INFORMATION SECURITY

Data too large for key size while trying to sign then encrypt

Asked 4 years, 1 month ago Modified 4 years, 1 month ago Viewed 20k times

3

I'm not very experienced in the security field, but I'm trying to sign then encrypt a large folder in order to deal with software updates in some machines that will be offline. I've thought about generatic a key pair which my company will keep in secret and will be deployed in each of the machines. I've used this commands in order to generate the key pair:

```
openssl genrsa -des3 -out private.pem 2048
```

Then:

```
openssl rsa -in private.pem -outform PEM -pubout -out public.pem
```

Now I'm writing one script in order to zip one folder, use aes-256 symmetric encryption with a random password over it and then sign and encrypt the password using my newly generated keys:

```
#!/bin/bash

zip update.zip update/*
#Generate a random password
openssl rand -hex 16 > symmetric_keyfile.key
#Encrypt the file with the random password
openssl aes-256-cbc -a -salt -k symmetric_keyfile.key -in update.zip -out
update.zip.enc
rm update.zip
#Sign the password with the private key
echo 'Signing...'
openssl rsautl -sign -inkey private.pem -in symmetric_keyfile.key -out
symmetric_keyfile.key.sign
#Encrypt the password with the public key
echo 'Encrypting...'
openssl rsautl -encrypt -inkey public.pem -pubin -in symmetric_keyfile.key.sign
-out symmetric_keyfile.key.enc
#Zip both files
zip update.zip symmetric_keyfile.key.enc update.zip.enc
```

However I've got this error while encrypting:

```
adding: update/deploy.zip (stored 0%)
Signing...
Enter pass phrase for private.pem:
Encrypting...
RSA operation error
140035711579800:error:0406D06E:rsa routines:RSA_padding_add_PKCS1_type_2:data
too large for key size:rsa_pk1.c:153:
```

I guess it's related with the signed key being to large to be encrypted. The size of it is 256 bytes. Any ideas?

openssl aes rsa

Share Improve this question Follow

asked Sep 27, 2019 at 8:42

Aritz
131 1 1 5

1 It's highly unlikely you need a 256 bit AES key. 128 bit AES is almost certainly enough, unless you need to protect your secrets for... let's say more than 30-50 years, and believe quantum computers will have massive advancements. Either use a 128 bit key, or increase your RSA key size (which will slow things down a bit). In addition, RSA is far more vulnerable to massive advancements of quantum computing than AES is. Though honestly, these are (likely) far future threats anyway. – Steve Sether Sep 27, 2019 at 14:15

@SteveSether FYI, despite the '128' in the name, AES-128 does not attain what is widely considered to be a '128-bit security level', meaning that the cost of an attack is at least the cost of about 2^128 bit operations. This is because there is a linear batch advantage to breaking *the first of many keys* so that, for example, if you have a billion target keys, it costs well under 2^100 to find the first one. And once you get a foothold in a network, it is often easy to use that to advance further into the network. See cr.yp.to/papers.html#bruteforce for details. – Squeamish Ossifrage Nov 10, 2019 at 4:05

Relevant question on crypto.SE about AES-256 and RSA-2048: crypto.stackexchange.com/q/67295 – Squeamish Ossifrage Nov 10, 2019 at 4:08

4 Answers

Sorted by: Highest score (default)

2

Based on this comment you can encrypt at most 214 bytes using 2048 bits RSA key.

When i run the same operation you did, i get the following:

```
$ ls -lah
total 44K
drwxr-xr-x  3 my_user my_user 4.0K Sep 27 10:50 .
drwxrwxrwt 20 root       root       4.0K Sep 27 10:49 ..
-rw-r--r--  1 my_user my_user  658 Sep 27 10:49 pouet.sh
-rw-------  1 my_user my_user 1.8K Sep 27 10:49 private.pem
-rw-r--r--  1 my_user my_user  451 Sep 27 10:50 public.pem
-rw-r--r--  1 my_user my_user   33 Sep 27 10:50 symmetric_keyfile.key
-rw-r--r--  1 my_user my_user    0 Sep 27 10:50 symmetric_keyfile.key.enc
-rw-r--r--  1 my_user my_user  256 Sep 27 10:50 symmetric_keyfile.key.sign
drwxr-xr-x  9 my_user my_user 4.0K Sep 27 10:49 update
-rw-r--r--  1 my_user my_user 3.5K Sep 27 10:50 update.zip
-rw-r--r--  1 my_user my_user 4.2K Sep 27 10:50 update.zip.enc
```

Since you're trying to encrypt `symmetric_keyfile.key.sign` which is 256 bytes, the failure is quite normal. Given my poor skills regarding crypto, i would suggest that you do not sign your key before encrypting. In my opinion, this step is useless...

Share Improve this answer Follow

answered Sep 27, 2019 at 8:54
binarym
754 4 8

Thanks for your answer! The given link explains my issue, however, I don't want to get rid of the signature, as signing the key allows us to verify our company has created the artifact, actually. – Aritz Sep 27, 2019 at 9:04

Maybe you can sign **after** encryption ? Why don't you use `gpg` to do this work ? It will perfectly do the job ... – binarym Sep 27, 2019 at 9:09

2

You are using `openssl` primitives to put together an encrypt/sign chain.

Doing so is only a tiny step above implementing your own cryptography.

Instead of using `openssl` to sign and encrypt, consider using a well-established tool that is actually designed to do so *securely*.

Here's an example of how you can use GnuPG (with symmetric encryption, but can be adapted for asymmetric encryption as well) to accomplish something similar, slightly adapted from a script I'm using:

```
ENCRYPTIONKEY=$(openssl rand -hex 32)
tar cfj - update/* | \
gpg -c --batch --yes -z 0 --cipher-algo AES256 -o update.tar.bz2.gpg \
  --passphrase-fd 9s 9< <(printf '%s' "$ENCRYPTIONKEY") -
```

(You'll also note that I used 32 hex digits for the passphrase rather than 16, to get 256 bits of security for AES-256 rather than your 128 bits.)

The above will create a symmetrically encrypted OpenPGP file, using a randomly generated passphrase, where the encrypted file itself consists of a compressed `tar` archive. You will need to transmit the passphrase separately, possibly stored as an asymmetrically encrypted file along with it; something like:

```
printf '%s\n' "$ENCRYPTIONKEY" | \
gpg --output update.tar.bz2.pass.gpg --encrypt --sign --recipient $rcptkeyid \
  --passphrase-fd 9s 9< <(printf '%s' "$SIGNPASS")
zip -0 update.zip update.tar.bz2.gpg update.tar.bz2.pass.gpg
```

The above is by no means guaranteed to be secure against every adversary, but it should be better than something homegrown using only cryptographic primitives (and doesn't have the size limitations inherent from using raw RSA). It also has the side benefit that the actual passphrase used to encrypt the payload data (possibly absent unencrypted swap) is never written to any kind of persistent storage. It's also not tested, so may need slight tweaking to work properly.

Share Improve this answer Follow

answered Sep 27, 2019 at 13:58

user
7,755 2 31 56

0

Another option that you have with big files, encrypted or not, is to generate the sha/blake hash of the file and then you generate the signature of the generated hash.

Share Improve this answer Follow

answered Sep 27, 2019 at 10:21

camp0

2,257 1 12 10

0

I think I misunderstood the signature concept when trying to sign the aes key itself. I changed the script so as to sign the whole file to be encrypted before with the dgst tool:

```
#!/bin/bash

zip update.zip update/*
#Hash and sign with the private key
echo 'Signing...'
openssl dgst -sha256 -sign private.pem -out update.zip.sha update.zip
#Generate a random password
openssl rand -hex 16 > symmetric_keyfile.key
#Encrypt the file with the random password
openssl aes-256-cbc -a -salt -k symmetric_keyfile.key -in update.zip -out
update.zip.enc
rm update.zip
#Encrypt the password with the public key
echo 'Encrypting...'
openssl rsautl -encrypt -inkey public.pem -pubin -in symmetric_keyfile.key -out
symmetric_keyfile.key.enc
#Zip all the files
zip update.zip symmetric_keyfile.key.enc update.zip.enc update.zip.sha
```

Share Improve this answer Follow

answered Sep 27, 2019 at 10:25

Aritz

131 1 1 5