Professional Issues Cryptoparty

William Waites

School of Informatics
&
Open Rights Group

October 26th, 2015

School of informatics

OPEN RIGHTS GROUP

While I'm talking, get Tails onto your USB Drive

If you have a computer with a DVD drive and a USB drive

1. Grab a Tails DVD from the spindle and boot from it
2. Install Tails onto your USB drive

If you have a USB stick but no DVD drive

1. Borrow someone's USB drive with Tails on it
2. Install from there onto your USB drive

Installing Tails is simple

Just two steps:

1. Applications → Tails → Tails Installer
2. Clone & Install

Careful to install onto the right disk!

If you didn't bring a USB drive

Install the Tor Browser Bundle from:

https://www.torproject.org/projects/torbrowser.html

What we'll do today

- Brief introduction to public key cryptography
- Learn how to use the GNU Privacy Guard
- Learn about the Tor Project and *hidden services*
  ... also what Tor does *not* do
- Try them out together with the InsecureDrop service

Symmetric Cryptography

Introducing Alice, Bob and Eve...

- Alice wants to send a message to Bob ...
- ... and doesn't want Eve to overhear
- *If* Alice and Bob share some secret information ...
- *then* they can use simple operations like $\otimes$ (XOR) ...
- ... or more complicated ones like Rijndael (aka AES)
- ... and Eve will not be able to eavesdrop

Symmetric Cryptography

Introducing Alice, Bob and Eve...

- Alice wants to send a message to Bob ...
- ... and doesn't want Eve to overhear
- *If* Alice and Bob share some secret information ...
- *then* they can use simple operations like $\otimes$ (XOR) ...
- ... or more complicated ones like Rijndael (aka AES)
- ... and Eve will not be able to eavesdrop

Symmetric Cryptography

Introducing Alice, Bob and Eve...

- Alice wants to send a message to Bob ...
- ... and doesn't want Eve to overhear
- *If* Alice and Bob share some secret information ...
- *then* they can use simple operations like $\otimes$ (XOR) ...
- ... or more complicated ones like Rijndael (aka AES)
- ... and Eve will not be able to eavesdrop

Symmetric Cryptography

Introducing Alice, Bob and Eve...

- Alice wants to send a message to Bob ...
- ... and doesn't want Eve to overhear
- *If* Alice and Bob share some secret information ...
- *then* they can use simple operations like $\otimes$ (XOR) ...
- ... or more complicated ones like Rijndael (aka AES)
- ... and Eve will not be able to eavesdrop

Symmetric Cryptography

Introducing Alice, Bob and Eve...

- Alice wants to send a message to Bob ...
- ... and doesn't want Eve to overhear
- *If* Alice and Bob share some secret information ...
- *then* they can use simple operations like $\otimes$ (XOR) ...
- ... or more complicated ones like Rijndael (aka AES)
- ... and Eve will not be able to eavesdrop

Symmetric Cryptography

Introducing Alice, Bob and Eve...

- Alice wants to send a message to Bob ...
- ... and doesn't want Eve to overhear
- *If* Alice and Bob share some secret information ...
- *then* they can use simple operations like $\otimes$ (XOR) ...
- ... or more complicated ones like Rijndael (aka AES)
- ... and Eve will not be able to eavesdrop

How can Alice and Bob share a secret key?

Quantum Key Distribution

- QKD gives enough random numbers for $\otimes$
- information-theoretically secure (great!)
- needs very expensive equipment

How can Alice and Bob share a secret key?

Public Key Cryptography

Find a function f such that $x' = f(x)$ is (relatively) cheap to compute but $x = f^{-1}(x')$ is very expensive

$\rightarrow$ a *one-way* function.

How can Alice and Bob share a secret key?

Rivest-Shamir-Adleman

Example: RSA

$$
\begin{array}{cl}
n = pq & p, q \text{ prime} \\
\varphi(n) = (p-1)(q-1) & \text{secret modulus} \\
1 < e < \varphi(n) & e, \varphi(n) \text{ co-prime} \\
d = e^{-1} & \text{mod } \varphi(n) \\
c = m^e, \quad m = c^d & \text{mod } n
\end{array}
$$

The one-way function $x' = f(x) = x^e$ because its inverse $\log_e(x')$ mod n is the discrete logarithm problem. Reverse engineering p and q from the known n and using that to arrive at d means factoring a large integer. Both of which have no known solution for Mr. Turing's machine in polynomial-time.

Question:

How do you know that those Tails DVDs are true?

How do you know I haven't tampered with them?

We can use *signatures* to check:

$$s = \text{hash}(m)^d$$

$$s^e = \left(\text{hash}(m)^d\right)^e = \text{hash}(m)^{e^{-1}e} = \text{hash}(m)$$

Signatures are important

- Do you distribute software to others?

  → sign it so users can check where it came from.
- Do you use software from others?

  → check the signature to be confident you are getting what you think you are.

Signatures are important

. . . but are they enough?

- How do you know that a public (encryption or signing) key belongs to who it is claimed?
- Someone you trust can *sign the public key* and you can check that
  - You can trust a central authority like a government or a cartel (this is the way SSL for web sites works)
  - You can build a *web of trust* incrementally as a weighted sum of signatures (this is the way PGP works)

Signatures are important

. . . but are they enough?

- How do you know that a public (encryption or signing) key belongs to who it is claimed?
- Someone you trust can *sign the public key* and you can check that
  - You can trust a central authority like a government or a cartel (this is the way SSL for web sites works)
  - You can build a *web of trust* incrementally as a weighted sum of signatures (this is the way PGP works)

Signatures are important

. . . but are they enough?

- How do you know that a public (encryption or signing) key belongs to who it is claimed?
- Someone you trust can *sign the public key* and you can check that
  - You can trust a central authority like a government or a cartel (this is the way SSL for web sites works)
  - You can build a *web of trust* incrementally as a weighted sum of signatures (this is the way PGP works)

Signatures are important

. . . but are they enough?

- How do you know that a public (encryption or signing) key belongs to who it is claimed?
- Someone you trust can *sign the public key* and you can check that
  - You can trust a central authority like a government or a cartel (this is the way SSL for web sites works)
  - You can build a *web of trust* incrementally as a weighted sum of signatures (this is the way PGP works)

Enough Theory! Let's generate a key-pair

In the terminal[1]:

```
% gpg --gen-key
...

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
```

If using Tails, make sure to enable persistence!

[1] This works on DICE, but it is recommended to use a secure computer for PGP key generation and signing, not one that belongs to someone else.

Inspect your key

```
% gpg --fingerprint s12345678@inf.ed.ac.uk
pub 4096R/1234ABCD 2014-05-31
    Key fingerprint = 1234 5678 ...
uid                  A Student <s12345678@inf.ed.ac.uk>
sub 4096R/5678EF90 2014-05-31
```

- 1234ABCD is the *key id*. It is also the last four bytes of the fingerprint
- When signing someone else's key (vouching for their identity), always check the fingerprint

Generate a revokation certificate

```
% gpg --gen-revoke 1234ABCD
```

Store this somewhere safe, and ideally not connected to the Internet. If you forget your passphrase, or someone gets your private key, you can publish it to let everyone know not to use that key anymore.

Publish your public key

Send your key to the network of keyservers

```
% gpg --send 1234ABCD
```

Now others can find and your public key with

```
% gpg --recv 1234ABCD
```

Or alternatively

```
% gpg --export -a 1234ABCD
```

and paste the output to https://pgp.mit.edu/

Signing and verifying.

Sign your friend's key (after checking the fingerprint)

```
% gpg --sign-key 1324ACBD
```

Sign a file, producing `somefile.asc`

```
% gpg --clearsign somefile
```

Create a standalone signature without the content embedded

```
% gpg -ba somefile
```

Verify the signature

```
% gpg --verify somefile.asc
```

Encrypting and Decrypting

Create a version of the file that only your teacher can read

```
% gpg -ear wadler@inf.ed.ac.uk somefile
```

Signed as well,

```
% gpg -easr wadler@inf.ed.ac.uk somefile
```

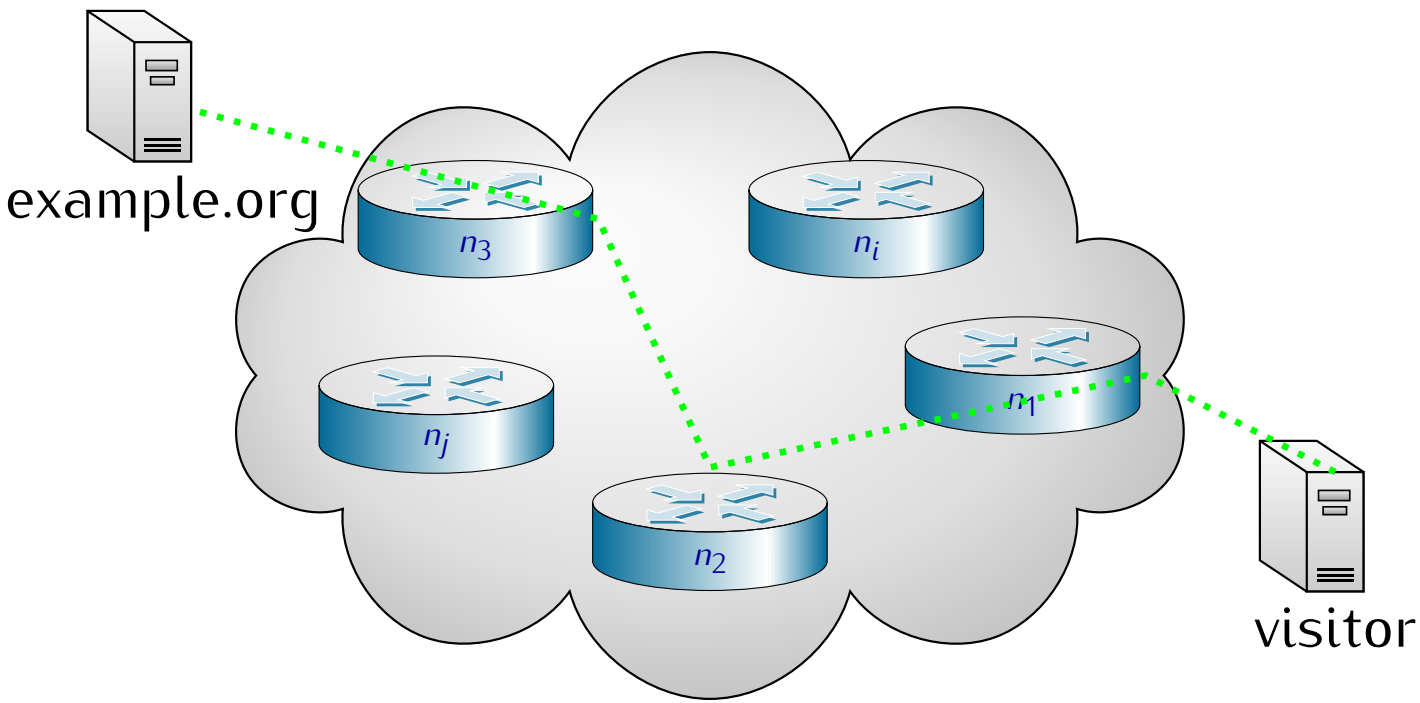
Decrypt a message that has been encrypted for you

```
% gpg -d somefile
```

Next: Tor

The Onion Router

https://www.torproject.org/



Tor: what does it do?

- Directs traffic through several (three) relays
- Works with any TCP traffic (not just for web sites)
- The server (example.org) cannot tell where the visitor is
- The relay n_1 can tell who the client is but not what they are looking at
- The relay n_3 can tell what server is being accessed but not by who
- The relay n_2 knows very little indeed

Tor: what does it do?

- Directs traffic through several (three) relays
- Works with any TCP traffic (not just for web sites)
- The server (example.org) cannot tell where the visitor is
- The relay n_1 can tell who the client is but not what they are looking at
- The relay n_3 can tell what server is being accessed but not by who
- The relay n_2 knows very little indeed

Tor: what does it do?

- Directs traffic through several (three) relays
- Works with any TCP traffic (not just for web sites)
- The server (example.org) cannot tell where the visitor is
- The relay n_1 can tell who the client is but not what they are looking at
- The relay n_3 can tell what server is being accessed but not by who
- The relay n_2 knows very little indeed

Tor: what does it do?

- Directs traffic through several (three) relays
- Works with any TCP traffic (not just for web sites)
- The server (example.org) cannot tell where the visitor is
- The relay n_1 can tell who the client is but not what they are looking at
- The relay n_3 can tell what server is being accessed but not by who
- The relay n_2 knows very little indeed

Tor: what does it do?

- Directs traffic through several (three) relays
- Works with any TCP traffic (not just for web sites)
- The server (example.org) cannot tell where the visitor is
- The relay n_1 can tell who the client is but not what they are looking at
- The relay n_3 can tell what server is being accessed but not by who
- The relay n_2 knows very little indeed

Tor: what does it do?

- Directs traffic through several (three) relays
- Works with any TCP traffic (not just for web sites)
- The server (example.org) cannot tell where the visitor is
- The relay n_1 can tell who the client is but not what they are looking at
- The relay n_3 can tell what server is being accessed but not by who
- The relay n_2 knows very little indeed

Tor: how does it work?

- The list of potential relays, and their public keys, is known
- Client selects an entry relay n_1, some other relay n_2, and an exit relay n_3
- Client creates a multiply-encrypted telescoping channel, and connects to n_1, then via n_1 to n_2, then via $n_1 n_2$ to n_3, and finally via $n_1 n_2 n_3$ to the server.
- This is all done with TLS
- Once the handshaking is done and the session keys, k_i are established, sending a message m means computing,

$$c = \mathrm{enc}(k_1, \mathrm{enc}(k_2, \mathrm{enc}(k_3, m)))$$

Tor hidden services

- Tor services *within* the Tor network itself
- Servers register with an *Introduction Point* (IP)
- Servers publish a name[2] to a distributed database[3]
- Client picks a *Rendez-vous Point* (RP)
- Client asks the IP to ask the server to meet her at the RP
- Client connects to the RP with a *fresh* Tor circuit
- All happens over Tor so in the end, 6 relays are involved, $(n_1, n_2, \text{RP}, n_4, n_5, n_6)$
- Hostnames look like j6vwvo5cbgkdwfoo.onion

[2] actually an encoding of a public key.
[3] actually a distributed hash table.

Limitations of Tor

Tor provides some strong anonymity guarantees at the transport layer, but it doesn't help for ...

- web browser fingerprinting
- eavesdropping on non-HTTPS connections at the last hop
- DNS queries leaking information
- deanonymisation by using strange protocols
- weak end-point security exploited by a malicious web site (e.g. flash, java plugins)
- if you type personal details into a web site
- if you download and run untrustworthy software on your computer

That all sounds very complicated . . .

The Tor Project as done a great job with the Tor Browser Bundle which is Tor with a version of Firefox configured to behave in the least risky way possible. Just install and run.

https://www.torproject.org/

The Tails Project has built an entire Linux distribution around GnuPG, the Tor Browser Bundle and other tools designed to be easy to use and to also behave in the least risky way possible. Just write to a USB disk, boot and run.

https://tails.boum.org/

InsecureDrop

- Inspired by https://securedrop.org/ used by The Intercept, The Guardian and others
- Much easier to set up
- Much less secure
- Available from
  http://tardis.ed.ac.uk/~wwaites/2015/10/insecuredrop.py
- Accessible for this class at
  http://j6vwvo5cbgkdwfoo.onion
- It just stores and serves small files (like PGP encrypted messages)

Now, practice

- Generate a key, send it to the keyservers
- Import, sign, and re-export some friends' keys
- Cleartext-sign a message and upload it to InsecureDrop
- Send secret messages to your friends in public

http://j6vwvo5cbgkdwfoo.onion

http://tardis.ed.ac.uk/~wwaites/2015/10/cryptoparty.pdf