

**POLITECNICO DI TORINO**

Faculty of Engineering

Mechatronic Engineering

**Master's Degree Thesis**

**An autopsy of password.link**



**Tutor**

Antonio Jose' Di Scala

**Author**

Manuel Pérez Anaya

AA2020/21



## **Abstract**

The aim of this Master Thesis is to learn and explain all the cryptographic instruments and how they are used in web page [password.link](#). The idea is to follow the interaction of an user with [password.link](#) explaining all the cryptographic/informatics passages with the aim of evaluate its security. So we have also to study the computer side of Internet e.g. the TCP/IP protocol. [Password.link](#) enables secrets to be sent through insecure channels.

The thesis is divided into three chapters. Chapter 1 is related to cryptographic concepts. Chapter 2 explains how the internet connection is established. Chapter 3 explains how [password.link](#) uses the tools described in Chapter 1 and Chapter 2 to achieve the goal.



# Contents

<b>List of Figures</b>	<b>9</b>
<b>List of Tables</b>	<b>11</b>
<b>1 Cryptography</b>	<b>13</b>
1.1 Random Number Generator (RNG)	15
1.1.1 True Random Number Generators (TRNG)	15
1.1.2 Pseudorandom Number Generators (PRNG)	15
1.1.3 Cryptographically Secure Pseudorandom Number Generators (CSPNG)	16
1.2 RC4	17
1.2.1 RC4 algorithm	17
1.2.2 RC4 RNG	20
1.3 Advanced Encryption System (AES)	23
1.3.1 Byte Substitution Layer	26
1.3.2 Diffusion Layer	28
1.3.3 Key Addition Layer	31
1.3.4 Decryption	33
1.4 Galois Counter Mode (GCM)	39
1.5 Introduction to Public-Key Cryptography	41
1.6 RSA	43
1.6.1 Encryption and decryption	43
1.6.2 RSA in practice	45
1.7 Discrete Logarithm Problem (DLP) and Diffie–Hellman Key Exchange (DHKE)	47
1.7.1 Discrete Logarithm Problem (DLP)	47
1.7.2 Diffie–Hellman Key Exchange (DHKE)	48

1.8	Elliptic Curve Cryptosystems (ECC)	51
1.8.1	Theory of Elliptic Curves	51
1.8.2	The Discrete Logarithm Problem with Elliptic Curves	55
1.8.3	Diffie–Hellman Key Exchange (DHKE) with Elliptic Curves	56
1.9	Hash Functions	59
1.9.1	Preimage resistance	60
1.9.2	Second preimage resistance	60
1.9.3	Collision resistance	61
1.10	SHA 256 algorithm	65
1.10.1	Preprocessing	65
1.10.2	Hash computation	67
1.11	PBKDF	71
1.12	Digital Signatures	75
1.12.1	The RSA Signature Scheme	75
1.12.2	A complete example	78
1.13	Certificates	81
1.13.1	Man in the middle attack	81
1.13.2	Certificates	82
1.13.3	X.509	83
<b>2</b>	<b>Internet connection</b>	<b>87</b>
2.1	Hypertext Transfer Protocol Secure (HTTPS)	89
2.1.1	Hypertext Transfer Protocol (HTTP)	89
2.1.2	Transport Layer Security (TLS)	90
2.2	TCP/IP	97
<b>3</b>	<b>Password.link</b>	<b>99</b>
3.1	Internet connection	105
3.2	Password.link operation	111
3.2.1	Generate the link	111
3.2.2	Encrypt the secret and send the information	120
3.2.3	View, decrypt and delete the secret	121
3.3	Additional Features	124
3.3.1	Personalize the service	124
3.3.2	Generator of passwords	124
3.4	Conclusion	126

3.5	Code . . . . .	128
3.5.1	simulation.html . . . . .	128
3.5.2	simulation.js . . . . .	129
<b>Bibliography</b>		<b>154</b>





# List of Figures

1.1	ASCII characters . . . . .	17
1.2	AES input/output parameters . . . . .	23
1.3	AES encryption block diagram . . . . .	25
1.4	AES encryption round . . . . .	27
1.5	AES substitution table S-Box . . . . .	28
1.6	AES subkeys 256 bits . . . . .	32
1.7	AES subkeys 128 bits . . . . .	34
1.8	AES decryption block diagram . . . . .	35
1.9	AES decryption round . . . . .	36
1.10	AES inverse substitution table S-Box . . . . .	38
1.11	GCM algorithm . . . . .	40
1.12	RSA encryption of a message M with padding . . . . .	45
1.13	$y^2 = x^3 - 3x + 3$ over $\mathbb{R}$ . . . . .	52
1.14	$P + Q$ . . . . .	52
1.15	$P + P$ . . . . .	53
1.16	$P$ and $-P$ . . . . .	54
1.17	Characteristics of hash functions . . . . .	60
1.18	Padding of a message . . . . .	65
1.19	Hash computation . . . . .	70
1.20	PBKDF algorithm . . . . .	73
1.21	RSA encryption of a message M with padding . . . . .	77
1.22	Chain of trust . . . . .	83
1.23	Version, serial number and signature . . . . .	84
1.24	Issuer name . . . . .	84
1.25	Validity period . . . . .	84
1.26	Subject name . . . . .	84
1.27	Subject of Public Key . . . . .	85

1.28 Fingerprint . . . . .	85
2.1 Evolution of internet users . . . . .	87
2.2 HTTPS . . . . .	89
2.3 Internet Protocol . . . . .	98
3.1 Dashboard of <a href="#">password.link</a> . . . . .	99
3.2 Plans offered by <a href="#">password.link</a> . . . . .	100
3.3 Dashboard fulfilled . . . . .	101
3.4 Link created . . . . .	102
3.5 View secret button . . . . .	103
3.6 Secret and message are revealed . . . . .	103
3.7 Secret viewed or expired . . . . .	104
3.8 Secret's situation . . . . .	104
3.9 Domain of <a href="#">password.link</a> . . . . .	105
3.10 Version of TLS and algorithms used . . . . .	105
3.11 End-entity certificate . . . . .	108
3.12 Intermediate certificate . . . . .	109
3.13 Root certificate . . . . .	110
3.14 Password generation . . . . .	124
3.15 Example using the code . . . . .	128

# List of Tables

1.1	Plaintext bytes . . . . .	24
1.2	Key bytes (128 bits) . . . . .	24
1.3	Key bytes (192 bits) . . . . .	26
1.4	Key bytes (256 bits) . . . . .	26
1.5	Result of substitution . . . . .	29
1.6	Result of shift rows . . . . .	29
1.7	Bit lengths of public-key algorithms for different security levels	42
1.8	RSA exponentiation with short public exponents . . . . .	44
1.9	Top 10 most common passwords by year according to Splash Data . . . . .	71
2.1	HTTP request and response . . . . .	90
2.2	TLS Ciphers supported . . . . .	90
2.3	TLS Key exchange algorithms supported . . . . .	91
3.1	Equivalence between numbers and characters . . . . .	112
3.2	Equivalence between numbers and characters Base64 . . . . .	119



# Chapter 1

## Cryptography

All the cryptographic functions that [password.link](#) uses are explained in this chapter. The main topics are the generation of random numbers, the encryption and decryption of data and mechanisms to ensure the authenticity.

The encryption and decryption of data can be done with symmetric algorithms or asymmetric algorithms.

Symmetric algorithms encryption and decryption of data can be realized by two parties with a secret key. The key must be preshared through a secure channel. Symmetric ciphers split into stream ciphers and block ciphers.

The encryption in a stream cipher is realized bit by bit and every bit of the message is XORed with a new and unpredictable bit produced by Random Number Generator.

In a block cipher the message is divided into blocks. Each block has the same length and it is encrypted and decrypted with the key. The last blocks are padded to guarantee that the length is the desired.

Asymmetric cipher encrypts and decrypts messages with two keys. One of them is the public key, this key enables to encrypt messages and can be known by anyone. The other key is the secret key which must be kept in secret and it is used to decrypt messages.



## 1.1 Random Number Generator (RNG)

The random numbers are very versatile in terms of cryptography so, a Random Number Generator (RNG) is essential. There are three different types of RNG. These are True Random Number Generators (TRNG), Pseudorandom Number Generators (PRNG) and Cryptographically Secure Pseudorandom Number Generators (CSPNG).

### 1.1.1 True Random Number Generators (TRNG)

TNRGs are based on physical processes and his main characteristic is that the output cannot be reproduced. To create a TRN an extra device is needed because a PC cannot create these numbers. Speaking in terms of a web page, the requirement of an extra device is enough to discard this method because the web pages must be used by anyone.

### 1.1.2 Pseudorandom Number Generators (PRNG)

PRNG create a sequence of numbers from an initial seed value.

$$s_0 = seed$$

$$s_{i+1} = f(i), \quad i = 0, 1, \dots$$

One of the most common PRNG is the *linear congruential generator* computed as:

$$s_0 = seed$$

$$s_{i+1} = (a \cdot s_i + b) \bmod m, \quad i = 0, 1, \dots$$

PRNGs can be broken easily if someone has enough values of  $s$ . Once the PRNG is broken all the past and futures values can be discovered. PRNGs are not cryptographically secure and cannot be used for these purposes because the PRNGs was born to perform simulations as Montecarlo. The initial purpose of the PRNGs was not to be cryptographically secure.

### 1.1.3 Cryptographically Secure Pseudorandom Number Generators (CSPNG)

A CSPRNG is PRNG which is unpredictable. It means that if someone has the values between  $s_0$  and  $s_n$ , the person cannot establish a relationship to discover the value of  $s_{n+1}$ .

The outputs of the CSPRNGs should be computationally indistinguishable (IND goal), the CSPRNG are tested to check it. For example, a CSPRNG must pass the statistical tests developed by the National Institute of Standards and Technology (NIST), see e.g. [A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications](#)



## 1.2 RC4

Once that RNG are explained, it is the time to explain the algorithm RC4 and the particular form it is implemented in [password.link](#).

### 1.2.1 RC4 algorithm

To perform the RC4 algorithm a key is needed. The length of the key is not limited but note that a key longer than 256 characters will produce the same effect that the same key truncated from the characters 1 to 256. If a key is shorter than 256 characters, also will be accepted. The first step to perform is a conversion from characters to numbers. It is done with ASCII table.

ASCII control characters			ASCII printable characters						Extended ASCII characters							
00	NULL	(Null character)	32	space	64	@	96	`	128	Ç	160	à	192	Ł	224	Ó
01	SOH	(Start of Header)	33	!	65	A	97	a	129	ü	161	í	193	±	225	ß
02	STX	(Start of Text)	34	"	66	B	98	b	130	é	162	ó	194	⌈	226	Ô
03	ETX	(End of Text)	35	#	67	C	99	c	131	â	163	û	195	⌋	227	Õ
04	EOT	(End of Trans.)	36	\$	68	D	100	d	132	ä	164	ñ	196	—	228	ö
05	ENQ	(Enquiry)	37	%	69	E	101	e	133	à	165	Ñ	197	†	229	Ö
06	ACK	(Acknowledgement)	38	&	70	F	102	f	134	á	166	ª	198	‡	230	µ
07	BEL	(Bell)	39	'	71	G	103	g	135	ç	167	º	199	Ä	231	þ
08	BS	(Backspace)	40	(	72	H	104	h	136	ê	168	¿	200	ℒ	232	ð
09	HT	(Horizontal Tab)	41	)	73	I	105	i	137	ë	169	®	201	ℓ	233	ú
10	LF	(Line feed)	42	*	74	J	106	j	138	è	170	™	202	ℓ	234	Û
11	VT	(Vertical Tab)	43	+	75	K	107	k	139	ï	171	½	203	ℓ	235	Ü
12	FF	(Form feed)	44	,	76	L	108	l	140	ì	172	¾	204	ℓ	236	Ý
13	CR	(Carriage return)	45	-	77	M	109	m	141	í	173	¿	205	=	237	Ÿ
14	SO	(Shift Out)	46	.	78	N	110	n	142	Ä	174	«	206	≠	238	—
15	SI	(Shift In)	47	/	79	O	111	o	143	Å	175	»	207	≠	239	·
16	DLE	(Data link escape)	48	0	80	P	112	p	144	É	176	≡	208	ø	240	≡
17	DC1	(Device control 1)	49	1	81	Q	113	q	145	æ	177	≡	209	Ð	241	±
18	DC2	(Device control 2)	50	2	82	R	114	r	146	Æ	178	≡	210	É	242	¼
19	DC3	(Device control 3)	51	3	83	S	115	s	147	ô	179	⌈	211	Ê	243	½
20	DC4	(Device control 4)	52	4	84	T	116	t	148	ö	180	⌋	212	Ë	244	¾
21	NAK	(Negative acknowl.)	53	5	85	U	117	u	149	õ	181	À	213	Ì	245	÷
22	SYN	(Synchronous idle)	54	6	86	V	118	v	150	ù	182	Á	214	Í	246	¸
23	ETB	(End of trans. block)	55	7	87	W	119	w	151	û	183	Â	215	Î	247	°
24	CAN	(Cancel)	56	8	88	X	120	x	152	ý	184	Ë	216	Ï	248	…
25	EM	(End of medium)	57	9	89	Y	121	y	153	Ö	185	⌈	217	⌋	249	·
26	SUB	(Substitute)	58	:	90	Z	122	z	154	Û	186	⌈	218	⌋	250	ˆ
27	ESC	(Escape)	59	;	91	[	123	{	155	ø	187	⌈	219	⌈	251	ˆ
28	FS	(File separator)	60	<	92	\	124		156	£	188	⌈	220	⌈	252	ˆ
29	GS	(Group separator)	61	=	93	]	125	}	157	Ø	189	¢	221	⌈	253	ˆ
30	RS	(Record separator)	62	>	94	^	126	~	158	×	190	¥	222	⌈	254	■
31	US	(Unit separator)	63	?	95	_			159	f	191	γ	223	⌈	255	nbsp
127	DEL	(Delete)														

Figure 1.1: ASCII characters

RC4 creates an initial array with a length of 256 called S-box. The initial

S-box is:

$$S = [0, 1, 2, 3, 4, \dots, 255]$$

Once it is defined, it is the time to combine the key with the S-box. To do it, each position of S-Box will have a new value which depends on the current value and the key.

$$j = 0$$

*for i from 0 to 255 :*

$$j = 1111111_2 \& (j + key(i \bmod length_{key}) + S(i))$$

*swap values of S(i) and S(j)*

*end for*

The symbol & indicates the bitwise AND operation. It is used when j has a value bigger than 256.

To understand it better an example will be performed. Note that the example is realized with an S-Box which has a length of 4 instead of 256. It also has influence where  $j$  is calculated, the bitwise and operation now is:

$$j = 11_2 \& (j + key(i \bmod length_{key}) + S(i))$$

Suppose that the key is:

$$key = [1, 2, 3, 0]$$

$$length_{key} = 4$$

$$S_{initial} = [0, 1, 2, 3]$$

Now, the operations inside the loop for are performed.

$$j = 0$$

for i=0

$$j = 11_2 \& (0 + 1 + 0) = 1$$

$$S(0) = 1$$

$$S(1) = 0$$

$$S = [1, 0, 2, 3]$$

for i=1

$$j = 11_2 \& (1 + 2 + 0) = 3$$

$$S(1) = 3$$

$$S(3) = 0$$

$$S = [1, 3, 2, 0]$$

for i=2

$$j = 11_2 \& (3 + 3 + 2) = 11_2 \& 8 = 11_2 \& 1000_2 = 00_2 = 0$$

$$S(2) = 1$$

$$S(0) = 2$$

$$S = [2, 3, 1, 0]$$

for i=3

$$j = 11_2 \& (0 + 0 + 0) = 0$$

$$S(3) = 2$$

$$S(0) = 0$$

$$S = [0, 3, 1, 2]$$

### 1.2.2 RC4 RNG

The previous RC4 algorithm can be used as RNG. To do it possible, another loop is implemented. The number of times that each operation is executed is chosen by the user through the variable *count*. It means, that if the value of *count* is for instance equal to 5, each operation inside the loop will be executed 5 times. Usually, this parameter is fixed. The another initial parameter is the S-box. The first time RC4 RNG is run, the S-Box initial value coincides with the S-box final value of the RC4 algorithm. Once RC4 RNG is compiled, it saves the final value of S-box and it will be used as the initial value if the function is called again. Note that if the initial S-box does not change and count is fixed, RC4 RNG will always return the same number.

```
i = 0
j = 0
r = 0
while count > 0 :
    count = count - 1
    i = 111111112&(i + 1)
    j = 111111112&(j + S(i)) = 0
    swap values of S(i) and S(j)
r = 256 · r + S(111111112 · (S(i) + S(j)))
return r
```

An example will be shown in this section, the values of the initial parameters are *count*=4 and *S*=[0,3,1,2]. This is a simulation of the first time that RC4 RNG is called so, S has the same value that the final S in the section 1.2.1 RC4 algorithm.

```
i = 0
j = 0
r = 0
```

$$count = 4$$

$$count = 3$$

$$i = 11_2 \& (0 + 1) = 1$$

$$j = 11_2 \& (0 + 3) = 3$$

$$S(1) = 2$$

$$S(3) = 3$$

$$S = [0, 2, 1, 3]$$

$$r = 4 * r + S(11_2 \& (2 + 3)) = 0 + S(11_2 \& 5) = S(11_2 \& 101_2) = S(01_2) = S(1) = 2$$

$$count = 3$$

$$count = 2$$

$$i = 11_2 \& (0 + 1) = 2$$

$$j = 11_2 \& (3 + 1) = 0$$

$$S(2) = 0$$

$$S(0) = 1$$

$$S = [1, 2, 0, 3]$$

$$r = 4 * r + S(11_2 \& (0 + 1)) = 8 + S(11_2 \& 1) = 10$$

$$count = 2$$

$$count = 1$$

$$i = 11_2 \& (0 + 1) = 3$$

$$j = 11_2 \& (0 + 3) = 3$$

$$S(3) = 3$$

$$S(3) = 3$$

$$S = [1, 2, 0, 3]$$

$$r = 4 * r + S(11_2 \& (3 + 3)) = 40 + S(11_2 \& 6) = 40$$

$$count = 1$$

$$count = 0$$

$$i = 11_2 \& (0 + 1) = 0$$

$$j = 11_2 \& (3 + 1) = 0$$

$$S(0) = 1$$

$$S(3) = 1$$

$$S = [1, 2, 0, 3]$$

$$r = 4 * r + S(11_2 \& (1 + 1)) = 160 + S(11_2 \& 2) = 160$$

*return*  $r = 160$

If RC4 RNG is called again the initial value of S-box is  $S=[1,2,0,3]$ .

Multiple vulnerabilities have been discovered in RC4, converting it insecure.

### 1.3 Advanced Encryption System (AES)

The Advanced Encryption System (AES) is the most widely symmetric block cipher used. Notice that a block cipher encrypts and decrypts a message splitting it into different blocks where each block has the same length. If the last block is shorter, it will be padded. To encrypt a secret, a key is used. The AES encrypts and decrypts 128 bits blocks of 128 bits and it supports keys of 128, 192, 256 bits. The ciphertext produced in the encryption also has a length of 128 bits.

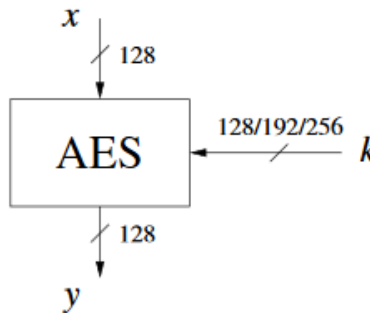


Figure 1.2: AES input/output parameters

According to the length of the key, the AES will execute 10 rounds (128 bits), 12 rounds (192 bits) or 14 rounds (256 bits). The AES encrypts all 128 bits in one round. All the rounds are composed of three layers unless the first and the last layer.

- **Byte Substitution layer (S-Box):** It provides confusion over the data assuring that changes in individual bits will propagate quickly across the data path. A non-linearity transformation is performed with tables which have special mathematical properties.
- **Diffusion layer:** It introduces diffusion to all state bits through two sublayers, which perform linear operations:
  - Shift rows layer
  - Mix column layer
- **Key Addition layer:** A 128-bit round key, or subkey, which has been derived from the main key in the key schedule, is XORed to the state.

The concepts of confusion and diffusion are two properties of the operation of a secure cipher identified by Claude Shannon.

- Confusion: hides the relationship between the ciphertext and the key, increasing the ambiguity of ciphertext.
- Diffusion: means that if a bit of the plaintext changes, the ciphertext will change the half of its bits and the same occurs in reverse.

The first round includes an additional Key Addition layer before the three layers and the last round does not have Mix Columns layer.

The plaintext has a length of 128 bits (16 bytes) and it is arranged in a four-by-four byte matrix. Something similar occurs to the key.

Plaintext			
$A_0$	$A_4$	$A_8$	$A_{12}$
$A_1$	$A_5$	$A_9$	$A_{13}$
$A_2$	$A_6$	$A_{10}$	$A_{14}$
$A_3$	$A_7$	$A_{11}$	$A_{15}$

Table 1.1: Plaintext bytes

Key 16 bytes			
$k_0$	$k_4$	$k_8$	$k_{12}$
$k_1$	$k_5$	$k_9$	$k_{13}$
$k_2$	$k_6$	$k_{10}$	$k_{14}$
$k_3$	$k_7$	$k_{11}$	$k_{15}$

Table 1.2: Key bytes (128 bits)

The internal structure of a round is shown in 1.4.



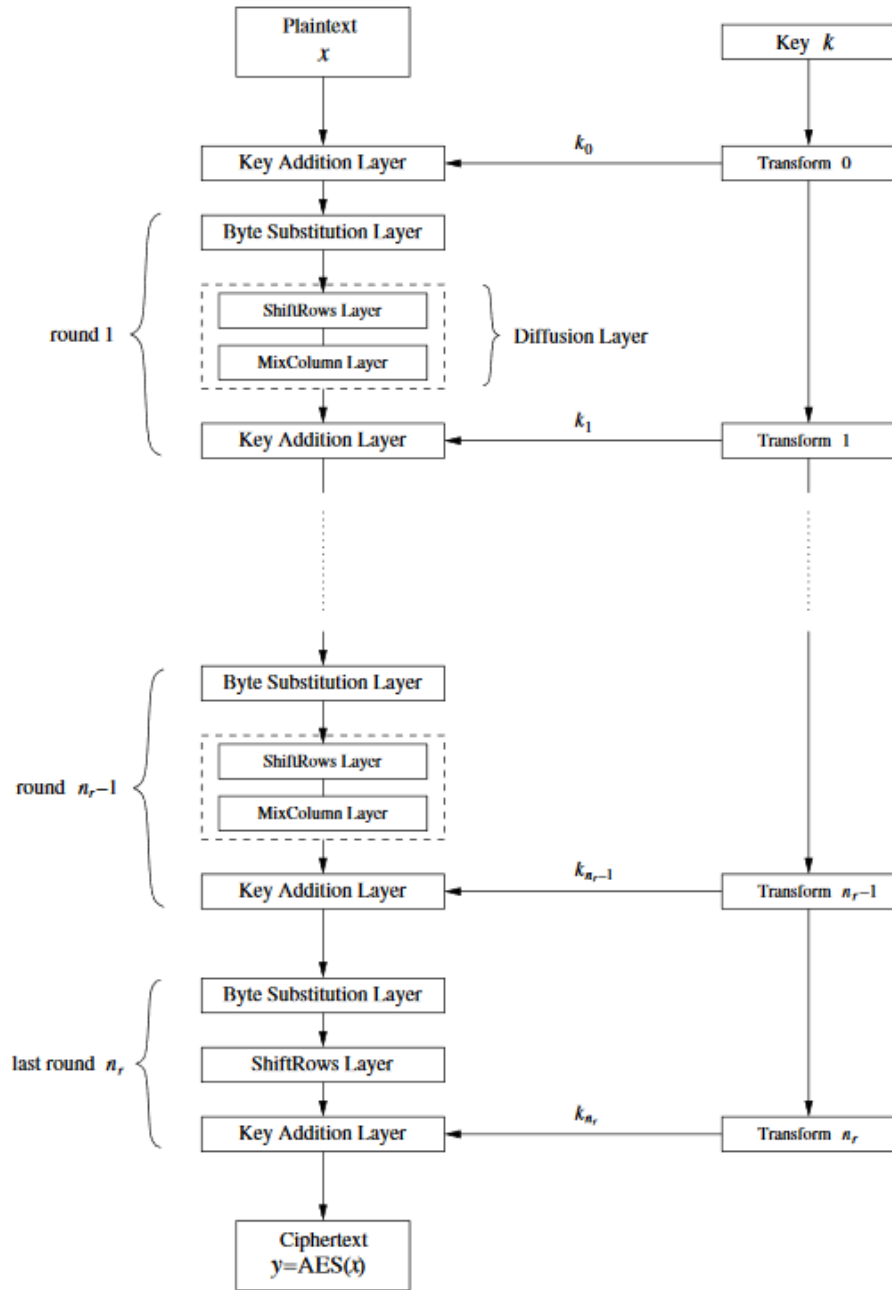


Figure 1.3: AES encryption block diagram

Key 24 bytes					
$k_0$	$k_4$	$k_8$	$k_{12}$	$k_{16}$	$k_{20}$
$k_1$	$k_5$	$k_9$	$k_{13}$	$k_{17}$	$k_{21}$
$k_2$	$k_6$	$k_{10}$	$k_{14}$	$k_{18}$	$k_{22}$
$k_3$	$k_7$	$k_{11}$	$k_{15}$	$k_{19}$	$k_{23}$

Table 1.3: Key bytes (192 bits)

Key 32 bytes							
$k_0$	$k_4$	$k_8$	$k_{12}$	$k_{16}$	$k_{20}$	$k_{24}$	$k_{28}$
$k_1$	$k_5$	$k_9$	$k_{13}$	$k_{17}$	$k_{21}$	$k_{25}$	$k_{29}$
$k_2$	$k_6$	$k_{10}$	$k_{14}$	$k_{18}$	$k_{22}$	$k_{26}$	$k_{30}$
$k_3$	$k_7$	$k_{11}$	$k_{15}$	$k_{19}$	$k_{23}$	$k_{27}$	$k_{31}$

Table 1.4: Key bytes (256 bits)

### 1.3.1 Byte Substitution Layer

In the Byte Substitution layer, each byte  $A_i$  is replaced by another byte  $B_i$  as shown in 1.4.

$$S(A_i) = B_i$$

It is a non-linear operation which means:

$$S(C) + S(D) \neq S(C + D)$$

To perform the substitution, a conversion to hexadecimal is needed. One the byte is in hexadecimal, the first character chooses the row and the second the column.

To try to understand it better an example will be realized. Suppose  $A_{example}$  is a matrix of 128 bits (16 bytes) where all the bytes are the character "M". The substitution is:

$$A_{example} = \begin{pmatrix} M & M & M & M \\ M & M & M & M \\ M & M & M & M \\ M & M & M & M \end{pmatrix}_{ASCII}$$

$$M_{ASCII} = 77_{10} = 1001101_2 = 4D_{hex}$$

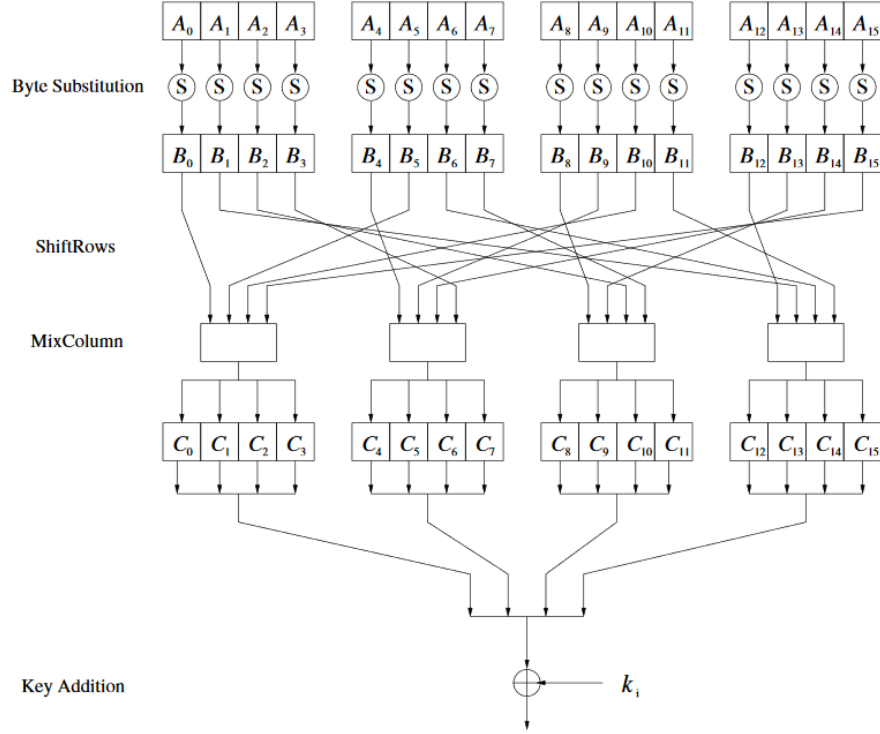


Figure 1.4: AES encryption round

$$S(4D_{hex}) = E3_{hex} = 11100011_2$$

$$S(A_{example}) = \begin{pmatrix} E3 & E3 & E3 & E3 \\ E3 & E3 & E3 & E3 \\ E3 & E3 & E3 & E3 \\ E3 & E3 & E3 & E3 \end{pmatrix}_{hex}$$

The result of this layer is a matrix where each byte has been substituted.

	y															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
x 8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Figure 1.5: AES substitution table S-Box

### 1.3.2 Diffusion Layer

As it is explained before, the Diffusion layer performs two linear operations:

$$DIFF(C) + DIFF(D) = DIFF(C + D)$$

#### Shift Rows

The Shift Rows shifts the second row three bytes to the right, the third row is shifted two bytes to the right and the fourth row is shifted one byte to the right.

In the previous example obviously, the shift rows do not change anything because all the bytes are the same.

Key 16 bytes			
$B_0$	$B_4$	$B_8$	$B_{12}$
$B_1$	$B_5$	$B_9$	$B_{13}$
$B_2$	$B_6$	$B_{10}$	$B_{14}$
$B_3$	$B_7$	$B_{11}$	$B_{15}$

Table 1.5: Result of substitution

Key 16 bytes			
$B_0$	$B_4$	$B_8$	$B_{12}$
$B_5$	$B_9$	$B_{13}$	$B_1$
$B_{10}$	$B_{14}$	$B_2$	$B_6$
$B_{15}$	$B_3$	$B_7$	$B_{11}$

Table 1.6: Result of shift rows

### Mix Columns

Mix Columns mixes each column of the state matrix. Multiplication and addition of the coefficients is done in the Galois Field ( $2^8$ ).

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}$$

$$\begin{pmatrix} C_4 \\ C_5 \\ C_6 \\ C_7 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} B_4 \\ B_9 \\ B_{14} \\ B_3 \end{pmatrix}$$

$$\begin{pmatrix} C_8 \\ C_9 \\ C_{10} \\ C_{11} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} B_8 \\ B_{13} \\ B_2 \\ B_7 \end{pmatrix}$$

$$\begin{pmatrix} C_{12} \\ C_{13} \\ C_{14} \\ C_{15} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} B_{12} \\ B_1 \\ B_6 \\ B_{11} \end{pmatrix}$$

The numbers 01, 02 and 03 represent polynomials in the  $\text{GF}(2^8)$ . The modular reduction polynomial is  $P(x)$ .

$$01 = 00000001_2 = (1)_{GF}$$

$$02 = 00000010_2 = (x)_{GF}$$

$$03 = 00000011_2 = (x + 1)_{GF}$$

$$P(x) = x^8 + x^4 + x^3 + x + 1$$

Continuing with the example:

$$E3_{hex} = 11100011_2 = (x^7 + x^6 + x^5 + x + 1)_{GF}$$

$$01 \cdot E3 = (x^7 + x^6 + x^5 + x + 1)_{GF} = 11100011_2$$

$$02 \cdot E3 = (x^8 + x^7 + x^6 + x^2 + x)_{GF} = (x^7 + x^6 + x^4 + x^3 + x^2 + 1)_{GF} = 11011101_2$$

$$03 \cdot E3 = (x^8 + x^5 + x^2 + 1)_{GF} = (x^5 + x^4 + x^3 + x)_{GF} = 00111110_2$$

Note that the results of  $02 \cdot E3$  and  $03 \cdot E3$  has a degree bigger than 8 and a modular reduction with the polynomial must be realized.

$$01 \cdot E3 + 01 \cdot E3 + 02 \cdot E3 + 03 \cdot E3 = 11100011_2 + 11100011_2 + 11011101_2 + 00111110_2$$

$$01 \cdot E3 + 01 \cdot E3 + 02 \cdot E3 + 03 \cdot E3 = 11100011_2 = E3_{hex}$$

$$C = \begin{pmatrix} E3 & E3 & E3 & E3 \\ E3 & E3 & E3 & E3 \\ E3 & E3 & E3 & E3 \\ E3 & E3 & E3 & E3 \end{pmatrix}_{hex}$$

### 1.3.3 Key Addition Layer

In the key addition layer, the state matrix (16 bytes) is XORed with a subkey which has also 16 bytes. The subkeys are obtained from the original key. The original key can be 128 bits, 192 bits or 256 bits. The question is how to obtain the subkeys from the original key. Only 256 bits and 128 bits will be considered this paper.

#### AES-256

It is important to notice that AES-256 does 14 rounds. In each round, a subkey is needed and in the first round, an extra subkey is needed. The total number of subkeys is 15. The original key has 32 bytes.

$$Key = [K_0, K_1, K_2, \dots, K_{31}]$$

The subkeys are stored in words, each word has 32 bits. The subkeys have a length of 128 bits. It indicates that each subkey is composed of four words and 60 words are required.

$$k_i = [W[4*(i-1)], W[4*(i-1)+1], W[4*(i-1)+2], W[4*(i-1)+3]] \quad i = 1, \dots, 15$$

The words are formed as follows:

$$\begin{aligned} W[i] &= [K_{4i}, K_{4i+1}, K_{4i+2}, K_{4i+3}] & i = 0, 1, 2, \dots, 7 \\ W[8i] &= [W[8(i-1)] + g(W[8i-1])] & i = 1, 2, \dots, 7 \\ W[8i+4] &= [W[8(i-1)+4] + h(W[8i+3])] & i = 1, 2, \dots, 6 \\ W[i] &= W[i-8] + W[i-1] & i > 8 \ \& \ i \bmod 4 \neq 0 \end{aligned}$$

The function  $g()$  is non-linear and has as input 1 word (4 bytes). Each byte is shifted three positions to the right, then the S-box is applied to each byte and finally, the most left byte is XORed with  $RC[i]$ . The values of  $RC[i]$  are:

$$\begin{aligned} RC[1] &= x^0 = 00000001_2 \\ RC[2] &= x^1 = 00000010_2 \\ RC[3] &= x^2 = 00000100_2 \end{aligned}$$

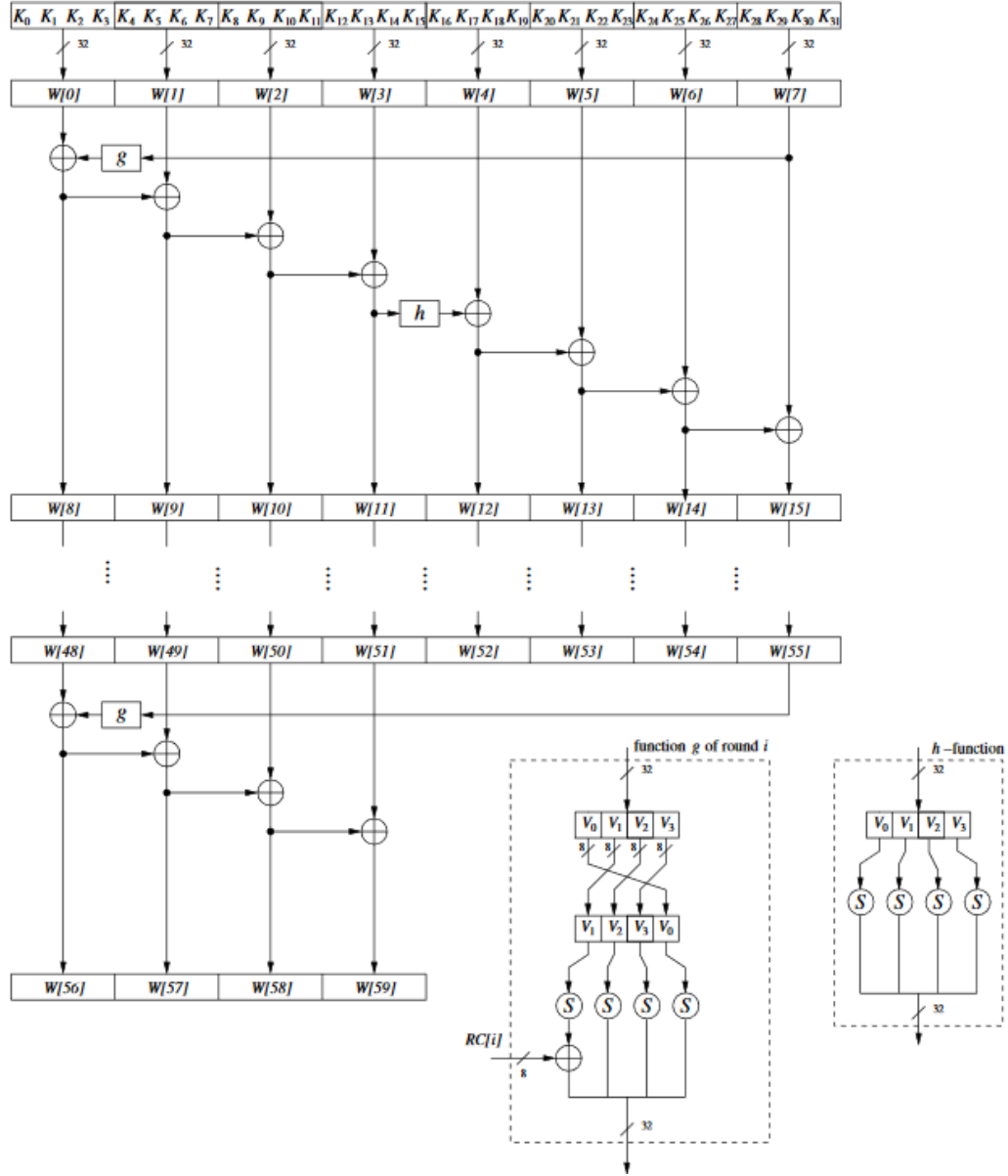


Figure 1.6: AES subkeys 256 bits

$$RC[4] = x^3 = 00001000_2$$

$$RC[5] = x^4 = 00010000_2$$

$$RC[6] = x^5 = 00100000_2$$



$$RC[7] = x^6 = 01000000_2$$

The function  $h()$  is non-linear, which has as input 1 word (4 bytes). Each byte is introduced to the S-box.

### AES-128

In the case of AES-128, 10 rounds are performed and in each round a subkey is needed but the first round needs an extra subkey so, the total number of subkeys is 11. The original key has 128 bits, it means 16 bytes and only 44 words are required. The process is similar to the AES-256.

$$Key = [K_0, K_1, K_2, \dots, K_{15}]$$

$$k_i = [W[4*(i-1)], W[4*(i-1)+1], W[4*(i-1)+2], W[4*(i-1)+3]] \quad i = 1, \dots, 11$$

$$\begin{aligned} W[i] &= [K_{4i}, K_{4i+1}, K_{4i+2}, K_{4i+3}] & i = 0, 1, 2, 3 \\ W[4i] &= [W[4(i-1)] + g(W[4i-1])] & i = 1, 2, \dots, 7 \\ W[i] &= W[i-4] + W[i-1] & i > 3 \ \& \ i \bmod 4 \neq 0 \end{aligned}$$

### 1.3.4 Decryption

To decrypt a message all the operations must be inverted.

The inversion of Key Addition layer is performed XORing the ciphertext with the subkeys. The only difference that exists is that the order of the subkeys is the inverse. The order to XOR the subkeys is:

$$k_{15}, k_{14}, k_{13}, \dots, k_1, k_0$$

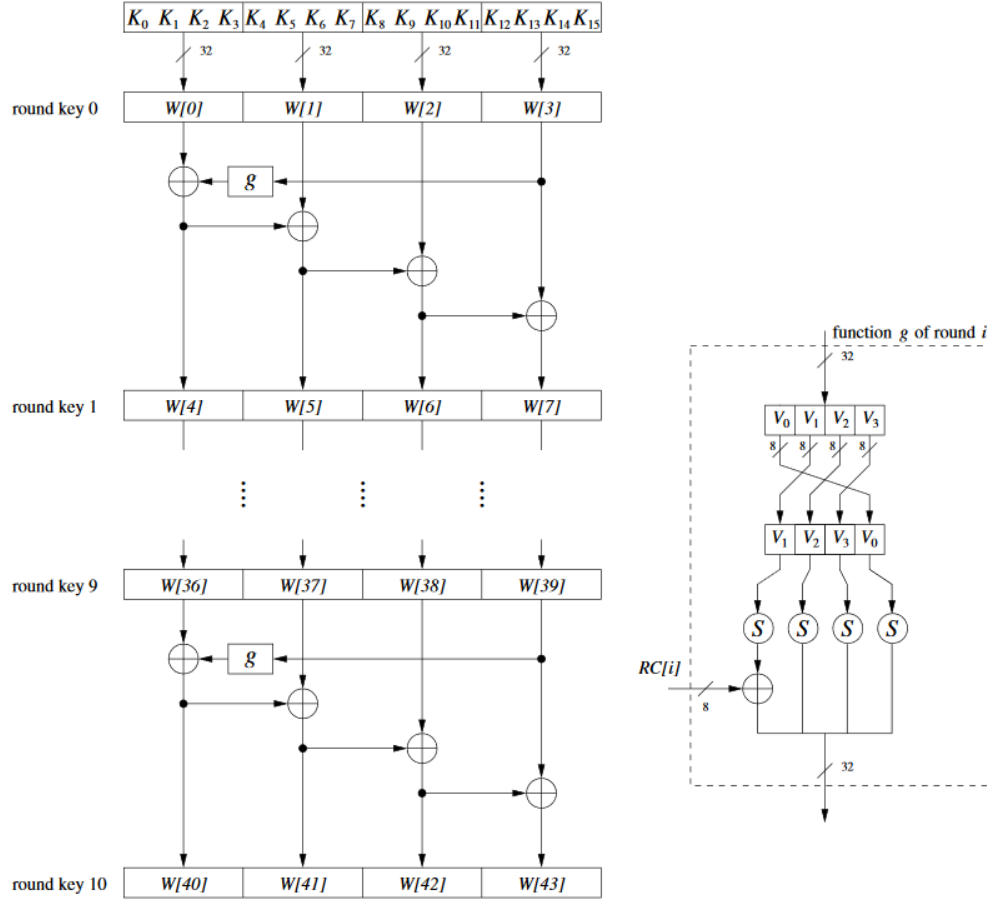


Figure 1.7: AES subkeys 128 bits

To invert the Mix Columns the process is the same, an operation with matrix in the Galois Field ( $2^8$ ) but the matrix changes.

$$\begin{pmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0B & 09 & 0E \end{pmatrix} \cdot \begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix}$$

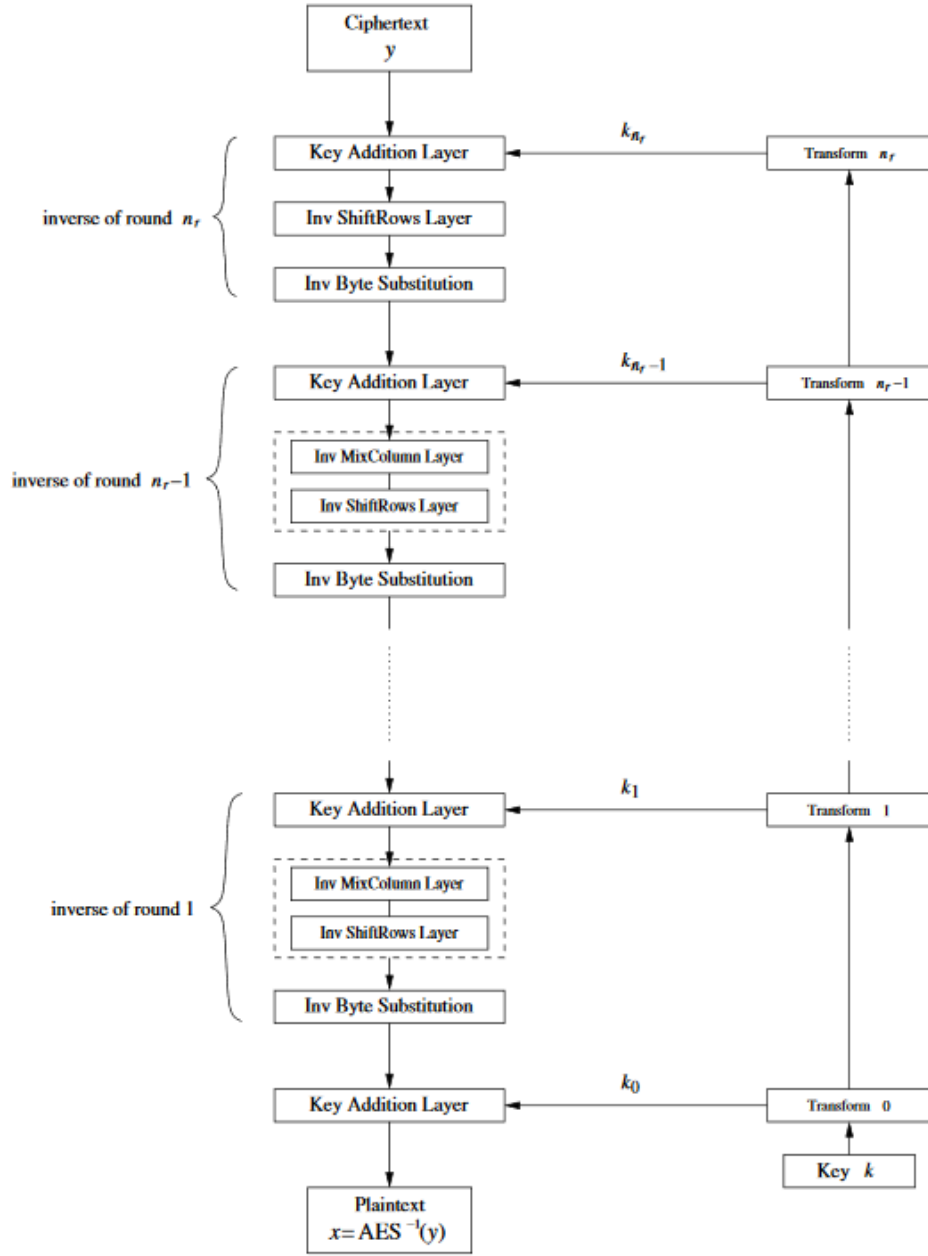


Figure 1.8: AES decryption block diagram

$$\begin{pmatrix} B_4 \\ B_5 \\ B_6 \\ B_7 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \cdot \begin{pmatrix} C_4 \\ C_5 \\ C_6 \\ C_7 \end{pmatrix}$$

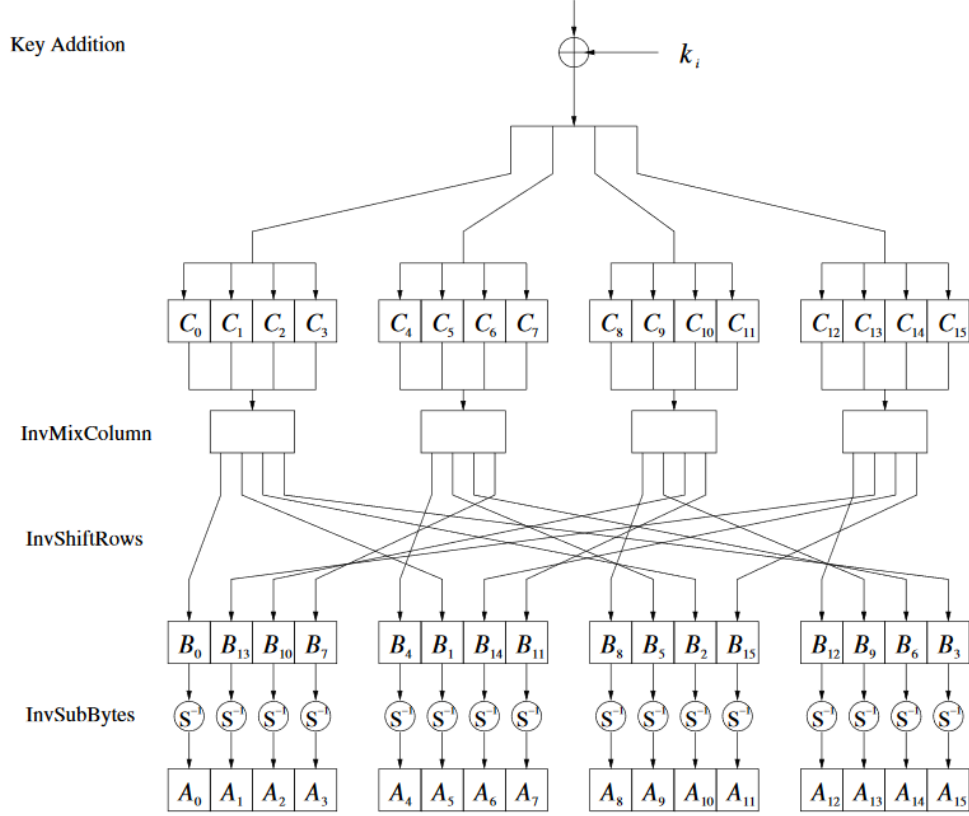


Figure 1.9: AES decryption round

$$\begin{pmatrix} B_8 \\ B_9 \\ B_{10} \\ B_{11} \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0B & 09 & 0E \end{pmatrix} \cdot \begin{pmatrix} C_8 \\ C_9 \\ C_{10} \\ C_{11} \end{pmatrix}$$

$$\begin{pmatrix} B_{12} \\ B_{13} \\ B_{14} \\ B_{15} \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0B & 09 & 0E \end{pmatrix} \cdot \begin{pmatrix} C_{12} \\ C_{13} \\ C_{14} \\ C_{15} \end{pmatrix}$$

$$\begin{aligned}
09_{hex} &= 00001001_2 = (x^3 + 1)_{GF} \\
0B_{hex} &= 00001011_2 = (x^3 + x + 1)_{GF} \\
0D_{hex} &= 00001101_2 = (x^3 + x^2 + 1)_{GF} \\
0E_{hex} &= 00001110_2 = (x^3 + x^2 + x)_{GF}
\end{aligned}$$

Using it to the inverse the example. It follows as:

$$C = \begin{pmatrix} E3 & E3 & E3 & E3 \\ E3 & E3 & E3 & E3 \\ E3 & E3 & E3 & E3 \\ E3 & E3 & E3 & E3 \end{pmatrix}_{hex}$$

$$\begin{aligned}
09 \cdot E3 &= (x^7 + x^5 + x^4 + x^3 + x + 1)_{GF} = 10111010_2 \\
0B \cdot E3 &= (x^3 + x + 1)_{GF} = 01100111_2 \\
0D \cdot E3 &= (x^4 + x^3 + x + 1)_{GF} = 00011011_2 \\
0E \cdot E3 &= (x^5 + x^2 + 1)_{GF} = 00100101_2
\end{aligned}$$

$$09 \cdot C5 + 0B \cdot C5 + 0D \cdot C5 + 0E \cdot C5 = 10111010_2 + 01100111_2 + 00011011_2 + 00100101_2$$

$$09 \cdot C5 + 0B \cdot C5 + 0D \cdot C5 + 0E \cdot C5 = 11100011_2 = E3_{hex}$$

$$B = \begin{pmatrix} E3 & E3 & E3 & E3 \\ E3 & E3 & E3 & E3 \\ E3 & E3 & E3 & E3 \\ E3 & E3 & E3 & E3 \end{pmatrix}_{hex}$$

The operation to invert Shift Rows is similar to the original operation. The second row of the state matrix is shifted one byte to the right, the third two bytes to the right and the fourth three bytes to the right.

Finally, to invert the substitution is necessary to realize other substitution. The table to do it is shown in Figure 1.10.

In the example:

$$S_{inv}(E3_{hex}) = D4_{hex} = 01001101_2 = 77_{10} = M_{ASCII}$$

	$y$															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
$x$ 8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Figure 1.10: AES inverse substitution table S-Box

## 1.4 Galois Counter Mode (GCM)

In the previous section is explained how to encrypt a 128 bits plaintext. In practice, the plaintexts are longer than 8 bytes. The way to encrypt and decrypt long plaintexts is through the operations mode. One of them is Galois Counter Mode (GCM). GCM not only enables to encrypt and decrypt long plaintexts but also it guarantees authentication and integrity. It means that the message was created by the correct person and that nobody tampered with the ciphertext during transmission.

The input parameters are plaintext, additional authenticated data (ADD) and initialization vector (IV).

The plaintext is divided in 128 bits plaintexts length as follows:

$$Plaintext = [x_1 || x_2 || \dots || x_n]$$

where  $||$  denotes concatenation.

ADD might include addresses, ports, sequence numbers, protocol version numbers, and other fields that indicate how the plaintext should be treated.

IV is a nonce, i.e, a value that is unique within the specified context. It is used for deriving a counter value ( $CTR_0$ ).

$$CTR_i = CTR_{i-1} + 1$$

GCM obtains the ciphertext encrypting  $CTR_i$  and then XORing it with the plaintext  $x_i$ .

$$y_i = e_k(CTR_i) \oplus x_i$$

The value H is generated by encryption of the all-zeros input with the block cipher.

$$H = e_k(0)$$

The values of  $g_i$  are obtained as:

$$g_0 = ADD \otimes H$$

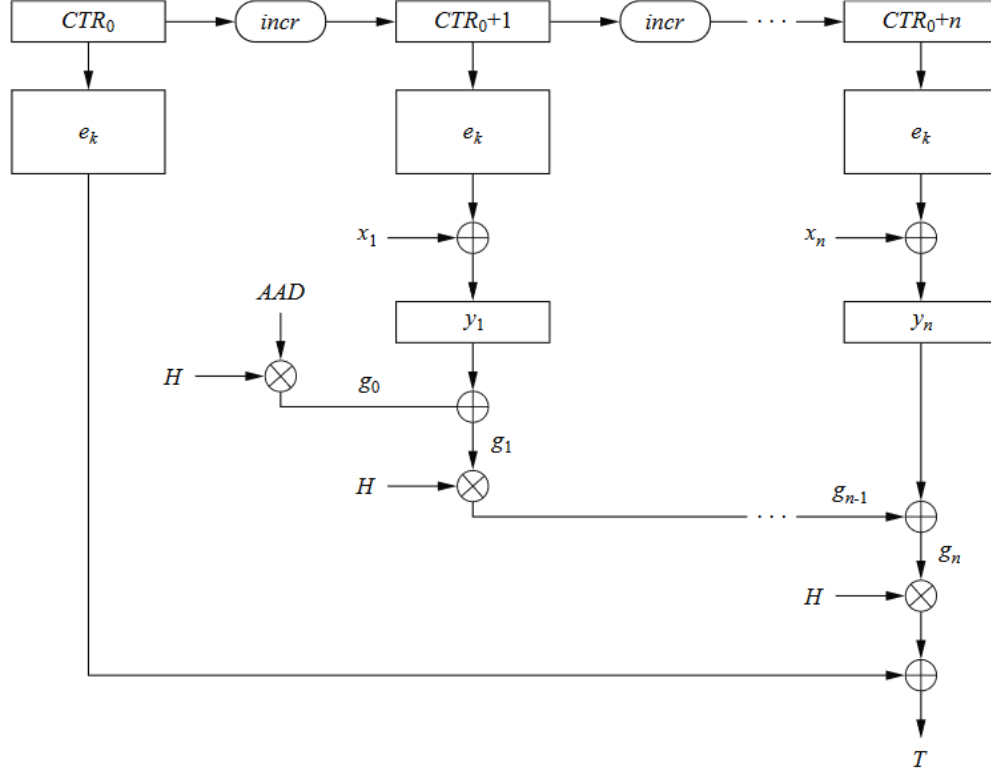


Figure 1.11: GCM algorithm

$$g_i = (g_{i-1} \oplus y_i) \otimes H \quad i = 1, 2, \dots, n$$

The operation  $\otimes$  is a multiplication in the GF ( $2^{128}$ ) with the irreducible polynomial  $P(x) = x^{128} + x^7 + x^2 + x + 1$ .

The authentication (T) is computed as:

$$T = (g_n \otimes H) \oplus e_k(CTR_0)$$

The person who receives the message must have the key and IV to decrypt the message. The message sent contains  $[(y_1, y_2, \dots, y_n), T, ADD]$ . This person realizes the same operations that the sender and finally has  $T'$ . If  $T = T'$ , it assures who is the sender and that the message has not been manipulated.



## 1.5 Introduction to Public-Key Cryptography

Public-Key Cryptography (PKC) or asymmetric cryptography was introduced in 1976 whereas symmetric cryptography is used for at least 4000 years. Symmetric cryptography has some issues. These are key distribution problem, number of keys, no protection against cheating and other malicious actions.

Key distribution problem is referred to as the key shared between two persons before a secure channel is used.

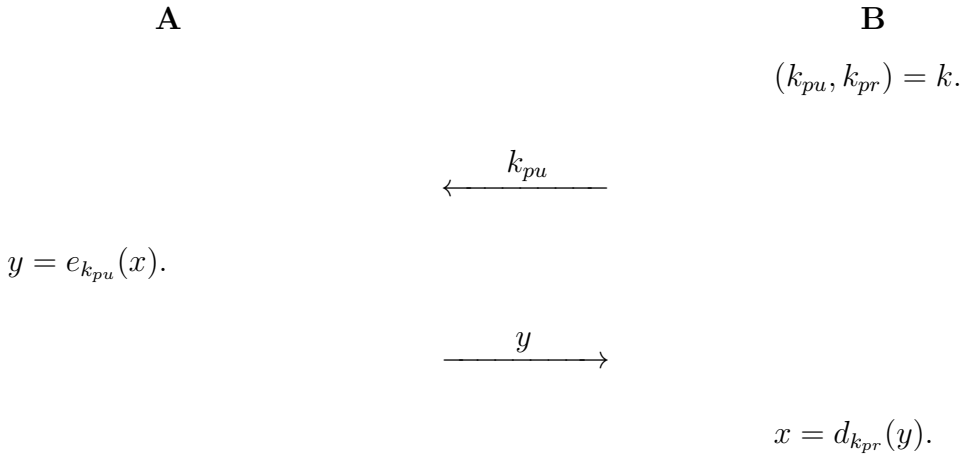
The number of keys required to establish a communication with  $n$  users is:

$$\frac{n(n-1)}{2}$$

because each person needs a different key to communicate with each person. If a sender uses the same key for everybody, any receiver could encrypt and decrypt the messages between the sender and the other receivers.

No protection against cheating means that having communication between a user A and a user B and having a message it is not possible to know who has sent it.

In asymmetric cryptography, the key used for encrypting messages can be public without problems. But the ciphertext is decrypted with a secret key. It means that in PKC there are two keys:  $k_{public}$  and  $k_{private}$ .



Two basic characteristics are: the algorithm used for encryption must be computationally infeasible to invert and the message only can be decrypted with the secret key. PKC is much slower than symmetric cryptography so, once the channel is established through PKC, AES can be used to communicate.

The main security mechanics that PKC enables are:

- **Key establishment** through an insecure channel.
- **Non-repudiation** and message integrity.
- **Identification**
- **Encryption**

The algorithms utilized are based on modular arithmetic and especially on:

- **Integer-Factorization Schemes:** The difficulty to factor large integers (RSA).
- **Discrete Logarithm Schemes:** The discrete logarithm problem in finite fields (DH, DSA, Elgamal).
- **Elliptic Curves Schemes:** A generalization of the discrete logarithm problem (ECDH, ECDSA).

Speaking in terms of security and key lengths, asymmetric cryptography requires longer keys than symmetric cryptography.

Algorithm family	Cryptosystems	Security level (bits)			
Symmetric key	AES, 3DES	80 bits	128 bits	192 bits	256 bits
Integer Factorization	RSA	1024 bits	3072 bits	7680 bits	15360 bits
Discrete Logarithm	DH, DSA, Elgamal	1024 bits	3072 bits	7680 bits	15360 bits
Elliptic Curves	ECDH, ECDSA	160 bits	256 bits	384 bits	512 bits

Table 1.7: Bit lengths of public-key algorithms for different security levels

## 1.6 RSA

RSA algorithm belongs to asymmetric cryptography. RSA is based on the integer factorization problem. The problem is that multiplying two primes is computationally easy whereas factoring the product is very hard.

### 1.6.1 Encryption and decryption

In asymmetric cryptography, there is a public key and a private key. In RSA the public key is  $k_{pu} = (n, e)$  and the private key is  $k_{pr} = d$ . RSA encrypts plaintexts  $x$  with a lower binary value than  $n$ . The numbers  $p$  and  $q$  are two primes number. As  $n$  should have a length equal or bigger than 1024 bits,  $p$  and  $q$  should have 512 bits or more.

$$n = p \cdot q$$

The value of  $e$  must satisfy the condition:

$$\Phi(n) = (p - 1) \cdot (q - 1)$$

$$\gcd(\Phi(n), e) = 1$$

it means that the inverse of  $e$  exists and it is the private key.

$$d = e^{-1} \mod \Phi(n)$$

RSA encrypts and decrypts as follow:

$$y = e_{k_{pu}}(x) = x^e \mod n$$

$$x = d_{k_{pr}}(y) = y^d \mod n$$

RSA must meet the following characteristics:

- For a given public key  $k_{pu} = (n, e)$  must be computationally infeasible to determine the private key  $k_{pr} = d$ .

- Plaintexts with more bit than the length of  $n$  cannot be encrypted.
- To calculate exponentiation with large numbers should be easy.
- There should be many private key/public key pairs for a value of  $n$ .

For instance:

**A**

**B**

Plaintext  $x = 2$ .

Choose two primes  $p = 5$   $q = 11$ .

$$n = 5 \cdot 11 = 55.$$

$$\Phi(n) = (5 - 1) \cdot (11 - 1) = 40.$$

Choose  $e = 7$ .

Verify  $\gcd(\Phi(n), e) = \gcd(40, 7) = 1$ .

$$d = e^{-1} \bmod \Phi(n) = 7^{-1} \bmod 40 = 23.$$

$$\xleftarrow{k_{pu} = (n, e) = (55, 7)}$$

$$y = x^e \bmod n = 2^7 \bmod 55 = 18.$$

$$\xrightarrow{y = 18}$$

$$x = y^d \bmod n = 18^{23} \bmod 55 = 2.$$

Public key e	e as a binary string
3	$11_2$
17	$10001_2$
$2^{16} + 1$	$10000000000000001_2$

Table 1.8: RSA exponentiation with short public exponents

Even the values of  $p$  and  $q$  should have 512 bits at least, the value of  $e$  can be a small value. The most common values for  $e$  are  $e = 3$ ,  $e = 17$  and  $e = 2^{16} + 1$ . These values enable fast exponentiation.

### 1.6.2 RSA in practice

The RSA described so far has some drawbacks.

- RSA encryption for a key is deterministic. It means that given a plaintext, the ciphertext always will be the same.
- Plaintexts  $x = 0$ ,  $x = 1$  and  $x = -1$  has as ciphertext 1, 0 or  $-1$ .
- Without padding small plaintexts and small public key exponent might be attacked but, there is not any attack discovered against  $e = 3$ .
- An attacker can manipulate the ciphertext by replacing it with a past ciphertext.

To solve these problems RSA padding is the solution. Being  $M$  the message,  $k$  the length of the modulus  $n$  in bytes,  $|H|$  the length of the hash function output in bytes and  $|M|$  the length of the message in bytes. Padding consist of:

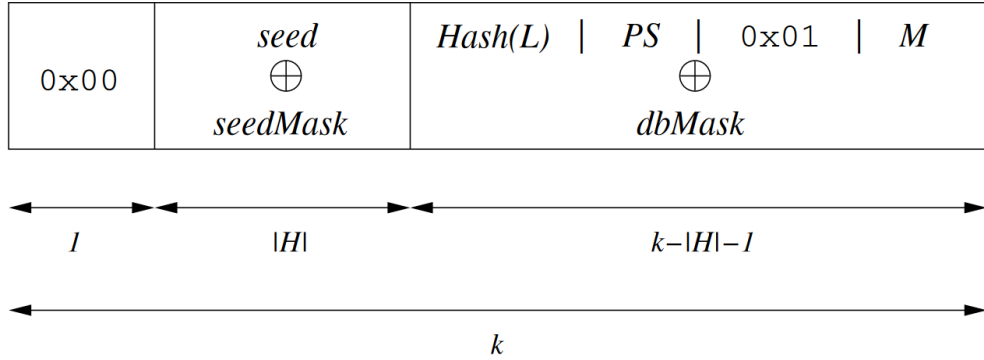


Figure 1.12: RSA encryption of a message M with padding

1. Generate a string PS of zeroed bytes which length is  $k - |M| - 2|H| - 2$ .
2. Concatenate  $Hash(L)$ ,  $PS$ , a single byte with hexadecimal value 0x01, and  $M$  to form a data block DB of length  $k - |H| - 1$  byte as:

$$DB = Hash(L) || PS || 0x01 || M$$

3. Generate a random byte string seed of length  $|H|$ .

4. Let  $dbMask = MGF(seed, k - |H| - 1)$ , where  $MGF$  is the mask generation function.
5. Let  $maskedDB = DB \oplus dbMask$ .
6. Let  $seedMask = MGF(maskedDB, |H|)$ .
7. Let  $maskedSeed = seed \oplus seedMask$ .
8. Concatenate a single byte with hexadecimal value  $0x00$ ,  $maskedSeed$  and  $maskedDB$  to form an encoded message  $EM$  of length  $k$  bytes as:

$$EM = 0x00 || maskedSeed || maskedDB$$

## 1.7 Discrete Logarithm Problem (DLP) and Diffie–Hellman Key Exchange (DHKE)

Discrete Logarithm Problem (DLP) is based on the fact that computing discrete logarithms with modulo prime is a very hard problem if the parameters are sufficiently large. DLP belongs to asymmetric cryptography.

### 1.7.1 Discrete Logarithm Problem (DLP)

The problem is started with a finite cyclic group  $\mathbb{Z}_p^*$  of order  $p - 1$  where  $p$  is prime and a primitive element  $\alpha \in \mathbb{Z}_p^*$  and another element  $\beta \in \mathbb{Z}_p^*$ . The DLP is the problem of determining the integer  $1 \leq x \leq p - 1$  such that:

$$\alpha^x = \beta \mod p$$

$$x = \log_{\alpha} \beta \mod p$$

For example, considering the cyclic group  $\mathbb{Z}_{59}^*$ , a primitive element  $\alpha = 2$  and  $\beta = 30$ :

$$2^x = 30 \mod 59$$

$$x = \log_2 30 \mod 59$$

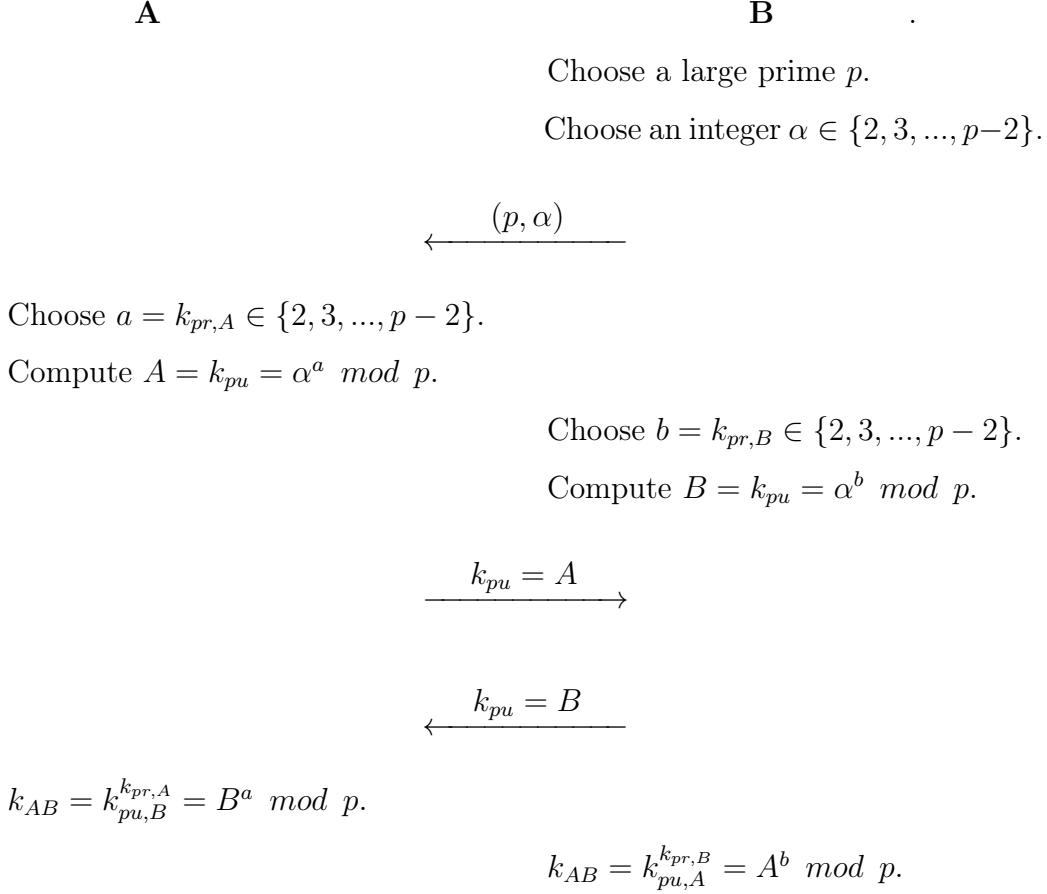
For small numbers, the value of  $x$  can be determined by using a brute force attack.

$$2^{18} = 262144 = 30 \mod 59$$

Using numbers with a length equal or bigger than 1024 bits, the DLP is a one-way function. It means, for a given  $p$ ,  $\alpha$  and  $\beta$  must be computationally infeasible to determine the value of  $x$ .

### 1.7.2 Diffie–Hellman Key Exchange (DHKE)

The Diffie–Hellman Key Exchange (DHKE) is an application of the DLP which enables two parties to derive a common secret key by communicating over an insecure channel.



Both parties share the same key  $k_{AB}$  because:

$$B^a = (\alpha^b)^a = \alpha^{ab} \bmod p$$

$$A^b = (\alpha^a)^b = \alpha^{ab} \bmod p$$

Before of using  $k_{AB}$  as a symmetric key, it must be verified that  $p$  is prime and has a minimum length of 1024 bits,  $\alpha$  is a primitive element of the group  $\mathbb{Z}_p^*$  and the private keys  $a$  and  $b$  are generated with a random generator.



For instance,  $p = 59$  and  $\alpha = 2$

**A**

**B**

.

Choose  $a = k_{pr,A} = 3$ .

Compute  $A = k_{pu} = 2^3 \bmod 59 =$   
 $= 8 \bmod 59$ .

Choose  $b = k_{pr,B} = 4$ .

Compute  $B = k_{pu} = 2^4 \bmod 59 =$   
 $= 16 \bmod 59$ .

$$\xrightarrow{k_{pu} = A = 8}$$

$$\xleftarrow{k_{pu} = B = 16}$$

$$k_{AB} = k_{pu,B}^{k_{pr,A}} = 16^3 \bmod 59 = 25 \bmod 59.$$

$$k_{AB} = k_{pu,A}^{k_{pr,B}} = 8^4 \bmod 59 = 25 \bmod 59.$$



## 1.8 Elliptic Curve Cryptosystems (ECC)

Elliptic Curve Cryptosystems (ECC) are Public Key Cryptographic algorithms (PKC) and are based on the Discrete Logarithm Problem (DLP). ECC has the same level of security that RSA and DLP with shorter operands. As RSA and DLP, ECC are one-way systems.

### 1.8.1 Theory of Elliptic Curves

Given a polynomial equation, an elliptic curve is formed by all the points  $(x, y)$  that fulfil the equation. For cryptographic use, the curve is considered over a finite field. The natural choice is prime fields  $GF(p)$ , where all arithmetic is performed modulo a prime  $p$ .

The elliptic curve over  $\mathbb{Z}_p$ ,  $p > 3$  is the set of pairs  $(x, y) \in \mathbb{Z}_p$  which fulfil:

$$y^2 = x^3 + a \cdot x + b \mod p$$

together with an imaginary point of infinity  $\vartheta$ , where:

$$a, b \in \mathbb{Z}_p$$

and the condition  $4 \cdot a^3 + 27 \cdot b^2 \neq 0$ .

The last condition states that the curve is non-singular. It means that the plot has no self-intersections or vertices. The elliptic curve is symmetric with respect to the  $x$ -axis.

The operation "*addition*" which is defined with the symbol "+" means that given two points  $P$  and  $Q$ , another point  $R$  which belongs to the elliptic curve is computed.

$$P + Q = R$$

$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$$

The addition operation varies if the two points to add are equal or not.

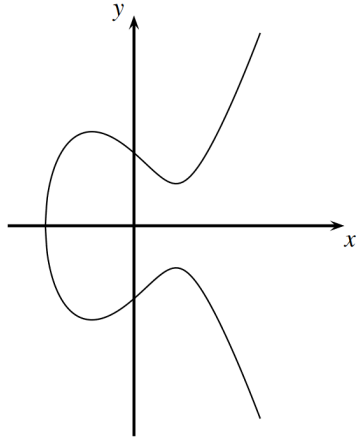


Figure 1.13:  $y^2 = x^3 - 3x + 3$  over  $\mathbb{R}$

- **P+Q:** The addition is computed drawing a line through  $P$  and  $Q$ . This line intersects the curve in a third point. The result of the addition is the symmetric point with respect to the  $x$ -axis of the third intersection.
- **P+P:** To obtain the result, a tangent line through  $P$  must be drawn. The symmetric point with respect to the  $x$ -axis of the intersection through the tangent line and the curve is the result.

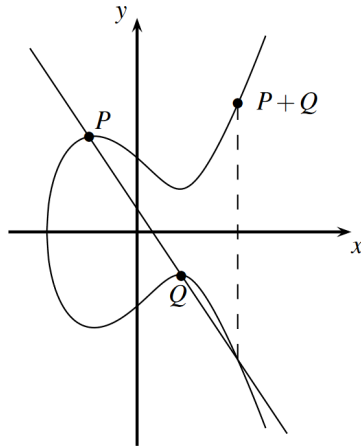


Figure 1.14:  $P + Q$

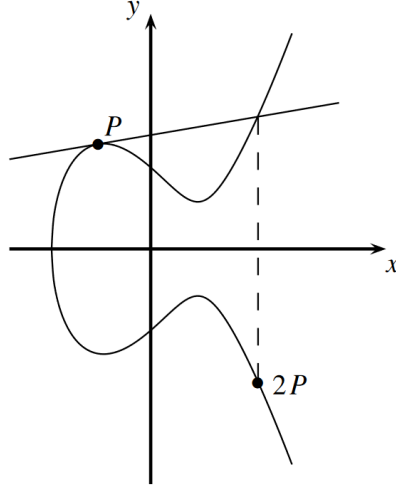


Figure 1.15:  $P + P$

The operation addition described above can be seen from a mathematical point of view as:

$$x_3 = s^2 - x_1 - x_2 \mod p$$

$$y_3 = s(x_1 - x_3) - y_1 \mod p$$

where:

$$s = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} \mod p & ; \text{ if } P \neq Q \\ \frac{3x_1^2 + a}{2y_1} \mod p & ; \text{ if } P = Q \end{cases}$$

Also, a neutral element  $\vartheta$  is established:

$$P + \vartheta = P$$

$$P + (-P) = \vartheta$$

$$-P = (-x_p, -y_p) = (-x_p, p - y_p)$$

For example, given the elliptic curve  $y^2 = x^3 + 3x + 2 \mod 7$  and the point  $P = (0, 3)$ , derive the points  $2P$  and  $3P$ .

First, a verification that  $P$  belongs to the elliptic curve is done.

$$9 = 0 + 0 + 2 \mod 7 = 2 \mod 7$$

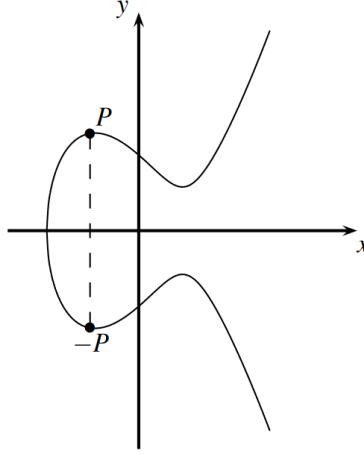


Figure 1.16:  $P$  and  $-P$

Then,  $2P$  can be calculated as  $P + P$ :

$$2P = P + P = (0, 3) + (0, 3)$$

$$s_{2P} = \frac{3x_1^2 + a}{2y_1} \bmod p = \frac{3 \cdot 0^2 + 3}{2 \cdot 3} = 3 \cdot (6)^{-1} \bmod 7 = 3 \cdot 6 \bmod 7 = 4 \bmod 7$$

$$x_{2P} = 4^2 - 0 - 0 \bmod 7 = 2 \bmod 7$$

$$y_{2P} = 4(0 - 2) - 3 \bmod 7 = -11 \bmod 7 = 3 \bmod 7$$

$$2P = (2, 3)$$

and  $3P$  is calculated as  $2P + P$ :

$$s_{3P} = \frac{y_2 - y_1}{x_2 - x_1} = \frac{3 - 3}{2 - 0} = 0 \cdot (2)^{-1} \bmod 7 = 0 \bmod 7$$

$$x_{3P} = 0^2 - 0 - 2 \bmod 7 = -2 \bmod 7 = 5 \bmod 7$$

$$y_{3P} = 0(0 - 5) - 3 \bmod 7 = -3 \bmod 7 = 4 \bmod 7$$

$$3P = (5, 4)$$

### 1.8.2 The Discrete Logarithm Problem with Elliptic Curves

The points on an elliptic curve together with  $\vartheta$  have cyclic subgroups. Under certain conditions, all points on an elliptic curve form a cyclic group.

Continuing with the previous example, where the elliptic curve is  $y^2 = x^3 + 3x + 2 \pmod{7}$  and the point is  $P = (0, 3)$ .

$$P = (0, 3)$$

$$2P = (2, 3)$$

$$3P = (5, 4)$$

$$4P = (4, 6)$$

$$5P = (4, 1)$$

$$6P = (5, 3)$$

$$7P = (2, 4)$$

$$8P = (0, 4)$$

$$9P = \vartheta$$

$$10P = (0, 3) = P$$

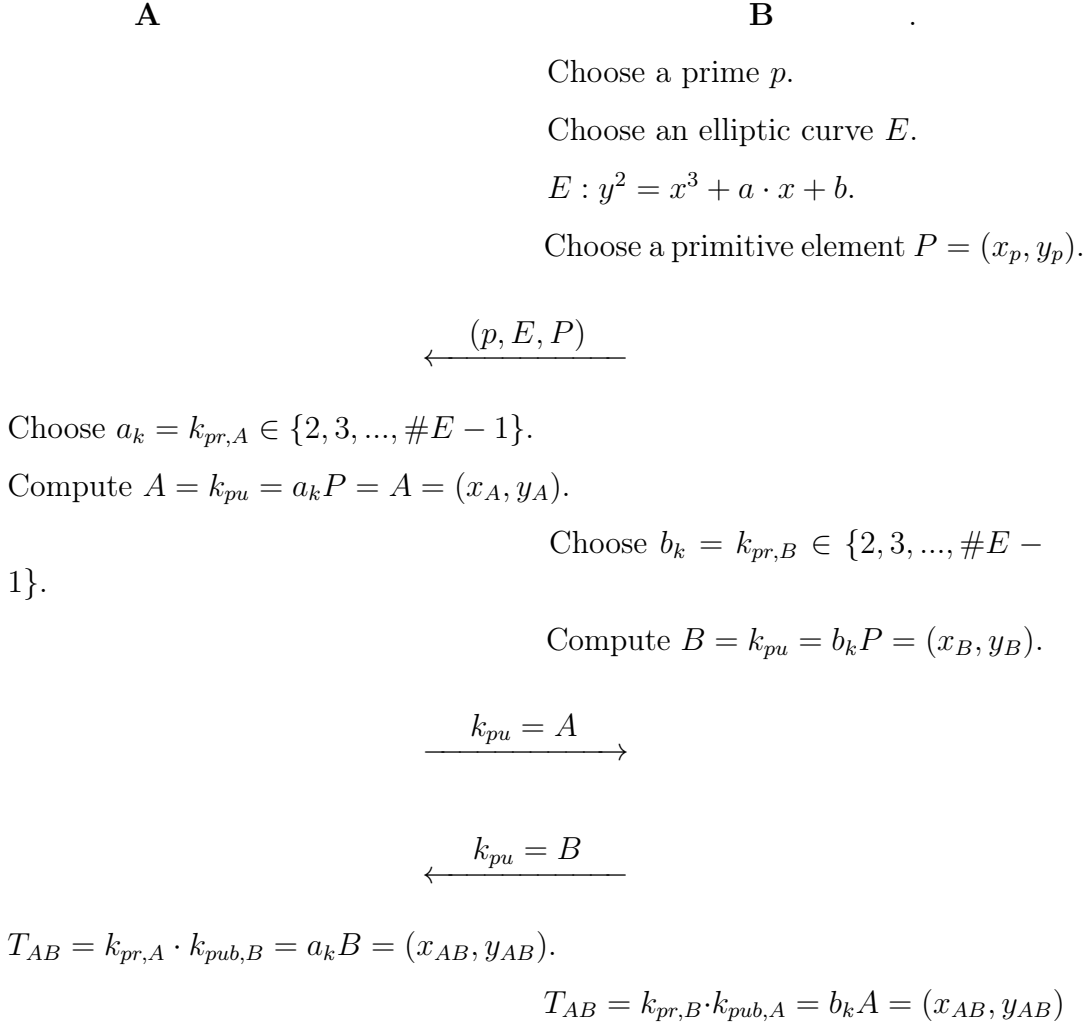
These points are part of a cyclic group. The order of this cyclic group is  $\#E = 9$  because the point  $10P$  is equal than  $P$ . The order is represented with  $\#E$ .

The Discrete Logarithm Problem with Elliptic Curves can be described as given an elliptic curve  $E$  and considering a primitive element  $P$  and another element  $T$ . The Discrete Logarithm Problem is finding the integer  $d$ , where  $1 \leq d \leq \#E$ , such that:

$$dP = T$$

### 1.8.3 Diffie–Hellman Key Exchange (DHKE) with Elliptic Curves

Also, the DHKE can be used with Elliptic Curves and the process is quite similar.



Both parties share the same key  $T_{AB}$  because:

$$a_k B = a_k (b_k P) = a_k b_k P$$

$$b_k A = b_k (a_k P) = a_k b_k P$$



For instance,  $p = 7$ ,  $E : y^2 = x^3 + 3x + 2$  and  $P = (0, 3)$ .

**A**

**B**

.

Choose  $a_k = k_{pr,A} = 3$ .

Compute  $A = k_{pu} = 3P = (5, 4)$ .

Choose  $b_k = k_{pr,B} = 4$ .

Compute  $B = k_{pu} = 4P = (4, 6)$ .

$$\xrightarrow{k_{pu} = A = (5, 4)}$$

$$\xleftarrow{k_{pu} = B = (4, 6)}$$

$$T_{AB} = a_k B = 3(4, 6) = (5, 4).$$

$$T_{AB} = b_k A = 4(5, 4) = (5, 4).$$



## 1.9 Hash Functions

Hash Functions digest messages of any size producing fixed-length output. Output length varies between 128 and 512 bits. Hash Functions do not require keys and are easy to compute. It means that for an input, the output will be always the same. The output is very sensitive to any change in the input.

The length of the input does not change the length of the output, here are three examples. The only difference between the second and third example is a character and the difference is noticeable.

↓

7CDD6D5083F029DD2870F395894A32A3

The budget is a million.

↓

1C2C95D5AE6EDE25134BFD0405AF5B02

The budget is a billion.

↓

51F1F275741BCA8795221C508C685B58

As hash functions do not have keys. These should meet three characteristics.

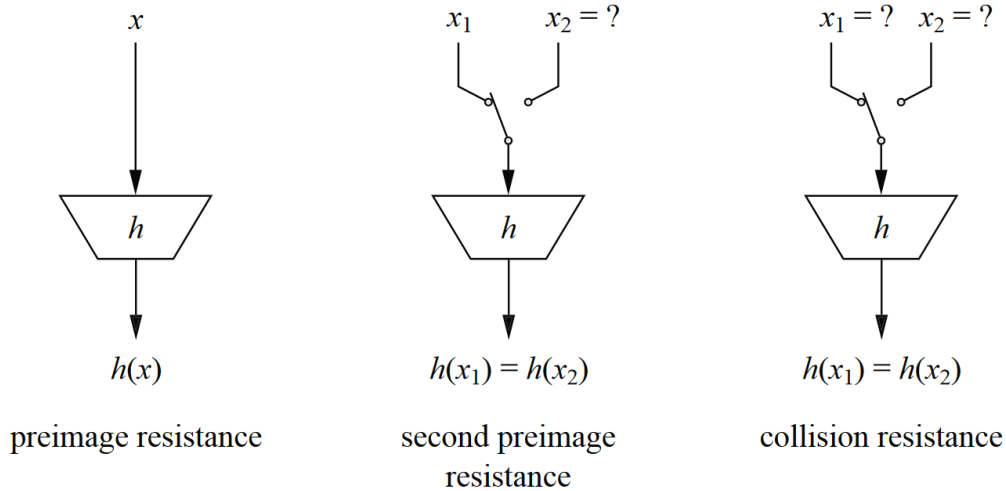


Figure 1.17: Characteristics of hash functions

### 1.9.1 Preimage resistance

Preimage resistance indicates that given the output of the hash  $z = h(x)$  it should be computationally infeasible to obtain the message  $x$ . The hash function has to be one-way.

### 1.9.2 Second preimage resistance

Second preimage resistance means that given a message  $x_1$ . It is computationally infeasible to find another message  $x_2$  that verify  $h(x_1) = h(x_2)$ . As the output has a fixed length whereas the input can have any length, the number of possible inputs is greater than the number of possible outputs. It means that some inputs share the same output. A person could find it with a process similar to a key exhaustive search. To avoid it the length of the output should be at least 80 bits.

### 1.9.3 Collision resistance

Collision resistance is similar to second preimage resistance. To find two messages  $x_1, x_2$  that have the same output  $h(x_1) = h(x_2)$ . Note that in the collision resistance, the freedom has increased because  $x_1$  and  $x_2$  can vary. In the second preimage resistance only  $x_2$  can change. The probability for no collision in  $t$  messages is calculated as:

$$\begin{aligned} P(\text{no collision}) &= \left(1 - \frac{1}{2^n}\right) \left(1 - \frac{2}{2^n}\right) \left(1 - \frac{t-1}{2^n}\right) \\ &= \prod_{i=1}^{t-1} \left(1 - \frac{i}{2^n}\right) \end{aligned}$$

$$e^{-x} \approx 1 - x$$

$$P(\text{no collision}) \approx \prod_{i=1}^{t-1} e^{-\frac{i}{2^n}} \approx e^{-\frac{1+2+3+\dots+t-1}{2^n}}$$

$$1 + 2 + 3 + \dots + t - 1 = t(t-1)/2$$

$$P(\text{no collision}) \approx e^{-\frac{t(t-1)}{2^{n+1}}}$$

The probability of at least one collision is:

$$\lambda = 1 - P(\text{no collision})$$

$$\lambda = 1 - e^{-\frac{t(t-1)}{2^{n+1}}}$$

$$\ln(\lambda - 1) \approx -\frac{t(t-1)}{2^{n+1}}$$

$$t(t-1) \approx 2^{n+1} \ln \left( \frac{1}{1-\lambda} \right)$$

As  $t$  is the number of messages and it is a natural number  $t \gg 1$ .

$$t \approx 2^{(n+1)/2} \sqrt{\ln \left( \frac{1}{1-\lambda} \right)}$$

The numbers of messages  $t$  require to find at least one collision in a hash function 80 bits output with a probability of 50% is:

$$t \approx 2^{(80+1)/2} \sqrt{\ln \left( \frac{1}{1-0.5} \right)} \approx 2^{40.2}$$

This shows that the number of messages hashed needed to find a collision is roughly equal to  $2^{n/2}$ .

Current laptops can compute  $2^{40}$  without any difficulty. The length of the output should be increased to avoid it.

To sum up, the requirements for hash functions are:

1. Arbitrary message size.
2. Fixed output length.
3. Easy to compute.
4. Preimage resistance.
5. Second preimage resistance.
6. Collision resistance.

A hash function can be constructed in various ways. Speaking in terms of hash functions types, exist two ways:

- **Dedicated hash functions:** These are functions designed explicitly to hash messages such as the MD4 family.
- **Block cipher-based hash functions:** Also block ciphers can be used for hashing messages.





## 1.10 SHA 256 algorithm

SHA 256 is a hash function which belongs to dedicated hash functions. In this section, the algorithm that uses SHA 256 to digest messages will be explained. Notice that the size of the message is arbitrary, the length of the output is fixed, hash functions are easy to compute and should have preimage resistance, second preimage resistance, collision resistance. SHA 256 has a 256 bits output length. The algorithm is divided into preprocessing and hash computation.

### 1.10.1 Preprocessing

First of all, as the length of the input depends on the message, it is necessary to convert the size of the input in a multiple of 512 bits. To do it, padding will be applied. Padding is performed adding a '1' bit, then adding K '0' bits and the last 64 bits are the length of the message in binary.

$$L + 1 + K + 64 = 512 \cdot n \quad n \in \mathbb{N}$$

$$K = 512 \cdot n - 64 - 1 - L \quad n \in \mathbb{N}$$

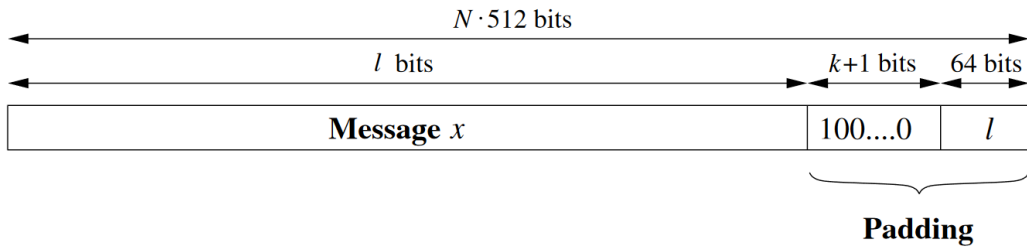


Figure 1.18: Padding of a message

For instance, padding is shown with the message 'Hello'. Each character has 8 bits so, the length of the message is 40 bits.

$$H_{ASCII} = 01001000_2$$

$$e_{ASCII} = 01100101_2$$

$$l_{ASCII} = 01101100_2$$

$$l_{ASCII} = 01101100_2$$

$$o_{ASCII} = 01101111_2$$

$$K = 512 - 64 - 1 - 40 = 407$$

$$\underbrace{01001000}_H \underbrace{01100101}_e \underbrace{01101100}_l \underbrace{01101100}_l \underbrace{01101111}_o 1 \underbrace{0 \dots 0}_{407 \text{ zeros}} \underbrace{0 \dots 00101000}_{l=40}$$

One that the message has been padded, it is the time to define constants.

$$h_0 = 0x6a09e667$$

$$h_1 = 0xbb67ae85$$

$$h_2 = 0x3c6ef372$$

$$h_3 = 0xa54ff53a$$

$$h_4 = 0x510e527f$$

$$h_5 = 0x9b05688c$$

$$h_6 = 0x1f83d9ab$$

$$h_7 = 0x5be0cd19$$

$$K = \begin{pmatrix} 0x428a2f98 & 0x71374491 & 0xb5c0fbcf & 0xe9b5dba5 \\ 0x3956c25b & 0x59f111f1 & 0x923f82a4 & 0xab1c5ed5 \\ 0xd807aa98 & 0x12835b01 & 0x243185be & 0x550c7dc3 \\ 0x72be5d74 & 0x80deb1fe & 0x9bdc06a7 & 0xc19bf174 \\ 0xe49b69c1 & 0xefbe4786 & 0x0fc19dc6 & 0x240ca1cc \\ 0x2de92c6f & 0x4a7484aa & 0x5cb0a9dc & 0x76f988da \\ 0x983e5152 & 0xa831c66d & 0xb00327c8 & 0xbf597fc7 \\ 0xc6e00bf3 & 0xd5a79147 & 0x06ca6351 & 0x14292967 \\ 0x27b70a85 & 0x2e1b2138 & 0x4d2c6dfc & 0x53380d13 \\ 0x650a7354 & 0x766a0abb & 0x81c2c92e & 0x92722c85 \\ 0xa2bfe8a1 & 0xa81a664b & 0xc24b8b70 & 0xc76c51a3 \\ 0xd192e819 & 0xd6990624 & 0xf40e3585 & 0x106aa070 \\ 0x19a4c116 & 0x1e376c08 & 0x2748774c & 0x34b0bcb5 \\ 0x391c0cb3 & 0x4ed8aa4a & 0x5b9cca4f & 0x682e6ff3 \\ 0x748f82ee & 0x78a5636f & 0x84c87814 & 0x8cc70208 \\ 0x90befffa & 0xa4506ceb & 0xbef9a3f7 & 0xc67178f2 \end{pmatrix}$$

Here  $K(0, 1, \dots, 63)$  has been shown as a matrix but it is an array. For example,  $K(6) = 0x923f82a4$ .

Another array  $W(0, 1, \dots, 63)$  is created, it is an array of words where each word has 32 bits. The initial values of the array  $W(0, 1, \dots, 63)$  are zero.

### 1.10.2 Hash computation

The first 16 words contain the message padded. As each word has 16 bits and there are 16 words, this makes a total of 512 bits.

In the example:

$$\begin{aligned} W(0) &= 01001000011001010110110001101100_2 \\ W(1) &= 01101111110000000000000000000000_2 \\ W(2, 3, \dots, 14) &= 00000000000000000000000000000000_2 \\ W(15) &= 00000000000000000000000000000101000_2 \end{aligned}$$

Before the algorithm will be explained, it is important to understand the meaning of  $\ggg$  and  $\gg$ . The symbol  $\ggg$  denotes a right rotation and  $\gg$  a right shift. In the right rotation  $\ggg$ , bits are shifted to the right and the least significant bit is moved to the most significant bit. In the right shift, the least significant bit is deleted and a '0' is put in the most significant bit.

$$0110010_{\ggg 1} = 0011001$$

$$0110010_{\ggg 2} = 1001100$$

$$0110010_{\gg 1} = 0011001$$

$$0110010_{\gg 2} = 0001100$$

The symbols  $\oplus$ ,  $\otimes$  and  $\bar{x}$  denote the operations XOR, AND and NOT respectively. The algorithm is:

for i from 16 to 63:

$$S_0 = (W(i - 15)_{\ggg 7}) \oplus (W(i - 15)_{\ggg 18}) \oplus (W(i - 15)_{\gg 3})$$

$$S_1 = (W(i - 15)_{\ggg 17}) \oplus (W(i - 15)_{\ggg 19}) \oplus (W(i - 15)_{\gg 10})$$

$$W(i) = W(i - 16) + S_0 + W(i - 7) + S_1$$

end for

$$a = h_0 \quad b = h_1 \quad c = h_2 \quad d = h_3 \quad e = h_4 \quad f = h_5 \quad g = h_6 \quad h = h_7$$

for i from 0 to 63:

$$S_1 = e_{\ggg 6} \oplus e_{\ggg 11} \oplus e_{\ggg 25}$$

$$ch = (e \otimes f) \oplus (\bar{e} \otimes g)$$

$$temp_1 = h + S_1 1 + ch + K(i) + W(i)$$

$$S_0 = a_{\ggg 2} \oplus a_{\ggg 13} \oplus a_{\ggg 22}$$

$$maj = (a \otimes b) \oplus (a \otimes c) \oplus (b \otimes c)$$

$$temp_2 = S_0 + maj$$

```

    h = g
    g = f
    f = e
    e = d + temp1
    d = c
    c = b
    b = a
    a = temp1 + temp2

```

end for

```

h0 = h0 + a
h1 = h1 + b
h2 = h2 + c
h3 = h3 + d
h4 = h4 + e
h5 = h5 + f
h6 = h6 + g
h7 = h7 + h

```

return

```

hash=h0 append h1 append h2 append h3 append h4 append h5 append h6
append h7

```

Another point of view of the same algorithm is shown below with the information included in Figure 1.19.

$$\begin{aligned}
 W(i) &= M_i & i &= 0, 1, \dots, 15 \\
 W(i) &= \sigma_1(W(i-2)) + W(i-7) + \sigma_0(W(i-15)) + W(i-16) & i &= 16, \dots, 63 \\
 \sigma_0(X) &= X_{>>>7} \oplus X_{>>>18} \oplus X_{>>3} \\
 \sigma_1(X) &= X_{>>>17} \oplus X_{>>>19} \oplus X_{>>10} \\
 f_0 &= (X, Y, Z) = (X \times Y) \oplus (Y \otimes Z) \oplus (X \otimes Z)
 \end{aligned}$$

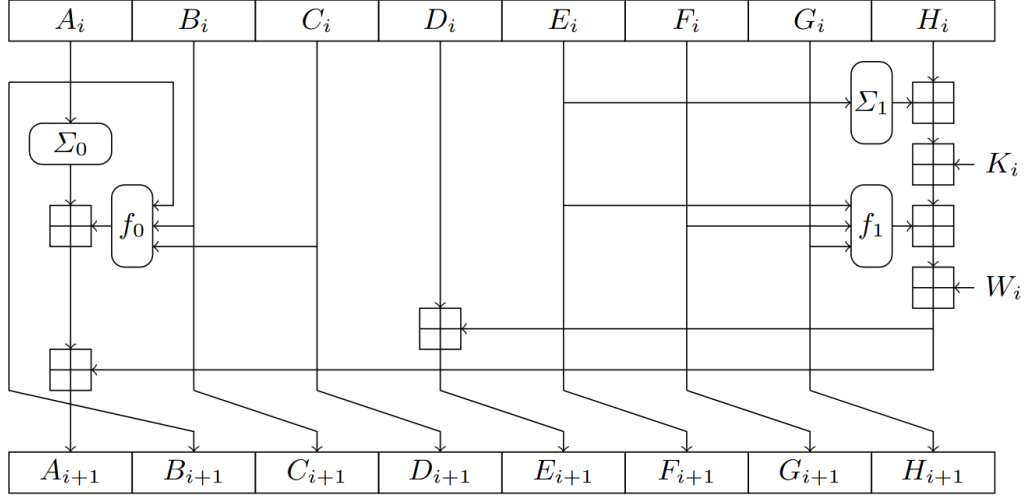


Figure 1.19: Hash computation

$$f_1 = (X, Y, Z) = (X \times Y) \oplus (\bar{X} \otimes Z)$$

$$\sum 0(X) = X_{\ggg 2} \oplus X_{\ggg 13} \oplus X_{\ggg 22}$$

$$\sum 1(X) = X_{\ggg 6} \oplus X_{\ggg 11} \oplus X_{\ggg 25}$$

## 1.11 PBKDF

For the security of cryptographic applications is essential keys with a high degree of randomness. Sometimes the keys are given by users and these do not have enough entropy. An example of it is shown in Table 1.9.

Rank	2015	2016	2017	2018	2019
1	123456	123456	123456	123456	123456
2	password	password	password	password	123456789
3	12345678	12345	12345678	123456789	qwerty
4	qwerty	12345678	qwerty	12345678	password
5	12345	football	12345	12345	1234567
6	123456789	qwerty	123456789	111111	12345678
7	football	1234567890	letmein	1234567	12345
8	1234	1234567	1234567	sunshine	iloveyou
9	1234567	princess	football	qwerty	111111
10	baseball	1234	iloveyou	iloveyou	123123

Table 1.9: Top 10 most common passwords by year according to Splash Data

Password-based key derivation functions (PBKDFs) derive cryptographic keys from passwords or passphrases. To do it, PBKDFs need as inputs a hash function, a key, a random number and the number of iterations. With this information PBKDFs derive a key. Note that the input key can be a key with low entropy.

$$MK = PBKDF(PRF, P, S, C)$$

where:

$MK \rightarrow$  *Derived Key*

$PRF \rightarrow$  *Hash Function*

$P \rightarrow$  *Input Key or password*

$S \rightarrow$  *Salt : Random Number*

$C \rightarrow$  *Number of iterations*

The algorithm that executes PBKDF is easy and fast to compute. It is explained below:

$$\begin{aligned}
U_1 &= PRF(P, S || int(j)) \\
U_i &= PRF(P, U_{i-1}) \quad i = 2, 3, \dots, C \\
T_j &= U_1, U_2, \dots, U_C \quad j = 1, 2, \dots, l \\
MK &= [T_1, T_2, \dots, T_l]
\end{aligned}$$

As the length of  $U_i$  is the length of hash function output and  $T_j$  has the same size that  $U_i$ , the value of  $l$  can be calculated with the size of hash function output and the length desired for the MK.

$$l = \frac{Length(MK)}{Length(Output \ Hash \ Function)}$$

The abbreviation  $int(j)$  represents the 32-bits encoding of integer  $j$ .

$$\begin{aligned}
int(0) &= 00000000000000000000000000000000_2 \\
int(5) &= 00000000000000000000000000000000101_2
\end{aligned}$$

Here is shown an example using:

$$\begin{aligned}
PRF &= SHA\ 256 \\
P &= 'Politecnico di Torino' \\
S &= F0AF0EE1_{hex} \\
C &= 3 \\
Length(MK) &= 512 - bits
\end{aligned}$$

Notice that the length of the output of SHA 256 is 256 bits.

$$l = \frac{Length(MK)}{Length(Output \ Hash \ Function)} = \frac{512bits}{256bits} = 2$$





[illegible]

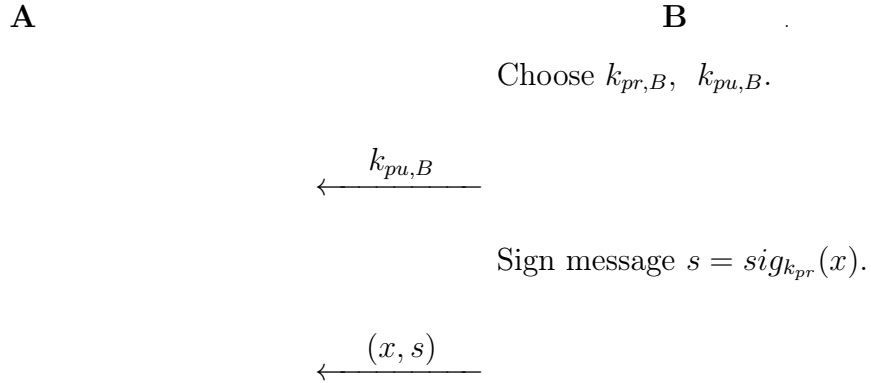
$$U_3 = SHA256(Politecnico di Torino', U_2)$$

$$T_2 = U_1 \oplus U_2 \oplus U_3$$

$$MK = [T1 || T2]$$

## 1.12 Digital Signatures

The Digital Signature is a tool widely used in cryptography. It enables to verify if a message was generated by the sender. The protocol to achieve that is described below and it is basen on Public Key Cryptography.

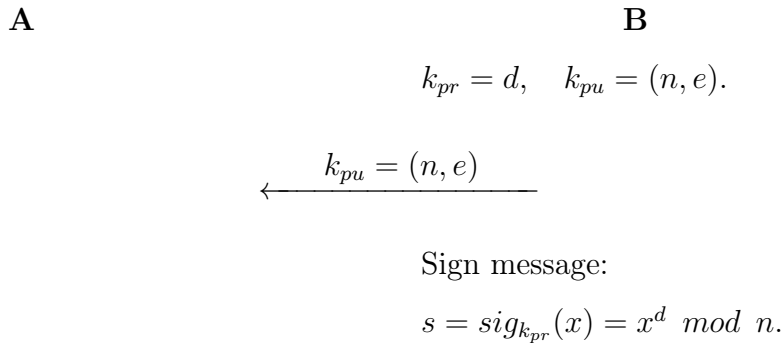


Verify signature:

$$ver_{k_{pu,B}}(x, s) = true/false.$$

### 1.12.1 The RSA Signature Scheme

The protocol to sign a message using the algorithm explained in 1.6.1 is:



$$\xleftarrow{(x, s)}$$

Verify signature:

$$ver_{k_{pu}, B}(x, s) = true/false.$$

$$x' = s^e \bmod n.$$

$$x' \begin{cases} = x \bmod n ; & \text{valid signature} \\ \neq x \bmod n ; & \text{invalid signature} \end{cases}$$

If the message is unaltered,  $x'$  must be equal than  $x$ :

$$x' = s^e = (x^d)^e = x^{de} = x$$

A practical example:

**A**

**B**

$$k_{pr} = 23 \quad k_{pu} = (55, 7).$$

$$\xleftarrow{k_{pu} = (55, 7)}$$

Sign message  $x = 2$ :

$$s = sig_{k_{pr}}(x) = 2^{23} \bmod 55 = 8.$$

$$\xleftarrow{(x, s) = (2, 8)}$$

Verify signature:

$$x' = 8^7 \bmod 55 = 2.$$

$$x' = x \text{ valid signature.}$$

This method has the same drawbacks shown in 1.6.1. This can be avoided using the padding scheme. Being  $M$  the message and  $n$  the length of the modulus RSA in bits, to perform the method:

1. Generate a random value *salt*.
2. Concatenate fixed padding (*padding<sub>1</sub>*), the message hashed (*mhash* =  $h(M)$ ) and *salt* to obtain *M'*.
3. Calculate the hash of *M'*.
4. Concatenate fixed padding (*padding<sub>2</sub>*) and *salt* to obtain *DB*.
5. Apply a mask generation function, often a hash function, to the hashed value of *M'* and obtain *dbMask*.
6. XOR *DB* and *dbMask* to compute *maskedDB*.
7. Compute the encoded message *EM* concatenating *maskedDB*, the hashed value of *M'* and the fixed padding *bc*.

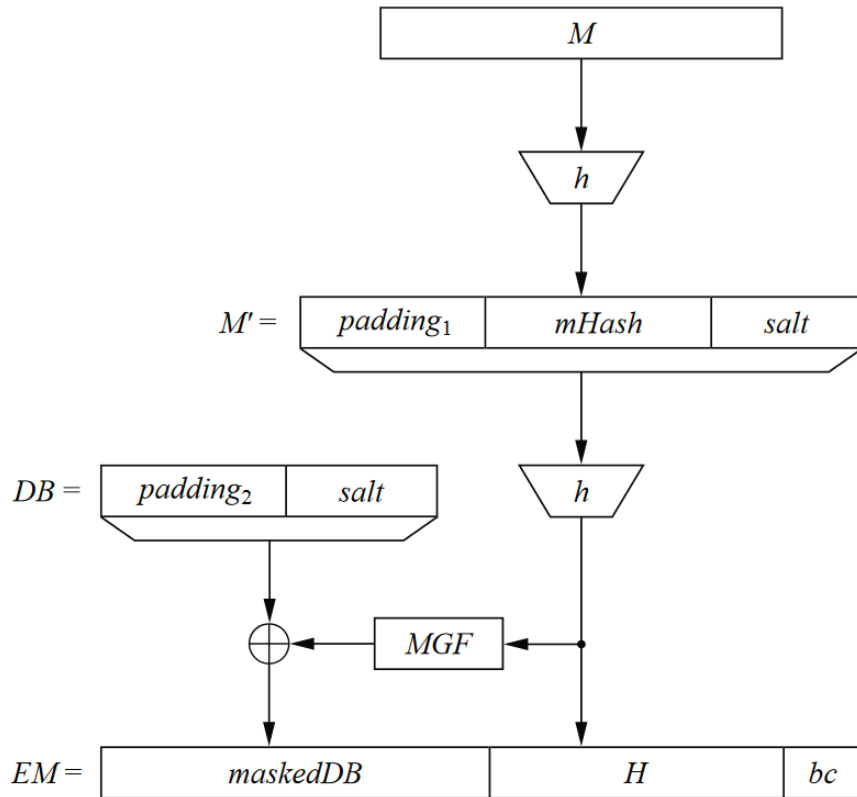


Figure 1.21: RSA encryption of a message  $M$  with padding

Once the encoded message  $EM$  is computed, the digital signature is calculated as:

$$s = sig_{k_{pr}}(x) = EM^d \bmod n$$

To verify the signature, the second party do the same process with the values  $salt$ ,  $padding_1$ ,  $padding_2$  and  $bc$  to obtain  $EM'$  and check if  $EM$  is equal than  $EM'$  where:

$$EM' = EM^e \bmod n$$

$$EM' \begin{cases} = EM \bmod n & ; \text{ valid signature} \\ \neq EM \bmod n & ; \text{ invalid signature} \end{cases}$$

### 1.12.2 A complete example

Note that the message can be encrypted using symmetric cryptography with a key shared previously through asymmetric cryptography. Adding it, the protocol is completed. In the example, Elliptic Curves with Diffie–Hellman Key Exchange, AES, SHA 256 and PBKDF are used. This is the most ambitious example written in this paper so far.

**A**

**B**

.

← Information about the functions used

Choose a prime  $p$ .

Choose an elliptic curve  $E$ .

$$E : y^2 = x^3 + a \cdot x + b.$$

Choose a primitive element  $P = (x_p, y_p)$ .

←  $(p, E, P)$

Choose  $a = k_{pr,A} \in \{2, 3, \dots, \#E - 1\}$ .

Compute  $A = k_{pu} = aP = A = (x_A, y_A)$ .

Choose  $b = k_{pr,B} \in \{2, 3, \dots, \#E - 1\}$ .

Compute  $B = k_{pu} = bP = (x_B, y_B)$ .

$$\xrightarrow{k_{pu} = A}$$

$$\xleftarrow{k_{pu} = B}$$

$T_{AB} = k_{pr,A} \cdot k_{pub,B} = aB = (x_{AB}, y_{AB})$ .

$P = SHA256(x_{AB})$ .

Compute  $salt = RC4RNG(x_{AB})$ .

$MK = PBKDF(SHA256, P, salt, C)$ .

$T_{AB} = k_{pr,B} \cdot k_{pub,A} = bA = (x_{AB}, y_{AB})$ .

$P = SHA256(x_{AB})$ .

Compute  $salt = RC4RNG(x_{AB})$ .

$MK = PBKDF(SHA256, P, salt, C)$ .

$[y_1, \dots, y_n, T, ADD] = GCM-AES_{MK}(x_1, \dots, x_n)$ .

$k_{pr} = d \quad k_{pu} = (n, e)$ .

$$\xleftarrow{k_{pu} = (n, e)}$$

$s = (EM(y_2, \dots, y_n))^d \mod n$ .

$$[y_2, \dots, y_n, T, ADD, s]$$

$[x_2, \dots, x_n, T'] = GCM - AES_{MK}^{-1}([y_2, \dots, y_n, T, ADD])$ .

Check  $T' = T$ .

$$s' = (EM(y_2, \dots, y_n))^e \bmod n.$$

Check  $s' = s$ .



## 1.13 Certificates

In this thesis has been shown in 1.12.2 how communications can be carried out but this requires that each party trust in the other. This is vulnerable to the man in the middle attack. In this subsection is explained in what consists the man in the middle attack and how to avoid it using certificates.

### 1.13.1 Man in the middle attack

Asymmetric cryptography is vulnerable to attacks where a user replaces the public keys sent by both parties with his keys. This is known as man in the middle attack.

<b>A</b>	<b>C</b>	<b>B</b>
$k_{pr,A} = a.$		
$A = k_{pu,A} = \alpha^a \mod n.$		
		$k_{pr,B} = b.$
		$B = k_{pu,B} = \alpha^b \mod n.$
	$\xrightarrow{A} \text{ substitute } \hat{A} = \alpha^o \xrightarrow{\hat{A}}$	
	$\xleftarrow{\hat{B}} \text{ substitute } \hat{B} = \alpha^o \xleftarrow{\hat{B}}$	
$k_{AO} = (\hat{B})^a \mod n.$		$k_{BO} = (\hat{A})^b \mod n.$
	$k_{AO} = A^o \mod n.$	
	$k_{BO} = B^o \mod n.$	

Note that the malicious user (C) has the same keys that the parties A and B.

$$k_{AO} = (\hat{B})^a \bmod n = \alpha^{oa} \bmod n$$

$$k_{AO} = A^o \bmod n = \alpha^{ao} \bmod n$$

$$k_{BO} = (\hat{A})^b \bmod n = \alpha^{ob} \bmod n$$

$$k_{BO} = B^o \bmod n = \alpha^{ob} \bmod n$$

These keys usually are used for the key of symmetric cryptography and the user C could intercept the ciphertexts, decrypt them, encrypt the message that he wants and nobody would notice it.

### 1.13.2 Certificates

Even though public-key schemes do not require a secure channel, these require authenticated channels for the distribution of the public keys. Certificates are used to authenticate and are based on digital signatures. A certificate contains the public key, the identifying information and digital signature of the public key and the identifying information. These certificates travel in the transport layer.

$$Cert_A = [(k_{pu,A}, ID_A), sig_{k_{pr}}(k_{pu,A}, ID_A)]$$

If a user tries to replace the public key, the change will be detected because the digital signature will not be valid.

The signatures for certificates are given by a trusted third party which is called Certification Authority (CA). This CA provides the public and private keys.

Now, the users do not need to trust in another party which is vulnerable to man in the middle attack. They only must trust in the CA. At the same time CA trust in others CA. This is called a chain of trust and there are three types of CA (root, intermediate and end-entity CA).

At the top level is a root CA, who signs his own certificate and also signs the intermediate certificates. An intermediate CA is referenced to a root

CA and signs the end-entity certificates. An end-entity CA is referenced to intermediate CA.

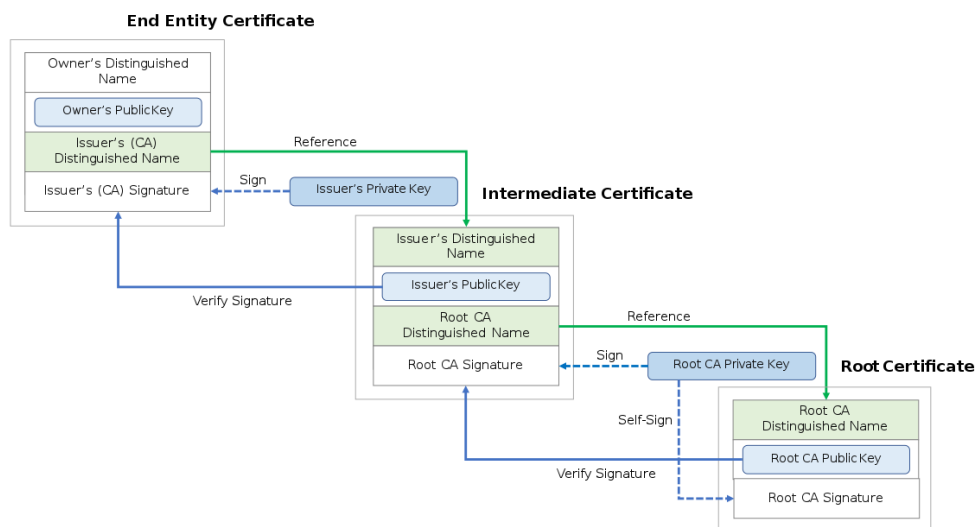


Figure 1.22: Chain of trust

### 1.13.3 X.509

X.509 is the most used certificate on internet connections. The structure of this certificate is:

- **Version:** The version number indicates changes in certificate format over time.
- **Serial number:** The serial number is an identifier for each certificate generated by an issuer. Two different certificates must have two different serial numbers.
- **Signature:** This field points out the algorithm and the parameters needed to sign the certificate.
- **Issuer name:** The name of the CA.

<b>Miscellaneous</b>	
<b>Serial Number</b>	03:FE:2D:9E:83:8E:44:A7:10:34:AD:BD:D3:C3:E1:E2:AE:0F
<b>Signature Algorithm</b>	SHA-256 with RSA Encryption
<b>Version</b>	3
<b>Download</b>	<a href="#">PEM (cert)</a> <a href="#">PEM (chain)</a>

Figure 1.23: Version, serial number and signature

<b>Issuer Name</b>	
<b>Country</b>	US
<b>Organisation</b>	Let's Encrypt
<b>Common Name</b>	<a href="#">Let's Encrypt Authority X3</a>

Figure 1.24: Issuer name

- **Validity period:** Two times and dates that indicate the start and the end of the period when the certificate can be used.

<b>Validity</b>	
<b>Not Before</b>	18/09/2020, 02:42:05 (Central European Standard Time)
<b>Not After</b>	17/12/2020, 01:42:05 (Central European Standard Time)

Figure 1.25: Validity period

- **Subject name:** A representation of its subject's identity in the form of a Distinguished Name.

<b>Subject Name</b>	
<b>Common Name</b>	password.link

Figure 1.26: Subject name

- **Subject of Public Key:** The algorithms and parameters associated with the subject.
- **Fingerprint:** A digest using a hash function of the certificate.

Public Key Info	
<b>Algorithm</b>	RSA
<b>Key Size</b>	2048
<b>Exponent</b>	65537
<b>Modulus</b>	BF:08:2C:DB:C5:2B:9C:9C:E8:64:93:1D:23:2F:DC:B8:5B:9B:96:E7:3C:D6:D4:83:DE:45:8C:7A:21:9E:0F:33:23:63:EA:FA:D6:C C:4D:6A:47:19:FF:CB:65:C0:C9:36:35:5C:5E:CA:F7:A9:8D:D0:8C:0E:BC:6B:62:98:56:FC:24:7A:CA:59:4C:04:0F:C5:A3:CA:56: AB:75:0C:A6:60:2C:E7:CC:9F:38:A3:24:ED:9F:E5:03:1D:F1:DE:8A:82:09:38:16:F6:9A:C8:C6:83:19:58:0D:51:24:78:BD:C5:2F: 28:8F:06:F9:24:0F:78:C7:49:FA:2C:BD:17:28:AB:34:B0:61:AE:4B:5E:23:22:83:88:C2:EE:D7:24:A5:E7:15:7F:4B:C5:8D:D8:0F:8 9:26:CF:DF:7E:F7:2B:50:2C:55:DE:65:D2:52:20:81:8C:12:57:70:C9:9E:B6:4D:CE:87:7D:14:05:04:11:8B:95:3D:C0:77:C6:6E:4 2:B2:EA:9F:07:AE:E0:06:9C:C2:A2:AD:51:C7:89:4A:00:42:A5:87:D2:60:80:BE:2A:9E:6B:5E:0B:28:BA:67:7C:A5:8B:75:D1:40: 69:96:F1:35:33:01:A4:D1:A2:F6:0E:3E:5E:B0:A2:0E:AF:AD:5E:E1:BF:6A:16:DA:78:45:AD:96:01

Figure 1.27: Subject of Public Key

Fingerprints	
<b>SHA-256</b>	61:A4:C7:8D:A8:52:CD:1A:BE:FB:55:DC:48:B5:C3:A8:D8:EE:23:7A:CD:BA:25:C8:7A:2F:A2:B5:C0:9A:9F:F6
<b>SHA-1</b>	32:ED:24:31:74:53:6B:F0:3D:D8:A4:D7:2E:52:1C:6E:85:D2:DB:31

Figure 1.28: Fingerprint



# Chapter 2

## Internet connection

Internet has become a fundamental tool in any field. Internet is used to communicate people, to share documents, to access to the bank, to make transactions, to shop online, to sell, to make money... These connections must be secure otherwise the attacks to stole money and other malicious actions will be a constant and anybody will use internet.

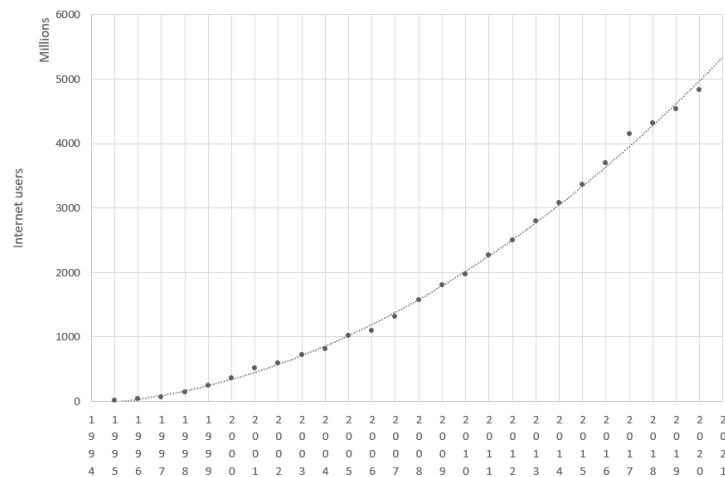


Figure 2.1: Evolution of internet users

The connection between a computer and a server through internet is carried out with the Internet Protocol Suite (TCP/IP). The structure of TCP/IP is explained in this chapter focusing on cryptographic aspects.





## 2.1 Hypertext Transfer Protocol Secure (HTTPS)

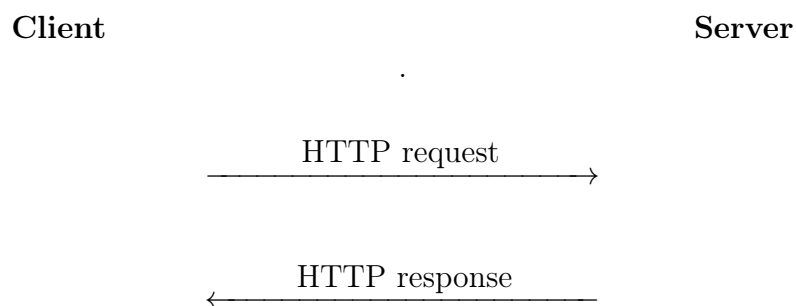
Hypertext Transfer Protocol Secure (HTTPS) is a combination of Hypertext Transfer Protocol (HTTP) and Transport Layer Security (TLS).



Figure 2.2: HTTPS

### 2.1.1 Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol (HTTP) is a protocol which allows data exchange.



The process that occurs in the exchange is:

1. A client (a browser as Mozilla, Chrome,...) sends a HTTP request.
2. The server receives the request.

3. The server processes the request.
4. The server returns a HTTP response to the client.
5. The client receives the response.

The HTTP response depends on the HTML request.

Request	Response
HTML page	HTML file
Style of sheet	CSS file
JPG image	JPG file
JavaScript code	JS file
Data	Data

Table 2.1: HTTP request and response

### 2.1.2 Transport Layer Security (TLS)

Transport Layer Security (TLS) is a cryptographic protocol whose function is to provide communication security over a computer network. TLS uses asymmetric cryptography to key exchange, symmetric cryptography to cipher HTTP request and response and digital certificates to authenticate the keys. Currently only there are two versions of TLS allowed which are TLS 1.2 and TLS 1.3.

Cipher	TLS 1.2	TLS 1.3
AES GCM	Yes	Yes
AES CCM	Yes	Yes
Camellia GCM	Yes	No
ARIA GCM	Yes	No
ChaCha20-Poly1305	Yes	Yes

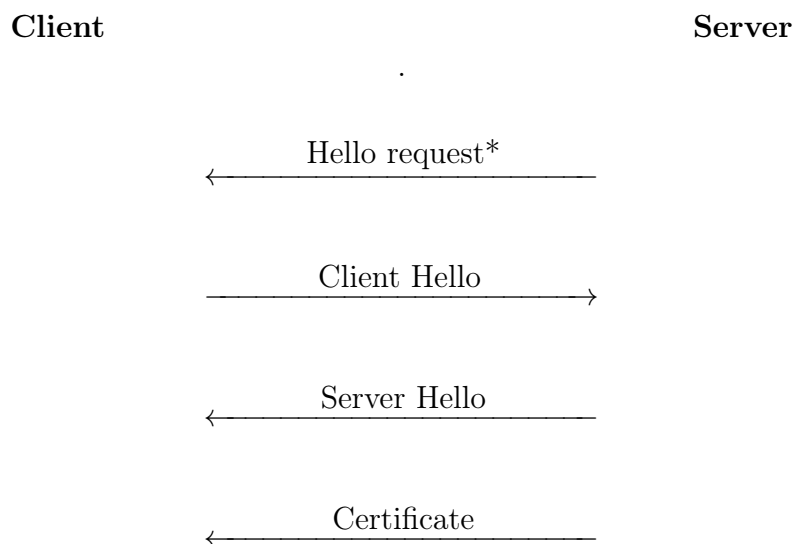
Table 2.2: TLS Ciphers supported

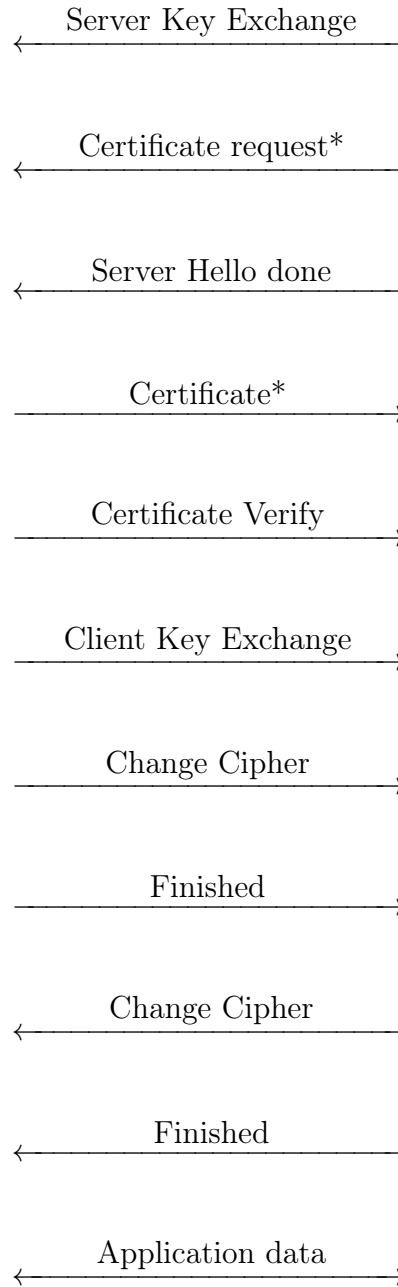
Algorithm	TLS 1.2	TLS 1.3
RSA	Yes	No
DH-RSA	Yes	No
DHE-RSA	Yes	Yes
ECDH-RSA	Yes	No
ECDHE-RSA	Yes	Yes
DH-DSS	Yes	No
DHE-DSS	Yes	No
ECDH-ECDSA	Yes	No
ECDHE-ECDSA	Yes	Yes
ECDH-EdDSA	Yes	No
ECDHE-EdDSA	Yes	Yes
DHE-PSK	Yes	Yes
ECDHE-PSK	Yes	Yes

Table 2.3: TLS Key exchange algorithms supported

## TLS 1.2

The protocol of TLS 1.2 establishes a new session as it is shown. The symbol \* denotes that the message is optional and the meaning of each communication between the client and the server is explained below.





- **Hello request:** This is an optional message sent by the server and is a notification that the client should begin the negotiation process.

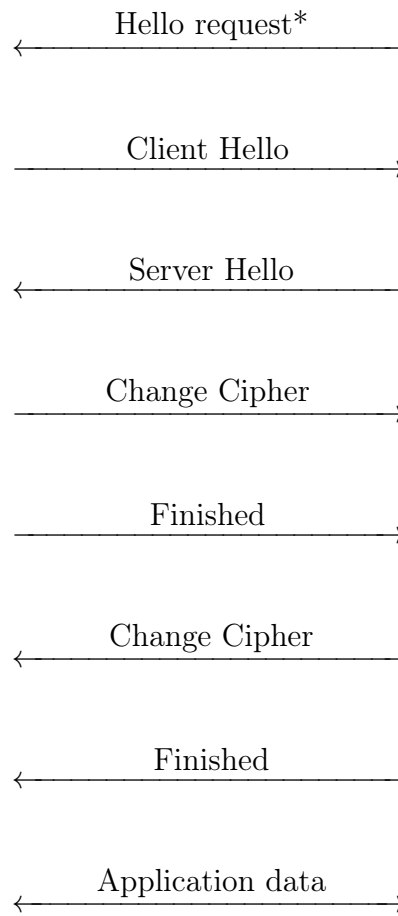
- **Client Hello:** This message is in response to a Hello request or by the client initiative. The Client Hello contains the following fields:
  - **Protocol version:** Indicate the version of the TLS protocol by which the client wishes to communicate during this session.
  - **Random value:** This field contains the time and date in milliseconds since the midnight Jan. 1, 1970, UTC and 28 bytes generated by a secure random number generator.
  - **Session ID:** Empty or provides the ID of a previous session that the client wishes to use.
  - **Cipher Suite:** List of cryptographic options supported by the client. If Session ID is not empty, it must contain the cipher suite from that session.
  - **Compression method:** List of compression methods supported by the client. If Session ID is not empty, it must contain the compression method from that session.
  - **Extension:** Empty or indicates the signature and hash algorithms that the client wishes to use for digital signatures.
- **Server Hello:** This message is in response to a Client Hello. The Server Hello contains the following fields:
  - **Protocol version:** Indicates the version chosen by the server.
  - **Random value:** This field contains the time and date in milliseconds since the midnight Jan. 1, 1970, UTC and 28 bytes generated by a secure random number generator. This number must be independent of the value generated by the client.
  - **Session ID:** If the client Session ID is empty, the server will assign a value to Session ID for this communication. If the client Session ID is not empty, the server will look for a match in the cache.
  - **Cipher Suite:** The cipher suite chosen by the server.
  - **Compression method:** The compression method chosen by the server.

- **Extension:** If the client Extension is empty, the server will send the signature algorithm for key exchange.
- **Server Certificate:** The server sends a certificate message for authentication in the key exchange method. This message is a chain of certificates. The first certificate in the chain is the server's certificate and it is referenced to other certificates. The last certificate is a root certificate self-signed.
- **Server Key Exchange:** This message contains the key exchange method and the parameters needed for the key exchange.
- **Certificate request:** The server can send this message to request a certificate to authenticate the client. In this message, the server includes the type of certificates, the signature algorithms, the hash algorithms and the certificate authorities (CA) that supports.
- **Server Hello done:** A message to indicate that the Server Hello protocol is completed.
- **Client Certificate:** In the case that the server has requested a certificate to the client, the client will send this message that includes his own chain of certificates and the signature algorithms and hash algorithms used for it.
- **Certificate Verify:** Message sent by the client in order to confirm that he has verified the certificates.
- **Client Key Exchange:** This message includes the key exchange parameters needed to complete the key exchange.
- **Change Cipher:** A short message which length with a single byte of value 1 encrypted by the current method.
- **Finished:** Both, the server and the client send this message to confirm that the change cipher has been received, the authentication and the key exchange processes have been a success.

In the case that a session had been established before and the client wants to recover it, the protocol change to avoid unnecessary repetitions. The client sends Client Hello and in the field Session ID, he includes the number of the previous version.

**Client**

**Server**





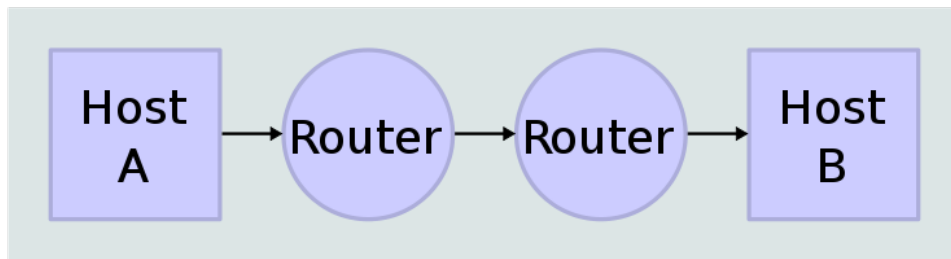


## 2.2 TCP/IP

The Transmission Control Protocol (TCP) and Internet Protocol (IP) are known as Internet Protocol Suite (TCP/IP). The Internet Protocol enables the communication between computers, specifying how the data should be packetized, addressed, transmitted, routed and received. To achieve that, the protocol is divided into five layers. The physical and data link layer constitute the link layer shown in Figure 2.3. From the lowest to the highest, these are:

- **Physical:** This layer deals with bit-level transmission between different devices and supports electrical or mechanical interfaces connecting to the physical medium for synchronized communication.
- **Data Link:** The data link layer is used for the encoding, decoding and logical organization of data bits. Data packets are framed and addressed by this layer.
- **Network Layer:** This layer is responsible for sending packets across multiple networks. The protocols used for it are IPv4 and IPv6. IPv4 works with addresses of 32 bytes whereas IPv6 uses addresses of 128 bytes.
- **Transport Layer:** It is responsible for end-to-end communication over a network. It provides logical communication between application processes running on different hosts within a layered architecture of protocols and other network components. The transport layer is also responsible for the management of error correction, providing quality and reliability to the end user. The protocols defined to do it are TCP and UDP. TCP perform a more exhaustive search for errors than UDP but UDP is faster and more efficient.
- **Application:** The highest layer, it provides the interfaces and protocols needed by the users. Some application layer protocols include the Hypertext Transfer Protocol (HTTP), the File Transfer Protocol (FTP), the Simple Mail Transfer Protocol (SMTP), and the Dynamic Host Configuration Protocol (DHCP).

# Network Topology



# Data Flow

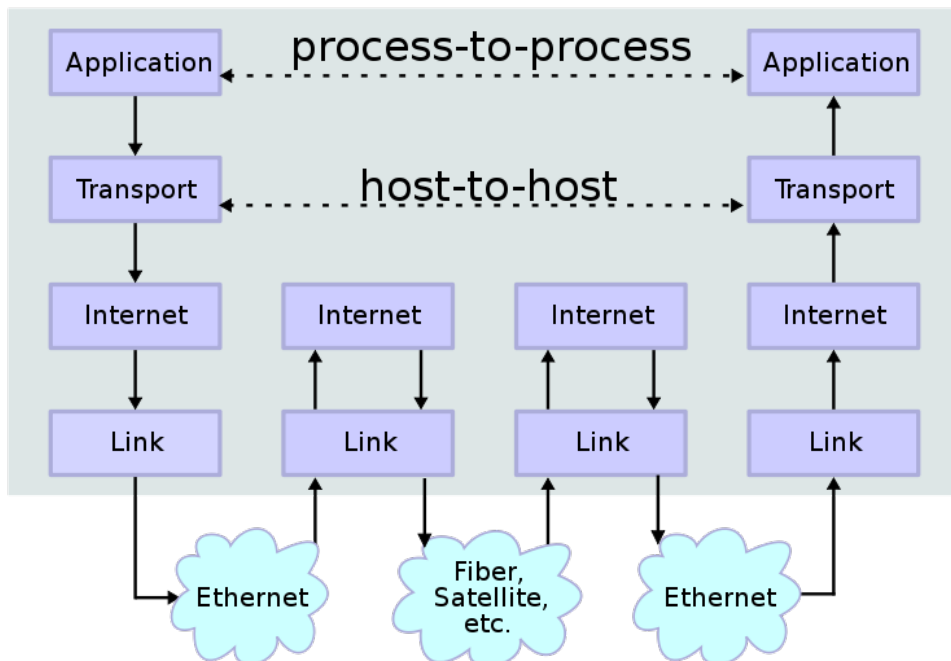


Figure 2.3: Internet Protocol

# Chapter 3

## Password.link

The web page [password.link](https://password.link) allows any message to be safely delivered through an insecure media.

The screenshot shows the Password.link dashboard. At the top is a dark navigation bar with the logo and links for Features, Plans, Docs, API, Contact, News, Log in, and Create account. The main heading is 'Safely deliver any secret through insecure media', followed by a subheading: 'Create a unique one-time link to a password, pre-shared key, license key, access token, you name it. You can rest assured no-one else than you and the link opener can see it.' Below this is a toolbar with 'Settings', 'Generator', and a '+ New' button. The 'Generator' tab is active, showing a large input field for the secret with a placeholder 'Password or other secret to be delivered safely' and a 'Create link' button. Below this are three sections: 'Description' with a text input and a note that it will be shown in the list of stored secrets; 'Message' with a text input and a note that it will be shown along with the secret; and 'Expiration time' with a text input and a note that it is in hours (1-350). At the bottom is a 'View button' section with a checkbox 'Enable view button' and a note that it shows a 'view secret' button first.

Figure 3.1: Dashboard of [password.link](https://password.link)

[Password.link](https://password.link) defines itself as:

"Do you know who reads your passwords and other secrets when you send them over email? We have a solution."

"[Password.link](#) is perfect for sending user passwords, Cisco ASA, IKE and other pre-shared VPN keys, WLAN passwords, license keys for games and software and anything alike. Create a link to the secret that works only one time and get a notification when the link has been accessed. Encryption and decryption of the secret always happen in the browser using an encryption key fully known only by the browser that generates the link."

"Our service encrypts the secret and seals it behind a link that can be opened just once. If the link recipient is unable to open the secret, then someone else has already seen it and proper actions should be taken. Different kind of notifications can also be configured to be sent right away when the secret has been viewed. Encrypting and decrypting the secret always happens in the browser and the actual secret cannot be seen by us."

"We also have an API which can be used to easily integrate the service into any application or service. In addition we also offer licenses to the full source code of the service so companies can run it in their internal networks or other private services, modify it to suit their needs and be sure that they are in full control of all data passed through the service."

"Password.link is being used by all kinds of technology companies all around the world."

Once that the description is completed. It is the time to prove that through a practical example. To do it, it is necessary to create an account. [Password.link](#) offers four different plans:

Plans			
<b>FREE</b> No credit card required  Secrets stored monthly: 8  Includes <b>all</b> features, no strings attached. A great way to test our service!  <a href="#">REGISTER</a>	<b>BASIC</b> 8.99€ / month  Secrets stored monthly: 50  All features with good amount of stored secrets. The best choice for small companies.  <a href="#">LOG IN</a>	<b>ADVANCED</b> 29.99€ / month  Secrets stored monthly: 200  All features, lots of stored secrets and a custom password.link subdomain.  <a href="#">LOG IN</a>	<b>PRO</b> 49.99€ / month  Secrets stored monthly: 500  All features, huge amount of stored secrets and a custom password.link subdomain.  <a href="#">LOG IN</a>

Figure 3.2: Plans offered by [password.link](#)

Now all is ready to perform the example. The secret to sending is "The code has changed. The new code is RC25js79.". There are other fields to complete. These are description, message, expiration time and view button. The description is only seen by the sender and it is a name to recognize the secret. The message is an additional message that will be shown with the secret. The expiration time is the time in hours that the viewer has available to view the secret once it has been sent. The view button is a confirmation button and it is shown before the message. Figure 3.3.

## Dashboard

⚙️ Settings ✏️ Generator + New

🔑 The code has changed. The new code is RC25js79

Create link

**Description**

Description for the secret. Will be shown in the list of stored secrets, can't be seen by the viewer.

**Message**

Message for the viewer. Will be shown along the secret.

**Expiration time**

Expiration time for the link, in hours (1-350).

**View button**

☒ Enable view button

Show a 'view secret' button first instead of showing the secret immediately after opening the link.

Figure 3.3: Dashboard fulfilled

Then the create link button is pressed and it returns a link. This link is sent to another party through any channel (secure or insecure). Figure 3.4.

When the viewer receives the link and introduces it in his navigator, he will see the view secret button because it was activated. Figure 3.5.

If he clicks in the view button the secret will be revealed with the message. Figure 3.6.

If anybody tries to use this link the message will not be shown again. The

same occurs when the first viewer refreshes the page. Another possibility is that the expiration time has over. Figure 3.7.

Finally, the sender can check if his message has been viewed, has expired or is still waiting for an action by the viewer. Figure 3.8.

## Dashboard

⚙ Settings 🔧 Generator + New

✓ <https://password.link/mzCLEg/#a184PCoxUT8teF10L1J8c3xN>

Copy

**Description**

Change in the code

Description for the secret. Will be shown in the list of stored secrets, can't be seen by the viewer.

**Message**

If you don't understand the message, close this tab.

Message for the viewer. Will be shown along the secret.

**Expiration time**

24

Expiration time for the link, in hours (1-350).

**View button**

☒ Enable view button

Show a 'view secret' button first instead of showing the secret immediately after opening the link.

Figure 3.4: Link created

Here's the secret

View secret

Make sure you copy the secret to a safe place right away.  
This page cannot be used to retrieve the secret again after it has been viewed.

When the secret has been viewed it is also deleted from our database. It is then no longer possible for anyone to view it.

Figure 3.5: View secret button

Here's the secret

If you don't understand the message, close this tab.



The code has changed. The new code is RC25js79.

Make sure you copy the secret to a safe place right away.  
This page cannot be used to retrieve the secret again after it has been viewed.

When the secret has been viewed it is also deleted from our database. It is then no longer possible for anyone to view it.

Figure 3.6: Secret and message are revealed

## Secret already viewed or expired

This secret has already been viewed or it has expired. It no longer exists in our database.

If this is your first time trying to use the link, you must notify the link sender immediately that someone else has seen the secret, it has expired or you were given a wrong link.

Figure 3.7: Secret viewed or expired

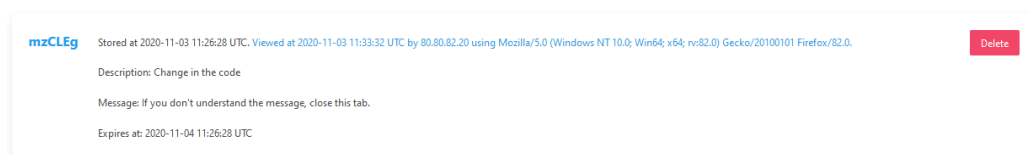


Figure 3.8: Secret's situation



## 3.1 Internet connection

The communication between a user and [password.link](https://password.link) is made through TCP/IP. In particular, the server uses HTTPS. It can be checked easily in the domain of the website.



Figure 3.9: Domain of [password.link](https://password.link)

Clicking on the lock is possible to see which version of TLS is using.

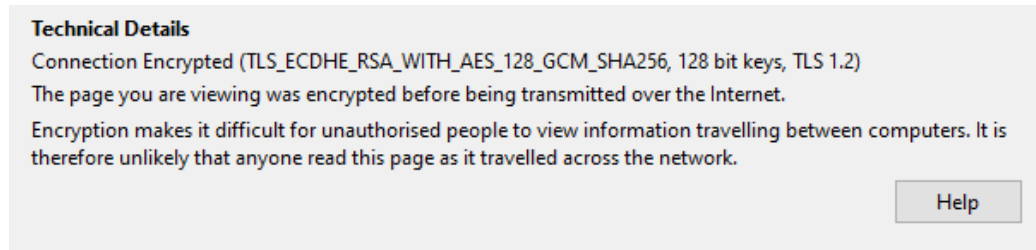


Figure 3.10: Version of TLS and algorithms used

TLS ECDHE RSA WITH AES 128 GCM SHA256, 128 bits keys, TLS 1.2 means:

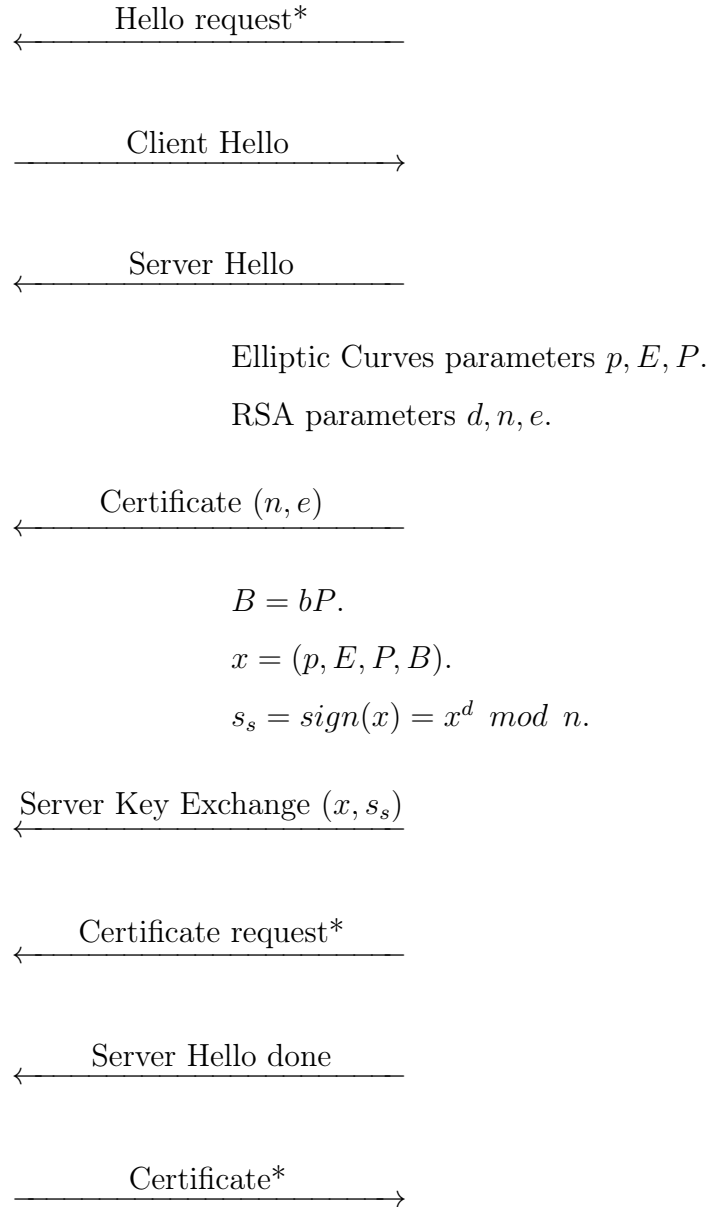
- **TLS version:** 1.2.
- **Key exchange:** Elliptic Curves Diffie–Hellman Ephemeral (ECDHE).
- **Authentication:** RSA.
- **Encryption:** Advanced Encryption System 128 bits (AES-128) with Galois Counter Mode (GCM).
- **Hash function:** SHA 256.

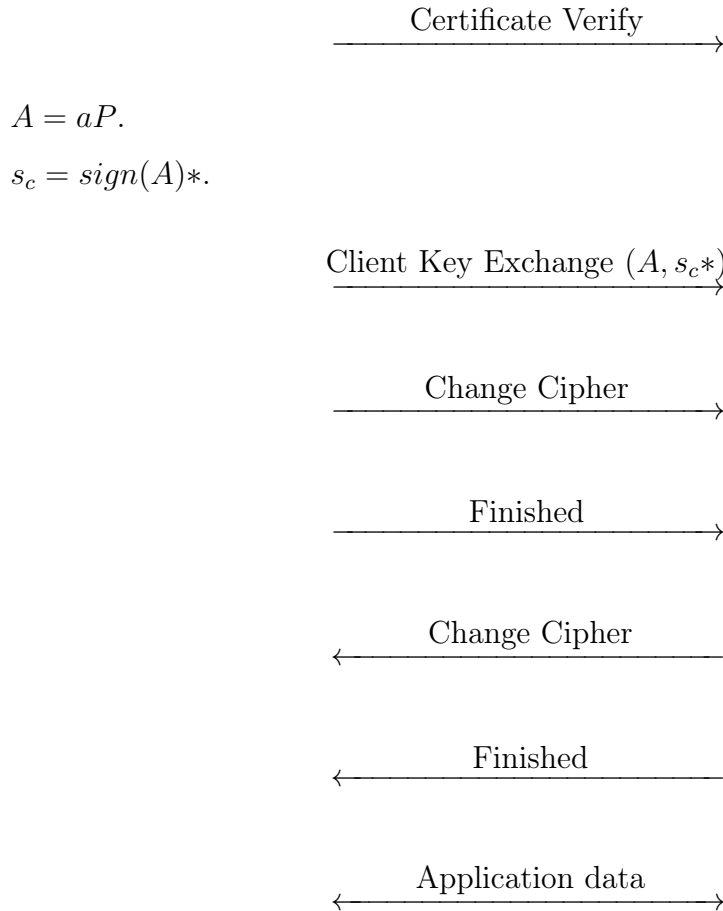
Elliptic Curves Diffie–Hellman Ephemeral (ECDHE) is Diffie–Hellman Key Exchange with Elliptic Curves viewed in 1.8.3. The only difference is that the word Ephemeral is added. It means that the parameters to generate the key exchange are refreshed each time that this process is performed. As it is explained in 1.13, authentication is necessary to avoid the man in the middle

attack. It is done with certificates. These certificates use RSA to sign. A scheme is shown below.

**Client**

**Server**





[Password.link](#) uses three certificates to authenticate. The first one is the end-entity certificate which belongs to [password.link](#) and is referred to the intermediate certificate. Figure 3.11.

The second is the intermediate certificate which belongs to [Let's Encrypt Authority X3](#) and is referred to the root certificate. Figure 3.12.

The root certificate belongs to Digital Signature Trust Co. and is self-signed. This CA is widely recognized. Figure 3.13.

The three certificates use RSA to verify to encrypt and the length of the key is 2048 bits. The user receives:

$$(n, e)$$

Certificate		
<a href="#">password.link</a>	Let's Encrypt Authority X3	DST Root CA X3
<b>Subject Name</b>		
<b>Common Name</b>	password.link	
<b>Issuer Name</b>		
<b>Country</b>	US	
<b>Organisation</b>	Let's Encrypt	
<b>Common Name</b>	<a href="#">Let's Encrypt Authority X3</a>	
<b>Validity</b>		
<b>Not Before</b>	18/09/2020, 02:42:05 (Central European Standard Time)	
<b>Not After</b>	17/12/2020, 01:42:05 (Central European Standard Time)	
<b>Subject Alt Names</b>		
<b>DNS Name</b>	password.link	
<b>Public Key Info</b>		
<b>Algorithm</b>	RSA	
<b>Key Size</b>	2048	
<b>Exponent</b>	65537	
<b>Modulus</b>	BF:08:2C:DB:C5:2B:9C:9C:E8:64:93:1D:23:2F:DC:B8:5B:9B:96:E7:3C:D6:D4:83:DE:45:8C:7A:21:9E:0F:33:23:63:EA:FA:D6:C C4D:6A:47:19:FF:CB:65:C0:C9:36:35:5C:5E:CA:F7:A9:8D:D0:8C:0E:BC:6B:62:98:56:FC:24:7A:CA:59:4C:04:0F:C5:A3:CA:56: AB:75:0C:A6:60:2C:E7:CC:9F:38:A3:24:ED:9F:E5:03:1D:F1:DE:8A:82:09:38:16:F6:9A:C8:C6:83:19:58:0D:51:24:78:BD:C5:2F: 28:8F:06:F9:24:0F:78:C7:49:FA:2C:BD:17:28:AB:34:B0:61:AE:4B:5E:23:22:83:88:C2:EE:D7:24:A5:E7:15:7F:4B:C5:8D:D8:0F:8 9:26:CF:DF:7E:F7:2B:50:2C:55:DE:65:D2:52:20:81:8C:12:57:70:C9:9E:B6:4D:CE:87:7D:14:05:04:11:8B:95:3D:C0:77:C6:6E:4 2:B2:EA:9F:07:AE:E0:06:9C:C2:A2:AD:51:C7:89:4A:00:42:A5:87:D2:60:80:BE:2A:9E:6B:5E:0B:28:BA:67:7C:A5:8B:75:D1:40: 69:96:F1:35:33:01:A4:D1:A2:F6:0E:3E:5E:B0:A2:0E:AF:AD:5E:E1:BF:6A:16:DA:78:45:AD:96:01	
<b>Miscellaneous</b>		
<b>Serial Number</b>	03:FE:2D:9E:83:8E:44:A7:10:34:AD:BD:D3:C3:E1:E2:AE:0F	
<b>Signature Algorithm</b>	SHA-256 with RSA Encryption	
<b>Version</b>	3	
<b>Download</b>	<a href="#">PEM (cert)</a> <a href="#">PEM (chain)</a>	
<b>Fingerprints</b>		
<b>SHA-256</b>	61:A4:C7:8D:A8:52:CD:1A:BE:FB:55:DC:48:B5:C3:A8:D8:EE:23:7A:CD:BA:25:C8:7A:2F:A2:B5:C0:9A:9F:F6	
<b>SHA-1</b>	32:ED:24:31:74:53:6B:F0:3D:D8:A4:D7:2E:52:1C:6E:85:D2:DB:31	

Figure 3.11: End-entity certificate

$$x = (p, E, P, B)$$

$$s_s = \text{sign}(x) = x^d \bmod n$$

he must compute

$$x' = s_s^e \bmod n$$

and check that

$$x' = x$$

Certificate

password.link	Let's Encrypt Authority X3	DST Root CA X3
<b>Subject Name</b> _____		
<b>Country</b>	US	
<b>Organisation</b>	Let's Encrypt	
<b>Common Name</b>	Let's Encrypt Authority X3	
<b>Issuer Name</b> _____		
<b>Organisation</b>	Digital Signature Trust Co.	
<b>Common Name</b>	DST Root CA X3	
<b>Validity</b> _____		
<b>Not Before</b>	17/03/2016, 17:40:46 (Central European Standard Time)	
<b>Not After</b>	17/03/2021, 17:40:46 (Central European Standard Time)	
<b>Public Key Info</b> _____		
<b>Algorithm</b>	RSA	
<b>Key Size</b>	2048	
<b>Exponent</b>	65537	
<b>Modulus</b>	9C:D3:0C:F0:5A:E5:2E:47:B7:72:5D:37:83:B3:68:63:30:EA:D7:35:26:19:25:E1:BD:BE:35:F1:70:92:2F:B7:B8:4B:41:05:AB:A9:9E:35:08:58:EC:B1:2A:C4:68:87:0B:A3:E3:75:E4:E6:F3:A7:62:71:BA:79:81:60:1F:D7:91:9A:9F:F3:D0:78:67:71:C8:69:0E:95:91:CF:FE:E6:99:E9:60:3C:48:CC:7E:CA:4D:77:12:24:9D:47:1B:5A:EB:B9:EC:1E:37:00:1C:9C:AC:7B:A7:05:EA:CE:4A:EB:BD:41:E5:36:98:B9:CB:FD:6D:3C:96:68:DF:23:2A:42:90:0C:86:74:67:C8:7F:A5:9A:B8:52:61:14:13:3F:65:E9:82:87:CB:DB:FA:0E:56:F6:86:89:F3:85:3F:97:86:AF:B0:DC:1A:EF:6B:0D:95:16:7D:C4:2B:A0:65:B2:99:04:36:75:80:6B:AC:4A:F3:1B:90:49:78:2F:A2:96:4F:2A:20:25:29:04:C6:74:C0:D0:31:CD:8F:31:38:95:16:BA:A8:33:B8:43:F1:B1:1F:C3:30:7F:A2:79:31:13:3D:2D:36:F8:E3:FC:F2:33:6A:B9:39:31:C5:AF:C4:8D:0D:1D:64:16:33:AA:FA:84:29:B6:D4:0B:C0:D8:7D:C3:93	
<b>Miscellaneous</b> _____		
<b>Serial Number</b>	0A:01:41:42:00:00:01:53:85:73:6A:0B:85:EC:A7:08	
<b>Signature Algorithm</b>	SHA-256 with RSA Encryption	
<b>Version</b>	3	
<b>Download</b>	<a href="#">PEM (cert)</a> <a href="#">PEM (chain)</a>	
<b>Fingerprints</b> _____		
<b>SHA-256</b>	25:84:7D:66:8E:B4:F0:4F:DD:40:B1:2B:6B:07:40:C5:67:DA:7D:02:43:08:EB:6C:2C:96:FE:41:D9:DE:21:8D	
<b>SHA-1</b>	E6:A3:B4:5B:06:2D:50:9B:33:82:28:2D:19:6E:FE:97:D5:95:6C:CB	

Figure 3.12: Intermediate certificate

if the two variable has the same value he will check that the end-entity certificate is signed by the intermediate certificate and that the intermediate certificate is signed by the root certificate. This operation is called Certificate Verify and is inside that TLS 1.2.

When it has been checked, the server and the user shares the same key, it will be used to transfer application data encrypting and decrypting it with AES-128 and the GCM. In this case, the application data is a hypertext transfer.

Certificate

password.link	Let's Encrypt Authority X3	DST Root CA X3
---------------	----------------------------	----------------

Subject Name

Organisation

Digital Signature Trust Co.

Common Name

DST Root CA X3

Issuer Name

Organisation

Digital Signature Trust Co.

Common Name

DST Root CA X3

Validity

Not Before

30/09/2000, 23:12:19 (Central European Standard Time)

Not After

30/09/2021, 16:01:15 (Central European Standard Time)

Public Key Info

Algorithm

RSA

Key Size

2048

Exponent

65537

Modulus

DF:AF:E9:97:50:08:83:57:B4:CC:62:65:F6:90:82:EC:C7:D3:2C:6B:30:CA:5B:EC:D9:C3:7D:C7:40:C1:18:14:8B:E0:E8:33:76:49:2A:E3:3F:21:49:93:AC:4E:0E:AF:3E:48:C8:65:EE:FC:D3:21:0F:65:D2:2A:D9:32:8F:8C:E5:F7:77:80:12:7B:B5:95:C0:89:A3:A9:BA:ED:73:2E:7A:0C:06:32:83:A2:7E:8A:14:30:CD:11:A0:E1:2A:38:B9:79:0A:31:FD:50:BD:80:65:DF:B7:51:63:83:C8:E2:88:61:EA:4B:61:81:EC:52:6B:89:A2:E2:4B:1A:28:9F:48:A3:9E:0C:DA:09:8E:3E:17:2E:1E:DD:20:DF:5B:C6:2A:8A:AB:2E:BD:70:AD:C5:0B:1A:25:90:74:72:C5:7B:6A:AB:34:D6:30:89:FF:E5:68:13:7B:54:0B:C8:D6:AE:EC:5A:9C:92:1E:3D:64:B3:8C:C6:DF:BF:C9:41:70:EC:16:72:D5:26:EC:38:55:39:43:D0:FC:FD:18:5C:40:F1:97:EB:D5:9A:9B:8D:1D:BA:DA:25:B9:C6:D8:DF:C1:15:02:3A:AB:DA:6E:F1:3E:2E:F5:5C:08:9C:3C:D6:83:69:E4:10:9B:19:2A:B6:29:57:E3:E5:3D:9B:9F:F0:02:5D

Miscellaneous

Serial Number

44:AF:B0:80:D6:A3:27:BA:89:30:39:86:2E:F8:40:6B

Signature Algorithm

SHA-1 with RSA Encryption

Version

3

Download

[PEM \(cert\)](#)
[PEM \(chain\)](#)

Fingerprints

SHA-256

06:87:26:03:31:A7:24:03:D9:09:F1:05:E6:9B:CF:0D:32:E1:BD:24:93:FF:C6:D9:20:6D:11:BC:D6:77:07:39

SHA-1

DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13

Figure 3.13: Root certificate

## 3.2 Password.link operation

To sum up the process, a user introduces a secret, a message and the expiration time, the web page returns a secret link and encrypt the secret, the viewer will open the link and will decrypt it to view the secret. The question is how occurs it and if there is any cryptographic function participating in the process.

### 3.2.1 Generate the link

The secret link that the user request to send the secret to the viewer has always the same structure.

$$\begin{aligned} \textit{Secret Link} = & \textit{Link Base} + "/" + \textit{Secret ID} + "/" + "#" + \\ & + \textit{Password Part Public Base64} \end{aligned}$$

The Link Base is *https : //password.link*.

The Secret ID is the value generated with a simple hash function, where the secret, the message, the expiration time and time expressed in milliseconds since the midnight Jan. 1, 1970, UTC are hashed. This hash function has 6 characters as output.

The Password Part Public Base 64 is slightly more complex to calculate. The web page generates two string called Password Part Public and Password Part Private. Each string has 18 characters. The strings are generated by a random function explained below. The strings only can have the characters included in Table 3.1. For instance, the characters ; : will not appear in any string.

The strings are generated character by character. To generate a character the function *Math.seedrandom* is used. This function returns a random number in  $[0, 1)$ . This random number is multiplied by 87 and the largest integer less or equal than the given number is utilized to choose the character.

Number	Character	Number	Character	Number	Character
0	<i>A</i>	29	<i>d</i>	58	6
1	<i>B</i>	30	<i>e</i>	59	7
2	<i>C</i>	31	<i>f</i>	60	8
3	<i>D</i>	32	<i>g</i>	61	9
4	<i>E</i>	33	<i>h</i>	62	\$
5	<i>F</i>	34	<i>i</i>	63	
6	<i>G</i>	35	<i>j</i>	64	/
7	<i>H</i>	36	<i>k</i>	65	\
8	<i>I</i>	37	<i>l</i>	66	!
9	<i>J</i>	38	<i>m</i>	67	_
10	<i>K</i>	39	<i>n</i>	68	+
11	<i>L</i>	40	<i>o</i>	69	,
12	<i>M</i>	41	<i>p</i>	70	.
13	<i>N</i>	42	<i>q</i>	71	—
14	<i>O</i>	43	<i>r</i>	72	?
15	<i>P</i>	44	<i>s</i>	73	(
16	<i>Q</i>	45	<i>t</i>	74	)
17	<i>R</i>	46	<i>u</i>	75	[
18	<i>S</i>	47	<i>v</i>	76	]
19	<i>T</i>	48	<i>w</i>	77	{
20	<i>U</i>	49	<i>x</i>	78	}
21	<i>V</i>	50	<i>y</i>	79	<
22	<i>W</i>	51	<i>z</i>	80	>
23	<i>X</i>	52	0	81	&
24	<i>Y</i>	53	1	82	#
25	<i>Z</i>	54	2	83	^
26	<i>a</i>	55	3	84	*
27	<i>b</i>	56	4	85	=
28	<i>c</i>	57	5	86	@

Table 3.1: Equivalence between numbers and characters

For example, 0.0988257879579245 is the value obtained from *Math.seedrandom*:

$$87 * 0.0988257879579245 = 8.597843552$$



The largest integer less than or equal than 8.5978... is 8.

$$8 \Rightarrow I$$

This process is repeated 18 times to generate a string with 18 characters. Once the Password Part Public and Password Part Private has been generated, these must be converted in Base64.

### **Math.seedrandom**

The main question here is how the numbers are generated. *Math.seedrandom* uses RC4 which is explained in 1.2. Notice that RC4 algorithm needs a key to calculate the S-Box. Once the S-box is calculated, the RC4 RNG is used to calculate a random number  $r$  with *count* = 6. Notice that the output of *Math.seedrandom* is in the interval  $[0, 1)$  while  $r$  is always bigger than 1. To convert the number in the interval:

$n = RC4RNG \text{ (count = 6)}$

$d = 2^{48}$

$x = 0$

$while(n < 2^{52})$

$n = (n + x) \cdot 256$

$d = d \cdot 256$

$x = RC4RNG \text{ (count = 1)}$

$while(n > 2^{52})$

$n = n/2 \cdot 256$

$d = d/2 \cdot 256$

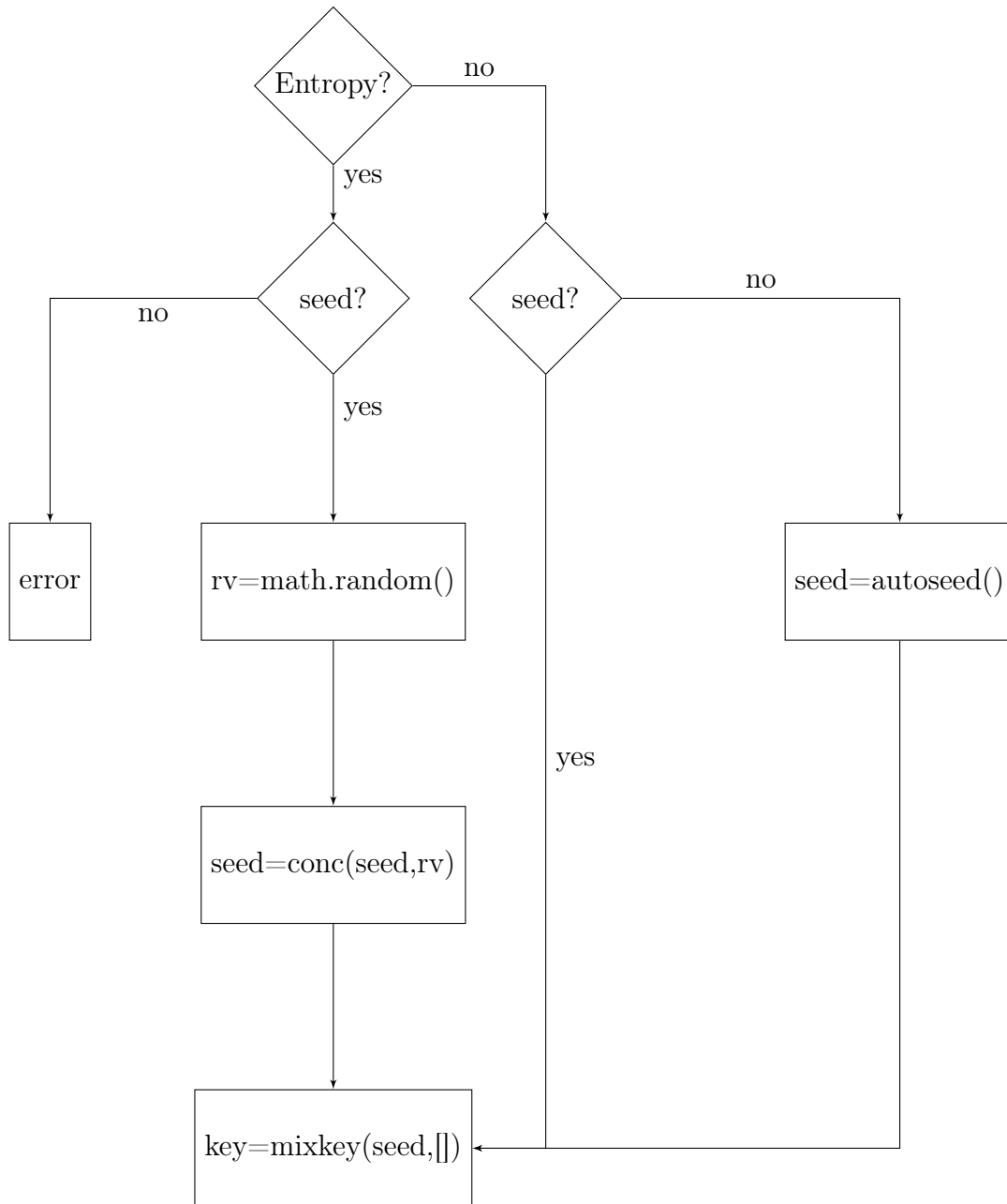
$x >>>= 1$

$return(n + x)/d \quad [0, 1)$

The operation  $>>>=$  called as right shift assignment moves the specified amount of bits to the right and assigns the result to the variable.

$$16 >>>= 2 \rightarrow 10000_2 >>>= 2 \rightarrow 00100_2 = 4$$

$$5 >>>= 1 \rightarrow 101_2 >>>= 1 \rightarrow 010_2 = 2$$



The key for generating the S-Box is obtained through a seed and entropy. The user can choose one of the following options: Entropy and seed, only seed or neither. These fields are introduced when the user calls the function.

*Math.seedrandom('hello', {entropy : true})*      Entropy and a seed

*Math.seedrandom('hello')*      A seed

*Math.seedrandom()*

*Math.random()* is a function which generates random values without any security. The values are in  $[0,1)$ . The function *conc()* concatenates two variables given as inputs and converts them into a string.

*conc('hello', 0.731861861) = 'hello0.731861861'*

The function *autoseed()* does not need any input. His output is a string that depends on the time expressed in milliseconds since the midnight Jan. 1, 1970, UTC and local variables such as the plugins, the screen...

The last function is *mixkey*, his goal is to obtain a key from a seed.

*mixkey(seed, initkey)*

*stringseed = seed + "*

*j = 0*

*mask = 255*       $255_{10} = 11111111_2$

*smear = 0*

*while(j < length(stringseed))*

*smear ^= initkey[mask & j] · 19*

*key[mask & j] = mask & ((smear ^ (initkey[mask & j] · 19) + ASCII(stringseed(j))))*

*smear = smear ^ initkey[mask & j] · 19*

*j = j + 1*

*return key*

The symbol & is the bitwise AND operation and it avoids keys longer than 256 characters. Notice that to obtain the S-Box a key longer than 256 characters will produce the same effect that the same key truncated from the characters 1 to 256. The symbol ^= represents the bitwise assignment XOR operation.

For instance,  $seed = 'hyt'$  and  $initkey = [37452]$ .

$mixkey('hyt', [3, 7, 4, 5, 2])$

$stringseed = 'hyt'$

$j = 0$

$mask = 255$

$smear = 0$

$while(0 < 3)$

$key[11111111_2 \& 0] = 11111111_2 \& ((0 \oplus (initkey[11111111_2 \& 0] \cdot 19) +$   
 $+ ASCII(stringseed(h))))$

$key[0] = 0 \oplus (3 \cdot 19) + 104 = 57 + 104 = 161$

$smear = 0 \oplus initkey[11111111_2 \& 0] \cdot 19 = initkey[0] \cdot 19 = 3 * 19 = 57$

$j = 1$

$while(1 < 3)$

$key[1] = 11111111_2 \& ((57 \oplus (initkey[1] \cdot 19) + ASCII(stringseed(y))))$

$key[1] = 11111111_2 \& ((57 \oplus (7 \cdot 19) + 121 = 11111111_2 \& ((57 \oplus 133) +$   
 $+ 161) = 11111111_2 \& ((111001_2 \oplus 10000101_2) + 121) = 11111111_2 \& (10111100_2 +$   
 $+ 161) = 11111111_2 \& (188 + 121) = 11111111_2 \& 309 = 11111111_2 \& 00110101_2 =$   
 $= 01011101_2 = 53$

$smear = 57 \oplus (initkey[1] \cdot 19) = 57 \oplus (7 * 19) = 57 \oplus 133 = 188$

$j = 2$

$while(2 < 3)$

$key[2] = 11111111_2 \& ((188 \oplus (initkey[2] \cdot 19) + ASCII(stringseed(t))))$

$key[2] = 11111111_2 \& ((188 \oplus (4 \cdot 19) + 116 = 11111111_2 \& ((188 \oplus 133) +$   
 $+ 116) = 11111111_2 \& ((10111100_2 \oplus 10000101_2) + 116) = 11111111_2 \& (11110000_2 +$   
 $+ 116) = 11111111_2 \& (240 + 116) = 11111111_2 \& 356 = 11111111_2 \& 101100100_2 =$

$$= 01100100_2 = 100$$

$$smear = 240$$

$$j = 3$$

$$return\ key = [161, 53, 100]$$

Note that the *initkey* is always empty when *Math.seedrandom* is called. When it occurs, *mixkey* performs a simply conversion from characters to numbers using ASCII. The same example but with *initkey* = [].

$$mixkey('hyt', [])$$

$$stringseed = 'hyt'$$

$$j = 0$$

$$mask = 255$$

$$smear = 0$$

$$while(0 < 3)$$

$$key[11111111_2 \& 0] = 11111111_2 \& ((0 \oplus (initkey[11111111_2 \& 0] \cdot 19) + \\ + ASCII(stringseed(h))))$$

$$key[0] = 0 \oplus (0 \cdot 19) + 104 = 104$$

$$smear = 0 \oplus initkey[11111111_2 \& 0] \cdot 19 = initkey[0] \cdot 19 = 0 * 19 = 0$$

$$j = 1$$

$$while(1 < 3)$$

$$key[1] = 11111111_2 \& ((0 \oplus (initkey[1] \cdot 19) + ASCII(stringseed(y))))$$

$$key[1] = 11111111_2 \& ((0 \oplus (0 \cdot 19) + 121) = 11111111_2 \& 121 = 121$$

$$smear = 0 \oplus (initkey[1] \cdot 19) = 0 \oplus (0 * 19) = 0$$

$$j = 2$$

$$while(2 < 3)$$

$$key[2] = 11111111_2 \& ((0 \oplus (initkey[2] \cdot 19) + ASCII(stringseed(t))))$$

$$key[2] = 11111111_2 \& ((0 \oplus (0 \cdot 19) + 116) = 11111111_2 \& 116 = 116$$

```

    smear = 0

    j = 3

    return key = [104, 121, 116]

```

To generate the random numbers that influence the characters and so that the Password Part Public and Password Part Private the function *Math.seedrandom* is called without any seed or entropy, it means *Math.seeedrandom()*. Random numbers are generated because entropy is introduced in *autoseed()* to generate the keys and then these are used in RC4 algorithm to provide the S-Boxes which enable to calculate random numbers through RC4 RNG.

### Conversion to Base64

The ASCII code has a Base256 because each character is defined by 8 bits. Each character in a Base64 must be defined with 6 bits.

To convert the text in ASCII to Base64, the text is written in binary and then groups of 6 bits are formed, the groups are substituted by letters using Table 3.2. An example with the word *Sun* is shown:

$$\begin{aligned}
 S &= 83_{10} = 01010011_2 \\
 u &= 117_{10} = 01110101_2 \\
 n &= 110_{10} = 01101110_2 \\
 Sun &= 010100110111010101101110_2 = \underbrace{010100}_{20_{10}} \underbrace{110111}_{55_{10}} \underbrace{010101}_{21_{10}} \underbrace{101110}_{46_{10}} \\
 Sun_{Base256} &= U3Vu_{Base64}
 \end{aligned}$$

When the number of characters in Base256 is not a multiple of 3, padding with the symbol = is required but in the case of [password.link](#) the strings have 18 characters and as 18 is a multiple of 3 any padding is required.

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

Table 3.2: Equivalence between numbers and characters Base64

### 3.2.2 Encrypt the secret and send the information

Simultaneously to the link generation, the secret is encrypted. The Password Part Public and Password Part Private are used as the initial key. The algorithms to do it are PBKDF, AES 256, GCM and SHA 256. The function to encrypt:

$$Ciphertext_{Base64} = Base64(Encrypt(PasswordPartPublic, PasswordPartPrivate, Secret, Mode, Cipher, Ks, Iter))$$

where:

$$Mode = 'gcm'$$

$$Cipher = 'aes'$$

$$Ks = 256$$

$$Iter = 10000$$

The process is:

1. To concatenate the *PasswordPartPublic* and *PasswordPartPrivate*  $PW = conc(PasswordPartPublic, PasswordPartPrivate)$  and to convert it to Base64.
2. To generate four random numbers called as *IV*.
3. To generate two random numbers called as *Salt*.
4. To perform  $Key = PBKDF(SHA256, PW, Salt, 10000)$ .
5. To encrypt  $Ct = AES256_{Encrypt} - GCM(Key, Secret, IV)$ .
6. To convert to Base64  $Ciphertext_{Base64} = Base64(IV, Iter, Ks, Mode, Cipher, Salt, Ct)$ .
7. To send the  $Ciphertext_{Base64}$ , the *PasswordPartPrivateBase64*, the *SecretID* and the message details to [Password.link](#).

The function to generate the random numbers IV and Salt depends on the time since the midnight Jan. 1, 1970, UTC and other parameters with entropy but it has not any security in terms of cryptography.



### 3.2.3 View, decrypt and delete the secret

When a user access to the link, [password.link](#) shows the secret and delete all the information about this secret.

1. To get the *SecretID* and the *PasswordPartPublicBase64* from the link.
2. To search for a *SecretID* match in [Password.link](#).
3. To obtain the *PasswordPartPrivateBase64*, the *Ciphertext<sub>Base64</sub>* and the message details from the match in Password Link.
4. To delete the information related with this secret which was stored in Password Link.
5. To invert the Base64 transformation to obtain *PasswordPartPublic*, *PasswordPartPrivate* and *Ciphertext = IV, Iter, Ks, Mode, Cipher, Salt, Ct*.
6. Concatenate the *PasswordPartPublic* and *PasswordPartPrivate*  
 $PW = conc>PasswordPartPublic, PasswordPartPrivate$ .
7. Perform  $Key = PBKDF(SHA256, PW, Salt, 10000)$ .
8. Finally the  $Secret = AES256_{Decrypt} - GCM(Key, Ct, IV)$ .
9. Show the *Secret* with the message details.

Note that for decryption both, Password Part Public and Password Part Private are needed. The user sent the Password Part Private and it guarantees that [Password.link](#) can not decrypt the secret until the user access to the link.

<p style="text-align: center;">Chipertext Base64</p> <p>eyJpdil6litxQlJ4TkdrUEIvOXVlNGNmVVBVZ2c9PSIsInYiOjEsImI0Z XIiOjEwMDAwLCJrcyI6MjU2LCJ0cyI6NjQsIm1vZGUiOiJnY20iLCJh ZGF0YSI6IiIsImNpcGhlciI6ImFlcyIsInNhbHQiOiI5TGhNeVpXbXNrT T0iLCJjdCI6IlpTbStxSmxOUHV1eWdHU3MwRFkvY2tqSUzyZlBCaW IvWjBvPSJ9</p>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Chipertext

```
{"iv":"+qBRxNGQPB/9ue4cfUPUgg==","v":1,"iter":10000,"ks":256,
"ts":64,"mode":"gcm","adata":"","cipher":"aes","salt":"9LhMyZWmsk
M=","ct":"ZSm+qJlNPuuygGSs0DY/ckjIK2fPBib/Z0o="}
```



## 3.3 Additional Features

There are two additional features which are an option to personalize the service in order to integrate with other app and a generator of passwords.

### 3.3.1 Personalize the service

[Password.link](#) provides all the code use and his API. It enables someone to incorporate this service in his own app and customize the to match his brand. Also, it enables configure notifications when the secret has been viewed.

### 3.3.2 Generator of passwords

It generates strong xkcd-style passwords using EFF's Long Wordlist for words. The user chooses the characters to separate the words, the padding characters and the amount of random words. The process is simple.

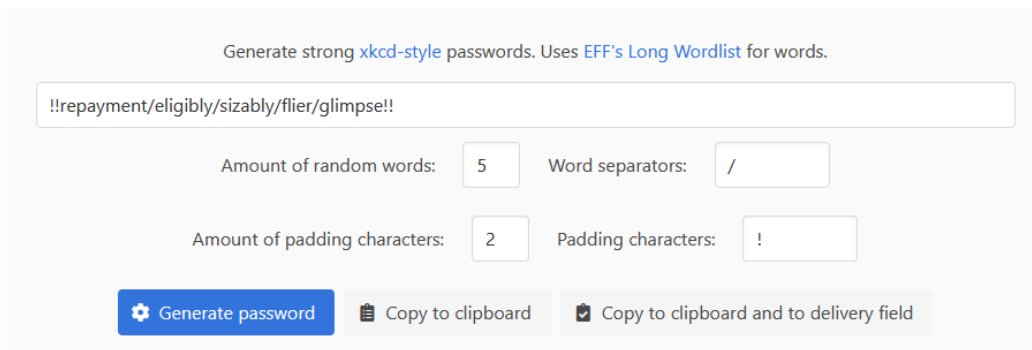
The image shows a web interface for generating passwords. At the top, it says "Generate strong xkcd-style passwords. Uses EFF's Long Wordlist for words." Below this is a text box containing the generated password: "!!repayment/eligibly/sizably/flier/glimpse!!". Underneath the text box are four input fields: "Amount of random words:" with a value of 5, "Word separators:" with a value of "/", "Amount of padding characters:" with a value of 2, and "Padding characters:" with a value of "!". At the bottom, there are three buttons: "Generate password" (blue), "Copy to clipboard" (grey), and "Copy to clipboard and to delivery field" (grey).

Figure 3.14: Password generation

1. Generate five random numbers. The numbers allowed only can be 1, 2, 3, 4, 5 or 6.
2. Use the [EFF's Long Wordlist](#) with the previous numbers. An example of it is the sequence 5,1,4,2,5 corresponds to the word 'repayment'.

3. Repeat steps 1 and 2 to come up with the correct number of random words
4. Form the password using the random words, the characters to separate the words and the padding characters.

## 3.4 Conclusion

[Password.link](#) offers a service that allows you to send a message through an insecure channel.

First of all, the connection between a user and [password.link](#) through internet is secure due to both parties are authenticated and the internet connection satisfies all the requirements.

From a cryptographic point of view, the [password.link](#) operation is secure but it can be improved replacing RC4 for a safer RNG. The reason because [password.link](#) is secure even when it uses RC4, which is no recommended, is because the whole process occurs in your own computer and only then Password Part Private is sent to [password.link](#) and both passwords are required to decrypt. All the other functions that it uses to encrypt the message can be considered secure until now since there are no known feasible attacks at present. Also, as consequence that all the encryption process is carried out by the user and the user only sends a half of the key [password.link](#) will never be able to decrypt the ciphertext unless someone accesses the link.

If someone intercepts the link and opens it, the secret will be revealed. It does not make [password.link](#) appropriate to send detailed information about credit cards, router keys, licence keys for games, etc. "

[Password.link](#) is an incomplete service since it does not allow sending the link to another user from their own page. That is why the option of integrating this system into other applications is suitable.

Finally, I have to say that having all the code at a single click has been quite useful and that despite being a simply registered user and not having bought any of their subscription plans, all the questions asked through the technical service have been answered the same day and with impeccable precision.



## 3.5 Code

All the code used for understanding how [password.link](#) and to perform some simulations is included above. To use it and simulate secret encryption the text must be copied in a text file and edit his format. Once it is done, simulation.html is opened with a navigator, F12 and Console are pressed.

Press F12 and then press Console

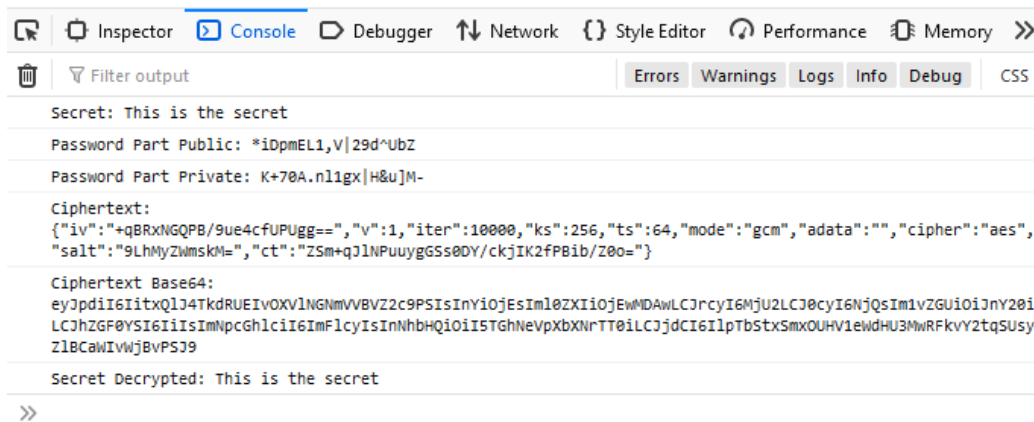


Figure 3.15: Example using the code

### 3.5.1 simulation.html

```
<!DOCTYPE html>

<head>
<meta charset="utf-8">
<title> Simulation </title>

<script type="text/javascript" src="simulation.js"></script>

</head>

<body>
```



Press F12 and then press Console

```
</body>
```

```
</html>
```

### 3.5.2 simulation.js

```
"use strict";

var sjcl={
  cipher:{},
  hash:{},
  keyexchange:{},
  mode:{},
  misc:{},
  codec:{},
  exception:{
    corrupt:function(a){
      this.toString=function(){
        return"CORRUPT: "+this.message
      };
      this.message=a
    },
    invalid:function(a){
      this.toString=function(){
        return"INVALID: "+this.message
      };
      this.message=a
    },
    bug:function(a){
      this.toString=function(){
        return"BUG: "+this.message
      };
      this.message=a
    },
    notReady:function(a){
      this.toString=function(){
        return"NOT READY: "+this.message
      };
      this.message=a
    }
  }
}
```

```

    }
};

sjcl.cipher.aes=function(a){
    this.s[0][0][0]||this.0();
    var b,c,d,e,f=this.s[0][4],g=this.s[1];
    b=a.length;
    var h=1;
    if(4!==b&&6!==b&&8!==b)throw new sjcl.exception.invalid("
        invalid aes key size");
    this.b=[d=a.slice(0),e=[]];
    for(a=b;a<4*b+28;a++){
        c=d[a-1];
        if(0===a%b||8===b&&4===a%b)c=f[c>>>24]<<24^f[c
        >>16&255]<<16^f[c>>8&255]<<8^f[c&255],0===a%b&&(c=c<<8^c
        >>>24^h<<24,h=h<<1^283*(h>>7));
        d[a]=d[a-b]^c
    }
    for(b=0;a;b++,a--)c=d[b&3?a:a-4],e[b]=4>=a||4>b?c:g[0][f[c
        >>>24]]^g[1][f[c>>16&255]]^g[2][f[c>>8&255]]^g[3][f[c
        &255]]
    };

sjcl.cipher.aes.prototype={
    encrypt:function(a){
        return t(this,a,0)
    },
    decrypt:function(a){
        return t(this,a,1)
    },
    s:[[],[],[],[],[],[],[],[],[],[]],
    0:function(){
        var a=this.s[0],b=this.s[1],c=a[4],d=b[4],e,f,g,h=[],k
        =[],l,n,m,p;
        for(e=0;0x100>e;e++)k[(h[e]=e<<1^283*(e>>7))^e]=e;
        for(f=g=0;!c[f];f^=1||1,g=k[g]||1)for(m=g^g<<1^g<<2^g<<3^
        g<<4,m=m>>8^m&255^99,c[f]=m,d[m]=f,n=h[e=h[l=h[f]]],p=0
        x1010101*n^0x10001*e^0x101*1^0x1010100*f,n=0x101*h[m]^0
        x1010100*m,e=0;4>e;e++)a[e][f]=n=n<<24^n>>>8,b[e][m]=p=p
        <<24^p>>>8;
        for(e=0;5>e;e++)a[e]=a[e].slice(0),b[e]=b[e].slice(0)
    }
};

function t(a,b,c){

```

```

    if(4!==b.length)throw new sjcl.exception.invalid("invalid
      aes block size");
    var d=a.b[c],e=b[0]^d[0],f=b[c?3:1]^d[1],g=b[2]^d[2];
    b=b[c?1:3]^d[3];
    var h,k,l,n=d.length/4-2,m,p=4,r=[0,0,0,0];
    h=a.s[c];
    a=h[0];
    var q=h[1],v=h[2],w=h[3],x=h[4];
    for(m=0;m<n;m++)h=a[e>>>24]^q[f>>>16&255]^v[g>>>8&255]^w[b
      &255]^d[p],k=a[f>>>24]^q[g>>>16&255]^v[b>>>8&255]^w[e&255]^d
      [p+1],l=a[g>>>24]^q[b>>>16&255]^v[e>>>8&255]^w[f&255]^d[p
      +2],b=a[b>>>24]^q[e>>>16&255]^v[f>>>8&255]^w[g&255]^d[p+3],p
      +=4,e=h,f=k,g=l;
    for(m=0;4>m;m++)r[c?3&-m:m]=x[e>>>24]<<24^x[f>>>16&255]<<16^
      x[g>>>8&255]<<8^x[b&255]^d[p++],h=e,e=f,f=g,g=b,b=h;
    return r
  }

  sjcl.bitArray={
    bitSlice:function(a,b,c){
      a=sjcl.bitArray.$(a.slice(b/32),32-(b&31)).slice(1);
      return void 0===c?a:sjcl.bitArray.clamp(a,c-b)
    },
    extract:function(a,b,c){
      var d=Math.floor(-b-c&31);
      return((b+c-1^b)&-32?a[b/32|0]<<32-d^a[b/32+1|0]>>>d:a[b
        /32|0]>>>d)&(1<<c)-1
    },
    concat:function(a,b){
      if(0===a.length||0===b.length)return a.concat(b);
      var c=a[a.length-1],d=sjcl.bitArray.getPartial(c);
      return 32===d?a.concat(b):sjcl.bitArray.$(b,d,c|0,a.slice
        (0,a.length-1))
    },
    bitLength:function(a){
      var b=a.length;
      return 0===b?0:32*(b-1)+sjcl.bitArray.getPartial(a[b-1])
    },
    clamp:function(a,b){
      if(32*a.length<b)return a;
      a=a.slice(0,Math.ceil(b/32));
      var c=a.length;
      b=b&31;
      0<c&&b&&(a[c-1]=sjcl.bitArray.partial(b,a[c
        -1]&2147483648>>b-1,1));
    }
  }

```

```

        return a
    },
    partial:function(a,b,c){
        return 32===a?b:(c?b|0:b<<32-a)+0x100000000000*a
    },
    getPartial:function(a){
        return Math.round(a/0x10000000000)||32
    },
    equal:function(a,b){
        if(sjcl.bitArray.bitLength(a) !== sjcl.bitArray.bitLength(b))return !1;
        var c=0,d;
        for(d=0;d<a.length;d++)c|=a[d]^b[d];
        return 0===c
    },
    $:function(a,b,c,d){
        var e;
        e=0;
        for(void 0===d&&(d=[]);32<=b;b-=32)d.push(c),c=0;
        if(0===b)return d.concat(a);
        for(e=0;e<a.length;e++)d.push(c|a[e]>>>b),c=a[e]<<32-b;
        e=a.length?a[a.length-1]:0;
        a=sjcl.bitArray.getPartial(e);
        d.push(sjcl.bitArray.partial(b+a&31,32<b+a?c:d.pop(),1));
        return d
    },
    i:function(a,b){
        return[a[0]^b[0],a[1]^b[1],a[2]^b[2],a[3]^b[3]]
    },
    byteswapM:function(a){
        var b,c;
        for(b=0;b<a.length;++b)c=a[b],a[b]=c>>>24|c>>>8&0xff00|(c&0xff00)<<8|c<<24;
        return a
    }
};

sjcl.codec.utf8String={
    fromBits:function(a){
        var b="",c=sjcl.bitArray.bitLength(a),d,e;
        for(d=0;d<c/8;d++)0==(d&3)&&(e=a[d/4]),b+=String.fromCharCode(e>>>8>>>8>>>8),e<=8;
        return decodeURIComponent(escape(b))
    },
    toBits:function(a){

```

```

        a=unescape(encodeURIComponent(a));
        var b=[],c,d=0;
        for(c=0;c<a.length;c++)d=d<<8|a.charCodeAt(c),3==(c&3)
        &&(b.push(d),d=0);
        c&3&&b.push(sjcl.bitArray.partial(8*(c&3),d));
        return b
    }
};

sjcl.codec.base64={
    B:"
    ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
    +/",
    fromBits:function(a,b,c){
        var d="",e=0,f=sjcl.codec.base64.B,g=0,h=sjcl.bitArray.
        bitLength(a);
        c&&(f=f.substr(0,62)+"-");
        for(c=0;6*d.length<h;)d+=f.charAt((g^a[c]>>>e)>>>26),6>e
        ?(g=a[c]<<6-e,e+=26,c++):(g<=6,e-=6);
        for(;d.length&3&&!b;)d+="";return d
    },
    toBits:function(a,b){
        a=a.replace(/\s|=/g,"");
        var c=[],d,e=0,f=sjcl.codec.base64.B,g=0,h;
        b&&(f=f.substr(0,62)+"-");
        for(d=0;d<a.length;d++){
            h=f.indexOf(a.charAt(d));
            if(0>h)throw new sjcl.exception.invalid("this isn't
            base64!");
            26<e?(e-=26,c.push(g^h>>>e),g=h<<32-e):(e+=6,g^=h<<32-e
            )
        }
        e&56&&c.push(sjcl.bitArray.partial(e&56,g,1));
        return c
    }
};

sjcl.codec.base64url={
    fromBits:function(a){
        return sjcl.codec.base64.fromBits(a,1,1)
    },
    toBits:function(a){
        return sjcl.codec.base64.toBits(a,1)
    }
};

```

```

sjcl.hash.sha256=function(a){
  this.b[0]||this.0();
  a?(this.F=a.F.slice(0),this.A=a.A.slice(0),this.l=a.l):this
    .reset()
};

sjcl.hash.sha256.hash=function(a){
  return(new sjcl.hash.sha256).update(a).finalize()
};

sjcl.hash.sha256.prototype={
  blockSize:512,reset:function(){
    this.F=this.Y.slice(0);
    this.A=[];
    this.l=0;return this
  },
  update:function(a){
    "string"===typeof a&&(a=sjcl.codec.utf8String.toBits(a));
    var b,c=this.A=sjcl.bitArray.concat(this.A,a);
    b=this.l;
    a=this.l=b+sjcl.bitArray.bitLength(a);
    if(0x1fffffffffffff<a)throw new sjcl.exception.invalid("
    Cannot hash more than 2^53 - 1 bits");
    if("undefined"!==typeof Uint32Array){
      var d=new Uint32Array(c),e=0;
      for(b=512+b-(512+b&0x1fff);b<=a;b+=512)u(this,d.subarray
      (16*e,16*(e+1))),e+=1;
      c.splice(0,16*e)
    }
    else for(b=512+b-(512+b&0x1fff);b<=a;b+=512)u(this,c.
    splice(0,16));
    return this
  },
  finalize:function(){
    var a,b=this.A,c=this.F,b=sjcl.bitArray.concat(b,[sjcl.
    bitArray.partial(1,1)]);
    for(a=b.length+2;a&15;a++)b.push(0);
    b.push(Math.floor(this.l/0x100000000));
    for(b.push(this.l|0);b.length;)u(this,b.splice(0,16));
    this.reset();
    return c
  },
  Y:[],
  b:[],

```

```

0:function(){
  function a(a){
    return 0x100000000*(a-Math.floor(a))|0
  }
  for(var b=0,c=2,d,e;64>b;c++){
    e=!0;
    for(d=2;d*d<=c;d++)
      if(0==c%d){
        e=!1;break
      }
    e&&(8>b&&(this.Y[b]=a(Math.pow(c,.5))),this.b[b]=a(Math.
    pow(c,1/3)),b++)
  }
}

function u(a,b){
var c,d,e,f=a.F,g=a.b,h=f[0],k=f[1],l=f[2],n=f[3],m=f[4],p=f
[5],r=f[6],q=f[7];
for(c=0;64>c;c++)16>c?d=b[c):(d=b[c+1&15],e=b[c+14&15],d=b[c
&15]=(d>>>7^d>>>18^d>>>3^d<<25^d<<14)+(e>>>17^e>>>19^e
>>>10^e<<15^e<<13)+b[c&15]+b[c+9&15]|0),d=d+q+(m>>>6^m
>>>11^m>>>25^m<<26^m<<21^m<<7)+(r^m&(p^r))+g[c],q=r,r=p,p=
m,m=n+d|0,n=1,l=k,k=h,h=d+(k&1^n&(k^1))+(k>>>2^k>>>13^k
>>>22^k<<30^k<<19^k<<10)|0;
f[0]=f[0]+h|0;
f[1]=f[1]+k|0;
f[2]=f[2]+l|0;
f[3]=f[3]+n|0;
f[4]=f[4]+m|0;
f[5]=f[5]+p|0;
f[6]=f[6]+r|0;
f[7]=f[7]+q|0
}

sjcl.mode.gcm={
name:"gcm",
encrypt:function(a,b,c,d,e){
  var f=b.slice(0);
  b=sjcl.bitArray;
  d=d||[];
  a=sjcl.mode.gcm.C(!0,a,f,d,c,e||128);
  return b.concat(a.data,a.tag)
},
decrypt:function(a,b,c,d,e){

```

```

var f=b.slice(0),g=sjcl.bitArray,h=g.bitLength(f);
e=e||128;
d=d||[];
e<=h?(b=g.bitSlice(f,h-e),f=g.bitSlice(f,0,h-e)):(b=f,f=[])
;
a=sjcl.mode.gcm.C(!1,a,f,d,c,e);
if(!g.equal(a.tag,b))throw new sjcl.exception.corrupt("gcm:
    tag doesn't match");
return a.data
},
ka:function(a,b){
    var c,d,e,f,g,h=sjcl.bitArray.i;
    e=[0,0,0,0];
    f=b.slice(0);
    for(c=0;128>c;c++){
        (d=0!==(a[Math.floor(c/32)]&1<<31-c%32))&&(e=h(e,f));
        g=0!==(f[3]&1);
        for(d=3;0<d;d--)f[d]=f[d]>>>1|(f[d-1]&1)<<31;
        f[0]>>>=1;
        g&&(f[0]^=-0x1f000000)
    }
    return e
},
j:function(a,b,c){
    var d,e=c.length;
    b=b.slice(0);
    for(d=0;d<e;d+=4)b[0]^=0xffffffff&c[d],b[1]^=0xffffffff&c[d
        +1],b[2]^=0xffffffff&c[d+2],b[3]^=0xffffffff&c[d+3],b=sjcl
        .mode.gcm.ka(b,a);
    return b
},
C:function(a,b,c,d,e,f){
    var g,h,k,l,n,m,p,r,q=sjcl.bitArray;
    m=c.length;
    p=q.bitLength(c);
    r=q.bitLength(d);
    h=q.bitLength(e);
    g=b.encrypt([0,0,0,0]);
    96===h?(e=e.slice(0),e=q.concat(e,[1])):(e=sjcl.mode.gcm.j(
        g,[0,0,0,0],e),e=sjcl.mode.gcm.j(g,e,[0,0,Math.floor(h/0
            x100000000),h&0xffffffff]));
    h=sjcl.mode.gcm.j(g,[0,0,0,0],d);
    n=e.slice(0);
    d=h.slice(0);
    a||(d=sjcl.mode.gcm.j(g,h,c));

```



```

    for(l=0;l<m;l+=4)n[3]++,k=b.encrypt(n),c[l]^=k[0],c[l+1]^=k
      [1],c[l+2]^=k[2],c[l+3]^=k[3];
    c=q.clamp(c,p);
    a&&(d=sjcl.mode.gcm.j(g,h,c));
    a=[Math.floor(r/0x100000000),r&0xffffffff,Math.floor(p/0
      x100000000),p&0xffffffff];
    d=sjcl.mode.gcm.j(g,d,a);
    k=b.encrypt(e);
    d[0]^=k[0];
    d[1]^=k[1];
    d[2]^=k[2];
    d[3]^=k[3];
    return{tag:q.bitSlice(d,0,f),data:c}
  }
};

sjcl.misc.hmac=function(a,b){
  this.W=b||sjcl.hash.sha256;
  var c=[],d,e=b.prototype.blockSize/32;
  this.w=[new b,new b];
  a.length>e&&(a=b.hash(a));
  for(d=0;d<e;d++)c[0][d]=a[d]^909522486,c[1][d]=a[d
    ]^1549556828;
  this.w[0].update(c[0]);
  this.w[1].update(c[1]);
  this.R=new b(this.w[0])
};

sjcl.misc.hmac.prototype.encrypt=sjcl.misc.hmac.prototype.mac
  =function(a){
  if(this.aa)throw new sjcl.exception.invalid("encrypt on
    already updated hmac called!");
  this.update(a);
  return this.digest(a)
};

sjcl.misc.hmac.prototype.reset=function(){this.R=new this.W(
  this.w[0]);this.aa=!1};

sjcl.misc.hmac.prototype.update=function(a){this.aa=!0;this.R
  .update(a)};

sjcl.misc.hmac.prototype.digest=function(){var a=this.R.
  finalize(),a=(new this.W(this.w[1])).update(a).finalize();
  this.reset();return a};

```

```

sjcl.misc.pbkdf2=function(a,b,c,d,e){
c=c||1E4;
if(0>d||0>c)throw new sjcl.exception.invalid("invalid params
    to pbkdf2");
"string"===typeof a&&(a=sjcl.codec.utf8String.toBits(a));
"string"===typeof b&&(b=sjcl.codec.utf8String.toBits(b));
e=e||sjcl.misc.hmac;
a=new e(a);
var f,g,h,k,l=[],n=sjcl.bitArray;
for(k=1;32*l.length<(d||1);k++){
    e=f.a.encrypt(n.concat(b,[k]));
    for(g=1;g<c;g++)for(f=a.encrypt(f),h=0;h<f.length;h++)e[h]
        ^=f[h];
    l=l.concat(e)
}
d&&(l=n.clamp(l,d));
return l
};

sjcl.prng=function(a){
this.c=[new sjcl.hash.sha256];
this.m=[0];
this.P=0;
this.H={};
this.N=0;
this.U={};
this.Z=this.f=this.o=this.ha=0;
this.b=[0,0,0,0,0,0,0,0];
this.h=[0,0,0,0];
this.L=void 0;
this.M=a;
this.D=!1;
this.K={progress:{},seeded:{}};
this.u=this.ga=0;
this.I=1;
this.J=2;
this.ca=0x10000;
this.T=[0,48,64,96,128,192,0x100,384,512,768,1024];
this.da=3E4;
this.ba=80
};

sjcl.prng.prototype={
randomWords:function(a,b){

```

```

var c=[],d;
d=this.isReady(b);
var e;
if(d===this.u)throw new sjcl.exception.notReady("generator
  isn't seeded");
if(d&this.J){
  d=!(d&this.I);
  e=[];
  var f=0,g;
  this.Z=e[0]=(new Date).valueOf()+this.da;
  for(g=0;16>g;g++)e.push(0x100000000*Math.random()|0);
  for(g=0;g<this.c.length&&(e=e.concat(this.c[g].finalize())
),f+=this.m[g],this.m[g]=0,d||!(this.P&1<<g));g++);
  this.P>=1<<this.c.length&&(this.c.push(new sjcl.hash.
sha256),this.m.push(0));
  this.f-=f;f>this.o&&(this.o=f);
  this.P++;
  this.b=sjcl.hash.sha256.hash(this.b.concat(e));
  this.L=new sjcl.cipher.aes(this.b);
  for(d=0;4>d&&(this.h[d]=this.h[d]+1|0,!this.h[d]);d++);
}
for(d=0;d<a;d+=4)0===(d+1)%this.ca&&y(this),e=z(this),c.
  push(e[0],e[1],e[2],e[3]);
y(this);
return c.slice(0,a)
},
setDefaultParanoia:function(a,b){
  if(0===a&&"Setting paranoia=0 will ruin your security; use
    it only for testing"!==b)throw new sjcl.exception.invalid
    ("Setting paranoia=0 will ruin your security; use it only
    for testing");
  this.M=a
},
addEntropy:function(a,b,c){
  c=c||"user";
  var d,e,f=(new Date).valueOf(),g=this.H[c],h=this.isReady()
    ,k=0;
  d=this.U[c];
  void 0===d&&(d=this.U[c]=this.ha++);
  void 0===g&&(g=this.H[c]=0);
  this.H[c]=(this.H[c]+1)%this.c.length;
  switch(typeof a){
    case "number":void 0===b&&(b=1);
      this.c[g].update([d,this.N++,1,b,f,1,a|0]);
      break;

```

```

    case "object":c=Object.prototype.toString.call(a);
    if("[object Uint32Array]"===c){
        e=[];
        for(c=0;c<a.length;c++)e.push(a[c]);a=e
    }
    else for("[object Array]"!==c&&(k=1),c=0;c<a.length&&!k;c
++)"number"!==typeof a[c]&&(k=1);
    if(!k){
        if(void 0===b)for(c=b=0;c<a.length;c++)for(e=a[c];0<e;)
b++,e=e>>>1;
        this.c[g].update([d,this.N++,2,b,f,a.length].concat(a))
    }
    break;
    case "string":void 0===b&&(b=a.length);
    this.c[g].update([d,this.N++,3,b,f,a.length]);
    this.c[g].update(a);
    break;
    default:k=1
}
if(k)throw new sjcl.exception.bug("random: addEntropy only
supports number, array of numbers or string");
this.m[g]+=b;
this.f+=b;
h===this.u&&(this.isReady()!==this.u&&A("seeded",Math.max(
this.o,this.f)),A("progress",this.getProgress()))
},
isReady:function(a){
    a=this.T[void 0!==a?a:this.M];
    return this.o&&this.o>=a?this.m[0]>this.ba&&(new Date).
valueOf()>this.Z?this.J|this.I:this.I:this.f>=a?this.J|
this.u:this.u
},
getProgress:function(a){
    a=this.T[a?a:this.M];
    return this.o>=a?1:this.f>a?1:this.f/a
},
startCollectors:function(){
    if(!this.D){
        this.a={
            loadTimeCollector:B(this,this.ma),mouseCollector:B(this
,this.oa),keyboardCollector:B(this,this.la),
            accelerometerCollector:B(this,this.ea),touchCollector:B(
this,this.qa)};
        if(window.addEventListener)window.addEventListener("load
",this.a.loadTimeCollector,!1),window.addEventListener("

```

```

        mousemove",this.a.mouseCollector,!1),window.
        addEventListener("keypress",this.a.keyboardCollector,!1),
        window.addEventListener("devicemotion",this.a.
        accelerometerCollector,!1),window.addEventListener("
        touchmove",this.a.touchCollector,!1);
        else if(document.attachEvent)document.attachEvent("onload
        ",this.a.loadTimeCollector),document.attachEvent("
        onmousemove",this.a.mouseCollector),document.attachEvent("
        keypress",this.a.keyboardCollector);
        else throw new sjcl.exception.bug("can't attach event");
        this.D=!0
    }
},

stopCollectors:function(){
    this.D&&(window.removeEventListener?(window.
        removeEventListener("load",this.a.loadTimeCollector,!1),
        window.removeEventListener("mousemove",this.a.
        mouseCollector,!1),window.removeEventListener("keypress",
        this.a.keyboardCollector,!1),window.removeEventListener("
        devicemotion",this.a.accelerometerCollector,!1),window.
        removeEventListener("touchmove",this.a.touchCollector,!1))
        :document.detachEvent&&(document.detachEvent("onload",this
        .a.loadTimeCollector),document.detachEvent("onmousemove",
        this.a.mouseCollector),document.detachEvent("keypress",
        this.a.keyboardCollector)),this.D=!1)
},
addEventListener:function(a,b){
    this.K[a][this.ga++]=b
},
removeEventListener:function(a,b){
    var c,d,e=this.K[a],f=[];
    for(d in e)e.hasOwnProperty(d)&&e[d]==b&&f.push(d);
    for(c=0;c<f.length;c++)d=f[c],delete e[d]
},
la:function(){C(this,1)},
oa:function(a){
    var b,c;
    try{
        b=a.x||a.clientX||a.offsetX||0,c=a.y||a.clientY||a.
        offsetY||0
    }
    catch(d){
        c=b=0
    }
}

```

```

    0!=b&&0!=c&&this.addEntropy([b,c],2,"mouse");
    C(this,0)
},
qa:function(a){
    a=a.touches[0]||a.changedTouches[0];
    this.addEntropy([a.pageX||a.clientX,a.pageY||a.clientY],1,"
    touch");
    C(this,0)
},
ma:function(){C(this,2)},
ea:function(a){
    a=a.accelerationIncludingGravity.x||a.
    accelerationIncludingGravity.y||a.
    accelerationIncludingGravity.z;
    if(window.orientation){
        var b=window.orientation;"number"===typeof b&&this.
        addEntropy(b,1,"accelerometer")
    }
    a&&this.addEntropy(a,2,"accelerometer");
    C(this,0)
}
};

function A(a,b){
    var c,d=sjcl.random.K[a],e=[];
    for(c in d)d.hasOwnProperty(c)&&e.push(d[c]);
    for(c=0;c<e.length;c++)e[c](b)
}

function C(a,b){
    "undefined"!==typeof window&&window.performance&&"function
    "===typeof window.performance.now?a.addEntropy(window.
    performance.now(),b,"loadtime"):a.addEntropy((new Date).
    valueOf(),b,"loadtime")
}

function y(a){
    a.b=z(a).concat(z(a));
    a.L=new sjcl.cipher.aes(a.b)
}

function z(a){
    for(var b=0;4>b&&(a.h[b]=a.h[b]+1|0,!a.h[b]);b++);
    return a.L.encrypt(a.h)
}

```

```

function B(a,b){
return function(){
    b.apply(a,arguments)
}
}

sjcl.random=new sjcl.prng(6);

a:try{
var D,E,F,G;
if(G="undefined"!==typeof module&&module.exports){
    var H;
    try{
        H=require("crypto")
    }
    catch(a){
        H=null
    }
    G=E=H
}
if(G&&E.randomBytes)D=E.randomBytes(128),D=new Uint32Array((
    new Uint8Array(D)).buffer),sjcl.random.addEntropy(D,1024,"
    crypto['randomBytes']");
else if("undefined"!==typeof window&&"undefined"!==typeof
    Uint32Array){
    F=new Uint32Array(32);
    if(window.crypto&&window.crypto.getRandomValues)window.
        crypto.getRandomValues(F);
    else if(window.msCrypto&&window.msCrypto.getRandomValues)
        window.msCrypto.getRandomValues(F);
    else break a;
    sjcl.random.addEntropy(F,1024,"crypto['getRandomValues']")
}
}
catch(a){
"undefined"!==typeof window&&window.console&&(console.log("
    There was an error collecting entropy from the browser:"),
    console.log(a))
}

sjcl.json={
defaults:{
    v:1,iter:1E4,ks:128,ts:64,mode:"ccm",adata:"",cipher:"aes"
},

```

```

ja:function(a,b,c,d){
  c=c||{};
  d=d||{};
  var e=sjcl.json,f=e.g({iv:sjcl.random.randomWords(4,0)},e.defaults),g;
  e.g(f,c);
  c=f.adata;
  "string"===typeof f.salt&&(f.salt=sjcl.codec.base64.toBits(f.salt));
  "string"===typeof f.iv&&(f.iv=sjcl.codec.base64.toBits(f.iv));
  if(!sjcl.mode[f.mode]||!sjcl.cipher[f.cipher]||"string"===typeof a&&100>f.iter||64!==f.ts&&96!==f.ts&&128!==f.ts||128!==f.ks&&192!==f.ks&&0x100!==f.ks||2>f.iv.length||4<f.iv.length)throw new sjcl.exception.invalid("json encrypt: invalid parameters");
  "string"===typeof a?(g=sjcl.misc.cachedPbkdf2(a,f),a=g.key.slice(0,f.ks/32),f.salt=g.salt):sjcl.ecc&&a instanceof sjcl.ecc.elGamal.publicKey&&(g=a.kem(),f.kemtag=g.tag,a=g.key.slice(0,f.ks/32));
  "string"===typeof b&&(b=sjcl.codec.utf8String.toBits(b));
  "string"===typeof c&&(f.adata=c=sjcl.codec.utf8String.toBits(c));
  g=new sjcl.cipher[f.cipher](a);
  e.g(d,f);
  d.key=a;
  f.ct="ccm"===f.mode&&sjcl.arrayBuffer&&sjcl.arrayBuffer.ccm&&b instanceof ArrayBuffer?sjcl.arrayBuffer.ccm.encrypt(g,b,f.iv,c,f.ts):sjcl.mode[f.mode].encrypt(g,b,f.iv,c,f.ts);
  return f
},
encrypt:function(a,b,c,d){
  var e=sjcl.json,f=e.ja.apply(e,arguments);
  return e.encode(f)
},
ia:function(a,b,c,d){
  c=c||{};
  d=d||{};
  var e=sjcl.json;
  b=e.g(e.g(e.g({},e.defaults),b),c,!0);
  var f,g;
  f=b.adata;
  "string"===typeof b.salt&&(b.salt=sjcl.codec.base64.toBits(b.salt));
  "string"===typeof b.iv&&(b.iv=sjcl.codec.base64.toBits(b.iv));

```



```

    ));
    if(!sjcl.mode[b.mode]||!sjcl.cipher[b.cipher]||"string"===
        typeof a&&100>=b.iter||64!==b.ts&&96!==b.ts&&128!==b.ts
        ||128!==b.ks&&192!==b.ks&&0x100!==b.ks||!b.iv||2>b.iv.
        length||4<b.iv.length)throw new sjcl.exception.invalid("
        json decrypt: invalid parameters");
    "string"===typeof a?(g=sjcl.misc.cachedPbkdf2(a,b),a=g.key.
        slice(0,b.ks/32),b.salt=g.salt):sjcl.ecc&&a instanceof
        sjcl.ecc.elGamal.secretKey&&(a=a.unkem(sjcl.codec.base64.
        toBits(b.kemtag)).slice(0,b.ks/32));
    "string"===typeof f&&(f=sjcl.codec.utf8String.toBits(f));
    g=new sjcl.cipher[b.cipher](a);
    f="ccm"===b.mode&&sjcl.arrayBuffer&&sjcl.arrayBuffer.ccm&&b
        .ct instanceof ArrayBuffer?sjcl.arrayBuffer.ccm.decrypt(g,
        b.ct,b.iv,b.tag,f,b.ts):sjcl.mode[b.mode].decrypt(g,b.ct,b
        .iv,f,b.ts);
    e.g(d,b);
    d.key=a;
    return 1===c.raw?f:sjcl.codec.utf8String.fromBits(f)
},
decrypt:function(a,b,c,d){
    var e=sjcl.json;
    return e.ia(a,e.decode(b),c,d)
},
encode:function(a){
    var b,c="{",d="";
    for(b in a)if(a.hasOwnProperty(b)){
        if(!b.match(/^[-a-z0-9]+$/i))throw new sjcl.exception.
        invalid("json encode: invalid property name");
        c+=d+"'"+b+"'";
        d=",";
        switch(typeof a[b]){
            case "number":case "boolean":c+=a[b];
            break;
            case "string":c+='"'+escape(a[b])+'"';
            break;
            case "object":c+='"'+sjcl.codec.base64.fromBits(a[b
            ],0)+"'";
            break;
            default:throw new sjcl.exception.bug("json encode:
            unsupported type");
        }
    }
    return c+"}"
},

```

```

decode:function(a){
  a=a.replace(/\s/g,"");
  if(!a.match(/^{\.*\}$/))throw new sjcl.exception.invalid("
    json decode: this isn't json!");
  a=a.replace(/^{\|\\}\$/g,"").split(/,/);
  var b={},c,d;
  for(c=0;c<a.length;c++){
    if(!(d=a[c].match(/^\\s*(?:("[']?)([a-z][a-z0-9]*)\\1)\\s*:\\
      s*(?:(-?\\d+)|"([a-z0-9+\\/%*_.@=-]*)"|(true|false))$/i)))
      throw new sjcl.exception.invalid("json decode: this isn't
        json!");
    null!=d[3]?b[d[2]]=parseInt(d[3],10):null!=d[4]?b[d[2]]=d
      [2].match(/^(ct|adata|salt|iv)$/)?sjcl.codec.base64.toBits
      (d[4]):unescape(d[4]):null!=d[5]&&(b[d[2]]="true"===d[5])
  }
  return b
},
g:function(a,b,c){
  void 0===a&&(a={});
  if(void 0===b)return a;
  for(var d in b)if(b.hasOwnProperty(d)){
    if(c&&void 0!==a[d]&&a[d]!==b[d])throw new sjcl.exception
      .invalid("required parameter overridden");
    a[d]=b[d]
  }
  return a
},
sa:function(a,b){
  var c={},d;
  for(d in a)a.hasOwnProperty(d)&&a[d]!==b[d]&&(c[d]=a[d]);
  return c
},
ra:function(a,b){
  var c={},d;
  for(d=0;d<b.length;d++)void 0!==a[b[d]]&&(c[b[d]]=a[b[d]]);
  return c
}
};

sjcl.encrypt=sjcl.json.encrypt;

sjcl.decrypt=sjcl.json.decrypt;

sjcl.misc.pa={};

```

```

sjcl.misc.cachedPbkdf2=function(a,b){
var c=sjcl.misc.pa,d;
b=b||{};
d=b.iter||1E3;
c=c[a]=c[a]||{};
d=c[d]=c[d]||{firstSalt:b.salt&&b.salt.length?b.salt.slice(0)
:sjcl.random.randomWords(2,0)};
c=void 0===b.salt?d.firstSalt:b.salt;
d[c]=d[c]||sjcl.misc.pbkdf2(a,c,b.iter);
return{key:d[c].slice(0),salt:c.slice(0)}
};

"undefined"!==typeof module&&module.exports&&(module.exports=
  sjcl);

"function"===typeof define&&define([],function(){return sjcl
  });

(function (global, pool, math) {
var width = 256,
chunks = 6,
digits = 52,
rngname = 'random',
startdenom = math.pow(width, chunks),
significance = math.pow(2, digits),
overflow = significance * 2,
mask = width - 1,
nodecrypto;

function seedrandom(seed, options, callback) {
  var key = [];
  options = (options == true) ? { entropy: true } : (options
    || {});
  var shortseed = mixkey(flatten(
    options.entropy ? [seed, toString(pool)] :
    (seed == null) ? autoseed() : seed, 3), key);
  var arc4 = new ARC4(key);
  var prng = function() {
    var n = arc4.g(chunks),
d = startdenom,
x = 0;
    while (n < significance) {
      n = (n + x) * width;
      d *= width;
      x = arc4.g(1);
    }

```

```

    }
    while (n >= overflow) {
        n /= 2;
        d /= 2;
        x >>>= 1;
    }
    return (n + x) / d;
};

prng.int32 = function() { return arc4.g(4) | 0; }
prng.quick = function() { return arc4.g(4) / 0x100000000; }
prng.double = prng;

mixkey(tostring(arc4.S), pool);

return (options.pass || callback ||
function(prng, seed, is_math_call, state) {
    if (state) {
        if (state.S) { copy(state, arc4); }
        prng.state = function() { return copy(arc4, {}); }
    }
    if (is_math_call) { math[rngname] = prng; return seed; }
    else return prng;
})(
prng,
shortseed,
'global' in options ? options.global : (this == math),
options.state);
}

function ARC4(key) {
    var t, keylen = key.length,
    me = this, i = 0, j = me.i = me.j = 0, s = me.S = [];
    if (!keylen) { key = [keylen++]; }
    while (i < width) {
        s[i] = i++;
    }
    for (i = 0; i < width; i++) {
        s[i] = s[j = mask & (j + key[i % keylen] + (t = s[i]))];
        s[j] = t;
    }
    (me.g = function(count) {
        var t, r = 0,
        i = me.i, j = me.j, s = me.S;
        while (count--) {

```

```

        t = s[i = mask & (i + 1)];
        r = r * width + s[mask & ((s[i] = s[j = mask & (j + t)
]) + (s[j] = t))];
    }
    me.i = i; me.j = j;
    return r;
})(width);
}

function copy(f, t) {
    t.i = f.i;
    t.j = f.j;
    t.S = f.S.slice();
    return t;
};

function flatten(obj, depth) {
    var result = [], typ = (typeof obj), prop;
    if (depth && typ == 'object') {
        for (prop in obj) {
            try { result.push(flatten(obj[prop], depth - 1)); }
            catch (e) {}
        }
    }
    return (result.length ? result : typ == 'string' ? obj :
        obj + '\0');
}

function mixkey(seed, key) {
    var stringseed = seed + '', smear, j = 0;
    while (j < stringseed.length) {
        key[mask & j] =
            mask & ((smear ^= key[mask & j] * 19) + stringseed.
                charCodeAt(j++));
    }
    return toString(key);
}

function autoseed() {
    try {
        var out;
        if (nodecrypto && (out = nodecrypto.randomBytes)) {
            out = out(width);
        } else {
            out = new Uint8Array(width);

```

```

        (global.crypto || global.msCrypto).getRandomValues(out)
    };
    }
    return toString(out);
} catch (e) {
    var browser = global.navigator,
        plugins = browser && browser.plugins;
    return [+new Date, global, plugins, global.screen,
        toString(pool)];
}
}

function toString(a) {
    return String.fromCharCode.apply(0, a);
}

mixkey(math.random(), pool);

if ((typeof module) == 'object' && module.exports) {
    module.exports = seedrandom;
    try {
        nodecrypto = require('crypto');
    } catch (ex) {}
} else if ((typeof define) == 'function' && define.amd) {
    define(function() { return seedrandom; });
} else {
    math['seed' + rngname] = seedrandom;
}

})(
(typeof self !== 'undefined') ? self : this,
[],
Math
);

function generate_string() {
    var len = 18;
    var chars = "
        ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789$
        |/\!_+,. -?() [] {} <> &#^*=@";
    Math.seedrandom();
    var str = "";
    for (var i = 0; i < len; i++) {str += chars.charAt(Math.floor
        (Math.random() * chars.length));
    }
}

```

```

return str;
}

function encrypt_secret(password_part_public,
    password_part_private, secret) {
try {
    var ciphertext_base64 = btoa(sjcl.encrypt(
        password_part_private + password_part_public, secret, { "
        mode": "gcm", "ks": 256, "iter": 10000 }));
    return ciphertext_base64;
}
catch(e) {
    console.log('Error during encryption')
}
}

function decrypt_secret(password_part_public,
    password_part_private, ciphertext) {
try {
    var decrypted_secret = sjcl.decrypt(atob(
        password_part_private) + atob(password_part_public), atob(
        ciphertext));
}
catch(e) {
    console.log('Error during decryption')
}
}

var secret = 'This is the secret'

var password_part_public = generate_string();
var password_part_public_base64 = btoa(password_part_public);

var password_part_private = generate_string();
var password_part_private_base64 = btoa(password_part_private
);

var ciphertext = encrypt_secret(password_part_public,
    password_part_private, secret);
var decrypted_secret = sjcl.decrypt(atob(
    password_part_private_base64) + atob(
    password_part_public_base64), atob(ciphertext));

```

```
console.log('Secret:',secret)
console.log('Password Part Public:',password_part_public)
console.log('Password Part Private:',password_part_private)
console.log('Ciphertext:',ciphertext)
console.log('Secret Decrypted:',decrypted_secret)
```





# Bibliography

- [1] Christof Paar, Jan Pelzl, *Understanding Cryptography. A Textbook for Students and Practitioners*, Springer, 2010.
- [2] <https://en.wikipedia.org/wiki/RC4>
- [3] [A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications](#), Published by National Institute of Standards and Technology (NIST), Technology Administration, U.S. Department of Commerce.
- [4] <https://en.wikipedia.org/wiki/SHA-2>
- [5] Mario Lamberger, Florian Mendel *Higher-Order Differential Attack on Reduced SHA-256*, Institute for Applied Information Processing and Communications, Graz University of Technology, 2011.
- [6] <https://en.wikipedia.org/wiki/PBKDF2>
- [7] Meltem Sönmez Turan, Elaine Barker, William Burr, Lily Chen, *Recommendation for Password-Based Key Derivation Part 1: Storage Applications*, National Institute of Standards and Technology, 2010.
- [8] Colin Percival, *Stronger Key Derivation Via Sequential Memory-Hard Functions*, Scrypt, presented at BSDCan'09, 2009.
- [9] [RFC: PKCS #5: Password-Based Cryptography Specification Version 2.0](#)
- [10] [RFC: PKCS #5: Password-Based Cryptography Specification Version 2.1](#)
- [11] [https://en.wikipedia.org/wiki/List\\_of\\_the\\_most\\_common\\_passwords](https://en.wikipedia.org/wiki/List_of_the_most_common_passwords)

- [12] <https://en.wikipedia.org/wiki/X.509>
- [13] *RFC: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*
- [14] *RFC: Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management*
- [15] *RFC: Internet X.509 Public Key Infrastructure: Certification Path Building*
- [16] <https://en.wikipedia.org/wiki/HTTPS>
- [17] [https://en.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)
- [18] [https://en.wikipedia.org/wiki/Transport\\_Layer\\_Security](https://en.wikipedia.org/wiki/Transport_Layer_Security)
- [19] *Let's Encrypt*
- [20] *RFC: HTTP Over TLS*
- [21] *RFC: The Transport Layer Security (TLS) Protocol Version 1.2*
- [22] *RFC: The Transport Layer Security (TLS) Protocol Version 1.3*
- [23] [https://en.wikipedia.org/wiki/Internet\\_protocol\\_suite](https://en.wikipedia.org/wiki/Internet_protocol_suite)
- [24] *RFC: Requirements for Internet Hosts – Communication Layers*
- [25] *RFC: Requirements for Internet Hosts – Application and Support*
- [26] *Password Link*
- [27] *Password Link: Features*
- [28] *Password Link: Plans*
- [29] *Password Link: API*
- [30] *Password Link: API examples*
- [31] *Password Link: Contact*
- [32] David Bau, *Seedrandom*
- [33] *Stanford Javascript Crypto Library (SJCL)*
- [34] <https://en.wikipedia.org/wiki/Base64>