Lily Haas, Ashok Khare, & Emma Roskopf

SSH Key Gen

```
id_rsa_homework.pub contents
ssh-rsa
```

```
AAAAB3NzaC1yc2EAAAADAQABAAABgQC/Lz/nsCCTBLfElPib2xEnb4rFWMlktBc/1F4Uda
MxTZdprwTF+fQBWEQpSla5/Ou7sWbt6z4OuGy8gIPhDKamixJh39cUfJMCDBTVICf/KDPz
WImZgo/IbOgDSPACpowMHxMuIwT7b/AL7sspxf3Q7gpme1GYrNAAwm7lNkfvXmByJDf5pi
Z2JokHcPZOKp9Knz0MMVyKrAlo7j7fyiCGwyoCgOyXqdMVfKkSGNvJZ1Hs/jwj+5hF6pxz
CbymE7dIkZ5AZy2k65CVrIO7soXlYAc67bClTQAYMajUo6YbD0ysyURXgc0G1MUJDRnD5U
9EfZQ9BWkmfDdrajSFijCcW2MeYUEnjIsjRhRdzWDWzKsMPayPh27Jh3YLnfckQzFvtayy
qCtjk7h3TRNbx1jKL1YH60ch3lNlUE3bX7x4bdgMCv1gAd9ruZdVsRT/mRsSL+tzZ0rxlw
54cBj+54EDYmGLLAXmWVOUBqobDuSAMfh9/75lttAp/e4VichVO/U= kali@kali
```

PRIVATE KEY

```
Id_rsa_homework contents
-----BEGIN RSA PRIVATE KEY-----
```

```
MIIG5AIBAAKCAYEAvy8/57AgkwS3xJT4m9sRJ2+KxVjJZLQXP9ReFHWjMU2Xaa8E
xfn0AVhEKUpWufzru7Fm7es+DrhsvICD4QymposSYd/XFHyTAgwU1SAn/ygz81iJ
mYKPyGzoA0jwAqaMDB8TLiME+2/wC+7LKcX90O4KZntRmKzQAMJu5TZH715gciQ3
+aYmdiaJB3D2TiqfSp89DDFciqwJaO4+38oghsMqAoDsl6nTFXypEhjbyWdR7P48
I/uYReqccwm8phO3SJGeQGctpOuQlayDu7KF5WAHOu2wpU0AGDGo1KOmGw9MrMlE
V4HNBtTFCQ0Zw+VPRH2UPQVpJnw3a2o0hYownFtjHmFBJ4yLI0YUXc1g1syrDD2s
j4duyYd2C533JEMxb7WssqgrY5O4d00TW8dYyi9WB+tHId5TZVBN21+8eG3YDAr9
YAHfa7mXVbEU/5kbEi/rc2dK8ZcOeHAY/ueBA2JhiywF5llTlAaqGw7kgDH4ff++
ZbbQKf3uFYnIVTv1AgMBAAECggGAEXBj6eRc9Ewn3xyfmMfgxrIb/Ghe5nqu7DmY
41Dfs+2Z9B/s1Yo5Qa4leYa4Znq1ngdOlkExBfK4qHEolmyq/uIMqTTbz5iVaEhl
1hL60wF9y6K12nTYXLXia/pJOlSqzag//aNgHvdU9CGyEntm7ZnpBPQWu1B4ILmb
l/jdku4O6hp3qKQJOhnFwhIBxZuJsCraTLRr9pI+wX2J6qvrT5+Zw2SzMBpjkMAw
Q+BUnAh6pp2CKt4WQ7pkzvGCbbu6gYnJf76KIvtCqhSKBhLZ2Lf0UeCEPoRPxa4v
bg4BAy+mHeNuGzOAzOCL//ml7fvtZ3SV1pP/S6ueeJJ7wQ6m8qmeoQh6jnirTZvc
/QBGifVTdMJ7SdfKLs/sIsk52nAsF8RbgxK/koJoaQRjDksmCxgs11cPQ7vpdv6X
I0huq+Z9salKf67KHzT/l7ylWNGnsWEQMTntJ2wkahtPJ1qySdu8SxUyhy9uPqay
dZ0NQBDZKXTIZhWOvCiGfGP6EcC5AoHBAPtRmcYx/gRmYbqarAglTwvZNsChShIZ
08zYZIPCWRsp4YQ8xITXPr1cANkNTPf9aT/hv8rMD1MjkME475Oodwx/G+hA/i/x
ekIsaDhKiNadIQIZvqxMXU3Hyie5Zt6l17OVpWGgTeqlTmS50O435H3i2eZhaTdq
tXjGDqrQuqx3muXrdxLTM6mDCTi2RPC7+jYCWiGbdBV4B8xZN+WKWD3S5doXlRcM
qxrX8mmPHSpGC81vCGjtCLP4HPISs+3EAwKBwQDCvucI5QZzftw7+JJ42IBwciD+
VLQKp3hDjn3yQ5QJNBKea0Sy/lGh40Yth8/lQ3tT6ytqAkj3nyH9JNYjBqN+iQ8r
```

```
i0CzHvj4xk/eBu6EFqQSn0JoXcEe0RpYUJ+IE2997QN8nTTdT5WxEwrQTZO4YFub
E4/babdSiy7sDyMvWUO8vBTG6u6mPnxwOAESPyw09kH/Zf0ANXkANPkjOEKy+VY6
kNE7eU4Gt57BtRmYS0h9m4I4JMmHWuw/4VkwyqcCgcEAjoPbm9TFo2EbkmIbpBLk
HGf08I+GHXWf8c3UUhNnzEuXoxGCr4IQBfmiUXOFbiu5/4vY4vuMxSyp7aiH00f5
GpEKFk6k1jMtjVD5WNKLfohkK3jF85bqYi0lqhktCJ0rEJ+PxSjOhn1vUxc8L5pq
bCi0QDyFGfPdLLvDnYH4Gc1JfZBlWbuC1GGGXS6fMZSncvEqg1xQG8yRP4RoejZV
i9mIloBA/20jqnlC8jFB7LMEBl8sebPQ951/bJbGidJZAoHAUgzUkm7MFQUyDSJh
tACjvLyZIg7V/Fcq2GvErLnEOUxzO8O+Ihv8kNSM0MoT0WhpIaZHmFZ2H74wN9O5
VPCLGdjtR09YZx95jncfV1BcVdKkNiRbelVl9HHtQ6BUVqS5qLifa3VgvI23Z++5
8qF8x4XxkpjfV3ZnAqQZlOOb02ssmuwzZMu+2NQi4bT23pU3LulCTTTPL7NX07p9
7VTa5UUgC3TQctpdAX+Dp+GBcyTk2ecXKkh1kE3Ck/eB+IipAoHBAK0DKUfKGRPU
yunni0udqXVOmKhkuNleN9zZVs6CYlwu3VRPrgYDkbRh7vZzVsKez4f6lbtIfV27
PxsJyFcj3SxJIzL2jUYQOce8EZ2Hwow1t4G/vSiJ5otfI4ZuIpMSBIT2OGfnCRyY
ZjvLY0VJOtytalZArsLYciwHM551pbvhMHmTJbiaKZUSNuNsfnRnpYwUy3R7SdGY
kz4d5txPp9gA+hgKRdvDU5p54XFefms/HRWCTm1aTLxWSsC4JSGvKQ==
-----END RSA PRIVATE KEY-----
```

Private Key Expected Contents:

- Sequence
    - version
    - modulus n
    - publicExponent e
    - privateExponent d
    - prime1 p
    - prime2 q
    - exponent1 d%(p-1)
    - exponent2 d%(q-1)
    - coefficient (inverse of q) % p
    - otherPrimeInfos (optional)

Decode Private Key with lapo.it

```
SEQUENCE (9 elem)
  INTEGER 0
  INTEGER (3072 bit)
  INTEGER 65537
  INTEGER (3069 bit)
  INTEGER (1536 bit)
  INTEGER (1536 bit)
  INTEGER (1536 bit)
  INTEGER (1535 bit)
```

INTEGER (1536 bit)

Type	What this represents	What it tells us
SEQUENCE (9 elem)	A series of data structures	We have a sequence (series) that consists of 9 data structures
INTEGER 0	Version	Version 1
INTEGER (3072 bit)	Modulus (n)	n = 433869928707414464589566967288251703079421537899 687121910921275007618169129835233800374878210055 254857969458673643590095128685264778831504467632 048294418909623998571169160163278423576998089228 293714631817299414268336144510600413521708980399 685787146621085615532648264695869887211788892727 35911124137250690747721476696329284476479696781 1999643463148507928856441047663559436714102701148 900428014392112724577333058407607663913909886928 919882416861479470612640146131094905156476414806 515854402947222291280651637145648473365197572570 024706559514296584805371505049785221407645102585 251019503936656326412340317700039342264770460670 325827674633651437292669508461532094154027724571 296338914929536812828605830128180060411205364734 049304795842228215781274076580912418276016684329 292797889256061387432384490392455721143888377347 73880116129809111987990718803093390387285474651 1386805018492551423518863711111253208990531514715 4417322900469
INTEGER 65537	publicExponent e	e = 65537
INTEGER (3069 bit)	privateExponent d	d = 395757272046771086457486813624238015320930459673 822362144054104093190322269581309437209671168913 791223422101095723237497700425791972146227887682 436593685435972391696057073020748343095243080611 980991816684287638814122323555299948431081112170 120944743046042664396321364473794983864393243621 794218075426834657201090967988550685262364202447 492846578681016891023971891135199705918321855969 624297522015968058886323301823501016964058974939

		512497839082949215759397408115900131033032878115137876465373532050933554924633561582088474741320420273541759530787684867293130797739715122125687292642716082914721919012016246850048183363518267970721971885342070395560970196614933226866590400546091297365047400517636532530806172541875538488925833798324455995912476358598522907207020590011182728994839796193650278322559607331824696278269314034933488542830197983469308463719535037460302591098036155604170760476364766500386874248080775253993832633
INTEGER (1536 bit)	Prime1 p	p = 236623715878936668944004408564187472121864723262213856935230192052952332354909291321517941195402511353423836312006267677866408860778620070749789229609655823449301709908811163418713108830889143160680219672972477019512639035300391998211372197281093785091694839493654186616986236310774059458946816800858803912395375797450987787262711172387259598634201628620483724044679894897613703564755123042107407540510423051464491221135764602937301072407782999089939754541106893 1
INTEGER (1536 bit)	prime2 q	q = 183358598311166109606125506072811165394444506795617195798780208855707645748494360077062887857490815328878590343754491907618057638603489722051872505757905148167328334935158695229993044429464019349117532729530697813306949105793605712891188735993578036670759791699639072912263180472240346438268783649354795646691960229395978197973984284455698201703478652559037714998250203056279498326720628747560566810620386423477127294982650056365685354172659897823407874862748739 9
INTEGER (1536 bit)	Exponent1 d%(p-1)	= 134181970137858039956589099895928455894181616114819350582737916863083761533757250143779739179180294061959556474959197582742988218898891256989870865758476113076122629156828327163175823986315579236515551275254423241118264752855696297076879264228673412410512336770712180774732997943964587114642102927920359317641359020651975374610241512589836515611661646490557351120687300516882305862040670044965129527374297912394927319564361440156886

		29407758742966840900617614014 33
INTEGER (1535 bit)	Exponent2 d%(q-1)	= 772525079965198074132831449972147321188245070973 737279765005893910889957490795469802990747138415 916636555783232639429719570594857427034836092024 906386816142209174967458016371315526792924052138 832695014072630976093051479120065465453766804000 527133794429561830479032314761647762210583402635 683301665621651494035187123927209057853983865967 428894431611540726635242158122069485937639470438 073298692703476639167176564450443117770626725255 04667828632660938679535380 2921
INTEGER (1536 bit)	Coefficient (inverse of q) % p	162896021068822507199598088288367689005860043788 130614310759235303774690983007533185496263303107 704016674538173998356566499635544275408755034456 271462617872251036422755842492880005793516302098 797413540560046070909270260548981987422779619715 535311327806075623589060286849873471336035068446 976578123725511069709422445392770837000691761259 637080716251981438742755138152548785501935168649 760950370865150840670032557647266924634561819442 27303989897782744219918890 22761

Steps:

1. Copy pasted into Lapo Luchini's ASN.1 decoder
2. Matched up each integer number with the corresponding name
3. Used inspect and edit HTML to get the decimal numbers that are abbreviated on the Lapo Luchini ASN.1 decoder

Integer	Hex Offset	DER Encoding Meaning
Sequence	0	30 (sequence type) 82 06 E4 (length of everything in the sequence)
Version	4	02 (means integer type) 01 (how many bytes follow)
Modulus (n)	7	02 (integer) 82 01 81 (how many bytes follow
publicExponent e	396	02 (integer) 03 (how many bytes follow)
privateExponent	401	02 (integer) 82 01 80 (how many bytes follow

d		
Prime1 p	789	02 (integer) 81 C1 (length)
prime2 q	985	02(integer) 81 C1 (length
Exponent1 d%(p-1)	1181	02 (integer) 81 C1 (length)
Exponent2 d%(q-1)	1377	02(integer) 81 C0
Coefficient (inverse of q) % p	1572	02 (integer) 81 C1 (length)

PUBLIC KEY
Expected Contents

- Sequence (2 elements)
    - modulus n
    - publicExponent e

Decoded with RapidTables
The hex according to copy-pasting the middle part into
https://www.rapidtables.com/convert/number/ascii-hex-bin-dec-converter.html (base64 with hexdump without the ssh-rsa and the host name)

```
00 00 00 07 73 73 68 2D 72 73 61 00 00 00 03 01 00 01 00 00 01 81 00
BF 2F 3F E7 B0 20 93 04 B7 C4 94 F8 9B DB 11 27 6F 8A C5 58 C9 64 B4
17 3F D4 5E 14 75 A3 31 4D 97 69 AF 04 C5 F9 F4 01 58 44 29 4A 56 B9
FC EB BB B1 66 ED EB 3E 0E B8 6C BC 80 83 E1 0C A6 A6 8B 12 61 DF D7
14 7C 93 02 0C 14 D5 20 27 FF 28 33 F3 58 89 99 82 8F C8 6C E8 03 48
F0 02 A6 8C 0C 1F 13 2E 23 04 FB 6F F0 0B EE CB 29 C5 FD D0 EE 0A 66
7B 51 98 AC D0 00 C2 6E E5 36 47 EF 5E 60 72 24 37 F9 A6 26 76 26 89
07 70 F6 4E 2A 9F 4A 9F 3D 0C 31 5C 8A AC 09 68 EE 3E DF CA 20 86 C3
2A 02 80 EC 97 A9 D3 15 7C A9 12 18 DB C9 67 51 EC FE 3C 23 FB 98 45
EA 9C 73 09 BC A6 13 B7 48 91 9E 40 67 2D A4 EB 90 95 AC 83 BB B2 85
E5 60 07 3A ED B0 A5 4D 00 18 31 A8 D4 A3 A6 1B 0F 4C AC C9 44 57 81
CD 06 D4 C5 09 0D 19 C3 E5 4F 44 7D 94 3D 05 69 26 7C 37 6B 6A 34 85
8A 30 9C 5B 63 1E 61 41 27 8C 8B 23 46 14 5D CD 60 D6 CC AB 0C 3D AC
8F 87 6E C9 87 76 0B 9D F7 24 43 31 6F B5 AC B2 A8 2B 63 93 B8 77 4D
13 5B C7 58 CA 2F 56 07 EB 47 21 DE 53 65 50 4D DB 5F BC 78 6D D8 0C
```

```
0A FD 60 01 DF 6B B9 97 55 B1 14 FF 99 1B 12 2F EB 73 67 4A F1 97 0E
78 70 18 FE E7 81 03 62 61 8B 2C 05 E6 59 53 94 06 AA 1B 0E E4 80 31
F8 7D FF BE 65 B6 D0 29 FD EE 15 89 C8 55 3B F5
```

Type	Hex Offset	DER Encoding Meaning
modulus n	23	n = 433869928707414464589566967288251703079421537899687121910921275007618169129835233800374878210055254857969458673643590095128685264778831504467632048294418909623998571169160163278423576998089228293714631817299414268336144510600413521708980399685787146621085615532648264695869887211788892727359111241372506907477214766963292844764796967811999643463148507928856441047663559436714102701148900428014392112724577333058407607663913909886928919882416861479470612640146131094905156476414806515854402947222291280651637145648473365197572570024706559514296584805371505049785221407645102585251019503936656326412340317700039342264770460670325827674633651437292669508461532094154027724571296338914929536812828605830128180060411205364734049304795842228215781274076580912418276016684329292797889256061387432384490392455721143888377347738801161298091119879907188030933903872854746511386805018492551423518863711111253208990531514715441732290046 9
publicExponent e	16	65537

Note: the original public key began with “ssh-rsa” and ended with “kali@kali” which we removed to be able to decode the base64. The “ssh-rsa” is telling us this is an ssh-rsa key that we generated and the “kali@kali” is telling us the host name.

SANITY CHECK
Does the RSA stuff work?

RSA Checking (done in Python, here's the replit.com link to our file, bignumbergobrrrrrrrrrrrrr.py) the 'gn' is silent like gnocchi

$e*d \bmod \lambda(n) = 1$

- $\lambda(n)$ is the LCM of (p-1, q-1)
- We ran it in python and the math mathed

$n = p*q$

- The math checked out The math is kinda poggers rn

$e < (p-1)(q-1)$

- Indeed the 6 digit number is smaller than the two absolutely colossal numbers p and q

Greatest common divisor of $(e, \lambda(n)) = 1$

- YAYYYYYYYYY IT'S 1 I can finally go to bed (Lily), I would sell my firstborn child for this number to be 1(Emma), best news since LDC fixed the froyo machine(Ashok)

Sanity Check on Team Members

- Failed
- We rolled a nat 1
- Backslash!

- We take no responsibility for emotional damage dealt by the subtext