

Certificates and RSA

CSE 468 Fall 2026

jedimaestro@asu.edu

TLS key exchange works just like we've learned,
with AES and Diffie-Hellman (basically)...

pop-ssl.pcapng

View Go Capture Analyze Statistics Telephony Wireless Tools Help



display filter ... <Ctrl-/>

Time	Source	Destination	Protocol	Length	Info
0.000000000	127.0.0.1	127.0.0.1	TCP	74	47261 → 110 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM TSval=689304940 TSecr=689304940
0.1537912832.7...	127.0.0.1	127.0.0.1	TCP	74	110 → 47261 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM TSval=689304940 TSecr=689304940
0.000082199	127.0.0.1	127.0.0.1	TCP	66	47261 → 110 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=689304940 TSecr=689304940
0.084187035	127.0.0.1	127.0.0.1	POP	95	S: +OK Dovecot (Ubuntu) ready.
0.084272014	127.0.0.1	127.0.0.1	TCP	66	47261 → 110 [ACK] Seq=1 Ack=30 Win=43776 Len=0 TSval=689304965 TSecr=689304965
0.084319002	127.0.0.1	127.0.0.1	POP	72	C: STLS
0.084334366	127.0.0.1	127.0.0.1	TCP	66	110 → 47261 [ACK] Seq=30 Ack=7 Win=43776 Len=0 TSval=689304965 TSecr=689304965
0.126639373	127.0.0.1	127.0.0.1	POP	98	S: +OK Begin TLS negotiation now.
0.126884360	127.0.0.1	127.0.0.1	TLSv1.2	428	Client Hello
0.126901694	127.0.0.1	127.0.0.1	TCP	66	110 → 47261 [ACK] Seq=62 Ack=369 Win=44800 Len=0 TSval=689304978 TSecr=689304978
0.175311356	127.0.0.1	127.0.0.1	TLSv1.2	1454	Server Hello, Certificate, Server Key Exchange, Server Hello Done
0.175360227	127.0.0.1	127.0.0.1	TCP	66	47261 → 110 [ACK] Seq=369 Ack=1450 Win=174720 Len=0 TSval=689304993 TSecr=689304993
0.182076773	127.0.0.1	127.0.0.1	TLSv1.2	224	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
0.182101908	127.0.0.1	127.0.0.1	TCP	66	110 → 47261 [ACK] Seq=1450 Ack=527 Win=45952 Len=0 TSval=689304995 TSecr=689304995
0.226073118	127.0.0.1	127.0.0.1	TLSv1.2	292	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
0.265203981	127.0.0.1	127.0.0.1	TCP	66	47261 → 110 [ACK] Seq=527 Ack=1676 Win=177536 Len=0 TSval=689305020 TSecr=689305020
4.782873574	127.0.0.1	127.0.0.1	TLSv1.2	100	Application Data
4.782908514	127.0.0.1	127.0.0.1	TCP	66	110 → 47261 [ACK] Seq=1676 Ack=561 Win=45952 Len=0 TSval=689306375 TSecr=689306375

Version: TLS 1.2 (0x0303)

Length: 66

Handshake Protocol: Server Hello

Handshake Type: Server Hello (2)

Length: 62

Version: TLS 1.2 (0x0303)

Random: 7b1ed73545bda76bd5f7be1824f1946f5fde6d0682a8422d62a3128d4028b627

Session ID Length: 0

Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)

Compression Method: null (0)

Extensions Length: 22

0060	06 82 a8 42 2d 62 a3 12 8d 40 28 b6 27 00 c0 30	...B-b... .@('..0
0070	00 00 16 ff 01 00 01 00 00 0b 00 04 03 00 01 02	
0080	00 23 00 00 00 0f 00 01 01 16 03 03 03 a5 0b 00	..#.....
0090	03 a1 00 03 9e 00 03 9b 30 82 03 97 30 82 02 7f	0...0...
00a0	a0 03 02 01 02 02 09 00 f5 1c 89 6d c1 6c a1 01	m.l...
00b0	30 0d 06 09 2a 86 48 86 f7 0d 01 01 0b 05 00 30	0...*.H.....0
00c0	62 31 1c 30 1a 06 03 55 04 0a 0c 13 44 6f 76 65	b1.0...U ...Dove
00d0	63 6f 74 20 6d 61 69 6c 20 73 65 72 76 65 72 31	cot mail server1
00e0	12 30 10 06 03 55 04 0b 0c 09 6c 6f 63 61 6c 68	.0...U... localh
00f0	6f 73 74 31 12 30 10 06 03 55 04 03 0c 09 6c 6f	ost1.0... .U....lo
0100	63 61 6c 68 6f 73 74 31 1a 30 18 06 09 2a 86 48	calhost1 .0...*.H

Session ID Length: 0

Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)

Compression Method: null (0)

But there is also a certificate for authentication
and non-repudiation...

```
TCP payload (1388 bytes)
▼ Transport Layer Security
  [Stream index: 0]
  ▶ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Certificate
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 933
    ▶ Handshake Protocol: Certificate
  ▶ TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
  ▶ TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
```

How do you know who you're talking to when you do encryption?

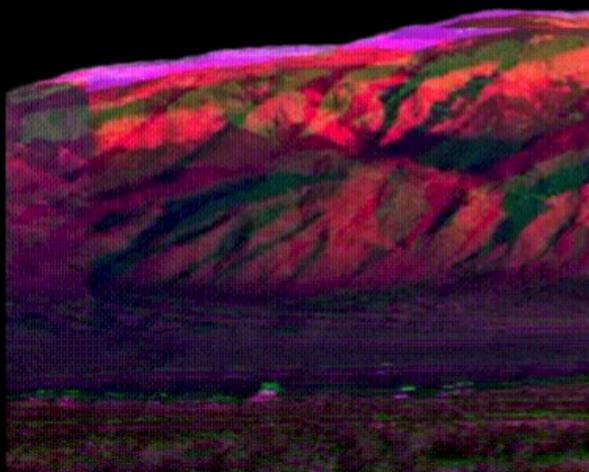
Authentication and non-repudiation

What if your private key gets stolen?

Forward secrecy and future secrecy

BREAKPOINT

BAD



We are a non-profit founded in 20
combined experience focusing on

goal is to provide technical expertise and capabilities to at risk populations subjected to

Certificate Viewer: breakpointingbad.com

General Details

Issued To

Common Name (CN)	breakpointingbad.com
Organization (O)	<Not Part Of Certificate>
Organizational Unit (OU)	<Not Part Of Certificate>

Issued By

Common Name (CN)	R3
Organization (O)	Let's Encrypt
Organizational Unit (OU)	<Not Part Of Certificate>

Validity Period

Issued On	Thursday, December 22, 2022 at 10:57:13 PM
Expires On	Wednesday, March 22, 2023 at 10:57:12 PM

Fingerprints

SHA-256 Fingerprint	81 05 41 B0 19 8B 06 9C 90 20 7F B3 EE 60 2E AB BD 64 25 F9 D8 DE 87 7D FD 70 34 AC F9 F5 DE 92
SHA-1 Fingerprint	C1 95 59 2C 66 12 BC 36 71 7E 99 C9 60 98 12 0A B8 02 1D 47

General

Details

Issued To

Common Name (CN)	breakpointingbad.com
Organization (O)	<Not Part Of Certificate>
Organizational Unit (OU)	<Not Part Of Certificate>

Issued By

Common Name (CN)	R3
Organization (O)	Let's Encrypt
Organizational Unit (OU)	<Not Part Of Certificate>

Validity Period

Issued On	Thursday, December 22, 2022 at 10:57:13 PM
Expires On	Wednesday, March 22, 2023 at 10:57:12 PM

Fingerprints

SHA-256 Fingerprint	81 05 41 B0 19 8B 06 9C 90 20 7F B3 EE 60 2E AB BD 64 25 F9 D8 DE 87 7D FD 70 34 AC F9 F5 DE 92
SHA-1 Fingerprint	C1 95 59 2C 66 12 BC 36 71 7E 99 C9 60 98 12 0A B8 02 1D 47

Certificate Fields

▼ breakpointingbad.com

▼ Certificate

Version

Serial Number

Certificate Signature Algorithm

Issuer

▼ Validity

Not Before

Field Value

04:B5:2A:1D:FD:B3:AC:F1:34:37:27:94:9F:F5:A8:5A:12:E6

Export...

breakpointingbad.com

Certificate Fields

▼ breakpointingbad.com

▼ Certificate

Version

Serial Number

Certificate Signature Algorithm

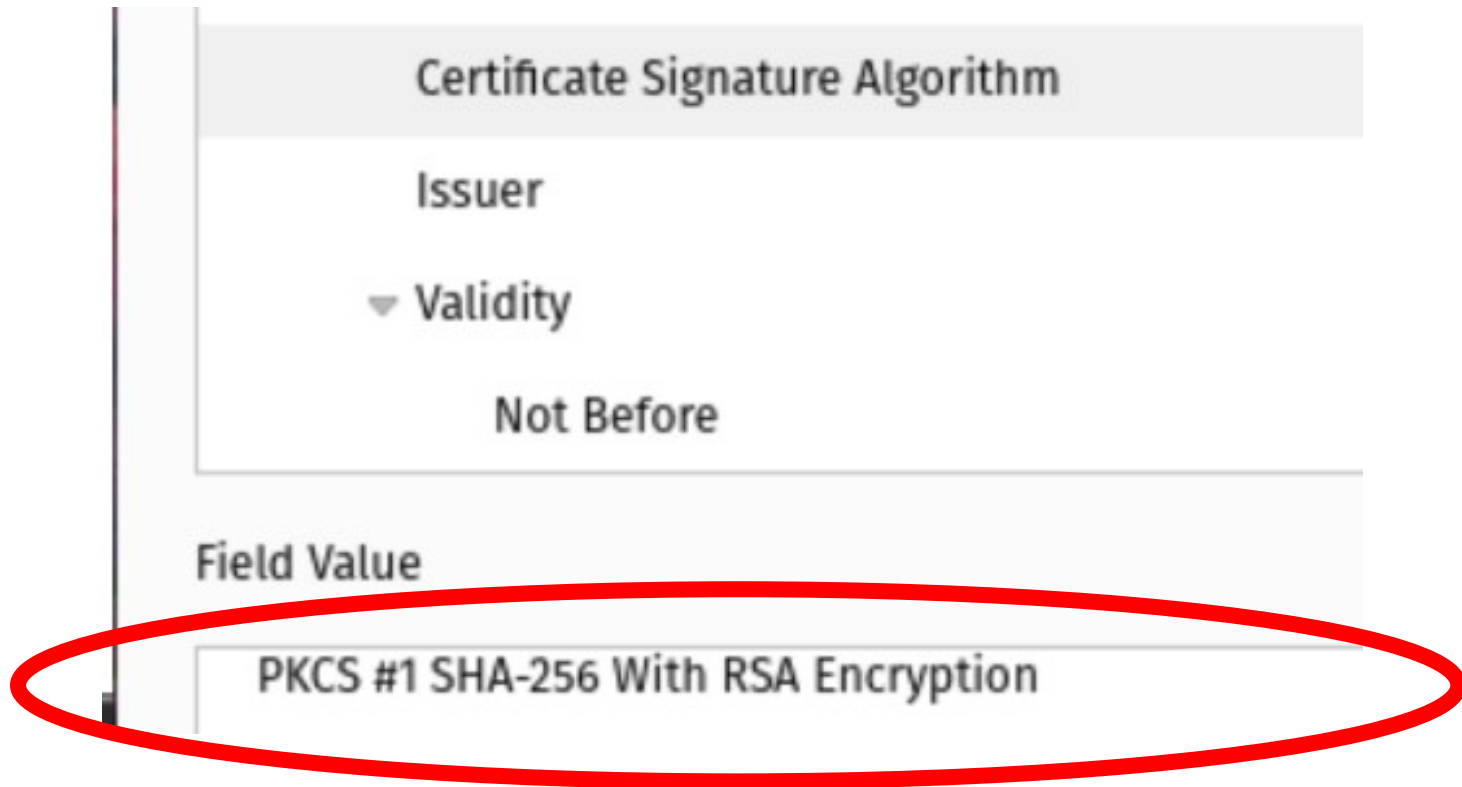
Issuer

▼ Validity

Not Before

Field Value

PKCS #1 SHA-256 With RSA Encryption



We need (1) a signature algorithm (like RSA), (2) a secure hash function (like SHA-256), and (3) some notion of semantic security (like PKCS #1).

breakpointingbad.com

Certificate Fields

Not After

Subject

▼ Subject Public Key Info

Subject Public Key Algorithm

Subject's Public Key

▼ Extensions

Certificate Key Usage

Extended Key Usage

Field Value

Modulus (2048 bits):

DC DA F0 96 47 5C 62 91 27 27 AD B2 95 EE 3D 51
CF 26 EB EC 27 EE ED 2E 9F DA 1D BF 83 2F 12 F0
EA CC 96 5B 8C C1 3E A1 C6 46 90 4D E5 93 20 E1
5C 9B 62 BB 82 3A 7F 77 7C 85 CB 8C F3 0F B9 0D
38 24 9C 0D 39 8C FF F4 B5 AD 0A 94 75 AA F9 41

Export

Certificate Fields

▼ breakpointingbad.com

▼ Certificate

Version

Serial Number

Certificate Signature Algorithm

Issuer

▼ Validity

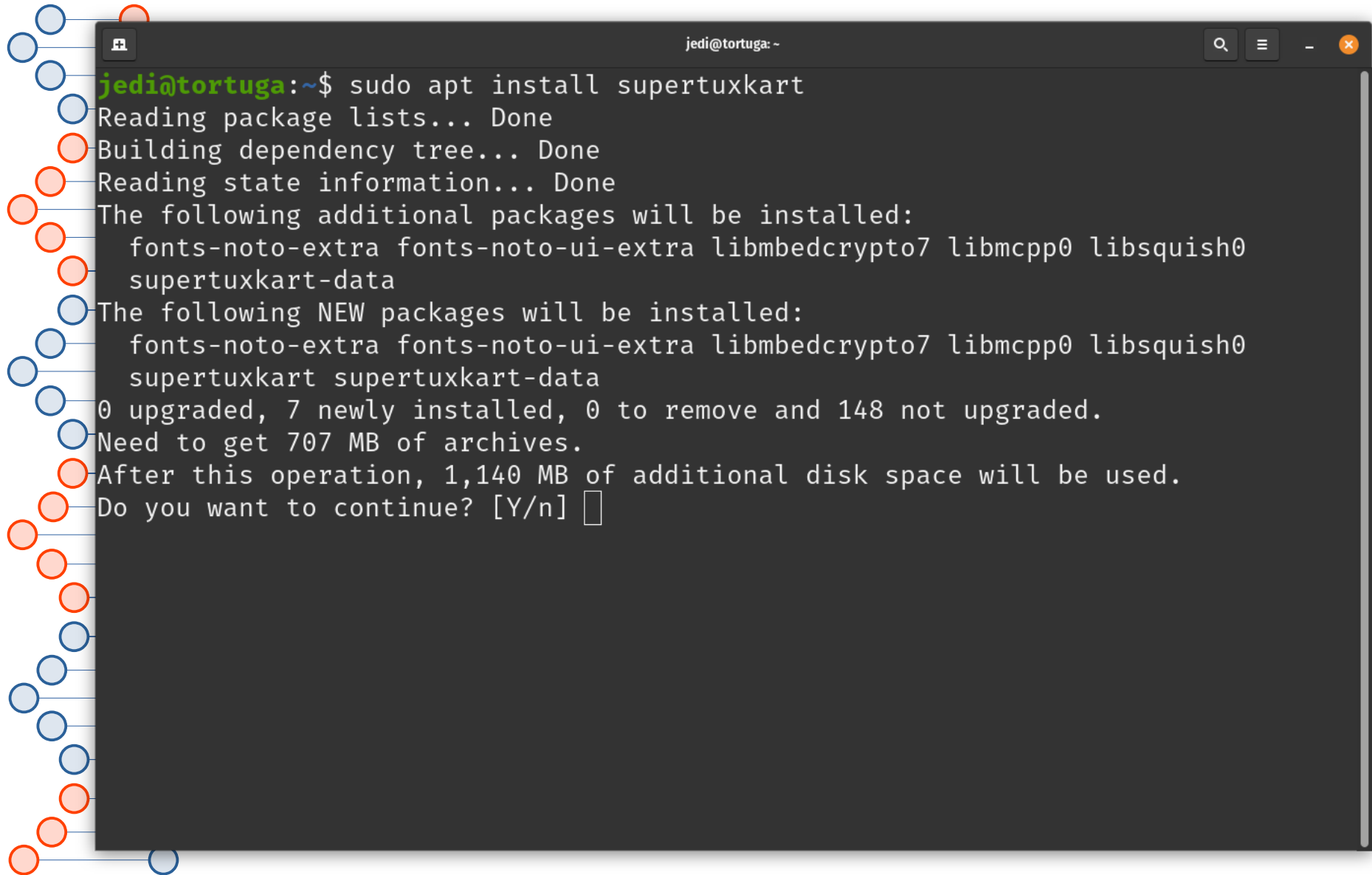
Not Before

Field Value

98 2F 52 F3 68 5E 6D BC 18 2C 93 42 8B C5 41 D1
40 B4 0F 53 D9 BD BA 22 F9 52 90 76 37 F0 C4 56
31 F8 8D C7 B8 21 3E FB 0F 83 B8 A7 CF F3 B4 A1

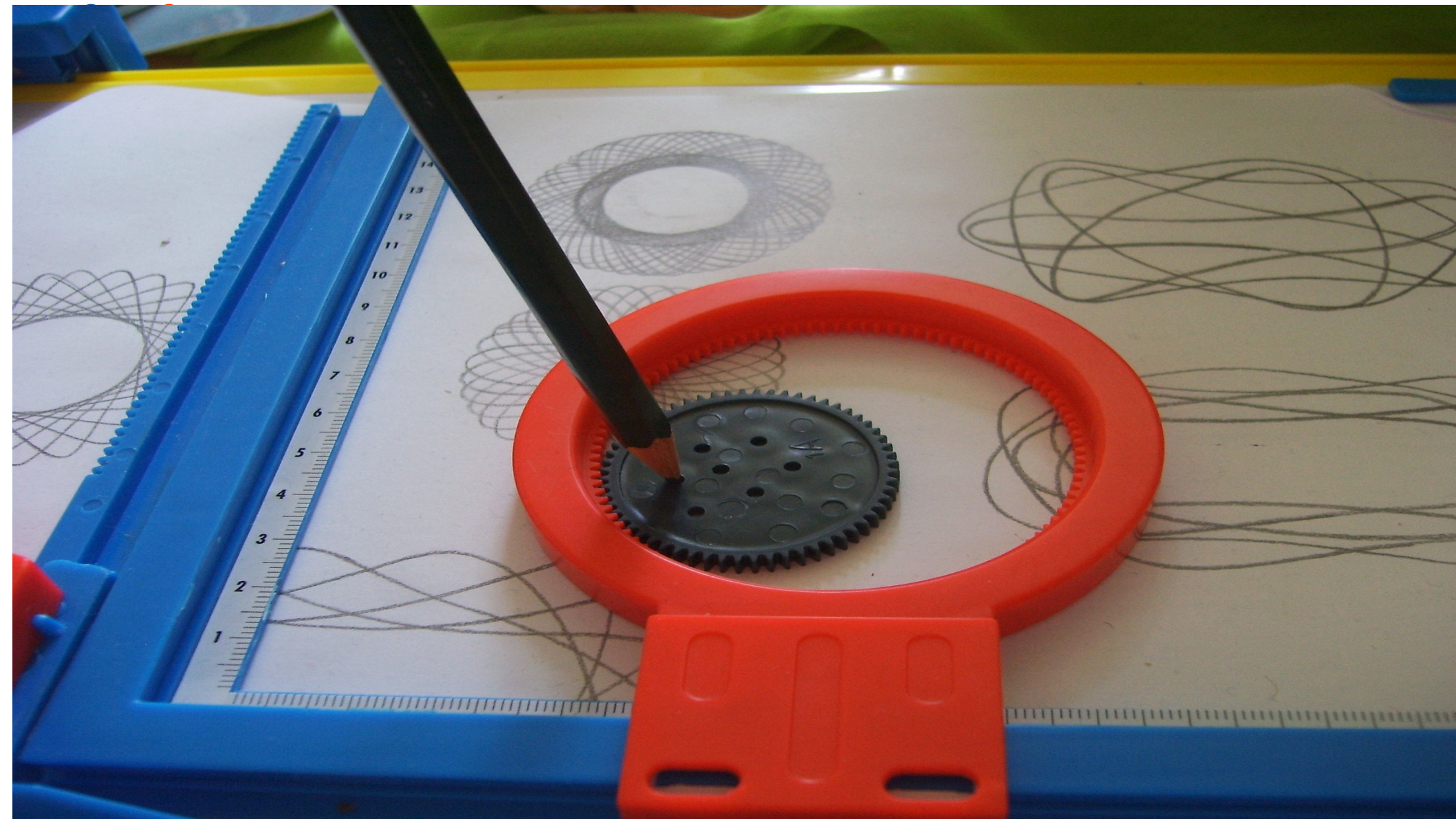
Public Exponent (17 bits):
01 00 01

Export...



A terminal window titled 'jedi@tortuga: ~' with standard window controls (search, menu, close). The terminal shows the command 'sudo apt install supertuxkart' and its output. The output indicates that several additional packages will be installed along with the requested package. It also shows the disk space requirements and asks for confirmation to continue.

```
jedi@tortuga:~$ sudo apt install supertuxkart
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  fonts-noto-extra fonts-noto-ui-extra libmbcrypto7 libmcpp0 libsquish0
  supertuxkart-data
The following NEW packages will be installed:
  fonts-noto-extra fonts-noto-ui-extra libmbcrypto7 libmcpp0 libsquish0
  supertuxkart supertuxkart-data
0 upgraded, 7 newly installed, 0 to remove and 148 not upgraded.
Need to get 707 MB of archives.
After this operation, 1,140 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```



Field



- Commutative

$$a + b = b + a$$

$$a * b = b * a$$

- Associative

$$(a + b) + c = a + (b + c)$$

$$(a * b) * c = a * (b * c)$$

- Identity

$$0 \neq 1, a + 0 = a, a * 1 = a$$

- Inverse

$$a + -a = 0$$

$$a * a^{-1} = 1$$

- Distributive

$$a * (b + c) = (a * b) + (a * c)$$

Field Ring



- Commutative

$$a + b = b + a$$

$$~~a * b = b * a~~$$

- Associative

$$(a + b) + c = a + (b + c)$$

$$(a * b) * c = a * (b * c)$$

- Identity

$$0 \neq 1, a + 0 = a, a * 1 = a$$

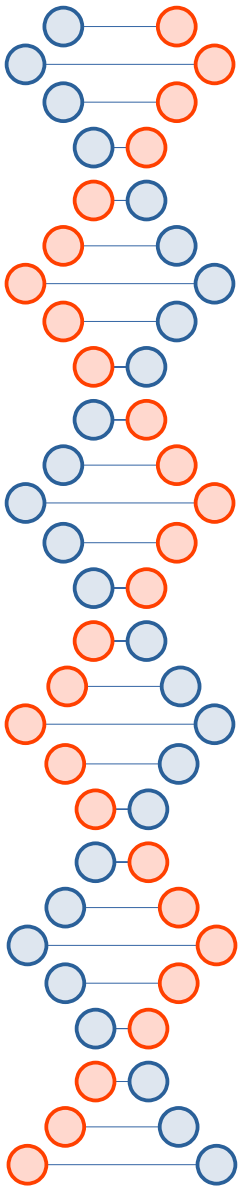
- Inverse

$$a + -a = 0$$

$$~~a * a^{-1} = 1~~$$

- Distributive

$$a * (b + c) = (a * b) + (a * c)$$

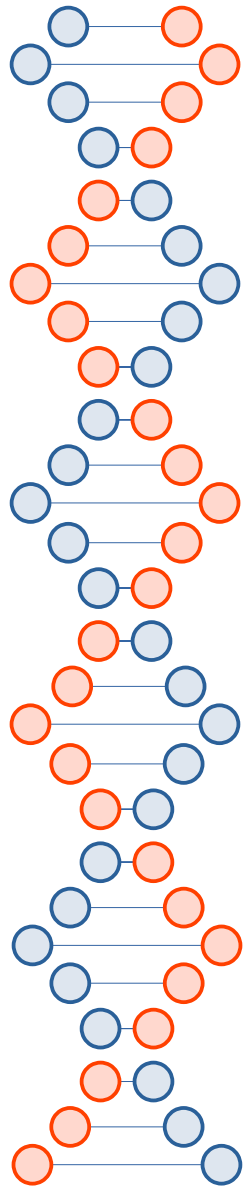


RSA

- Security is based on the hardness of integer factorization

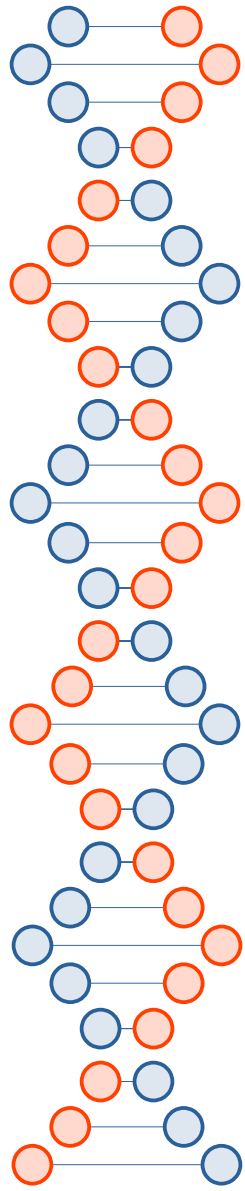
Feature	Diffie-Hellman (DH) Finite-field version	RSA
Modulus type	Prime modulus p	Composite modulus $n = pq$
Role of φ	Public and easy to compute $\varphi(p) = p - 1$	Secret trapdoor information $\varphi(n) = (p - 1)(q - 1)$
Role of EEA	Not needed for the core key-agreement operation	Computes the private exponent $d = e^{-1} \pmod{\varphi(n)}$
Hardness	Computational Diffie-Hellman (CDH) assumption Solving discrete logs breaks DH.	RSA inversion assumption Factoring n breaks RSA.

EEA = Extended Euclidean Algorithm. A sufficient attack need not be equivalent to breaking the scheme.



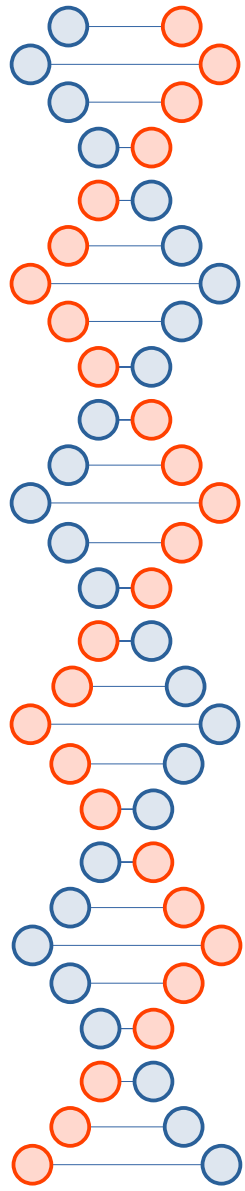
$$n = pq$$

- p and q are primes, suppose $p = 61$, $q = 53$
- $n = 3233$
- Euler's totient counts the positive integers up to n that are relatively prime to n
 - $(61 - 1)(53 - 1) = 3120$
- Carmichael's totient function = $\text{lcm}(60, 52)$
 - 52, 104, 156, 208, 260, 312, 364, 416, 468, 520, 572, 624, 676, 728, 780
 - 60, 120, 180, 240, 300, 360, 420, 480, 540, 600, 660, 720, 780
- Choose $1 < e < 780$ coprime to 780, *e.g.*, $e = 17$
- d is the modular multiplicative inverse of e , $d = 413$
- $413 * 17 \bmod 780 = 1$



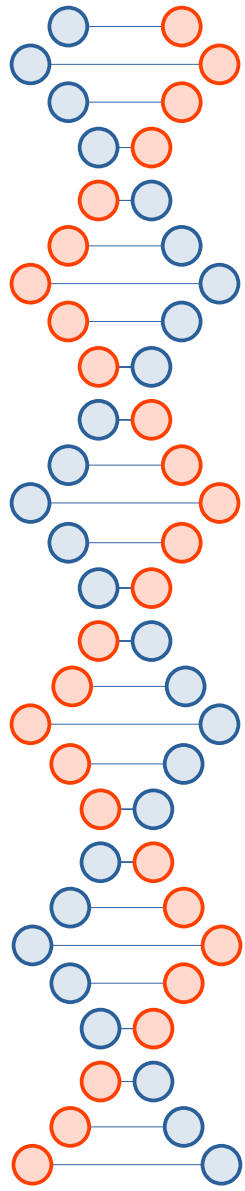
$$n = pq$$

- p and q are primes, suppose $p = 61$, $q = 53$
- $n = 3233$
- Euler's totient counts the positive integers up to n that are relatively prime to n
 - $(61 - 1)(53 - 1) = 3120$
- Carmichael's totient function = $\text{lcm}(60, 52)$
 - 52, 104, 156, 208, 260, 312, 364, 416, 468, 520, 572, 624, 676, 728, 780
 - 60, 120, 180, 240, 300, 360, 420, 480, 540, 600, 660, 720, 780
- Choose $1 < e < 780$ coprime to 780, *e.g.*, $e = 17$
- d is the modular multiplicative inverse of e , $d = 413$
- $413 * 17 \bmod 780 = 1$

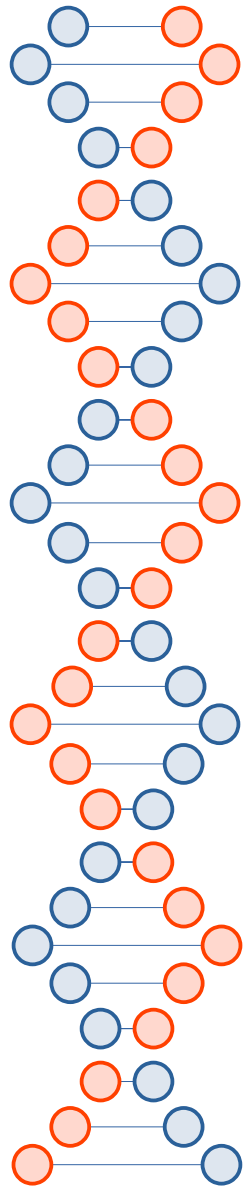


$$n = pq$$

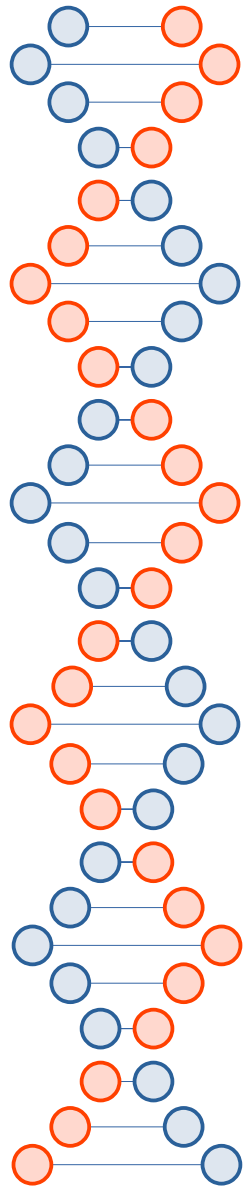
- p and q are primes, suppose $p = 61$, $q = 53$
- $n = 3233$
- Euler's totient counts the positive integers up to n that are relatively prime to n
 - $(61 - 1)(53 - 1) = 3120$
- Carmichael's totient function = $\text{lcm}(60, 52)$
 - 52, 104, 156, 208, 260, 312, 364, 416, 468, 520, 572, 624, 676, 728, 780
 - 60, 120, 180, 240, 300, 360, 420, 480, 540, 600, 660, 720, 780
- Choose $1 < e < 780$ coprime to 780, e.g., $e = 17$
- d is the modular multiplicative inverse of e , $d = 413$
- $413 * 17 \bmod 780 = 1$



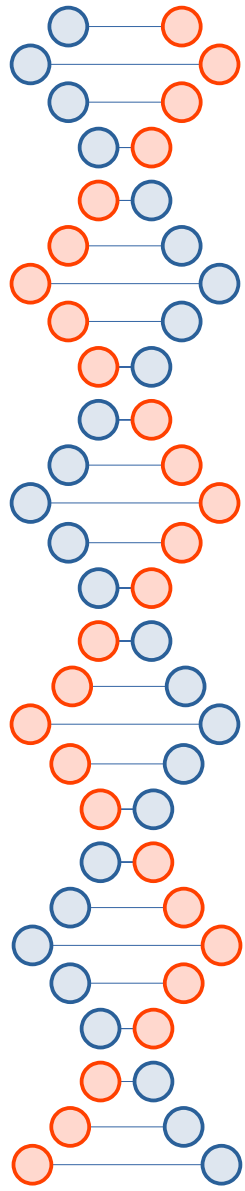
- Public key is ($n = 3233$, $e = 17$)
- Private key is ($n = 3233$, $d = 413$)
- Encryption: $c(m = 65) = 65^{17} \bmod 3233 = 2790$
 - What if m is a multiple of 61 or 53?
- Decryption: $m = 2790^{413} \bmod 3233 = 65$
- But, signatures are a more important application of RSA...
 - Signature: $s = 100^{413} \bmod 3233 = 1391$
 - Verification: $100 = 1391^{17} \bmod 3233$
- Fast modular exponentiation is the trick
 - Also need extended Euclidean algorithm
- Using RSA for key exchange or encryption is often a red flag, more commonly used for signatures



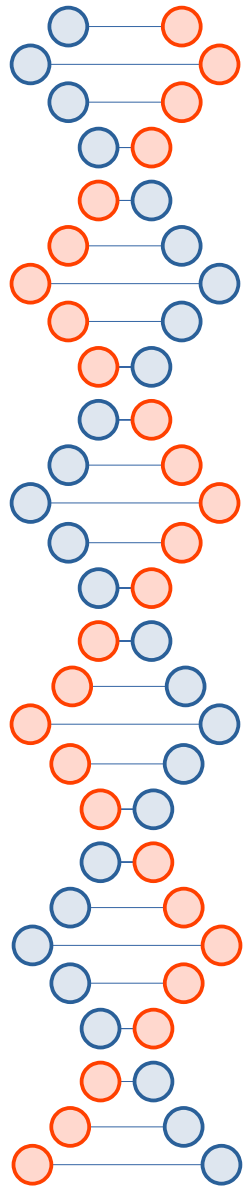
- Public key is $(n = 3233, e = 17)$
- Private key is $(n = 3233, d = 413)$
- Encryption: $c(m = 65) = 65^{17} \bmod 3233 = 2790$
 - What if m is a multiple of 61 or 53?
- Decryption: $m = 2790^{413} \bmod 3233 = 65$
- But, signatures are a more important application of RSA...
 - Signature: $s = 100^{413} \bmod 3233 = 1391$
 - Verification: $100 = 1391^{17} \bmod 3233$
- Fast modular exponentiation is the trick
 - Also need extended Euclidean algorithm
- Using RSA for key exchange or encryption is often a red flag, more commonly used for signatures



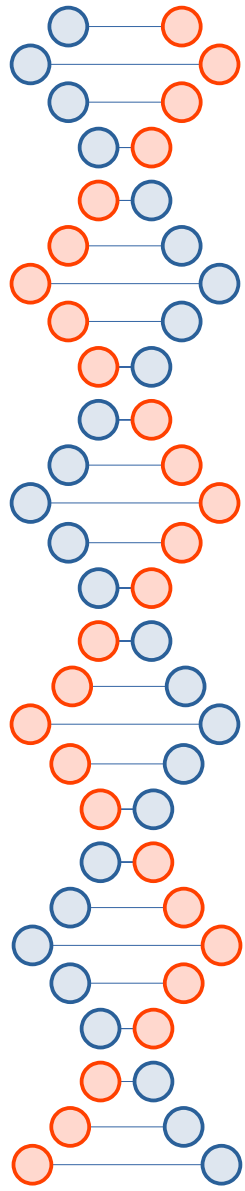
- Public key is $(n = 3233, e = 17)$
- Private key is $(n = 3233, d = 413)$
- Encryption: $c(m = 65) = 65^{17} \bmod 3233 = 2790$
 - What if m is a multiple of 61 or 53?
- Decryption: $m = 2790^{413} \bmod 3233 = 65$
- But, signatures are a more important application of RSA...
 - Signature: $s = 100^{413} \bmod 3233 = 1391$
 - Verification: $100 = 1391^{17} \bmod 3233$
- Fast modular exponentiation is the trick
 - Also need extended Euclidean algorithm
- Using RSA for key exchange or encryption is often a red flag, more commonly used for signatures



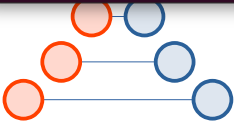
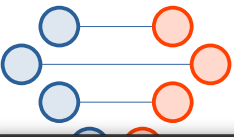
- Public key is $(n = 3233, e = 17)$
- Private key is $(n = 3233, d = 413)$
- Encryption: $c(m = 65) = 65^{17} \bmod 3233 = 2790$
 - What if m is a multiple of 61 or 53?
- Decryption: $m = 2790^{413} \bmod 3233 = 65$
- But, signatures are a more important application of RSA...
 - Signature: $s = 100^{413} \bmod 3233 = 1391$
 - Verification: $100 = 1391^{17} \bmod 3233$
- Fast modular exponentiation is the trick
 - Also need extended Euclidean algorithm
- Using RSA for key exchange or encryption is often a red flag, more commonly used for signatures



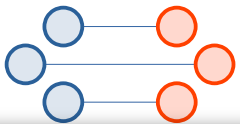
- Public key is $(n = 3233, e = 17)$
- Private key is $(n = 3233, d = 413)$
- Encryption: $c(m = 65) = 65^{17} \bmod 3233 = 2790$
 - What if m is a multiple of 61 or 53?
- Decryption: $m = 2790^{413} \bmod 3233 = 65$
- But, signatures are a more important application of RSA...
 - Signature: $s = 100^{413} \bmod 3233 = 1391$
 - Verification: $100 = 1391^{17} \bmod 3233$
- Fast modular exponentiation is the trick
 - Also need extended Euclidean algorithm
- Using RSA for key exchange or encryption is often a red flag, more commonly used for signatures



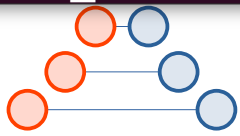
- Public key is ($n = 3233$, $e = 17$)
- Private key is ($n = 3233$, $d = 413$)
- Encryption: $c(m = 65) = 65^{17} \bmod 3233 = 2790$
 - What if m is a multiple of 61 or 53?
- Decryption: $m = 2790^{413} \bmod 3233 = 65$
- But, signatures are a more important application of RSA...
 - Signature: $s = 100^{413} \bmod 3233 = 1391$
 - Verification: $100 = 1391^{17} \bmod 3233$
- Fast modular exponentiation is the trick
 - **Also need extended Euclidean algorithm**
- Using RSA for key exchange or encryption is often a red flag, more commonly used for signatures

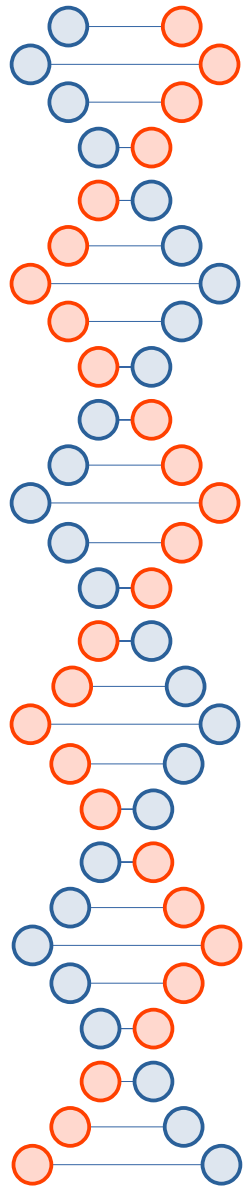


```
jedi@route66:~$ python3
Python 3.8.2 (default, Jul 16 2020, 14:00:26)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> for i in range (52, 781, 52):
...     for j in range (60, 781, 60):
...         if (i == j):
...             print(i)
...
780
>>> print((413 * 17) % 780)
1
>>> print(pow(2790, 413, 3233))
65
>>> print(pow(65, 17, 3233))
2790
>>> print(pow(100, 413, 3233))
1391
>>> print(pow(1391, 17, 3233))
100
>>> □
```



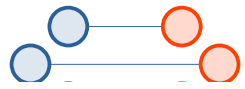
```
jedi@route66: ~  
1  
>>> print(pow(2790, 413, 3233))  
65  
>>> print(pow(65, 17, 3233))  
2790  
>>> print(pow(100, 413, 3233))  
1391  
>>> print(pow(1391, 17, 3233))  
100  
>>> print(pow(7, 17, 3233))  
2369  
>>> print((2369*2790) % 3233)  
1258  
>>> print(pow(1258, 413, 3233))  
455  
>>> print(7*65)  
455  
>>> print("{0:b}".format(78913))  
10011010001000001  
>>> print("{0:b}".format(78913*32))  
1001101000100000100000  
>>> print("{0:b}".format(78913<<5))  
1001101000100000100000  
>>> 
```





“Relatively prime”

- 9 is not prime, $9 = 3^2$
- 13 is prime
- 10 is not prime, $10 = 5 \cdot 2$
- 9 and 10 are relatively prime, $\gcd(9, 10) = 1$
- 5 and 10 are not relatively prime, $\gcd(5, 10) = 5$
- Also called “coprime”



$$M^{\phi(n)} \equiv 1 \pmod{n} . \quad (3)$$

Here $\phi(n)$ is the Euler totient function giving number of positive integers less than n which are relatively prime to n . For prime numbers p ,

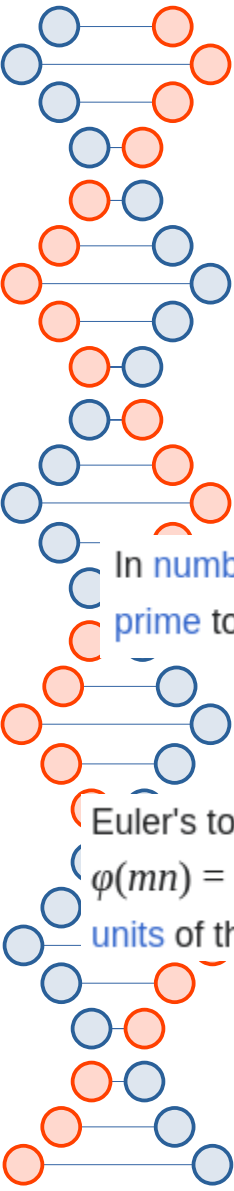
$$\phi(p) = p - 1 .$$

In our case, we have by elementary properties of the totient function [7]:

$$\begin{aligned} \phi(n) &= \phi(p) \cdot \phi(q) \\ &= (p - 1) \cdot (q - 1) \\ &= n - (p + q) + 1 . \end{aligned} \quad (4)$$

Since d is relatively prime to $\phi(n)$, it has a multiplicative inverse e in the ring of integers modulo $\phi(n)$:



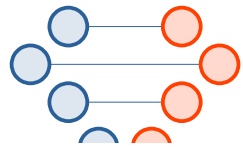


Euler's totient function

- https://en.wikipedia.org/wiki/Euler%27s_totient_function

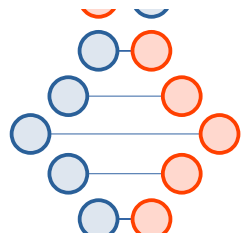
In **number theory**, **Euler's totient function** counts the positive integers up to a given integer n that are **relatively prime** to n . It is written using the Greek letter **phi** as $\varphi(n)$ or $\phi(n)$, and may also be called **Euler's phi function**.

Euler's totient function is a **multiplicative function**, meaning that if two numbers m and n are relatively prime, then $\varphi(mn) = \varphi(m)\varphi(n)$.^{[4][5]} This function gives the **order** of the **multiplicative group of integers modulo n** (the **group** of **units** of the **ring $\mathbb{Z}/n\mathbb{Z}$**).^[6] It is also used for defining the **RSA encryption system**.



Since d is relatively prime to $\phi(n)$, it has a multiplicative inverse e in the ring of integers modulo $\phi(n)$:

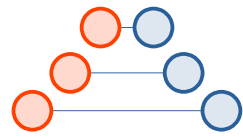
$$e \cdot d \equiv 1 \pmod{\phi(n)}. \quad (5)$$



$$\begin{aligned} D(E(M)) &\equiv (E(M))^d \equiv (M^e)^d \pmod{n} = M^{e \cdot d} \pmod{n} \\ E(D(M)) &\equiv (D(M))^e \equiv (M^d)^e \pmod{n} = M^{e \cdot d} \pmod{n} \end{aligned}$$

and

$$M^{e \cdot d} \equiv M^{k \cdot \phi(n) + 1} \pmod{n} \text{ (for some integer } k\text{).}$$





From (3) we see that for all M such that p does not divide M

$$M^{p-1} \equiv 1 \pmod{p}$$

and since $(p-1)$ divides $\phi(n)$

$$M^{k \cdot \phi(n) + 1} \equiv M \pmod{p}.$$

This is trivially true when $M \equiv 0 \pmod{p}$, so that this equality actually holds for *all* M . Arguing similarly for q yields

$$M^{k \cdot \phi(n) + 1} \equiv M \pmod{q}.$$

Together these last two equations imply that for all M ,

$$M^{e \cdot d} \equiv M^{k \cdot \phi(n) + 1} \equiv M \pmod{n}.$$

This implies (1) and (2) for all $M, 0 \leq M < n$. Therefore E and D are inverse permutations. (We thank Rich Schroepel for suggesting the above improved version of the authors' previous proof.)





Computing $M^e \pmod n$ requires at most $2 \cdot \log_2(e)$ multiplications and $2 \cdot \log_2(e)$ divisions using the following procedure (decryption can be performed similarly using d instead of e):

Step 1. Let $e_k e_{k-1} \dots e_1 e_0$ be the binary representation of e .

Step 2. Set the variable C to 1.

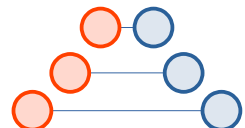
Step 3. Repeat steps 3a and 3b for $i = k, k-1, \dots, 0$:

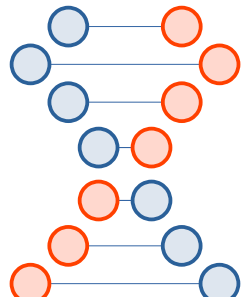
Step 3a. Set C to the remainder of C^2 when divided by n .

Step 3b. If $e_i = 1$, then set C to the remainder of $C \cdot M$ when divided by n .

Step 4. Halt. Now C is the encrypted form of M .

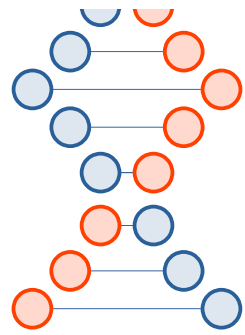
This procedure is called “exponentiation by repeated squaring and multiplication.” This procedure is half as good as the best; more efficient procedures are known. Knuth [3] studies this problem in detail.





Each user must (privately) choose two large random numbers p and q to create his own encryption and decryption keys. These numbers must be large so that it is not computationally feasible for anyone to factor $n = p \cdot q$. (Remember that n , but not p or q , will be in the public file.) We recommend using 100-digit (decimal) prime numbers p and q , so that n has 200 digits.

To find a 100-digit “random” prime number, generate (odd) 100-digit random numbers until a prime number is found. By the prime number theorem [7], about $(\ln 10^{100})/2 = 115$ numbers will be tested before a prime is found.

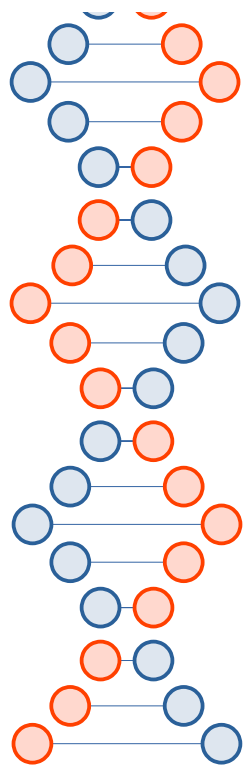


(About 665 bits, 2048 or 4096 are standard today)



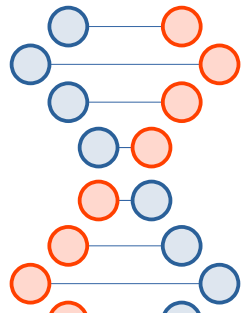
To test a large number b for primality we recommend the elegant “probabilistic” algorithm due to Solovay and Strassen [12]. It picks a random number a from a uniform distribution on $\{1, \dots, b-1\}$, and tests whether

$$\gcd(a, b) = 1 \text{ and } J(a, b) = a^{(b-1)/2} \pmod{b}, \tag{6}$$



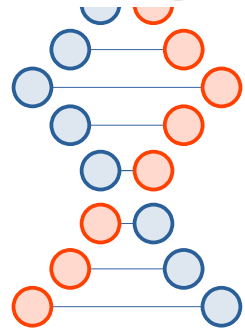
$n \backslash k$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1														
3	0	1	-1												
5	0	1	-1	-1	1										
7	0	1	1	-1	1	-1	-1								
9	0	1	1	0	1	1	0	1	1						
11	0	1	-1	1	1	1	-1	-1	-1	1	-1				
13	0	1	-1	1	1	-1	-1	-1	-1	1	1	-1	1		
15	0	1	1	0	1	0	0	-1	1	0	0	-1	0	-1	-1
17	0	1	1	-1	1	-1	-1	-1	1	1	-1	-1	-1	1	-1

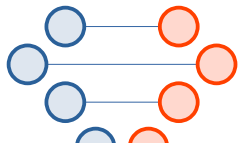
https://en.wikipedia.org/wiki/Jacobi_symbol



To gain additional protection against sophisticated factoring algorithms, p and q should differ in length by a few digits, both $(p - 1)$ and $(q - 1)$ should contain large prime factors, and $\gcd(p - 1, q - 1)$ should be small. The latter condition is easily checked.

To find a prime number p such that $(p - 1)$ has a large prime factor, generate a large random prime number u , then let p be the first prime in the sequence $i \cdot u + 1$, for $i = 2, 4, 6, \dots$ (This shouldn't take too long.) Additional security is provided by ensuring that $(u - 1)$ also has a large prime factor.

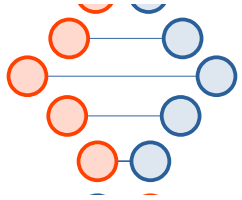




C How to Choose d

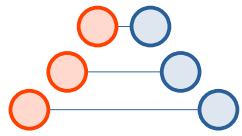
It is very easy to choose a number d which is relatively prime to $\phi(n)$. For example, any prime number greater than $\max(p, q)$ will do. It is important that d should be chosen from a large enough set so that a cryptanalyst cannot find it by direct search.

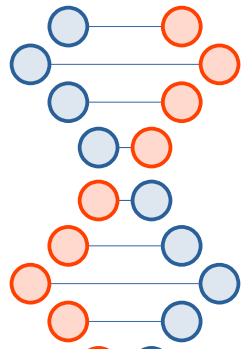
D How to Compute e from d and $\phi(n)$



Extended Euclidean Algorithm

If e turns out to be less than $\log_2(n)$, start over by choosing another value of d . This guarantees that every encrypted message (except $M = 0$ or $M = 1$) undergoes some “wrap-around” (reduction modulo n) .

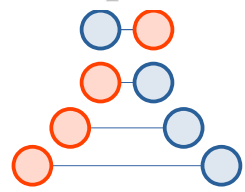


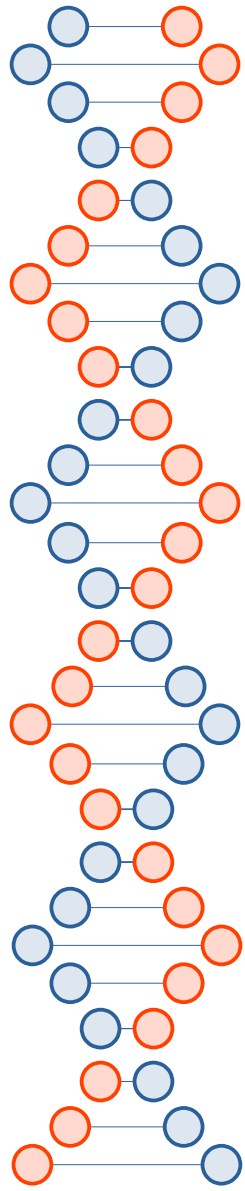


The era of “electronic mail” [10] may soon be upon us;

encryption keys. (We assume that the intruder cannot modify or insert messages into the channel.) Ralph Merkle has developed another solution [5] to this problem.

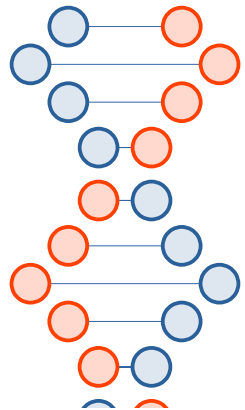
A public-key cryptosystem can be used to “bootstrap” into a standard encryption scheme such as the NBS method. Once secure communications have been established, the first message transmitted can be a key to use in the NBS scheme to encode all following messages. This may be desirable if encryption with our method is slower than with the standard scheme. (The NBS scheme is probably somewhat faster if special-purpose hardware encryption devices are used; our scheme may be faster on a general-purpose computer since multiprecision arithmetic operations are simpler to implement than complicated bit manipulations.)



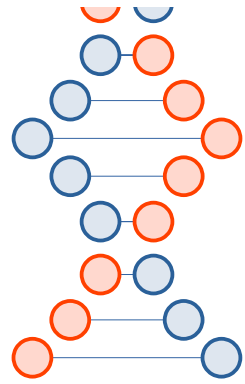


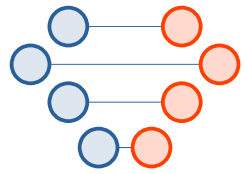
200-digit message M

$$\log_2(10^{200}) = \text{about 665 bits}$$



Since no techniques exist to *prove* that an encryption scheme is secure, the only test available is to see whether anyone can think of a way to break it. The NBS standard was “certified” this way; seventeen man-years at IBM were spent fruitlessly trying to break that scheme. Once a method has successfully resisted such a concerted attack it may for practical purposes be considered secure. (Actually there is some controversy concerning the security of the NBS method [2].)



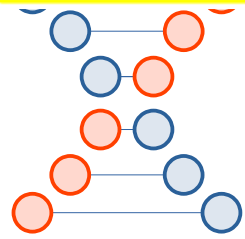


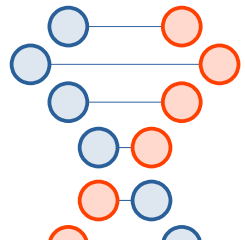
How can n be factored using $\phi(n)$? First, $(p + q)$ is obtained from n and $\phi(n) = n - (p + q) + 1$. Then $(p - q)$ is the square root of $(p + q)^2 - 4n$. Finally, q is half the difference of $(p + q)$ and $(p - q)$.

Therefore breaking our system by computing $\phi(n)$ is no easier than breaking our system by factoring n . (This is why n must be composite; $\phi(n)$ is trivial to compute if n is prime.)



A knowledge of d enables n to be factored as follows. Once a cryptanalyst knows d he can calculate $e \cdot d - 1$, which is a multiple of $\phi(n)$. Miller [6] has shown that n can be factored using any multiple of $\phi(n)$. Therefore if n is large a cryptanalyst should not be able to determine d any easier than he can factor n .

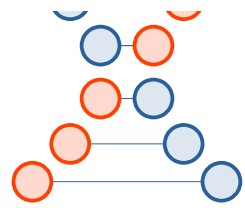


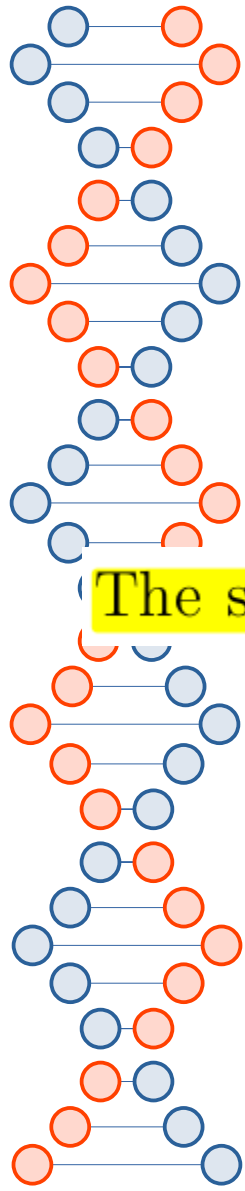


D Computing D in Some Other Way

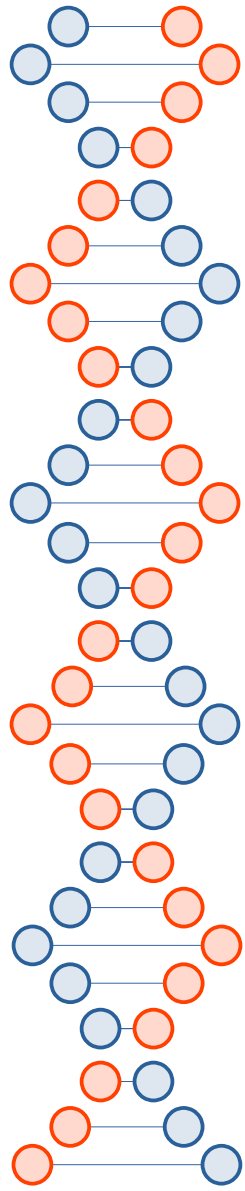
Although this problem of “computing e -th roots modulo n without factoring n ” is not a well-known difficult problem like factoring, we feel reasonably confident that it is computationally intractable. It may be possible to prove that any general method of breaking our scheme yields an efficient factoring algorithm. This would establish that any way of breaking our scheme must be as difficult as factoring. We have not been able to prove this conjecture, however.

Our method should be certified by having the above conjecture of intractability withstand a concerted attempt to disprove it. The reader is challenged to find a way to “break” our method.





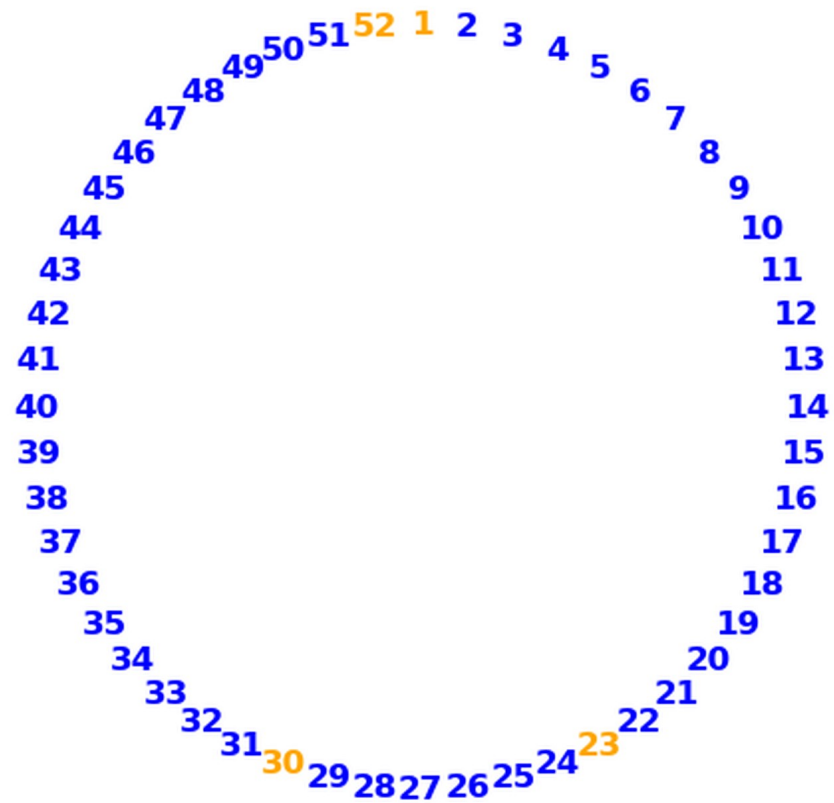
The security of this system needs to be examined in more detail.



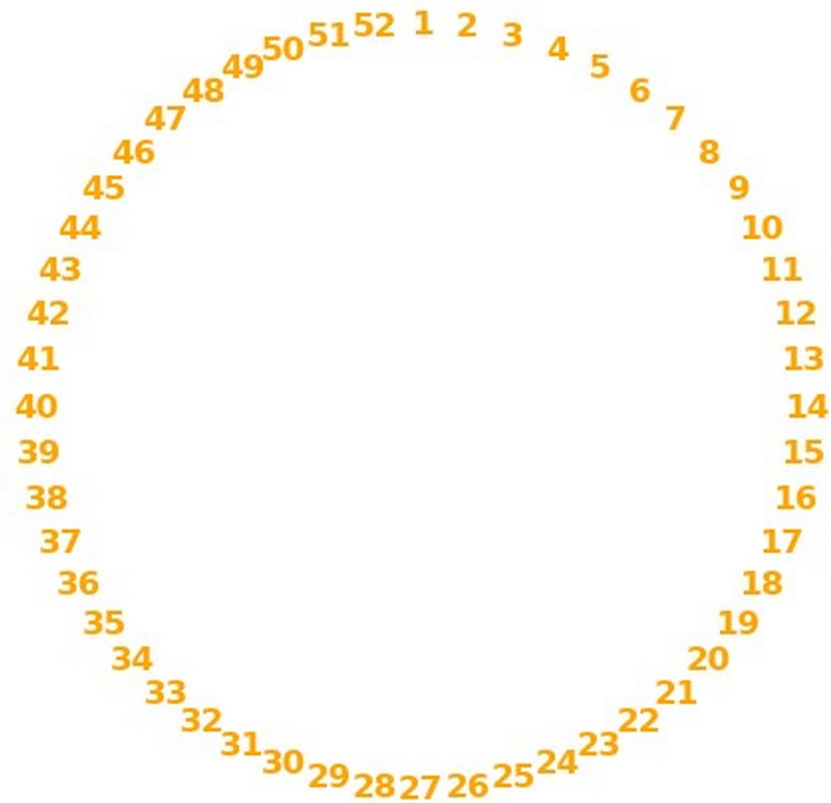
Side notes

- GCHQ claims to have invented RSA in 1973, and declassified this info in 1997
- In my own research (e.g., looking for amateurish crypto in Android apps) using RSA for key distribution is often a red flag
 - An authenticated version of Diffie-Hellman is better, most common thing these days is ECDH (Elliptic Curve Diffie-Hellman)

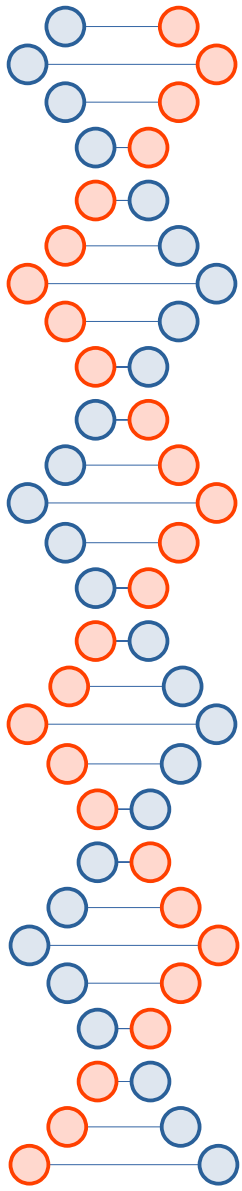
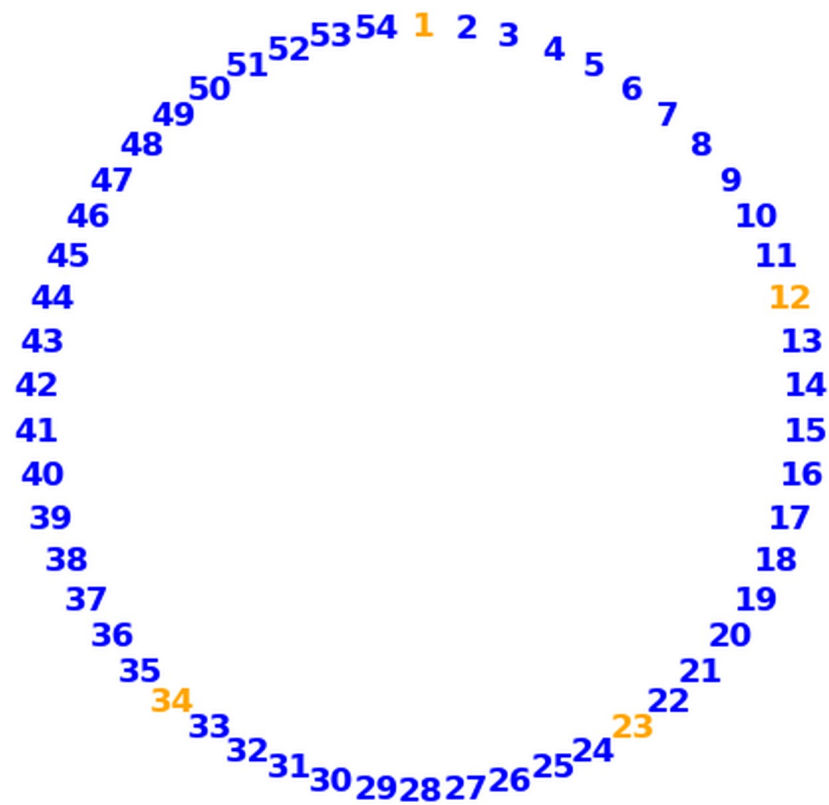
Orange are powers of 30 mod 53
Order is 4

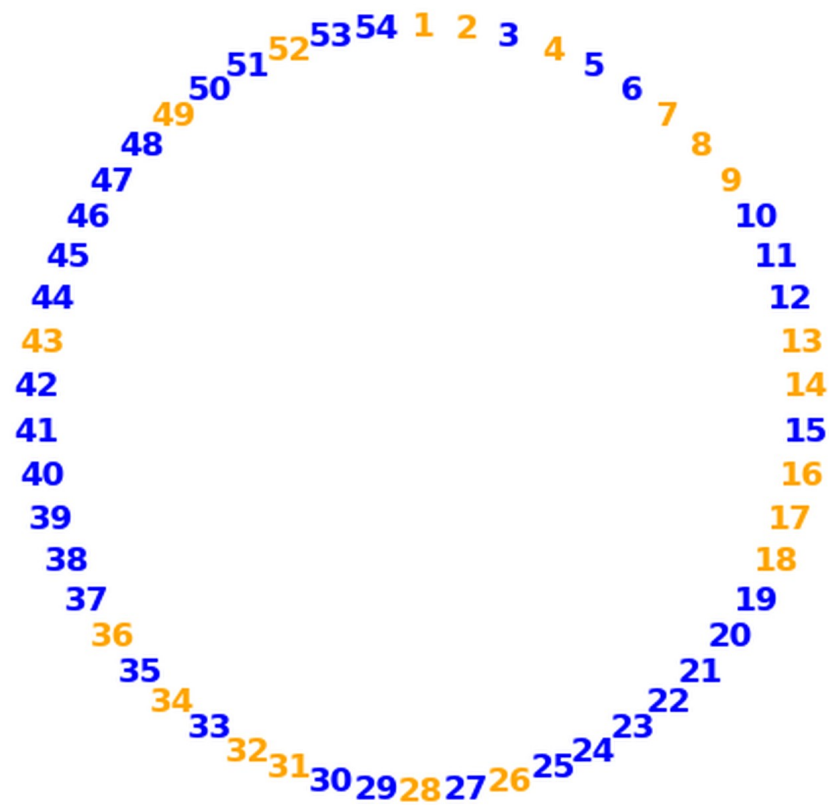
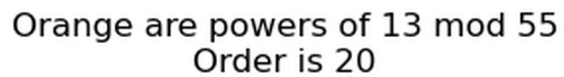


Orange are powers of 12 mod 53
Order is 52

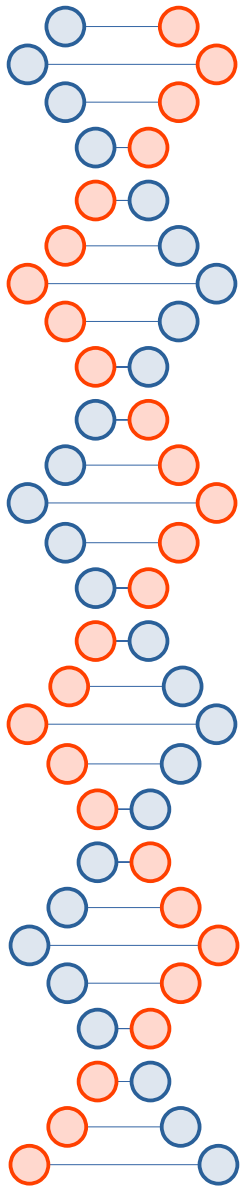
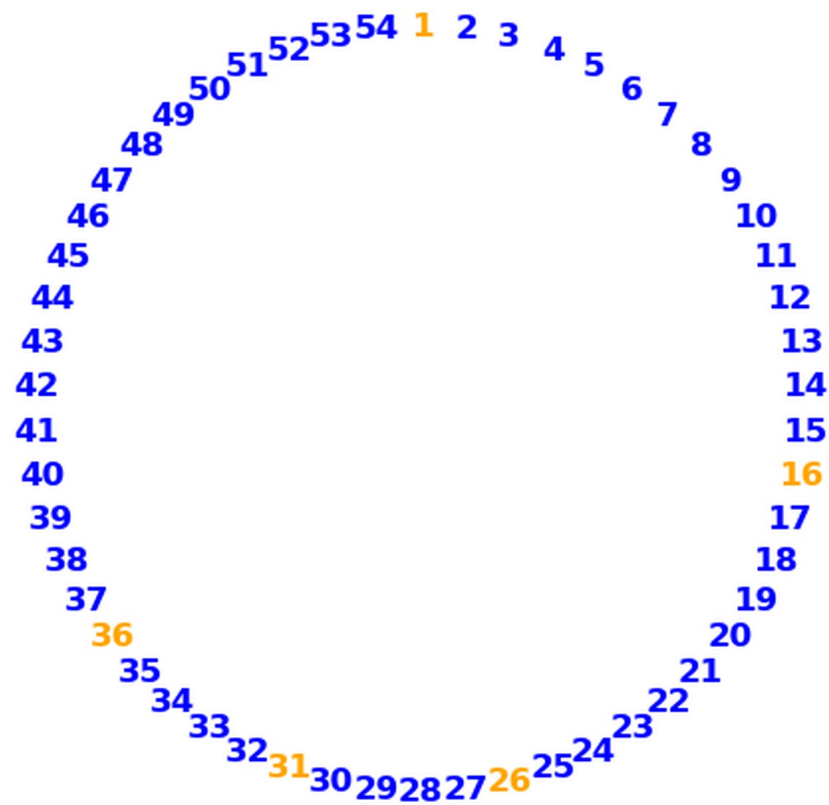


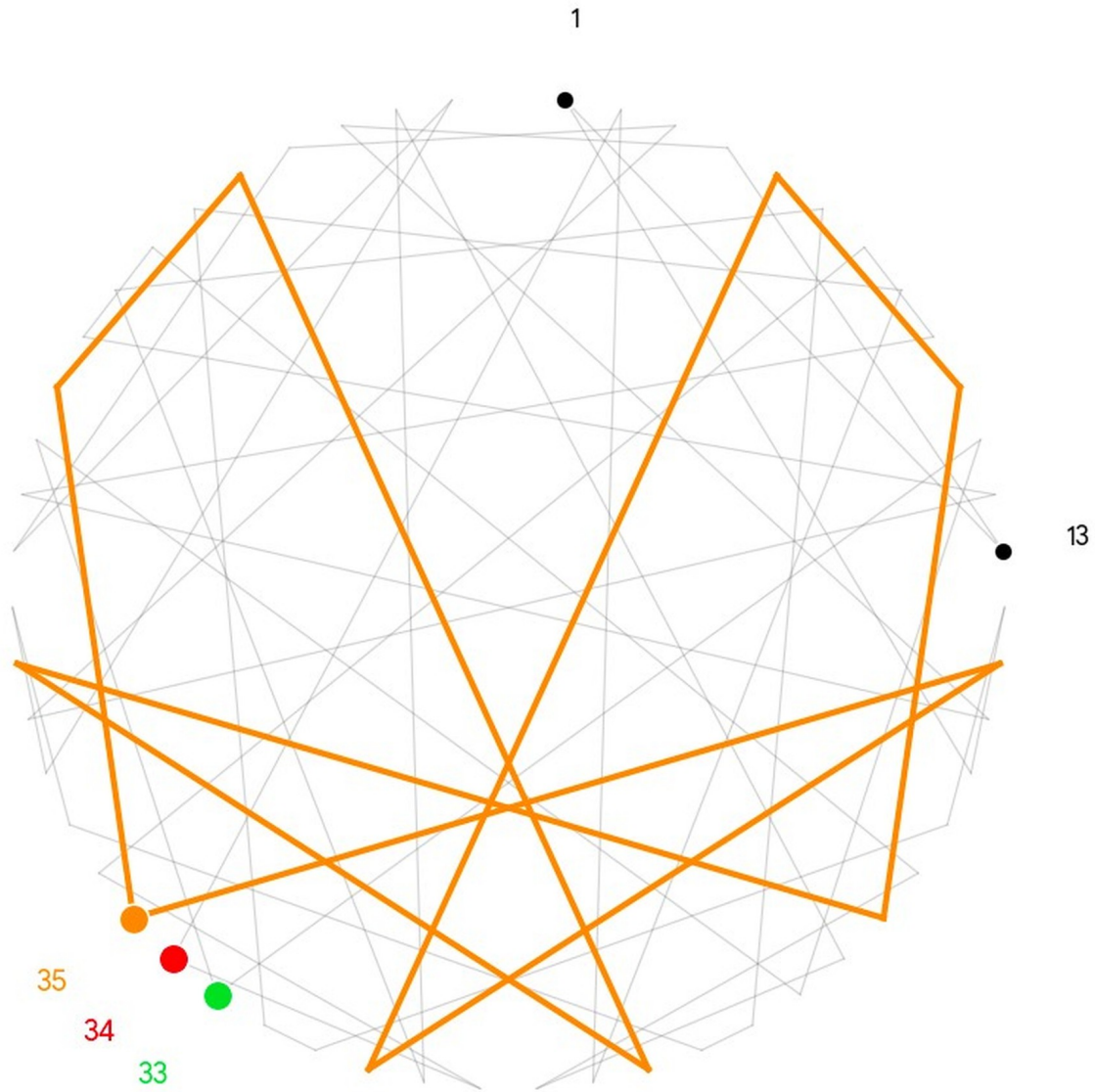
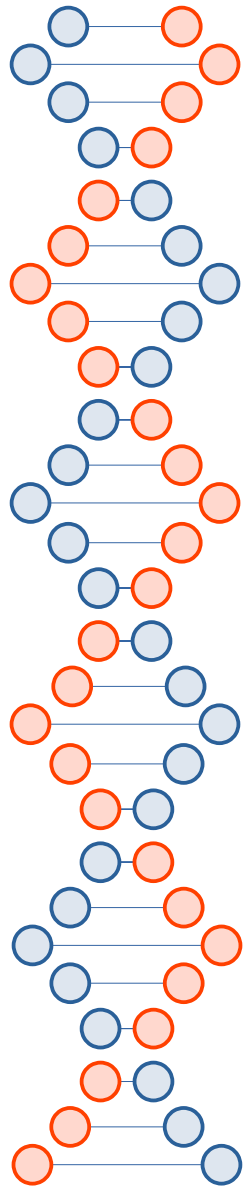
Orange are powers of 12 mod 55
Order is 4

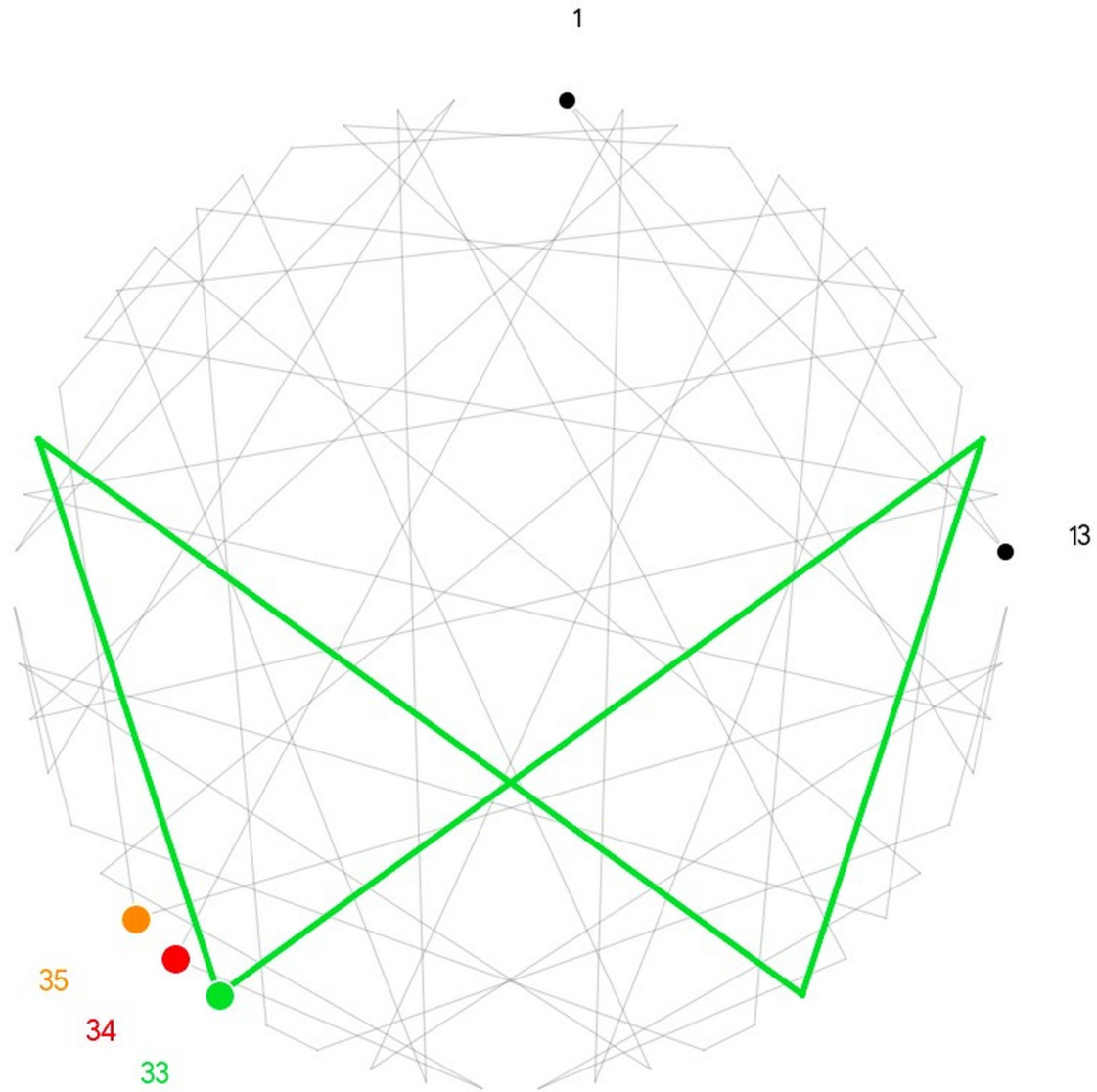
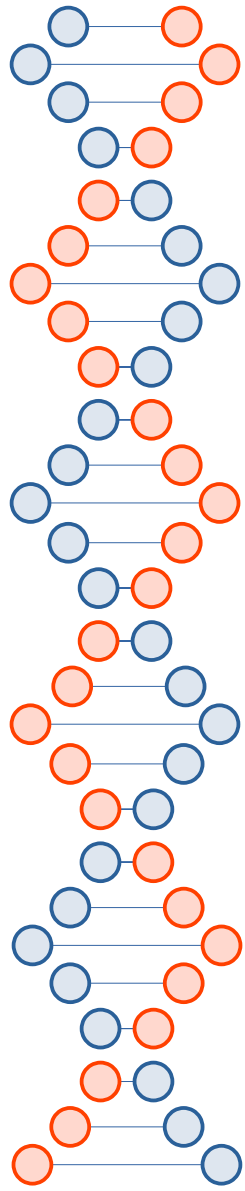


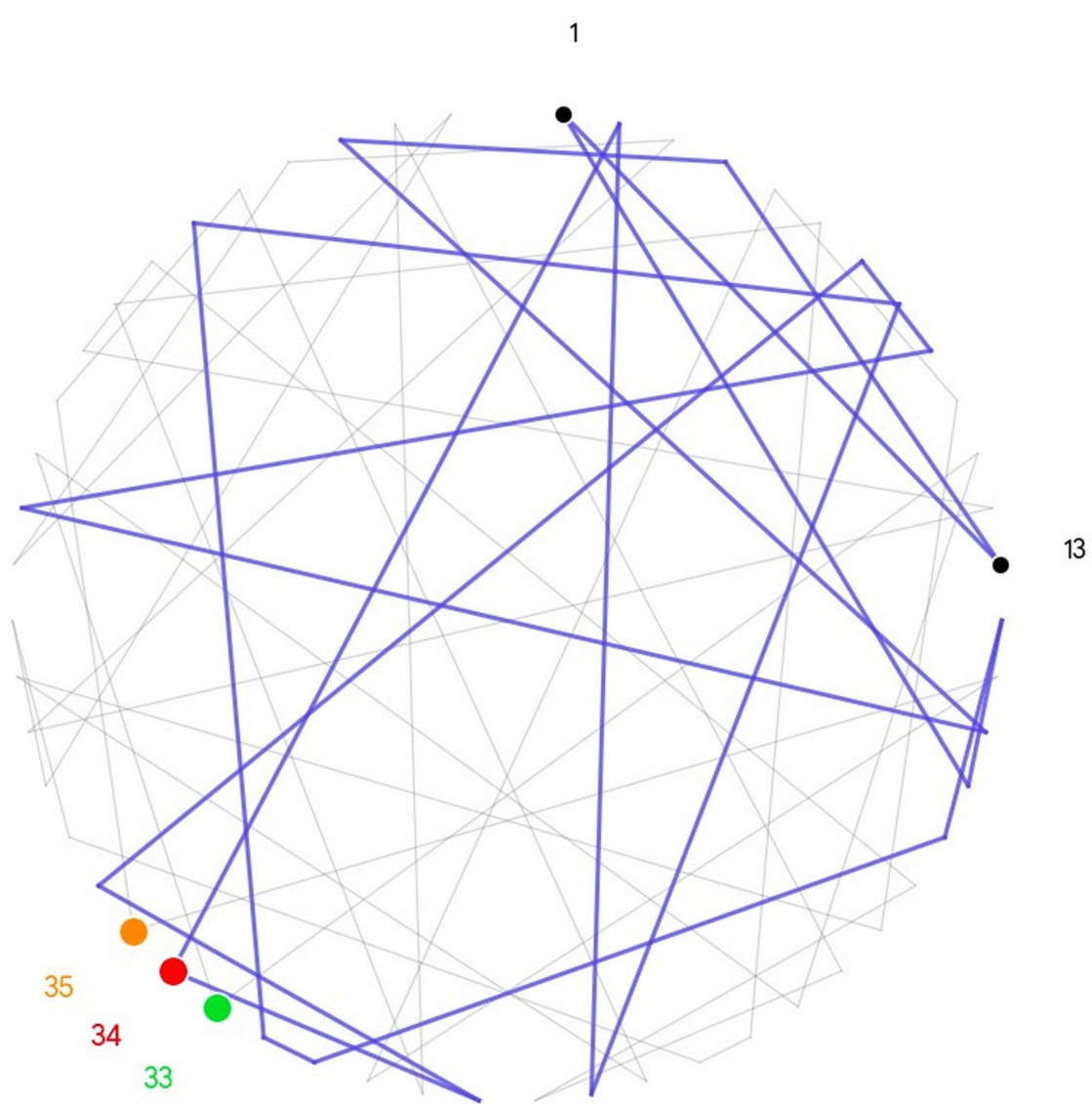
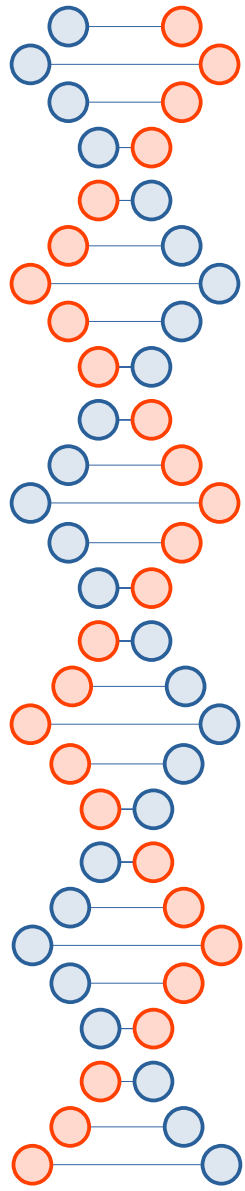


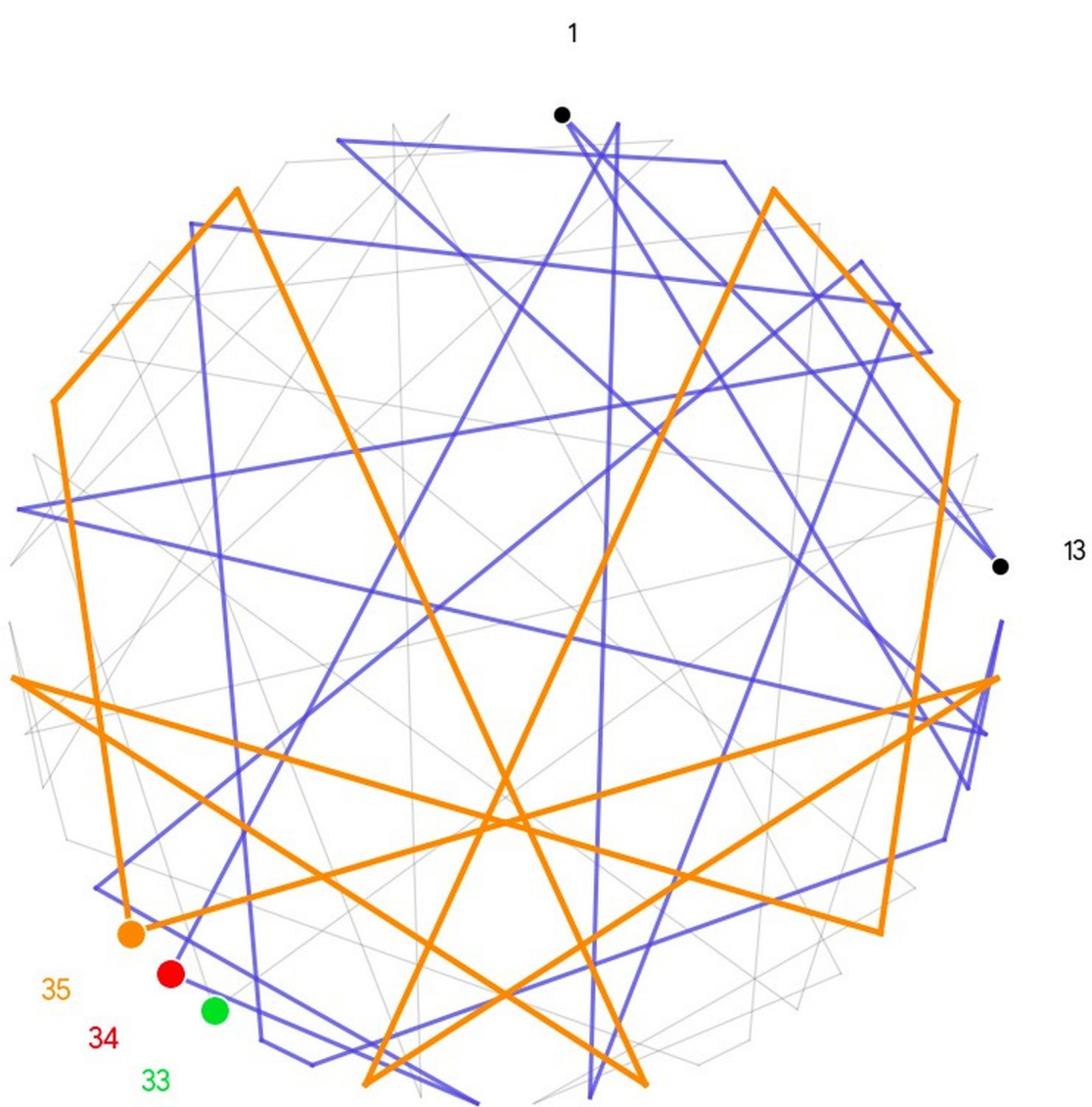
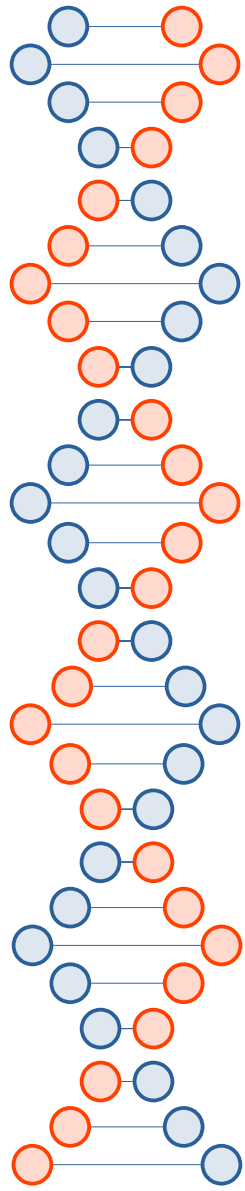
Orange are powers of 16 mod 55
Order is 5

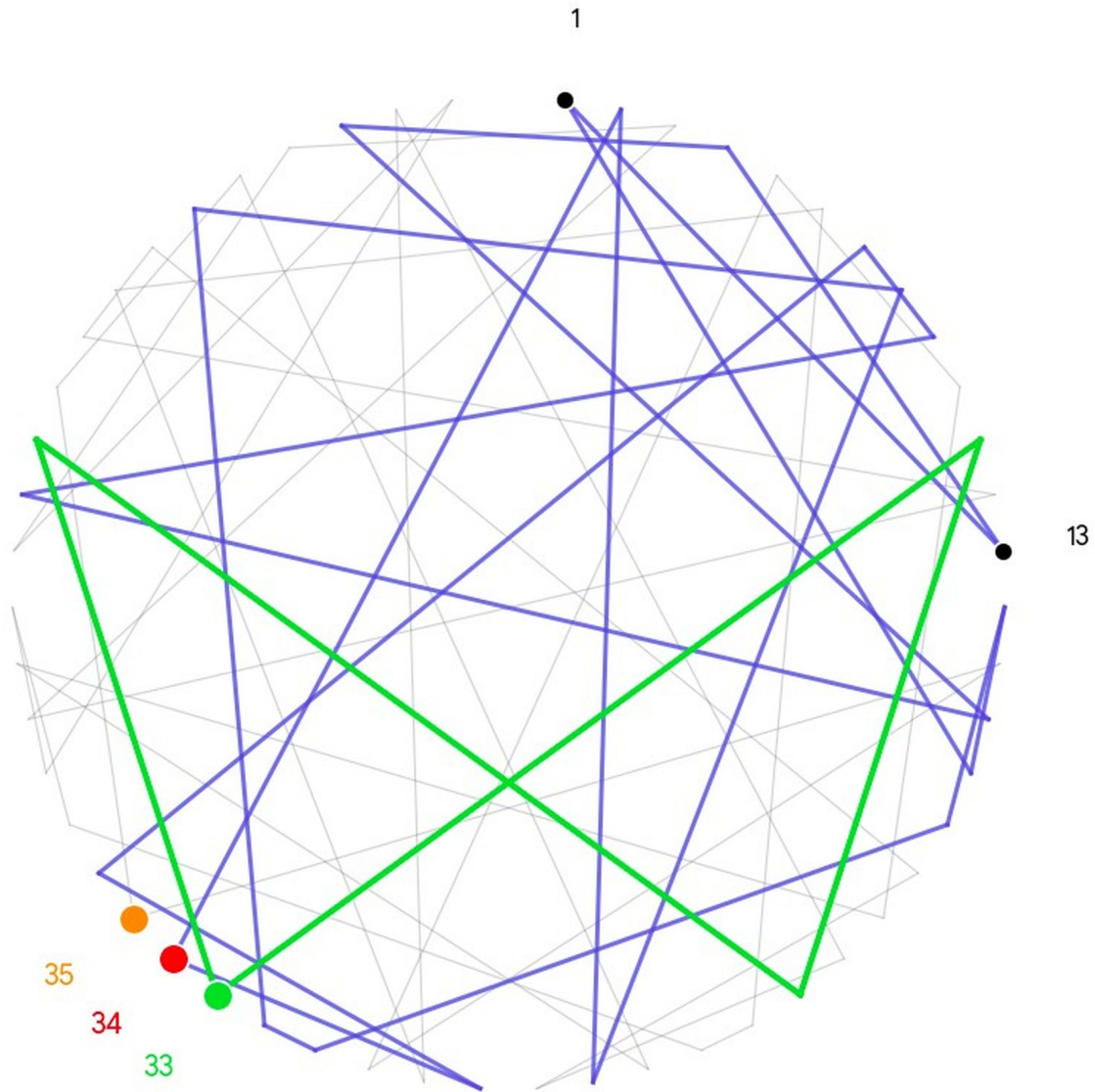
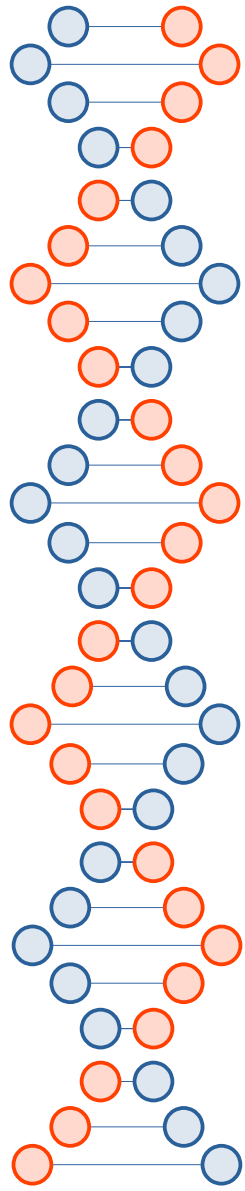


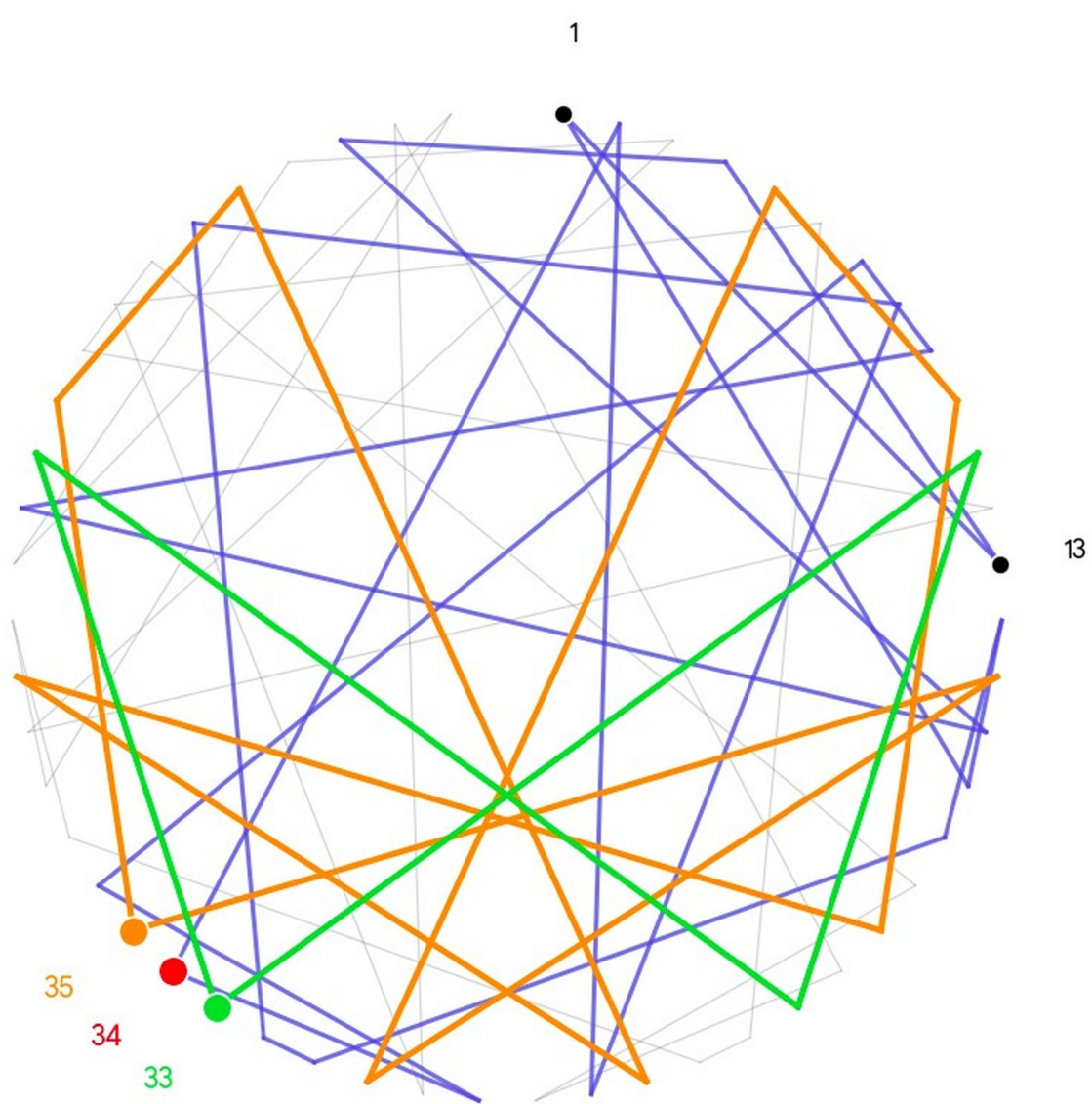
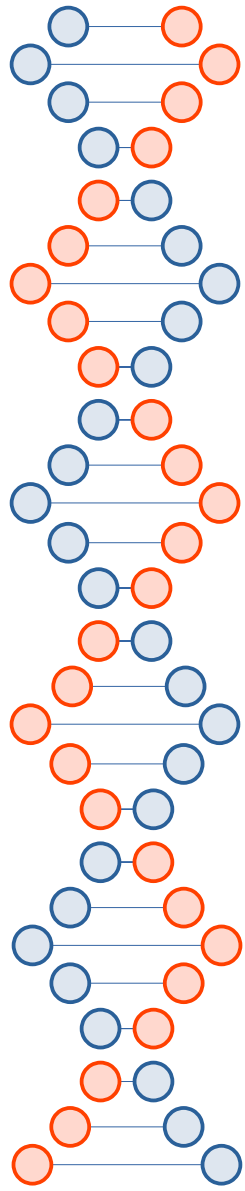


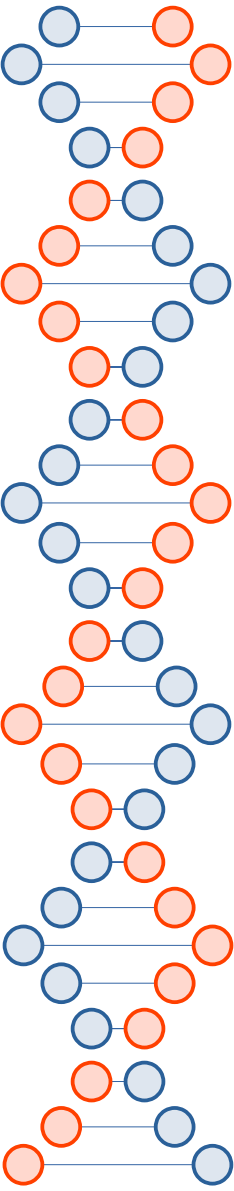






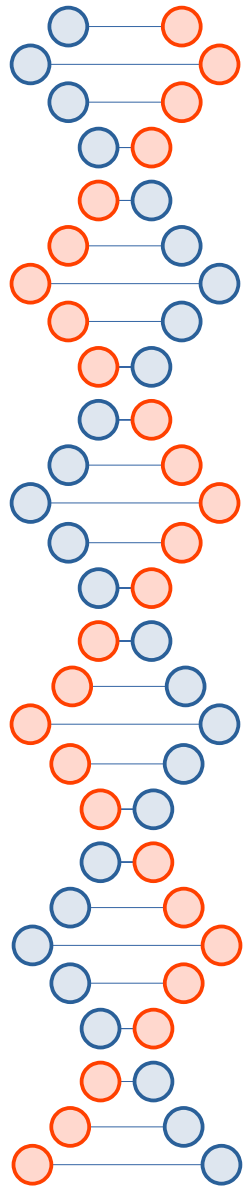






Spirograph takeaways

- Don't try to encrypt/sign multiples of p or q with RSA
 - $n = pq$
 - We never actually worry about this, because it's astronomically unlikely. Why?
- Composite makes all the difference
 - DH has no trap door
 - RSA has no primitive root, or guarantee of a multiplicative inverse
- If you know the order of $g^x \bmod n$, you can break RSA
 - Guess what quantum computers are good at? FFTs.



Cryptography Engineering by Ferguson *et al.*

