Name: Vishal Salvi
UID: 2019230069
Batch: C

LABORATORY

CEL62: Cryptography and System Security
Winter 2021

Experiment 6:	**Creating and Reading QR Codes** **Ref:** **https://www.linux-magazine.com/Online/Features/Generating-QR-Codes-in-Linux**

Note: Students are advised to read through this lab sheet before doing experiment. On-the-spot evaluation may be carried out during or at the end of the experiment. Your performance, teamwork/Personal effort, and learning attitude will count towards the marks.

Experiment 6: Creating and Reading QR codes

1 OBJECTIVE

Creating and Reading QR Codes for given exercise

2 INTRODUCTION

With the right tools, you can create your own QR code squares with information you want to share on a business card, in a letter, or on your website.

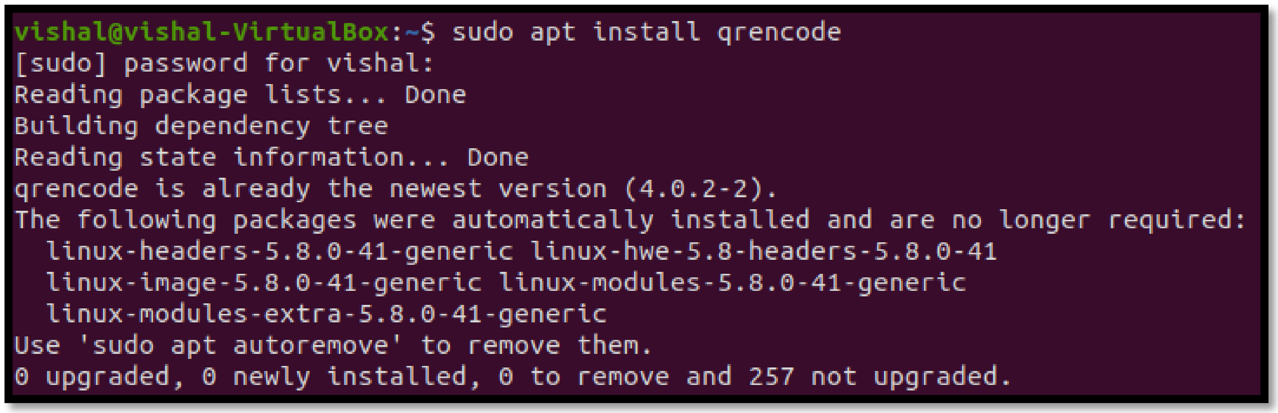
Read errors of unreliable, one-dimensional bar codes often caused interruptions in industrial production, prompting companies like Toyota and its subsidiary Denso Wave to develop as early as 1994 a new code for acquiring stock data. The new matrix bar code was designed to store more information than the traditional bar code and to stay legible, even if the label was dirty, wrinkled, or partially destroyed.

The quick response code, or QR code, comprises a matrix of square dots instead of the usual lines. Measuring up to 177 by 177 dots, the QR code encodes up to 4,296 characters, compared with a bar code that encodes just 13.

Thanks to numerous free reader apps for smartphones, QR codes have gained in popularity in recent years. Posters, catalogs, magazines, business cards, and even television screens display the small squares, offering additional information or URLs for microsites.

Data Grabber

If you want to encode your address into a QR code for a business card, you could try one of the many services on the Internet. However, restrictions typically apply. For example, some services allow only QR codes in a certain size or in limited quantities. Also, you never know for sure, what the services do with your data. For example, some sites explicitly allow sharing and selling the data to third parties in their terms of use, more or less pre-programming an increase in junk mail in the future. Fortunately, several QR code tools can help you avoid these problems in Linux.

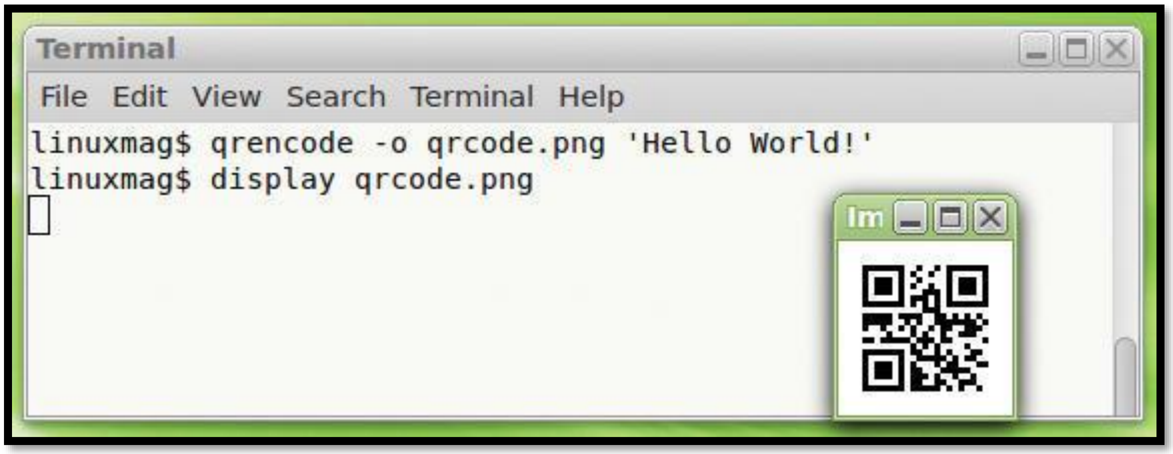


```
vishal@vishal-VirtualBox:~$ sudo apt install qrencode
[sudo] password for vishal:
Reading package lists... Done
Building dependency tree
Reading state information... Done
qrencode is already the newest version (4.0.2-2).
The following packages were automatically installed and are no longer required:
  linux-headers-5.8.0-41-generic linux-hwe-5.8-headers-5.8.0-41
  linux-image-5.8.0-41-generic linux-modules-5.8.0-41-generic
  linux-modules-extra-5.8.0-41-generic
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 257 not upgraded.
```

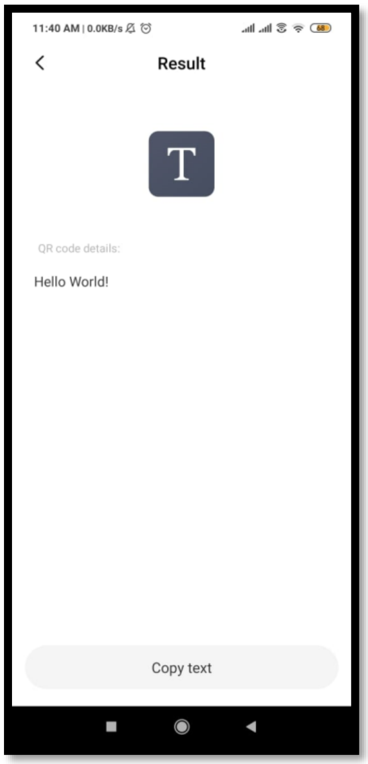
3 PROCEDURE/TASK

The quickest way to create a QR code is with the Qrencode command-line utility. Any major distribution can install Qrencode via the package manager. The following command then creates a QR code containing the text “Hello World!”:

$ qrencode -o qrcode.png 'Hello World!'



```
linuxmag$ qrencode -o qrcode.png 'Hello World!'
linuxmag$ display qrcode.png
```

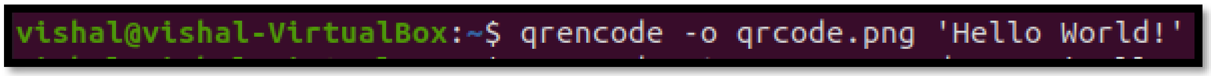


QR code details:

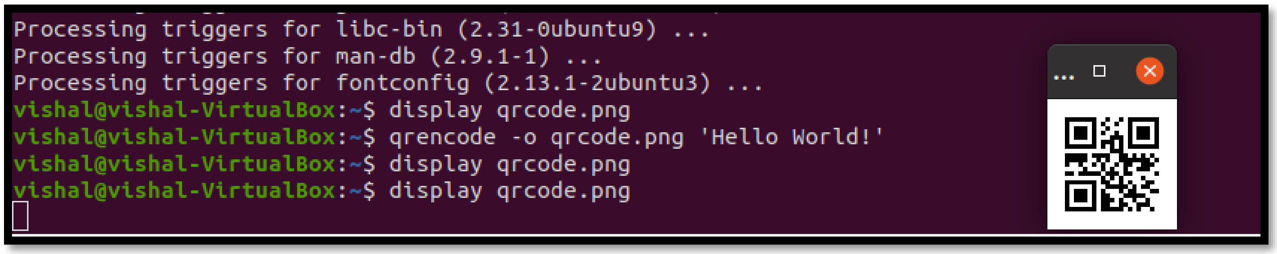
Hello World!

QR Code relevant Hello World will pop up: Qrencode encoding the string “Hello World!” as a QR code and storing it in the qrcode.png file.

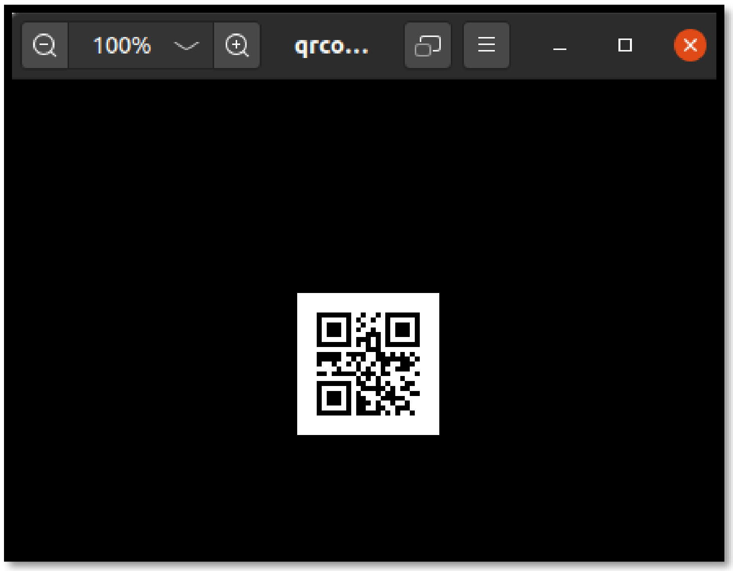
The generated QR code ends up in the *qrcode.png* file. If the file already exists, it’s overwritten without prompting. If you have at least version 3.3.0 (*qrencode -V*), Qrencode can generate an EPS graphic,



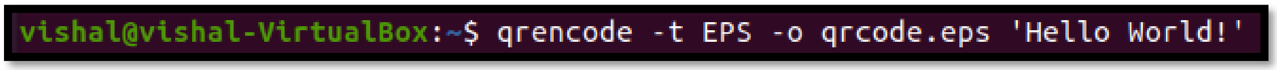
```
vishal@vishal-VirtualBox:~$ qrencode -o qrcode.png 'Hello World!'
```



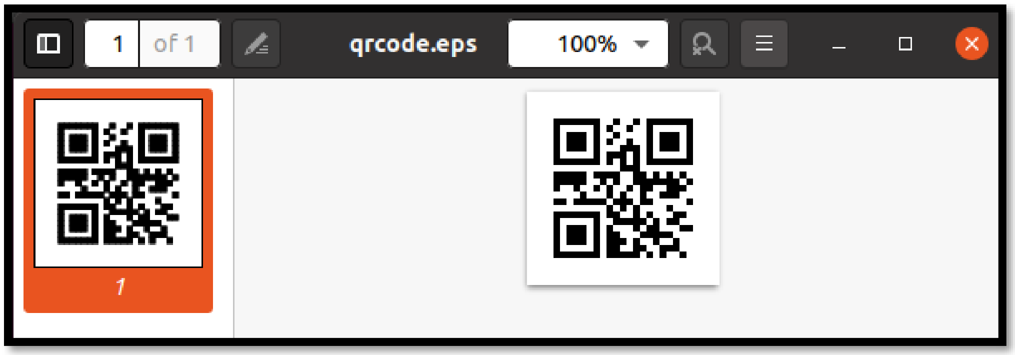
```
Processing triggers for libc-bin (2.31-0ubuntu9) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for fontconfig (2.13.1-2ubuntu3) ...
vishal@vishal-VirtualBox:~$ display qrcode.png
vishal@vishal-VirtualBox:~$ qrencode -o qrcode.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode.png
vishal@vishal-VirtualBox:~$ display qrcode.png
```

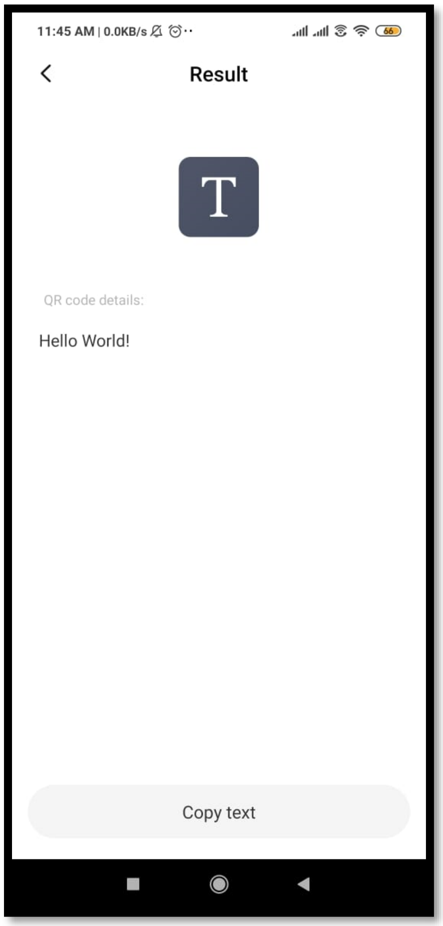


$ qrencode -t EPS -o qrcode.eps 'Hello World!'

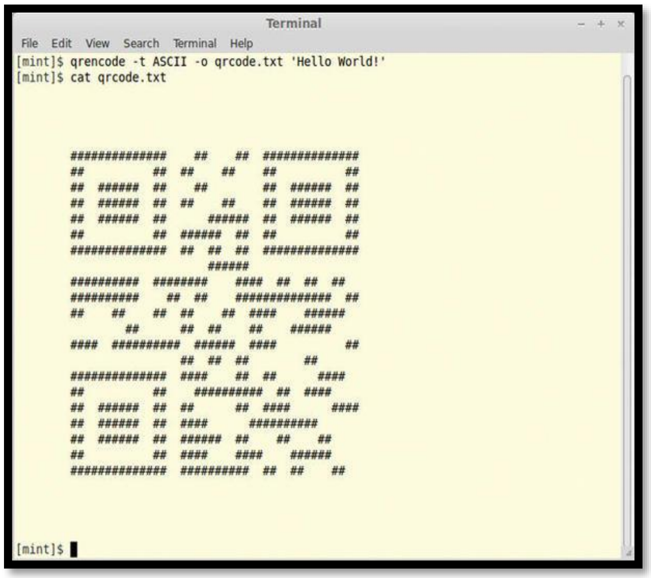


```
vishal@vishal-VirtualBox:~$ qrencode -t EPS -o qrcode.eps 'Hello World!'
```



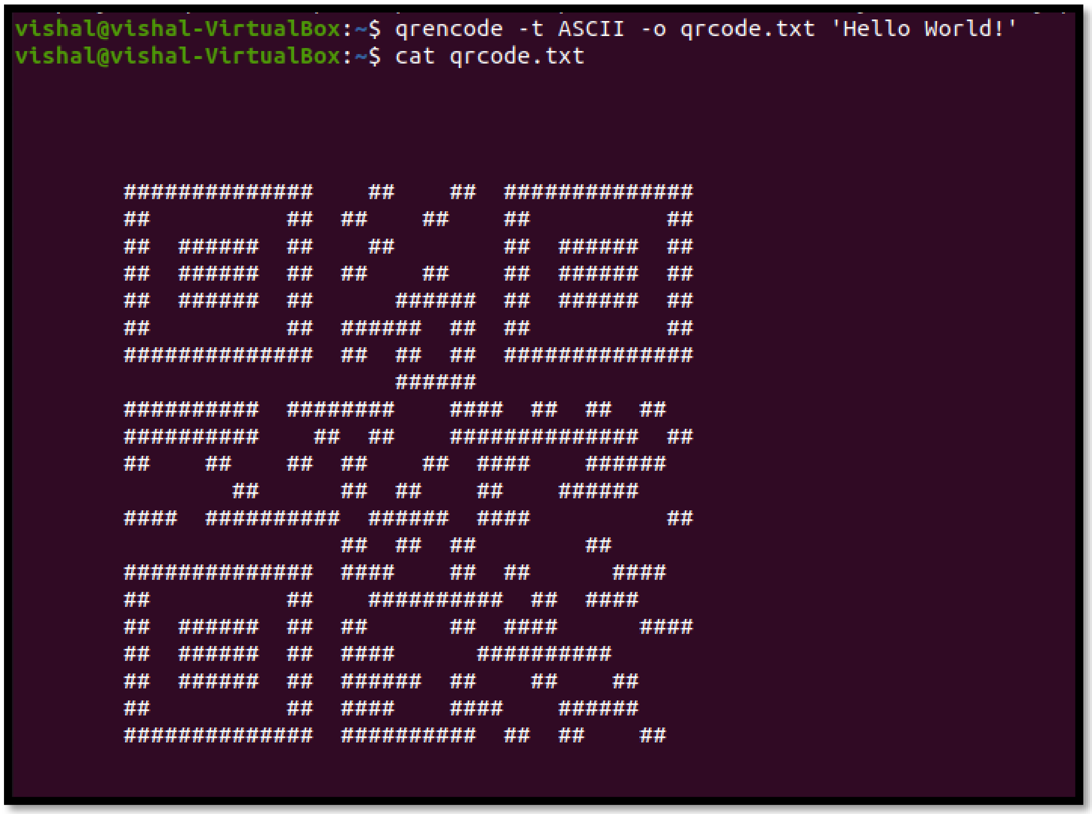


or ASCII output.

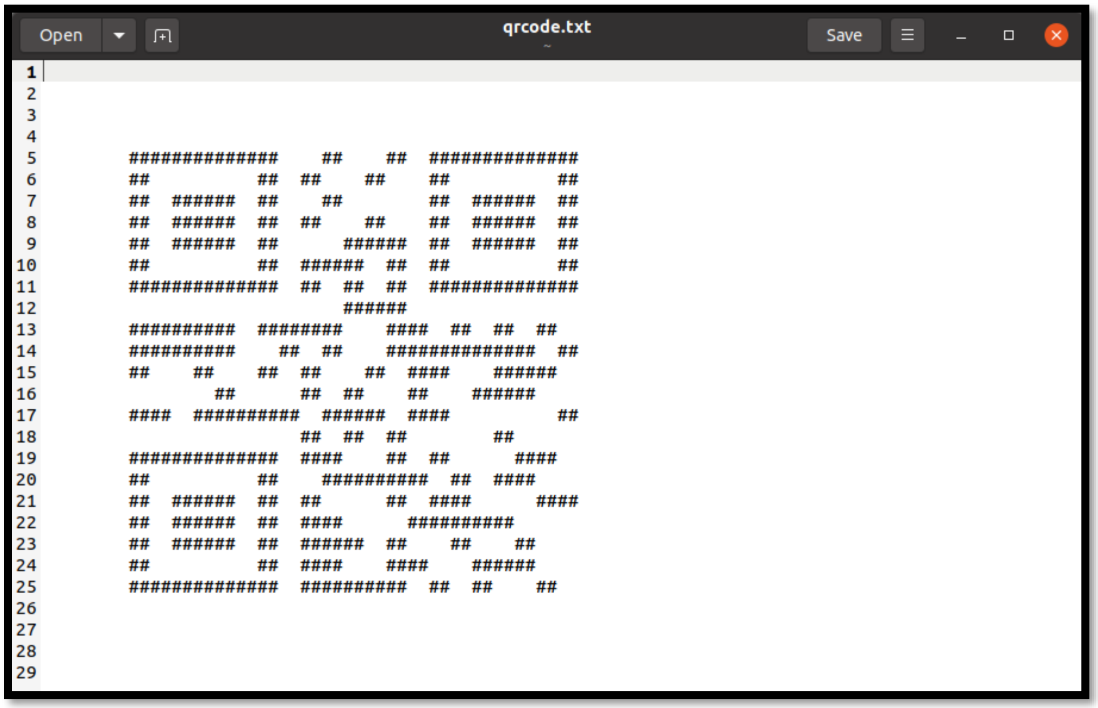


```
[mint]$ qrencode -t ASCII -o qrcode.txt 'Hello World!'
[mint]$ cat qrcode.txt




        ##############    ##    ##  ##############
        ##          ##  ##    ##    ##          ##
        ##  ######  ##    ##        ##  ######  ##
        ##  ######  ##  ##    ##    ##  ######  ##
        ##  ######  ##    ######  ##  ######  ##
        ##          ##  ######  ##  ##          ##
        ##############  ##  ##  ##  ##############
                            ######
        ##########  ########    ####  ##  ##  ##
        ##########    ##  ##    ##############  ##
        ##    ##    ##  ##    ##  ####    ######
                ##      ##  ##    ##    ######
        ####  ##########  ######  ####          ##
                        ##  ##  ##        ##
        ##############  ####    ##  ##      ####
        ##          ##    ##########  ##  ####
        ##  ######  ##  ##      ##  ####      ####
        ##  ######  ##  ####      ##########
        ##  ######  ##  ######  ##    ##    ##
        ##          ##  ####    ####    ######
        ##############  ##########  ##  ##    ##




[mint]$
```

Outputting QR codes in ASCII characters just for fun. Each # corresponds to a dot.



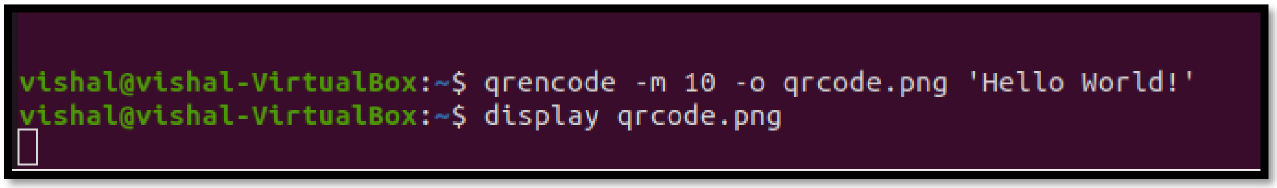
```
vishal@vishal-VirtualBox:~$ qrencode -t ASCII -o qrcode.txt 'Hello World!'
vishal@vishal-VirtualBox:~$ cat qrcode.txt




        ##############    ##    ##  ##############
        ##          ##  ##    ##    ##          ##
        ##  ######  ##    ##        ##  ######  ##
        ##  ######  ##  ##    ##    ##  ######  ##
        ##  ######  ##      ######  ##  ######  ##
        ##          ##  ######  ##  ##          ##
        ##############  ##  ##  ##  ##############
                            ######
        ##########  ########    ####  ##  ##  ##
        ##########    ##  ##    ##############  ##
        ##    ##    ##  ##    ##  ####    ######
                ##      ##  ##    ##    ######
        ####  ##########  ######  ####          ##
                        ##  ##  ##        ##
        ##############  ####    ##  ##      ####
        ##          ##    ##########  ##  ####
        ##  ######  ##  ##      ##  ####      ####
        ##  ######  ##  ####      ##########
        ##  ######  ##  ######  ##    ##    ##
        ##          ##  ####    ####    ######
        ##############  ##########  ##  ##    ##
```



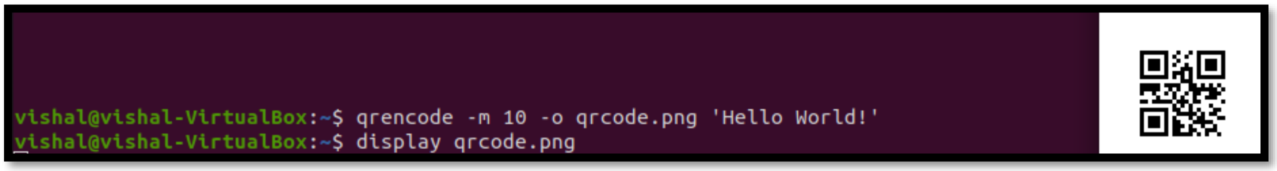
```




        ##############    ##    ##  ##############
        ##          ##  ##    ##    ##          ##
        ##  ######  ##    ##        ##  ######  ##
        ##  ######  ##  ##    ##    ##  ######  ##
        ##  ######  ##      ######  ##  ######  ##
        ##          ##  ######  ##  ##          ##
        ##############  ##  ##  ##  ##############
                            ######
        ##########  ########    ####  ##  ##  ##
        ##########    ##  ##    ##############  ##
        ##    ##    ##  ##    ##  ####    ######
                ##      ##  ##    ##    ######
        ####  ##########  ######  ####          ##
                        ##  ##  ##        ##
        ##############  ####    ##  ##      ####
        ##          ##    ##########  ##  ####
        ##  ######  ##  ##      ##  ####      ####
        ##  ######  ##  ####      ##########
        ##  ######  ##  ######  ##    ##    ##
        ##          ##  ####    ####    ######
        ##############  ##########  ##  ##    ##




```

In the QR code image, the software creates a white border the width of one dot. This facilitates the process of deciphering the code for programs or the smartphone later on. If desirable, you can increase or decrease the edge with the *-m* parameter; in the following example the border width would be 10 code pixels:

$ qrencode -m 10 -o qrcode.png 'Hello World!'



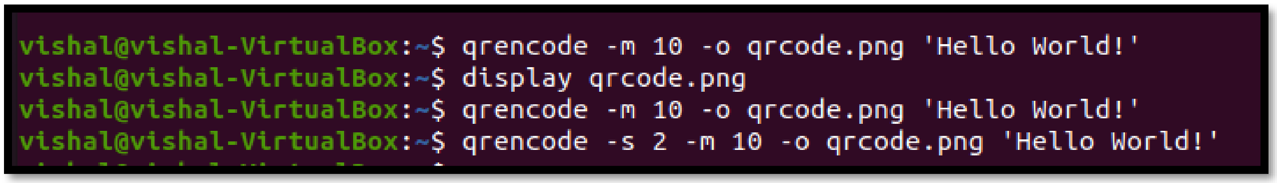
```
vishal@vishal-VirtualBox:~$ qrencode -m 10 -o qrcode.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode.png
```



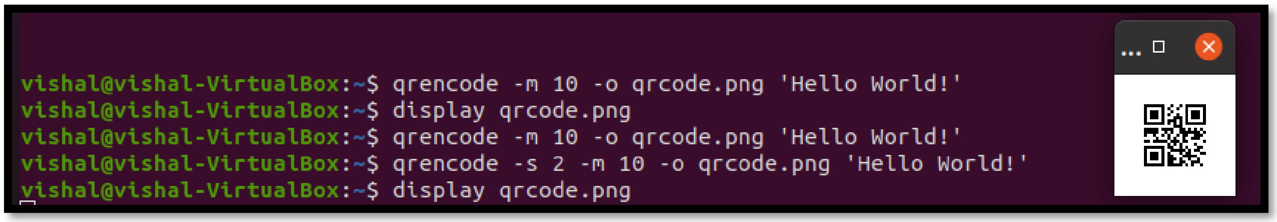
```
vishal@vishal-VirtualBox:~$ qrencode -m 10 -o qrcode.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode.png
```

If you are saving the QR code in PNG format, the *-s* parameter specifies the height of a black QR code pixel. By default, Qrencode draws every black dot three by three pixels. The program creates a slightly smaller QR code with quite a wide margin with the following command:

$ qrencode -s 2 -m 10 -o qrcode.png 'Hello World!'

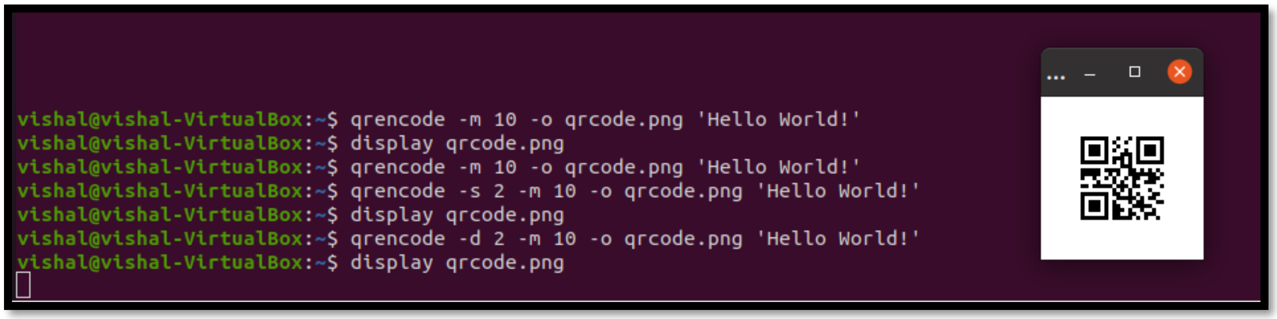


```
vishal@vishal-VirtualBox:~$ qrencode -m 10 -o qrcode.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode.png
vishal@vishal-VirtualBox:~$ qrencode -m 10 -o qrcode.png 'Hello World!'
vishal@vishal-VirtualBox:~$ qrencode -s 2 -m 10 -o qrcode.png 'Hello World!'
```



```
vishal@vishal-VirtualBox:~$ qrencode -m 10 -o qrcode.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode.png
vishal@vishal-VirtualBox:~$ qrencode -m 10 -o qrcode.png 'Hello World!'
vishal@vishal-VirtualBox:~$ qrencode -s 2 -m 10 -o qrcode.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode.png
```

A white border appears around the QR code that is exactly 20 screen pixels wide (10 QR code dots in width, with each dot two pixels). If you are creating EPS images, the only parameter available is *-m*; *-s* moves the entire QR code out of the image in this case. To control the resolution of PNGs further, use the *-d* parameter to define the dpi.

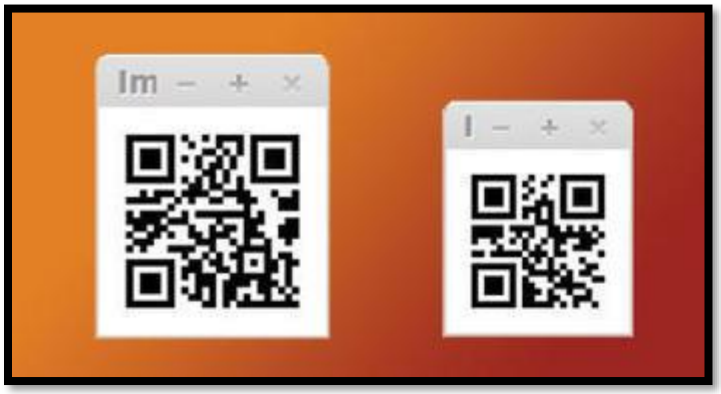


```
vishal@vishal-VirtualBox:~$ qrencode -m 10 -o qrcode.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode.png
vishal@vishal-VirtualBox:~$ qrencode -m 10 -o qrcode.png 'Hello World!'
vishal@vishal-VirtualBox:~$ qrencode -s 2 -m 10 -o qrcode.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode.png
vishal@vishal-VirtualBox:~$ qrencode -d 2 -m 10 -o qrcode.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode.png
```

Tolerance

Besides the specified data, the QR code contains additional error correction information. If a portion of the image is damaged, it allows you to reconstruct the missing or illegible data. The more additional information the QR code contains, the more heavily damaged it can be without becoming useless.

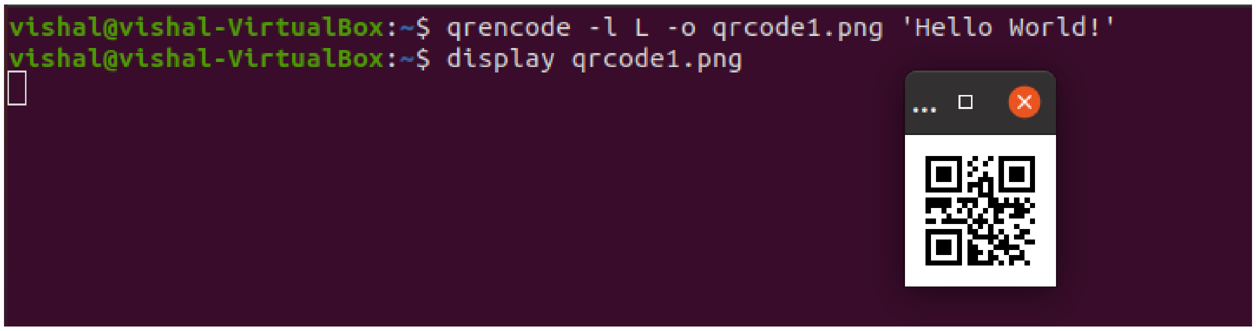
Increasing the error tolerance increases the size of the image because you need more black dots



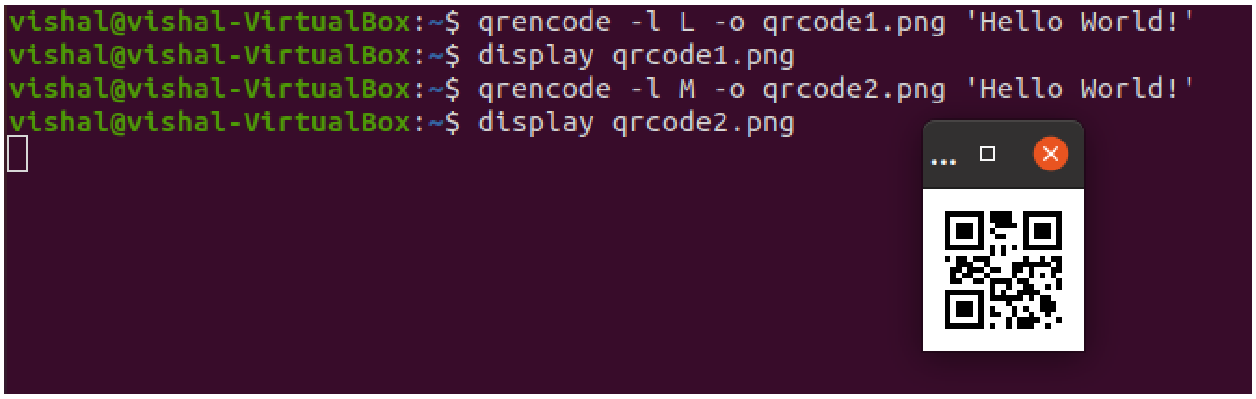
Both QR codes contain the text Hello World!; the one on the left uses the highest error correction level, H, and therefore is more robust – but also larger.

The QR code standard thus uses four levels of error correction: *H* level allows you to read all data if 30 percent of the QR code is destroyed, *Q* level if 25 percent is unreadable, and *M* level if just 15 percent is unintelligible. At the lowest level, *L*, only 7 percent of the data can be faulty. In Qrencode, the -*l* parameter selects the error correction level. The possible values are pretty much what you would expect: *L*, *M*, *Q*, and *H*.

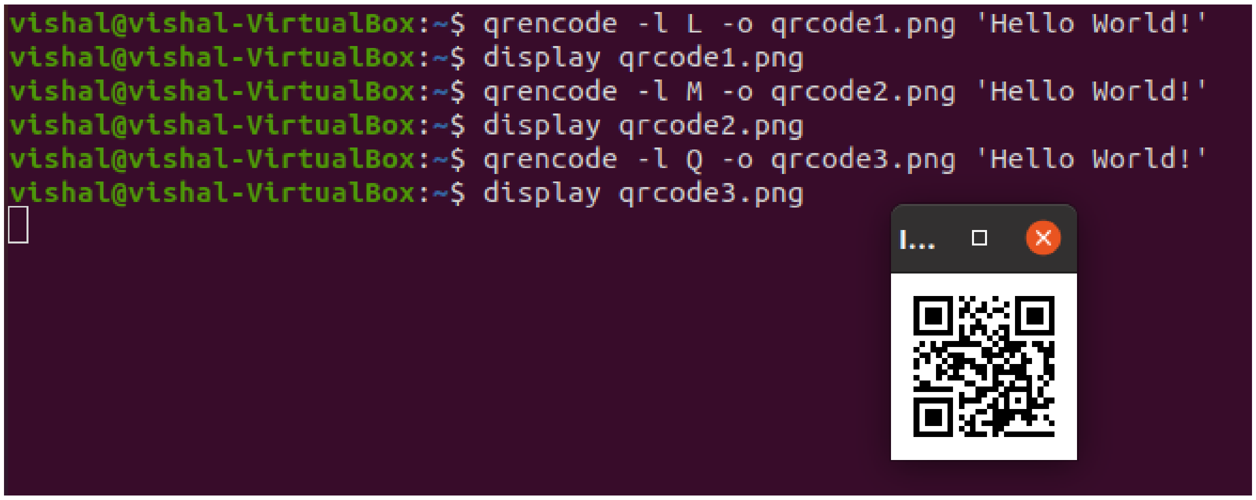
Of the tested programs, Portable QR-Code Generator offers the greatest functionality. Data entry is also convenient in a variety of tabs. Qrencode creates QR codes more quickly, and it is easy to scripted so you generate QR codes in batches. KBarcode4-light is deprecated, but it is the only program that can generate PDF files; the EPS images created by Qrencode offer similar output.



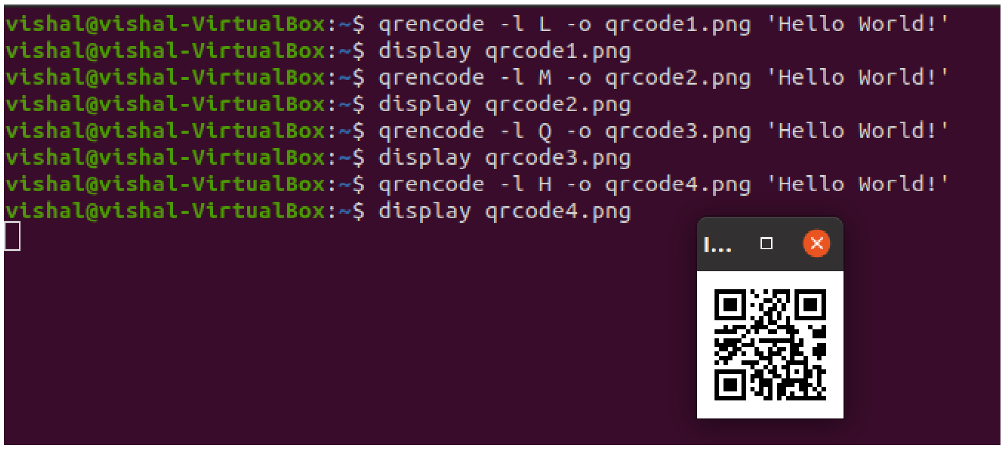
```
vishal@vishal-VirtualBox:~$ qrencode -l L -o qrcode1.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode1.png
```



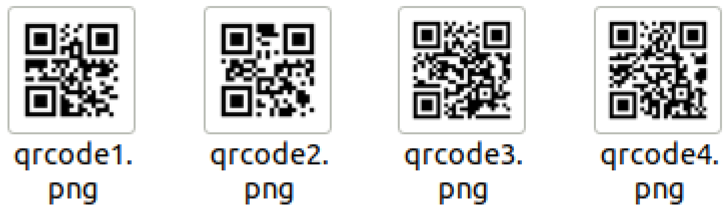
```
vishal@vishal-VirtualBox:~$ qrencode -l L -o qrcode1.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode1.png
vishal@vishal-VirtualBox:~$ qrencode -l M -o qrcode2.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode2.png
```

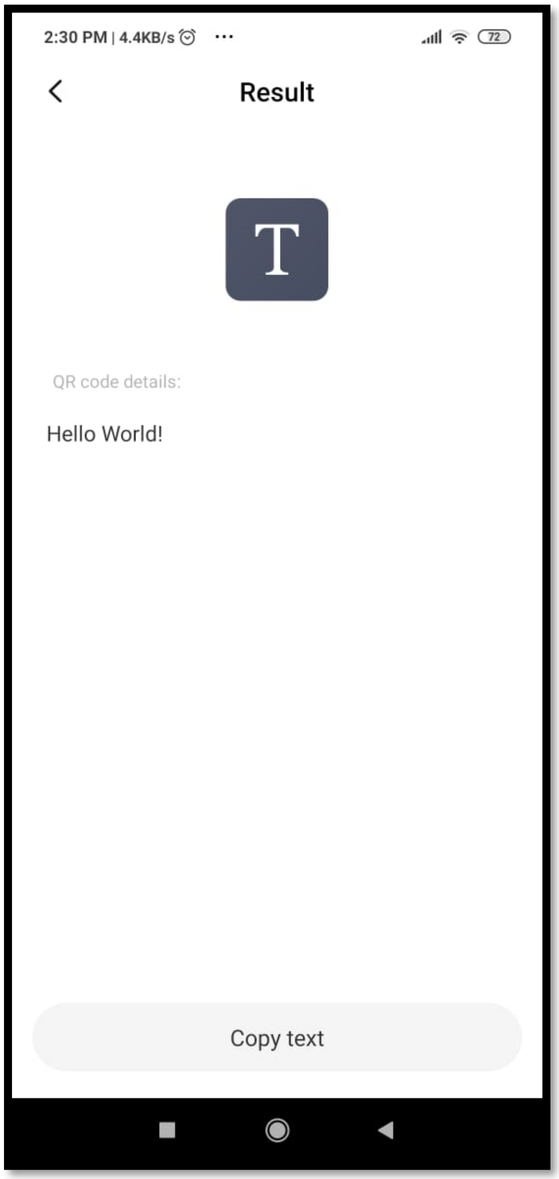


```
vishal@vishal-VirtualBox:~$ qrencode -l L -o qrcode1.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode1.png
vishal@vishal-VirtualBox:~$ qrencode -l M -o qrcode2.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode2.png
vishal@vishal-VirtualBox:~$ qrencode -l Q -o qrcode3.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode3.png
```



```
vishal@vishal-VirtualBox:~$ qrencode -l L -o qrcode1.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode1.png
vishal@vishal-VirtualBox:~$ qrencode -l M -o qrcode2.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode2.png
vishal@vishal-VirtualBox:~$ qrencode -l Q -o qrcode3.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode3.png
vishal@vishal-VirtualBox:~$ qrencode -l H -o qrcode4.png 'Hello World!'
vishal@vishal-VirtualBox:~$ display qrcode4.png
```





QR code details:

Hello World!

EXERCISE

TPO of SPIT has asked you to submit simple QR code of your CV so that company can make off line process to create short list of potential candidates. Hence as a part of exercise create CV showing your Profile, Ability and Capability and generate QR code of CV. This QR code can only be shared between you and company. You will provide your digital signature as an authentication to company. Once authentication is successful company is able to scan this QR code of your CV.

Show Implementation steps that you will generate QR code and sign it with your digital signature. Share it with your colleague from same lab with who is capable to verify your digital signature. Successful verification makes scanning of QR code feasible. Show you creativity in terms of robustness of generating digital signature, generating of QR code and Scanning of QR code if relevant secret key is entered.

Key.py

```
import rsa
from cryptography.fernet import Fernet
# Creating asymmetric keys
def create_keys(public_key_file_name, private_key_file_name):
# Creating the public and private keys
   (pubkey, privkey) = rsa.newkeys(2048)
# Write the public key to a file
   with open(public_key_file_name, 'wb') as key_file:
      key_file.write(pubkey.save_pkcs1('PEM'))
# Write the private key to a file
   with open(private_key_file_name, 'wb') as key_file:
      key_file.write(privkey.save_pkcs1('PEM'))

create_keys("my_public_key.pem", "my_priv_key.pem")
create_keys("sign_public_key.pem", "sign_private_key.pem")
# Creating symmetric key
key = Fernet.generate_key()
# Writing the symmetric key to a file
k = open('symmetric_key.pem', 'wb')
k.write(key)
k.close()
```

cv.py

```
import qrcode as qr
import rsa
from cryptography.fernet import Fernet


# Function used to read files
def file_open(file_name):
    file = open(file_name, 'rb')
    file_data = file.read()
    file.close()
    return file_data
```

```
resume_text = file_open("cv.txt").decode("UTF-8")

# Creating the qrcode
qr_object = qr.QRCode(None, error_correction=qr.constants.ERROR_CORRECT_L,
                      box_size=3, border=4)
qr_object.add_data(resume_text)
qr_object.make(fit=True)
qrcode = qr_object.make_image()
qrcode.save('qr_cv.png')

privkey_applicant = rsa.PrivateKey.load_pkcs1(file_open('my_priv_key.pem'))
pubkey_company = rsa.PublicKey.load_pkcs1(file_open('sign_public_key.pem'))

symkey = file_open('symmetric_key.pem')
# Reading and hashing the PNG Image
qrcode_png = file_open("qr_cv.png")
# Creating the hash and signing it with the private

signature = rsa.sign(qrcode_png, privkey_applicant, 'SHA-512')
s = open('signature.bin', 'wb')
s.write(signature)
# Encrypting the QR Code file
cipher = Fernet(symkey)
qrcode_encrypted = cipher.encrypt(qrcode_png)
e = open('encrypted.bin', 'wb')
e.write(qrcode_encrypted)
# Encrypting the symmetric key using the company's public key
symkey_encrypted = rsa.encrypt(symkey, pubkey_company)
e = open('encrypted_symmetric_key.bin', 'wb')
e.write(symkey_encrypted)
```

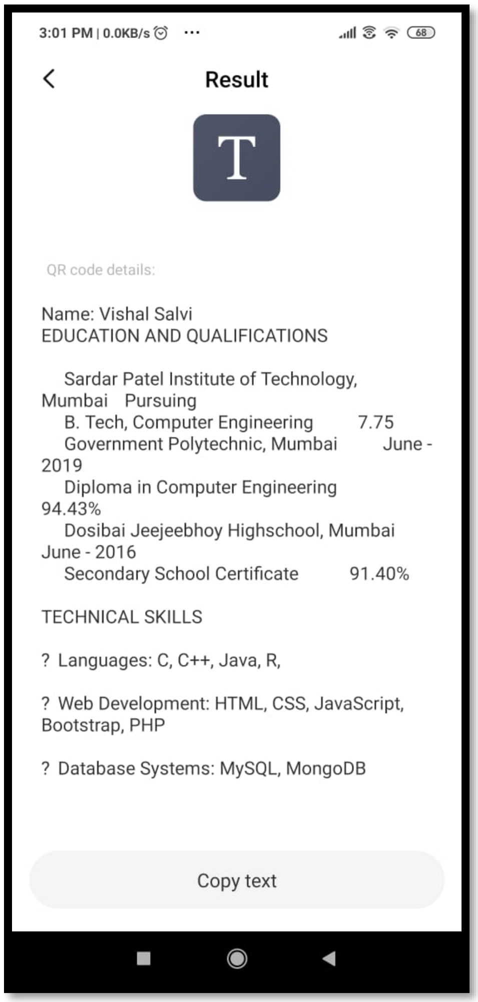
sign.py

```
import qrcode as qr
import rsa
from cryptography.fernet import Fernet


def file_open(file_name):
   file = open(file_name, 'rb')
   file_data = file.read()
   file.close()
   return file_data

encrypted_qrcode = file_open('encrypted.bin')
encrypted_symmetric_key = file_open('encrypted_symmetric_key.bin')
signature = file_open('signature.bin')

privkey_company =
rsa.PrivateKey.load_pkcs1(file_open('sign_private_key.pem'))
pubkey_applicant = rsa.PublicKey.load_pkcs1(file_open('my_public_key.pem'))

# Decrypt and obtain the symmetric key
symkey = rsa.decrypt(encrypted_symmetric_key, privkey_company)

# Decrypt and obtain the qrcode
cipher = Fernet(symkey)
qrcode_png = cipher.decrypt(encrypted_qrcode)
```

```python
# Verify the signature
try:
   rsa.verify(qrcode_png, signature, pubkey_applicant)
   print("Signature verified.")
# Decrypt the QR Code
   q = open("decrypted_qrcode.png", "wb")
   q.write(qrcode_png)
except:
   print("Signature could not verified!!!")
```

```
C:\Users\Vishal\AppData\Local\Programs\Python\Python38\python.exe C:/Python/sign.py
Signature verified.

Process finished with exit code 0
```

CV QR code:



Name: Vishal Salvi
EDUCATION AND QUALIFICATIONS

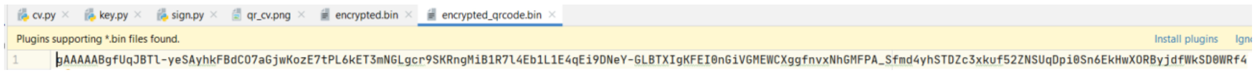
Sardar Patel Institute of Technology, Mumbai Pursuing
B. Tech, Computer Engineering 7.75
Government Polytechnic, Mumbai June - 2019
Diploma in Computer Engineering 94.43%
Dosibai Jeejeebhoy Highschool, Mumbai June - 2016
Secondary School Certificate 91.40%

TECHNICAL SKILLS

? Languages: C, C++, Java, R,

? Web Development: HTML, CSS, JavaScript, Bootstrap, PHP

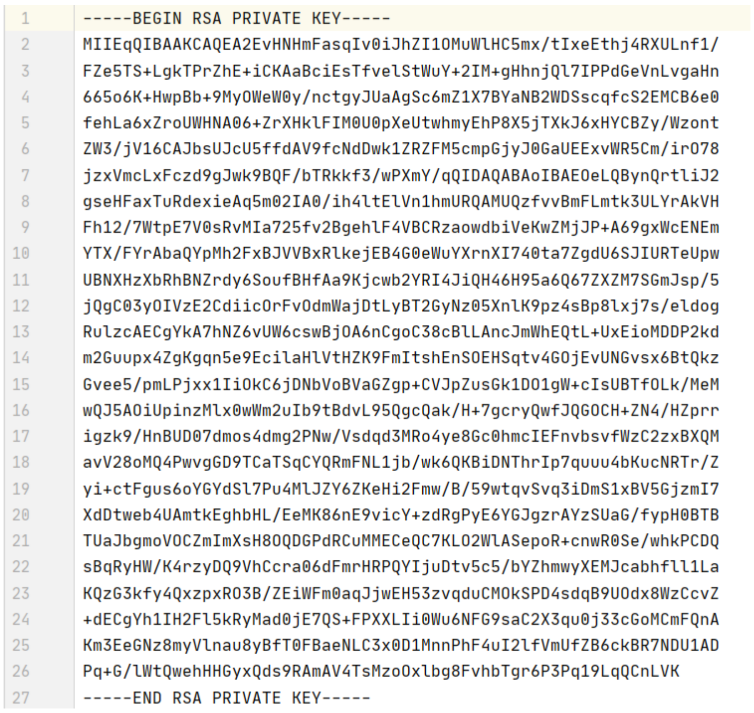
? Database Systems: MySQL, MongoDB



```
gAAAAABgfUqJBTl-yeSAyhkFBdC07aGjwKozE7tPL6kET3mNGLgcr9SKRngMiB1R7l4Eb1L1E4qEi9DNeY-GLBTXIgKFEI0nGiVGMEWCXggfnvxNhGMFPA_Sfmd4yhSTDZc3xkuf52ZNSUqDpi0Sn6EkHwXORByjdfWkSD0WRf4
```

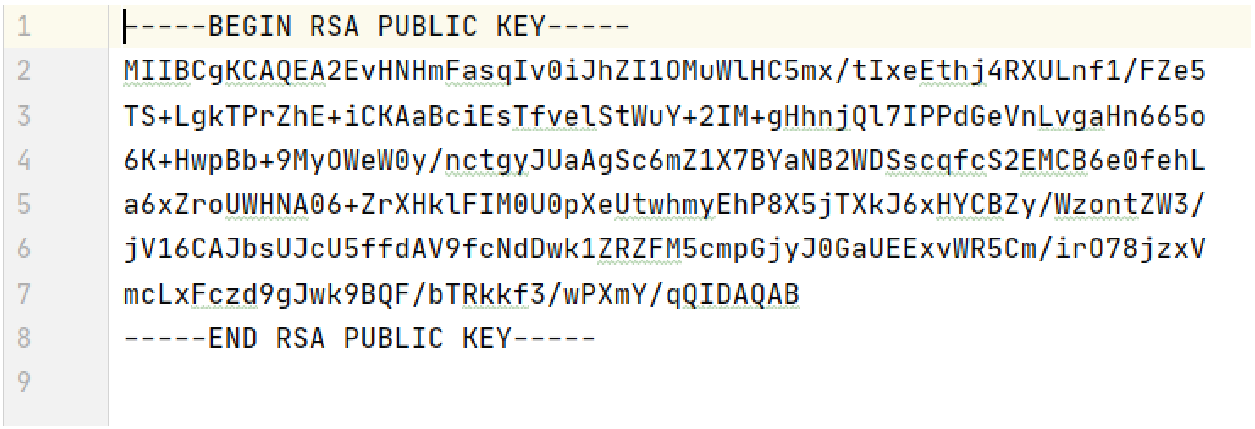
Decrypted qr code:

Sign Private key:



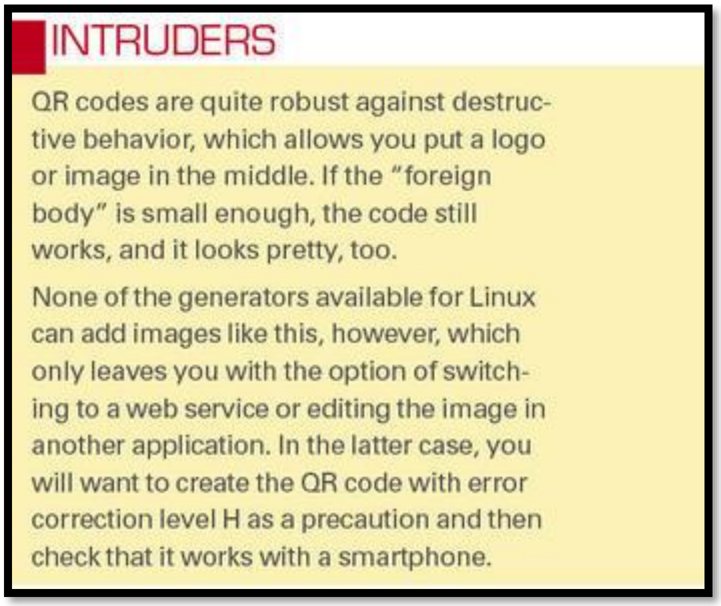
```
-----BEGIN RSA PRIVATE KEY-----
MIIEqQIBAAKCAQEA2EvHNHmFasqIv0iJhZI1OMuWlHC5mx/tIxeEthj4RXULnf1/
FZe5TS+LgkTPrZhE+iCKAaBciEsTfvelStWuY+2IM+gHhnjQl7IPPdGeVnLvgaHn
665o6K+HwpBb+9MyOWeW0y/nctgyJUaAgSc6mZ1X7BYaNB2WDSscqfcS2EMCB6e0
fehLa6xZroUWHNA06+ZrXHklFIM0U0pXeUtwhmyEhP8X5jTXkJ6xHYCBZy/Wzont
ZW3/jV16CAJbsUJcU5ffdAV9fcNdDwk1ZRZFM5cmpGjyJ0GaUEExvWR5Cm/irO78
jzxVmcLxFczd9gJwk9BQF/bTRkkf3/wPXmY/qQIDAQABAoIBAEOeLQBynQrtliJ2
gseHFaxTuRdexieAq5m02IA0/ih4ltElVn1hmURQAMUQzfvvBmFLmtk3ULYrAkVH
Fh12/7WtpE7V0sRvMIa725fv2BgehlF4VBCRzaowdbiVeKwZMjJP+A69gxWcENEm
YTX/FYrAbaQYpMh2FxBJVVBxRlkejEB4G0eWuYXrnXI740ta7ZgdU6SJIURTeUpw
UBNXHzXbRhBNZrdy6SoufBHfAa9Kjcwb2YRI4JiQH46H95a6Q67ZXZM7SGmJsp/5
jQgC03yOIVzE2CdiicOrFvOdmWajDtLyBT2GyNz05XnlK9pz4sBp8lxj7s/eldog
RulzcAECgYkA7hNZ6vUW6cswBjOA6nCgoC38cBlLAncJmWhEQtL+UxEioMDDP2kd
m2Guupx4ZgKgqn5e9EcilaHlVtHZK9FmItshEnSOEHSqtv4GOjEvUNGvsx6BtQkz
Gvee5/pmLPjxx1IiOkC6jDNbVoBVaGZgp+CVJpZusGk1DO1gW+cIsUBTfOLk/MeM
wQJ5AOiUpinzMlx0wWm2uIb9tBdvL95QgcQak/H+7gcryQwfJQGOCH+ZN4/HZprr
igzk9/HnBUD07dmos4dmg2PNw/Vsdqd3MRo4ye8Gc0hmcIEFnvbsvfWzC2zxBXQM
avV28oMQ4PwvgGD9TCaTSqCYQRmFNL1jb/wk6QKBiDNThrIp7quuu4bKucNRTr/Z
yi+ctFgus6oYGYdSl7Pu4MlJZY6ZKeHi2Fmw/B/59wtqvSvq3iDmS1xBV5GjzmI7
XdDtweb4UAmtkEghbHL/EeMK86nE9vicY+zdRgPyE6YGJgzrAYzSUaG/fypH0BTB
TUaJbgmoVOCZmImXsH8OQDGPdRCuMMECeQC7KLO2WlASepoR+cnwR0Se/whkPCDQ
sBqRyHW/K4rzyDQ9VhCcra06dFmrHRPQYIjuDtv5c5/bYZhmwyXEMJcabhfll1La
KQzG3kfy4QxzpxRO3B/ZEiWFm0aqJjwEH53zvqduCMOkSPD4sdqB9UOdx8WzCcvZ
+dECgYh1IH2Fl5kRyMad0jE7QS+FPXXLIi0Wu6NFG9saC2X3qu0j33cGoMCmFQnA
Km3EeGNz8myVlnau8yBfT0FBaeNLC3x0D1MnnPhF4uI2lfVmUfZB6ckBR7NDU1AD
Pq+G/lWtQwehHHGyxQds9RAmAV4TsMzoOxlbg8FvhbTgr6P3Pq19LqQCnLVK
-----END RSA PRIVATE KEY-----
```

Sign Public key:



```
-----BEGIN RSA PUBLIC KEY-----
MIIBCgKCAQEA2EvHNHmFasqIv0iJhZI1OMuWlHC5mx/tIxeEthj4RXULnf1/FZe5
TS+LgkTPrZhE+iCKAaBciEsTfvelStWuY+2IM+gHhnjQl7IPPdGeVnLvgaHn665o
6K+HwpBb+9MyOWeW0y/nctgyJUaAgSc6mZ1X7BYaNB2WDSscqfcS2EMCB6e0fehL
a6xZroUWHNA06+ZrXHklFIM0U0pXeUtwhmyEhP8X5jTXkJ6xHYCBZy/WzontZW3/
jV16CAJbsUJcU5ffdAV9fcNdDwk1ZRZFM5cmpGjyJ0GaUEExvWR5Cm/irO78jzxV
mcLxFczd9gJwk9BQF/bTRkkf3/wPXmY/qQIDAQAB
-----END RSA PUBLIC KEY-----
```

Following is for your information only, but if you would like to try you can give a go….



INTRUDERS

QR codes are quite robust against destructive behavior, which allows you put a logo or image in the middle. If the "foreign body" is small enough, the code still works, and it looks pretty, too.

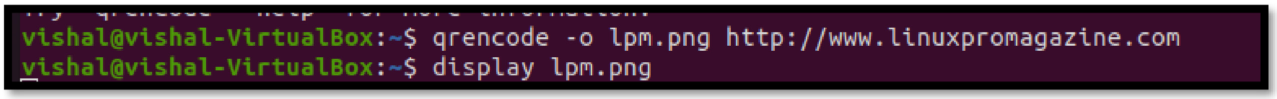
None of the generators available for Linux can add images like this, however, which only leaves you with the option of switching to a web service or editing the image in another application. In the latter case, you will want to create the QR code with error correction level H as a precaution and then check that it works with a smartphone.

The level you choose depends on the proposed use. If you expect the QR code to be exposed to external influences (e.g., wind, weather, graffiti) or if you are planning to manipulate the image deliberately (see the "Intruders" box), you will want the highest level, *H*. On a business card, the lowest level, *L*, is sufficient. If in doubt, simply omit the *-l* parameter. The program then uses level *L* by default.

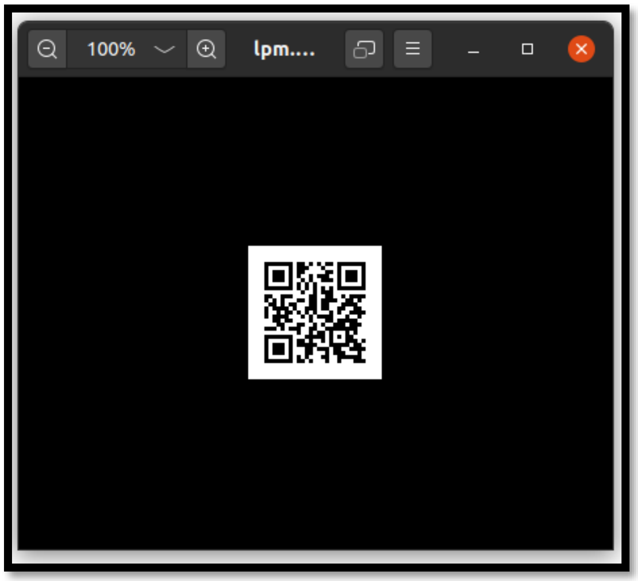
Variety

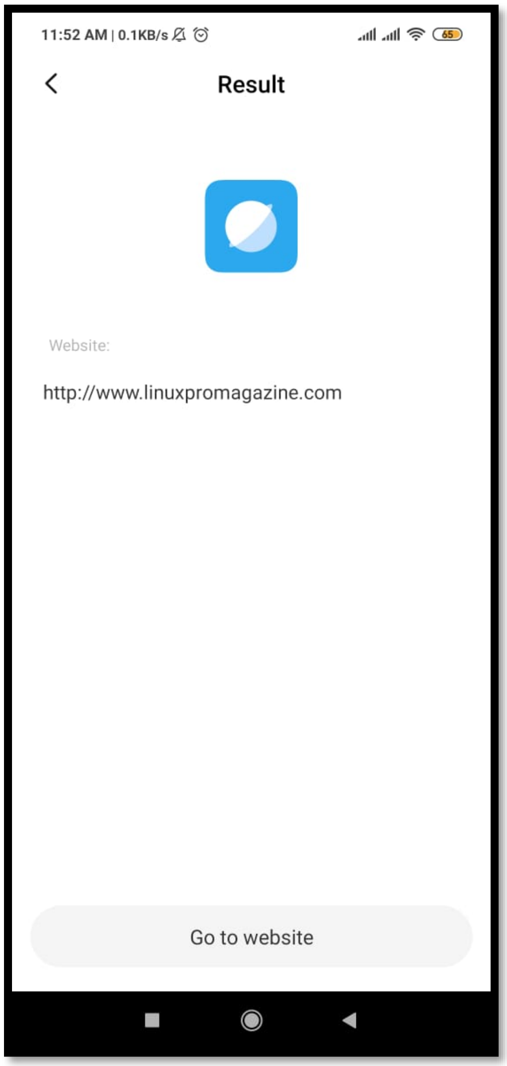
Qrencode lets you encode any text, including URLs. You only need to use single quotes if blanks and non-standard characters appear in the text:

$ qrencode -o lpm.png http://www.linuxpromagazine.com



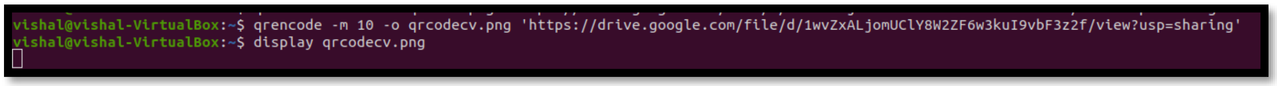
```
vishal@vishal-VirtualBox:~$ qrencode -o lpm.png http://www.linuxpromagazine.com
vishal@vishal-VirtualBox:~$ display lpm.png
```



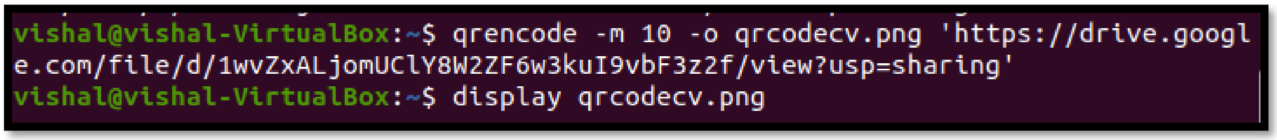


Website:

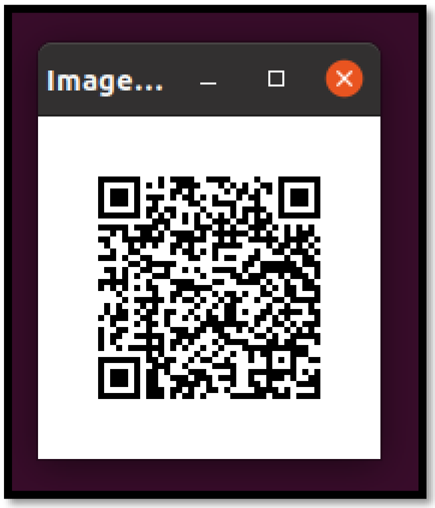
http://www.linuxpromagazine.com

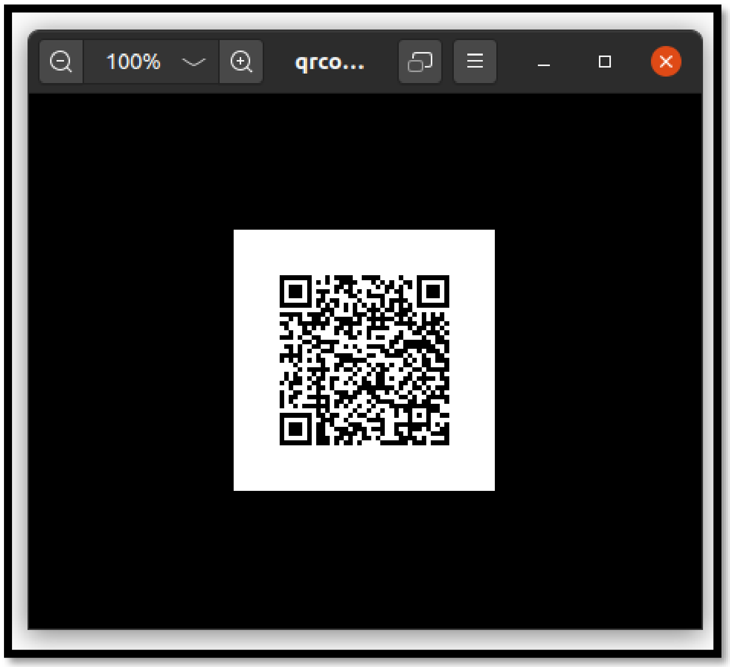


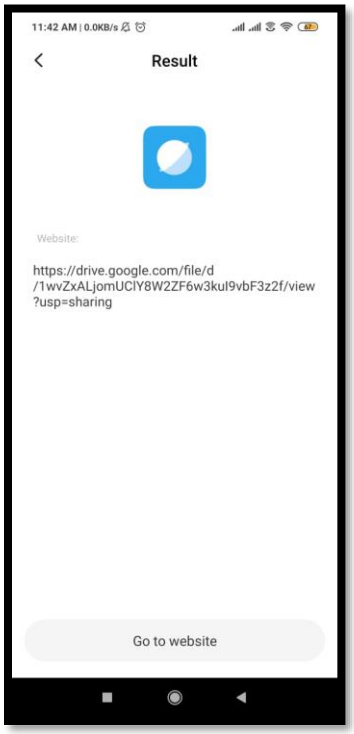
```
vishal@vishal-VirtualBox:~$ qrencode -m 10 -o qrcodecv.png 'https://drive.google.com/file/d/1wvZxALjomUClY8W2ZF6w3kuI9vbF3z2f/view?usp=sharing'
vishal@vishal-VirtualBox:~$ display qrcodecv.png
```



```
vishal@vishal-VirtualBox:~$ qrencode -m 10 -o qrcodecv.png 'https://drive.googl
e.com/file/d/1wvZxALjomUClY8W2ZF6w3kuI9vbF3z2f/view?usp=sharing'
vishal@vishal-VirtualBox:~$ display qrcodecv.png
```





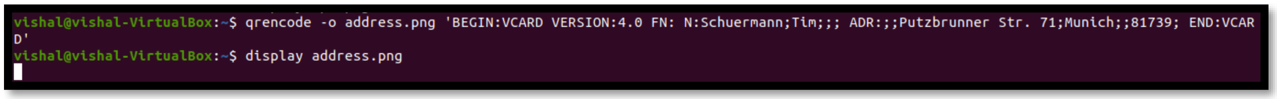


https://drive.google.com/file/d
/1wvZxALjomUClY8W2ZF6w3kul9vbF3z2f/view
?usp=sharing

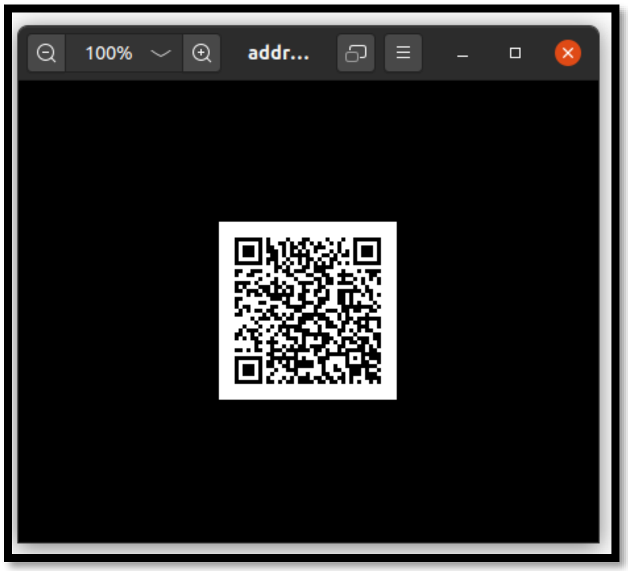
All modern QR code readers recognize that this is a web address and will offer to open it in the browser.

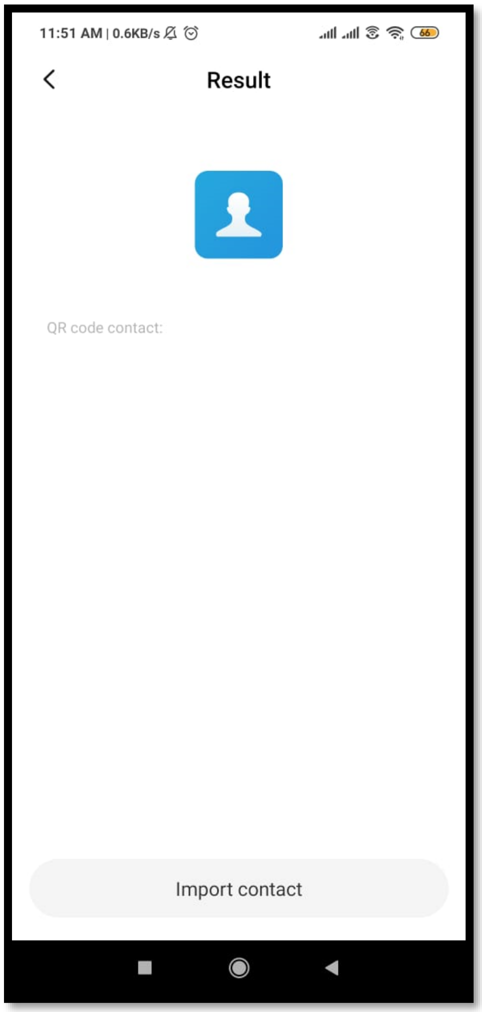
Getting an QR-encoded address automatically to enter a smartphone owner’s address book is more difficult. In this case, you need to enter your address and some additional, cryptic information:

$ qrencode -o address.png 'BEGIN:VCARD VERSION:4.0 FN: N:Schuermann;Tim;;; ADR:;;Putzbrunner Str. 71;Munich;;81739; END:VCARD'



```
vishal@vishal-VirtualBox:~$ qrencode -o address.png 'BEGIN:VCARD VERSION:4.0 FN: N:Schuermann;Tim;;; ADR:;;Putzbrunner Str. 71;Munich;;81739; END:VCAR
D'
vishal@vishal-VirtualBox:~$ display address.png
```





Typing this is not exactly easy, so to make life easier, you might prefer a QR code generator with a graphical user interface, such as QtQR.

Of the tested programs, Portable QR-Code Generator offers the greatest functionality. Data entry is also convenient in a variety of tabs. Qrencode creates QR codes more quickly, and it is easy to scripted so you generate QR codes in batches. KBarcode4-light is deprecated, but it is the only program that can generate PDF files; the EPS images created by Qrencode offer similar output.

Vunerabilities: https://analyticsindiamag.com/25-years-of-qr-codes-a-look-at-vulnerabilities-of-this-popular-payment-method/

Currently, over 23% of Trojans and viruses are transmitted via QR codes. On the 25th anniversary of QR codes, its creator, Masahiro Hara wants to make QR scanning more secure.

Usually, in the case of QR scanning, possible scenarios of attacks can be summarised as follows:

- QR codes cannot be hacked. One way hackers to infiltrate this system by changing the QR code added in the poster. These fake posters can be circulated in the public domains and clueless customers scan these fake QR codes and end up visiting phishing websites.
- This usually happens because of the increase in the number of mobile users. Mobiles make it hard to verify the full link in the address bar. This makes users more vulnerable. When they use this phishing page to login, their passwords are compromised.
- An attacker might set up a fake website and redirect users by changing the QR Code. This is dangerous if some form of credentials are needed to access the website. The user has no possibility to verify that the link is not modified.
- SQL injection is another form of attack that occurs when SQL queries are made with user input text inserted into the query string. QR code readers are subject to data injection into their structured objects when they attempt to interpret the data of a QR code.
- A malicious party can create a QR code that injects arbitrary strings into a user's data structures potentially causing harm to the user.
- Criminals can simply prepare malicious QR codes and affix them over legitimate codes which may result in victims inadvertently making payments to a criminal rather a legitimate service provider.
- QRLJacking or Quick Response Code Login Jacking is a simple social engineering attack vector capable of session hijacking affecting all applications that rely on "Login with QR code" feature as a secure way to login to accounts.
- QRLJacking attack gives attackers the ability to apply a full account hijacking scenario on the vulnerable Login with QR Code feature resulting in accounts stealing and reputation affection.

Conclusion:

1. The inclusion of QR Code adds an extra level of security to the encrypted message and the receiver can access the original message very quickly, just by scanning the QR Code.
2. QR Codes have fast response time and have large storage capacity, QR Codes can be used perfectly to send encrypted data (messages) to the receiver.
3. Person can even use this method to keep his important documents, like passport number, pan-card id, social security number, perfectly secured with him all the time, without the information getting leaked to outside world.