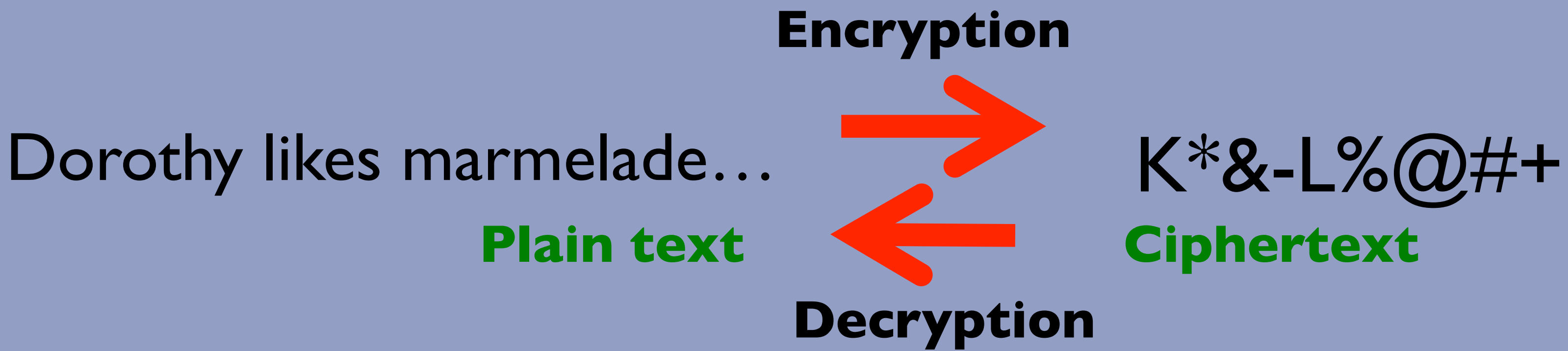
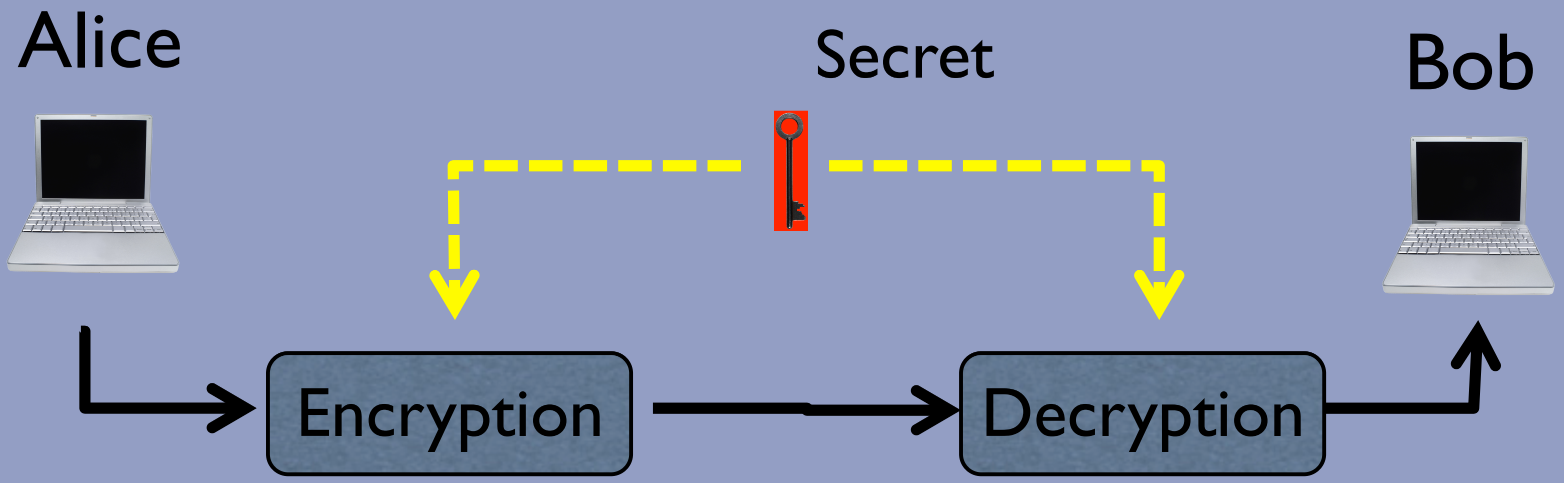
Data Encryption

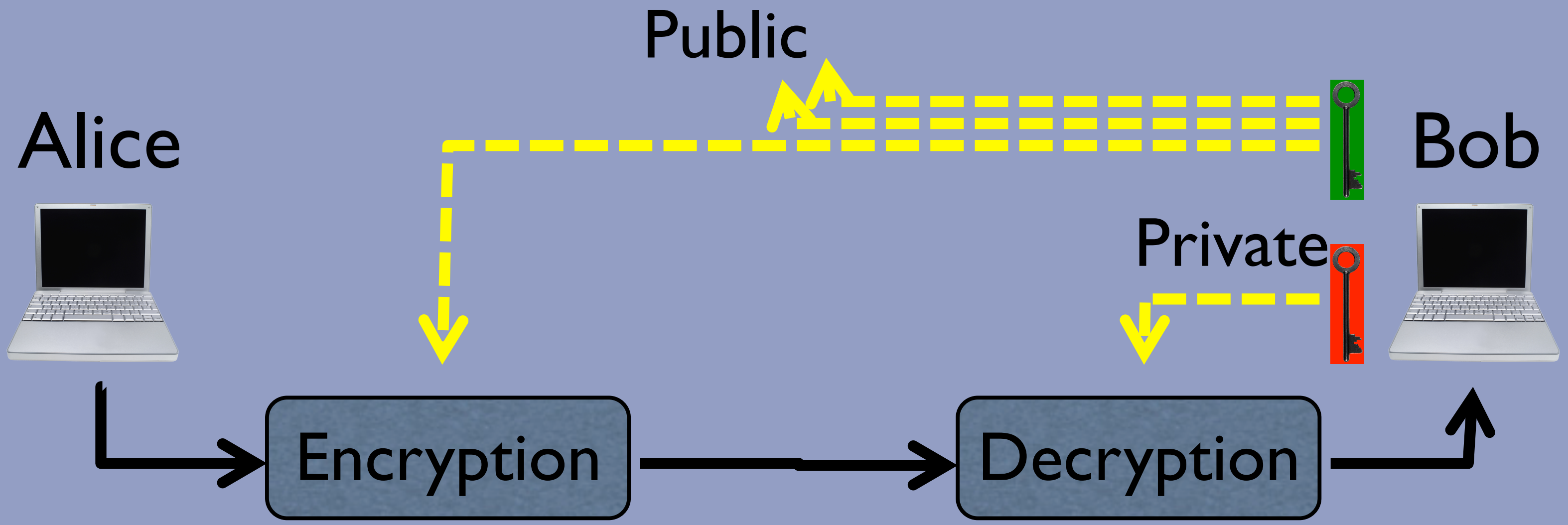
Not to be confused with Data Encodings!!!!!!



- Symmetric Encryption / Private Key: the same key for both encryption and decryption
- Asymmetric Encryption/ Public Key Encryption: one public key for encrypting and one local/private key for decrypting

In other words (pictures)





Symmetric-key cryptography

Traditionally as early as Ceasar (warfare)

- Substitution ciphers: one symbol (character) at a time is replaced with another symbol
    - Monoalphabetic: a symbol is always replaced by the same symbol regardless of its position
    - Polyalphabetic: depending on its position symbol is being replaced
- Transposition ciphers: permutes symbols in a block of symbols

Quiz!!!!!

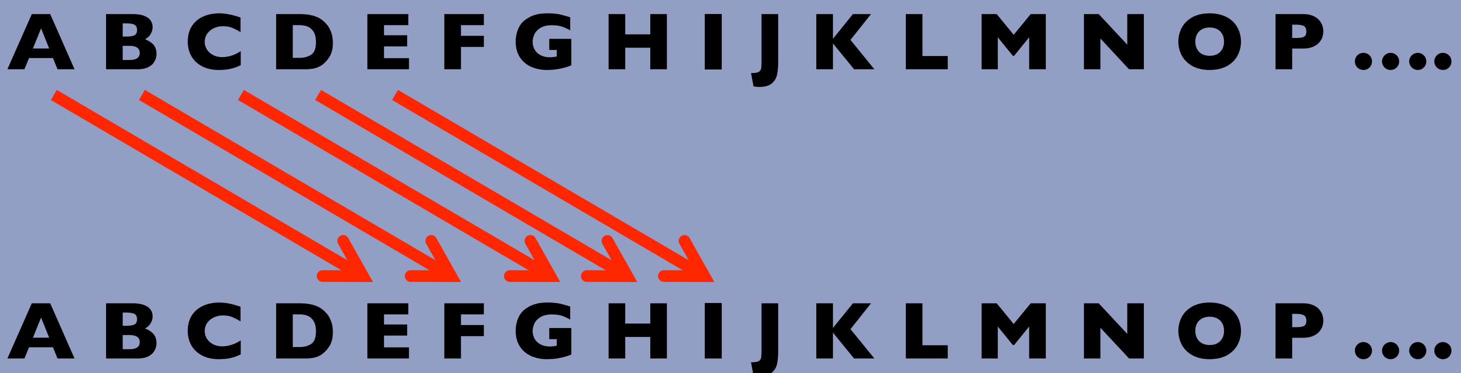
Plaintext: HELLO
Cyphertext: KHOOR
What type of cipher?

Plaintext: HELLO
Cyphertext: ABNZF
What type of cipher?

Plaintext: HELLO
Cyphertext: LOHEL
What type of cipher?

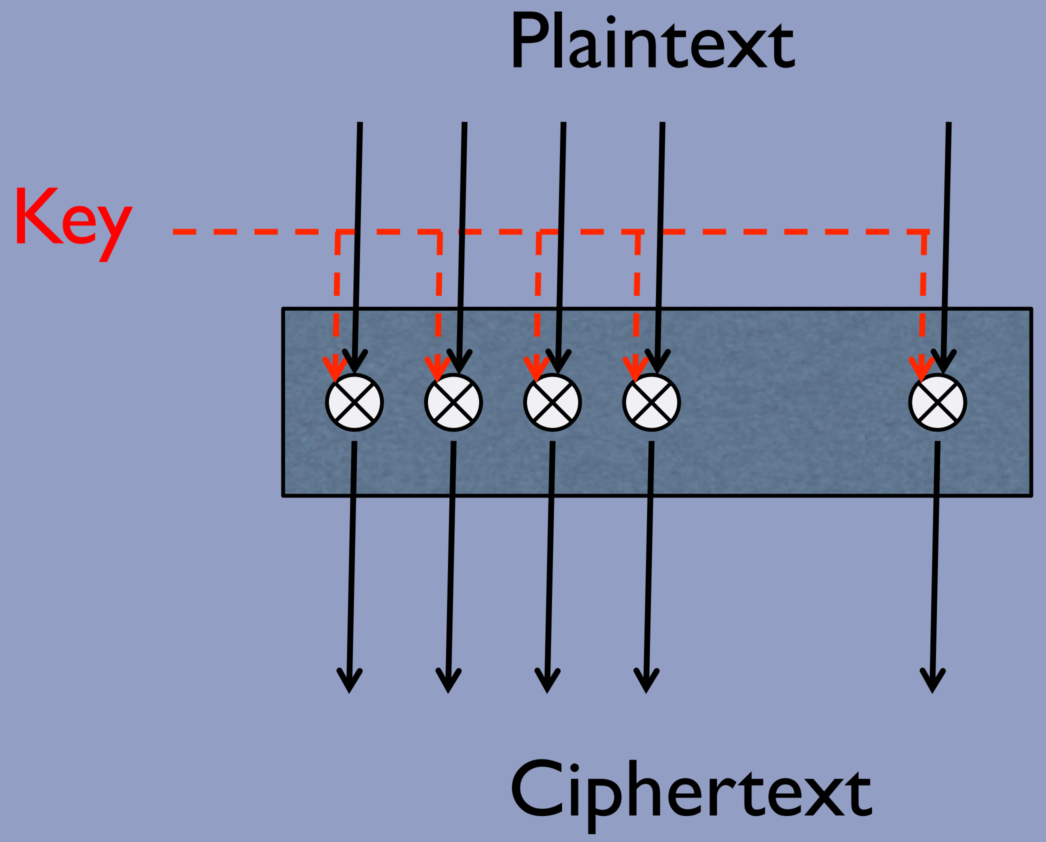
Sample Ciphers

Shift cipher (Ceasar cipher):


A B C D E F G H I J K L M N O P ....
A B C D E F G H I J K L M N O P ....

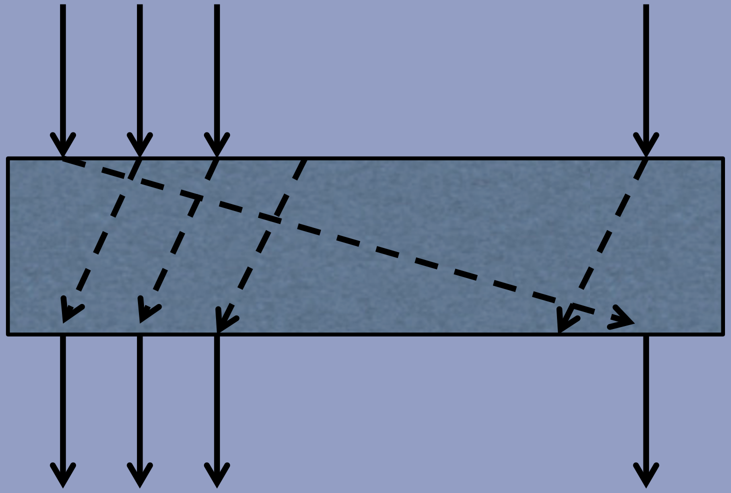
Key is 4, four characters down.(Ceasar used 3.)

XOR Cipher



Rotation Cipher

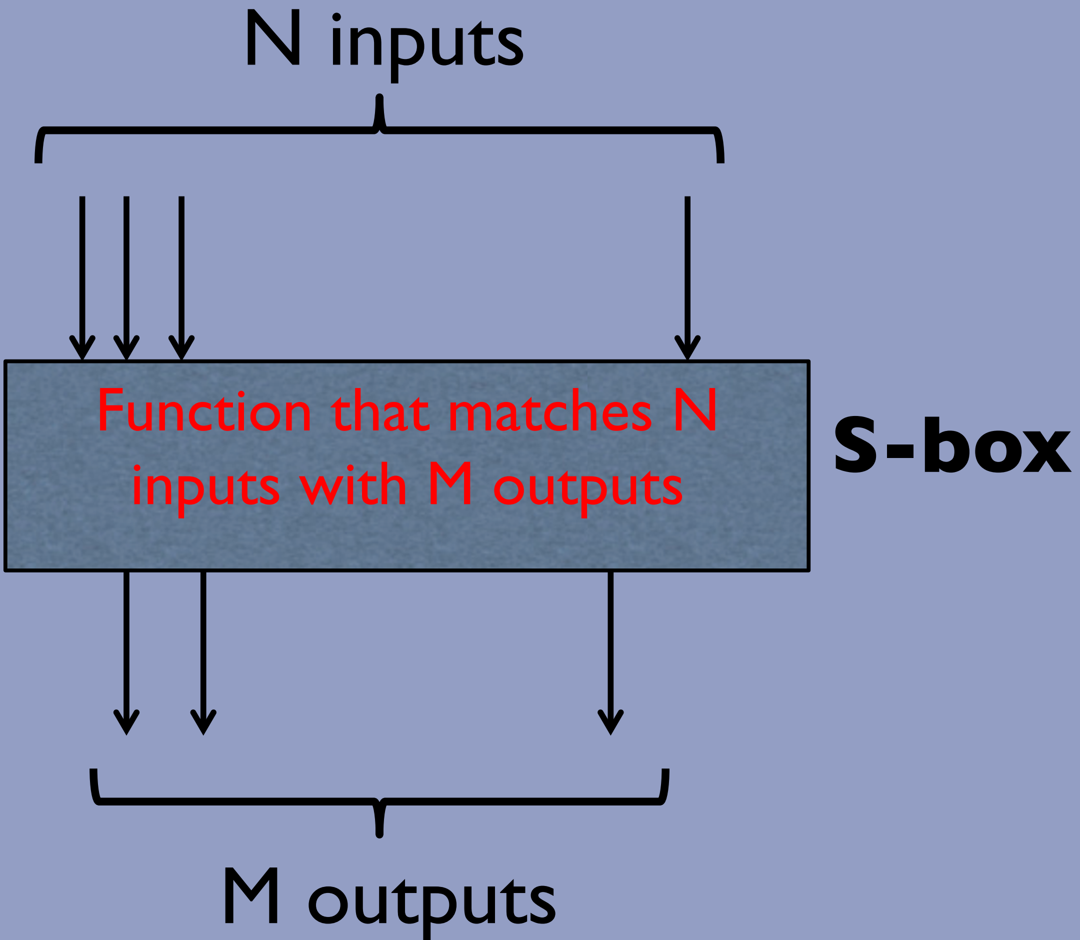
Plaintext



Ciphertext

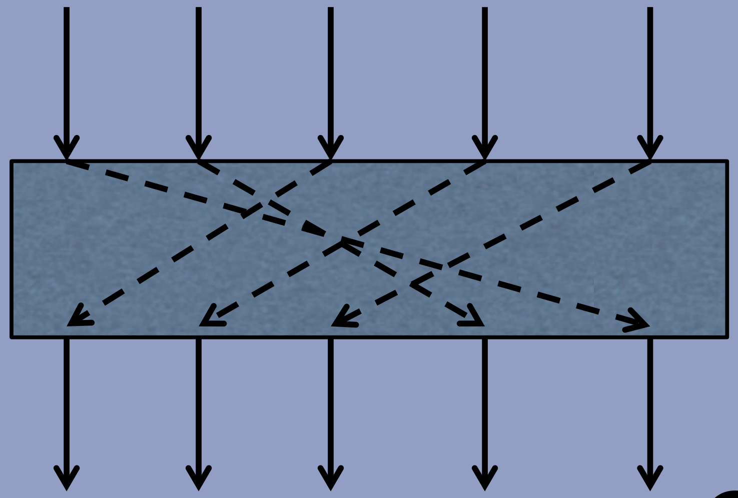
Key = number of rotations

Substition Cipher (S-box)



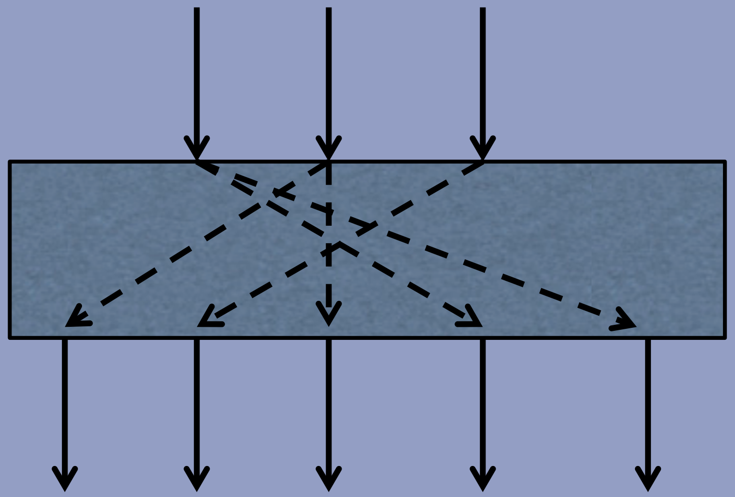
Key-less

Transposition Cipher (P-box)

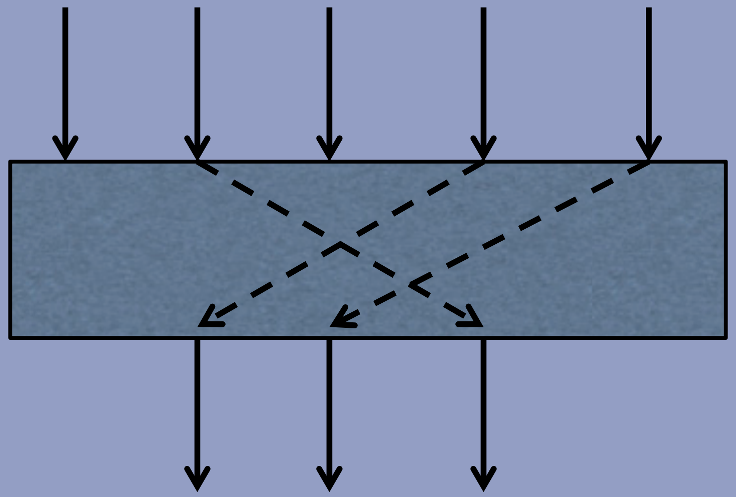


Straight P-box

Expansion P-box



Compression P-box



Modern Round Cipher: DES

Data Encryption Standard:

- ➢ First initial permutation (IP)
- ➢ Bits are split up into 2 groups of 32 bits
- ➢ First group is XOR-ed with F-function of the last 32 bits
- ➢ Initial last 32 bits are XOR-ed with F-function of result bits
- ➢ Etc. etc. 16 rounds
- ➢ Final permutation (FP)

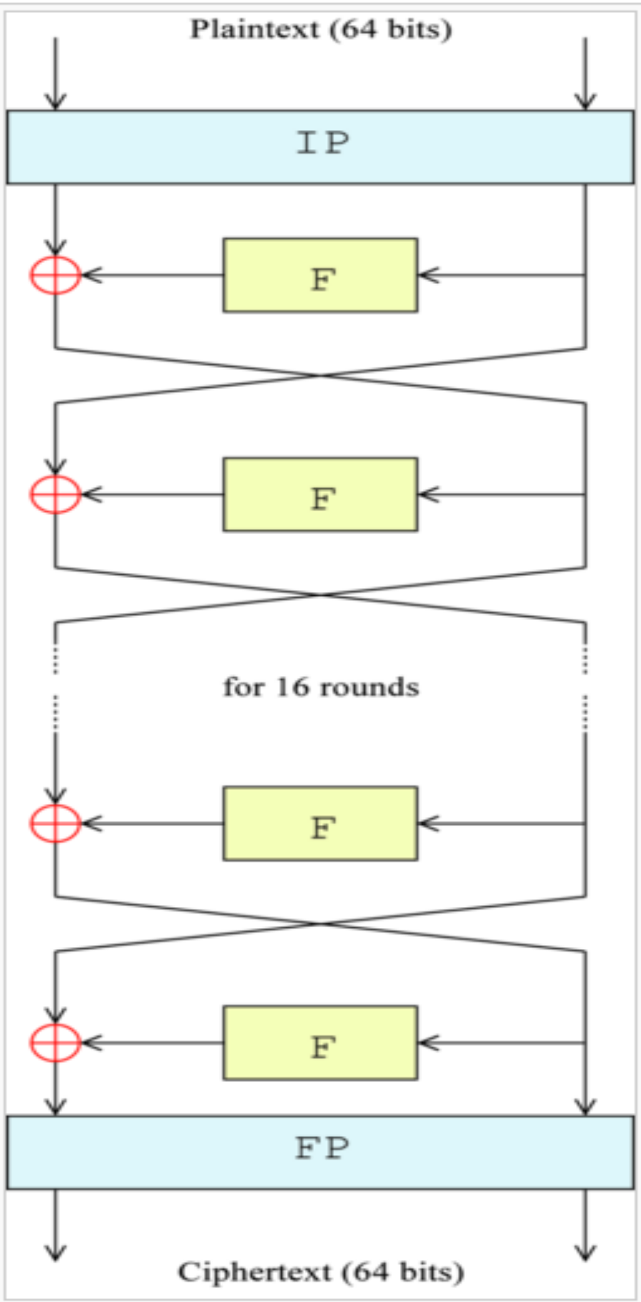


Figure 1— The overall Feistel structure of DES

DES: F-function

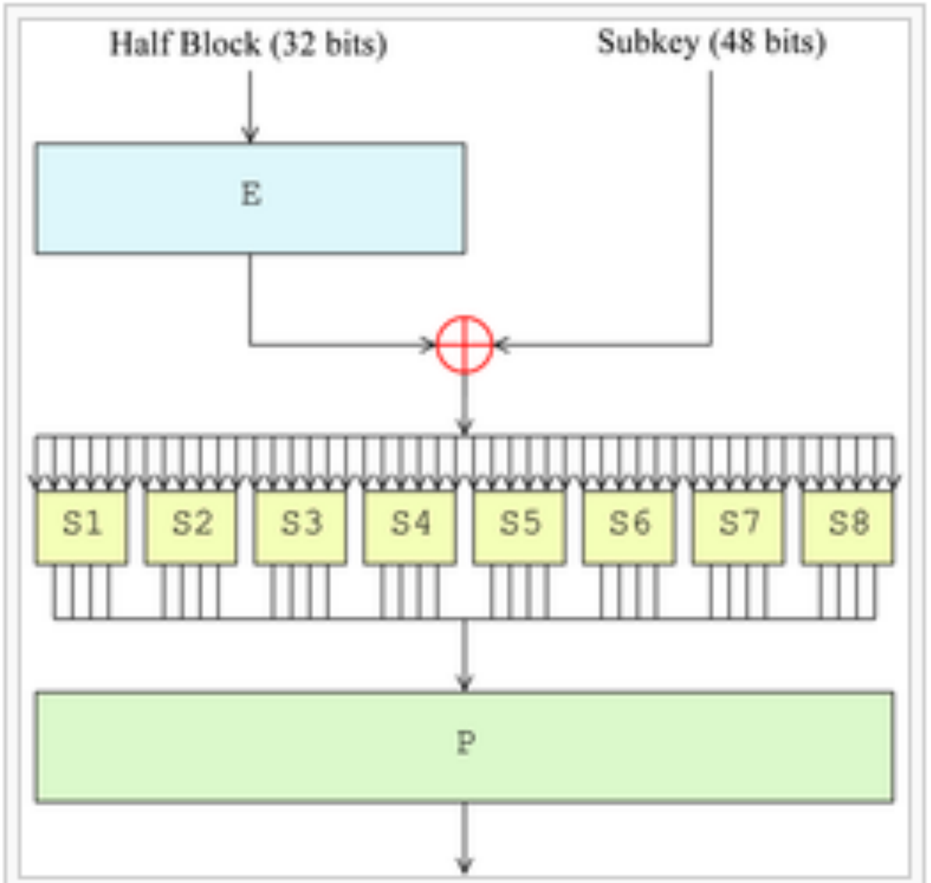


Figure 2—The Feistel function (F-function) of DES

- 32 bits are expanded to 48 bits(Expansion P-box)
- These bits are XOR-ed with 48 bits subkey and divided in 8 groups of 6
- Each group is substituted by 4 bits (subst. box)
- Final 32 bits are permuted by straight P-box

DES: key schedule

- Permuted Choice (PC1) selects 56 bits from original 64 bits
- 56 bits are divided into two halves of 28 bits
- Each half is rotated left by one or two bits
- Two halves are merged and PC2 selects 24 bits from each half of 28 bits
- This is being repeated 16 times.

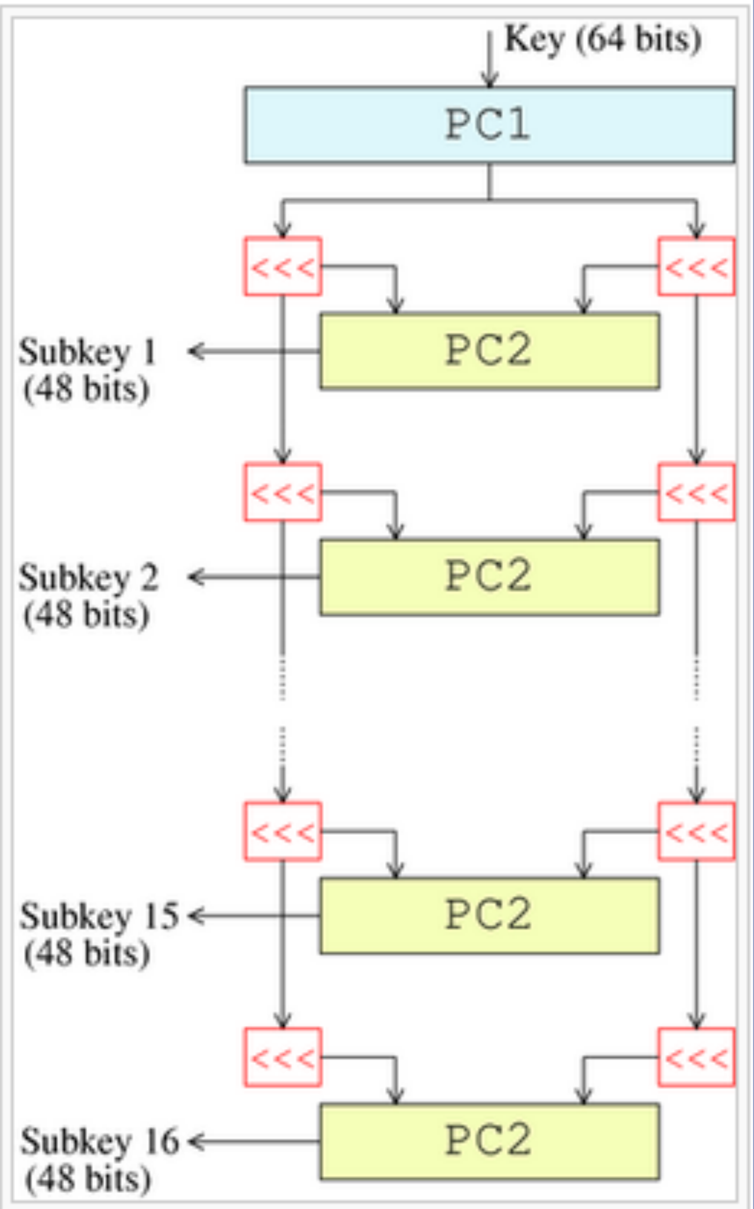


Figure 3— The key-schedule of DES

DES: Security

Breakable by brute force:

1977: Diffie and Hellman proposed a $20 Million machine to break the code in one day

1993: Wiener proposed a $1 Million machine to break the code within 7 hours

1998: A $250000 machine was build by the Electronic Frontier Foundation which could break DES in 2 days

2006: COPACOBANA was build for $10000, SciEngines GmbH

➔ Triple DES, apply DES three times with 2 different keys 2TDES or with 3 different keys 3TDES

➔ AES (Advanced Encryption Standard) a new cipher was issued by NIST in 2001

AES (Advanced Encryption Standard)

Op 2 januari 1997 het Amerikaans Nationaal Instituut voor Standaardisatie en Technologie (NIST) een wereldwijde wedstrijd om tot een nieuwe AES (Advanced Encryption Standard) te komen die de verouderde DES zou vervangen.

Verschillende grote kandidaten, zoals IBM en RSA Security stuurden hun algoritmen in. Op 2 oktober 2000 werd de winnaar bekendgemaakt: Rijndael van Vincent Rijmen en Joan Daemen uit Leuven. Hun algoritme is gekozen vanwege de combinatie van veiligheid, prestatie, efficiëntie, eenvoud en flexibiliteit.

In programma's zoals WinRAR, WinZip, PowerArchiver, e.d. wordt AES als encryptie aangeboden.

Source: Wikipedia https://nl.wikipedia.org/wiki/Advanced_Encryption_Standard

Rijndael representeert tekst en sleutel mbv matrices en werkt in een (variabel) aantal rondes afhankelijk van de key keuze en elke ronde bestaat uit een SubBytes stap (elke Bytes wordt vervangen door een andere Byte), een ShiftRow stap, een MixColumn stap en een AddKey stap.

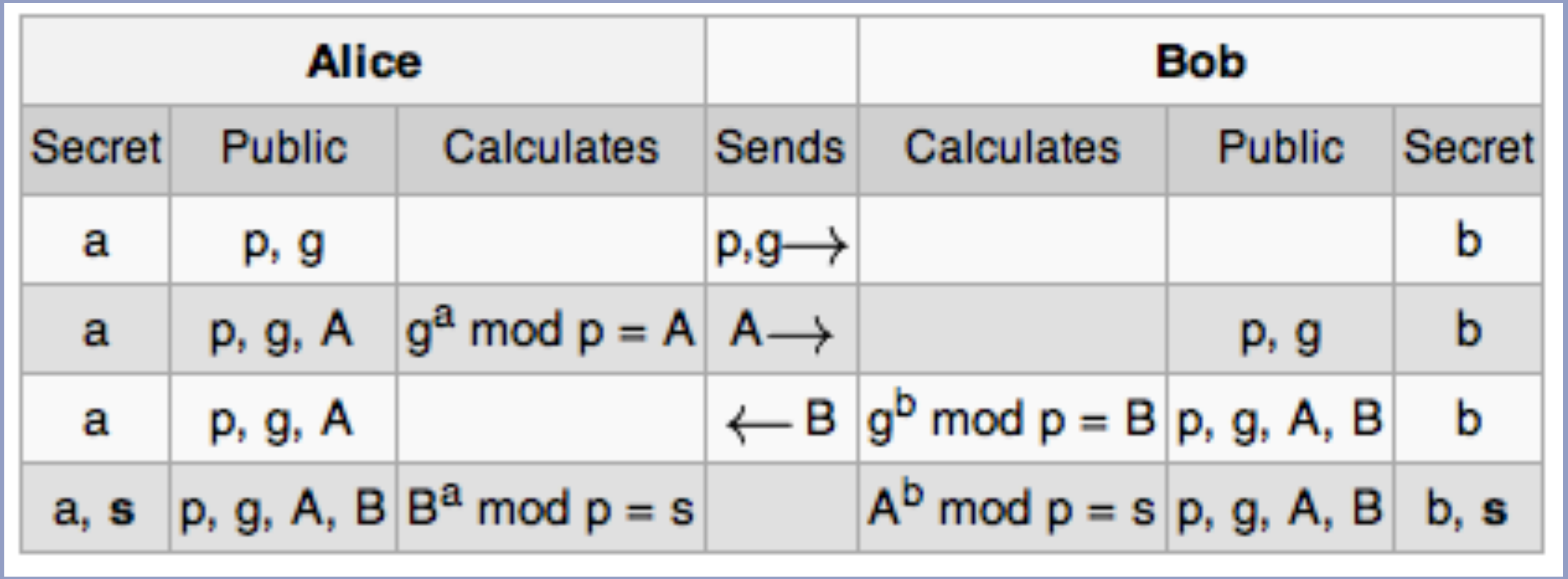
Het grote voordeel van Rijndael ten opzicht van DES is dat het in software efficiënt te implementeren is. In het DES-algoritme is het namelijk het geval dat in veel stappen bits verwisseld worden. Rijndael is gebaseerd op 32-bit woorden en een snelle implementatie kan derhalve verkregen worden via een software implementatie.

Op 17 augustus 2011 raakte bekend dat onderzoekers aan de Katholieke Universiteit Leuven in samenwerking met Microsoft en de Ecole Normale Supérieure in Parijs een zwak puntje in het algoritme gevonden hadden. Door dit te benutten kan het kraken van het algoritme vier keer sneller gebeuren, al duurt het nog altijd twee miljard jaar met duizend miljard computers die duizend keer sneller zijn dan de huidige generatie computer.

Key Sharing!!!!!!!

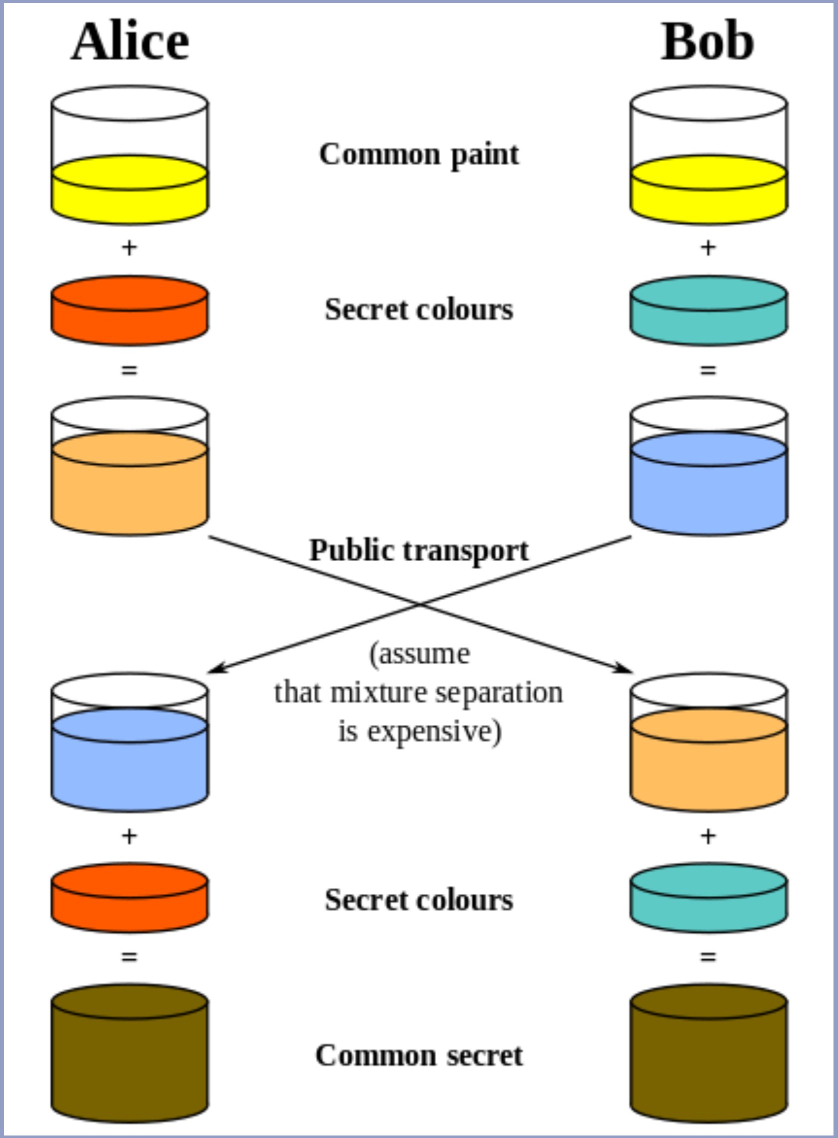
Diffie–Hellman(-Merkle) key exchange (D-H)

two parties that have no prior knowledge of each other jointly establish a shared secret key over an insecure communications channel.

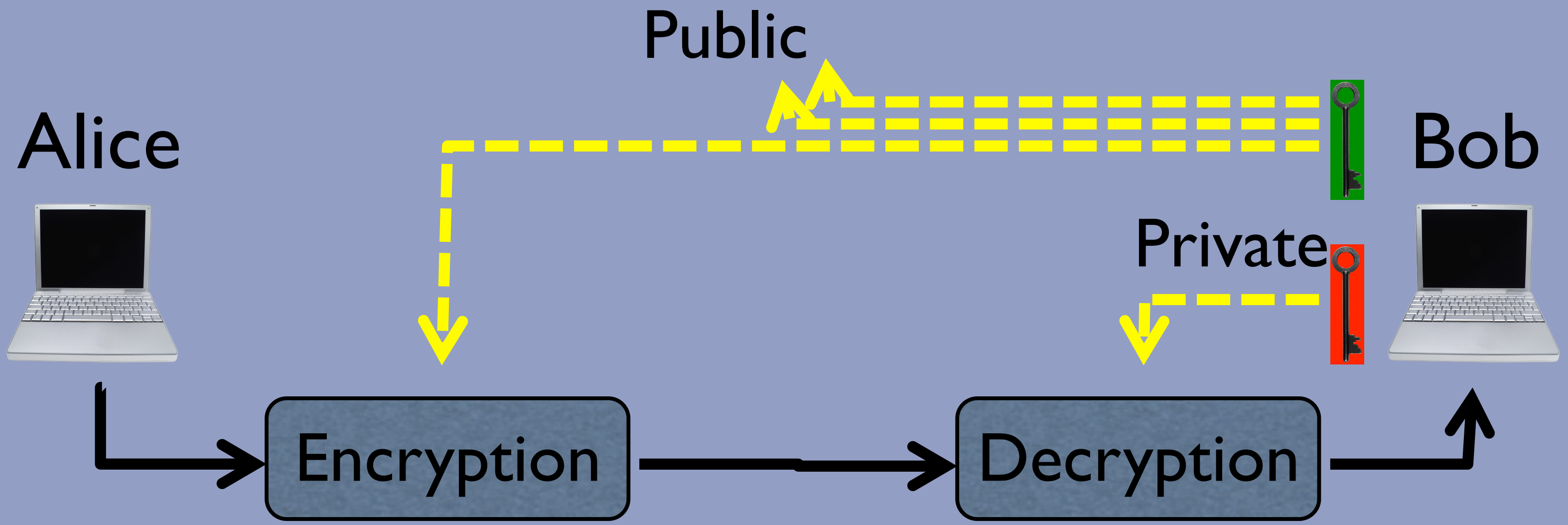


Alice				Bob		
Secret	Public	Calculates	Sends	Calculates	Public	Secret
a	p, g		p,g$\rightarrow$			b
a	p, g, A	g^a mod p = A	A$\rightarrow$		p, g	b
a	p, g, A		$\leftarrow$B	g^b mod p = B	p, g, A, B	b
a, **s**	p, g, A, B	B^a mod p = s		A^b mod p = s	p, g, A, B	b, **s**

In COLOURS


Alice
Bob
Common paint
+
Secret colours
=
Public transport
(assume
that mixture separation
is expensive)
+
Secret colours
=
Common secret

Asymmetric-key cryptography



RSA (Rivest, Shamir and Adleman)

A little history (*The Code Book*, by Simon Singh, Doubleday, 1999; pp. 279-92.)

According to the British Government, public-key cryptography was originally invented at the Government Communications Headquarters (GCHQ) in Cheltenham, the top-secret establishment that was formed from the remnants of Bletchley Park after the Second World War.

Looking ahead to the 1970s, senior military officials imagined a scenario in which miniaturization of radios and a reduction in cost meant that every soldier could be in continual radio contact with his officer. The advantages of widespread communication would be enormous, but communications would have to be encrypted, and the problem of distributing keys would be insurmountable.

At the beginning of 1969, the military asked James Ellis, one of Britain's foremost government cryptographers, to look into ways of coping with the key-distribution problem.

MEMORANDUM of Ellis: “Can we produce a secure encrypted message, readable by the authorized recipient without any prior secret exchange of the key? This question actually occurred to me in bed one night, and the proof of the theoretical possibility took only a few minutes. We had an existence theorem. The unthinkable was actually possible.”

Ellis's ideas were very similar to those of Diffie, Hellman and Merkle, except that he was several years ahead of them. However, nobody knew of Ellis's work because he was an employee of the British Government and therefore sworn to secrecy. By the end of 1969, Ellis appears to have reached the same impasse that the Stanford trio would reach in 1975. He had proved to himself that public-key cryptography (or non-secret encryption, as he called it) was possible.

Then, in September 1973, a new mathematician joined the team. Clifford Cocks had recently graduated from Cambridge University, where he had specialized in number theory. Cocks was beginning to formulate what would be known as the RSA asymmetric cipher. Rivest, Shamir and Adleman discovered their formula for public-key cryptography in 1977, but four years earlier the young Cambridge graduate was going through exactly the same thought processes. Cocks recalls: 'From start to finish, it took me no more than half an hour. I was quite pleased with myself. I thought, "Ooh, that's nice. I've been given a problem, and I've solved it." '

Some basics

- ➔ If gcd $(a, b) = 1$: $\{k.a \ (mod\ b) \mid k \geq 0\} = \{0, 1, 2, ..., b - 1\}$
- ➔ If gcd $(a, b) = 1$: $\{a^k \ (mod\ b) \mid k \geq 0\} \neq \{0, 1, 2, ..., b - 1\}$

  $7^k \ (mod\ 9) = \{1, 3, 4, 5, 7\}$
- ➔ If gcd $(a, b) = 1$: #divisors of ab = #divisors of a * #divisors of b
- ➔ For any a: $k.a + r \ (mod\ a) = r \ (mod\ a)$ *for all* k, r

Totient Function ϕ

Euler's totient function: $\phi(n)$ is an arithmetic function that counts the number of positive integers less than or equal to n that are relatively prime to n. That is, if n is a positive integer, then $\phi(n)$ is the number of integers k in the range $1 \leq k \leq n$ for which $\gcd(n, k) = 1$.

For example let $n = 9$. Then $\gcd(9, 3) = \gcd(9, 6) = 3$ and $\gcd(9, 9) = 9$. The other six numbers in the range $1 \leq k \leq 9$, that is, 1, 2, 4, 5, 7 and 8, are relatively prime to 9. Therefore, $\phi(9) = 6$.

➔ $\phi(nm) = \phi(n)\phi(m)$, if $\gcd(n, m) = 1$

➔ $\phi(p) = p - 1$, *if p is prime*

Theorem 1.3. *The Euler phi function is multiplicative.*

Proof. Let n and m be relatively prime integer. The statement clearly holds if $n = 1$ or $m = 1$. So let us assume that $n, m > 1$. We would like to calculate $\phi(nm)$ and so below we arrange the integers from 1 to nm in m columns of n integers.

$$\begin{array}{cccccc} 1 & 2 & \cdots & r & \cdots & m \\ m+1 & m+2 & \cdots & m+r & \cdots & 2m \\ 2m+1 & 2m+2 & \cdots & 2m+r & \cdots & 3m \\ \vdots & \vdots & & \vdots & & \vdots \\ (n-1)m+1 & (n-1)m+2 & \cdots & (n-1)m+r & \cdots & nm \end{array}$$

So to calculate $\phi(nm)$ we need to determine how many elements of this array are relatively prime with nm, which are the elements that are relatively prime to both n and m. So what was the point of us arranging the integers in such an array. We notice that since gcd$(km + r, m)$ =gcd(r, m) we see that an entry in the r^{th} column is relatively prime to m if and only if r is relatively prime to m, and in this case then all of the entries of the column are relatively prime to m. So looking at it this way, there are $\phi(m)$ columns with r's that are relatively prime to n, and so we need to show that in each column there are $\phi(n)$ entries relatively prime to n and then we will be done.

So let us choose such a column, and let r be the corresponding element of the column (mod m). So gcd(r, m)=1. The entries of this column are

$$r, m + r, 2m + r, ..., (n - 1)m + r.$$

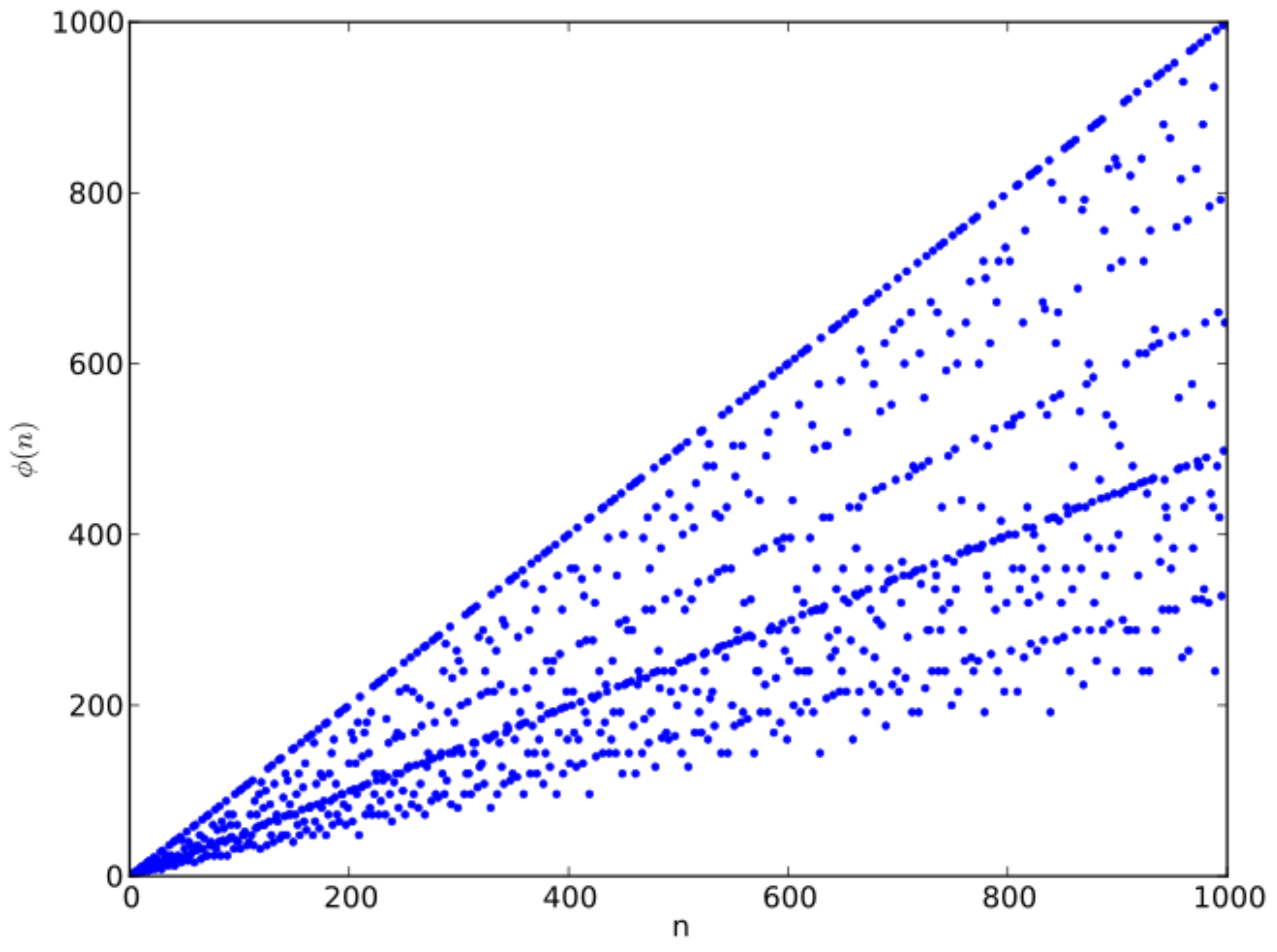
So we see that there are n integers in this column, so we would like to consider their equivalence class module n.

Now if $[km + r]_n = [lm + r]_n \Rightarrow [km]_n = [lm]_n \Rightarrow [k]_n = [l]_n$ since gcd(n, m)=1. However, as we can see no two of the coeffiecients of m in the column are equivalent mod n. Thus if we look at the column there are all of the equivalence classes modulo n. Therefore the number of them that are relatively prime to n is $\phi(n)$.

So we have divided the numbers that are relatively prime to nm into $\phi(m)$ columns where in each column with $\phi(n)$ such numbers in each column. Thus the total amount of such numbers is $\phi(n)\phi(m)$

$\square$

The first 1000 values of $\phi(n)$



The Algorithm

Choose two distinct prime numbers p and q. For security purposes, the integers p and q should be chosen at random, and should be of similar bit-length.

Compute $n = pq$. n is used as the modulus for both public and private keys, its length, usually expressed in bits, is the key length.

Compute $\phi(n) = \phi(p)\phi(q) = (p - 1)(q - 1)$, where ϕ is Euler's totient function.

Choose an integer e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n))=1$, i.e. e and $\phi(n)$ are coprime.

e is released as the public key exponent. e having a short bit-length i.e. $2^{16} + 1 = 65537$ results in more efficient encryption, but e should not be too small

Solve for d given $de \equiv 1 \pmod{\phi(n)}$. d is kept as the private key exponent.

The **public key** consists of the modulus n and the public (or encryption) exponent e.
The **private key** consists of the modulus n and the private (or decryption) exponent d, which must be kept secret.
p, q, and $\phi(n)$ must also be kept secret because they can be used to calculate d.

Encryption

Alice transmits her public key (n, e) to Bob and keeps the private key secret. Bob then wishes to send message M to Alice.

He first turns M into an integer m, such that $0 \leq m < n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext c corresponding to

$$c = m^e \pmod{n}$$

Decryption

Alice can recover m from c by using her private key exponent d via computing

$$m = c^d \pmod{n}$$

Given m, she can recover the original message M by reversing the padding scheme.

Why does this work???

Lemma

For any prime p:

$$(x+y)^p = x^p + y^p \pmod{p}$$

Proof

$(x+y)^p = \Sigma_{i=0, p} \binom{p}{i} x^{p-i} y^i$, with $\binom{p}{i} = p! / ((p-i)!\, i!)$.
The binomial coefficients are all integers and when $0 < i < p$, neither of the terms in the denominator includes a factor p, leaving the coefficient itself to possess a prime factor of p which must exist in the numerator, implying that $\binom{p}{i} = 0 \pmod{p}$. So the only remainder coefficients are $i = 0$ and $i = p$. ∎

Why does this work???

Fermat's Little Theorem

If p is prime, then for all integer a:

$$a^p = a \pmod p$$

Proof (by induction)

Assume $k^p = k \pmod p$, and consider $(k+1)^p$. By the lemma we have $(k+1)^p = k^p + 1^p \pmod p$. Using the induction hypothesis, we have that $k^p \equiv k \pmod p$, and, trivially, $1^p = 1$. Thus

$$(k+1)^p = k + 1 \pmod p$$

which is the statement of the theorem for $a = k+1$. ∎

Note $a^p = a \pmod p$ is equivalent to $a^{p-1} = 1 \pmod p$ if to $a \neq 0 \pmod p$

Why does this work???

Proof using Fermat's little theorem

The proof of the correctness of RSA is based on Fermat's little theorem. This theorem states that if *p* is prime and *p* does not divide an integer *a* then

$$a^{(p-1)} \equiv 1 \pmod{p}.$$

We want to show that $(m^e)^d \equiv m \pmod{pq}$ for every integer *m* when *p* and *q* are distinct prime numbers and e and *d* are positive integers satisfying

$$ed \equiv 1 \pmod{(p-1)(q-1)}.$$

We can write

$$ed - 1 = h(p-1)(q-1).$$

for some nonnegative integer *h*.

To check two numbers, like m^{ed} and *m*, are congruent mod *pq* it suffices (and in fact is equivalent) to check they are congruent mod *p* and mod *q* separately. To show $m^{ed} \equiv m \pmod{p}$, we consider two cases: $m \equiv 0 \pmod{p}$ and *m is not equivalent to* 0 (mod *p*). In the first case m^{ed} is a multiple of *p*, so $m^{ed} \equiv 0 \equiv m \pmod{p}$. In the second case

$$m^{ed} = m^{(ed-1)}m = m^{h(p-1)(q-1)}m = \left(m^{p-1}\right)^{h(q-1)}m \equiv 1^{h(q-1)}m \equiv m \pmod{p},$$

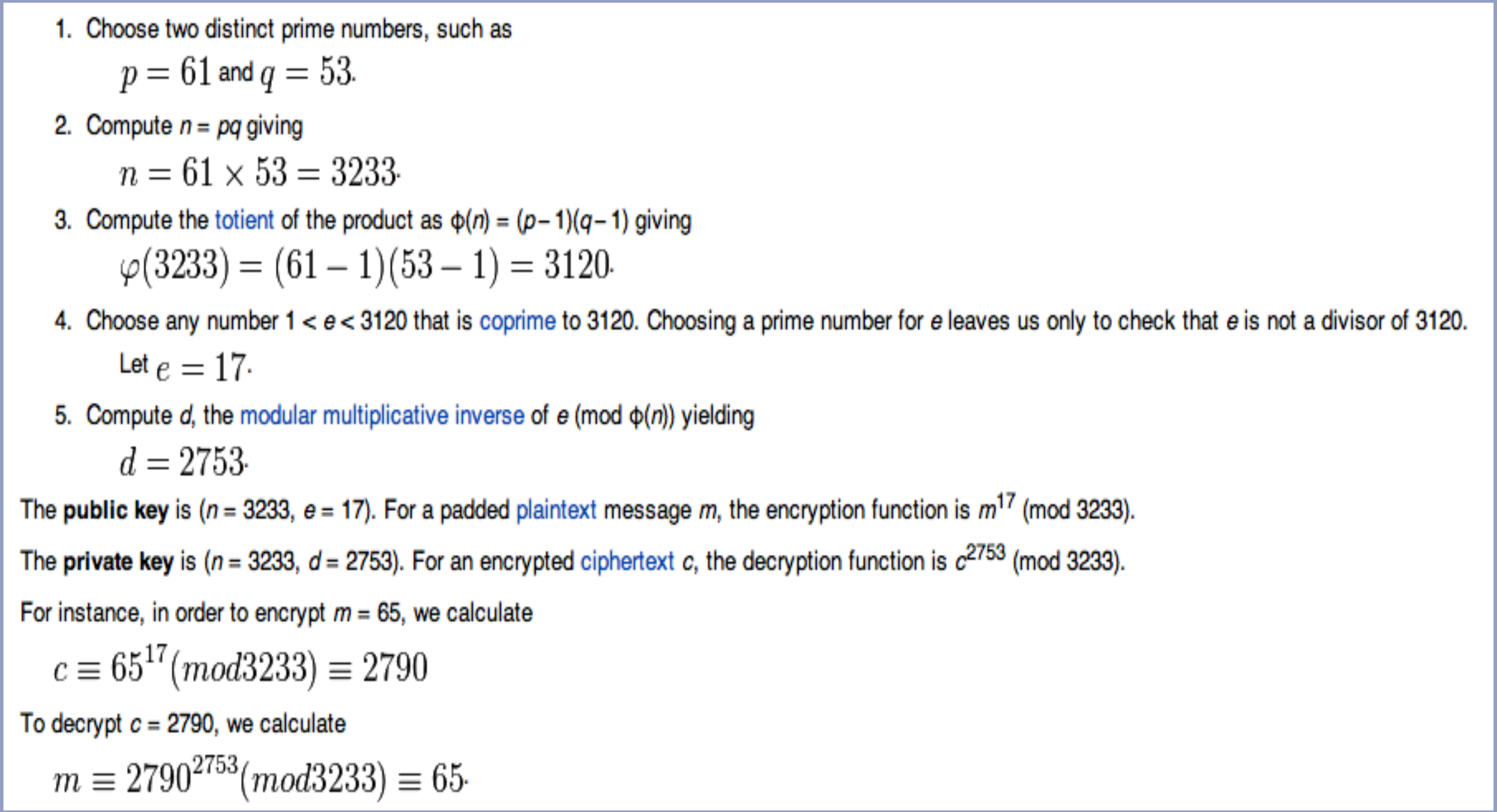
where we used Fermat's little theorem to replace m^{p-1} mod *p* with 1.

The verification that $m^{ed} \equiv m \pmod{q}$ proceeds in a similar way, treating separately the cases $m \equiv 0 \pmod{q}$ and *m is not equivalent to* 0 (mod *q*), using Fermat's little theorem for modulus *q* in the second case.

This completes the proof that, for any integer *m*,

$$\left(m^e\right)^d \equiv m \pmod{pq}. \quad \blacksquare$$

A simple Example



1. Choose two distinct prime numbers, such as

   $p = 61$ and $q = 53$.

2. Compute $n = pq$ giving

   $n = 61 \times 53 = 3233$.

3. Compute the totient of the product as $\phi(n) = (p-1)(q-1)$ giving

   $\varphi(3233) = (61 - 1)(53 - 1) = 3120$.

4. Choose any number $1 < e < 3120$ that is coprime to 3120. Choosing a prime number for e leaves us only to check that e is not a divisor of 3120.

   Let $e = 17$.

5. Compute d, the modular multiplicative inverse of e (mod $\phi(n)$) yielding

   $d = 2753$.

The **public key** is ($n = 3233$, $e = 17$). For a padded plaintext message m, the encryption function is m^{17} (mod 3233).

The **private key** is ($n = 3233$, $d = 2753$). For an encrypted ciphertext c, the decryption function is c^{2753} (mod 3233).

For instance, in order to encrypt $m = 65$, we calculate

$$c \equiv 65^{17} (mod 3233) \equiv 2790$$

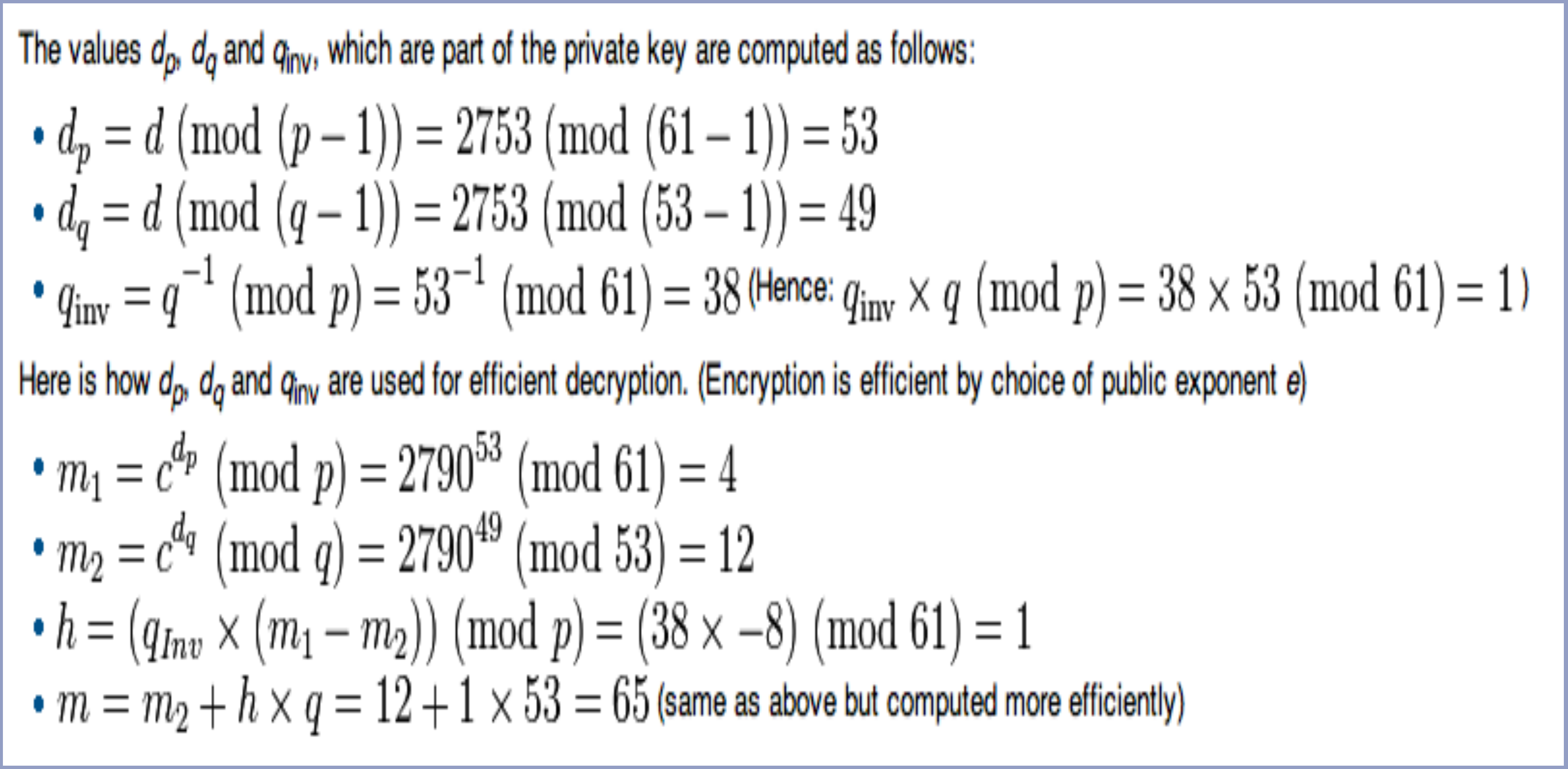
To decrypt $c = 2790$, we calculate

$$m \equiv 2790^{2753} (mod 3233) \equiv 65.$$

Illustration

https://www.youtube.com/watch?v=tXXnHXslVhw

Efficient decrypting



The values d_p, d_q and q_{inv}, which are part of the private key are computed as follows:

- $d_p = d \pmod{(p-1)} = 2753 \pmod{(61-1)} = 53$
- $d_q = d \pmod{(q-1)} = 2753 \pmod{(53-1)} = 49$
- $q_{\text{inv}} = q^{-1} \pmod{p} = 53^{-1} \pmod{61} = 38$ (Hence: $q_{\text{inv}} \times q \pmod{p} = 38 \times 53 \pmod{61} = 1$)

Here is how d_p, d_q and q_{inv} are used for efficient decryption. (Encryption is efficient by choice of public exponent e)

- $m_1 = c^{d_p} \pmod{p} = 2790^{53} \pmod{61} = 4$
- $m_2 = c^{d_q} \pmod{q} = 2790^{49} \pmod{53} = 12$
- $h = (q_{Inv} \times (m_1 - m_2)) \pmod{p} = (38 \times -8) \pmod{61} = 1$
- $m = m_2 + h \times q = 12 + 1 \times 53 = 65$ (same as above but computed more efficiently)

Based On Chinese Remainder Theorem:

$c^d = c^{d(mod(q-1))}(\text{mod } q) + q(q_{inv} \times (c^{d(mod(p-1))} - c^{d(mod(q-1))})) \ (\text{mod } p) \ (\text{mod } pq)$

Security Considerations

If *n* is 300 bits or shorter, it can be factored in a few hours on a personal computer, using software already freely available. Keys of 512 bits have been shown to be practically breakable in 1999 when RSA-155 was factored by using several hundred computers and are now factored in a few weeks using common hardware.

Exploits using 512-bit code-signing certificates that may have been factored were reported in 2011. A theoretical hardware device named TWIRL and described by Shamir and Tromer in 2003 called into question the security of 1024 bit keys. It is currently recommended that *n* be at least 2048 bits long

Example RSA generation

```
Harrys-MacBook-Pro:~ harryw$ openssl genrsa -out private_key.pem 1024
Generating RSA private key, 1024 bit long modulus
....................................++++++
..................................................................++++++
e is 65537 (0x10001)
Harrys-MacBook-Pro:~ harryw$ openssl rsa -pubout -in private_key.pem -out
public_key.pem
writing RSA key
Harrys-MacBook-Pro:~ harryw$ openssl rsa -text -in private_key.pem
Private-Key: (1024 bit)
modulus:
    00:de:c0:ef:f7:ed:10:6a:4f:1f:58:80:1f:4b:67:
    d8:9d:64:71:01:21:d4:89:d1:3e:56:8e:e5:85:36:
    1d:e7:6f:67:14:4e:fe:f9:35:64:ef:ab:32:01:e2:
    63:ec:88:13:68:94:dc:55:2b:5f:3f:a6:0f:7d:3b:
    3a:c8:fb:4b:92:d8:02:f0:80:72:cb:f5:2c:25:5b:
    6b:20:01:1a:94:96:23:aa:f2:d8:19:0f:86:c5:0e:
    da:02:4b:0f:31:6b:2a:0b:ef:8a:6e:a8:6d:8c:b7:
    b4:bd:8f:52:3c:8f:0a:eb:44:05:74:50:09:c6:13:
    8d:65:23:15:30:51:6c:82:23
publicExponent: 65537 (0x10001)
```

```
privateExponent:
    00:c4:a4:b0:73:3e:dd:51:ec:1d:70:e4:52:3c:20:
    25:b2:f4:5b:6a:33:72:4c:63:e2:d3:48:fc:c7:b7:
    79:78:b8:f8:d7:8d:d1:3b:30:ee:b5:41:7d:38:fa:
    a1:59:ca:da:cf:65:32:89:21:6b:c9:65:90:a0:ee:
    2b:bc:07:53:b3:5d:a9:4d:90:86:86:30:8d:48:a0:
    9d:0a:67:8b:75:3c:29:c6:f8:39:e4:bf:68:c9:24:
    66:aa:91:3d:19:d0:87:52:c1:7c:79:cd:67:a6:34:
    cb:70:e9:09:a3:10:1a:32:1d:f8:50:0e:8e:e8:f6:
    c0:b3:f2:70:a2:1a:b5:65:59
prime1:
    00:f5:07:fc:cd:d0:0b:a7:f5:62:36:13:9e:31:74:
    d9:a7:cf:bb:e1:4f:08:df:60:9f:13:7a:b9:ad:a4:
    ea:5c:09:0c:63:5e:bc:97:99:dc:7b:67:63:c0:2b:
    a1:34:06:84:9a:2d:68:fa:40:8c:a4:da:45:f2:14:
    a1:7e:0e:ea:af
prime2:
    00:e8:b9:a7:42:59:a8:83:64:e8:87:0a:27:f6:3b:
    94:32:8c:db:e9:cd:01:ca:ed:97:83:97:9b:97:17:
    ef:69:c7:c1:a9:90:60:a0:75:cb:72:4a:97:4c:9d:
    7a:eb:07:02:be:bc:76:cb:14:8a:bb:55:d2:17:94:
    2d:72:43:ac:cd
```

```
exponent1:
    61:66:6a:6c:59:6d:b8:b7:06:f2:1d:fc:3d:06:88:
    da:76:ed:e5:12:e8:a0:fa:a4:61:36:e0:86:10:cf:
    04:04:a8:c2:fb:4e:96:28:98:07:09:c3:12:09:85:
    cb:cb:67:7c:6d:de:93:d3:82:d4:a8:db:32:ee:56:
    7f:68:68:8b
exponent2:
    42:e5:0a:94:e1:dc:b4:58:0f:16:b1:ee:a6:b2:9d:
    78:a2:50:9c:35:d7:6c:13:3b:58:11:fe:21:42:3a:
    09:37:e8:0c:eb:79:3a:e6:61:22:6b:1a:6e:65:5d:
    ed:ac:c8:37:37:49:16:3a:c3:5d:f1:df:3f:f3:d1:
    d4:64:6b:89
coefficient:
    0e:30:15:15:74:5d:9b:ad:e4:7a:03:93:11:66:14:
    e6:49:a8:23:82:be:3f:1f:7a:1a:79:78:c3:f8:48:
    b2:8e:98:2e:f6:60:8c:be:54:34:51:c7:c9:41:3a:
    82:b2:1f:ef:83:5a:d8:03:aa:bc:27:24:f7:35:13:
    cd:d6:a9
```

```
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIICWwIBAAKBgQDewO/37RBqTx9YgB9LZ9idZHEBIdSJ0T5WjuWFNh3nb2cUTv75
NWTvqzIB4mPsiBNoINxVK18/pg99OzrI+0uS2ALwgHLL9SwIW2sgARqUIiOq8tgZ
D4bFDtoCSw8xayoL74puqG2Mt7S9j118jwrrRAV0UAnGE411IxUwUWyCIwIDAQAB
AoGBAMSksHM+3VHsHXDkUjwgJbL0W2ozckxj4tNI/Me3eXi4+NeN0Tsw7rVBfTj6
oVnK2s9IMokha8IIkKDuK7wHU7NdqU2QhoYwjUignQpni3U8Kcb4OeS/aMkkZqqR
PRnQh1LBfHnNZ6Y0y3DpCaMQGjId+FAOjuj2wLPycKIatWVZAkEA9Qf8zdALp/Vi
NhOeMXTZp8+74U8I32CfE3q5raTqXAkMY168I5nce2djwCuhNAaEmi1o+kCMpNpF
8hShfg7qrwJBAOi5p0JZqINk6IcKJ/Y7IDKM2+nNAcrtI4OXm5cX72nHwamQYKB1
y3JKI0ydeusHAr68dssUirtV0heULXJDrM0CQGFmamxZbbi3BvId/D0GiNp27eUS
6KD6pGE24IYQzwQEqML7TpYomAcJwxIJhcvLZ3xt3pPTgtSo2zLuVn9oaIsCQELI
CpTh3LRYDxax7qaynXiiUJw112wTO1gR/iFCOgk36AzreTrmYSJrGm5IXe2syDc3
SRY6w13x3z/z0dRka4kCPw4wFRV0XZut5HoDkxFmFOZJqCOCvj8fehp5eMP4SLKO
mC72YIy+VDRRx8IBOoKyH++DWtgDqrwnJPc1E83WqQ==
-----END RSA PRIVATE KEY-----
Harrys-MacBook-Pro:~ harryw$
```

openssl rsa -text -in private_key.pem basically results in:

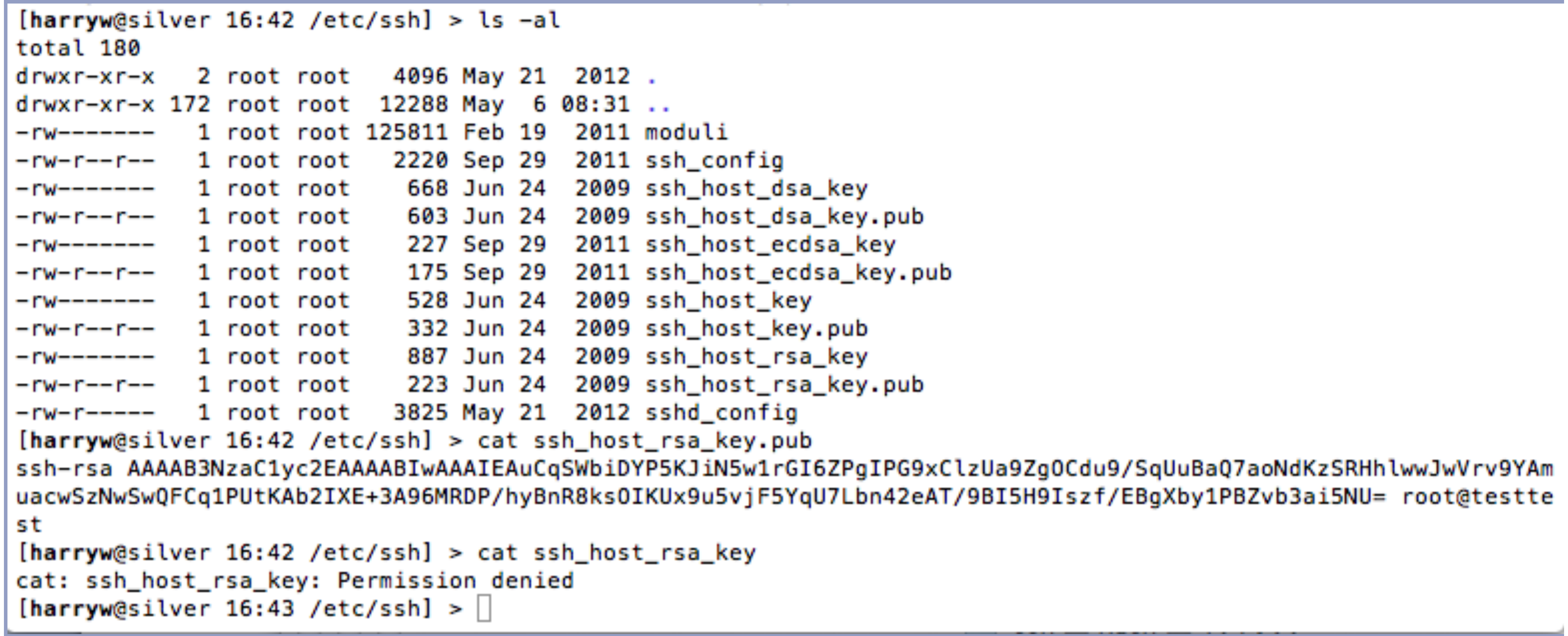
All parts of private_key.pem are printed to the screen. This includes the modulus (also referred to as public key and n), public exponent (also referred to as e and exponent; default value is 0x010001), private exponent, and primes used to create keys (prime1, also called p, and prime2, also called q), a few other variables used to perform RSA operations faster, and the Base64 PEM encoded version of all that data. (The Base64 PEM encoded version of all that data is identical to the private_key.pem file).

Base64 encoding

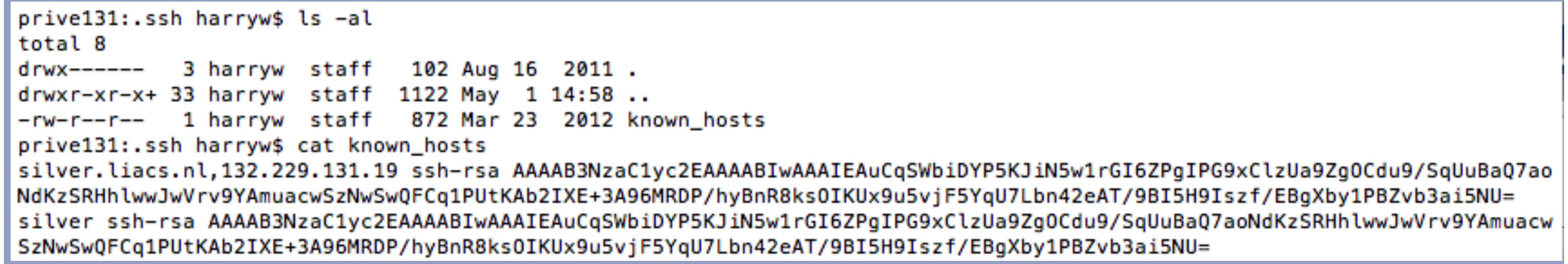
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

- modulus = prime1 x prime2
- publicExponent x exponent1 = 1 mod (prime1 - 1)
- publicExponent x exponent2 = 1 mod (prime2 – 1)
- prime2 x coeeficient = 1 mod (prime1)
- publicExponent x privateExponent = 1 mod (prime1 – 1)(prime2 – 1)
- So, privateExponent = exponent1 mod (prime1 – 1)
- And privateExponent = exponent2 mod (prime2 – 1)

SSH at LIACS (via ssh-keygen)



```
[harryw@silver 16:42 /etc/ssh] > ls -al
total 180
drwxr-xr-x   2 root root   4096 May 21  2012 .
drwxr-xr-x 172 root root  12288 May  6 08:31 ..
-rw-------   1 root root 125811 Feb 19  2011 moduli
-rw-r--r--   1 root root   2220 Sep 29  2011 ssh_config
-rw-------   1 root root    668 Jun 24  2009 ssh_host_dsa_key
-rw-r--r--   1 root root    603 Jun 24  2009 ssh_host_dsa_key.pub
-rw-------   1 root root    227 Sep 29  2011 ssh_host_ecdsa_key
-rw-r--r--   1 root root    175 Sep 29  2011 ssh_host_ecdsa_key.pub
-rw-------   1 root root    528 Jun 24  2009 ssh_host_key
-rw-r--r--   1 root root    332 Jun 24  2009 ssh_host_key.pub
-rw-------   1 root root    887 Jun 24  2009 ssh_host_rsa_key
-rw-r--r--   1 root root    223 Jun 24  2009 ssh_host_rsa_key.pub
-rw-r-----   1 root root   3825 May 21  2012 sshd_config
[harryw@silver 16:42 /etc/ssh] > cat ssh_host_rsa_key.pub
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAuCqSWbiDYP5KJiN5w1rGI6ZPgIPG9xClzUa9Zg0Cdu9/SqUuBaQ7aoNdKzSRHhlwwJwVrv9YAm
uacwSzNwSwQFCq1PUtKAb2IXE+3A96MRDP/hyBnR8ksOIKUx9u5vjF5YqU7Lbn42eAT/9BI5H9Iszf/EBgXby1PBZvb3ai5NU= root@testte
st
[harryw@silver 16:42 /etc/ssh] > cat ssh_host_rsa_key
cat: ssh_host_rsa_key: Permission denied
[harryw@silver 16:43 /etc/ssh] > 
```

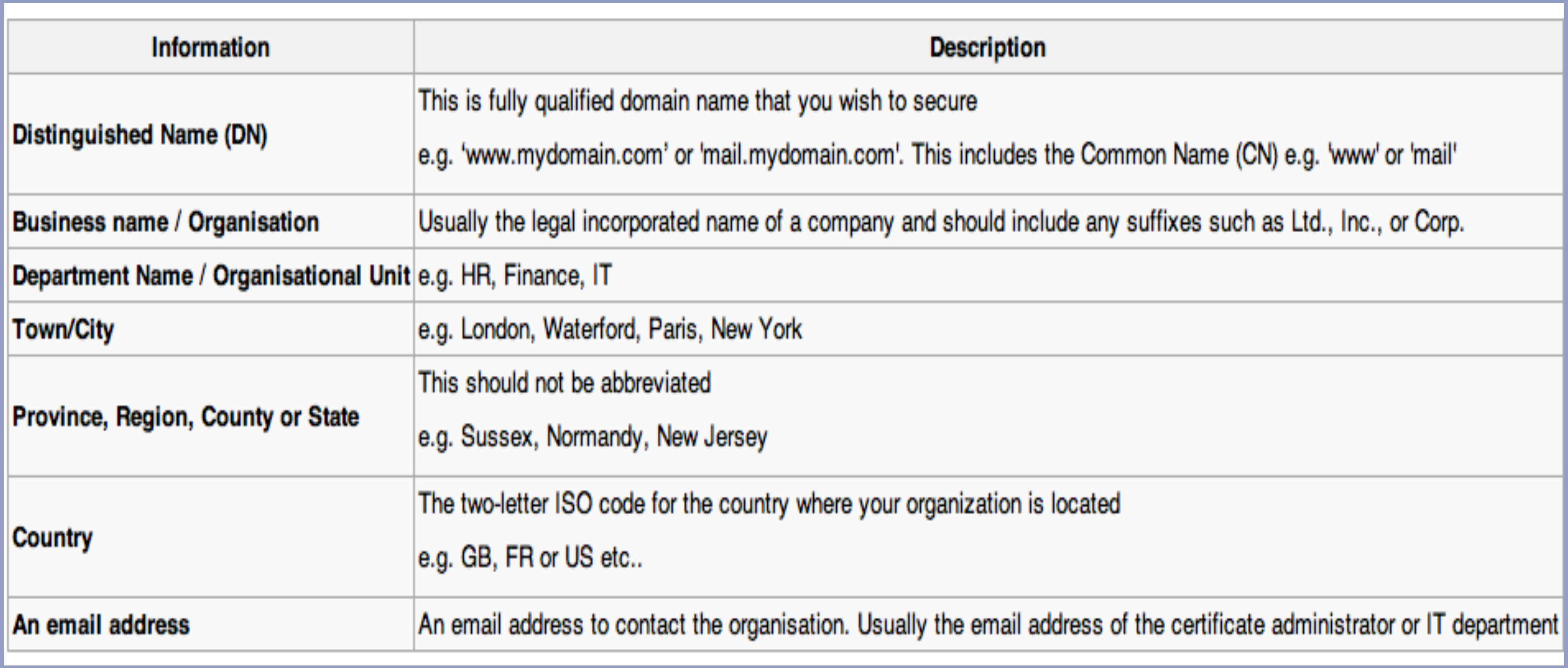


```
prive131:.ssh harryw$ ls -al
total 8
drwx------   3 harryw  staff   102 Aug 16  2011 .
drwxr-xr-x+ 33 harryw  staff  1122 May  1 14:58 ..
-rw-r--r--   1 harryw  staff   872 Mar 23  2012 known_hosts
prive131:.ssh harryw$ cat known_hosts
silver.liacs.nl,132.229.131.19 ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAuCqSWbiDYP5KJiN5w1rGI6ZPgIPG9xClzUa9Zg0Cdu9/SqUuBaQ7ao
NdKzSRHhlwwJwVrv9YAmuacwSzNwSwQFCq1PUtKAb2IXE+3A96MRDP/hyBnR8ksOIKUx9u5vjF5YqU7Lbn42eAT/9BI5H9Iszf/EBgXby1PBZvb3ai5NU=
silver ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAuCqSWbiDYP5KJiN5w1rGI6ZPgIPG9xClzUa9Zg0Cdu9/SqUuBaQ7aoNdKzSRHhlwwJwVrv9YAmuacw
SzNwSwQFCq1PUtKAb2IXE+3A96MRDP/hyBnR8ksOIKUx9u5vjF5YqU7Lbn42eAT/9BI5H9Iszf/EBgXby1PBZvb3ai5NU=
```

HTTP Secure

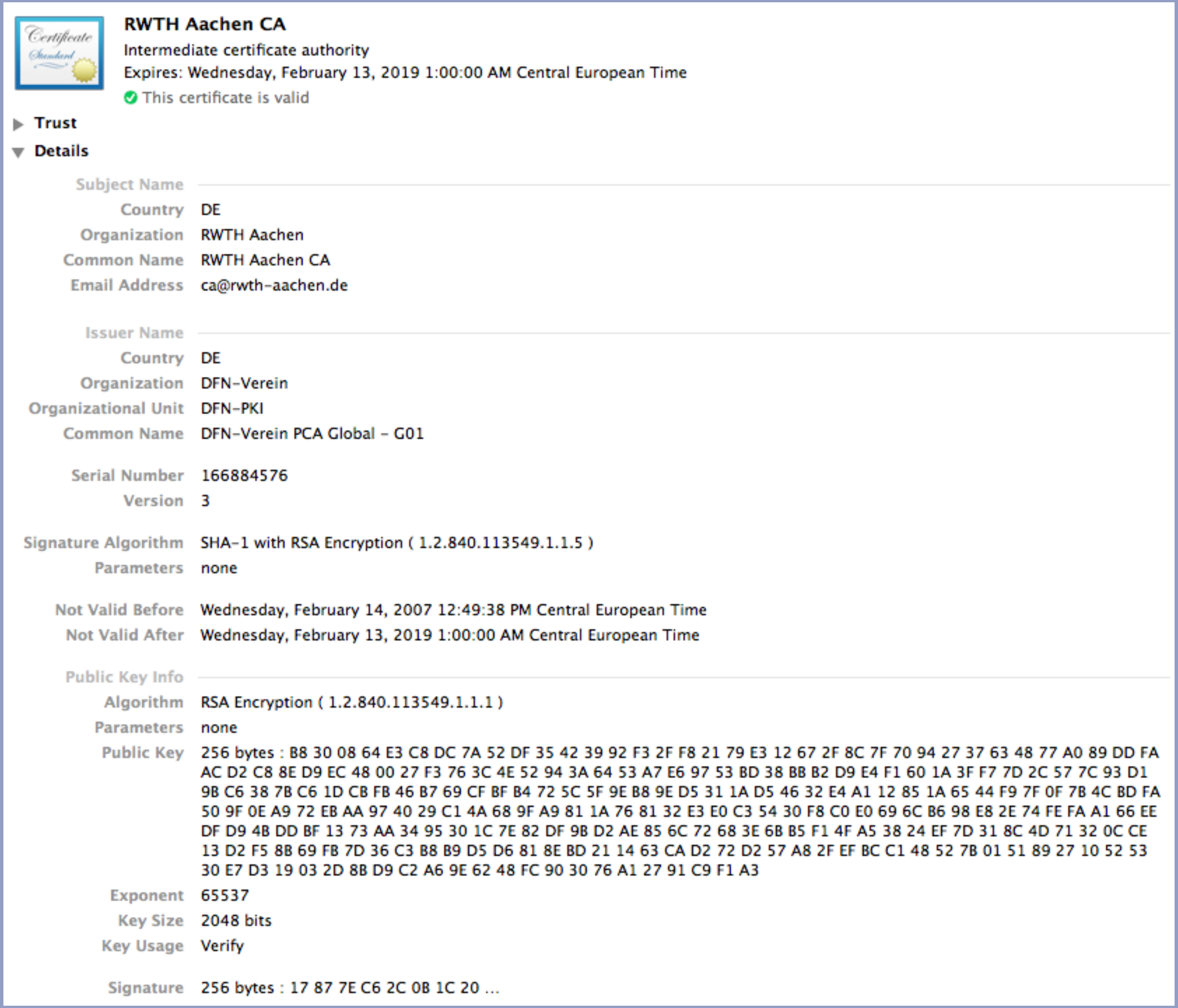
- HTTPS URLs begin with "https://" and use port 443 by default (HTTP URLs begin with "http://" and use port 80 by default)
- HTTPS is not a separate protocol, but refers to use of ordinary HTTP over an encrypted SSL/TLS connection.
- To prepare a web server to accept HTTPS connections, the administrator must create a public key certificate for the web server.
- This certificate must be signed by a trusted certificate authority
- This is done by sending a certificate signing request (CSR)
- Before doing so the server creates private/public key openSSL
- If the request is successful, the certificate authority will send back an identity certificate that has been digitally signed with the private key of the certificate authority.

Typical information required in a CSR



Information	Description
Distinguished Name (DN)	This is fully qualified domain name that you wish to secure e.g. 'www.mydomain.com' or 'mail.mydomain.com'. This includes the Common Name (CN) e.g. 'www' or 'mail'
Business name / Organisation	Usually the legal incorporated name of a company and should include any suffixes such as Ltd., Inc., or Corp.
Department Name / Organisational Unit	e.g. HR, Finance, IT
Town/City	e.g. London, Waterford, Paris, New York
Province, Region, County or State	This should not be abbreviated e.g. Sussex, Normandy, New Jersey
Country	The two-letter ISO code for the country where your organization is located e.g. GB, FR or US etc..
An email address	An email address to contact the organisation. Usually the email address of the certificate administrator or IT department

Sample Certificate



RWTH Aachen CA
Intermediate certificate authority
Expires: Wednesday, February 13, 2019 1:00:00 AM Central European Time
✓ This certificate is valid

▶ **Trust**

▼ **Details**

```
       Subject Name
            Country   DE
       Organization   RWTH Aachen
        Common Name   RWTH Aachen CA
      Email Address   ca@rwth-aachen.de

        Issuer Name
            Country   DE
       Organization   DFN-Verein
  Organizational Unit DFN-PKI
        Common Name   DFN-Verein PCA Global - G01

      Serial Number   166884576
            Version   3

Signature Algorithm   SHA-1 with RSA Encryption ( 1.2.840.113549.1.1.5 )
         Parameters   none

   Not Valid Before   Wednesday, February 14, 2007 12:49:38 PM Central European Time
    Not Valid After   Wednesday, February 13, 2019 1:00:00 AM Central European Time

    Public Key Info
          Algorithm   RSA Encryption ( 1.2.840.113549.1.1.1 )
         Parameters   none
         Public Key   256 bytes : B8 30 08 64 E3 C8 DC 7A 52 DF 35 42 39 92 F3 2F F8 21 79 E3 12 67 2F 8C 7F 70 94 27 37 63 48 77 A0 89 DD FA
                      AC D2 C8 8E D9 EC 48 00 27 F3 76 3C 4E 52 94 3A 64 53 A7 E6 97 53 BD 38 BB B2 D9 E4 F1 60 1A 3F F7 7D 2C 57 7C 93 D1
                      9B C6 38 7B C6 1D CB FB 46 B7 69 CF BF B4 72 5C 5F 9E B8 9E D5 31 1A D5 46 32 E4 A1 12 85 1A 65 44 F9 7F 0F 7B 4C BD FA
                      50 9F 0E A9 72 EB AA 97 40 29 C1 4A 68 9F A9 81 1A 76 81 32 E3 E0 C3 54 30 F8 C0 E0 69 6C B6 98 E8 2E 74 FE FA A1 66 EE
                      DF D9 4B DD BF 13 73 AA 34 95 30 1C 7E 82 DF 9B D2 AE 85 6C 72 68 3E 6B B5 F1 4F A5 38 24 EF 7D 31 8C 4D 71 32 0C CE
                      13 D2 F5 8B 69 FB 7D 36 C3 B8 B9 D5 D6 81 8E BD 21 14 63 CA D2 72 D2 57 A8 2F EF BC C1 48 52 7B 01 51 89 27 10 52 53
                      30 E7 D3 19 03 2D 8B D9 C2 A6 9E 62 48 FC 90 30 76 A1 27 91 C9 F1 A3
           Exponent   65537
           Key Size   2048 bits
          Key Usage   Verify

          Signature   256 bytes : 17 87 7E C6 2C 0B 1C 20 ...
```

Certificate Authorities

More than 50 root certificates are trusted in the most popular web browser versions. A W3Techs survey from November 2012 shows:

- Symantec (which owns VeriSign, Thawte and Geotrust) with 42.9% market share
- Comodo with 26%
- GoDaddy with 14%
- GlobalSign with 7.7%

SSL/TLS

Transport Layer Security (**TLS**) and its predecessor, **Secure Sockets Layer** (**SSL**), are cryptographic protocols that provide communication security over the Internet. They use

- asymmetric cryptography for authentication of key exchange,
- symmetric encryption for confidentiality and
- message authentication codes for message integrity.

Several versions of the protocols are in widespread use in applications such as

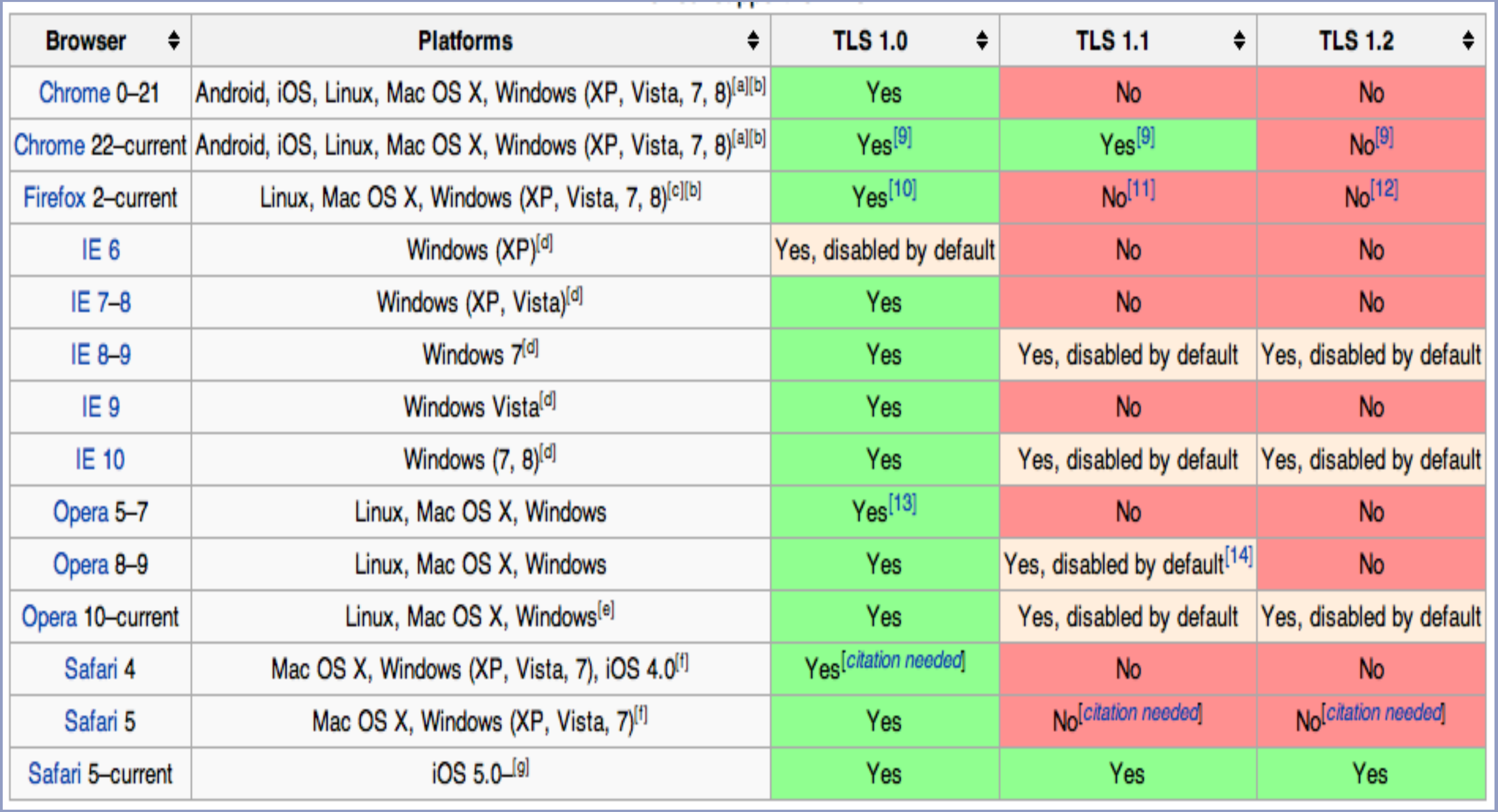
web browsing (HTTPS), electronic mail, Internet faxing, instant messaging and voice-over-IP (VoIP).

SSL/TLS handshake protocol

- The client sends the server the client's SSL version number, cipher settings, session-specific data, and other information that the server needs to communicate with the client using SSL.
- The server sends the client the server's SSL version number, cipher settings, session-specific data, and other information that the client needs to communicate with the server over SSL. The server also sends its own certificate, and if the client is requesting a server resource that requires client authentication, the server requests the client's certificate.
- The client uses the information sent by the server to authenticate the server. If the server cannot be authenticated, the user is warned of the problem and informed that an encrypted and authenticated connection cannot be established. If the server can be successfully authenticated, the client proceeds to the next step.
- Using all data generated in the handshake thus far, the client (with the cooperation of the server, depending on the cipher in use) creates the pre-master secret for the session, encrypts it with the server's public key (obtained from the server's certificate, sent in step 2), and then sends the encrypted pre-master secret to the server.
- If the server has requested client authentication (an optional step in the handshake), the client also signs another piece of data that is unique to this handshake and known by both the client and server. In this case, the client sends both the signed data and the client's own certificate to the server along with the encrypted pre-master secret.

- If the server has requested client authentication, the server attempts to authenticate the client. If the client cannot be authenticated, the session ends. If the client can be successfully authenticated, the server uses its private key to decrypt the pre-master secret, and then performs a series of steps (which the client also performs, starting from the same pre-master secret) to generate the master secret.
- Both the client and the server use the master secret to generate the session keys, which are symmetric keys used to encrypt and decrypt information exchanged during the SSL session and to verify its integrity (that is, to detect any changes in the data between the time it was sent and the time it is received over the SSL connection).
- The client sends a message to the server informing it that future messages from the client will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the client portion of the handshake is finished.
- The server sends a message to the client informing it that future messages from the server will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the server portion of the handshake is finished.

Browser support for SSL/TLS



Browser	Platforms	TLS 1.0	TLS 1.1	TLS 1.2
Chrome 0–21	Android, iOS, Linux, Mac OS X, Windows (XP, Vista, 7, 8)[a][b]	Yes	No	No
Chrome 22–current	Android, iOS, Linux, Mac OS X, Windows (XP, Vista, 7, 8)[a][b]	Yes[9]	Yes[9]	No[9]
Firefox 2–current	Linux, Mac OS X, Windows (XP, Vista, 7, 8)[c][b]	Yes[10]	No[11]	No[12]
IE 6	Windows (XP)[d]	Yes, disabled by default	No	No
IE 7–8	Windows (XP, Vista)[d]	Yes	No	No
IE 8–9	Windows 7[d]	Yes	Yes, disabled by default	Yes, disabled by default
IE 9	Windows Vista[d]	Yes	No	No
IE 10	Windows (7, 8)[d]	Yes	Yes, disabled by default	Yes, disabled by default
Opera 5–7	Linux, Mac OS X, Windows	Yes[13]	No	No
Opera 8–9	Linux, Mac OS X, Windows	Yes	Yes, disabled by default[14]	No
Opera 10–current	Linux, Mac OS X, Windows[e]	Yes	Yes, disabled by default	Yes, disabled by default
Safari 4	Mac OS X, Windows (XP, Vista, 7), iOS 4.0[f]	Yes[citation needed]	No	No
Safari 5	Mac OS X, Windows (XP, Vista, 7)[f]	Yes	No[citation needed]	No[citation needed]
Safari 5–current	iOS 5.0–[g]	Yes	Yes	Yes

Fragment of RFC 6101 specifying SSL 3.0

Freier, et al. Historic [Page 12]

RFC 6101 The SSL Protocol Version 3.0 August 2011

The session state includes the following elements:

session identifier: An arbitrary byte sequence chosen by the server to identify an active or resumable session state.

peer certificate: X509.v3 [X509] certificate of the peer. This element of the state may be null.

compression method: The algorithm used to compress data prior to encryption.

cipher spec: Specifies the bulk data encryption algorithm (such as null, DES, etc.) and a MAC algorithm (such as MD5 or SHA). It also defines cryptographic attributes such as the hash_size. (See Appendix A.7 for formal definition.)

master secret: 48-byte secret shared between the client and server.

is resumable: A flag indicating whether the session can be used toinitiate new connections.

The connection state includes the following elements:
server and client random: Byte sequences that are chosen by the server and client for each connection.

server write MAC secret: The secret used in MAC operations on data written by the server.

client write MAC secret: The secret used in MAC operations on data written by the client.

server write key: The bulk cipher key for data encrypted by the server and decrypted by the client.

client write key: The bulk cipher key for data encrypted by the client and decrypted by the server.

initialization vectors: When a block cipher in Cipher Block Chaining (CBC) mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL handshake protocol. Thereafter, the final ciphertext block from each record is preserved for use with the following record.

sequence numbers: Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero. Sequence numbers are of type uint64 and may not exceed 2^64-1.