

uid Trolling the Web of Trust

sig sig3 9956D611 2013-07-21 _____ _____ [selfsig]

sig sig 95EFF6CE 2013-07-21 _____ _____ .....

sig sig C62763A5 2013-07-21 _____ _____ ..... 777777777777777.....

sig sig 45ADBBA4 2013-07-21 _____ _____ ..... 77777~~~~~~~~~~~~~77777 .....

sig sig BFCCC274 2013-07-21 _____ _____ .....?7777~~~~~~~~~~~~~~~~~~~~~777 .....

sig sig 4DB5A261 2013-07-21 _____ _____ .....7777~~~~~~~~~~~~~~~~~~~~~~~~:777.....

sig sig 9343A480 2013-07-21 _____ _____ .....777~~~~~~~~~~~~~~~~~~~~~~~~~~~~~777.....

sig sig CC2ED846 2013-07-21 _____ _____ .77777777777777777777777777777~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~77 .....

sig sig C7F85972 2013-07-21 _____ _____ .77.....~~~~~+77777~~~~~~~~~~~~~~~777.....

sig sig 6B013A94 2013-07-21 _____ _____ .77.....~~~~~+77777~~~~~~~~~~~~~~~~77 .....

sig sig 8A7A766C 2013-07-21 _____ _____ .77.....~~~~~+77~~~~~~~~~~~~~~~~~~77 .....

sig sig 310556D9 2013-07-21 _____ _____ .77.....~~~~~+77~~~~~~~~~~~~~~~~~~77 .....

sig sig 94BF3A05 2013-07-21 _____ _____ .77.....777777 7 77~~~~~~~~~~~~~~~~~+7777~~~~~~~~~~~~~~~~~~~7 .....

sig sig B15BF7C9 2013-07-21 _____ _____ .77.....77.....77~~~~~~~~~~~~~~~~~+7777~~~~~~~~~~~~~~~~~~+7 ...

sig sig 21618B27 2013-07-21 _____ _____ .77.....77.....777~~~~~~~~~~~~~~~~~+77~~~~~~~~~~~~~~~~~~~77...

sig sig 4A58BA85 2013-07-21 _____ _____ .77.....77777777 ~~~~~~~~~~~~~~~~~+77~~~~~~~~~~~~~~~~~~~77 ..

sig sig 78BE9390 2013-07-21 _____ _____ .77.....~~~~~~+77~~~~~~~~~~~~~~~~~~~77..

sig sig B4409156 2013-07-21 _____ _____ .77.....~~~~~~~~~~~~~~~~~~~~~~~~~~~~77..

sig sig 89404F8F 2013-07-21 _____ _____ .77.....~~~~~~~~~~~~~~~~~~~~~~~~~~~~77..

sig sig 4D4BB488 2013-07-21 _____ _____ .77.....~~~~~~~~~~~~~~~~~~~~~~~~~~~~7 ..

sig sig 9DE3D6F3 2013-07-21 _____ _____ .77.....777777777~~~~~~~~~~~~~~~~~+77777~~~~~~~~~~~~~~~~~~7 ..

sig sig EEBCFC4B 2013-07-21 _____ _____ .77.....77..... 7~~~~~~~~~~~~~~~~~+77~~~~~~~~~~~~~~~~~~~777..

sig sig B627ACEB 2013-07-21 _____ _____ .77.....77.....777~~~~~~~~~~~~~~~~~+77~~~~~~~~~~~~~~~~~~~7 ...

sig sig 06A691F8 2013-07-21 _____ _____ .77.....77..... 7~~~~~~~~~~~~~~~~~+77~~~~~~~~~~~~~~~~~~~~=7 ...

sig sig 0CF7D10E 2013-07-21 _____ _____ .77.....77777777 7~~~~~~~~~~~~~~~~~+7 77~~~~~~~~~~~~~~~~~~~7 ....

sig sig 09EDD9E4 2013-07-21 _____ _____ .77.....~~~~~+77~~~~~~~~~~~~~~~~~~~777....

sig sig 566106EA 2013-07-21 _____ _____ .77.....~~~~~+77~~~~~~~~~~~~~~~~~~77 .....

sig sig 670F0CF4 2013-07-21 _____ _____ .77.....~~~~~+77~~~~~~~~~~~~~~~~~~77 .....

sig sig 1DA8056A 2013-07-21 _____ _____ .77.....~~~~~+77~~~~~~~~~~~~~~~~~~777.....

sig sig F89FD656 2013-07-21 _____ _____ .77777777777777777777777777777~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~77 .....

sig sig 0A74104C 2013-07-21 _____ _____ .....~777~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~77 .....

sig sig DDC90DE2 2013-07-21 _____ _____ .....7777~~~~~~~~~~~~~~~~~~~~~~~~~~~~~777,.....

sig sig 626F7469 2013-07-21 _____ _____ .....7777?~~~~~~~~~~~~~~~~~~~~~~~~~7777,.....

sig sig 20D9380A 2013-07-21 _____ _____ ..... 777 7~~~~~~~~~~~~~~777777.....

sig sig 775FE52E 2013-07-21 _____ _____ .....,777777 7777777777.....

sig sig 8FC7A497 2013-07-21 _____ _____ .....

sig sig D944A73F 2013-07-21 _____ _____ .....

sub 1024R/2FE0387B 2013-07-21

Search results for '0xbc0fa9c0'

```
Type bits/keyID     cr. time   exp time   key expir
----------------------------------------------------

pub  4096R/BC0FA9C0 2013-07-21

uid Trolling the Web of Trust <troll1@localhost>
sig  sig3  BC0FA9C0 2013-07-21 __________ __________ [selfsig]
sig  sig   E5FBD04D 2013-07-21 __________ __________ ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
sig  sig   D3C5A658 2013-07-21 __________ __________ :::::::::::::::::::::::::::::::::::::,:::::::::::::,:::::::::::,,,,::,:,:::,:::,,,
sig  sig   15476588 2013-07-21 __________ __________ ::::::::::::,,,:,,,,,,,,,,,,,,,,,,,,,,,,,,:,,,,:,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
sig  sig   0E86E3E2 2013-07-21 __________ __________ ,,:,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
sig  sig   4219C297 2013-07-21 __________ __________ ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,$I,:$I$7$+:,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
sig  sig   766EA58C 2013-07-21 __________ __________ ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,I=~Z8D8800+Z+::,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
sig  sig   0C2C2C35 2013-07-21 __________ __________ ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,::00888800D88D80$,:,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
sig  sig   16BA6347 2013-07-21 __________ __________ ,,,,,,,,,,,,,,,,,,,,,,,,,,,:$0080800888888D0807Z:,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
sig  sig   100EFF19 2013-07-21 __________ __________ ,,,,,,,,,,,,,,,,,,,,,,,,,,?888880ZI77Z08D888DZ007=.,.........................
sig  sig   84EA6F19 2013-07-21 __________ __________ ,,........................,=Z888$I+?$ZZ0887$0880$=~:.........................
sig  sig   F18C302D 2013-07-21 __________ __________ ...........................0888$+++==++70$Z$ZZZ88$,.........................
sig  sig   48D9B756 2013-07-21 __________ __________ ..........................,8Z80I==========++?7088Z,.........................
sig  sig   52BB3A2D 2013-07-21 __________ __________ .........................+Z800?+========+++?IZ080,..........................
sig  sig   D468549A 2013-07-21 __________ __________ .........................$$8DZ+7$Z7$+===+=+?7088~...........................
sig  sig   39899DEF 2013-07-21 __________ __________ ..........................7?87$Z?Z0Z0I+Z00I?Z880............................
sig  sig   20F58BF1 2013-07-21 __________ __________ ..........................=~$++==+?=~++I+?IOZ80............................
sig  sig   6BDFF797 2013-07-21 __________ __________ ...........................~=+==~7$Z~=I8+==?IO.............................
sig  sig   0F7E86CD 2013-07-21 __________ __________ ...........................~:7===+=+=~?I+==+IZ.............................
sig  sig   79A4D700 2013-07-21 __________ __________ ............................~$?+I?IIIZ07??+?7..............................
sig  sig   C4712038 2013-07-21 __________ __________ .............................7I+??+??II7I?+7~..............................
sig  sig   B7B777A9 2013-07-21 __________ __________ .............................$7?I=??I+$????$...............................
sig  sig   407C5748 2013-07-21 __________ __________ ..............................$$7I=?++?II7$=...............................
sig  sig   C0333F76 2013-07-21 __________ __________ ..............................?0$$I?+??7$87................................
sig  sig   096B9327 2013-07-21 __________ __________ ........................=.7887I?=087?I?7Z80D088............................
sig  sig   22E51D8B 2013-07-21 __________ __________ .......................88888?+$+=+I0D8D8D877~888880.........................
sig  sig   11FA5447 2013-07-21 __________ __________ ,,,...............:,,,,,Z88888$$~+==+=++?I?$,ZDDDD88Z.......................
sig  sig   08A24A18 2013-07-21 __________ __________ :~~:~:::~::~::~~~~~~~~:88DD88DD$N+?===+??+$$=8DD8DDDDDD:,,...................,:::
sig  sig   02C69309 2013-07-21 __________ __________ ~~~~=~~~~~~~~~~~:~~~888DDDDDDDDDDDDZ7.,,,ZDDDDDDDDDDDDDDDDDD?,:::::::::::::::::::~
sig  sig   E90E489A 2013-07-21 __________ __________ ~~~~~~~~~~~~~~~~~~DDDDDDDDDDDDDNDDDNDD?=0DNDDDDDDDDDDDDDDDDDDDDD~~:::~~:~~~~~:~~~~~
sig  sig   352978A9 2013-07-21 __________ __________ ??????????II?I?DDDDDDDDDDDDDDDDDDDDDDD8DDDDDDDDDDDDDDDDDDDDDDDDD??????+?++++++?+++
sig  sig   CF0DE27F 2013-07-21 __________ __________ ++++++++++++++DDDDDDDDDDDDDDNDDDDDDD0DDDDDDDDDDDDDNDDDDDDDDDD8=~====~===~===~==
sig  sig   48128D3A 2013-07-21 __________ __________ +++++=++===++DDDDDDDDDDDDDNDDDDDDDDD8DDDDD$DZ=,7$=770DDDDDDDDD====~=====+++==+=
sig  sig   3EA39C56 2013-07-21 __________ __________ ==+==+=?+===ZDDDDDDDDDDDDDDDDDNNDNNNDDNDNDD?DNDD80?I$0NNNNNNNDDD=+~~===+++===+=+
sig  sig   67E9685B 2013-07-21 __________ __________ ++++=+++====NDDDDDDDDDDDNNDDDDNNDNNNN8NDNND?DDDDD0088NNNNNNNNNN+==:=:~::~~~~~~=
sig  sig   269B774B 2013-07-21 __________ __________ ==~~======~DDDDNDDDDDDDDDNNNNNNNNNNNN8NNNNDDDNDNN?008$NNNNNNNNNNI=~~~=~~::::::~~=
sig  sig   9AFBA737 2013-07-21 __________ __________ ==~=+==+=~=DDDDDDDDNDDDNNNNNNNNNNNNNNNNNNNNDZDNND8NNNNNNNNNNNNNNN~::~~~~:~~:~:~~
sig  sig   87FA153F 2013-07-21 __________ __________ =~=++=+=~:?DDDDDDDDNNNDDNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN=~~~~~:~=:=~~~~
sig  sig   69304FD5 2013-07-21 __________ __________ =~+~==?+~=DNDDNNDDDNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNO~:::=++==~~===
sig  sig   DBB83C37 2013-07-21 __________ __________ =+==~==+=+DNDNNNDDDNNNNNNNNNNNNNNNNNNNNNNNNNNNMMNNMMNNNNNNNNNNNNNNN===+I+++=+==?~
sig  sig   D6019B7D 2013-07-21 __________ __________ ===++++==?DNNNNNNDNNNNNNNNNNNNNNNNNNNNNNNNNNNNMMMMMMMNNNNMNNNNNNNNNNN:~~~:::====+==
```

```
gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:  19  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 19u
pub   4096R/9DE875A0 2013-07-28
uid                  Trolling the Web of Trust
sig 3        9DE875A0 2013-07-28  Trolling the Web of Trust
sig          81B3B5B9 2013-07-28
sig          4E9F442B 2013-07-28
sig          928507EF 2013-07-28
sig          66B7B5A5 2013-07-28
sig          AA25AB50 2013-07-28
sig          8FCFF27E 2013-07-28
sig          E1FEACCC 2013-07-28
sig          95052A83 2013-07-28
sig          FC2A81AF 2013-07-28
sig          B3F8B3B2 2013-07-28
sig          6634D278 2013-07-28
sig          A53E2C33 2013-07-28
sig          28849595 2013-07-28
sig          C56ABEF2 2013-07-28
sig          C9FC781F 2013-07-28
sig          DC389C69 2013-07-28
sig          AF92F6FD 2013-07-28
sig          F96A6E79 2013-07-28
sub   4096R/037F5DA2 2013-07-28
sig          9DE875A0 2013-07-28  Trolling the Web of Trust

Does this look good? Type "YES" to upload, "no" to quit: YES
```

Fake Signatures

- As you already saw, "Barack Obama" signed my key...
- I could make the entire Obama administration sign "Barack Obama"'s key
- Would "Osama bin Laden" sign NSA Director "Keith Alexander"'s key?
- And would "Keith Alexander" sign "bin Laden"'s key back?
- Nothing is stopping you from signing the goatse key
- Nothing is stopping "Barack Obama" from signing the goatse key

NOTHING STOPS ANYONE FROM MAKING ANY KEY AND USING IT TO SIGN ANY OTHER KEY, AND UPLOADING EVERYTHING TO THE WEB OF TRUST

Introducing fake_sign

https://github.com/micahflee/trollwot/

Usage:

```
./fake_sign [NAME] [EMAIL] [KEYID]
```

What else can go inside of keys?

Any arbitrary text, so...

- Wikileaks cables and other leaked documents
- Credit card numbers, passwords, other personal information (but don't do this, it's mean)
- DRM secret keys
- ANY OTHER INFORMATION THAT WANTS TO BE FREE

If any of this ends up on a key server,
it gets synced to all the other key servers.

Timestamps Can't be Trusted

You can set back your computer's time
before generating a key or signature

Or you can...

added optional unix timestamp creation date question

Browse code

master

micahflee authored a day ago 1 parent 583df93 commit cb8f9a32f6fab5732c81c9c6c5570128def4be01

Showing **1 changed file** with **10 additions** and **1 deletion**.

Show Diff Stats

11 g10/keygen.c

View file @ cb8f9a3

```
...  ...  @@ -3109,7 +3109,16 @@ static int gen_card_key_with_backup (int algo, int keyno, int is_primary,
3109 3109       start_tree(&pub_root);
3110 3110       start_tree(&sec_root);
3111 3111
3112      -     timestamp = get_parameter_u32 (para, pKEYCREATIONDATE);
     3112 +     // trollwot: add optional unix timestamp question
     3113 +     if(!opt.batch) {
     3114 +       char* str_timestamp;
     3115 +       str_timestamp = cpr_get_no_help("", "Creation Timestamp (blank for now): ");
     3116 +       if(strcmp(str_timestamp, "") == 0) {
     3117 +         timestamp = get_parameter_u32 (para, pKEYCREATIONDATE);
     3118 +       } else {
     3119 +         timestamp = atoi(str_timestamp);
     3120 +       }
     3121 +     }
3113 3122
3114 3123       /* Note that, depending on the backend (i.e. the used scdaemon
3115 3124          version or the internal code), the card key generation may
```

Edward Snowden's public key

Our Top-Secret Message to NSA Whistleblower Edward Snowden

BY KEVIN POULSEN 06.14.13 3:58 PM

Follow @kpoulsen

Don't read this if you aren't him.

```
-----BEGIN PGP MESSAGE-----
Version: GnuPG/MacGPG2 v2.0.19 (Darwin)
Comment: GPGTools - http://gpgtools.org

hQIMA1tQlAt53r41ARAAsCwY0VverVliy5i29NafjAEhFpmwDDAHVdzOYtnGbOHL
Hi1t1hgRPe5NBD+AnDENmUbJf4hNxH88Uh4qTqy8ja4qAWyRSJXENijZs2Pjhv+8
ovJhDSDK3N8bGDcM7XS7o1FGrLJtpV2CqP4DP4rSr4fcQz1ZnRWrnBP9XI6FAbEp
XXRtW6mbtPWTLfgvn91Ka3aJGegXl6rFYeqmXgmZiPYrnmNSAgFGSKg+Er2Kz+jE
sl4tS/hqP9vhAAWWCOvT7U5LMuDGjawsBXjHTPA9FokPO7euxRPxMraz5FmrtZYb
erFhkMlW5IV5zG1BEO5TetyM66hAZid/QwdFzlDW3wHQoYJdJWcZEYY0tGWbL3+h
```

CRYPTOME

7 July 2013

Edward Snowden, Laura Poitras, Jacob Appelbaum

About the time Laura Poitras and Jacob Appelbaum claim in Der Spiegel to have communicated by encrypted emails with Edward Snowden in "mid-May" the following PGP keys were generated ("Verax" is allegedly a Snowden pseudonym):

Search results for 'verax informed front democracy'

```
Type bits/keyID     Date       User ID
```

```
pub  4096R/2BE0BC29 2013-05-20 Verax (Informed Democracy Front)
        Fingerprint=5091 7466 B18F 35B3 F644  F700 1D0D 97F2 2BE0 BC29

From PGPdump Interface:

Public key creation time - Mon May 20 17:41:14 UTC 2013
```

```
pub  4096R/C920FAA6 2013-05-20 Verax (Informed Democracy Front)
        Fingerprint=AC5E 06C5 17D0 A8C1 75D3  17F5 53B9 0192 C920 FAA6

From PGPdump Interface:

Public key creation time - Mon May 20 21:41:34 UTC 2013
```

```
pub  4096R/E87C2665 2013-05-20 Verax (Informed Democracy Front)
        Fingerprint=7F99 43F6 5CC9 BAD1 92A9  8DF8 96E6 0F93 E87C 2665

From PGPdump Interface:

Public key creation time - Mon May 20 21:41:46 UTC 2013
```

Verax (Informed Democracy Front)

```
micah@spock:~/projects/trollwot/trollwot$ gpg --homedir homedir_verax --list-keys
--fingerprint
homedir_verax/pubring.gpg
-------------------------
pub   4096R/71A3AA96 2013-05-20
      Key fingerprint = 2B5D D0BF F454 8592 1FAF  22FB 4569 3580 71A3 AA96
uid                  Verax (Informed Democracy Front)
sub   4096R/9E06D59D 2013-05-20

pub   4096R/0E8CD2B6 2013-05-20
      Key fingerprint = F606 1774 A693 72A1 8AD0  1CD7 0C4D AF57 0E8C D2B6
uid                  Verax (Informed Democracy Front)
sub   4096R/DCF43355 2013-05-20

pub   4096R/79B82638 2013-05-20
      Key fingerprint = 4ECC 0702 A2E9 5FA6 2074  C7BE 574F C888 79B8 2638
uid                  Verax (Informed Democracy Front)
sub   4096R/B124BA64 2013-05-20

pub   4096R/E87C2665 2013-05-20 [expires: 2013-08-18]
      Key fingerprint = 7F99 43F6 5CC9 BAD1 92A9  8DF8 96E6 0F93 E87C 2665
uid                  Verax (Informed Democracy Front)
sub   4096R/E02DE9EB 2013-05-20 [expires: 2013-08-18]
```

Verax (Informed Democracy Front)

```
pub   4096R/C920FAA6 2013-05-20
      Key fingerprint = AC5E 06C5 17D0 A8C1 75D3  17F5 53B9 0192 C920 FAA6
uid                  Verax (Informed Democracy Front)
sub   4096R/F48E7DE5 2013-05-20

pub   4096R/2BE0BC29 2013-05-20
      Key fingerprint = 5091 7466 B18F 35B3 F644  F700 1D0D 97F2 2BE0 BC29
uid                  Verax (Informed Democracy Front)
sub   4096R/79DEBE35 2013-05-20

pub   4096R/9DCA85F7 2013-05-19
      Key fingerprint = BDE4 AA86 8507 1371 7793  11A8 105D A7AB 9DCA 85F7
uid                  Verax (Informed Democracy Front)
sub   4096R/4FEB7EDC 2013-05-19

pub   4096R/BE452B27 2013-05-13
      Key fingerprint = 134D 970C 5872 5AA6 8F2A  BD75 D18D FE89 BE45 2B27
uid                  Verax (Informed Democracy Front)
sub   4096R/A22F0C5D 2013-05-13
```

Here's the Verax (Informed Democracy Front)
key that's from one week before all the others

TIMESTAMPS CAN'T BE TRUSTED


```
-----BEGIN PGP PRIVATE KEY BLOCK-----
Version: GnuPG v1.4.13 (GNU/Linux)

lQcYBFGRXlUBEADJqo7F0vibWC8WbHyeH3K0q8wDhmPq1DiB4Bp8+tUox9LAeURe
im54igzSqjR1zYIyl46Z3LdyZ5bXflkC8qJMlW6Cqv864NTjPlgauDdFPQa2H9x1
G2FeHGejdp7Bt728xcyOvLI8Qukn8sBfT6L5FKNyHCkvSW8cH+7tFRqxuWbYK1tY
MbZDouj7zVJM+WBd+SUUEPsNDUrbaUnYhbkG/vYJmsK51+guAW14KU1nYjE03Pvv
c10gwdd7nuv5QAeVtrHsstzQNTyLpXsTazylm9J1JHDoB4vZ2zIsC051es27pGdY
o7oWamMPUxPkiDJ4MEJG77BGJK9TC+vUX67sqwqmkF5lNr/EcPaPzaE/bZIoWXsx
pnMDygFa2u2UPanyvnIVKLabCdgzKxGCzLxHfM75aGyYWvhXo3ySPUwgdCdzigTj
I8vC8AO7wuJFmueX8ozuXSV2w+CyA9/qKDsTZ7RxIh+Oj6yoNfSm5+A6znjWuakx
ptyl9o//3FRd9H4G/3D/6dRvsO/T4HYwPilinXjZ/mbgB4i73uwYgoVZbI6VQh7V
fhy5IrADnyDDzY/ESU2xcluJV93Yo5ek1XKXTop83bQjXMvt79XdEiZUZ3DWGvtL
wJ7zw85qbY5Mdmu7awCh7RYbf1EkoFBiLalGjasavXHgDm3hJCIEszS4MwARAQAB
AA//SdLU9xgXCA8NFcFbESTj+z2Uromo1BMiKHEYYT4VgBs+TFVYMN+uNxQaQ5gp
IiK8Ebcnba29vfSr5HENNqYJwc1GKjFcw3oAlLuC1y0ulnsTlOIu+vOt/hyZ194Z
mcKgg5VJH8t119eKsKQNxhZKC7KBbsleTkAsm3gZJ9L89ahwPJwKkUuJdFMTxPHR
ufnkTCJjb2gT7yA5kf6GGOxC4ou9F6D+vbeK+CRvpXhBLKJmbvqig3KQ2ef78PYp
XhIRdL3vfj2v1cxXQKuwVViBWAKl1VokttvkVzkFsdflvd3ZqGIzruAoxAHoB3kb
JCKzjwnjiUI/Hj5ZrCYGC7WHO9eQoSqtHZbU9e/6KNQ3r8pERtzDqMK2kcs4a2f0
XthenYRPxLnH5blzupTdeuIBehXqgn2cF8TQ26ia7qBwbmEpnTErAWCcUPUtm6kQ
gzaa62S+2AVbn19d5FvDUu0e/gZQBbabNr1CtP5OtN2JCeZV3aEhc6uPxfq4cjFY
mHivbaIBjCbPOWyw/Gfon9pbBX6GO8UDU0Q01bgbR8uVG/gPy1nqrTheh9evlRaF
mNn3wrSRM6oS0EGLQpMD47UL1dOkVDBUpgp3UfuerVYxj1fuNMZ+mYfYfPURbLYK
iip1ScokoqIBjVCYS3FXj8UFeUxeybhR6iVESI6vq6Zrh1kIAN1cOR6qbZE73jHf
bQ9RWFF/cQ0EM1cyTW4U5BSMPnFLVQAAAAXymV7w36F26YnGAlxi1VxVcDF/iBga
USGCiAaObPtUzkvKJ/WHt08jcrBbwJHsGdfrTny2R5JYupMrQRvBbWIhAaISL8Xj
QGhaDVWywTQDmKcrEnm9Em/LHW4hQhW9TVfv8VzabsZywYbKUY1GJCpIg6oHTsU7
Hd5GOB5OdURmPqXdE8fQ/iCq1xlD1BUPE0/pGHLLWaJJIoIKaB014xnB4VkZRvS6
Q97GzzOYzagV/DtNHJha+3Szjh24HT3vLscf3zUb6SNN0P4JMl/HCkiUpjY8mxyS
U6Hcx00IAO2EHwWrjllU2tnQZClHsmscKzlxsvT8b51izj/C7mRC+vluF2Gm5Dvx
9vEZGiW7L4DxZEtGoycxP4c3R/SE0HFamTIugylHAFgy6hPmTWoJuSPuUrLsO3mB
z75p4zUiJDZPBFHJ2Cl2CcAyfQAAAAAELVQPNRbVZwc//kZxsXMU4KQuPOEYzyua
+CUA/c9nLZMNzBs3W/Q/v77Mb40HOzOZAdl7sLW+Cags9mIZ8z1H91jmOuezLSoN
eA45UfIYaWrddCplqvhMOECDciMBY30X1zRGuQ9CXAY3hSFN5jgXguhrUg/3tx5N
Y+8chxXAYUQNjQZUou1j+MYzdiSyvX8IAISxpQ+WwU4rzleyFmZP4wYXuz/3Fg3r
WK7H9aKrUjBFsuK4NRhVtJ4STN7n6ejrIKYYpUPpJPWbmpPU7SHMmpNemdBcroZ+
UMT0sqbXp+lLVbkX3v6WgGcxA9TVbhI4Yxf8C4pJmTBE1kda2M+3LMxqM0ICcvt5
hhy/zUmmkS0o7rxjL2Z17qfp7ii65psKMiODcL4ZZeS6CITs3HUp4h4zFX7386Wx
6F7t2uww+DJ4+Z4Tz4tzhRicHDQmW+/DZmU0+YJdeHqDD3jsH5cD+vix9kTaKmad
N98zCoujHqpD0g7h7yaSU210rbRv/fPfKi4oNB71TJr0UsVtsra9icVmJbQgVmVy
YXggKE1uZm9ybWVkIERlbW9jcmFjeSBGcm9udCmJAjgEEwECACIFAlGRXlUCGwMG
CwkIBwMCBhUIAgkKCwQWAgMBAh4BAheAAAoJENGN/om+RSsnepMP/3pevPjmQli+
LAj7nltq6te0xjOP56WvGawEeOaloOLwDDPy8U35rJJMnjausKf7fwlGpzE21oKH
SYTSaHWflpseK7SCMMnNOF6jyQMdcWFeIplulJrIdgnx4Ch7Ag79XJTsoE3JeJBI
XnxRPtsaAdiSW1JCCtV1RggD+8J1y+gar35YlezbYNnzqsB7lwVDCSISwLyBILv+
DDK32Mb7fzxRV4DeTU/TXxAHp4sKQENtK0UlZMx1cSvjSHfz6KAlKK8a8qgtdvN1
B8thuD5vPlTe52ilko0t19Umot7tseB02/R0aXMTcFHzMMSyINb5S89xph2vjT5d
vpfyLHXIeCTXGgQ4nhtzH3ma/bWZZMheX46WK1uSSPDsO6HaRqG9OHLE9Cu3j/AO
dekTN6BF3wxPBjX8pt4xhqlW1LjBd+qRMJkHud1dmW1qS8QYaX+zbdts6Jr3KeOR
87XkGwk8Q9gCyFYS0EY7oduwjIm36Z4cjCinjTQNyAZE9c4FNovBBdRDYNqLZTdy
kLSBKpFgU7zYxbj0PToySakgwv2toU2dAlz3g+5DRooTMOVruvNF2pgE7z83bR7P
r/9u871daglAZSE9bsI/Ftk0JlBp6X7My6ypnkrl+AFg2i6OGjozgGwNpfVbOPtx
q2m0lRN44c/rCwUO7OOifHWQPlhLVfHjnQcYBFGRXlUBEADUKdY2RZK+jeTM/m9i
AtEvgftQEyj2AmilK47fKrhi7ns5va+1iqq1uxMYciCcS0eqj1gIR00mA9TYyDjS
+bpnEulyq9ccM2abFB1zP8o3lh4goa+7QarDR96ik/yKgF92qsoZJKuzY6SSkj6g
JYqiOofJ2owxCPiwQVAOdOJseqRElf2Ag9MIih77Zs9uv1uLTbuDWTDpZ2bI60y+
qXhcwtQi7X561BKMd1CO5SX7sX9Jhf4M0sGY+BZdtEaO0CjZtWPwJUnNqJnSgvZ2
SUDbP4QbeZXHvVTc+gv3A7M4QU/7TCnQikz5o+NOYePe9YxK/xcDhzBkyitttroQ
BVnwgV+22WoebDA/Qfswje5047KG2TqnHU5Ml1uvtmrFCxJR4bYZwP418fDVMWQR
qsvi4/fzHjKPqgNKQmwcOmRD7K2wUduw0ZD2Ma/YYNt0DQ81kqvUTF4g/Za4/vAf
LGb6ub8Um3VU5IH/raqixPRwNZUgPYJo+ioyHWjX4wTF4g8EC0w0yNBRZYDoD+8a
pPBkUwXnAmuN/Ay4INnFBeA2Kp/zbqJffKj/NT4qSGRnnmxVCZQUn0aLtMMi27w3
kFttj8IGerct72QByESR0harkGgswSAdj4GoZa1cvSev8lp1a7bKoEktNpC9y9oL
a/70v3HzQK9hFWz5ZbKAMWI66wARAQABAA/+JUwJhVbruiX1n/fmmsNGBOUDygrk
O4Mm5Epo/mBjM7ka3oazGeX7pqDxdW2yv+cX/Srf3eyy6kjoJQflvWj7VN1lPqxo
z2NV2VNK7hS7ttPkDf2+jxu3Kkc7IzeCwNJadeEGUC87I/kTWFTvP3X6DhTeBCUw
WfpqxYllynTPzSQuNoRroSkLc0fctHNZae2n6TBME4YdbBqqXg3HWwktyJjTdCRu
BKe9bwYWmlg8Oif06Xbtt1FhiUOrouNrN/XuCH+zxua3N7k4+WHec+LizOg4kB/q
rcdh4MrwSYdT/UK2DcZQ34FAkO43qnLY/TwLYq+/WvZsEnvb8Jib0u8t5SDQYSaf
izWqYBD1EvsxPTj65oTKgakzmdC20TlIVcFAWsoA2WtNpdlvYf1LELKPaJGCqVUY
bCYgreFUbg+cF5kWPbiGX+06TB7T2Fim0veZW8190zBJZyot46kshOJpQc0jb6HK
8eBqc6Z57daXrld85ykF0dziZCW5N4Vk8T2nKOE5lJeZHjoBMrK0A7S+HHASI46L
zleuVxqfLP4UBhBly/r9TUYME2avB/yqptXd85hF05fp8OCxWjlJAF+WNYaE6P0v
x6q45wXYQyZaSUMYS9FDA4X0djb/8R5oXzvXi8516nDoj13OU9CcYY33FANSxMkI
bg+cGt/mSqtMW6EIAOKgDhLnMOYLHQLgat6FZ3HiiO0c6UXcUy9+wX0mdy9n1vZa
dsWUqFnOF9EKjKvAPFb+YU/Cj443E3B6IaOvegk+UIdGDfX4BDkX43GQ/bt8/am5
G6qaDyoBv4tC+uFAZ/+E1T1VTsBVzwnaOMJR5mY8cy8jWr/qIXBXkF1eEz41QvDQ
LY12cEg8maUmJHm+SnafTBxsF2ckSvDtMUyWpxIxrA9+GQi/PNhBaE+IqnFN25dN
dOyx4nArWuxWGejUOvmm23Vkcc9IquWQN/ZQlREODN9yq6QcevAy1ngOkXQwAAAA
AMA8sDHQ4loXPWYMcxr8mlNAOut0m3lKKFzsazsIAO+p6AlbW3Fx6S1qAtoL1ERT
Ja5o7MyvTUfDSitHIBI8jjLZQsPLHngAAAAU9TCCCqOXDG5JOpW0vvvSBHqEXps
7ylw2f7EZn3TxRsJ8WMvQ0pCaQlMoCJwx/dCy02GbUFYRx9+2GdnZ9YyZgsiPWP+
pa4/LAgSOVYtlSVtT5dJiGODKj9b/ydHW2sO24gxEysor3WOHrY5IqlDT7laiDhR
6tQxIol7NAwvS0F7eDFj261lC98qOwqMYkNdR52Pe7gpAAAKNgl5UI7zKvtzQloT
gqkbwFXrbcbByEdU2vA6Pi5TVS2Y+y1f/C0ePNQBFVqgDWe1KcNDqSOZXuIDlBEH
/27tA7RnmzhHPBNcXQn2q6hu+ae5ht0CMnRwp/0JrqIg40MYgLaTcee3mv900kdA
5RkYxn7yGvLN6zf4yrhcXMRf999Sod+LKMvFocI7FET/lRim9mhnjqcyS+zr48uz
6sp3F9amQVLYN6KGGyFDehv5j8+xgw1oLaOTJ18DNHDV+u1+FOcZK2+ToktjFgUr
gVImIF6Aic/nz8HX6OEG6u+XxL4Y0mnkn9EanBnzH1nW7P/wLXGE6lPBlxn4aZqO
fCuRARdF8EWpP/n51YRQapvlZyzIJBkSInS5UnhxPeIhIUYHgyIbRoYwzHpIwWS2
zokXaBpyqS/fSlHh+nclDjdpiokCHwQYAQIACQUCUZFeVQIbDAAKCRDRjf6JvkUr
J0svD/92D32R77PC1bNSTxbHAV3zKrhh7kXC78M7LZRn0kgNR8f9i4cTjgZolTmo
GFVNTnPpVfnutJiPFFqk8BR1Tk+/U5Xz4HM1Jp+G83ZoOL8suBgN/mwQ13swFO2k
07Bn7MHAjQ934ZV+O653ZvKXR2OtMmAITV+J8BxFwZ+2yGOtQUMyJn1SOQrG7COP
mw9YG0q9PuPN81HLmB6HkbsGmjADZFNrIuH47U8NJX3kpKqd65sBBIPmBJ1Oq1O5
XHoF3RwTJke4zpE60VQUvyKox/v3fZjg+wQrj7/cb0445G3edUVv8tOtqBc5v/qS
Y0RUri//agjwZenj5KDBIuV5YKG7pvlO89WgZKRvkKLWcsG1gwIVaBf3VLWtTppY
LpJL1Rp9Zzeme3Z0NqaZLB6g85YyyIpvFrs29/FOE9lU368q+lSqFeL8jNdnShhS
z3a0+mrsP3ySXswPNDRn241KTlTFRAbqSQ7hJbQ+ttDwzRdxV/FdHTemFh7V+RKQ
wpVGUJQhLtOSajUl+mjIIv4gVtxSDiEoC/D5//j1/y/plZ87K8nDmojZRJ+cuDY
1NRT+azu030OvYMcGtjndkonJqoM5yO/1Of5n+eBniG56jmLEg2Gvea3F41zKtNc
LyY8BtB9aSe8EaGnl4SX3gHU2J9nR47frPs/SDe0dprZEhVyFQ==
=i+iY
-----END PGP PRIVATE KEY BLOCK-----
```

Mere days after making first contact with the Vulcans, Zefram Cochrane generated a new PGP key

Search results for '0x027

zimmerman.mayfirst.org/pks/lookup?op=vindex&se

Search results for '0x027aa55a'

```
Type bits/keyID     cr. time   exp time   key expir
------------------------------------------------------------------------------
pub  4096R/027AA55A 2063-04-12
      Fingerprint=4B82 D3B6 5289 A2E2 5B34  0014 AD7C 75E6 027A A55A

uid Zefram Cochrane (I invented warp drive!) <zcochrane@ufop.org>
sig  sig3  027AA55A 2063-04-12 __________ __________ [selfsig]

sub  4096R/C3BD0CF5 2063-04-12
sig sbind  027AA55A 2063-04-12 __________ __________ []
```

Faking Key IDs

My fingerprint is:
0x5C17616361BD9F92422AC08BB4D25A1E99999697
(20 bytes)

My key ID:
0x99999697 (4 bytes)

People often use short key IDs to uniquely identify keys. This is a bad idea.

Key IDs Can't Be Trusted

- My public key has these user IDs and this key ID:
  Micah Lee <micahflee@riseup.net>
  Micah Lee <micah@eff.org>
  Micah Lee <micahflee@gmail.com>
  Micah Lee <micah@pressfreedomfoundation.org>
  0x99999697, 2011-06-24
- User IDs and timestamps can be faked
- If someone can fake my key ID as well, they can make an imposter key that has all the above information
- Let me should you how fake key IDs

Menu

Home

This section

Jotted-down notes at the intersection of my mind and permanence

Recycling the past

Comments

Comments are welcome. Email me.

Mon, 26 Dec 2011

Short key IDs are bad news (with OpenPGP and GNU Privacy Guard)

Summary: It is important that we (the Debian community that relies on OpenPGP through GNU Privacy Guard) stop using short key IDs. There is no vulnerability in OpenPGP and GPG. However, using short key IDs (like 0x70096AD1) is fundementally insecure; it is easy to generate collisions for short key IDs. We should *always use 64-bit* (or longer) key IDs, like: 0x37E1C17570096AD1 or 0xEC4B033C70096AD1.

TL;DR: This now gives two results: `gpg --recv-key 70096AD1`

Some background, and my two keys

Years ago, I read dkg's instructions on migrating the Debian OpenPGP infrastructure. It told me that the time and effort I had spent getting my key into the strong set wasn't as useful as I thought it had been.

I felt deflated. I had put in quite a bit of effort over the years to strongly-connect my key to a variety of signatures, and I had helped people get their own keys into the strong set this way. If I migrated off my old key and revoked it, I'd be abandoning some people for whom I was their only link into the strong set. And what fun it was to first become part of the strong set! And all the eyebrows I raised when I told people I was going meet up with people I met on a website called Biglumber... I even made it my Facebook.com user ID. So if I had to generate a new key, I decided I had better really love the short key ID.

But at that point, I already felt pretty attached to the number 0x70096AD1. And I couldn't come up with anything better. So that settled it: no key upgrade until I had a new key whose ID is the same as my old key.

That dream has become a reality. Search for my old key ID, and you get two keys!

```
$ gpg --keyserver pgp.mit.edu --recv-key 0x70096AD1
gpg: requesting key 70096AD1 from hkp server pgp.mit.edu
gpg: key 70096AD1: public key "Asheesh Laroia <asheesh@asheesh.org>" imported
gpg: key 70096AD1: public key "Asheesh Laroia <asheesh@asheesh.org>" imported
gpg: no ultimately trusted keys found
gpg: Total number processed: 2
gpg:               imported: 2  (RSA: 1)
```

I also saw it as an opportunity: I know that cryptography tools are tragically easy to mis-use. The use of 32-bit key IDs is fundamentally incorrect -- too little entropy. Maybe shocking people by creating two "identical" keys will help speed the transition away from this mis-use.

A neat stunt abusing --refresh-keys

What is an OpenPGP fingerprint?

fingerprint = hash(public_key)

Hash functions takes data and returns output that is indistinguishable from random.

PGP fingerprints are 20 bytes long of (indistinguishable from random) data.

What are the chances two keys share the same last 4 bytes?

Birthday attack

From Wikipedia, the free encyclopedia

A **birthday attack** is a type of cryptographic attack that exploits the mathematics behind the birthday problem in probability theory. This attack can be used to abuse communication between two or more parties. The attack depends on the higher likelihood of collisions found between random attack attempts and a fixed degree of permutations (pigeonholes), as described in the birthday problem/paradox.

Contents [hide]

Understanding the problem [edit]

Main article: Birthday problem

As an example, consider the scenario in which a teacher with a class of 30 students asks for everybody's birthday, to determine whether any two students have the same birthday (corresponding to a hash collision as described below; for simplicity, ignore February 29). Intuitively, this chance may seem small. If the teacher picked a specific day (say September 16), then the chance that at least one student was born on that specific day is $1 - (364/365)^{30}$, about 7.9%. However, the probability that at least one student has the same birthday as *any* other student is around 70% (using the formula $1 - 365!/((365 - n)! \cdot 365^n)$ for n = 30[1]).

Collisions are unintiutively common.
If there are 30 people in a room, there's 70% chance of a birthday collision.

Birthday Attack

- Key IDs are 4 bytes == 32 bits, so the keyspace is 2^32.
- What happens if we generate 3 billion keys?

```
Python 2.7.3 (default, Jan  2 2013, 13:56:14)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> keyspace = 2**32 * 1.0
>>> num_keys = 3000000000
>>> 1 - ((keyspace-1)/keyspace)**num_keys
0.5026652463584134
>>>
```

If you have 3 billion random fingerprints there's a 50% chance of finding a *specific* key ID collision

So how can we generate keys really really quickly?

18 cipher/random.c

View file @ b010e5f

```
...  ...  @@ -323,12 +323,26 @@ static int (*
323  323       }
324  324
325  325       buf = secure && secure_alloc ? xmalloc_secure( nbytes ) : xmalloc( nbytes );
326       -     for( p = buf; nbytes > 0; ) {
     326  +
     327  +     /* trollwot addition: instead of randomly generating bits, use the insecure rand() */
     328  +     struct timeval tv;
     329  +     gettimeofday(&tv, NULL);
     330  +     unsigned long time_in_micros = 1000000 * tv.tv_sec + tv.tv_usec;
     331  +     srand(time_in_micros);
     332  +
     333  +     int i, rnd;
     334  +     size_t nrands = nbytes / sizeof(rnd);
     335  +     for(i = 0; i < nrands; i++) {
     336  +       rnd = rand();
     337  +       strncpy(buf+(sizeof(rnd)*i), (char*)&rnd, sizeof(rnd));
     338  +     }
     339  +
     340  +     /*for( p = buf; nbytes > 0; ) {
327  341     size_t n = nbytes > POOLSIZE? POOLSIZE : nbytes;
328  342     read_pool( p, n, level );
329  343     nbytes -= n;
330  344     p += n;
331       -     }
     345  +     }*/
332  346       return buf;
333  347   }
334  348
```

Oh, and by the way:

Both ascii_sign and fake_sign use this horrible, maimed, insecure version of GnuPG to generate keys

Is there a quicker way?

```
micah@spock:~$ gpg --fingerprint micahflee
pub   4096R/99999697 2011-06-24
      Key fingerprint = 5C17 6163 61BD 9F92 422A  C08B B4D2 5A1E 9999 9697
```

```
public_key = timestamp +
             public_vars

fingerprint = hash(public_key)

fingerprint = hash(timestamp +
              public_vars)
```

Introducing bruteforce_keyid

https://github.com/micahflee/trollwot/

Usage:

```
./brute_force_keyid [KEYID] [USERID]
```

(Only this implementation is really slow. Patches welcome!)

Client-Side Exploits?

Terminal

File Edit View Search Terminal Help

```
[micah@alexia] (master) ~/code/trollwot/trollwot$ gpg --homedir homedir_xss --list-keys --fingerprint
homedir_xss/pubring.gpg
------------------------
pub   2048R/BF74A1A6 2012-07-11 [revoked: 2012-07-11]
      Key fingerprint = D1C0 0710 A98C 6068 25AF  16C8 6E5D 912B BF74 A1A6
uid                  <script>alert('hello!');</script> <bill@inputoutput.io>

pub   4096R/C65A27EC 2012-02-27
      Key fingerprint = 6F81 A4C1 3444 3CF7 E017  C292 BDE9 9D48 C65A 27EC
uid                  '"<>?!$' <script>alert('foo')</script> ('"<>?!$' <script></script>) <john.waters@nosebridge.com>
sub   4096R/A1EA5190 2012-02-27

pub    768D/A7B3C04D 2006-06-08
      Key fingerprint = 9820 0A68 87AF 72CD 6C59  3559 06AB 7A6A A7B3 C04D
uid                  DanBTesting (<script>alert('Alert!');</script>) <dan-test@f-box.org>
sub    768g/B25DEBB9 2006-06-08

pub   1024D/306E2139 2006-02-07
      Key fingerprint = 13BA 2118 E9C8 5051 8191  E05F C1BB D7FB 306E 2139
uid                  <script> alert('foo'); </script>
sub   2048g/E1957A8B 2006-02-07

[micah@alexia] (master) ~/code/trollwot/trollwot$
```

Downloading the Web of Trust

- I tried recursively downloading keys, starting with my own (see download_strong_set)
- Turns out this is really slow
- But some key servers provide weekly static dumps of all the keys:
  - ftp://ftp.prato.linux.it/pub/keyring/
  - http://keys.niif.hu/keydump/
  - http://keyserver.borgnet.us/dump/

But what can you do with a copy of the web of trust?

Mirror World Web of Trust

Mirror World Web of Trust

With all of the public keys you can:

- Generate your own mirror copy of all these keys, with the same user IDs
- They can even have the same 4-byte key IDs
- Make them randomly sign each other
- Make them randomly sign real keys too!

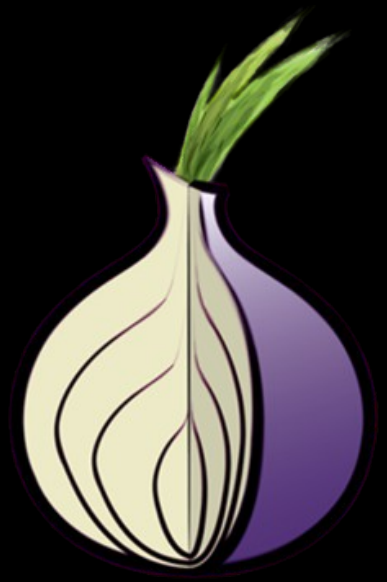
Design as many intricate mirror trust universes as you wish, and make them collide with normal spacetime in the key servers

Denial of Service

- What happens if you append 8GB of signatures to a public key?
  - How will key servers handle it?
  - Will Enigmail crash?
- What happens if you create a key and add terabytes of user IDs?
  - Will it cause the key servers to run out of disk space?
  - Will all key servers run out of disk space when they try syncing this key?

Botnet Command & Control

- Bots can be hard-coded with a public key (botnet_key)
- Bots can refresh botnet_key from key servers to receive new commands
- Commands are sigs on botnet_key. As long as the key that generated the sig is signed by botnet_key, it trusts the command
- There can be a command to switch botnet_key to a different public key



+

=

hkp://2eghzlv2wwcq7u7y.onion

Hacking on OpenPGP

python-gnupg

- A python module that lets you easily script stuff with GnuPG
- Useful for key generation, signing, verify, encrypting, decrypting, etc.
- Official: https://code.google.com/p/python-gnupg/
- Isis's version: https://github.com/isislovecruft/python-gnupg

Hacking on OpenPGP

python-pgpdump

- Takes any sort of OpenPGP data as input (public keys, secret keys, signatures, encryption blocks of data)
- Separates it into packets to inspect, grab data from
- https://github.com/toofishes/python-pgpdump

Hacking on OpenPGP

gnupg

- Hacking on GnuPG itself is not as daunting as it sounds
- You don't need to be a cryptographer, just bit of a C programmer
- Grep is your friend
- http://git.gnupg.org/

Fear not, OpenPGP isn't broken

- KEY SERVERS AND THE WEB OF TRUST ARE NOT NECESSARY FOR OPENPGP TO BE USEFUL
- Encryption, decryption, signing, verifying all work great
- Key servers provide convenience in exchanging keys, *but we cannot trust what's in them without knowing exactly what we're doing*
- Many of OpenPGP users refuse to use the web of trust anyway

Be Excellent to Each Other

- Don't be a jerk! No one likes jerks.
- People rely on OpenPGP for life and death situations
- Some of whom may not realize that the web of trust is about as informative as YouTube comments
- Most users won't become OpenPGP experts, nor should they

The Cypherpunk Future

- It's not going to be using OpenPGP
- It's going to be easy to use
- It's going to be turned on by default
- Key management won't be quite as ridiculously cumbersome (I hope)
- Associations will be private
- It will probably be in a web browser
- It probably doesn't exist yet

Code for Trolling the Web of Trust

https://github.com/micahflee/trollwot

Pull requests welcome :)

Thanks!
Feel free to sign my key

Micah Lee
Staff Technologist
Electronic Frontier Foundation

```
micah@spock:~$ gpg --fingerprint 99999697
pub   4096R/99999697 2011-06-24
      Key fingerprint = 5C17 6163 61BD 9F92 422A C08B B4D2 5A1E 9999 9697
uid                  Micah Lee <micahflee@riseup.net>
uid                  Micah Lee <micah@eff.org>
uid                  Micah Lee <micahflee@gmail.com>
uid                  Micah Lee <micah@pressfreedomfoundation.org>
sub   4096R/E8839F99 2011-06-24
```

Twitter: @micahflee