Practical Public Key Cryptography

Originalartikel

Backup

<html> Encryption is one of the pillars of modern-day communications. You have devices that use encryption all the time, even if you are not aware of it. There are so many applications and systems using it that it’s hard to begin enumerating them. Ranging from satellite television to your mobile phone, from smart power meters to your car keys, from your wireless router to your browser, and from your Visa to your Bitcoins — the list is endless. One of the great breakthroughs in the history of encryption was the invention of public key cryptography or asymmetrical cryptography in the 70’s. For centuries traditional cryptography methods were used, where some secret key or scheme had to be agreed and shared between the sender and the receiver of an encrypted message. Asymmetric cryptography changed that. Today you can send an encrypted message to anyone. This is accomplished by the use of a pair of keys: one public key and one private key. The key properties are such that when something is encrypted with the public key, only the private key can decrypt it and vice-versa. In practice, this is usually implemented based on mathematical problems that admit no efficient solution like certain integer factorization, discrete logarithm and elliptic curve relationships. But the game changer is that the public key doesn’t have to be kept secret. This allows cryptography to be used for authentication — proving who someone is — as well as for encryption, without requiring you to have previously exchanged secrets. In this article, I’ll get into the details of how to set yourself up so that anyone in the world is able to send you an e-mail that only you can read.
 <h2>Public Key Cryptography in a Nutshell</h2> But first, how does it work in theory? Let’s say that Alice wants to talk to Bob. (Yes, it’s Alice and Bob again.) Alice and Bob generate their respective pair of keys. They tell the whole world about their public keys, including one another. Alice can now use Bob’s public key to encrypt a message that only Bob can read — only Bob’s private key can decrypt the message and, as the name implies, it should be kept private and known only to Bob. Imagine Alice wants to send the world (or Bob) an important message, and prove that it comes from her? In the times we live in, how could we make sure the message Alice claims to have sent is not a fake? Well, Alice uses her own private key to encrypt her important message. The world just has to use Alice public key to decrypt the message, since it is the only way to decrypt it and only the person with Alice private key could have encrypted it, hence proving that it was Alice that wrote that message. <h2>Let’s Get Alice Started</h2> So, assuming Alice has GPG installed, she would start by creating her own public/private key pair: <pre class=„brush: bash; title: ; notranslate“ title=„“>

 alice@wonderland ~ $ gpg –gen-key
 gpg (GnuPG) 1.4.20; Copyright (C) 2015 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)

 Requested keysize is 2048 bits
 Please specify how long the key should be valid.
 0 = key does not expire
 <n> = key expires in n days
 <n>w = key expires in n weeks
 <n>m = key expires in n months
 <n>y = key expires in n years
 Key is valid for? (0)
 Key does not expire at all
 Is this correct? (y/N) Y

 You need a user ID to identify your key; the software constructs the user ID
 from the Real Name, Comment and Email Address in this form:
 „Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>“

 Real name: Alice
 Email address: alice@wonderland.xyz
 Comment: Mushroom
 You

selected this USER-ID:
 „Alice (Mushroom) <alice@wonderland.xyz>“

 Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
 You need a Passphrase to protect your secret key.
 …
 public and secret key created and signed.

 pub 2048R/96FE8CE5 2017-10-13
 Key fingerprint = B5FF 1BE3 4502 F425 A444 D6EC FBEC 78AE 96FE 8CE5
 uid Alice (Mushroom) <alice@wonderland.xyz>
 sub 2048R/76E5C437 2017-10-13
 </pre> Depending on Alice’s needs, setting up key expiration might be a good idea. After this command, it is a good idea to export and store a copy of the keys somewhere safe, ideally with redundancy. She can, for example, store a copy in a USB drive and print a hardcopy and put it in a safe. <pre class=„brush: bash; title: ; notranslate“ title=„“>

 alice@wonderland ~ $ gpg –export-secret-key alice@wonderland.xyz > ./alice-gpg-backup.gpg

 alice@wonderland ~ $ gpg –export-secret-key -a alice@wonderland.xyz
 —–BEGIN PGP PRIVATE KEY BLOCK—–
 Version: GnuPG v1

 lQPGBFngvxUBCADNVg9j2iNv3FuzscUOe9oZqP0xCk8p9s+ApIDTqD6vZFkXLpYs…
 —–END PGP PRIVATE KEY BLOCK—–
 </pre> The above commands perform a backup and outputs an ASCII version of the private key, suitable for printing. The private key implementation of OpenPGP actually contains a complete copy of the public key, so this backup file is enough to recover the key pair. Restoring the backup is done with the

```
--import
```

flag. <h2>Tell The World Who You Are</h2> Now that Alice has her key, it’s time to tell the world about it. There are several ways to do it. On the command line: <pre class=„brush: bash; title: ; notranslate“ title=„“>

 alice@wonderland ~ $ gpg –list-public-keys
 /home/alice/.gnupg/pubring.gpg
 ————————————
 pub 2048R/96FE8CE5 2017-10-13
 uid Alice (Mushroom) <alice@wonderland.xyz>
 sub 2048R/76E5C437 2017-10-13

 alice@wonderland ~ $ gpg –send-keys 96FE8CE5
 gpg: sending key 96FE8CE5 to hkp server keys.gnupg.net

 alice@wonderland ~ $ gpg –search-keys 96FE8CE5
 gpg: searching for „0x96FE8CE5“ from hkp server keys.gnupg.net
 (1) Alice (Mushroom) <alice@wonderland.xyz>
 2048 bit RSA key 96FE8CE5, created: 2017-10-13
 Keys 1-1 of 1 for „0x96FE8CE5“. Enter number(s), N)ext, or Q)uit > q

 </pre> This way, Alice just published her key in a known OpenPGP Key server, accessible to anyone. (

```
keys.gnupg.net
```

is the default on my distro.) There are several key servers available to publish public keys, some of them are synchronized. I like to use

```
pgp.mit.edu
```

. Another option is to export the public key with the

```
--export
```

flag and sent it via email and/or manually publish it in several servers — the more the better. Now Alice can digitally sign her messages and receive encrypted messages directed to her. Lets see an example: <pre class=„brush: bash; title: ; notranslate“ title=„“>

 alice@wonderland ~ $ echo „This is a test file“ > file.txt
 alice@wonderland ~ $ gpg –detach-sign file.txt

 You need a passphrase to unlock the secret key for
 user: „Alice

(Mushroom) <alice@wonderland.xyz>“
 2048-bit RSA key, ID 96FE8CE5, created 2017-10-13

 alice@wonderland ~ $ ls -l file*
 -rw-rw-r– 1 alice alice 37 Out 13 15:56 file.txt
 -rw-rw-r– 1 alice alice 287 Out 13 15:56 file.txt.sig

 alice@wonderland ~ $ gpg –verify file.txt.sig
 gpg: assuming signed data in `file.txt' gpg: Signature made Sex 13 Out 2017 15:56:23 WEST using RSA key ID 96FE8CE5 gpg: Good signature from „Alice (Mushroom) <alice@wonderland.xyz>“  alice@wonderland ~ $ echo „The file contents have been tampered“ > file.txt  alice@wonderland ~ $ gpg –verify file.txt.sig  gpg: assuming signed data in `file.txt'
 gpg: Signature made 13 Out 2017 15:56:23 WEST using RSA key ID 96FE8CE5
 gpg: BAD signature from „Alice (Mushroom) <alice@wonderland.xyz>“
 </pre> When Alice wishes to shared

```
file.txt
```

she also shares

```
file.txt.sig
```

so that anyone can verify her signature. So Alice can sign but before she can send Bob a message, Bob must also have a key pair and publish his public key somewhere or sent it to Alice. Lets assume Bob already did it and Alice imported Bob’s public key into GPG, either with the

```
--import
```

flag or

```
--recv-keys
```

from a server. If Alice wants to send Bob a message she would issue the following commands: <pre class=„brush: bash; title: ; notranslate“ title=„“>

 alice@wonderland ~ $ gpg –import bob.asc
 gpg: key 81DBD5F6: public key „Robert (Nope) <bob@whatdoesthebobsay.xyz>“ imported
 gpg: Total number processed: 1
 gpg: imported: 1 (RSA: 1)

 alice@wonderland ~ $ gpg –list-public-keys
 /home/alice/.gnupg/pubring.gpg
 ————————————
 pub 2048R/96FE8CE5 2017-10-13
 uid Alice (Mushroom) <alice@wonderland.xyz>
 sub 2048R/76E5C437 2017-10-13

 pub 2048R/81DBD5F6 2017-10-13
 uid Robert (Nope) <bob@whatdoesthebobsay.xyz>
 sub 2048R/21B662BE 2017-10-13

 alice@wonderland ~ $ echo „This is a secret message to Bob“ > message.txt
 alice@wonderland ~ $ ls -l message*
 -rw-rw-r– 1 alice alice 32 Out 13 16:25 message.txt
 alice@wonderland ~ $ gpg -r bob@whatdoesthebobsay.xyz –sign –encrypt message.txt

 You need a passphrase to unlock the secret key for
 user: „Alice (Mushroom) <alice@wonderland.xyz>“
 2048-bit RSA key, ID 96FE8CE5, created 2017-10-13

 gpg: gpg-agent is not available in this session
 gpg: 21B662BE: There is no assurance this key belongs to the named user

 pub 2048R/21B662BE 2017-10-13 Robert (Nope) <bob@whatdoesthebobsay.xyz>
 Primary key fingerprint: 1558 11B0 C87D 0E02 1C8B 304F 4982 D1D3 81DB D5F6
 Subkey fingerprint: 6CC7 BC9C D69E 9465 78E4 53E3 A931 7A64 21B6 62BE

 It is NOT certain that the key belongs to the person named
 in the user ID. If you *really* know what you are doing,
 you may answer the next question with yes.

 Use this key anyway? (y/N) y
 alice@wonderland ~ $ ls -l message*
 -rw-rw-r– 1 alice alice 32 Out 13 16:25 message.txt
 -rw-rw-r– 1 alice alice 678 Out 13 16:25 message.txt.gpg

 </pre> This would encrypt

```
message.txt
```

with Bob’s public key and sign the file using Alice private key. Now Alice can send Bob the file

```
message.txt.gpg
```

and not only is Bob the only person able to decrypt it, he can also verify that it came from Alice as long as he has Alice’s public key. Let’s see what Bob would have to do: <pre class=„brush: bash; title: ; notranslate“ title=„“>

 bob@whatdoesthebobsay ~ $ gpg –import alice-public-key.asc
 gpg: key 96FE8CE5: public key „Alice (Mushroom) <alice@wonderland.xyz>“ imported
 gpg: Total number processed: 1
 gpg: imported: 1 (RSA: 1)

 bob@whatdoesthebobsay ~ $ gpg –decrypt message.txt.gpg

 You need a passphrase to unlock the secret key for
 user: „Robert (Nope) <bob@whatdoesthebobsay.xyz>“
 2048-bit RSA key, ID 21B662BE, created 2017-10-13 (main key ID 81DBD5F6)

 gpg: encrypted with 2048-bit RSA key, ID 21B662BE, created 2017-10-13
 „Robert (Nope) <bob@whatdoesthebobsay.xyz>“

 This is a secret message to Bob

 gpg: Signature made Sex 13 Out 2017 16:19:51 WEST using RSA key ID 96FE8CE5
 gpg: Good signature from „Alice (Mushroom) <alice@wonderland.xyz>“
 gpg: WARNING: This key is not certified with a trusted signature!
 gpg: There is no indication that the signature belongs to the owner.
 Primary key fingerprint: B5FF 1BE3 4502 F425 A444 D6EC FBEC 78AE 96FE 8CE5

 </pre> <h2>Who Can You Trust?</h2> You might have notice the WARNING message saying that Alice key is not certified with a trusted signature and there is no indication that the signature belongs to her. If Bob knows for sure that Alice key is from Alice, he can sign it and GPG will see it as trusted. <pre class=„brush: bash; title: ; notranslate“ title=„“>

 bob@whatdoesthebobsay ~ $ gpg –edit-key alice@wonderland.xyz sign

 pub 2048R/96FE8CE5 created: 2017-10-13 expires: never usage: SC
 trust: undefined validity: unknown
 sub 2048R/76E5C437 created: 2017-10-13 expires: never usage: E
 [unknown] (1). Alice (Mushroom) <alice@wonderland.xyz>

 pub 2048R/96FE8CE5 created: 2017-10-13 expires: never usage: SC
 trust: undefined validity: unknown
 Primary key fingerprint: B5FF 1BE3 4502 F425 A444 D6EC FBEC 78AE 96FE 8CE5

 Alice (Mushroom) <alice@wonderland.xyz>

 Are you sure that you want to sign this key with your
 key „Robert (Nope) <bob@whatdoesthebobsay.xyz>“ (81DBD5F6)

 Really sign? (y/N) y

 You need a passphrase to unlock the secret key for
 user: „Robert (Nope) <bob@whatdoesthebobsay.xyz>“
 2048-bit RSA key, ID 81DBD5F6, created 2017-10-13

 gpg> save
 bob@whatdoesthebobsay ~ $ gpg –decrypt message.txt.gpg

 You need a passphrase to unlock the secret key for
 user: „Robert (Nope) <bob@whatdoesthebobsay.xyz>“
 2048-bit RSA key, ID 21B662BE, created 2017-10-13 (main key ID 81DBD5F6)

 gpg: encrypted with 2048-bit RSA key, ID 21B662BE, created 2017-10-13
 „Robert (Nope) <bob@whatdoesthebobsay.xyz>“
 This is a secret message to Bob
 gpg: Signature made Sex 13 Out 2017 16:19:51 WEST using RSA key ID 96FE8CE5
 gpg: checking the trustdb
 gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
 gpg: depth: 0 valid: 8 signed: 2 trust: 0-, 0q, 0n, 0m, 0f, 8u
 gpg: depth: 1 valid: 2 signed: 0 trust: 1-, 1q, 0n, 0m, 0f, 0u
 gpg: next trustdb check due at 2019-03-19
 gpg: Good signature from „Alice (Mushroom) <alice@wonderland.xyz>“

 </pre> The warning will disappear. But why was there a warning? So far, someone sent an email claiming to have Alice’s public key, but how does Bob know it is the Alice that he knows? Or that the initial public key exchange was not intercepted? Even though public key cryptography eliminated the need to distribute secret keys, public keys have to be distributed to others with whom they want to communicate, and if the

encryption is also used for authentication, the provenance of the public keys is important. So what are Bob’s options? Bob can meet Alice face to face where she guarantees him that her key (or her key fingerprint) is correct, and Bob can mark it as trusted by signing it as in the above example. Sometimes it is impossible to meet face to face or you are talking with someone you don’t actually know. Bob can alternately choose to trust a particular key server, which is usually quite secure. If Bob does not want to choose to trust any key server, there is another way. <h3>The Web of Trust</h3> GnuPG addresses this problem with a mechanism known as the web of trust. In the web of trust model, responsibility for validating public keys is delegated to people you trust. This is different from normal public key infrastructure (PKI) approach since PKI permits each certificate to be signed only by a single party: a certificate authority (CA). By contrast, OpenPGP identity certificates (which include public key(s) and owner information) can be digitally signed by other users who, by signing, acknowledge the association of that public key with the person listed in the certificate. There are even events, known as key signing parties, where users gather with their keys and identity documents and sign each other keys. Basically, as more people sign a key, the more sure you can be that the key is from who it claims to be. <h2>I Take It All Back</h2> On a last note, it might be a good idea for Alice to create a revocation certificate. A revocation certificate can be used if and when Alice loses her key or she believes the key was compromised. It is published to notify others that the public key should no longer be used. A revoked public key can still be used to verify signatures made by Alice in the past, but it cannot be used to encrypt future messages to Alice. It also does not affect the ability to decrypt messages sent to Alice in the past if she still has the key. <pre class=„brush: bash; title: ; notranslate“ title=„“>

 alice@wonderland ~ $ gpg -a –gen-revoke alice@wonderland.xyz

 sec 2048R/96FE8CE5 2017-10-13 Alice (Mushroom) <alice@wonderland.xyz>

 Create a revocation certificate for this key? (y/N) y
 Please select the reason for the revocation:
 0 = No reason specified
 1 = Key has been compromised
 2 = Key is superseded
 3 = Key is no longer used
 Q = Cancel
 (Probably you want to select 1 here)
 Your decision? 0
 Enter an optional description; end it with an empty line:
 > revoke!
 >
 Reason for revocation: No reason specified
 revoke!
 Is this okay? (y/N) y

 You need a passphrase to unlock the secret key for
 user: „Alice (Mushroom) <alice@wonderland.xyz>“
 2048-bit RSA key, ID 96FE8CE5, created 2017-10-13

 Revocation certificate created.

 Please move it to a medium which you can hide away; if Mallory gets
 access to this certificate he can use it to make your key unusable.
 As with the private key, it is smart to print this certificate and store it away, just in case
 your media become unreadable. But have some caution: the print system of
 your machine might store the data and make it available to others!

 —–BEGIN PGP PUBLIC KEY BLOCK—–
 Version: GnuPG v1
 Comment: A revocation certificate should follow

 iQEmBCABAgAQBQJZ5N6XCR0AcmV2b2tlIQAKCRD77Hiulv6M5XnuB/41jjJCVx/S
 …
 —–END PGP PUBLIC KEY BLOCK—–

 </pre> The above code demonstrates how to generate a revocation certificate. <h2>“This is too confusing!”</h2> I get that. I do. Users don’t want to be encrypting stuff via the command line and worry about certificates and trust models. It might seem a bit overwhelming at first but luckily there are easy to use software solutions available to the end-user that solve your basic public key cryptography needs. It’s an extensive list and I’m pretty sure our readers have plenty of suggestions depending on your internet use and operating system. Let’s hear them in the comments! For example, to secure emails, there are several options. If you use Thunderbird, you can install Enigmail. It should work on Linux, Windows, Mac and SunOS/Solaris. For Outlook users, look for gpg4o or gpg4win. Apple Mail has GPGTools. Android has K9 and R2Mail2 while iOS has iPGMail. If you don’t use a specific email client but use webmail instead, you are not left out, you can try

Mailvelope. Mailvelope is an extension for Chrome and Firefox that implements OpenPGP and works over your regular webmail client. Regarding email encryption, you can see that there is a lot to choose from. There is probably a OpenPGP implementation available to your email client of choice, it’s a matter of choosing a solution for a software that you are already accustomed and start from there. So if you don’t use encryption yet, what is your excuse now? Go and create your key pair, share it with the world! And do it now, before the next revolution in cryptography, quantum computing, kills it all. </html>