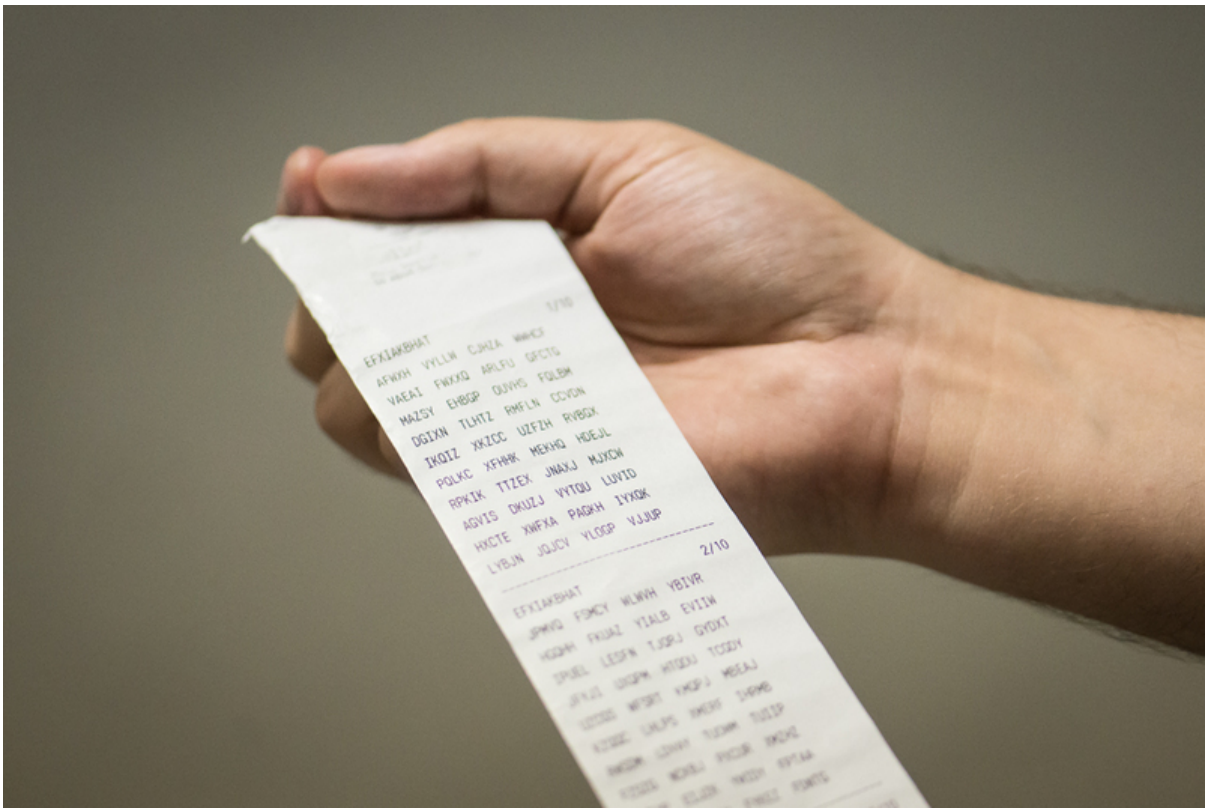
Raspberry Pi Thermal Printer One Time Pads

Created by Abigail Torres



https://learn.adafruit.com/raspberry-pi-thermal-printer-one-time-pads

Last updated on 2024-03-08 01:51:11 PM EST

Table of Contents

Overview 3

- TOO MANY SECRETS
-

A Brief History of One Time Pads 4

- Introduction
- How It Works

Encryption and Decryption 6

- Encryption
- Decryption

The Bad News 13

- Ways This Can Get Messed Up
- Why Computers Are Especially Awful At This

The Good News 15

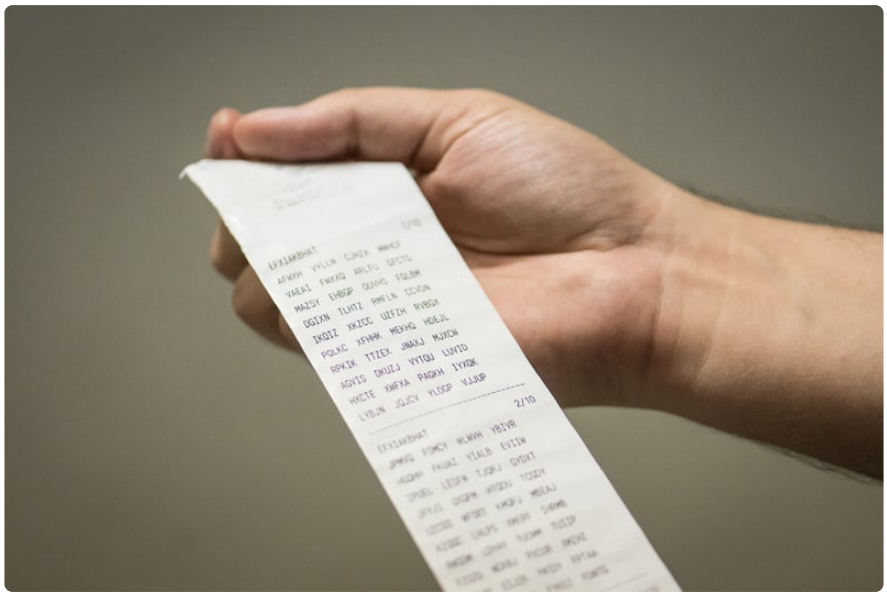
- Why the Raspberry Pi Is Actually Pretty OK For This

The Code 15

- Get It
- Read It
- Enable it
- Install rng-tools
- Test It
- Deploy It

Securing The System 19

Overview



TOO MANY SECRETS

In the climax to the hacker classic Sneakers (https://adafru.it/cAz), the good guys have to break in and steal a box which can decode anything. In real life, of course, such devices don't exist. But what if they did? Is there any type of encryption that will withstand a brute force attack by any amount of computing power?

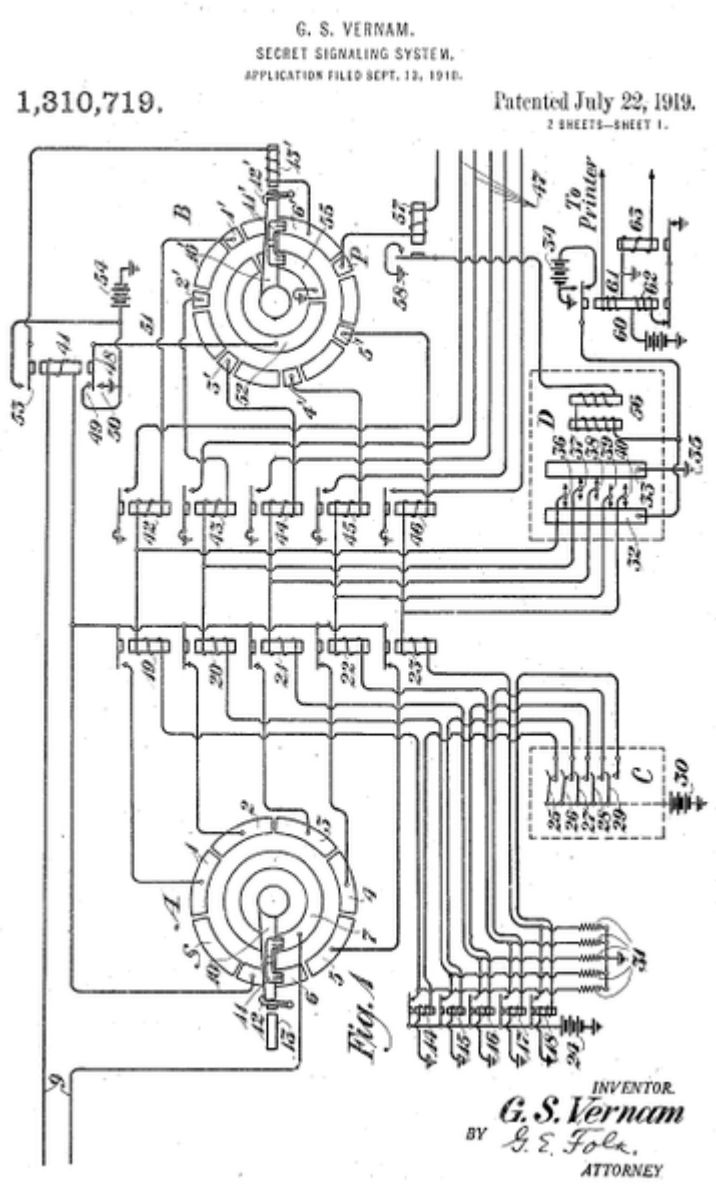
As it turns out, there is! Although, as we'll see later, unfortunately this doesn't readily translate into practical security. Still, thinking about security is a great habit to get into, and we can learn all kinds of great lessons by studying historical methods of encryption such as this one.

So, without further ado, let's get started! To follow this tutorial you'll need an Adafruit Internet of Things Thermal Printer. If you haven't done so already, head on over to http://learn.adafruit.com/pi-thermal-printer/overview (https://adafru.it/cAp) and put one together! Then come back and learn a bit about the history of the one time pad, and how to print your own on your IoT printer!

A Brief History of One Time Pads

Introduction

The Vernam Cipher, or one-time pad, is a cipher that was first invented by Frank Miller in 1882, then later re-invented and patented by Gilbert Vernam in 1919.



How It Works

Each character of the message you wish to send (the "plaintext") is combined with one character from the pad (the "key") to produce one character of the coded message (the "ciphertext"). The cipher must also meet the following two conditions:

1. Each new digit of the key is completely unpredictable.
2. The key never repeats, in whole or in part.

Assuming those conditions are met, the Vernam Cipher is the only known cipher which enjoys Shannon Security... which means that no amount of computing power could brute force its way in and crack the code.

Why? Because the key is as long as the message, and all keys are equally likely. Therefore, if the ciphertext is "EQNVZ" then the key could be "XMCKL", which would mean that "HELLO" was the message. Or maybe the key was "TQURI", which would

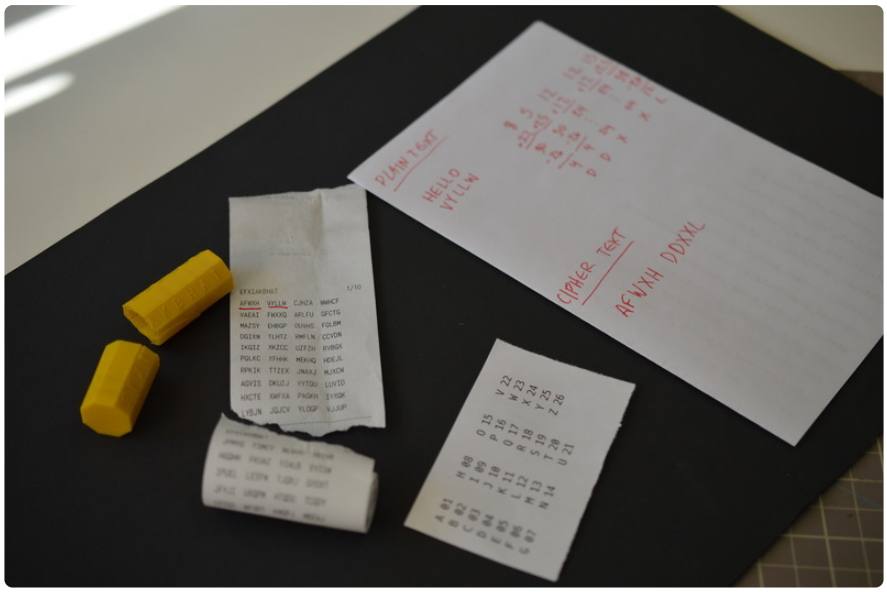
mean that "LATER" was the message. The message could be anything with the same number of characters, and there's no way to tell!

Perfect secrecy! Sound too good to be true? Well... it kind of is. There is a catch that makes this cipher very, very hard to perform in real life. Several, actually, some of which were exploited by this lady (https://adafru.it/cAA):

Her name was Gene Grabeel, and she was a Virginia schoolteacher before she went to work for the Army Signals Intelligence Service (the precursor to the NSA). Within weeks she had started the program that broke the codes which protected Soviet diplomatic messages.

But that's enough history for now, let's learn how to encrypt and decrypt using a one time pad...

Encryption and Decryption



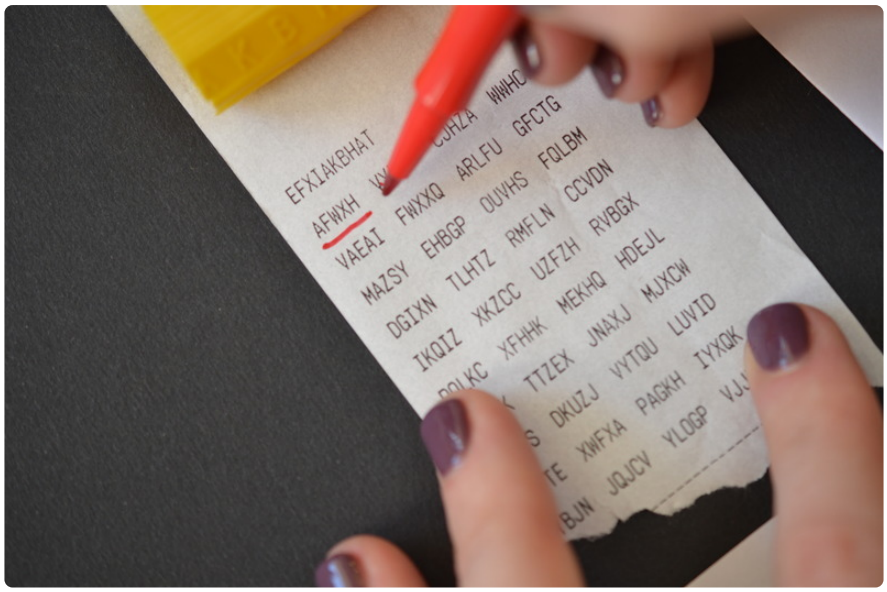
Encryption

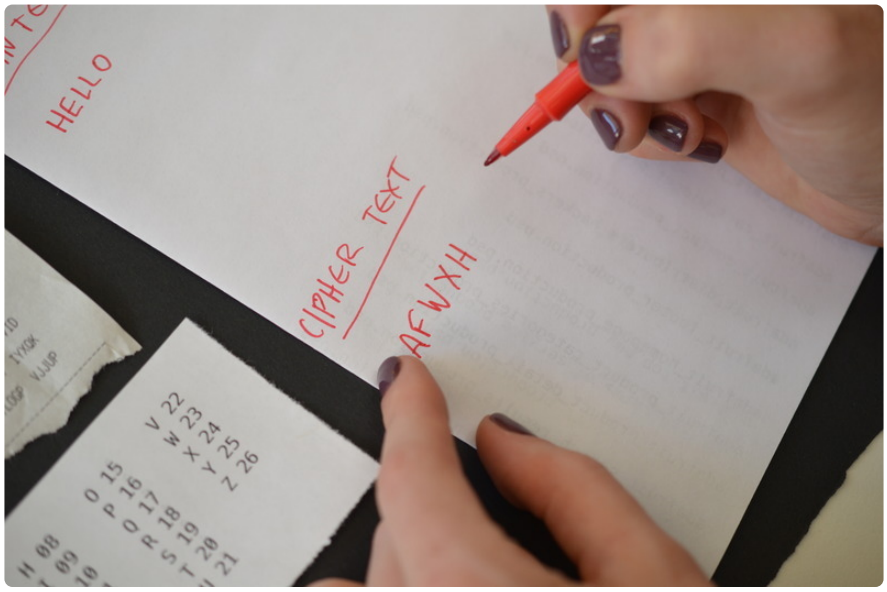
Let's say that the message you want to send is "HELLO". Here are the steps you'd go through:

Break open a tamper-resistant container, and remove the one time pad. Verify that the serial on the outside of the tube matches the serial of the pad inside of it.

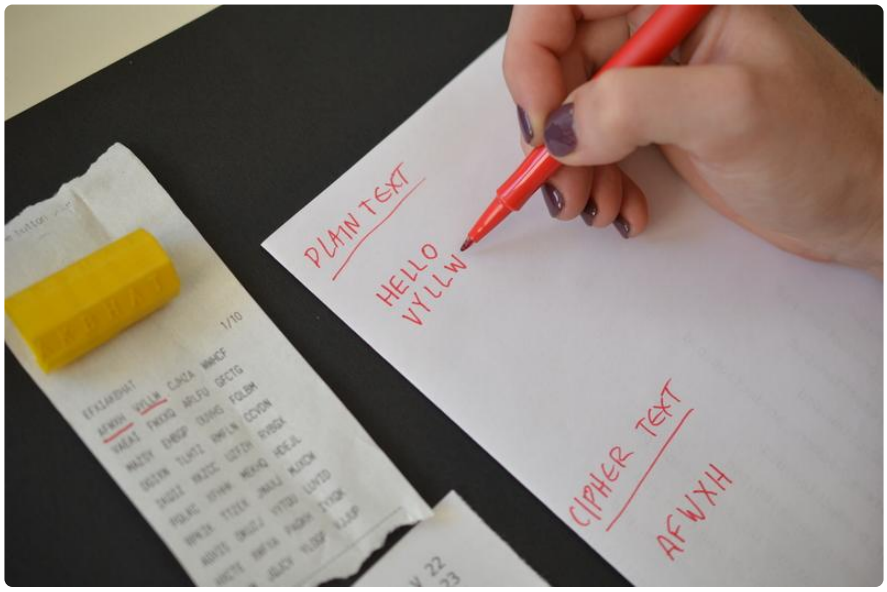
Tear off the first page of the pad.

The first 5 letter grouping on your page is "AFWXH". Write that down as the first 5 letter grouping of the ciphertext. This will allow your recipient to decrypt using the right page of his copy of the pad. Otherwise he'll just get gibberish!





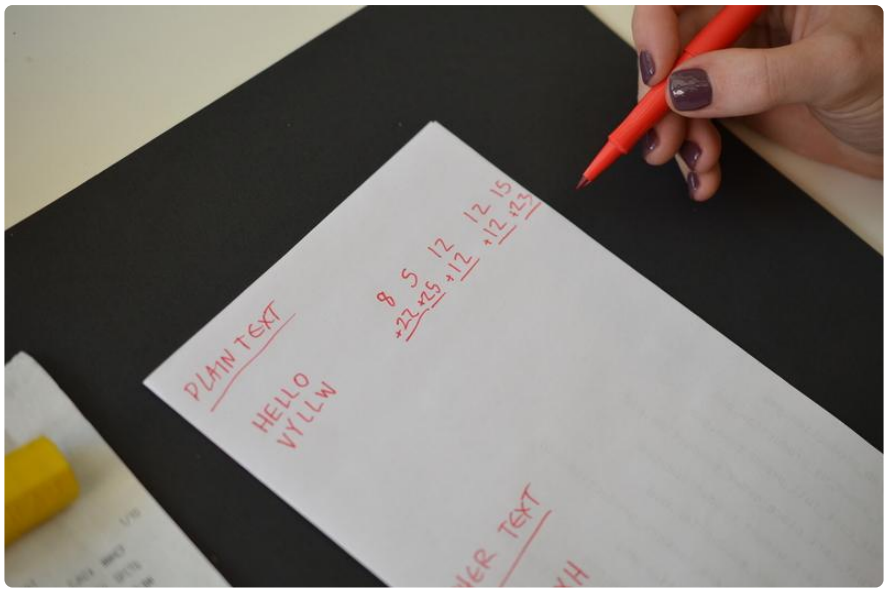
The next 5 letter grouping is "VYLLW". Since our message is only 5 letters long, this is all we'll need. Write our message ("HELLO") down if you haven't already, and "VYLLW" underneath it, so you have 5 pairs of letters.

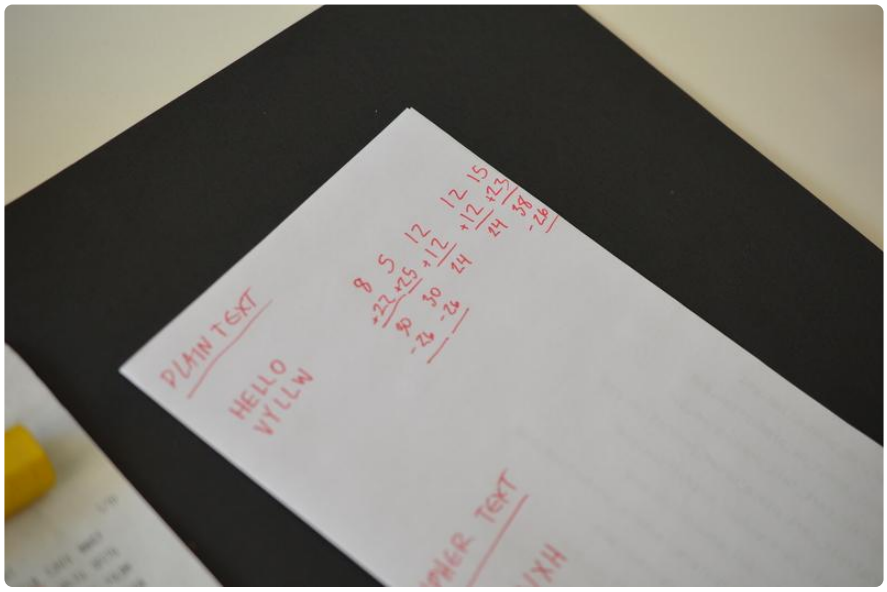


Next we're going to add the pairs of letters together, using the table below to turn the letters into numbers. Every letter in our Plaintext will get added to the letter below it (the key).

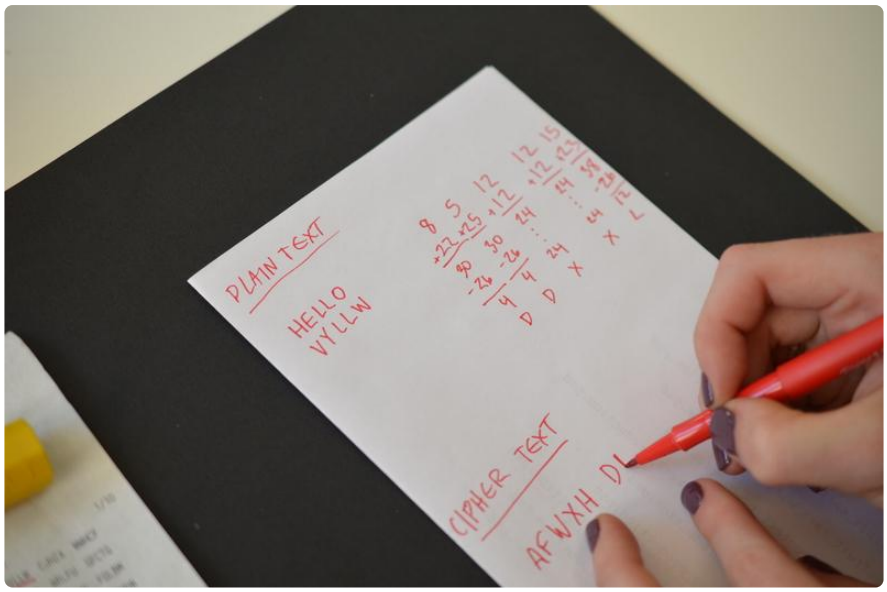
```
A=01 B=02 C=03 D=04 E=05 F=06 G=07
H=08 I=09 J=10 K=11 L=12 M=13 N=14
O=15 P=16 Q=17 R=18 S=19 T=20 U=21
V=22 W=23 X=24 Y=25 Z=26
```

Ok, so let's translate each letter of the plaintext and key into numbers, and add them together. If the result is greater than 26, subtract 26 from it.

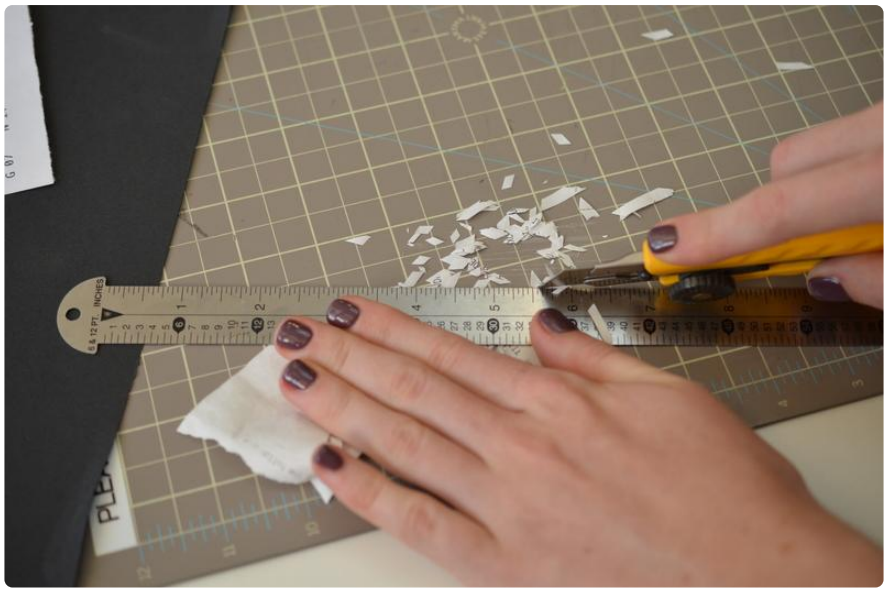




Then, once you've done that, turn all the numbers back into letters using the same chart. In our case, the numbers are 4,4,24,24,12. That translates into "DDXXL". That's the rest of the ciphertext, so write it down next to the "AFWXH" you wrote down earlier.



The final encrypted string you'll send to your friend is "AFWXH DDXXL". Send it via email, phone call, IM, whatever you want. Immediately after you do, you need to destroy the entire page that you used to encrypt your message. You can shred it...



... but why do that, when it's possible (however unlikely) that someone would root around in your trash and scotch tape the whole thing back together (https://adafru.it/cBj)? Better burn it instead.

As for the rest of the roll, you're going to either have to acquire a new tamper-resistant container to put it back into, or else just burn the whole thing.

Decryption

Decryption works just like encryption, except you subtract instead of adding. Let's say you received a message "AFWXH DDXXL" in your email, and you happen to know that the next pad you're supposed to use is the one with serial "EFXIAKBHAT". Here's the letter conversion table from above, for convenience.

```
A=01 B=02 C=03 D=04 E=05 F=06 G=07
H=08 I=09 J=10 K=11 L=12 M=13 N=14
```

```
O=15 P=16 Q=17 R=18 S=19 T=20 U=21
V=22 W=23 X=24 Y=25 Z=26
```

1. First, pop open the container marked EFXIAKBHAT and verify that the pad inside has the same serial.
2. Find the page in your pad that starts with "AFWXH".
3. D-V=H, D-Y=E, X-L=L, X-L=L, L-W=O.
4. The full recovered message is "HELLO".
5. Rip out and burn the entire page you were using, even if you didn't use the whole thing.

But, remember, I said there was a catch, right? Next we will talk about that...

The Bad News

Ways This Can Get Messed Up

It's time to talk about some of those limitations I mentioned before.

- If the key is predictable, even a little bit, you're sunk. That means you can't use real words, anything that you got out of your favorite programming language's rand() function, any algorithm that the NSA has put a back door into (https://adafru.it/cB9), the digits of pi, random numbers out of A Million Random Digits (https://adafru.it/cAq), or anything that has ever been written down publicly. Some Russian spies found that one out the hard way. (https://adafru.it/cAr)
- Oh, and "random" numbers you made up in your head or generate by banging on a keyboard aren't random, either. German and Russian spies found out that one the hard way.
- If you use the key even twice, ever, you're sunk. British, German, Russian, and American spies have all found that one out the hard way. **Make sure all copies of the key are destroyed immediately after they have served their purpose, preferably by burning.**
- If someone can make a copy of part or all of your pad, you're sunk. What's worse, if it was your pad that was compromised, you won't have a way of letting your buddy know that he should stop sending messages with his copy. **You therefore have to have perfect physical control over your copy of the pad at all times.** This is *much* easier said than done. A certain three letter agency has been known to drill out the lock on your door, photograph everything, and replace the lock with an identical one in under an hour.

- This algorithm doesn't tolerate error at all. If line noise or an attacker between you and your recipient changes an "E" to an "R", there will be no way for you to know why one letter of your message wouldn't decrypt right.
- Worse, this algorithm doesn't synchronize. So if line noise or an attacker changes an "E" to an "EE", then the entire rest of the message will be totally garbled and totally unrecoverable.

Why Computers Are Especially Awful At This

- The whole point of computers is to make copies of things. Just in the process of generating and printing your one time pads, you will have copies of the pads in memory, on the hard drive, in the system logs, and possibly in the cache of the printer. If your computer is on a network, any one of those copies could silently leak out.
- Even if your computer isn't on a network, it can still wirelessly transmit data (https://adafru.it/19oA). That delivery truck that's been sitting outside for the past hour can use an antenna to listen (https://adafru.it/cAt) to the electromagnetic noise that your keyboard or monitor is creating, and use it to figure out what you are typing. **You should therefore only operate your printer inside of a faraday cage (https://adafru.it/cAu) or in a secure location with plenty of space between you and the public.**
- Even if the attacker has to physically break in and steal the hard drive out of your machine, and you've already deleted everything, you might still be hosed. When you delete something, it's not really gone (https://adafru.it/cAv). Your computer just forgets where it is, and that space will be reclaimed next time you have a file that needs to go there. **Be sure to securely delete any files you make by overwriting the data with all zeros.**
- Even if the attacker can't recover any old data, he can still alter the program that generates the pads to make it output the digits of pi after a certain offset. You would have no way of knowing that your pads weren't random. **You therefore have to have complete physical control over the computer as well.**

Wow, what a downer, right? Well, don't worry. The Raspberry Pi has some features that make it actually pretty ok for doing this sort of thing...

The Good News

Why the Raspberry Pi Is Actually Pretty OK For This

- The Raspberry Pi is small. It can be locked in a safe when not in use, or even locked inside of a small faraday cage (https://adafru.it/cAu) while in use.
- The Raspberry Pi is cheap. Even if you're totally paranoid, and want to destroy both the SD card and the Pi once you're done generating pads, you wouldn't be out an unreasonable amount of money.
- The Raspberry Pi has a hardware random number generator on it. This. Is. BIG. This means that, with no additional hardware, you can put the Pi to work generating truly random numbers, and doing it fast. Most computers only have software pseudo-random number generators, and therefore cannot be used to generate one time pads.
- Thermal paper printers do not secretly encode the timestamp and serial number (https://adafru.it/cAw) of the printer you used onto every print like laser printers do.

So, I hope I've sold you on the idea of foolishly forging ahead and printing one time pads on the pi... let's get to it!

The Code

Get It

Once you've got your IoT printer all set up, we're going to need to do a bad thing and hook it up to a network so we can download some code. Head on over to GitHub, and clone the repo onto your local box. To do that, ssh into the pi and run the following command from somewhere in your home directory:

```
git clone https://github.com/iworkinpixels/otp-gen.git
```

Read It

Here's a quick tour through the code:

- otp.sh is a bash script that will generate random numbers, and dump the final one time pad to a text file. This text file will later be sent to the printer every time you press the button on the IoT printer. It runs on every startup, so you should have a new pad every time you start up the printer. Make sure that this is the

case, because you can't use the same pad twice.
- otp.py is the python script that gets run when you start up your IoT printer. It listens to the button and prints a copy of the otp text file every time you press the button, and shuts down the pi if you hold down the button.
- otp.txt is an example of the text file that has the one time pad in it. Open it up if you want to take a look without wasting paper. The real otp.txt is written to a ramdisk so that it never goes to the SD card, and is therefore destroyed whenever you shut off the box.

And that's it! Everything else is just library files or README files.

Enable it

First, we need to enable the hardware random number generator. Make sure you are using the appropriate commands for the version of the Raspberry Pi you are using.

Raspberry Pi v1 & v2

Enable the random number generator module:

```
sudo modprobe bcm2708-rng
```

This will add the driver for the rng once, but if we want it to show up again after we reboot, we need to add it to /etc/modules-load.d using the following command:

```
echo bcm2708-rng | sudo tee /etc/modules-load.d/rng-tools.conf
```

Raspberry Pi v3

Enable the random number generator module:

```
sudo modprobe bcm2835_rng
```

This will add the driver for the rng once, but if we want it to show up again after we reboot, we need to add it to /etc/modules-load.d using the following command:

```
echo bcm2835_rng | sudo tee /etc/modules-load.d/rng-tools.conf
```

Install rng-tools

While we're at it, let's set up some tools to deal with random numbers:

```
sudo apt-get install rng-tools
```

These tools will let us test the numbers we get and make sure they're really random. We won't go into how to do that here, but check out this link (https://adafru.it/cAx) to learn more.

Test It

Let's test it out and see if everything works! go to wherever you downloaded the otp-gen repo to, and run the following command:

```
./otp.sh
```

This should take a long time to run, but when it's finished, you should have a brand-new one time pad waiting for you in the same directory! Assuming there were no errors, your next step is to open up the file and see what's in there!

```
sudo nano otp.txt
```

You should be looking at something that looks a little bit like this:

```
EFXIAKBHAT                    1/10
AFWXH  VYLLW  CJHZA  WWHCF
VAEAI  FWXXQ  ARLFU  GFCTG
MAZSY  EHBGP  OUVHS  FQLBM
DGIXN  TLHTZ  RMFLN  CCVDN
IKQIZ  XKZCC  UZFZH  RVBGX
PQLKC  XFHHK  MEKHQ  HDEJL
RPKIK  TTZEX  JNAXJ  MJXCW
AGVIS  DKUZJ  VYTQU  LUVID
HXCTE  XWFXA  PAGKH  IYXQK
LYBJN  JQJCV  YLOGP  VJJUP
--------------------------------
EFXIAKBHAT                    2/10
JPMVQ  FSMCY  WLWVH  YBIVR
HGQHH  FKUAZ  YIALB  EVIIW
IPUEL  LESFN  TJQRJ  GYDXT
JFYJI  UXQPM  HTQDU  TCGDY
UZCGS  WFSRT  KMQPJ  MBEAJ
KZQQC  LHLPS  XMERF  IHRMB
RWSDM  LDVAY  TUOWM  TUIIP
FZSZG  NCKBJ  PXCUR  XMZHZ
OVEAP  EIJZK  YWIDY  FPTAA
HXQZC  XAUQN  FYKEZ  FDNTG
--------------------------------

[...]
```

```
EFXIAKBHAT                 10/10
PADNW  MQGYF  PAJMU  GCVHK
YYEFY  YIAVW  SPIRE  IMIPQ
ILWJU  BTKNH  CNFFF  QSXTG
AFGBC  WIJCU  DOPJP  HMLLY
HCZIR  RMSTQ  ZRUDR  NXMFS
QMOGV  RJGKC  JTJXY  MINUZ
QMEPB  EAYYC  GHDJF  EVAHM
ZKRYI  VXNUA  GERDJ  FPRZX
SIRKO  XPCAD  WHJHW  JNLXS
OCMAR  BGHRU  NBXQB  SFCMP
--------------------------------
```

(If your file looks *exactly* like that, then there's a problem, and otp.txt isn't being overwritten.)

Deploy It

Assuming everything worked, there's only one step left... we have to hook everything up so it runs when the machine starts!

```
sudo nano /etc/rc.local
```

Then replace the file with the following:

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi

cd /home/pi/otp-gen
./otp.sh
python ./otp.py &amp;

exit 0
```

Be sure to change the directory from /home/pi/otp-gen to whatever the path to your actual otp-gen directory is.

And that's it! Let's reboot the pi!

```
sudo shutdown -r now
```

If all goes well, you should get kicked off of your ssh session, and a few minutes later (after waiting for the new pad to be generated) the printer should print out a warning that it's on a network, a welcome message, and then pause.

Every time you press the button after that point, one copy of the pad will be printed. If everything works, then congratulations! You now have a stand-alone one time pad generator. You'll have to find a secure way of getting one copy of the pad to each of your friends, but if you can, then you'll have lots of fun sending super secret messages using state of the art (in WWII) technology!

Next up, security!

Securing The System

In the real world, making a one time pad system secure is a big problem... one that's big enough to stump some of the most famous names in security. But thinking about security is still a good exercise, and here's some ideas for how to make your system more secure:

- Take it off the network! Your pi no longer has any reason to have a wireless connection, and while it has one, that wireless connection is one giant security hole.
- When it's not in use, stick the whole thing inside a safe.
- If you add a battery for a power supply, it never has to leave the safe. If you line the inside of the safe with copper mesh, the safe doubles as a faraday cage. You open the door, turn on the machine, and press the button. Close the door, wait a few minutes, and open it up.
- Each copy of the pad should be stored in a tamper-resistant container. If you have access to a 3D printer, this one (https://adafru.it/cAy) will work well. Edit the OpenSCAD to specify the serial number of the pad you've just generated (it's on the top left corner of every page). Pause the print just before the top starts to print, and insert one tightly-rolled copy of the pad. Then resume the print and you will have a copy of the pad sealed inside. The only way to get it out and read it is to break open the container. Also, secretly reading the pad and then putting it into a replacement container takes time.
- Make two containers, and give one to your friend. When encoding messages, add the serial number, and then the first 5 random letters on the page to the beginning of your ciphertext. That way your recipient will know which container to break open in order to decode the message. When he's done, he should place the remaining pages in another tamper-proof container, or burn them all.
- If you're into 3D design, why not add a loop to the top of the container, and wear it as a necklace? You're a lot closer to having physical control of the pad at all times if you have it physically on you at all times.
- Have a secret, known only to you and your friend, that is used somewhere in every message. For instance, you might agree to use an onomatopoeia, such as "bang" or "buzz" in every message. If someone steals his pad and uses it to send you a message, you will know, because they won't include the secret. Even if they know there's a secret, they hopefully won't know which one it is, because it's never the same word or in the same place in any two messages.
- Note that if someone steals your buddy's copy of the pad, there is no protection for the messages you send him. The attacker can now read everything you send him.

And that's it! Have fun, and if you want to test it all out, feel free to email me at tony@adafruit.com with messages encoded using one of the pages of the example otp.txt included in the GitHub repo.