

LABORATORY

CEL62: Cryptography and System Security Winter 2021

Experiment 7:	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.

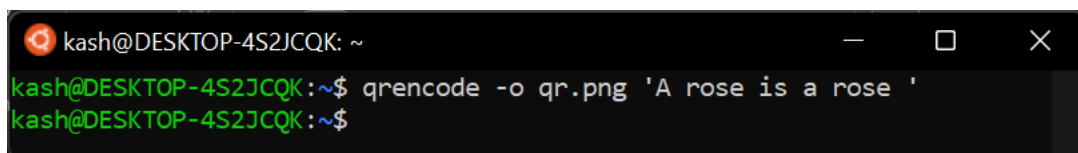
Data Grabber

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.

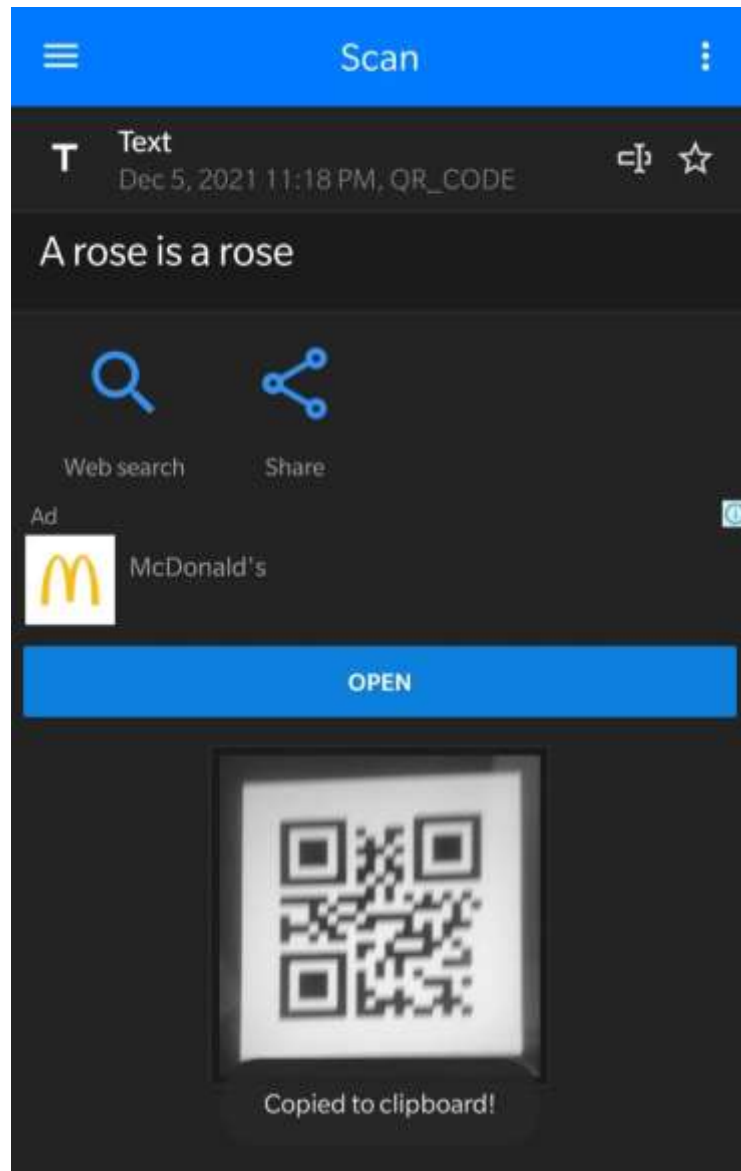
3 PROCEDURE/TASK

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!'
```

A terminal window with a dark background. The prompt is 'kash@DESKTOP-4S2JCQK: ~'. The command 'qrencode -o qr.png 'A rose is a rose '' is entered and executed. The prompt changes to 'kash@DESKTOP-4S2JCQK: ~\$' after the command is run.

```
kash@DESKTOP-4S2JCQK: ~$ qrencode -o qr.png 'A rose is a rose '
kash@DESKTOP-4S2JCQK: ~$
```



QR Code relevant A rose is a rose will pop up: Qrcode encoding the string “A rose is a rose” as a QR code and storing it in the qr.png file.

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

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

or ASCII output.

```
kash@DESKTOP-4S2JCQK:~$ qrencode -o qr.txt -t ASCII 'Never tell me the odds'
kash@DESKTOP-4S2JCQK:~$ cat qr.txt

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

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

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:

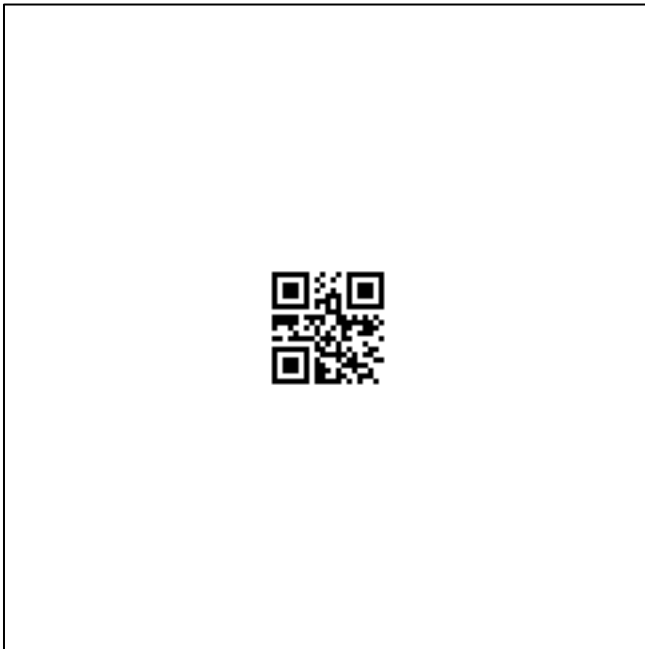
```
$ qrencode -m 10 -o qrcode.png 'Hello World!'
```



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!'
```

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.



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.

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.

Code for generation of keys:

```
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("company_public_key.pem", "company_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()
```

Applicant Code:

```
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('company_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_file.bin', 'wb')
s.write(signature)
# Encrypting the QR Code file
cipher = Fernet(symkey)
qrcode_encrypted = cipher.encrypt(qrcode_png)
e = open('encrypted_qrcode.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)

```

Company side Code:

```

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_qrcode.bin')
encrypted_symmetric_key = file_open('encrypted_symmetric_key.bin')
signature = file_open('signature_file.bin')

privkey_company = rsa.PrivateKey.load_pkcs1(file_open('company_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)

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

```

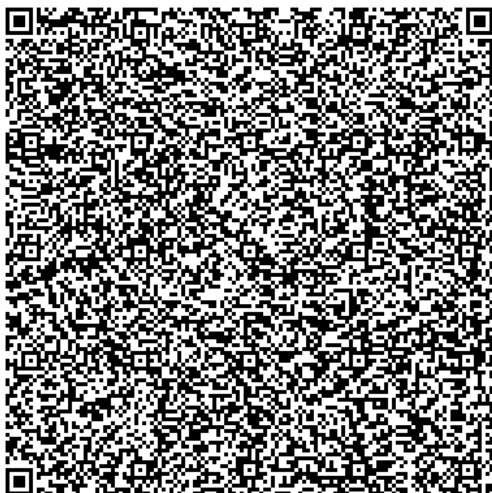
```
(MyEnv) C:\Users\KashMir\Desktop\Kashish\Semester V\CSS Lab\Experiment 7>python keys.py
```

```
(MyEnv) C:\Users\KashMir\Desktop\Kashish\Semester V\CSS Lab\Experiment 7>python enc.py
```

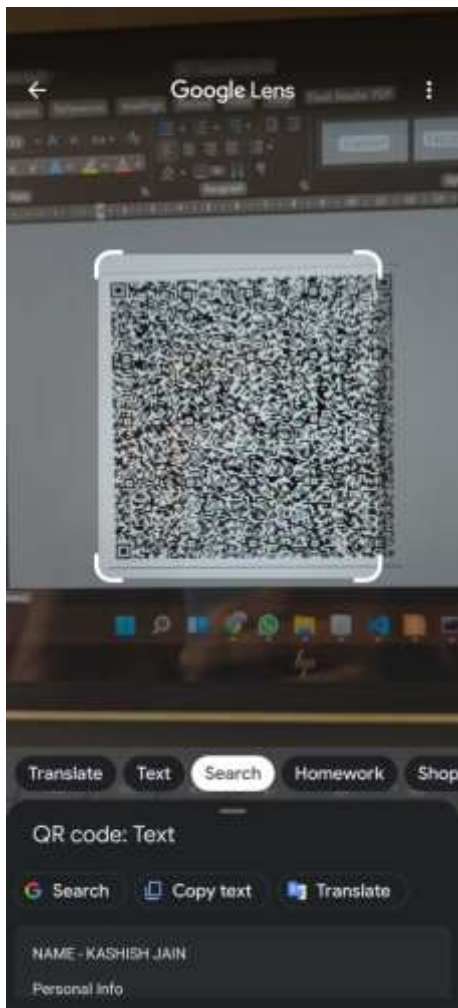
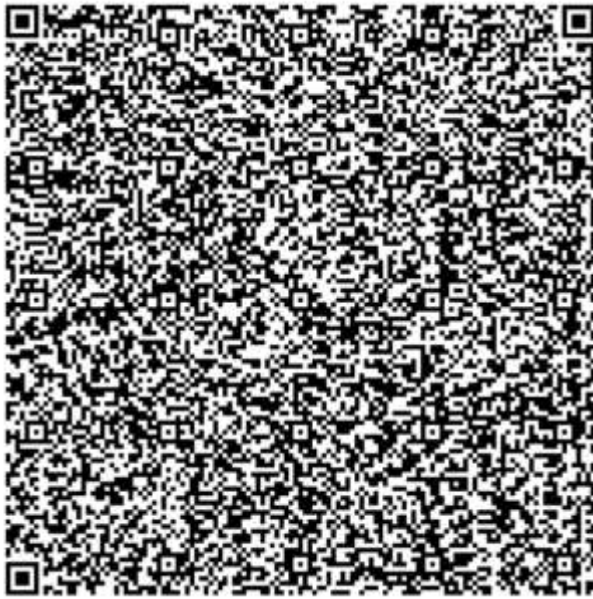
```
(MyEnv) C:\Users\KashMir\Desktop\Kashish\Semester V\CSS Lab\Experiment 7>python dec.py
Signature has been verified.
```

I tried to read pdf and word files to turn into qr codes. But the they could not be read properly and resulted in gibberish text.

Original QR code:



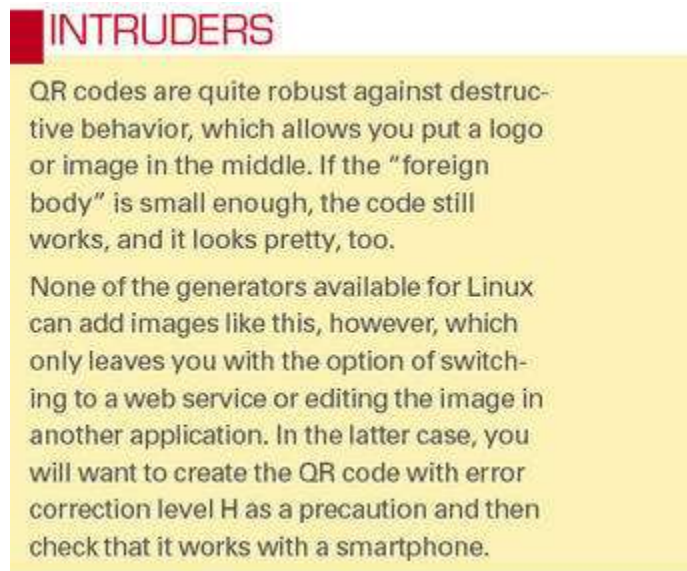
Decrypted QR code:



cv.txt

```
1  NAME - KASHISH JAIN
2
3  Personal Info
4  Address: Mumbai - 28
5  Mobile: +91 9821261411
6  Email: jainvkashish@gmail.com
7
8  Projects and Experience
9
10 Research Intern
11 2020 - 2021
12 Working on a project in Django to solve password leak issues by
13 leveraging the use of ML algorithms with cybersecurity to train the
14 model on the pattern and speed of the user.
15
16 Counter Share Analysis for FMCG companies
17 2021
18 An intelligent counter share analyzer that can analyze video,
19 images and derive the % of brand presence on a particular section
20 of a modern retail chain, automating a manual process.
21
22 Image Classifier Web App
23 2020
24 A web app with functionalities such as Bulk Training Model and
25 Inference, Retraining model using the given images, and Training
26 the model by changing parameters using Django and PyTorch.
27
28 Accomplishments
29
30 CSE C, D, E, F, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, AA, AB, AC, AD, AE, AF, AG, AH, AI, AJ, AK, AL, AM, AN, AO, AP, AQ, AR, AS, AT, AU, AV, AW, AX, AY, AZ, BA, BB, BC, BD, BE, BF, BG, BH, BI, BJ, BK, BL, BM, BN, BO, BP, BQ, BR, BS, BT, BU, BV, BW, BX, BY, BZ, CA, CB, CC, CD, CE, CF, CG, CH, CI, CJ, CK, CL, CM, CN, CO, CP, CQ, CR, CS, CT, CU, CV, CW, CX, CY, CZ, DA, DB, DC, DD, DE, DF, DG, DH, DI, DJ, DK, DL, DM, DN, DO, DP, DQ, DR, DS, DT, DU, DV, DW, DX, DY, DZ, EA, EB, EC, ED, EE, EF, EG, EH, EI, EJ, EK, EL, EM, EN, EO, EP, EQ, ER, ES, ET, EU, EV, EW, EX, EY, EZ, FA, FB, FC, FD, FE, FF, FG, FH, FI, FJ, FK, FL, FM, FN, FO, FP, FQ, FR, FS, FT, FU, FV, FW, FX, FY, FZ, GA, GB, GC, GD, GE, GF, GG, GH, GI, GJ, GK, GL, GM, GN, GO, GP, GQ, GR, GS, GT, GU, GV, GW, GX, GY, GZ, HA, HB, HC, HD, HE, HF, HG, HH, HI, HJ, HK, HL, HM, HN, HO, HP, HQ, HR, HS, HT, HU, HV, HW, HX, HY, HZ, IA, IB, IC, ID, IE, IF, IG, IH, II, IJ, IK, IL, IM, IN, IO, IP, IQ, IR, IS, IT, IU, IV, IW, IX, IY, IZ, JA, JB, JC, JD, JE, JF, JG, JH, JI, JJ, JK, JL, JM, JN, JO, JP, JQ, JR, JS, JT, JU, JV, JW, JX, JY, JZ, KA, KB, KC, KD, KE, KF, KG, KH, KI, KJ, KK, KL, KM, KN, KO, KP, KQ, KR, KS, KT, KU, KV, KW, KX, KY, KZ, LA, LB, LC, LD, LE, LF, LG, LH, LI, LJ, LK, LL, LM, LN, LO, LP, LQ, LR, LS, LT, LU, LV, LW, LX, LY, LZ, MA, MB, MC, MD, ME, MF, MG, MH, MI, MJ, MK, ML, MM, MN, MO, MP, MQ, MR, MS, MT, MU, MV, MW, MX, MY, MZ, NA, NB, NC, ND, NE, NF, NG, NH, NI, NJ, NK, NL, NM, NN, NO, NP, NQ, NR, NS, NT, NU, NV, NW, NX, NY, NZ, OA, OB, OC, OD, OE, OF, OG, OH, OI, OJ, OK, OL, OM, ON, OO, OP, OQ, OR, OS, OT, OU, OV, OW, OX, OY, OZ, PA, PB, PC, PD, PE, PF, PG, PH, PI, PJ, PK, PL, PM, PN, PO, PP, PQ, PR, PS, PT, PU, PV, PW, PX, PY, PZ, QA, QB, QC, QD, QE, QF, QG, QH, QI, QJ, QK, QL, QM, QN, QO, QP, QQ, QR, QS, QT, QU, QV, QW, QX, QY, QZ, RA, RB, RC, RD, RE, RF, RG, RH, RI, RJ, RK, RL, RM, RN, RO, RP, RQ, RR, RS, RT, RU, RV, RW, RX, RY, RZ, SA, SB, SC, SD, SE, SF, SG, SH, SI, SJ, SK, SL, SM, SN, SO, SP, SQ, SR, SS, ST, SU, SV, SW, SX, SY, SZ, TA, TB, TC, TD, TE, TF, TG, TH, TI, TJ, TK, TL, TM, TN, TO, TP, TQ, TR, TS, TT, TU, TV, TW, TX, TY, TZ, UA, UB, UC, UD, UE, UF, UG, UH, UI, UJ, UK, UL, UM, UN, UO, UP, UQ, UR, US, UT, UY, UZ, VA, VB, VC, VD, VE, VF, VG, VH, VI, VJ, VK, VL, VM, VN, VO, VP, VQ, VR, VS, VT, VU, VV, VW, VX, VY, VZ, WA, WB, WC, WD, WE, WF, WG, WH, WI, WJ, WK, WL, WM, WN, WO, WP, WQ, WR, WS, WT, WU, WV, WW, WX, WY, WZ, XA, XB, XC, XD, XE, XF, XG, XH, XI, XJ, XK, XL, XM, XN, XO, XP, XQ, XR, XS, XT, XU, XV, XW, XX, XY, XZ, YA, YB, YC, YD, YE, YF, YG, YH, YI, YJ, YK, YL, YM, YN, YO, YP, YQ, YR, YS, YT, YU, YV, YW, YX, YY, YZ, ZA, ZB, ZC, ZD, ZE, ZF, ZG, ZH, ZI, ZJ, ZK, ZL, ZM, ZN, ZO, ZP, ZQ, ZR, ZS, ZT, ZU, ZV, ZW, ZX, ZY, ZZ, AA, AB, AC, AD, AE, AF, AG, AH, AI, AJ, AK, AL, AM, AN, AO, AP, AQ, AR, AS, AT, AU, AV, AW, AX, AY, AZ, BA, BB, BC, BD, BE, BF, BG, BH, BI, BJ, BK, BL, BM, BN, BO, BP, BQ, BR, BS, BT, BU, BV, BW, BX, BY, BZ, CA, CB, CC, CD, CE, CF, CG, CH, CI, CJ, CK, CL, CM, CN, CO, CP, CQ, CR, CS, CT, CU, CV, CW, CX, CY, CZ, DA, DB, DC, DD, DE, DF, DG, DH, DI, DJ, DK, DL, DM, DN, DO, DP, DQ, DR, DS, DT, DU, DV, DW, DX, DY, DZ, EA, EB, EC, ED, EE, EF, EG, EH, EI, EJ, EK, EL, EM, EN, EO, EP, EQ, ER, ES, ET, EU, EV, EW, EX, EY, EZ, FA, FB, FC, FD, FE, FF, FG, FH, FI, FJ, FK, FL, FM, FN, FO, FP, FQ, FR, FS, FT, FU, FV, FW, FX, FY, FZ, GA, GB, GC, GD, GE, GF, GG, GH, GI, GJ, GK, GL, GM, GN, GO, GP, GQ, GR, GS, GT, GU, GV, GW, GX, GY, GZ, HA, HB, HC, HD, HE, HF, HG, HH, HI, HJ, HK, HL, HM, HN, HO, HP, HQ, HR, HS, HT, HU, HV, HW, HX, HY, HZ, IA, IB, IC, ID, IE, IF, IG, IH, II, IJ, IK, IL, IM, IN, IO, IP, IQ, IR, IS, IT, IU, IV, IW, IX, IY, IZ, JA, JB, JC, JD, JE, JF, JG, JH, JI, JJ, JK, JL, JM, JN, JO, JP, JQ, JR, JS, JT, JU, JV, JW, JX, JY, JZ, KA, KB, KC, KD, KE, KF, KG, KH, KI, KJ, KK, KL, KM, KN, KO, KP, KQ, KR, KS, KT, KU, KV, KW, KX, KY, KZ, LA, LB, LC, LD, LE, LF, LG, LH, LI, LJ, LK, LL, LM, LN, LO, LP, LQ, LR, LS, LT, LU, LV, LW, LX, LY, LZ, MA, MB, MC, MD, ME, MF, MG, MH, MI, MJ, MK, ML, MM, MN, MO, MP, MQ, MR, MS, MT, MU, MV, MW, MX, MY, MZ, NA, NB, NC, ND, NE, NF, NG, NH, NI, NJ, NK, NL, NM, NN, NO, NP, NQ, NR, NS, NT, NU, NV, NW, NX, NY, NZ, OA, OB, OC, OD, OE, OF, OG, OH, OI, OJ, OK, OL, OM, ON, OO, OP, OQ, OR, OS, OT, OU, OV, OW, OX, OY, OZ, PA, PB, PC, PD, PE, PF, PG, PH, PI, PJ, PK, PL, PM, PN, PO, PP, PQ, PR, PS, PT, PU, PV, PW, PX, PY, PZ, QA, QB, QC, QD, QE, QF, QG, QH, QI, QJ, QK, QL, QM, QN, QO, QP, QQ, QR, QS, QT, QU, QV, QW, QX, QY, QZ, RA, RB, RC, RD, RE, RF, RG, RH, RI, RJ, RK, RL, RM, RN, RO, RP, RQ, RR, RS, RT, RU, RV, RW, RX, RY, RZ, SA, SB, SC, SD, SE, SF, SG, SH, SI, SJ, SK, SL, SM, SN, SO, SP, SQ, SR, SS, ST, SU, SV, SW, SX, SY, SZ, TA, TB, TC, TD, TE, TF, TG, TH, TI, TJ, TK, TL, TM, TN, TO, TP, TQ, TR, TS, TT, TU, TV, TW, TX, TY, TZ, UA, UB, UC, UD, UE, UF, UG, UH, UI, UJ, UK, UL, UM, UN, UO, UP, UQ, UR, US, UT, UY, UZ, VA, VB, VC, VD, VE, VF, VG, VH, VI, VJ, VK, VL, VM, VN, VO, VP, VQ, VR, VS, VT, VU, VV, VW, VX, VY, VZ, WA, WB, WC, WD, WE, WF, WG, WH, WI, WJ, WK, WL, WM, WN, WO, WP, WQ, WR, WS, WT, WU, WV, WW, WX, WY, WZ, XA, XB, XC, XD, XE, XF, XG, XH, XI, XJ, XK, XL, XM, XN, XO, XP, XQ, XR, XS, XT, XU, XV, XW, XX, XY, XZ, YA, YB, YC, YD, YE, YF, YG, YH, YI, YJ, YK, YL, YM, YN, YO, YP, YQ, YR, YS, YT, YU, YV, YW, YX, YY, YZ, ZA, ZB, ZC, ZD, ZE, ZF, ZG, ZH, ZI, ZJ, ZK, ZL, ZM, ZN, ZO, ZP, ZQ, ZR, ZS, ZT, ZU, ZV, ZW, ZX, ZY, ZZ
```

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



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
```



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'
```

```
KashishDESKTOP-4523CQK ~$ qrencode -o address.png 'BEGIN:VCARD VERSION:4.0 FN: Kashish;Jain;;;ADR;;;Dadar;Mumbai;400028; END:VCARD'
```



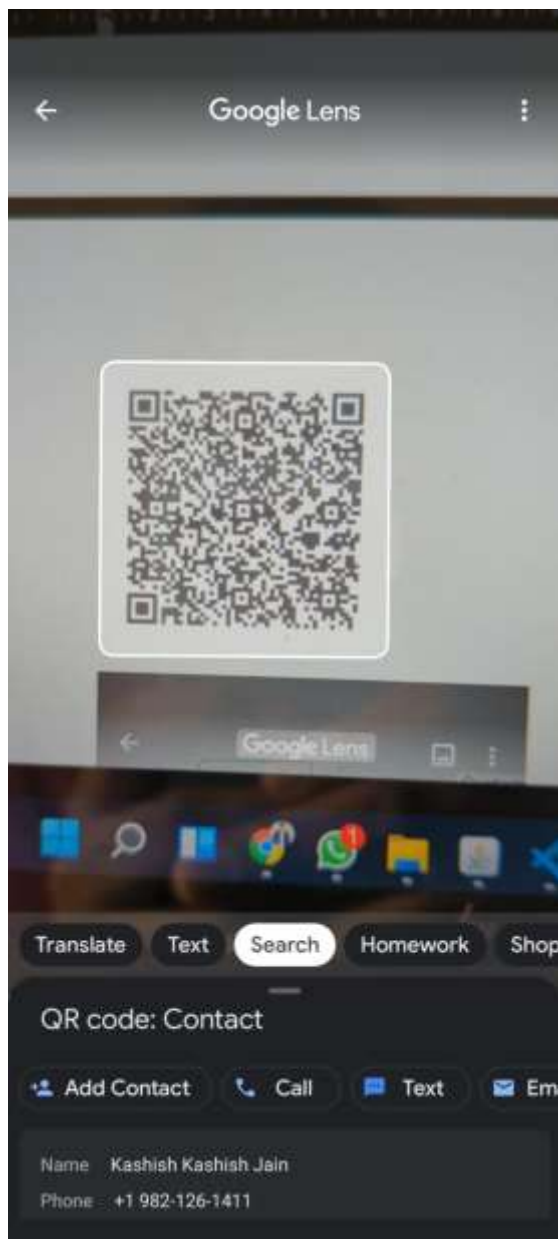
Typing this is not exactly easy, so to make life easier, you might prefer a QR code generator with a graphical user interface, such as [QtQR](#).

Demo -

I have used email encoding in QR code –



The screenshot shows the QtQR web application interface. On the left is a sidebar with navigation links: Sign-in, QR Code, Generate (Static QR Code), Scan, About, What's a QR Code?, Privacy, Terms of use, and Imprint. The main area has a green header 'QR Code Generator' and tabs for URL, FREE TEXT, CONTACT (selected), and AI. The CONTACT tab contains form fields for Name (Kashish Jain), First Name (Kashish), Organization, Email (kashishjain32@gmail.com), Phone (9821261411), cell, Fax, and Email. To the right, there's a 'Static QR Code' section with a QR code image and a note: 'Content is directly encoded in the image. If you want to be able to update it later, create a Dynamic QR Code.' A 'MAKE DYNAMIC' button is visible above the QR code. A green circular button with a plus sign is at the bottom right.



Of the tested programs, Portable QR-Code Generator offers the greatest functionality. Data entry is also convenient in a variety of tabs. **Qrcode 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 Qrcode offer similar output.

Vulnerabilities: <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. QR codes are an easy and quick way to transmit information,
 2. The security levels of QR codes can be set depending on how confidential/secure we want the information contained in it to be.
 3. QR codes also provide error correction measures so that information can be extracted from the code even if part of the code is corrupted.
 4. QR code themselves can be encrypted, so that only people with keys to the encrypted message can scan the code.
-