[Dave Steele's Blog](https://davesteele.github.io/)

[Debian](https://davesteele.github.io/category/debian/) [Development](https://davesteele.github.io/category/development/) [Linux](https://davesteele.github.io/category/linux/) [Raspberry Pi](https://davesteele.github.io/category/raspberrypi/)

Anatomy of a GPG Key

Sep 20, 2014

I have a confession to make. I had a fairly hard time understanding all of the ins and outs of managing keys using the gnupg tool ‘gpg’. Pretty much all of the documentation is procedural - how to use the tool to accomplish some specific tasks. Many questions that I had were tangential to the particular procedure, and therefore not covered where I needed it to be.

For me, the key to understanding how to work with gpg was to understand the packet structure of the underlying OpenPGP Message Format ([RFC4880](http://tools.ietf.org/html/rfc4880)), which defines how gpg messages, signatures, and key material are stored. The goal of this post is to grease the skids for the next guy, by tying the key storage format to the RFC definition, and to the associated gpg commands and parameters.

TL;DR

Here are some takeaways I wish I had going into this:

* Most key parameters are stored in the self signature. That means they can be changed at will by the key owner without affecting the status of external key signatures.
* Subkeys need only be self-signed (which is automatic). Trust from external signatures is provided transitively.
* (Edit - 19 Apr 2015) gpg [automatically uses the newest subkey](http://support.gpgtools.org/discussions/problems/8919-force-subkey-for-signing#19%20Jun,%202013%2012:15%20PM) to sign/encrypt.

Some Terms

It’s best that you have an understanding of data encryption and data signing using public key cryptography before you read this. You should also know about key signing and the the reason for it. Oh, and also binary-to-hexadecimal conversion for one (small) part. Having said that, let’s be clear on a couple of terms:

* Primary key vs. subkey - A PGP key certificate may contain other information in addition to the key itself. A subkey is a key that is stored as a sub-component of another key. The primary key is the top level key. It is often referred to elsewhere as the master key.
* Public key - This post is working with the published version of the key certificate. Therefore, only public keys are described (the ones that encrypt and verify signatures). Your local version of your key also includes the associated private keys (for decryption and signature creation), to define the key pair.
* Key certificate - Part of the challenge of understanding gpg key management documentation is the flexibility in the definition of the word ‘key’. It can refer to a specific private or public key, or to a particular key pair, or to the OpenPGP ‘certificate’ that defines a suite of information associated with a key or set of keys. I will use the term “key/public key” and “key certificate” to distinguish between the possible interpretations. Key pairs and private keys will not come up here. We will be focusing on the key certificate.
* Key ID - A hexadecimal string that identifies a key (usually the primary key).
* UID, or User ID - The name and email of the user is stored in one or more UID entries, stored under the Primary key.
* Certification vs. signing - ‘Signing’ is an action against arbitrary data. ‘Certification’ is the signing of another key. Ironically, the act of certifying a key is universally called “key signing”. Just embrace the contradiction.
* Key packet - ‘Packet’ is the term used by RFC4880 to identify a component of the message/certificate format. Messages and keys certificates are made up of packets and subpackets of various types.
* Trust, Validity, and the Web of Trust - gpg uses a model of ‘trust’ of users (defined locally-only using the ‘trust’ edit command) and reported ‘validity’ of keys (defined by key signatures/certificates). The combination creates a “Web of Trust”, starting with locally-defined trust statements about users, and passing through multiple levels of key-signature-defined validity links to other keys. Gpg uses the web of trust to determine if a key is acceptable for use without warning the user. There is a writeup in the [GNU Privacy Handbook](https://www.gnupg.org/gph/en/manual/x334.html#AEN384) that covers the concepts well enough if you have the terms straight. Documentation often uses the word ‘trust’ for both ‘trust’ and ‘validity’. I mention all of this only to note that this document is concerned with ‘validity’.

My Key

Following is an annotated and edited dump of my key certificate, originally generated with:

gpg -a --export "David Steele" | gpg --list-packets --verbose

where “David Steele” matches a UID for my key. Substitute your name or primary key id to see your key certificate. Add the ‘--debug 0x02’ option to the second gpg invocation to see the entire contents, including the binary key data (thanks [superuser.com](http://superuser.com/questions/696941/human-readable-dump-of-gpg-public-key)).

Note that there are [other tools](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#OtherTools) which provide more information or features for this task. I use gpg as the least common denominator tool.

This is a pretty standard published key certificate, which is to say that it contains a primary certification/signing public key, with a public subkey dedicated to encryption (GPG always creates a separate encryption subkey to the primary, to avoid [problems](http://serverfault.com/questions/397973/gpg-why-am-i-encrypting-with-subkey-instead-of-primary-key)). I also have an extra public signing subkey with an expiration date, for a couple of [reasons](https://wiki.debian.org/Subkeys#Why.3F).

I’ve linked aspects of the key dump to explanation paragraphs below.

*#*[*Primary key*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#PrimaryKey) *:public key*[*packet*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#PacketType)*:*[*version 4*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#PacketVer)*,*[*algo 1*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#KeyAlg)*, created*[*1281838967*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#DatesExpir)*, expires 0  
    pkey[0]: [4096 bits]  
    pkey[1]: [17 bits]*[*keyid: 8A3171EF366150CE*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#KeyId)

*#*[*User ID #1*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#UID1) *:user ID packet: “David Steele <daves@users.sourceforge.net>”*

*#*[*Endorsing Signatures*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#EndorsingSigs) *#*[*Self Signature*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#SelfSig) *:signature packet: algo 1, keyid 8A3171EF366150CE  
    version 4, created 1281838967, md5len 0,*[*sigclass 0x13*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#SigClass)[*digest algo 8*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#DigestAlgo)*, begin of digest f9 34*[*hashed subpkt*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#SigSubpacket)*2 len 4 (sig created 2010-08-15)  
    hashed subpkt 27 len 1 (*[*key flags*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#KeyFlags)*: 03)  
    hashed subpkt 11 len 4 (*[*pref-sym-algos*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#PrefSymAlgos)*: 9 8 7 3)  
    hashed subpkt 21 len 4 (*[*pref-hash-algos*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#PrefHashAlgos)*: 10 9 8 11)  
    hashed subpkt 22 len 4 (pref-zip-algos: 2 3 1 0)  
    hashed subpkt 30 len 1 (*[*features*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#Features)*: 01)  
    hashed subpkt 23 len 1 (*[*key server preferences*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#KeyServPrefs)*: 80)  
    subpkt 16 len 8 (issuer key ID 8A3171EF366150CE)  
    data: [4092 bits]  
#*[*External Signatures*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#ExternalSigs) *:signature packet: algo 17, keyid F7EBEE8EB7982329  
    version 4, created 1284226833, md5len 0, sigclass 0x10  
    digest algo 2, begin of digest fe c2  
    hashed subpkt 2 len 4 (sig created 2010-09-11)  
    subpkt 16 len 8 (issuer key ID F7EBEE8EB7982329)  
    data: [160 bits]  
    data: [157 bits]  
:signature packet: algo 17, keyid DABDF1E4C88B0C9C  
    version 4, created 1349019663, md5len 0, sigclass 0x13  
    digest algo 2, begin of digest 1b fd  
    hashed subpkt 2 len 4 (sig created 2012-09-30)  
    subpkt 16 len 8 (issuer key ID DABDF1E4C88B0C9C)  
    data: [160 bits]  
    data: [158 bits]*

*# User ID #2  
:user ID packet: “David Steele <dsteele@gmail.com>”*

*:signature packet: algo 1, keyid 8A3171EF366150CE  
    version 4, created 1322364912, md5len 0, sigclass 0x13  
    digest algo 2, begin of digest e6 05  
    hashed subpkt 2 len 4 (sig created 2011-11-27)  
    hashed subpkt 27 len 1 (key flags: 03)  
    hashed subpkt 11 len 5 (pref-sym-algos: 9 8 7 3 2)  
    hashed subpkt 21 len 5 (pref-hash-algos: 8 2 9 10 11)  
    hashed subpkt 22 len 3 (pref-zip-algos: 2 3 1)  
    hashed subpkt 30 len 1 (features: 01)  
    hashed subpkt 23 len 1 (key server preferences: 80)  
    subpkt 16 len 8 (issuer key ID 8A3171EF366150CE)  
    data: [4096 bits]  
:signature packet: algo 17, keyid F7EBEE8EB7982329  
    version 4, created 1322413422, md5len 0, sigclass 0x10  
    digest algo 2, begin of digest b4 2e  
    hashed subpkt 2 len 4 (sig created 2011-11-27)  
    subpkt 16 len 8 (issuer key ID F7EBEE8EB7982329)  
    data: [157 bits]  
    data: [159 bits]*

*#*[*Encryption subkey*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#EncryptSubkey) *:public sub key packet:  
    version 4, algo 1, created 1281839112, expires 0  
    pkey[0]: [4096 bits]  
    pkey[1]: [17 bits]  
    keyid: 2DC87C4C0D929394  
:signature packet: algo 1, keyid 8A3171EF366150CE  
    version 4, created 1281839112, md5len 0, sigclass 0x18  
    digest algo 8, begin of digest 87 3b  
    hashed subpkt 2 len 4 (sig created 2010-08-15)  
    hashed subpkt 27 len 1 (key flags: 0C)  
    subpkt 16 len 8 (issuer key ID 8A3171EF366150CE)  
    data: [4096 bits]*

*#*[*Signing subkey*](https://davesteele.github.io/gpg/2014/09/20/anatomy-of-a-gpg-key/#SigningSubkey) *:public sub key packet:  
    version 4, algo 1, created 1408105689, expires 0  
    pkey[0]: [4096 bits]  
    pkey[1]: [17 bits]  
    keyid: 627EBB290A817A82  
:signature packet: algo 1, keyid 8A3171EF366150CE  
    version 4, created 1408105689, md5len 0, sigclass 0x18  
    digest algo 2, begin of digest 62 1d  
    hashed subpkt 2 len 4 (sig created 2014-08-15)  
    hashed subpkt 27 len 1 (key flags: 02)  
    hashed subpkt 9 len 4 (key expires after 5y0d0h0m)  
    subpkt 16 len 8 (issuer key ID 8A3171EF366150CE)  
    subpkt 32 len 540 (signature: v4, class 0x19, algo 1, digest algo 2)  
    data: [4094 bits]*

Primary Key

The first packet in a published OpenPGP/gpg key certificate is the primary signing/certification public key. The overall key certificate is referenced by the Key ID of this key.

Packet Type

The ‘types’ of packets in an OpenPGP key certificate or message are defined in Section 5 of the RFC ([RFC4880-5](http://tools.ietf.org/html/rfc4880#section-5)). The primary signing public key uses a packet with a ‘tag’ value of 6 ([RFC4880-5.5.1.1](http://tools.ietf.org/html/rfc4880#section-5.5.1.1)). The tag values are not shown in the gpg key dump - just the resulting type.

Packet Version

The RFC defines both ‘version 3’ and ‘version 4’ packet types. For key packets, a modern gpg will only create version 4 packets ([RFC4880-5.5.2](http://tools.ietf.org/html/rfc4880#section-5.5.2)). You should avoid working with version 3 keys - there are known weaknesses with the format.

Key Algorithm

The “algo” parameter in the dump identifies the encryption algorithm associated with the key packet. The list of required and possible algorithms is listed in the “Constants” section of the RFC ([RFC4880-9.1](http://tools.ietf.org/html/rfc4880#section-9.1)).

That section states that ‘DSA’ (for signing) and ‘Elgamal’ (for encryption) can be expected to be the most available across implementations and versions. My key uses RSA (encrypt or sign), which is the current default in gpg:

$ gpg --gen-key

gpg (GnuPG) 1.4.18; Copyright (C) 2014 Free Software Foundation, Inc.

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:

(1) RSA and RSA (default)

(2) DSA and Elgamal

(3) DSA (sign only)

(4) RSA (sign only)

Your selection? 1

RSA keys may be between 1024 and 4096 bits long.

What keysize do you want? (2048) 4096

...

The information stored for the key (pkey[0] and pkey[1] in this case) is dependent on the algorithm.

The length of the key is important. Mine is 4096 bits, based on the current [recommendation](http://keyring.debian.org/creating-key.html) from Debian (as an exercise, you can verify that the key listed in that document is RSA).

Dates and Expiration

The dates in the key certificate dump are [Unix epochs](http://en.wikipedia.org/wiki/Unix_time). Convert to human-readable with:

$ date -d @1281838967

Sat Aug 14 22:22:47 EDT 2010

The ‘expires’ value of ‘0’ means that the key itself has no expiration date. Note that we will find out shortly that there are multiple ways to express key expiration. This mechanism is part of the key definition - changing it would change the Key ID for the Primary key, which would in turn invalidate all signatures for the key certificate.

Perhaps for that reason, gpg does not use this field to define the expiration date for a generated key. It is expressed in the key self-signature, as shown later.

There has been a fair amount of mulling about the right strategy for key expiration. My sense is that the consensus is that opinion on expiring Primary keys is mixed, encryption subkeys should have no expiration, and signing subkeys should.

Key ID

The RFC defines a 160-bit ‘fingerprint’ for a key, which is typically expressed as a hexadecimal string, divided into 10 4-character groups ([rfc4880-12.2](http://tools.ietf.org/html/rfc4880#section-12.2)). When validating keys for key signing, the fingerprint is used.

$ gpg --fingerprint "David Steele"

pub 4096R/366150CE 2010-08-15

Key fingerprint = AE0D BF5A 92A5 ADE4 9481 BA6F 8A31 71EF 3661 50CE

uid David Steele <dsteele@gmail.com>

uid David Steele <daves@users.sourceforge.net>

sub 4096R/0D929394 2010-08-15

sub 4096R/0A817A82 2014-08-15 [expires: 2019-08-14]

The key certificate dump is expressing this fingerprint as a ‘key id’ (or ‘long key id’), taking the last 16 characters of that fingerprint (again, [rfc4880-12.2](http://tools.ietf.org/html/rfc4880#section-12.2)).

The gpg program muddies the waters a bit by using the last 8 characters of the fingerprint as its definition of the key id (‘short key id’), shown on the ‘pub’ line for the fingerprint call above. It is using the definition of key id from section 3.3 ([rfc4880-3.3](http://tools.ietf.org/html/rfc4880#section-3.3)).

Going one step further down the rabbit hole, in some contexts this value needs to have “0x” prepended (‘0x366150CE’). I’ve run across this in a key server search function.

The key id is a shorthand method for referring to a particular key or key certificate. The 8-character version is the primary mechanism for referring to a particular key, even though it is spoof-able, and many consider this a [terrible idea](https://www.debian-administration.org/users/dkg/weblog/105).

The Key ID of the Primary public key (‘366150CE’ in this case) is used to refer to some of its own subkeys, such as the associated private signing key, as well as the encryption subkey.

The fingerprint/key id is a hash of the entire key packet, and only the key packet. It is invalidated (changed) if any information in the key packet is changed, but is unaffected by any changes in any other packets.

User ID

The user ID packet defines a name/email address that is associated with the key certificate ([RFC4880-5.11](http://tools.ietf.org/html/rfc4880#section-5.11)). The gpg program will store it in [RFC2822](http://tools.ietf.org/html/rfc2822#section-3.4) format (“David Steele <dsteele@gmail.com>”) based on the name and email you provide it when you generated the key.

The key certificate can have more than one user id. For instance, if you want to use the key certificate with more than one email account, multiple user ids would be needed.

Endorsing Signatures

A user id packet is followed by one or more ‘signature packets’.

This is a key point - the key signature covers the contents of the primary key packet and the last-encountered user id packet. So, the signature is a certification by the signer that the user identified in the user id is who he says he is, and that he claims ownership of the key.

A key signature only covers one user id. Separate signatures are needed if certification is desired for multiple user ids.

Signatures are identified by the ‘signature type’, shown as ‘sigclass’ in the packet certificate dump. Types/classes 0x10 through 0x13 are key signatures ([RFC4880-5.2.1](http://tools.ietf.org/html/rfc4880#section-5.2.1)).

Key signatures can be created using the ‘sign-key’ command in the key edit mode.

Self Signature

The ‘self signature’ is a key signature generated by the primary key being signed. It serves as verification that the user id is valid for that key. For version 4 packets, it also provides a number of parameters to be associated with the key/user id pair ([RFC4880-5.2](http://tools.ietf.org/html/rfc4880#section-5.2)).

Self signatures are generated automatically by gpg, as keys are generated and as parameters are changed.

Signature Class

Again, the key signature type, or ‘sigclass’, identifies the signature as a key signature. It also claims to provide a level of assurance of that certification, from “does not make any particular assertion” to “substantial verification of the claim of identity” ([RFC4880-5.2](http://tools.ietf.org/html/rfc4880#section-5.2)).

It appears that the levels are largely ignored for key validation purposes. The Debian key signing guidelines recommends using ‘[casual](https://www.debian.org/events/keysigning)’, which I believe maps to 0x12. However, the RFC states:

*Most OpenPGP implementations make their “key signatures” as 0x10 certifications. Some implementations can issue 0x11-0x13 certifications, but few differentiate between the types.*

I have both 0x10 and 0x13 key signatures in my key certificate.

Digest Algorithm

A signature is actually a cryptographic operation over a hash of the entity being signed. The list of digest algorithms (‘hash’ and ‘digest’ are interchangeable) are in [RFC4880-9.4](http://tools.ietf.org/html/rfc4880#section-9.4).

Note that MD5 (digest algo 1) is [broken](https://www.gnupg.org/faq/weak-digest-algos.html#sec-3). SHA-1 (digest algo 2) is the standard, but is showing signs of age. You can set the signature preferences using the [personal-digest-preferences](https://www.gnupg.org/documentation/manuals/gnupg/OpenPGP-Options.html#index-personal_002ddigest_002dpreferences-280) and [cert-digest-algo](https://www.gnupg.org/documentation/manuals/gnupg/GPG-Esoteric-Options.html#index-cert_002ddigest_002dalgo-324) parameters in ~/.gnupg/gpg.conf. [Debian recommends](http://keyring.debian.org/creating-key.html) that you set this to SHA256.

“Endorsing Signature” Subpackets

Here is where things got interesting for me, and enlightening. The key self signature contains subpackets with additional information about the key and how it is to be used ([RFC4880-5.2](http://tools.ietf.org/html/rfc4880#section-5.2)). That means such information is validated by the primary key, but that the information is unaffected by (and unaffecting of) other externally-generated key signatures. Some of these subpackets are detailed in the following sections, with the gpg mechanisms for manipulation. The full list of subpackets is in [RFC4880-5.2.3.1](http://tools.ietf.org/html/rfc4880#section-5.2.3.1)

Key Flag Subpacket

I’ve said a couple of times now that the primary key is used for signing, and there is a separate encryption subkey. Well, it is the ‘keyflags’ field that defines these roles for the keys ([RFC4880-5.2.3.21](http://tools.ietf.org/html/rfc4880#section-5.2.3.21)).

The flag data is reflected in the output of gpg when you invoke --edit-key:

$ gpg --edit-key "David Steele"

gpg (GnuPG) 1.4.18; Copyright (C) 2014 Free Software Foundation, Inc.

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

Secret key is available.

pub 4096R/366150CE created: 2010-08-15 expires: never usage: SC

trust: ultimate validity: ultimate

sub 4096R/0D929394 created: 2010-08-15 expires: never usage: E

sub 4096R/0A817A82 created: 2014-08-15 expires: 2019-08-14 usage: S

[ultimate] (1). David Steele <dsteele@gmail.com>

[ultimate] (2) David Steele <daves@users.sourceforge.net>

The ‘usage’ characters map to the key flags as follows:

| **Flag** | **gpg character** | **Description** |
| --- | --- | --- |
| 0x01 | “C” | Key Certification |
| 0x02 | “S” | Sign Data |
| 0x04 | “E” | Encrypt Communications |
| 0x08 | “E” | Encrypt Storage |
| 0x10 |  | Split key |
| 0x20 | “A” | Authentication |
| 0x80 |  | Held by more than one person |

If you look in my key certificate dump, you’ll see that my primary key’s key flag is 0x03, which is Key Certification plus Sign Data. The encryption subkey is 0x0C, which is Encrypt Communications plus Encrypt Storage. The signing subkey is 0x02, or Sign Data.

There is not much to work with on this flag, under the normal gpg mode. I don’t believe that it is editable in gpg, and the values are created automatically on key creation. Also, there is no option on subkeys to create a key which can both sign and encrypt.

It is possible to set these flags on creation, if you use the gpg “--expert” option along with “--gen-key”. Additional key algorithms are shown on the edit “addkey” command which permit toggling some key flags.

Preferred Symmetric Algorithms

First, be aware that when you use public key algorithms, you are actually managing a symmetric encryption key, which is used to do the actual work of encrypting your message. The why of that is out of scope here.

This parameter is listing the algorithms a sender should use to send you encrypted material. This default list to use (as well as the list for hash and compression/zip algorithms) can be set using the [default-preference-list](https://www.gnupg.org/documentation/manuals/gnupg/GPG-Esoteric-Options.html#index-default_002dpreference_002dlist-360) parameter on the command line or ~/.gnupg/gpg.conf.

The defined list of algorithms is at [RFC4880-9.2](http://tools.ietf.org/html/rfc4880#section-9.2). You will probably not need to work with this list.

Preferred Hash Algorithms

The hash algorithms available are defined at [RFC4880-9.4](http://tools.ietf.org/html/rfc4880#section-9.4). These are used for signatures. MD5 should definitely not be in the list. For compatibility, SHA-1 should be in the list somewhere. Note that this is not current [Debian practice](http://keyring.debian.org/creating-key.html) (see the section on gpg.conf).

Key Features

“Features” doesn’t yet serve much purpose ([RFC4880-5.2.3.24](http://tools.ietf.org/html/rfc4880#section-5.2.3.24)).

Key Server Preferences

The key server preferences subpacket is also a bit light ([RFC4880-5.2.3.17](http://tools.ietf.org/html/rfc4880#section-5.2.3.17))

External Key Signatures

Following the self signature are the externally generated key signatures. You get these when you import a copy of your key that has been signed remotely and exported by a person who is certifying your key certificate/user id. Note that I show two signatures from the same user/key, (‘F7EBEE8EB7982329’), certifying my two user id’s independently.

Encryption Subkey

You can see that this was created at the same time as the primary key. As mentioned previously, this is created automatically with --gen-key.

The parameters have already been discussed.

Signing Subkey

This signing subkey was created in 2014. It, and the encryption subkey, are only self-signed. That alone allows them to inherit the trust from the top level primary keys defined by the other primary key signatures.

The new wrinkle in this key is an expiration subpacket on the self signature, which puts a time limit on the certification ([RFC4880-5.2.3.6](http://tools.ietf.org/html/rfc4880#section-5.2.3.6)).

(Added - 19 Apr 2015) Note that gpg will automatically use the [most recent signing/encryption subkey](http://support.gpgtools.org/discussions/problems/8919-force-subkey-for-signing#19%20Jun,%202013%2012:15%20PM) when the master is referenced. To force a specific key/subkey, add an [exclamation mark](http://www.enricozini.org/2006/tips/gpg-select-subkey/) after the Key ID.

Other Tools

The [hokey](https://help.riseup.net/en/security/message-security/openpgp/best-practices#openpgp-key-checks) utility, from the hopenpgp-tools package, can perform a ‘lint’ operation on your certificate:

$ hkt export-pubkeys <fingerprint> | hokey lint

The pgpdump program provides more information in the dump, and also has a [web interface](http://www.pgpdump.net/).

Should you have made it this far, I hope you have found this useful. Please let me know if you find any issues with the explanations.

[comments powered by Disqus](http://disqus.com/)

* Dave Steele's Blog
* [dsteele@gmail.com](mailto:dsteele@gmail.com)
* [AE0D BF5A 92A5 ADE4 9481  
  BA6F 8A31 71EF 3661 50CE](https://davesteele.github.io/key-366150CE.pub.txt)
* [ RSS](https://davesteele.github.io/feed.xml)
* [davesteele](https://github.com/davesteele)
* [ dsteele](https://twitter.com/dsteele)

Linux Musings. Things to remember/rant about.