COMP 9121 Week 11

1. Bitcoin mining.

In this question, you will need to understand "mining" of bitcoins. It is based on the fact that nobody can find the input value that results a given hashed value. However, given the input, it is easy to verify its hashed value.

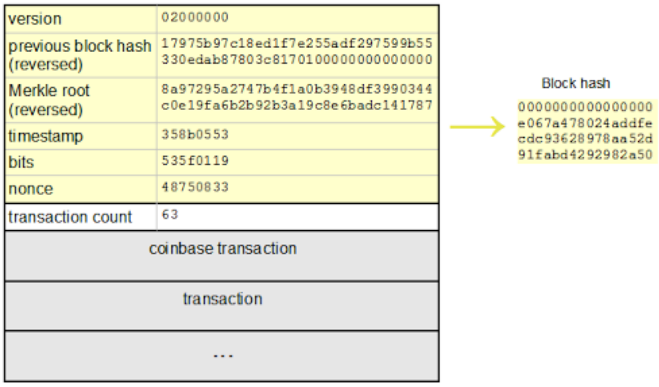
Bitcoin mining can be regarded as "proof of work" [1]. "Proof of work" is a piece of data which is difficult (costly, time-consuming) to produce but easy for others to verify and which satisfies certain requirements. Producing a proof of work can be a random process with low probability so that a lot of trial and error is required on average before a valid proof of work is generated. Bitcoin uses the Hashcash proof of work system.

Let's say the base string that we are going to do work on is "Hello, world!". Our target is to find a variation of it that SHA-256 hashes to a value beginning with '000' (12-bit '0's). We vary the string by adding an integer value to the end called a nonce and incrementing it each time. Finding a match for "Hello, world!" takes us 4251 tries (but happens to have zeroes in the first four digits):

"Hello, world!0" => 1312af178c253f84028d480a6adc1e25e81caa44c749ec81976192e2ec934c64
"Hello, world!1" => e9afc424b79e4f6ab42d99c81156d3a17228d6e1eef4139be78e948a9332a7d8
"Hello, world!2" => ae37343a357a8297591625e7134cbea22f5928be8ca2a32aa475cf05fd4266b7
...
"Hello, world!4248" =>
6e110d98b388e77e9c6f042ac6b497cec46660deef75a55ebc7cfdf65cc0b965
"Hello, world!4249" =>
c004190b822f1669cac8dc37e761cb73652e7832fb814565702245cf26ebb9e6
"Hello, world!4250" =>
0000c3af42fc31103f1fdc0151fa747ff87349a4714df7cc52ea464e12dcd4e9

(Note: the hashed value is in hexadecimal format, each hexadecimal number is equivalent to 4 bits.)

4251 hashes on a modern computer is not very much work (most computers can achieve at least 4 million hashes per second). Bitcoin automatically varies the difficulty. A successful hash requires a value beginning with 64-bit '0's.



version	02000000
previous block hash (reversed)	17975b97c18ed1f7e255adf297599b55 330edab87803c81701000000000000000
Merkle root (reversed)	8a97295a2747b4f1a0b3948df3990344 c0e19fa6b2b92b3a19c8e6badc141787
timestamp	358b0553
bits	535f0119
nonce	48750833
transaction count	63
coinbase transaction	
transaction	
...	

Block hash
0000000000000000
e067a478024addfe
cdc93628978aa52d
91fabd4292982a50

The figure above shows a transaction block. You need to "guess" the nonce so that the hash of the yellow field begins with 64-bit '0's. There is no fast way to guess the nonce. All you can do is exhaustive search. You can assume that other fields are given in this example.

Figure source http://www.righto.com/2014/02/bitcoin-mining-hard-way-algorithms.html

Question.
(1) What is the successful probability if you guess once? $\frac{1}{2^{64}} = p$
(2) What is the successful probability if 10^7 computers guess and each guesses 10^{11} times?

$$\approx np = np - \underbrace{\binom{n}{2}p^2 + \binom{n}{3}p^3}_{\text{very small}}$$

[1] https://en.bitcoin.it/wiki/Proof_of_work $n = 10^{18}$/second $\Rightarrow 1 - \underbrace{(1-p)}_{\text{failure probability each time}}^{10^{18}}$

guess #

2. Private Key.

In this experiment, you will need to check public and private keys in a Linux system.
Use your Linux/MacOS/Windows system. Use the following command to generate a pair of public and private keys.

```
ssh-keygen -b 1024 -t rsa
```

If you are using the latest version of MacOS or Windows, you need to use the following command.

or

```
ssh-keygen -b 1024 -t rsa -m PEM
```

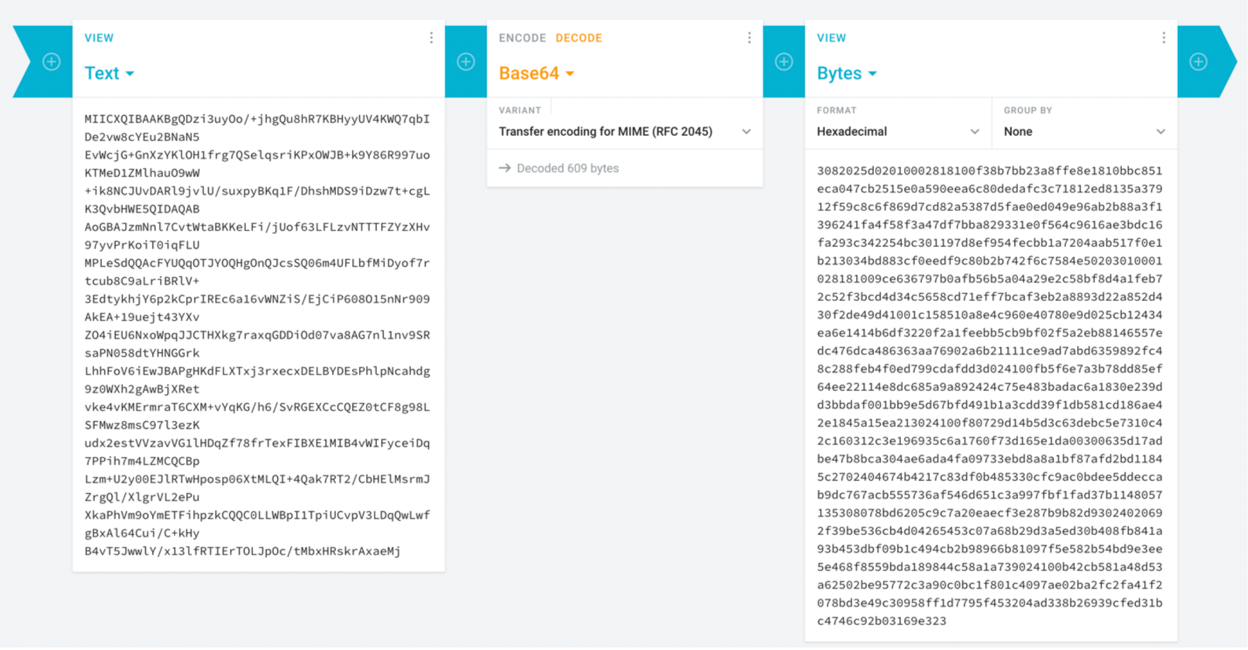
Enter "testfile" for file name. (You can choose any name you like.)
Do not enter anything for passphrase. (Otherwise, you need password to access the private key.)
Then, your public key is stored in testfile.pub and your private key is stored in testfile.
Use the following command to check your private key

```
Cat testfile
```

Then it will show something like

```
david@david-Latitude-E5570:~/keytry$ cat testfile
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDzi3uyOo/+jhgQu8hR7KBHyyUV4KWQ7qbIDe2vw8cYEu2BNaN5
EvWcjG+GnXzYKlOH1frg7QSelqsriKPxOWJB+k9Y86R997uoKTMeD1ZMlhauO9wW
+ik8NCJUvDARl9jvlU/suxpyBKq1F/DhshMDS9iDzw7t+cgLK3QvbHWE5QIDAQAB
AoGBAJzmNnl7CvtWtaBKKeLFi/jUof63LFLzvNTTTFZYzXHv97yvPrKoiT0iqFLU
MPLeSdQQAcFYUQqOTJYOQHgOnQJcsSQ06m4UFLbfMiDyof7rtcub8C9aLriBRlV+
3EdtykhjY6p2kCprIREc6a16vWNZiS/EjCiP608O15nNr909AkEA+19uejt43YXv
ZO4iEU6NxoWpqJJCTHXkg7raxqGDDiOd07va8AG7nl1nv9SRsaPN058dtYHNGGrk
LhhFoV6iEwJBAPgHKdFLXTxj3rxecxDELBYDEsPhlpNcahdg9z0WXh2gAwBjXRet
vke4vKMErmraT6CXM+vYqKG/h6/SvRGEXCcCQEZ0tCF8g98LSFMwz8msC97l3ezK
udx2estVVzavVG1lHDqZf78frTexFIBXE1MIB4vWIFyceiDq7PPih7m4LZMCQCBp
Lzm+U2y00EJlRTwHposp06XtMLQI+4Qak7RT2/CbHElMsrmJZrgQl/XlgrVL2ePu
XkaPhVm9oYmETFihpzkCQQC0LLWBpI1TpiUCvpV3LDqQwLwfgBxAl64Cui/C+kHy
B4vT5JwwlY/x13lfRTIErTOLJpOc/tMbxHRskrAxaeMj
-----END RSA PRIVATE KEY-----
```

The above red string is coded by BASE64. You need to decode it by BASE64-HEX (RFC2045) to convert it to hexadecimal numbers. Use the following website.
https://cryptii.com/pipes/base64-to-hex



```
MIICXQIBAAKBgQDzi3uyOo/+jhgQu8hR7KBHyyUV4KWQ7qbI
De2vw8cYEu2BNaN5
EvWcjG+GnXzYKlOH1frg7QSelqsriKPxOWJB+k9Y86R997uo
KTMeD1ZMlhauO9wW
+ik8NCJUvDARl9jvlU/suxpyBKq1F/DhshMDS9iDzw7t+cgL
K3QvbHWE5QIDAQAB
AoGBAJzmNnl7CvtWtaBKKeLFi/jUof63LFLzvNTTTFZYzXHv
97yvPrKoiT0iqFLU
MPLeSdQQAcFYUQqOTJYOQHgOnQJcsSQ06m4UFLbfMiDyof7r
tcub8C9aLriBRlV+
3EdtykhjY6p2kCprIREc6a16vWNZiS/EjCiP608O15nNr909
AkEA+19uejt43YXv
ZO4iEU6NxoWpqJJCTHXkg7raxqGDDiOd07va8AG7nl1nv9SR
saPN058dtYHNGGrk
LhhFoV6iEwJBAPgHKdFLXTxj3rxecxDELBYDEsPhlpNcahdg
9z0WXh2gAwBjXRet
vke4vKMErmraT6CXM+vYqKG/h6/SvRGEXCcCQEZ0tCF8g98L
SFMwz8msC97l3ezK
udx2estVVzavVG1lHDqZf78frTexFIBXE1MIB4vWIFyceiDq
7PPih7m4LZMCQCBp
Lzm+U2y00EJlRTwHposp06XtMLQI+4Qak7RT2/CbHElMsrmJ
ZrgQl/XlgrVL2ePu
XkaPhVm9oYmETFihpzkCQQC0LLWBpI1TpiUCvpV3LDqQwLwf
gBxAl64Cui/C+kHy
B4vT5JwwlY/x13lfRTIErTOLJpOc/tMbxHRskrAxaeMj
```

VARIANT
Transfer encoding for MIME (RFC 2045)

→ Decoded 609 bytes

FORMAT
Hexadecimal

GROUP BY
None

```
3082025d02010002818100f38b7bb23a8ffe8e1810bbc851
eca047cb2515e0a590eea6c80dedafc3c71812ed8135a379
12f59c8c6f869d7cd82a5387d5fae0ed049e96ab2b88a3f1
396241fa4f58f3a47df7bba829331e0f564c9616ae3bdc16
fa293c342254bc301197d8ef954fecbb1a7204aab517f0e1
b213034bd883cf0eedf9c80b2b742f6c7584e50203010001
028181009ce636797b0afb56b5a04a29e2c58bf8d4a1feb7
2c52f3bcd4d34c5658cd71eff7bcaf3eb2a8893d22a852d4
30f2de49d41001c158510a8e4c960e40780e9d025cb12434
ea6e1414b6df3220f2a1feebb5cb9bf02f5a2eb88146557e
dc476dca486363aa76902a6b21111ce9ad7abd6359892fc4
8c288feb4f0ed799cdafdd3d024100fb5f6e7a3b78dd85ef
64ee22114e8dc685a9a892424c75e483badac6a1830e239d
d3bbdaf001bb9e5d67bfd491b1a3cdd39f1db581cd186ae4
2e1845a15ea213024100f80729d14b5d3c63debc5e7310c4
2c160312c3e196935c6a1760f73d165e1da00300635d17ad
be47b8bca304ae6ada4fa09733ebd8a8a1bf87afd2bd1184
5c2702404674b4217c83df0b485330cfc9ac0bdee5ddecca
b9dc767acb555736af546d651c3a997fbf1fad37b1148057
135308078bd6205c9c7a20eaecf3e287b9b82d9302402069
2f39be536cb4d04265453c07a68b29d3a5ed30b408fb841a
93b453dbf09b1c494cb2b98966b81097f5e582b54bd9e3ee
5e468f8559bda189844c58a1a739024100b42cb581a48d53
a62502be95772c3a90c0bc1f801c4097ae02ba2fc2fa41f2
078bd3e49c30958ff1d7795f453204ad338b26939cfed31b
c4746c92b03169e323
```

This will show the private key in hexadecimal format:

```
3082025d02010002818100f38b7bb23a8ffe8e1810bbc851eca047cb2515e0a590eea6c80de
dafc3c71812ed8135a37912f59c8c6f869d7cd82a5387d5fae0ed049e96ab2b88a3f1396241fa
4f58f3a47df7bba829331e0f564c9616ae3bdc16fa293c342254bc301197d8ef954fecbb1a720
4aab517f0e1b213034bd883cf0eedf9c80b2b742f6c7584e50203010001028181009ce63679
7b0afb56b5a04a29e2c58bf8d4a1feb72c52f3bcd4d34c5658cd71eff7bcaf3eb2a8893d22a85
2d430f2de49d41001c158510a8e4c960e40780e9d025cb12434ea6e1414b6df3220f2a1feeb
b5cb9bf02f5a2eb88146557edc476dca486363aa76902a6b21111ce9ad7abd6359892fc48c2
88feb4f0ed799cdafdd3d024100fb5f6e7a3b78dd85ef64ee22114e8dc685a9a892424c75e48
3badac6a1830e239dd3bbdaf001bb9e5d67bfd491b1a3cdd39f1db581cd186ae42e1845a15
ea213024100f80729d14b5d3c63debc5e7310c42c160312c3e196935c6a1760f73d165e1da0
0300635d17adbe47b8bca304ae6ada4fa09733ebd8a8a1bf87afd2bd11845c2702404674b4
217c83df0b485330cfc9ac0bdee5ddeccab9dc767acb555736af546d651c3a997fbf1fad37b11
48057135308078bd6205c9c7a20eaecf3e287b9b82d93024020692f39be536cb4d04265453c
07a68b29d3a5ed30b408fb841a93b453dbf09b1c494cb2b98966b81097f5e582b54bd9e3ee
5e468f8559bda189844c58a1a739024100b42cb581a48d53a62502be95772c3a90c0bc1f801
c4097ae02ba2fc2fa41f2078bd3e49c30958ff1d7795f453204ad338b26939cfed31bc4746c92
b03169e323
```

They are ordered in a predefined format. It will tell the values of n, e, d, p, q, and a few other values. (Please review the slides to check the meanings of n, e, d, p, q).

The above “predefined format” is called Abstract Syntax Notation One (ASN.1). It is a standard way to define data structure used in telecommunications and computer networking, and especially in cryptography. You can use the following website to see what it represents for

https://holtstrom.com/michael/tools/asn1decoder.php

(This website can translate directly from BASE64 to ASN.1, but you should know there are two stages: BASE64->HEX->ASN.1)

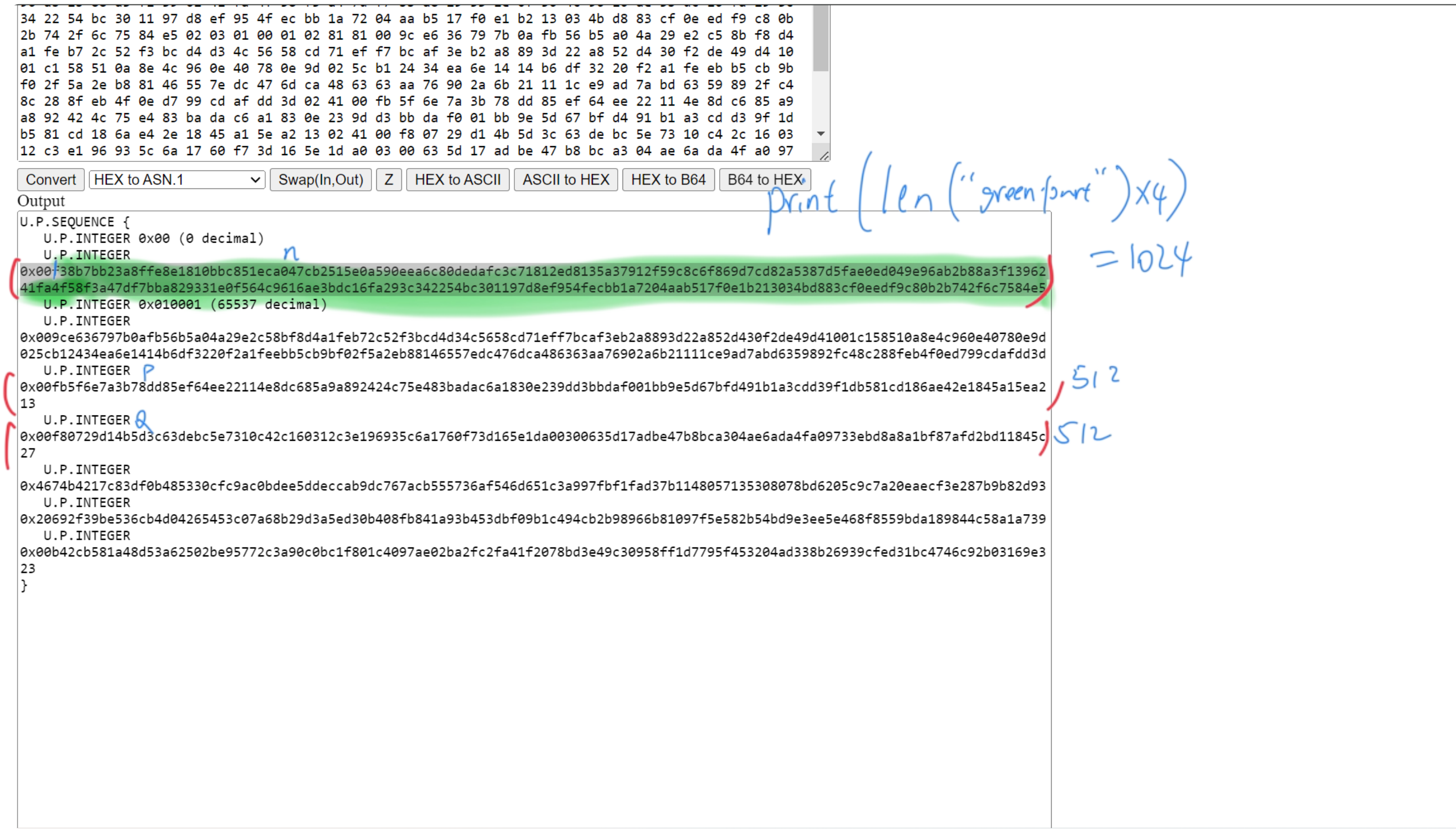
The translated numbers are actually in the following order.

```
RSAPrivateKey ::= SEQUENCE {
  version           Version,
  modulus           INTEGER,  -- n
  publicExponent    INTEGER,  -- e
  privateExponent   INTEGER,  -- d
  prime1            INTEGER,  -- p
  prime2            INTEGER,  -- q
  exponent1         INTEGER,  -- d mod (p-1)
  exponent2         INTEGER,  -- d mod (q-1)
  coefficient       INTEGER,  -- (inverse of q) mod p
  otherPrimeInfos   OtherPrimeInfos OPTIONAL
}
```

In the class, we know that the large number n can be factorized as p*q. This is a secret only known from the private key. The first “purple” part shows the value of n and the second and third purple parts show the values of p and q.

Question:
(1) How long is n, p, and q.
(2) Verify n is the product of p and q using Python.
(3) Re-do the experiments by generating your own keys.

(1)



```
34 22 54 bc 30 11 97 d8 ef 95 4f ec bb 1a 72 04 aa b5 17 f0 e1 b2 13 03 4b d8 83 cf 0e ed f9 c8 0b
2b 74 2f 6c 75 84 e5 02 03 01 00 01 02 81 81 00 9c e6 36 79 7b 0a fb 56 b5 a0 4a 29 e2 c5 8b f8 d4
a1 fe b7 2c 52 f3 bc d4 d3 4c 56 58 cd 71 ef f7 bc af 3e b2 a8 89 3d 22 a8 52 d4 30 f2 de 49 d4 10
01 c1 58 51 0a 8e 4c 96 0e 40 78 0e 9d 02 5c b1 24 34 ea 6e 14 14 b6 df 32 20 f2 a1 fe eb b5 cb 9b
f0 2f 5a 2e b8 81 46 55 7e dc 47 6d ca 48 63 63 aa 76 90 2a 6b 21 11 1c e9 ad 7a bd 63 59 89 2f c4
8c 28 8f eb 4f 0e d7 99 cd af dd 3d 02 41 00 fb 5f 6e 7a 3b 78 dd 85 ef 64 ee 22 11 4e 8d c6 85 a9
a8 92 42 4c 75 e4 83 ba da c6 a1 83 0e 23 9d d3 bb da f0 01 bb 9e 5d 67 bf d4 91 b1 a3 cd d3 9f 1d
b5 81 cd 18 6a e4 2e 18 45 a1 5e a2 13 02 41 00 f8 07 29 d1 4b 5d 3c 63 de bc 5e 73 10 c4 2c 16 03
12 c3 e1 96 93 5c 6a 17 60 f7 3d 16 5e 1d a0 03 00 63 5d 17 ad be 47 b8 bc a3 04 ae 6a da 4f a0 97
```

```
U.P.SEQUENCE {
   U.P.INTEGER 0x00 (0 decimal)
   U.P.INTEGER
0x00f38b7bb23a8ffe8e1810bbc851eca047cb2515e0a590eea6c80dedafc3c71812ed8135a37912f59c8c6f869d7cd82a5387d5fae0ed049e96ab2b88a3f13962
41fa4f58f3a47df7bba829331e0f564c9616ae3bdc16fa293c342254bc301197d8ef954fecbb1a7204aab517f0e1b213034bd883cf0eedf9c80b2b742f6c7584e5
   U.P.INTEGER 0x010001 (65537 decimal)
   U.P.INTEGER
0x009ce636797b0afb56b5a04a29e2c58bf8d4a1feb72c52f3bcd4d34c5658cd71eff7bcaf3eb2a8893d22a852d430f2de49d41001c158510a8e4c960e40780e9d
025cb12434ea6e1414b6df3220f2a1feebb5cb9bf02f5a2eb88146557edc476dca486363aa76902a6b21111ce9ad7abd6359892fc48c288feb4f0ed799cdafdd3d
   U.P.INTEGER
0x00fb5f6e7a3b78dd85ef64ee22114e8dc685a9a892424c75e483badac6a1830e239dd3bbdaf001bb9e5d67bfd491b1a3cdd39f1db581cd186ae42e1845a15ea2
13
   U.P.INTEGER
0x00f80729d14b5d3c63debc5e7310c42c160312c3e196935c6a1760f73d165e1da00300635d17adbe47b8bca304ae6ada4fa09733ebd8a8a1bf87afd2bd11845c
27
   U.P.INTEGER
0x4674b4217c83df0b485330cfc9ac0bdee5ddeccab9dc767acb555736af546d651c3a997fbf1fad37b1148057135308078bd6205c9c7a20eaecf3e287b9b82d93
   U.P.INTEGER
0x20692f39be536cb4d04265453c07a68b29d3a5ed30b408fb841a93b453dbf09b1c494cb2b98966b81097f5e582b54bd9e3ee5e468f8559bda189844c58a1a739
   U.P.INTEGER
0x00b42cb581a48d53a62502be95772c3a90c0bc1f801c4097ae02ba2fc2fa41f2078bd3e49c30958ff1d7795f453204ad338b26939cfed31bc4746c92b03169e3
23
}
```

print(len("green part")x4)

=1024

n

P

512

Q

512

(2) n = red part (直接复制就好，都复制!

p = ...,

q = ...

print(n == p×q)