Solved Examples

1) A very simple example of RSA encryption

This is an extremely simple example using numbers you can work out on a pocket calculator (those of you over the age of 35 45 can probably even do it by hand).

1. Select primes p=11, q=3.
2. n = pq = 11.3 = 33
   phi = (p-1)(q-1) = 10.2 = 20
3. Choose e=3
   Check gcd(e, p-1) = gcd(3, 10) = 1 (i.e. 3 and 10 have no common factors except 1),
   and check gcd(e, q-1) = gcd(3, 2) = 1
   therefore gcd(e, phi) = gcd(e, (p-1)(q-1)) = gcd(3, 20) = 1
4. Compute d such that ed ≡ 1 (mod phi)
   i.e. compute d = e^{-1} mod phi = 3^{-1} mod 20
   i.e. find a value for d such that phi divides (ed-1)
   i.e. find d such that 20 divides 3d-1.
   Simple testing (d = 1, 2, ...) gives d = 7
   Check: ed-1 = 3.7 - 1 = 20, which is divisible by phi.
5. Public key = (n, e) = (33, 3)
   Private key = (n, d) = (33, 7).

This is actually the smallest possible value for the modulus n for which the RSA algorithm works.

Now say we want to encrypt the message m = 7,
c = m^e mod n = 7^3 mod 33 = 343 mod 33 = 13.
Hence the ciphertext c = 13.

To check decryption we compute
m' = c^d mod n = 13^7 mod 33 = 7.
Note that we don't have to calculate the full value of 13 to the power 7 here. We can make use of the fact that
a = bc mod n = (b mod n).(c mod n) mod n
so we can break down a potentially large number into its components and combine the results of easier, smaller calculations to calculate the final value.

One way of calculating m' is as follows:-
Note that any number can be expressed as a sum of powers of 2. So first compute values of 13^2, 13^4, 13^8, ... by repeatedly squaring successive values modulo 33.
13^2 = 169 ≡ 4, 13^4 = 4.4 = 16, 13^8 = 16.16 = 256 ≡ 25.

Then, since $7 = 4 + 2 + 1$, we have $m' = 13^7 = 13^{(4+2+1)} = 13^4.13^2.13^1 \equiv 16 \times 4 \times 13 = 832 \equiv 7 \bmod 33$

Now if we calculate the ciphertext c for all the possible values of m (0 to 32), we get

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
c	0	1	8	27	31	26	18	13	17	3	10	11	12	19	5	9	4

m	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
c	29	24	28	14	21	22	23	30	16	20	15	7	2	6	25	32

Note that all 33 values of m (0 to 32) map to a unique code c in the same range in a sort of random manner. In this case we have nine values of m that map to the same value of c - these are known as *unconcealed messages*. m = 0, 1 and n-1 will always do this for any *n*, no matter how large. But in practice, higher values shouldn't be a problem when we use large values for *n* in the order of several hundred bits.

If we wanted to use this system to keep secrets, we could let A=2, B=3, ..., Z=27. (We specifically avoid 0 and 1 here for the reason given above). Thus the plaintext message "HELLOWORLD" would be represented by the set of integers m_1, m_2, ...

(9,6,13,13,16,24,16,19,13,5)

Using our table above, we obtain ciphertext integers c_1, c_2, ...

(3,18,19,19,4,30,4,28,19,26)

Note that this example is no more secure than using a simple Caesar substitution cipher, but it serves to illustrate a simple example of the mechanics of RSA encryption.

Remember that calculating m^e mod n is easy, but calculating the inverse c^{-e} mod n is very difficult, well, for large n's anyway. However, if we can factor n into its prime factors p and q, the solution becomes easy again, even for large n's. Obviously, if we can get hold of the secret exponent d, the solution is easy, too.

2) A slightly less simple example of the RSA algorithm

This time, to make life slightly less easy for those who can crack simple Caesar substitution codes, we will group the characters into blocks of three and compute a message representative integer for each block.

ATTACKxATxSEVEN = ATT ACK XAT XSE VEN

In the same way that a decimal number can be represented as the sum of powers of ten, e.g. $135 = 1 \times 10^2 + 3 \times 10^1 + 5$, we could represent our blocks of three characters in base 26 using A=0, B=1, C=2, ..., Z=25

ATT = $0 \times 26^2 + 19 \times 26^1 + 19 = 513$
ACK = $0 \times 26^2 + 2 \times 26^1 + 10 = 62$
XAT = $23 \times 26^2 + 0 \times 26^1 + 19 = 15567$
XSE = $23 \times 26^2 + 18 \times 26^1 + 4 = 16020$
VEN = $21 \times 26^2 + 4 \times 26^1 + 13 = 14313$

For this example, to keep things simple, we'll not worry about numbers and punctuation characters, or what happens with groups AAA or AAB.

In this system of encoding, the maximum value of a group (ZZZ) would be $26^3-1 = 17575$, so we require a modulus n greater than this value.

1. We "generate" primes p=137 and q=131 (we cheat by looking for suitable primes around $\sqrt{n}$)
2. n = pq = 137.131 = 17947
   phi = (p-1)(q-1) = 136.130 = 17680
3. Select e = 3
   check gcd(e, p-1) = gcd(3, 136) = 1, OK and
   check gcd(e, q-1) = gcd(3, 130) = 1, OK.
4. Compute $d = e^{-1} \bmod phi = 3^{-1} \bmod 17680 = 11787$.
5. Hence public key, (n, e) = (17947, 3) and private key (n, d) = (17947, 11787).

Question: Why couldn't we use e=17 here?

To encrypt the first integer that represents "ATT", we have $c = m^e \bmod n = 513^3 \bmod 17947 = 8363$.
We can verify that our private key is valid by decrypting $m' = c^d \bmod n = 8363^{11787} \bmod 17947 = 513$.

Overall, our plaintext is represented by the sequence of integers m

(513, 62, 15567, 16020, 14313)

We compute corresponding ciphertext integers $c = m^e \bmod n$, (which is still possible by using a calculator) and send this to the person who has the private key.

(8363, 5017, 11884, 9546, 13366)

You are welcome to compute the inverse of these ciphertext integers using $m = c^d \bmod n$ to verify that the RSA algorithm still holds. However, this is now outside the realms of hand calculations unless you are very patient.

To help you carry out these modular arithmetic calculations, download our free modular arithmetic command line programs (*last updated 18 June 2009*).

Note that this is still a very insecure example. Starting with the knowledge that the modulus 17947 is probably derived from two prime numbers close to its square root, a little testing of suitable candidates from a table of prime numbers will get you the answer pretty quickly.

$\sqrt{17947} = 133.97$, so working downwards from a table of prime numbers we try: 131: 17947 / 131 = 137 exactly, so we have it.

You could also write a simple computer program to factor n that just divides by every odd number starting from 3 until it reaches a number greater than the square root of n.

3) A real example

In practice, we use a modulus of size in the order of 1024 bits. That is over 300 decimal digits. One example is

n =

119294134840169509055527211331255649644606569661527638012067481954943056851150
33

380631595703771562029730500011862877084668996911289221224545711806057499598951
70

800421052634273763222742663931161935178395707735056322315966811219273374739732
20

31251259906123132225094550626006655753823851757539062126294038391396

This is composed of the two primes

p =

1093376618363257581761151703473066828715579998463222345413874567112127345628767
70

00829084330287552127497024531459322294612906453835858101861553982847914 6469

q =

1091061696734911023172373407861492264533706088214174896820983422513897601117999
93

3942998101597369044685540217082898243965534121805148279964448454381760 99727

With a number this large, we can encode all the information we need in one big integer. We put our message into an octet string and then convert to a large integer.

Also, rather than trying to represent the plaintext as an integer directly, we generate a random *session key* and use that to encrypt the plaintext with a conventional, much faster symmetrical algorithm like Triple DES or AES-128. We then use the much slower public key encryption algorithm to encrypt just the session key.

The sender A then transmits a message to the recipient B in a format something like this:-

Session key encrypted with RSA = xxxx
Plaintext encrypted with session key = xxxxxxxxxxxxxxxxx

The recipient B would extract the encrypted session key and use his private key (n,d) to decrypt it. He would then use this session key with a conventional symmetrical decryption algorithm to decrypt the actual message. Typically the transmission would include in plaintext details of the encryption algorithms used, padding and encoding methods, initialisation vectors and other details required by the recipient. The only secret required to be kept, as always, should be the private key.

If Mallory intercepts the transmission, he can either try and crack the conventionally-encrypted plaintext directly, or he can try and decrypt the encryped session key and then use that in turn. Obviously, this system is as strong as its weakest link.

When signing, it is usual to use RSA to sign the message digest of the message rather than the message itself. A one-way hash function like SHA-1 or SHA-256 is used. The sender A then sends the signed message to B in a format like this

Hash algorithm = hh
Message content = xxxxxxxxx...xxx
Signature = digest signed with RSA = xxxx

The recipient will decrypt the signature to extract the signed message digest, m; independently compute the message digest, m', of the actual message content; and check that m and m' are equal. Putting the message digest algorithm at the beginning of the message enables the recipient to compute the message digest on the fly while reading the message.

4) A worked example of RSA public key encryption

Let's suppose that Alice and Bob want to communicate, using RSA technology (It's always Alice and Bob in the computer science literature.) The message that Alice wants to send Bob is the number 1275. [That's not very interesting. If she wanted

to send the message "Hi Bob", she would turn that into a number by writing it using the ASCI encoding. In hex, this is 4869 2042 6F62 - and so in decimal, "Hi Bob" becomes the number 79616349990754. But that's too big for this purpose, so I'll just use 1275 - even though that is 04FB in hexadecimal, which doesn't mean much at all when you convert it to text.] Alice has put up on the internet somewhere that her modulus is 186101 and that her public key is 907. Bob on the other hand, has disclosed to the world that his modulus is 189781 and that his public key is 5437.

The security of the system relies on the difficulty in factoring the two modulii. Alice and Bob both know how to do that for their two numbers (because they chose them by picking two primes and multiplying them together). In practice, one uses much larger numbers than the 6 digit numbers we've used here, so it might not take you too long to discover that $186101 = 149 \times 1249$ and that $189781 = 173 \times 1097$.

If you know that information, it is easy to compute Alice and Bob's private keys. For Alice, we are going to use the fact that elements of Z186101 which have inverses under multiplication form a group with $(148-1)\times(1249-1) = 184704$ elements, to tell us that $x^{184704} \equiv 1 \bmod 186101$ for almost all x. [It doesn't work for x which are divisible by 149 or 1249, but there are only 1397=149+1249-1 such numbers amongst the 186101 possibilities modulo 186101. The odds of getting such a nasty number go down even further as the size of the numbers increases.[1]] To find Alice's private key we have to solve

$$907x \equiv 1 \bmod 184704.$$

We can do this very quickly using Euclid's algorithm.

$$184704 = 203 \times 907 + 583$$

$$907 = 1 \times 583 + 324$$

$$583 = 1 \times 324 + 259$$

$$324 = 1 \times 259 + 65$$

$$259 = 3 \times 65 + 64$$

$$65 = 1 \times 64 + 1$$

$$64 = 64 \times 1$$

and writing this in reverse, we can compute that $907 \times 2851 - 14 \times 184704 = 1$, and so Alice's private key is 2851.

In a similar way, we can compute Bob's private key. This time we want to solve

$$5437x \equiv 1 \bmod 188512 .$$

Remember that the 188512 comes from $(173 - 1) \times (1097 - 1)$, and so you can't find it without knowing how to factor Bob's modulus. Bob can do that because he got it by multiplying 173 by 1097, but it is hard to do without that inside information. Bob finds his private key the same way as for Alice, and I'll leave it to you to check that it is 49269.

Now to send the message 1275, Alice first "decodes" it using her private key. That is, she computes $1275^{2851} \bmod 186101$. If you do that, you get 127296. Only Alice knows how to do this, because only she knows her private key, 2851. Then she takes this number, 127296 and encodes it with Bob's public key. That is, she computes $127296^{5437} \bmod 189781$. When she does this, she obtains the number 182522. That is the message she transmits to Bob.

To decode the message, Bob first uses his private key. So he computes $182522^{49269} \bmod 189781$. The answer he gets from this is 127296. He is the only one who can do this, because he is the only one who knows his private key. At this stage he has recovered the intermediate number in Alice's encoding of the message. Now he can complete the decoding, by using the publicly available details from Alice's public key. He computes 127296^{907} mod 186101, and obtains 1275, the original message. If 1275 seems to be a sensible message, he will know that it came from Alice, because she was the only one who knows how to transform the 1275 into the 127296 intermediate step.

Try duplicating this with smaller numbers, where you can do the computations with your calculator. Unfortunately, you'll need to keep the primes really small, (less than 15 will probably work) and that makes the examples uninspiring, but will help you see that you have at least got the right idea.

You might ask, how did Alice compute 1275^{2851} in a reasonable amount of time? Alice does this by first writing 2851 = 2048 + 512 + 256 + 32 + 2 + 1, that is, writing 2851 in binary as 101100100011. She then does the computations

$1275^1 \equiv 1275^1 \equiv 1275 \bmod 186101$

$1275^2 \equiv 1275^2 \equiv 136817 \bmod 186101$

$1275^4 \equiv 136817^2 \equiv 108505 \bmod 186101$

$1275^8 \equiv 108505^2 \equiv 27462 \bmod 186101$

$1275^{16} \equiv 27462^2 \equiv 80192 \bmod 186101$

$1275^{32} \equiv 80192^2 \equiv 36809 \bmod 186101$

$1275^{64} \equiv 36809^2 \equiv 87201 \bmod 186101$

$1275^{128} \equiv 87201^2 \equiv 113642 \bmod 186101$

$1275^{256} \equiv 113642^2 \equiv 25269 \bmod 186101$

$1275^{512} \equiv 25269^2 \equiv 9830 \bmod 186101$

$1275^{1024} \equiv 9830^2 \equiv 42481 \bmod 186101$

$1275^{2048} \equiv 42481^2 \equiv 13964 \bmod 186101$

So far, she has done just 11 multiplications. Then she uses this information to compute

$1275^3 \equiv 1275^1 \times 1275^2 \equiv 65038 \bmod 186101$

$1275^{35} \equiv 1275^3 \times 1275^{32} \equiv 166579 \bmod 186101$

$1275^{291} \equiv 1275^{35} \times 1275^{256} \equiv 52333 \bmod 186101$

$1275^{803} \equiv 1275^{291} \times 1275^{512} \equiv 50226 \bmod 186101$

$1275^{2851} \equiv 1275^{803} \times 1275^{2048} \equiv 127296 \bmod 186101$

which is an additional 5 multiplications, so it only takes her a total of 16 multiplications to compute 12752851 mod 186101. And you could do it yourself in just this way on any calculator which will handle 11 digit numbers, because, in the intermediate computations, the worst you ever might have to do is 186100 × 186100 = 34 633 210 000.

I should also say a little bit about security. Suppose that you were able to discover Alice's private key of 2851. You know that that has been chosen so that $907 \times 2851 \equiv 1 \bmod |G|$, where $|G|$ is the number of elements in the group of integers with inverses modulo 186101. So the number of elements in this group is a divisor of $907\times2851-1 = 2585856$. We know that this number is about 186101, because most elements have inverses, so if we calculate 2585856/186101 = 13.8949, it's easy to guess that the order of the group is actually 2585856/14 = 184704. Now, the way things work, if $186101 = pq$, then $184704 = (p-1)(q-1) = pq-p-q+1$, so $p + q = 186101 - 184704 + 1 = 1398$. So $(x - p)(x - q) = x^2 - 1398x + 186101$, and so p and q are the solutions of the quadratic equation $x2 - 1398x + 186101 = 0$. There's a formula for this, and you quickly get $x = 149$ or 1249. So, you see that any method to hack RSA encryption provides a way of factoring the modulus. Mathematicians haven't come up with any really good ways of factoring very large numbers, despite much trying, and believe that this is a very hard problem. The security of RSA depends on that belief being correct.

---

[1]Even the nasty numbers still work for encoding and decoding — its just that they can betray the factors of the modulus and so give anyone who stumbles upon such a thing a way of cracking the code. If you want to see what happens in this case, try sending the message 1249. The main thing to understand is that, while $1249^{184704} \neq 1 \bmod 186101$, it still happens to be the case that $1249^k \equiv 1249 \bmod 186101$ whenever $k \equiv 1 \bmod 184704$.

5) An Example of the RSA Algorithm

P = 61 <= first prime number (destroy this after computing E and D)

Q = 53 <= second prime number (destroy this after computing E and D)

PQ = 3233 <= modulus (give this to others)

E = 17 <= public exponent (give this to others)

D = 2753 <= private exponent (keep this secret!)

Your public key is (E,PQ).

Your private key is D.

The encryption function is: encrypt(T) = (T^E) mod PQ

= (T^17) mod 3233

The decryption function is: decrypt(C) = (C^D) mod PQ

= (C^2753) mod 3233

To encrypt the plaintext value 123, do this:

encrypt(123) = (123^17) mod 3233

= 337587917446653715596592958817679803 mod 3233

= 855

To decrypt the ciphertext value 855, do this:

decrypt(855) = (855^2753) mod 3233

= 50432888958416068734422899127394466631453878360035509315554967564501

055628612082559978744245428110054383498654289336384930246451441507 85

1720917966547826353070996380353873265008966860747718297458229503429 5

04079035818459409563779385865989368838083602840132509768620766977396
67533250542826093475735137988063256482639334453092594385562429233017
51977190016924916912809150596019178760171349725439279215696701789902
13430714646897127961027718137839458696772898693423652403116932170892
69617643726521315665833158712459759803042503144006837883246101784830
71758547454725206968892599589254436670143220546954317400228550092386
3694244485597333063051607385302863219302913503745471946757776713579
54965202919790505781532871558392070303159585937493663283548602090830
63550704455658896319318011934122017826923344101330116480696334024075
04695258866987658669006224024102088466507530263953870526631933584734
81094876156227126037327597360375237388364148088948438096157757045380
08107946980066734877795883758289985132793070353355127509043994817897
90548993381217329458535447413268056981087263348285463816885048824346
58897839333466254454006619645218766694795528023088412465948239275105
77049113329025684306505229256142730389832089007051511055250618994171
23177795157979429711795475296301837843862913977877661298207389072796
76720235011399271581964273076407418989190486860748124549315795374377
12441601438765069145868196402276027766869530903951314968319097324505
45234594477256587887692693353918692354818518542420923064996406822184
49011913571088542442852112077371223831105455431265307394075927890822
60604317113339575226603445164525976316184277459043201913452893299321
61307440532227470572894812143586831978415597276496357090901215131304
15756920979851832104115596935784883366531595132734467524394087576977
78908490126915322842080949630792972471304422194243906590308142893930

2915848308736874507897708692184529674114632115566786552833816480679 5

4559418910069509196589908545679807239237084630255354568691923554629 9

5715735879062274586195721721110788286575638597094190776320509783239 5

7134641190250047020848560408217509491077165531176529747380317676582 0

5876731402889103288343185088447211644271939037404131556498699591373 6

5162108451137402243351859957665775396936281254253900685526245456141 9

2588094374021288866697441097218453422181719808991195370754554203391 1

9645393664617929681653426522346399367423309701835339046236776936703 8

0534264482173582384219251590438148524738896864244370318665419961537 7

9139696490030395876065491524494504360013593927713395210125192857209 2

5978875116019596296156902711643189463734265002363100455571800369358 6

0552649100009072451837866895644171649072783562810097085452413546966 0

8448116133878065485451517616730860510806578293652410872326366722805 4

0038794108643482267500907782651210137281958316531396983090887317417 4

7453598868429855980718519221597004650810606844559536480892249440542 7

6632967459230889848486843586547985051154284401646235269693179937784 4

3021785701919709875162965466513027800996658005217820813931723237901 3

2324946826092008199810376848471678749891936949979148247163450609371 2

5654122501953795166897601855087599313367797793952782227323337529580 2

6312266535894820556651528946636903208328768043239061154935095459093 4

0667640225867084833760536998679410262047090571567447056531112428629 0

7354888492989983560999636092141128497745861469604028702967070147817 9

4902482829074841600836804586668550760461922520943498047157452688181 3

1850859150194852763596503458153641656549316013061330407434457965108 3

8030406224027889804282518909471629226689801668448096364519809051090579651307570379245958074479752371266761011473878742144149154813591743927994969564156538668838917154463056118053697283434702192063489995319176401611039249043917980339897549176539592360851180765318470647331801578207412764787592739087492955716853665185912666373831235945891267870958380002245150942445756487448408687753084539552173063669389170239403718478036277464317147085583049195989514677629439214310024561306111429937000557751339717282549110056008940898419671319709118165542908761090083249978313382407869615784923419862991680086774959340775930660220781494380785499679894539936406368572269742236185841142504837245124465580270859179795591086523099756519838277952945756996574245578688383544423685722368139902126136374408213147848320356361561134628701985142390184290974163862023205103971218498335528630868518428263461502744187358639504042281512399505995983653792227285847422071677836679451343638070865797742198535953931662799887897216959634553463364979492211301766131620747726611310701232140371388227022172323308547267953301507998062253835458948024820043144726191596190526034069061930939290724102849487001671729695177034679099794409750637649296356755580071162182772760318292179035029048609097626628539662702439253689025633710147168327404504583060228676314215815990079164262770005461232291921929971699076901690259464681041412142044724026616582756805241668614733933226595912700645630447416085291672187007045144649793226668732146346749041185886760836840306190695786990096521390675205019744076776510438851519416193184799191349243881528220384647292694460849152999588185988555

1951490663073117772381322675169458825936387861072430256598091490103278384821401136556784934102431512482864529170314100400120163648299853
2516634905605379458508942440385525245547779224010461489075274516342513992163738356814149047932037426337301987825405699619163520193896982
5447863130977374915447842763453259399874170013816319811664537720894400285485000269685982644562183794116702151847721909339232185087775790
9593326763114131296193984959261389879016697108810276638623167694057295932538078643444100512138025081797622723797210352196773268441946486
1640296105989902771053257045701633261343107641770004323715247462639399011899727845362949303636914900881060531231630009010150839331880116
6821516389310466665951378274989237455605110040164777168227162672707837012242465512648784549235041852167426383189733332434674449039780017
8468972640546214802412412583384350170488532060147568786231809409001263241969092252022679880113408073012216264404133887392600523096072386
1585549651580010347461197921307672245438036718832537086067133113258199227975522771848648475326124302804177943090938992370938053652046462
5514726788496152777327411926570911661358008414542148768731039444105479639308530896880365608504772144592172500126500717068969428154627563
7045883890421917739819064873190801482873905815946222786727741861011102763247972904122211994117388204526335701759090678628159281519982214
5765279685389251721872009007038913856284000733225850759048534804656454349837073287625935891427854318266587294608072389652291599021738887
95773647738726574610400822551124182720096168188828493894678810468847
31265541726209789056784581096517975300873063154649030211213352818084

7612299040957642785731636412488093094977073956758842296317115846456984202455109029882398517953684125891446352791897307683834073696131409745229856386682726910433575176771288945278813686239650666540898943949516191200216077789887686473648183782532484669916830728122031079193564666840159148582699993374427677252275403853322196852298590851548110402296579163382573855133148234595916332814458198436145963060249936175309792556123803901469066516367371885958277252568311998998464602721646279764077057074816406450769779869955106180046471937808223250148934078511378332510737538234034662695532926088138438957840998041704104177760846306286261061405961520706669524301843857503176293954302631267377406936404705896083462601885911184367532529845888040849710922999195655397019111919191883273086037667753396077224556321135065721910675875118681278634419757239219526333385653838824005719010256494923394451965959203992392217400247234147190970964562108299547746193228981181286055565880938518988118129056142740858091687657119112247632886587127553892843812661199193792462411263299073986785455875665245305619750989114578114735771283607554001774268660965093305172102723066635739462334136380459142377599652203094185588800394967558297112583616218901403595423493042474905369399277611426179640710012764328042870608353159458230594632682786127020335698034614324569702148437 5 mod 3233

= 123