In the attached archive we have 3 public RSA key of Bob':

bob.pub

```
-----BEGIN PUBLIC KEY-----
MDgwDQYJKoZIhvcNAQEBBQADJwAwJAIdDFtp4ZeeVB+F2s3iqhTSciqEb0Gz24Pm
Z+Oz0R0CAwEAAQ==
-----END PUBLIC KEY-----
```

bob2.pub

```
-----BEGIN PUBLIC KEY-----
MDgwDQYJKoZIhvcNAQEBBQADJwAwJAIdCiM3Dn0PsAIyFkrG1kKED8VOkgJDP5J6
YOta29kCAwEAAQ==
-----END PUBLIC KEY-----
```

bob3.pub

```
-----BEGIN PUBLIC KEY-----
MDgwDQYJKoZIhvcNAQEBBQADJwAwJAIdDFtp4ZeeVB+F2s3iqhTSciqEb0Gz24Pm
Z+Oz0R0CAwEAAQ==
-----END PUBLIC KEY-----
```

And file **secret.enc** with the encrypted message (3 parts):

```
DK9dt2MTybMqRz/N2RUMq2qauvqFIOnQ89mLjXY=

AK/WPYsK5ECFsupuW98bCFKYUApgrQ6LTcm3KxY=

CiLSeTUCCKkyNf8NVnifGKKS2FJ7VnWKnEdygXY=
```

For a start we learn what information we can receive from keys:

```
openssl rsa -pubin -in bob.pub -text -modulus
```

```
Public-Key: (228 bit)
Modulus:
    0d:56:4b:97:8f:9d:23:35:04:95:8e:ed:8b:74:43:
    73:28:1e:d1:41:8b:29:f1:ec:fa:80:93:d8:cf
Exponent: 65537 (0x10001)
Modulus=D564B978F9D233504958EED8B744373281ED1418B29F1ECFA8093D8CF
writing RSA key
-----BEGIN PUBLIC KEY-----
MDgwDQYJKoZIhvcNAQEBBQADJwAwJAIdDVZLl4+dIzUElY7ti3RDcyge0UGLKfHs
+oCT2M8CAwEAAQ==
-----END PUBLIC KEY-----
```

```
Public-Key: (228 bit)
Modulus:
    0a:23:37:0e:7d:0f:b0:02:32:16:4a:c6:d6:42:84:
    0f:c5:4e:92:02:43:3f:92:7a:60:eb:5a:db:d9
Exponent: 65537 (0x10001)
Modulus=A23370E7D0FB00232164AC6D642840FC54E9202433F927A60EB5ADBD9
writing RSA key
-----BEGIN PUBLIC KEY-----
MDgwDQYJKoZIhvcNAQEBBQADJwAwJAIdCiM3Dn0PsAIyFkrG1kKED8VOkgJDP5J6
YOta29kCAwEAAQ==
-----END PUBLIC KEY-----
```

```
Public-Key: (228 bit)
Modulus:
    0c:5b:69:e1:97:9e:54:1f:85:da:cd:e2:aa:14:d2:
    72:2a:84:6f:41:b3:db:83:e6:67:e3:b3:d1:1d
Exponent: 65537 (0x10001)
Modulus=C5B69E1979E541F85DACDE2AA14D2722A846F41B3DB83E667E3B3D11D
writing RSA key
-----BEGIN PUBLIC KEY-----
MDgwDQYJKoZIhvcNAQEBBQADJwAwJAIdDFtp4ZeeVB+F2s3iqhTSciqEb0Gz24Pm
Z+Oz0R0CAwEAAQ==
-----END PUBLIC KEY-----
```

Module for all keys small, so we can try to pick up these two «random» numbers **n (modulus)=p • q**. (Wiki_RSA)
Decoding module from **HEX to DEC** the following command:

```
python -c "print
int('D564B978F9D233504958EED8B744373281ED1418B29F1ECFA8093D8CF', 16)"
```

- **3595672605160272402368143140718423687035016566478191408433163038783 51**
- **27330804584972405981562438938898756274452743557857583103893926647292 1**
- **33314633555506058962332645774471621313964699173149327274769507495554 9**

There are two methods: brute force (yafu) or search in DB (factordb.com).
Download the **Yafu**, launch from the console and enter parameters. Depending on the PC, this

process only takes some minute.

factor(3595672605160272402368143140718423687035016566478191408433163038783511)

```
factor(3595672605160272402368143140718423687035016566478191408433163038783511)


fac: factoring 3595672605160272402368143140718423687035016566478191408433163038783511
fac: using pretesting plan: normal
fac: no tune info: using qs/gnfs crossover of 95 digits
div: primes less than 10000
fmt: 1000000 iterations
rho: x^2 + 3, starting 1000 iterations on C69
rho: x^2 + 2, starting 1000 iterations on C69
rho: x^2 + 1, starting 1000 iterations on C69
pm1: starting B1 = 150K, B2 = gmp-ecm default on C69
ecm: 30/30 curves on C69, B1=2K, B2=gmp-ecm default
ecm: 74/74 curves on C69, B1=11K, B2=gmp-ecm default
ecm: 44/44 curves on C69, B1=50K, B2=gmp-ecm default, ETA: 0 sec

starting SIQS on c69: 3595672605160272402368143140718423687035016566478191408433163038783511

==== sieving in progress (1 thread):   10848 relations needed ====
====           Press ctrl-c to abort and save state           ====
10546 rels found: 5396 full + 5150 from 50445 partial, (3790.21 rels/sec)

SIQS elapsed time = 15.3701 seconds.
Total factoring time = 23.1690 seconds


***factors found***

P35 = 17963604736595708916714953362445519
P35 = 20016431322579245244930631426505729

ans = 1
```

We get two 35-digit number (**p** and **q**).

```
***factors found***

P35 = 16549930833331357120312254608496323
P35 = 16514150337068782027309734859141427

ans = 1
```

```
***factors found***

P35 = 17357677172158834256725194757225793
P35 = 19193025210159847056853811703017693

ans = 1
```

The same information we can get out of the site **factordb**:

Search | Sequences | Report results | Factor tables | Status | Downloads | Login

3595672605160272402368143140718423687035016566478191408433163038 78351 Factorize! (?)

Result:		
status (?)	digits	number
FF	69 (show)	3595672605...51<69> = 179636047365957089167149533624455 19<35> · 20016431322579245244930631426505729<35>

Knowing all of these parameters, we can now create your own certificate.

For this we use **RSATool** (https://github.com/ius/rsatool):

```
python rsatool.py -p 17963604736595708916714953362445519 -q
20016431322579245244930631426505729 -o priv.key
```

```
Using (p, q) to initialise RSA instance

n =
d564b978f9d233504958eed8b744373281ed1418b29f1ecfa8093d8cf

e = 65537 (0x10001)

d =
10266f631885301d037017a38f3b3196b3491ce1c97007055fd220001

p = 17963604736595708916714953362445519 (0x375ac9161ad7e431ebddf01e514cf)

q = 20016431322579245244930631426505729 (0x3dae2e1d28965d328b06d615dfc01)

Saving PEM as priv.key
```

We repeat this procedure a couple of times and get 3 private key.
And now try to decrypt the message, do not forget to decoded **base64**:

```
echo "DK9dt2MTybMqRz/N2RUMq2qauvqFIOnQ89mLjXY=" | base64 -d | openssl
rsautl -inkey priv.key -decrypt
```

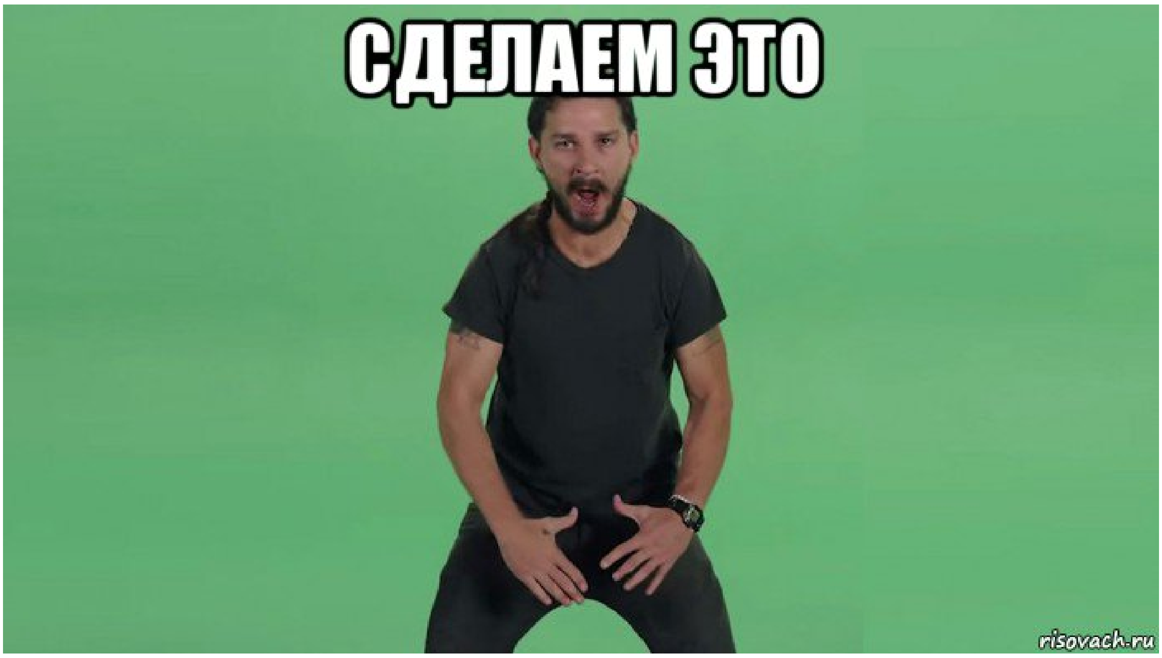
We get:

- IW{WEAK_R
- 3_SO_BAD!}
- SA_K3YS_4R

Flag: **IW{WEAK_RSA_K3YS_4R3_SO_BAD!}**
(crypto60, solved by 167)

Решение

По заданию мы смогли перехватить зашифрованное сообщение от Алисы для Боба и теперь настало время расшифровать его. **Сделаем это!**



В приложенном архиве у нас есть 3 публичных ключа **RSA** от Bob'a:
bob.pub

```
-----BEGIN PUBLIC KEY-----
MDgwDQYJKoZIhvcNAQEBBQADJwAwJAIdDFtp4ZeeVB+F2s3iqhTSciqEb0Gz24Pm
Z+Oz0R0CAwEAAQ==
-----END PUBLIC KEY-----
```