Verifying signatures

34-43 minutes ⋮ 9/19/2024

The Qubes OS Project uses digital signatures to guarantee the authenticity and integrity of certain important assets. This page explains how to verify those signatures. It is extremely important for your security to understand and apply these practices.

What digital signatures can and cannot prove

Most people — even programmers — are confused about the basic concepts underlying digital signatures. Therefore, most people should read this section, even if it looks trivial at first sight.

Digital signatures can prove both **authenticity** and **integrity** to a reasonable degree of certainty. **Authenticity** ensures that a given file was indeed created by the person who signed it (i.e., that a third party did not forge it). **Integrity** ensures that the contents of the file have not been tampered with (i.e., that a third party has not undetectably altered its contents *en route*).

Digital signatures **cannot** prove, e.g., that the signed file is not malicious. In fact, there is nothing that could stop someone from signing a malicious program (and it happens from time to time in reality).

The point is that we must decide who we will trust (e.g., Linus Torvalds, Microsoft, or the Qubes Project) and assume that if a trusted party signed a given file, then it should not be malicious or negligently buggy. The decision of whether to trust any given party is beyond the scope of digital signatures. It's more of a social and political decision.

Once we decide to trust certain parties, digital signatures are useful, because they make it possible for us to limit our trust only to those few parties we choose and not to worry about all the bad things that can happen between them and us, e.g., server compromises (qubes-os.org will surely be compromised one day, so don't blindly trust the live version of this site), dishonest IT staff at the hosting company, dishonest staff at the ISPs, Wi-Fi attacks, etc. We call this philosophy distrusting the infrastructure.

By verifying all the files we download that purport to be authored by a party we've chosen to trust, we eliminate concerns about the bad things discussed above, since we can easily detect whether any files have been tampered with (and subsequently choose to refrain from executing, installing, or opening them).

However, for digital signatures to make sense, we must ensure that the public keys we use for signature verification are the original ones. Anybody can generate a cryptographic key that purports to belong to "The Qubes OS Project," but of course only the keys that we (the real Qubes developers) generate are the genuine ones. The rest of this page explains how to verify the authenticity of the various keys used in the project and how to use those keys to verify certain important assets.

OpenPGP software

We use PGP (specifically, the OpenPGP standard). Before we begin, you'll need software that can manage PGP keys and verify PGP signatures. Any program that complies with the OpenPGP standard will do, but here are some examples for popular operating systems:

Linux: GnuPG (documentation). Open a terminal and use the gpg2 command. If you don't already have GnuPG installed, install it via your distro's package manager or from the GnuPG website.

Mac: GPG Suite (documentation). Open a terminal to enter commands.

Windows: Gpg4win (documentation). Use the Windows command line (`cmd.exe`) to enter commands.

Throughout this page, we'll use GnuPG via the gpg2 command. If that doesn't work for you, try gpg instead. If that still doesn't work, please consult the documentation for your specific program (see links above) and the troubleshooting FAQ below.

How to import and authenticate the Qubes Master Signing Key

Many important Qubes OS Project assets (e.g., ISOs, RPMs, TGZs, and Git objects) are digitally signed by an official team member's key or by a release signing key (RSK). Each such key is, in turn, signed by the **Qubes Master Signing Key (QMSK)** (`0x427F11FD0FAA4B080123F01CDDFA1A3E36879494`). In this way, the QMSK is the ultimate root of trust for the Qubes OS Project.

The developer signing keys are set to expire after one year, while the QMSK and RSKs have no expiration date. The QMSK was generated on and is kept only on a dedicated, air-gapped "vault" machine, and the private portion will (hopefully) never leave this isolated machine.

Before we proceed, you must first complete the prerequisite step of installing OpenPGP software.

Once you have appropriate OpenPGP software installed, there are several ways to get the QMSK.

- If you're on Qubes OS, it's available in every qube (except dom0):

```
$ gpg2 --import /usr/share/qubes/qubes-master-key.asc
```

- If you're on Fedora, you can get it in the distribution-gpg-keys package:

```
$ dnf install distribution-gpg-keys
$ gpg2 --import /usr/share/distribution-gpg-keys/qubes/*
```

- If you're on Debian, it may already be included in your keyring.
- Fetch it with GPG:

```
$ gpg2 --fetch-keys https://keys.qubes-os.org/keys/qubes-master-signing-key.asc
```

- Get it from a public keyserver (specified on first use with `--keyserver <URI>` along with keyserver options to include key signatures), e.g.:

  ```
  $ gpg2 --keyserver-options no-self-sigs-only,no-import-clean --keyserver hkp://keyserver.ubuntu.com --
  recv-keys 0x427F11FD0FAA4B080123F01CDDFA1A3E36879494
  ```

- Download it as a file, then import the file.

  Here are some example download locations:

  - Qubes security pack
  - Qubes keyserver
  - Email to qubes-devel
  - Email to qubes-users

  Once you have the key as a file, import it:

  ```
  $ gpg2 --import /<PATH_TO_FILE>/qubes-master-signing-key.asc
  ```

Once you've obtained the QMSK, you must verify that it's authentic rather than a forgery. Anyone can create a PGP key with the name "Qubes Master Signing Key" and the short key ID `0x36879494`, so you cannot rely on these alone. You also should not rely on any single website, not even over HTTPS.

So, what *should* you do? One option is to use the PGP Web of Trust. In addition, some operating systems include the means to acquire the QMSK securely. For example, on Fedora, `dnf install distribution-gpg-keys` will get you the QMSK along with several other Qubes keys. On Debian, your keyring may already contain the necessary keys.

Perhaps the most common route is to rely on the key's fingerprint, which is a string of 40 alphanumeric characters, like this:

```
427F 11FD 0FAA 4B08 0123  F01C DDFA 1A3E 3687 9494
```

Every PGP key has one of these fingerprints, which uniquely identifies it among all PGP keys. (On the command line, you can view a key's fingerprint with the `gpg2 --fingerprint <KEY_ID>` command.) Therefore, if you know the genuine QMSK fingerprint, then you always have an easy way to confirm whether any purported copy of it is authentic, simply by comparing the fingerprints.

But how do you know which fingerprint is the real one? After all, this website could be compromised, so the fingerprint you see here may not be genuine. That's why we strongly suggest obtaining the fingerprint from *multiple independent sources in several different ways*, then comparing the strings of letters and numbers to make sure they match.

For the purpose of convincing yourself that you know the authentic QMSK fingerprint, spaces and capitalization don't matter. In other words, all of these fingerprints are considered the same:

```
427F 11FD 0FAA 4B08 0123  F01C DDFA 1A3E 3687 9494
427f 11fd 0faa 4b08 0123  f01c ddfa 1a3e 3687 9494
427F11FD0FAA4B080123F01CDDFA1A3E36879494
427f11fd0faa4b080123f01cddfa1a3e36879494
```

Instead, what matters is that *all* the characters are present in *exactly* the same order. If even one character is different, the fingerprints should not be considered the same. Even if two fingerprints have all the same characters, if any of those characters are in a different order, sequence, or position, then the fingerprints should not be considered the same.

However, for the purpose of *searching for*, *looking up*, or *entering* keys, spaces and capitalization can matter, depending on the software or tool you're using. You may need to try different variations (e.g., with and without spaces). You may also sometimes see (or need to enter) the entire fingerprint prefixed with `0x`, as in:

```
0x427F11FD0FAA4B080123F01CDDFA1A3E36879494
0x427f11fd0faa4b080123f01cddfa1a3e36879494
```

The `0x` prefix is sometimes used to indicate that the string following it is a hexadecimal value, and some PGP-related tools may require this prefix. Again, for the purpose of convincing yourself that you know the authentic QMSK fingerprint, you may safely ignore the `0x` prefix, as it is not part of the fingerprint. As long as the 40-character string after the `0x` matches exactly, the fingerprint is considered the same. The `0x` prefix only matters if the software or tool you're using cares about it.

The general idea of "comparing fingerprints" is to go out into the world (whether digitally, physically, or both) and find other 40-character strings purporting to be the QMSK fingerprint, then compare them to your own purported QMSK fingerprint to ensure that the sequence of alphanumeric characters is exactly the same (again, regardless of spaces or capitalization). If any of the characters do not match or are not in the same order, then at least one of the fingerprints is a forgery. Here are some ideas to get you started:

- Check the fingerprint on various websites (e.g., mailing lists, discussion forums, social media, personal websites).
- Check against PDFs, photographs, and videos in which the fingerprint appears (e.g., slides from a talk, on a T-shirt, or in the recording of a presentation).
- Ask people to post the fingerprint on various mailing lists, forums, and chat rooms.
- Download old Qubes ISOs from different sources and check the included Qubes Master Signing Key.

- Repeat the above over Tor.
- Repeat the above over various VPNs and proxy servers.
- Repeat the above on different networks (work, school, internet cafe, etc.).
- Text, email, call, video chat, snail mail, or meet up with people you know to confirm the fingerprint.
- Repeat the above from different computers and devices.

Once you’ve observed enough matching fingerprints from enough independent sources in enough different ways that you feel confident that you have the genuine fingerprint, keep it in a safe place. Every time you need to check whether a key claiming to be the QMSK is authentic, compare that key’s fingerprint to your trusted copy and confirm they match.

Now that you’ve imported the authentic QMSK, set its trust level to “ultimate” so that it can be used to automatically verify all the keys signed by the QMSK (in particular, RSKs).

```
$ gpg2 --edit-key 0x427F11FD0FAA4B080123F01CDDFA1A3E36879494
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.

pub  4096R/36879494  created: 2010-04-01  expires: never       usage: SC
                     trust: unknown       validity: unknown
[ unknown] (1). Qubes Master Signing Key

gpg> fpr
pub   4096R/36879494 2010-04-01 Qubes Master Signing Key
Primary key fingerprint: 427F 11FD 0FAA 4B08 0123  F01C DDFA 1A3E 3687 9494

gpg> trust
pub  4096R/36879494  created: 2010-04-01  expires: never       usage: SC
                     trust: unknown       validity: unknown
[ unknown] (1). Qubes Master Signing Key

Please decide how far you trust this user to correctly verify other users' keys
(by looking at passports, checking fingerprints from different sources, etc.)

   1 = I don't know or won't say
   2 = I do NOT trust
   3 = I trust marginally
   4 = I trust fully
   5 = I trust ultimately
   m = back to the main menu

Your decision? 5
Do you really want to set this key to ultimate trust? (y/N) y

pub  4096R/36879494  created: 2010-04-01  expires: never       usage: SC
                     trust: ultimate      validity: unknown
[ unknown] (1). Qubes Master Signing Key
Please note that the shown key validity is not necessarily correct
unless you restart the program.

gpg> q
```

Now, when you import any of the release signing keys and many Qubes team member keys, they will already be trusted in virtue of being signed by the QMSK.

As a final sanity check, make sure the QMSK is in your keyring with the correct trust level.

```
$ gpg2 -k "Qubes Master Signing Key"
pub   rsa4096 2010-04-01 [SC]
      427F11FD0FAA4B080123F01CDDFA1A3E36879494
uid           [ultimate] Qubes Master Signing Key
```

If you don’t see the QMSK here with a trust level of “ultimate,” go back and follow the instructions in this section carefully and consult the troubleshooting FAQ below.

How to import and authenticate release signing keys

Every Qubes OS release is signed by a **release signing key (RSK)**, which is, in turn, signed by the Qubes Master Signing Key (QMSK).

Before we proceed, you must first complete the following prerequisite steps:

1. Install OpenPGP software.
2. Import and authenticate the QMSK.

After you have completed these two prerequisite steps, the next step is to obtain the correct RSK. The filename pattern for RSKs is `qubes-release-X-signing-key.asc`, where `X` is either a major or minor Qubes release number, such as `4` or `4.2`. There are several ways to get the RSK for your Qubes release.

- If you have access to an existing Qubes installation, the release keys are available in dom0 in `/etc/pki/rpm-gpg/RPM-GPG-KEY-qubes-*`. These can be copied into other qubes for further use. In addition, every other qube contains the release key corresponding to that installation's release in `/etc/pki/rpm-gpg/RPM-GPG-KEY-qubes-*`. If you wish to use one of these keys, make sure to import it into your keyring, e.g.:

```
$ gpg2 --import /etc/pki/rpm-gpg/RPM-GPG-KEY-qubes-*
```

- Fetch it with GPG:

```
$ gpg2 --keyserver-options no-self-sigs-only,no-import-clean --fetch-keys https://keys.qubes-
os.org/keys/qubes-release-X-signing-key.asc
```

- Download it as a file. You can find the RSK for your Qubes release on the downloads page. You can also download all the currently used developers' signing keys, RSKs, and the Qubes Master Signing Key from the Qubes security pack and the Qubes keyserver. Once you've downloaded your RSK, import it with GPG:

```
$ gpg2 --keyserver-options no-self-sigs-only,no-import-clean --import ./qubes-release-X-signing-key.asc
```

Now that you have the correct RSK, you simply need to verify that it is signed by the QMSK:

```
$ gpg2 --check-signatures "Qubes OS Release X Signing Key"
pub   rsa4096 2017-03-06 [SC]
      5817A43B283DE5A9181A522E1848792F9E2795E9
uid           [  full  ] Qubes OS Release X Signing Key
sig!3        1848792F9E2795E9 2017-03-06  Qubes OS Release X Signing Key
sig!         DDFA1A3E36879494 2017-03-08  Qubes Master Signing Key

gpg: 2 good signatures
```

This is just an example, so the output you receive may not look exactly the same. What matters is the line with a `sig!` prefix showing that the QMSK has signed this key. This verifies the authenticity of the RSK. Note that the `!` flag after the `sig` tag is important because it means that the key signature is valid. A `sig-` prefix would indicate a bad signature, and `sig%` would mean that gpg encountered an error while verifying the signature. It is not necessary to independently verify the authenticity of the RSK, since you already verified the authenticity of the QMSK.

As a final sanity check, make sure the RSK is in your keyring with the correct trust level:

```
$ gpg2 -k "Qubes OS Release"
pub   rsa4096 2017-03-06 [SC]
      5817A43B283DE5A9181A522E1848792F9E2795E9
uid           [  full  ] Qubes OS Release X Signing Key
```

If you don't see the correct RSK here with a trust level of "full" or higher, go back and follow the instructions in this section carefully, and consult the troubleshooting FAQ below.

How to obtain and authenticate other signing keys

Please see the Qubes security pack documentation.

How to verify the cryptographic hash values of Qubes ISOs

There are two ways to verify Qubes ISOs: cryptographic hash values and detached PGP signatures. Both methods are equally secure. Using just one method is sufficient to verify your Qubes ISO. Using both methods is not necessary, but you can do so if you like. One method might be more convenient than another in certain circumstances, so we provide both. This section covers cryptographic hash values. For the other method, see how to verify detached PGP signatures on Qubes ISOs.

Before we proceed, you must first complete the following prerequisite steps:

1. Install OpenPGP software.
2. Import and authenticate the Qubes Master Signing Key.
3. Import and authenticate your release signing key.

Each Qubes ISO is accompanied by a set of **cryptographic hash values** contained in a plain text file ending in `.DIGESTS`, which can find on the downloads page alongside the ISO. This file contains the output of running several different cryptographic hash functions on the ISO (a process known as "hashing") to obtain alphanumeric outputs known as "hash values" or "digests."

One convenient property of hash values is that they can be generated on any computer. This means, for example, that you can download a Qubes ISO on one computer, hash it, then visually compare that hash value to the one you generated or have saved on a different computer.

In addition to the `.DIGESTS` files on the downloads page alongside each ISO, and you can always find all the digest files for every Qubes ISO in the Qubes security pack.

If the filename of your ISO is `Qubes-RX-x86_64.iso`, then the name of the digest file for that ISO is `Qubes-RX-x86_64.iso.DIGESTS`, where `X` is a specific release of Qubes. The digest filename is always the same as the ISO filename followed by `.DIGESTS`. Since the digest file is a plain text file, you can open it with any text editor. Inside, you should find text that looks similar to this:

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA256

3c951138b8b9867d8657f173c1b58b82 *Qubes-RX-x86_64.iso
1fc9508160d7c4cba6cacc3025165b0f996c843f *Qubes-RX-x86_64.iso
6b998045a513dcdd45c1c6e61ace4f1b4e7eff799f381dccb9eb0170c80f678a *Qubes-RX-x86_64.iso
de1eb2e76bdb48559906f6fe344027ece20658d4a7f04ba00d4e40c63723171c62bdcc869375e7a4a4499d7bff484d7a621c3acfe9c2b2
*Qubes-RX-x86_64.iso
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v2

iQIcBAEBCAAGBQJX4XO/AAoJEMsRyh0D+lCCL9sP/jlZ26zhvlDEX/eaA/ANa/6b
Dpsh/sqZEpz1SWoUxdm0gS+anc8nSDoCQSMBxnafuBbmwTChdHI/P7NvNirCULma
9nw+EYCsCiNZ9+WCeroR8XDFSiDjvfkve0R8nwfma1XDqu1bN2ed4n/zNoGgQ8w0
t5LEVDKCVJ+65pI7RzOSMbWaw+uWfGehbgumD7a6rfEOqOTONoZOjJJTnM0+NFJF
Qz5yBg+0FQYc7FmfX+tY801AwSyevj3LKGqZN1GVcU9hhoHH7f2BcbdNk9I5WHHq
doKMnZtcdyadQGwMNB68Wu9+0CWsXvk6E00QfW69M4d6w0gbyoJyUL1uzxgixb5O
qodxrqeitXQSZZvU4kom5zlSjqZs4dGK+Ueplpkr8voT8TSWer0Nbh/VMfrNSt1z
0/j+e/KMjor7XxehR+XhNWa2YLjA5l5H9rP+Ct/LAfVFp4uhsAnYf0rUskhCStxf
Zmtqz4FOw/iSz0Os+IVcnRcyTYWh3e9XaW56b9J/ou0wlwmJ7oJuEikOHBDjrUph
2a8AM+QzNmnc0tDBWTtT2frXcotqL+Evp/kQr5G5pJM/mTR5EQm7+LKSl7yCPoCj
g8JqGYYptgkxjQdX3YAy9VDsCJ/6EkFc2lkQHbgZxjXqyrEMbgeSXtMltZ7cCqw1
3N/6YZw1gSuvBlTquP27
=e9oD
-----END PGP SIGNATURE-----
```

Four digests have been computed for this ISO. The hash functions used, in order from top to bottom, are MD5, SHA-1, SHA-256, and SHA-512. One way to verify that the ISO you downloaded matches any of these hash values is by using the respective `*sum` command:

```
$ md5sum -c Qubes-RX-x86_64.iso.DIGESTS
 Qubes-RX-x86_64.iso: OK
md5sum: WARNING: 23 lines are improperly formatted
$ sha1sum -c Qubes-RX-x86_64.iso.DIGESTS
Qubes-RX-x86_64.iso: OK
sha1sum: WARNING: 23 lines are improperly formatted
$ sha256sum -c Qubes-RX-x86_64.iso.DIGESTS
Qubes-RX-x86_64.iso: OK
sha256sum: WARNING: 23 lines are improperly formatted
$ sha512sum -c Qubes-RX-x86_64.iso.DIGESTS
Qubes-RX-x86_64.iso: OK
sha512sum: WARNING: 23 lines are improperly formatted
```

The `OK` response tells us that the hash value for that particular hash function matches. The program also warns us that there are 23 improperly formatted lines, but this is expected. This is because each file contains lines for several different hash values (as mentioned above), but each `*sum` program verifies only the line for its own hash function. In addition, there are lines for the PGP signature that the `*sum` programs do not know how to read. Therefore, it is safe to ignore these warning lines.

Another way is to use `openssl` to compute each hash value, then compare them to the contents of the digest file:

```
$ openssl dgst -md5 Qubes-RX-x86_64.iso
MD5(Qubes-RX-x86_64.iso)= 3c951138b8b9867d8657f173c1b58b82
$ openssl dgst -sha1 Qubes-RX-x86_64.iso
SHA1(Qubes-RX-x86_64.iso)= 1fc9508160d7c4cba6cacc3025165b0f996c843f
$ openssl dgst -sha256 Qubes-RX-x86_64.iso
SHA256(Qubes-RX-x86_64.iso)= 6b998045a513dcdd45c1c6e61ace4f1b4e7eff799f381dccb9eb0170c80f678a
$ openssl dgst -sha512 Qubes-RX-x86_64.iso
SHA512(Qubes-RX-x86_64.iso)=
de1eb2e76bdb48559906f6fe344027ece20658d4a7f04ba00d4e40c63723171c62bdcc869375e7a4a4499d7bff484d7a621c3acfe9c2b2
```

(Notice that the outputs match the values from the digest file.)

However, it is possible that an attacker replaced `Qubes-RX-x86_64.iso` with a malicious ISO, computed the hash values for that malicious ISO, and replaced the values in `Qubes-RX-x86_64.iso.DIGESTS` with his own set of values. Therefore, we should also verify the authenticity of the listed hash values. Since `Qubes-RX-x86_64.iso.DIGESTS` is a clearsigned PGP file, we can use GPG to verify the signature in the digest file:

```
$ gpg2 -v --verify Qubes-RX-x86_64.iso.DIGESTS
gpg: armor header: Hash: SHA256
gpg: armor header: Version: GnuPG v2
gpg: original file name=''
gpg: Signature made Tue 20 Sep 2016 10:37:03 AM PDT using RSA key ID 03FA5082
gpg: using PGP trust model
gpg: Good signature from "Qubes OS Release X Signing Key"
gpg: textmode signature, digest algorithm SHA256
```

This is just an example, so the output you receive will not look exactly the same. What matters is the line that says `Good signature from "Qubes OS Release X Signing Key"`. This confirms that the signature on the digest file is good.

If you don't see a good signature here, go back and follow the instructions in this section carefully, and consult the troubleshooting FAQ below.

How to verify detached PGP signatures on Qubes ISOs

There are two ways to verify Qubes ISOs: cryptographic hash values and detached PGP signatures. Both methods are equally secure. Using just one method is sufficient to verify your Qubes ISO. Using both methods is not necessary, but you can do so if you like. One method might be more convenient than another in certain circumstances, so we provide both. This section covers detached PGP signatures. For the other method, see how to verify the cryptographic hash values of Qubes ISOs.

Before we proceed, you must first complete the following prerequisite steps:

1. Install OpenPGP software.
2. Import and authenticate the Qubes Master Signing Key.
3. Import and authenticate your release signing key.

Every Qubes ISO is released with a **detached PGP signature** file, which you can find on the downloads page alongside the ISO. If the filename of your ISO is `Qubes-RX-x86_64.iso`, then the name of the signature file for that ISO is `Qubes-RX-x86_64.iso.asc`, where `X` is a specific release of Qubes. The signature filename is always the same as the ISO filename followed by `.asc`.

Download both the ISO and its signature file. Put both of them in the same directory, then navigate to that directory. Now, you can verify the ISO by executing this GPG command in the directory that contains both files:

```
$ gpg2 -v --verify Qubes-RX-x86_64.iso.asc Qubes-RX-x86_64.iso
gpg: armor header: Version: GnuPG v1
gpg: Signature made Tue 08 Mar 2016 07:40:56 PM PST using RSA key ID 03FA5082
gpg: using PGP trust model
gpg: Good signature from "Qubes OS Release X Signing Key"
gpg: binary signature, digest algorithm SHA256
```

This is just an example, so the output you receive will not look exactly the same. What matters is the line that says `Good signature from "Qubes OS Release X Signing Key"`. This confirms that the signature on the ISO is good.

If you don't see a good signature here, go back and follow the instructions in this section carefully, and consult the troubleshooting FAQ below.

This is an optional section intended for advanced users.

After you have authenticated your Qubes ISO and written it onto your desired medium (such as a USB drive or optical disc), you can re-verify the data that has been written to your medium. Why would you want to do this when you've already verified the original ISO? Well, it's conceivable that a sufficiently sophisticated adversary might allow your initial ISO verification to succeed (so as not to alert you that your machine has been compromised, for example), then surreptitiously modify the data as it is being written onto your installation medium, resulting in a compromised Qubes installer. This might increase the odds that the attack goes undetected. One way to mitigate this risk is to re-verify the installer after writing it onto an installation medium that cannot be altered, such as a USB drive with a properly-implemented physical write-protect switch and firmware that is either unflashable or cryptographically-signed (or both), as discussed in our installation security considerations.

This section will walk through an example of re-verifying the installer on such a device. We begin by assuming that you have just written your desired Qubes ISO onto the USB drive. First, unplug your USB drive and flip the write protect switch so that the data on the drive can no longer be altered. If you have a different computer from the one you used to create the installation medium, consider using that computer. If not, try to at least use a fresh VM (e.g., if it's a Qubes system). The idea is that the original machine may have been compromised, and using a different one for re-verification forces your hypothetical adversary to compromise an additional machine in order to succeed.

Now, our goal is to perform the same verification steps as we did with the original ISO, except, this time, we'll be reading the installer data directly from the write-protected USB drive instead of from the original ISO file. First, let's compute the SHA-256 hash value of the data on the drive. (This assumes you're already familiar with how to verify the cryptographic hash values of Qubes ISOs.) In order to do this, we have to know the exact size, in bytes, of the original ISO. There are two ways to get this information: from the ISO itself and from the Qubes website. Here's an example of the first way:

```
$ dd if=/dev/sdX bs=1M count=$(stat -c %s /path/to/iso) iflag=count_bytes | sha256sum
```

(Where `/dev/sdX` is your USB drive and `/path/to/iso` is the path to your Qubes ISO.)

This command reads exactly the number of bytes of your Qubes ISO (obtained with `stat -c %s /path/to/iso`) from the USB drive and pipes them into `sha256sum`. The output should look something like this:

```
0e68dd3347b68618d9e5f3ddb580bf7ecdd2166747630859b3582803f1ca8801  -
5523+0 records in
```

```
5523+0 records out
5791285248 bytes (5.8 GB, 5.4 GiB) copied, 76.3369 s, 75.9 MB/s
```

Note that your actual SHA-256 hash value and byte number will depend on which Qubes ISO you're using. This is just an example. Your SHA-256 hash value should match the hash value of your genuine original Qubes ISO.

Now, reading the number of bytes directly from the ISO is fine, but you may be concerned that a sufficiently sophisticated adversary may have compromised the machine on which you're performing this re-verification and may therefore be capable of feeding you a false success result. After all, if your adversary knows the answer you're looking for — namely, a match to the genuine ISO — and has access to that very ISO in the same re-verification environment, then there is little to prevent him from simply hashing the original ISO and feeding you that result (perhaps while also reading from the USB drive and piping it into `/dev/null` so that you see the light on the USB drive blinking to support the illusion that the data is being read from the USB drive).

Therefore, in order to make things a bit more difficult for your hypothetical adversary, you may instead wish to perform the re-verification in an environment that has never seen the original ISO, e.g., a separate offline computer or a fresh VM the storage space of which is too small to hold the ISO. (Note: If you're doing this in Qubes, you can attach the block device from sys-usb to a separate new qube. You don't have to perform the re-verification directly in sys-usb.) In that case, you'll have to obtain the size of the ISO in bytes and enter it into the above command manually. You can, of course, obtain the size by simply using the `stat -c %s /path/to/iso` command from above on the machine that has the ISO. You can also obtain it from the Qubes website by hovering over any ISO download button on the downloads page. (You can also view these values directly in the downloads page's source data.) Once you have the exact size of the ISO in bytes, simply insert it into the same command, for example:

```
$ dd if=/dev/sdX bs=1M count=5791285248 iflag=count_bytes | sha256sum
```

If you wish to compute the values of other hash functions, you can replace `sha256sum`, e.g., with `md5sum`, `sha1sum`, or `sha512sum`.

In addition to checking hash values, you can also use GnuPG to verify the detached PGP signature directly against the data on the USB drive. (This assumes you're already familiar with how to verify detached PGP signatures on Qubes ISOs.)

```
$ dd if=/dev/sdX bs=1M count=<ISO_SIZE> iflag=count_bytes | gpg -v --verify Qubes-RX-x86_64.iso.asc -
gpg: Signature made Thu 14 Jul 2022 08:49:38 PM PDT
gpg:                using RSA key 5817A43B283DE5A9181A522E1848792F9E2795E9
gpg: using pgp trust model
gpg: Good signature from "Qubes OS Release X Signing Key" [full]
gpg: binary signature, digest algorithm SHA256, key algorithm rsa4096
5523+0 records in
5523+0 records out
5791285248 bytes (5.8 GB, 5.4 GiB) copied, 76.6013 s, 75.6 MB/s
```

(Where `/dev/sdX` is your USB drive, `<ISO_SIZE>` is the size of the original ISO in bytes, and `Qubes-RX-x86_64.iso.asc` is the detached signature file of the original ISO.)

This command reads the exact number of bytes from your USB drive as the size of the original ISO and pipes them into gpg. The usual form of a gpg verification command is `gpg --verify <SIGNATURE> <SIGNED_DATA>`. Our command is using shell redirection in order to use data from your USB drive as the `<SIGNED_DATA>`, which is why the `-` at the end of the command is required. Remember that you still must have properly imported and trusted the QMSK and appropriate RSK in order for this to work. You should receive a `Good signature` message for the appropriate RSK, which should be signed by a copy of the QMSK that you previously confirmed to be genuine.

Before we proceed, you must first complete the following prerequisite steps:

1. Install OpenPGP software.
2. Import and authenticate the Qubes Master Signing Key.
3. Import and authenticate keys from the Qubes security pack (qubes-secpack). Please see our PGP key policies for important information about these keys.

Whenever you use one of the Qubes repositories, you should use Git to verify the PGP signature in a tag on the latest commit or on the latest commit itself. (One or both may be present, but only one is required.) If there is no trusted signed tag or commit on top, any commits after the latest trusted signed tag or commit should **not** be trusted. If you come across a repo with any unsigned commits, you should not add any of your own signed tags or commits on top of them unless you personally vouch for the trustworthiness of the unsigned commits. Instead, ask the person who pushed the unsigned commits to sign them.

You should always perform this verification on a trusted local machine with properly authenticated keys rather than relying on a third party, such as GitHub. While the GitHub interface may claim that a commit has a verified signature from a member of the Qubes team, this is only trustworthy if GitHub has performed the signature check correctly, the account identity is authentic, an admin has not replaced the user's key, GitHub's servers have not been compromised, and so on. Since there's no way for you to be certain that all such conditions hold, you're much better off verifying signatures yourself. (Also see: distrusting the infrastructure.)

How to verify a signature on a Git tag

or

```
$ git verify-tag <tag name>
```

How to verify a signature on a Git commit

```
$ git log --show-signature <commit ID>
```

or

```
$ git verify-commit <commit ID>
```

Troubleshooting FAQ

Why am I getting “Can’t check signature: public key not found”?

You don’t have the correct release signing key.

Why am I getting “BAD signature from ‘Qubes OS Release X Signing Key’”?

The problem could be one or more of the following:

- You’re trying to verify the wrong file(s). Reread this page carefully.
- You’re using the wrong GPG command. Follow the provided examples carefully, or try using gpg instead of gpg2 (or vice versa).
- The ISO or detached PGP signature file is bad (e.g., incomplete or corrupt download). Try downloading the signature file again from a different source, then try verifying again. If you still get the same result, try downloading the ISO again from a different source, then try verifying again.

Why am I getting “bash: gpg2: command not found”?

You don’t have gpg2 installed. Please install it using the method appropriate for your environment (e.g., via your package manager), or try using gpg instead.

Why am I getting “No such file or directory”?

Your working directory does not contain the required files. Go back and follow the instructions more carefully, making sure that you put all required files in the same directory *and* navigate to that directory.

Why am I getting “can’t open signed data ‘Qubes-RX-x86_64.iso’ / can’t hash datafile: file open error”?

The correct ISO is not in your working directory.

Why am I getting “can’t open ‘Qubes-RX-x86_64.iso.asc’ / verify signatures failed: file open error”?

The correct detached PGP signature file is not in your working directory.

Why am I getting “no valid OpenPGP data found”?

Either you don’t have the correct detached PGP signature file, or you inverted the arguments to gpg2. (The signature file goes first.)

Why am I getting “WARNING: This key is not certified with a trusted signature! There is no indication that the signature belongs to the owner.”?

There are several possibilities:

- You don’t have the Qubes Master Signing Key.
- You have not set the Qubes Master Signing Key’s trust level correctly.
- In the case of a key that is not directly signed by the Qubes Master Signing Key, you have not set that key’s trust level correctly.

Why am I getting “X signature not checked due to a missing key”?

You don’t have the keys that created those signatures in your keyring. For the purpose of verifying a Qubes ISO, you don’t need them as long as you have the Qubes Master Signing Key and the release signing key for your Qubes release.

Why am I seeing additional signatures on a key with “[User ID not found]” or from a revoked key?

This is just a fundamental part of how OpenPGP works. Anyone can sign anyone else’s public key and upload the signed public key to keyservers. Everyone is also free to revoke their own keys at any time (assuming they possess or can create a revocation certificate). This has no impact on verifying Qubes ISOs, code, or keys.

Why am I getting “verify signatures failed: unexpected data”?

You’re not verifying against the correct detached PGP signature file.

Why am I getting “not a detached signature”?

You’re not verifying against the correct detached PGP signature file.

Why am I getting “CRC error; […] no signature found […]”?

You’re not verifying against the correct detached PGP signature file, or the signature file has been modified. Try downloading it again or from a different source.

Do I have to verify both the detached PGP signature file and the cryptographic hash values?

No, either method is sufficient by itself, but you can do both if you like.

Why am I getting “no properly formatted X checksum lines found”?

You’re not checking the correct cryptographic hash values.

Why am I getting “WARNING: X lines are improperly formatted”?

Read how to verify the cryptographic hash values of Qubes ISOs again.

Why am I getting “WARNING: 1 listed file could not be read”?

The correct ISO is not in your working directory.

I have another problem that isn’t mentioned here.

Carefully reread this page to be certain that you didn’t skip any steps. In particular, make sure you have the Qubes Master Signing Key, the release signing key for your Qubes release, *and* the cryptographic hash values and/or detached PGP signature file, all for the *correct* Qubes OS release. If your question is about GPG, please see the GnuPG documentation. Still have question? Please see help, support, mailing lists, and forum for places where you can ask!