

PRÁCTICA: CRIPTOGRAFÍA

Objetivo:

Contestar un conjunto de preguntas / ejercicios relacionados con la materia aprendida en el curso demostrando la adquisición de conocimientos relacionados con la criptografía.

Detalles:

En esta práctica el alumno aplicará las técnicas y utilizará las diferentes herramientas vistas durante el módulo.

Cualquier password que sea necesaria tendrá un valor 123456.

Evaluación

Es obligatorio la entrega de un informe para considerar como APTA la práctica. Este informe ha de contener:

- Los enunciados seguido de las respuestas justificadas y evidenciadas.
- En el caso de que se hayan usado comandos / herramientas también se deben nombrar y explicar los pasos realizados.

El código escrito para la resolución de los problemas se entrega en archivos separados junto al informe.

Se va a valorar el proceso de razonamiento aunque no se llegue a resolver completamente los problemas. Si el código no funciona, pero se explica detalladamente la intención se valorará positivamente.

El objetivo principal de este módulo es adquirir conocimientos de criptografía y por ello se considera fundamental usar cualquier herramienta que pueda ayudar a su resolución, demostrando que no sólo se obtiene el dato sino que se tiene un conocimiento profundo del mismo. Si durante la misma no se indica claramente la necesidad de resolverlo usando programación, el alumno será libre de usar cualquier herramienta, siempre y cuando aporte las evidencias oportunas.

Ejercicios:

1. Tenemos un sistema que usa claves de 16 bytes. Por razones de seguridad vamos a proteger la clave de tal forma que ninguna persona tenga acceso directamente a la clave. Por ello, vamos a realizar un proceso de disociación de la misma, en el cuál tendremos, una clave fija en código, la cual, sólo el desarrollador tendrá acceso, y otra parte en un fichero de propiedades que llenará el Key Manager. La clave final se generará por código, realizando un XOR entre la que se encuentra en el properties y en el código.

La clave fija en código es B1EF2ACFE2BAEEFF, mientras que en desarrollo sabemos que la clave final (en memoria) es 91BA13BA21AABB12. ¿Qué valor ha puesto el Key Manager en properties para forzar dicha clave final?

- ➔ Resultado: 20553975c31055ed
- ➔ Ejecución:

The screenshot shows a terminal window with the following content:

```
1 # Ejercicio 1 de Ejercicios de Criptografia Finales
2
3 #XOR de datos binarios
4 def xor_data(binary_data_1, binary_data_2):
5     return bytes([b1 ^ b2 for b1, b2 in zip(binary_data_1, binary_data_2)])
6
7
8 m = bytes.fromhex("B1EF2ACFE2BAEFFF")
9 k = bytes.fromhex("91BA13BA21AABB12")
10
11 print(xor_data(m,k).hex())
12
13 num1=0xB1EF2ACFE2BAEFFF
14 num2=0x91BA13BA21AABB12
15 num3=(hex(num1^num2))
16 print(num3[2:])
17
18
```

At the bottom of the terminal window, there is a status bar with tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The OUTPUT tab is currently selected. Below the status bar, the output of the command is shown:

```
[Running] python -u "c:\Users\d_gar\Desktop\GitHub Cripto\criptografia\tempCodeRunnerFile.py"
20553975c31055ed
20553975c31055ed
```

La clave fija, recordemos es B1EF2ACFE2BAEEFF, mientras que en producción sabemos que la parte dinámica que se modifica en los ficheros de propiedades es B98A15BA31AEBB3F. ¿Qué clave será con la que se trabaje en memoria?

→ Resultado: 08653f75d31455c0

→ Ejecución:

The screenshot shows the KeepCoding XOR tool interface. On the left, there's a 'Recipe' panel with three steps: 'From Hex', 'XOR', and 'To Hex'. In the 'From Hex' step, the input is 'B98A15BA31AEBB3F'. In the 'XOR' step, the 'Key' is set to '1EF2ACFE2BAEEFF' (HEX) and the 'Scheme' is 'Standard'. In the 'To Hex' step, the 'Delimiter' is 'None' and 'Bytes per line' is '0'. On the right, the 'Input' field contains 'B98A15BA31AEBB3F'. The 'Output' field shows the result: '08653f75d31455c0'.

2. Dada la clave con etiqueta “cifrado-sim-aes-256” que contiene el keystore. El iv estará compuesto por el hexadecimal correspondiente a ceros binarios (“00”). Se requiere obtener el dato en claro correspondiente al siguiente dato cifrado:

```
TQ9SOMKc6aFS9SlxhfK9wT18UXpPCd505Xf5J/5nLl7Of/o0QKIWXg3nu1RRz4QWElezdrLAD5LO4US
t3aB/i50nnvJbBiG+le1ZhpR84ol=
```

Para este caso, se ha usado un AES/CBC/PKCS7. Si lo desciframos, ¿qué obtenemos?

- Obtenemos: Esto es un cifrado en bloque típico. Recuerda, vas por el buen camino. Ánimo.
- Código usado con importación de la librería usando mezcla de Python+Java por problemas de compatibilidad con C++ en KeyStore


```
C:\> Users > d_gar > Desktop > Python VS Code > ejercicio2_final.py > descifrar_mensaje
1 import subprocess
2 import os
3 from Crypto.Cipher import AES
4 from Crypto.Util.Padding import unpad
5 import base64
6
7 def extraer_clave_con_test():
8     """Usa el test que sabemos que funciona"""
9     print("Ejecutando extractor probado...")
10
11     # Código Java que SABEMOS funciona
12     java_code = '''
13     import java.io.FileInputStream;
14     import java.security.KeyStore;
15     import javax.crypto.SecretKey;
16
17     public class TestExtract {
18         public static void main(String[] args) throws Exception {
19             // Ruta ABSOLUTA - MODIFICA ESTA LÍNEA
20             String keystorePath = "C:\\\\Users\\\\d_gar\\\\Desktop\\\\KeyStore\\\\KeyStorePracticas";
21             String keystorePass = "123456";
22             String keyAlias = "cifrado-sim-aes-256";
23             String keyPass = "123456";
24
25             KeyStore ks = KeyStore.getInstance("JCEKS");
26             ks.load(new FileInputStream(keystorePath), keystorePass.toCharArray());
27
28             SecretKey key = (SecretKey) ks.getKey(keyAlias, keyPass.toCharArray());
29
30             byte[] keyBytes = key.getEncoded();
31             StringBuilder hex = new StringBuilder();
32             for (byte b : keyBytes) {
33                 hex.append(String.format("%02x", b));
34             }
35
36             System.out.print(hex.toString());
37         }
38     }
39     ...
40
41     # Crear archivo
42     with open('TestExtract.java', 'w', encoding='utf-8') as f:
43         f.write(java_code)
44
45     # Compilar
46     compile_result = subprocess.run(['javac', 'TestExtract.java'],
47                                     capture_output=True, text=True, shell=True)
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
878
879
880
881
882
883
884
885
886
887
888
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
978
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2128
2129
2129
```

```
C:\> Users\> d_gar\> Desktop\> Python VS Code\> ejercicio2_final.py\> descifrar_mensaje
 7 def extraer_clave_con_test():
 8     """Extrae la clave de la ejecución del compilador"""
 9     compile_result = subprocess.run(['javac', 'TestExtract.java'],
10                                     capture_output=True, text=True, shell=True)
11
12     if compile_result.returncode != 0:
13         print(f"Error compilando: {compile_result.stderr[:200]}")
14         # Limpiar y salir
15         if os.path.exists('TestExtract.java'):
16             os.remove('TestExtract.java')
17         return None
18
19     # Ejecutar
20     run_result = subprocess.run(['java', 'TestExtract'],
21                                capture_output=True, text=True, shell=True)
22
23     # Llimpiar
24     for f in ['TestExtract.java', 'TestExtract.class']:
25         if os.path.exists(f):
26             os.remove(f)
27
28     if run_result.returncode == 0 and run_result.stdout.strip():
29         clave = run_result.stdout.strip()
30         print(f"Clave extraída: {clave[:16]}...{clave[-16:]}")
31         return clave
32     else:
33         print(f"[!]Error ejecutando: {run_result.stderr[:200]}")
34         return None
35
36 def descifrar_mensaje(clave_hex):
37     """Descifra el mensaje"""
38     try:
39         clave_bytes = bytes.fromhex(clave_hex)
40
41         cifrado_base64 = "TQ950MK6Af59SlxhfK9wT18UxpPcd505Xf5J/5nLI7Of/o0QKIwXg3nu1RRz4QWElezdrLAD5L04US_t3aB/i50nvvJbB1g+le1ZhpR84oI="
42         cifrado_base64 = cifrado_base64.replace(" ", "")
43
44         iv = b'\x00' * 16
45         texto_cifrado_bytes = base64.b64decode(cifrado_base64)
46
47         cipher = AES.new(clave_bytes, AES.MODE_CBC, iv)
48         mensaje_des_bytes = unpad(cipher.decrypt(texto_cifrado_bytes), AES.block_size)
49
50         texto_claro = mensaje_des_bytes.decode('utf-8')
51
52         print(f"\nTEXTO DESCIFRADO:")
53         print(f"  {texto_claro}")
54
55         print(f"\nANÁLISIS:")
56         print(f"  Longitud: {len(texto_claro)} caracteres")
```

```
C:\> Users > d_gar > Desktop > Python VS Code > ejercicio2_final.py > descifrar_mensaje
 73 def descifrar_mensaje(clave_hex):
 74     print(f"\n ANÁLISIS:")
 75     print(f" Longitud: {len(texto_claro)} caracteres")
 76     print(f" Bytes UTF-8: {len(texto_claro.encode('utf-8'))}")
 77
 78     padding = 16 - (len(texto_claro.encode('utf-8')) % 16)
 79     if padding == 16:
 80         padding = 0
 81     print(f"Padding PKCS7: {padding} byte(s)")
 82
 83     return texto_claro
 84
 85 except Exception as e:
 86     print(f"[!]Error descifrando: {e}")
 87     return None
 88
 89 # --- PROGRAMA PRINCIPAL ---
107 if __name__ == "__main__":
108     print("Sistema de descifrado importando de libreria KeyStore")
109     print("=" * 50)
110
111     #Extraer clave automáticamente
112     print("\n1. EXTRAYENDO CLAVE DEL KEYSTORE...")
113     clave_hex = extraer_clave_con_test()
114     # DESCIFRAR
115     print(f"\n2. DESCIFRANDO CON CLAVE: {clave_hex[:16]}...")
116     resultado = descifrar_mensaje(clave_hex)
117
118     print("\n" + "=" * 50)
119     if resultado:
120         print("EJERCICIO 2 RESUELTO EXITOSAMENTE")
121     else:
122         print("[!]HUBO ERRORES EN LA RESOLUCIÓN")
123     print("=" * 50)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 364
Clave extraída: a2cff885901a5449...a0148fb3a426db72
2. DESCIFRANDO CON CLAVE: a2cff885901a5449...
TEXTO DESCIFRADO:
Este es un cifrado en bloque típico. Recuerda, vas por el buen camino. ñimo.

ANÁLISIS:
Longitud: 77 caracteres
Bytes UTF-8: 79
Padding PKCS7: 1 byte(s)
```

¿Qué ocurre si decidimos cambiar el padding a x923 en el descifrado?

→ En este caso como el padding es solo 1 byte el resultado es el mismo.

¿Cuánto padding se ha añadido en el cifrado?

→ Longitud texto claro: 79 bytes

→ Padding PKCS7 añadido: 1 bytes

→ Para resolverlo he añadido esto al final de mi comando original:

```
# Analisis Padding
→ longitud_texto_claro = len(mensaje_des_bytes)
→ bloques = (longitud_texto_claro + 15) // 16 # división hacia arriba
→ padding = 16 - (longitud_texto_claro % 16)
→ print(f"Longitud texto claro: {longitud_texto_claro} bytes")
→ print(f"Padding PKCS7 añadido: {padding} bytes")
```

Se valorará positivamente, obtener el dato de la clave desde el keystore mediante codificación en Python (u otro lenguaje).

3. Se requiere cifrar el texto “KeepCoding te enseña a codificar y a cifrar”. La clave para ello, tiene la etiqueta en el Keystore “cifrado-sim-chacha-256”. El nonce “9Yccn/f5nJJhAt2S”. El algoritmo que se debe usar es un Chacha20.

→ Primero he comprobado que el ChaCha20 de manera normal funciona bien:

```

1 # Ejercicio 3 ChaCha20
2
3 from Crypto.Cipher import ChaCha20
4 import base64
5
6 # Key de KeyStore
7 CLAVE_HEX = "AF9DF304748987A45605CCB9B936D33B780D03CABC81719D52383480DC3120"
8 clave_bytes = bytes.fromhex(CLAVE_HEX)
9
10 texto_original = "KeepCoding te enseña a codificar y a cifrar"
11 nonce_b64 = "9Yccn/f5nJhAt2S"
12
13 # Decodificar el nonce (está en Base64)
14 nonce_bytes = base64.b64decode(nonce_b64)
15 print(f"Nonce decodificado: {nonce_bytes.hex()}")
16 print(f"Longitud del nonce: {len(nonce_bytes)} bytes")
17
18 # Cifrar con ChaCha20
19 cipher = ChaCha20.new(key=clave_bytes, nonce=nonce_bytes)
20 texto_cifrado = cipher.encrypt(texto_original.encode('utf-8'))
21
22 # Resultados
23 print(f"\n== RESULTADOS CIFRADO CHACHA20 ==")
24 print(f"Texto original: {texto_original}")
25 print(f"Texto cifrado (hex): {texto_cifrado.hex()}")
26 print(f"Texto cifrado (Base64): {base64.b64encode(texto_cifrado).decode()}")
27 print(f"Longitud texto cifrado: {len(texto_cifrado)} bytes")
28
29 # Descifrado para comprobar que funciona en ambas direcciones
30 cipher_decrypt = ChaCha20.new(key=clave_bytes, nonce=nonce_bytes)
31 texto_descifrado = cipher_decrypt.decrypt(texto_cifrado)
32 print(f"Texto descifrado: {texto_descifrado.decode('utf-8')}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

Nonce decodificado: f5871c9ff7f99c926102dd92
Longitud del nonce: 12 bytes

== RESULTADOS CIFRADO CHACHA20 ==
Texto original: KeepCoding te enseña a codificar y a cifrar
Texto cifrado (hex): 69ac4ee74c552537a00a19bcdf7f0aaed7c9c8f769956a09bc6fadef6c3535f2211c9467067cf5c4a842ab
Texto cifrado (Base64): aax058TfU1NGAKGbyvfwqu18nI92mVagm85vre9sNTXyIRyUZwZ89cScQqs=
Longitud texto cifrado: 44 bytes
Texto descifrado: KeepCoding te enseña a codificar y a cifrar

```

¿Cómo podríamos mejorar de forma sencilla el sistema, de tal forma, que no sólo garanticemos la confidencialidad sino, además, la integridad del mismo? Se requiere obtener el dato cifrado, demuestra, tu propuesta por código, así como añadir los datos necesarios para evaluar tu propuesta de mejora.

- ➔ La manera de mejorarlo sería usando el ChaCha20-Poly1305, ya que Poly1305 añade un tag de autenticación que verifica integridad.
- ➔ Este es el código que he empleado:

```

1  # Ejercicio 3 ChaCha20 Poly305
2
3  from Crypto.Cipher import ChaCha20_Poly1305
4  import base64
5
6  # Key de KeyStore
7  CLAVE_HEX = "AF9DF30474898787A45605CCB9B936D33B780D03CABC81719D52383480DC3120"
8  clave_bytes = bytes.fromhex(CLAVE_HEX)
9
10 texto_original = "KeepCoding te enseña a codificar y a cifrar"
11 nonce_b64 = "9Yccn/f5nJJhAt2S"
12
13 # Nonce
14 nonce_bytes = base64.b64decode(nonce_b64)
15 print(f"Nonce: {nonce_b64} -> {nonce_bytes.hex()} ({len(nonce_bytes)} bytes)")
16
17 # En esta parte cifra y autentica
18 cipher = ChaCha20_Poly1305.new(key=clave_bytes, nonce=nonce_bytes)
19 texto_cifrado, tag = cipher.encrypt_and_digest(texto_original.encode('utf-8'))
20
21 print(f"\n CIFRADO COMPLETADO:")
22 print(f"Texto cifrado (Base64): {base64.b64encode(texto_cifrado).decode()}")
23 print(f"Tag autenticación (hex): {tag.hex()}")
24 print(f"Nonce (Base64): {nonce_b64}")
25
26
27 # Descifrado
28 try:
29     cipher_verif = ChaCha20_Poly1305.new(key=clave_bytes, nonce=nonce_bytes)
30     texto_descifrado = cipher_verif.decrypt_and_verify(texto_cifrado, tag)
31     print(f"\n\n Comprobacion desencriptado: {texto_descifrado.decode('utf-8')}")
32 except ValueError as e:
33     print(f"Error desencriptado: {e}")
34
35

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

[Running] python -u "c:\Users\d_gar\Desktop\GitHub Cripto\criptografia\tempCodeRunnerFile.py"
Nonce: 9Yccn/f5nJJhAt2S -> f5871c9ff7f99c926102dd92 (12 bytes)

CIFRADO COMPLETADO:
Texto cifrado (Base64): TslZICqLdX4jNmBcfbq49NQLW00iDmaql490DT5zsM1w4yFyQpkcwUC7Hho=
Tag autenticaci n (hex): 710cd4723da6d5ef37f23bee66285e7
Nonce (Base64): 9Yccn/f5nJJhAt2S

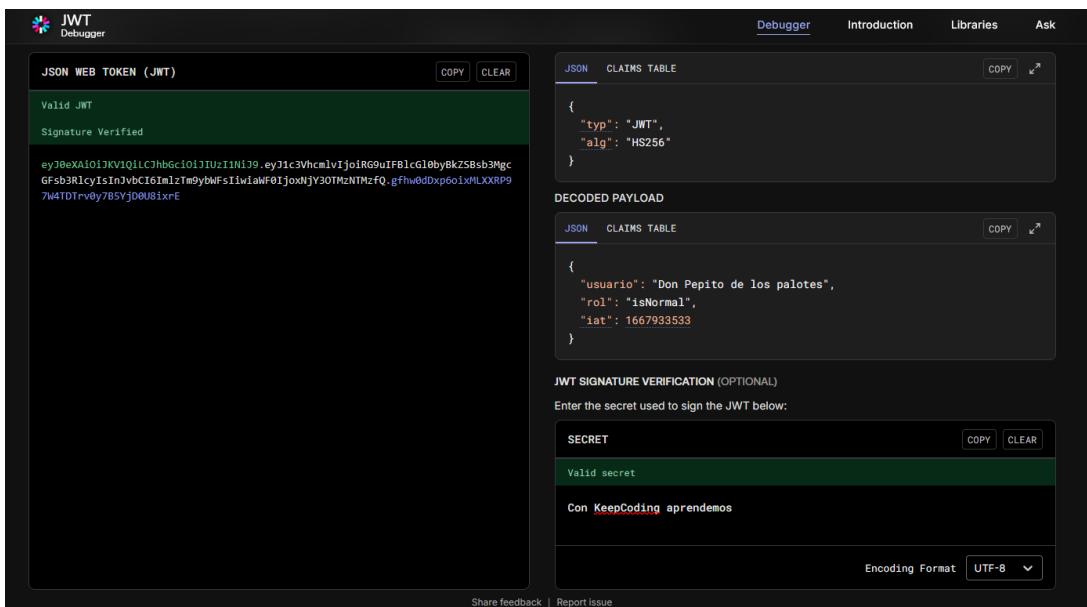
Comprobacion desencriptado: KeepCoding te ense a a codificar y a cifrar

4. Tenemos el siguiente jwt, cuya clave es “Con KeepCoding aprendemos”.

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3Vhcm1vIjoiRG9uIFBlcGl0by BkZSB

sb3MgcGFsb3RlcylsInJvbCI6ImlzTm9ybWFsliwiaWF0IjoxNjY3OTMzMzNTMzfQ .gfhw0

dDxp6oixMLXXRP97W4TDTrv0y7B5YjD0U8ixrE



¿Qué algoritmo de firma hemos realizado?

→ **Algoritmo: HS256 (HMAC con SHA-256)**

¿Cuál es el body del jwt?

```
{
  "usuario": "Don Pepito de los palotes",
  "rol": "isNormal",
  "iat": 1667933533
}
```

Un hacker está enviando a nuestro sistema el siguiente jwt:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3Vhcm1vIjoiRG9uIFBlcGl0byBkZSsb3MgcGFsb3RlcycIsInvbCI6ImlzQWRtaW4iLCJpYXQiOjE2Njc5MzM1MzN9.krgBkzCBQ5WZ8JnZHURvmnAZdg4ZMeRNv2CIAODlHRI
```

¿Qué está intentando realizar? ¿Qué ocurre si intentamos validarla con pyjwt?

→ Se ha intentado cambiar el payload y reemplazar el "isNormal" por "isAdmin" para intentar escalar privilegios de administrador.

```
{  
  "usuario": "Don Pepito de los palotes",  
  "rol": "isAdmin",  
  "iat": 1667933533  
}
```

5. El siguiente hash se corresponde con un SHA3 Keccak del texto “En KeepCoding aprendemos cómo protegernos con criptografía”.

bced1be95fb85d2ffcce9c85434d79aa26f24ce82fb4439517ea3f072d56fe

¿Qué tipo de SHA3 hemos generado?

→ SHA3-256 (Keccak de 256 bits)

Y si hacemos un SHA2, y obtenemos el siguiente resultado:

4cec5a9f85dcc5c4c6ccb603d124cf1cdc6dfe836459551a1044f4f2908aa5d63739506f

6468833d77c07cf869c488823b8d858283f1d05877120e8c5351c833

¿Qué hash hemos realizado?

→ SHA-512 (SHA2 de 512 bits)

Genera ahora un SHA3 Keccak de 256 bits con el siguiente texto: “En KeepCoding aprendemos cómo protegernos con criptografía.” ¿Qué propiedad destacarías del hash, atendiendo a los resultados anteriores?

→ Efecto avalancha: Un cambio mínimo en la entrada (1 carácter: punto final) produce un cambio drástico en la salida

6. Calcula el hmac-256 (usando la clave contenida en el Keystore) del siguiente texto:

Siempre existe más de una forma de hacerlo, y más de una solución válida.

Se debe evidenciar la respuesta. Cuidado si se usan herramientas fuera de los lenguajes de programación, por las codificaciones es mejor trabajar en hexadecimal.

HMAC-SHA256:

8ad99ac91bac38320673143235d232057a3e216e2e56660f932795c54a48f195

```
C: > Users > d_gar > Desktop > Python VS Code > # Ejercicio 6.py > ...
1  # Ejercicio 6
2
3  from Crypto.Hash import HMAC, SHA256
4
5
6  clave_bytes = bytes.fromhex("A212A51C997E14B4DF08D55967641B0677CA31E049E672A4B06861AA4D5826EB")
7
8  datos = bytes("Siempre existe más de una forma de hacerlo, y más de una solución válida.", "utf8")
9
10 hmac256 = HMAC.new(clave_bytes, msg=datos, digestmod=SHA256)
11 print(hmac256.hexdigest())
12
13 # Resultado esperado : 857d5ab916789620f35bcfe6a1a5f4ce98200180cc8549e6ec83f408e8ca0550
14
15 # Comparativa resultado facilitado con el mio
16 resultado = hmac256.hexdigest()
17 esperado = "857d5ab916789620f35bcfe6a1a5f4ce98200180cc8549e6ec83f408e8ca0550"
18 print(f"'Ok' if resultado == esperado else '[!]' {resultado}'")

PROBLEMS OUTPUT DEBUG CONSOLE ... Filter Code
[Done] exited with code=1 in 0.28 seconds

[Running] python -u "c:\Users\d_gar\Desktop\Python VS Code\Ejercicio 6.py"
857d5ab916789620f35bcfe6a1a5f4ce98200180cc8549e6ec83f408e8ca0550
Ok 857d5ab916789620f35bcfe6a1a5f4ce98200180cc8549e6ec83f408e8ca0550

[Done] exited with code=0 in 0.142 seconds
```

7. Trabajamos en una empresa de desarrollo que tiene una aplicación web, la cual requiere un login y trabajar con passwords. Nos preguntan qué mecanismo de almacenamiento de las mismas proponemos.

Tras realizar un análisis, el analista de seguridad propone un hash SHA-1. Su responsable, le indica que es una mala opción. ¿Por qué crees que es una mala opción?

→ Es un sistema que se considera obsoleto actualmente e inseguro debido a vulnerabilidades descubiertas. Además tiene problemas de colisiones demostradas.

Después de meditarlo, propone almacenarlo con un SHA-256, y su responsable le pregunta si no lo va a fortalecer de alguna forma. ¿Qué se te ocurre?

→ Para mejorar se podría añadir un “Salt”, que es único por usuario, dificultando el uso de diccionarios o rainbow tables para sacar claves por fuerza bruta. Se puede combinar con HMAC, usando una clave secreta del servidor.

Parece que el responsable se ha quedado conforme, tras mejorar la propuesta del SHA-256, no obstante, hay margen de mejora. ¿Qué propondrías?

- Se podría añadir un “Pepper” que añadiría un punto más de resistencia.
- Se podrían usar algoritmos especializados, como bcrypt, script y el más recomendado Argon2.

→ Implementar políticas de contraseñas fuertes.

8. Tenemos la siguiente API REST, muy simple.

Request:

Post /movimientos

Campo	Tipo	Requiere Confidencialidad	Observaciones
idUsuario	Number	N	Identificador
Usuario	String	S	Nombre y Apellidos
Tarjeta	Number	S	

Petición de ejemplo que se desea enviar:

```
{"idUsuario":1,"usuario":"José Manuel Barrio Barrio","tarjeta":4231212345676891}
```

Response:

Campo	Tipo	Requiere Confidencialidad	Observaciones
idUsuario	Number	N	Identificador
movTarjeta	Array	S	Formato del ejemplo
Saldo	Number	S	Tendra formato 12300 para indicar 123.00
Moneda	String	N	EUR, DOLLAR

```
{  
    "idUsuario": 1,  
    "movTarjeta": [{  
        "id": 1,  
        "comercio": "Comercio Juan",  
        "importe": 5000  
    }, {  
        "id": 2,  
        "comercio": "Rest Paquito",  
        "importe": 6000  
    }],  
    "Moneda": "EUR",  
    "Saldo": 23400  
}
```

Como se puede ver en el API, tenemos ciertos parámetros que deben mantenerse confidenciales. Así mismo, nos gustaría que nadie nos modificase el mensaje sin que nos enterásemos. Se requiere una redefinición de dicha API para garantizar la integridad y la confidencialidad de los mensajes. Se debe asumir que el sistema end to end no usa TLS entre todos los puntos.

¿Qué algoritmos usarías?

- ➔ En el escenario propuesto, se entiende que puede ser transacciones con una tarjeta bancaria, por lo que además de un sistema robusto y con sistema de garantía de integridad, se necesita que sea rápido.
- ➔ Considero que lo mejor sería un sistema AES-CBC+MAC

9. Se requiere calcular el KCV de la siguiente clave AES:

A2CFF885901A5449E9C448BA5B948A8C4EE377152B3F1ACFA0148FB3A426DB

72

Para lo cual, vamos a requerir el KCV(SHA-256) así como el KCV(AES). El KCV(SHA-256) se corresponderá con los 3 primeros bytes del SHA-256. Mientras que el KCV(AES) se corresponderá con cifrar un texto del tamaño del bloque AES (16 bytes) compuesto con ceros binarios (00), así como un iv igualmente compuesto de ceros binarios. Obviamente, la clave usada será la que queremos obtener su valor de control.

- ➔ KCV(SHA-256): DB7DF2
- ➔ KCV(AES): 5244DB

→ Código aplicado:

```
1  v import hashlib
2   from Crypto.Cipher import AES
3
4   # Clave AES-256 proporcionada
5   clave_hex = "A2CFF885901A5449E9C448BA5B948A8C4EE377152B3F1ACFA0148FB3A426DB72"
6   clave_bytes = bytes.fromhex(clave_hex)
7
8   Q
9   print("\n CALCULO KCV - KEY CHECK VALUE")
10  print(f"Clave AES-256: {clave_hex}")
11  print(f"Longitud: {len(clave_bytes)} bytes = {len(clave_bytes)*8} bits\n")
12
13  # 1. KCV(SHA-256) - SHA-256 de la clave completa
14  print("1. KCV(SHA-256):")
15  sha256_hash = hashlib.sha256(clave_bytes).digest() # bytes, no hexdigest
16  print(f"  SHA-256 completo: {sha256_hash.hex()}")
17  print(f"  Primeros 3 bytes: {sha256_hash[:3].hex()}")
18  print(f"  KCV(SHA-256): {sha256_hash[:3].hex().upper()}\n")
19
20  # 2. KCV(AES) - Cifrar 16 bytes de 0x00 con IV de 0x00
21  print("2. KCV(AES):")
22  # Bloque de 16 bytes de ceros (NO padding)
23  bloque_ceros = b'\x00' * 16 # Exactamente 16 bytes
24  iv_ceros = b'\x00' * 16
25
26  # Cifrar en CBC (sin padding - bloque exacto)
27  cipher = AES.new(clave_bytes, AES.MODE_CBC, iv_ceros)
28  texto_cifrado = cipher.encrypt(bloque_ceros)
29
30  print(f"  Texto plano: {bloque_ceros.hex()}")
31  print(f"  IV: {iv_ceros.hex()}")
32  print(f"  Texto cifrado completo: {texto_cifrado.hex()}")
33  print(f"  Primeros 3 bytes: {texto_cifrado[:3].hex()}")
34  print(f"  KCV(AES): {texto_cifrado[:3].hex().upper()}\n")
35
36  print("\n" + "*60)
37  print("RESULTADOS FINALES:")
38  print("*60)
39  print(f"KCV(SHA-256): {sha256_hash[:3].hex().upper()}")
40  print(f"KCV(AES): {texto_cifrado[:3].hex().upper()}")
41  print("*60)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 259

.. .\nvv(HE2).

=====

RESULTADOS FINALES:

=====

KCV(SHA-256): DB7DF2

KCV(AES): 5244DB

=====

10. El responsable de Raúl, Pedro, ha enviado este mensaje a RRHH:
Se debe ascender inmediatamente a Raúl. Es necesario mejorarle sus condiciones económicas un 20% para que se quede con nosotros.

Lo acompaña del siguiente fichero de firma PGP (MensajeRespoDeRaulARRHH.txt.sig). Nosotros, que pertenecemos a RRHH vamos al directorio a recuperar la clave para verificarlo. Tendremos los ficheros Pedropriv.txt y Pedro-publ.txt, con las claves privada y pública.

Las claves de los ficheros de RRHH son RRHH-priv.txt y RRHH-publ.txt que también se tendrán disponibles.

Se requiere verificar la misma, y evidenciar dicha prueba.

```
d_gar@MINI-Gaming-G1 MINGW64 ~/Desktop/PGP Bootcamp
$ gpg --verify MensajeRespoDeRaulARRHH.sig
gpg: Signature made Sun, Jun 26, 2022 1:47:01 PM RDT
gpg:           using EDDSA key 1BDE635E4EAE6E68DFAD2F7CD730BE196E466101
gpg:           issuer "pedro.pedrito.pedro@empresa.com"
gpg: Good signature from "Pedro Pedrito Pedro <pedro.pedrito.pedro@empresa.com>" [expired]
gpg: Note: This key has expired!
Primary key fingerprint: 1BDE 635E 4EAE 6E68 DFAD 2F7C D730 BE19 6E46 6101
```

Así mismo, se requiere firmar el siguiente mensaje con la clave correspondiente de las anteriores, simulando que eres personal de RRHH.

Viendo su perfil en el mercado, hemos decidido ascenderle y mejorarle un 25% su salario.
Saludos.

- Como la firma estaba caducada he tenido que editarla para no tener problemas al firmar el documento de respuesta como RRHH.

```
d_gar@MINI-Gaming-G1 MINGW64 ~/Desktop/PGP Bootcamp
$ gpg --edit-key F2B1D0E8958DF2D3BDB6A1053869803C684D287B
gpg (GnuPG) 2.4.7-unknown; Copyright (C) 2024 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Secret key is available.

sec ed25519/3869803C684D287B
    created: 2022-06-26  expired: 2024-06-25  usage: SC
    trust: unknown      validity: expired
ssb cv25519/7C1A46EA20B0546F
    created: 2022-06-26  expired: 2024-06-25  usage: E
[ expired] (1). RRHH <RRHH@RRHH>

gpg> expire
Changing expiration time for the primary key.
Please specify how long the key should be valid.
    0 = key does not expire
    <n>  = key expires in n days
    <n>w = key expires in n weeks
    <n>m = key expires in n months
    <n>y = key expires in n years
Key is valid for? (0) 2y
Key expires at Sun, Dec  5, 2027  6:54:43 PM RST
Is this correct? (y/N) y

sec ed25519/3869803C684D287B
    created: 2022-06-26  expires: 2027-12-05  usage: SC
    trust: unknown      validity: unknown
ssb cv25519/7C1A46EA20B0546F
    created: 2022-06-26  expired: 2024-06-25  usage: E
[ unknown] (1). RRHH <RRHH@RRHH>

gpg: WARNING: Your encryption subkey expires soon.
gpg: You may want to change its expiration date too.
gpg: WARNING: No valid encryption subkey left over.
gpg> save

d_gar@MINI-Gaming-G1 MINGW64 ~/Desktop/PGP Bootcamp
$ gpg --list-keys F2B1D0E8958DF2D3BDB6A1053869803C684D287B
gpg: checking the trustdb
gpg: no ultimately trusted keys found
pub   ed25519 2022-06-26 [SC] [expires: 2027-12-05]
      F2B1D0E8958DF2D3BDB6A1053869803C684D287B
uid            [ unknown] RRHH <RRHH@RRHH>
```

- Una vez la firma estaba correcta nuevamente he procedido con la firma del texto que tiene que enviar RRHH

```
d_gar@MINI-Gaming-G1 MINGW64 ~/Desktop/PGP Bootcamp
● $ gpg --clearsign --output RRHH_firmado.asc --local-user F2B1D0E8958DF2D3BDB6A1053869803C684D287B mensaje_rrhh.txt

d_gar@MINI-Gaming-G1 MINGW64 ~/Desktop/PGP Bootcamp
● $ gpg --verify RRHH_firmado.asc
gpg: Signature made Fri, Dec 5, 2025 6:58:09 PM RST
gpg:                               using EDDSA key F2B1D0E8958DF2D3BDB6A1053869803C684D287B
gpg: Good signature from "RRHH <RRHH@RRHH>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg: There is no indication that the signature belongs to the owner.
Primary key fingerprint: F2B1 D0E8 958D F2D3 BDB6 A105 3869 803C 684D 287B

d_gar@MINI-Gaming-G1 MINGW64 ~/Desktop/PGP Bootcamp
● $ cat RRHH_firmado.asc
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

Viendo su perfil en el mercado, hemos decidido ascenderle y mejorarle un 25% su salario.
Saludos.
-----BEGIN PGP SIGNATURE-----
iHUEARYKAB0WIQTysdDo1Y3y0722oQU4aYA8aE0ewUCaTMdMQAKCRA4aYA8aE0o
e906AQDbkBWYehk57YqbzqU9iq2VYJUu51q7EgLfp2/9cwgEA/1S3mxACKw0m
IJrVSOT7DJ0jHBkQGIZPEvyOehPgBAQ=
=uZt3
-----END PGP SIGNATURE-----
```

Por último, cifra el siguiente mensaje tanto con la clave pública de RRHH como la de Pedro y adjunta el fichero con la práctica.

Estamos todos de acuerdo, el ascenso será el mes que viene, agosto, si no hay sorpresas.

➔ Aquí al intentar crear y firmar el texto me daba problemas, porque aunque había actualizado la fecha de expiración, la subclase para firmar no se había actualizado, por lo que he tenido que meterme dentro para actualizar la fecha.

```
d_gar@MINI-Gaming-G1 MINGW64 ~/Desktop/PGP Bootcamp
$ gpg --edit-key F2B1D0E8958DF2D3BDB6A1053869803C684D287B
There is NO WARRANTY, to the extent permitted by law.

Secret key is available.

sec  ed25519/3869803C684D287B
      created: 2022-06-26  expires: 2027-12-05  usage: SC
      trust: ultimate    validity: ultimate
ssb  cv25519/7C1A46EA20B0546F
      created: 2022-06-26  expired: 2024-06-25  usage: E
[ultimate] (1). RRHH <RRHH@RRHH>

gpg> list

sec  ed25519/3869803C684D287B
      created: 2022-06-26  expires: 2027-12-05  usage: SC
      trust: ultimate    validity: ultimate
ssb  cv25519/7C1A46EA20B0546F
      created: 2022-06-26  expired: 2024-06-25  usage: E
[ultimate] (1). RRHH <RRHH@RRHH>

gpg> key 1 cv25519/7C1A46EA20B0546F

sec  ed25519/3869803C684D287B
      created: 2022-06-26  expires: 2027-12-05  usage: SC
      trust: ultimate    validity: ultimate
ssb* cv25519/7C1A46EA20B0546F
      created: 2022-06-26  expired: 2024-06-25  usage: E
[ultimate] (1). RRHH <RRHH@RRHH>

gpg> expire
Changing expiration time for a subkey.
Please specify how long the key should be valid.
      0 = key does not expire
      <n> = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 2y
Key expires at Sun, Dec  5, 2027  7:20:30 PM RST
Is this correct? (y/N) y

sec  ed25519/3869803C684D287B
      created: 2022-06-26  expires: 2027-12-05  usage: SC
      trust: ultimate    validity: ultimate
ssb* cv25519/7C1A46EA20B0546F
      created: 2022-06-26  expires: 2027-12-05  usage: E
[ultimate] (1). RRHH <RRHH@RRHH>

gpg> save
```

→ Una vez esto estaba correcto en ambos, Pedro y RRHH, he procedido a crear el texto y probar que se había encriptado correctamente.

```
d_gar@MINI-Gaming-G1 MINGW64 ~/Desktop/PGP Bootcamp
$ gpg --encrypt --trust-model always \
--recipient 1BDE635E4EAE6E68DFAD2F7CD730BE196E466101 \
--recipient F2B1D0E8958DF2D3BDB6A1053869803C684D287B \
--output FINAL.asc \
mensaje_final.txt

d_gar@MINI-Gaming-G1 MINGW64 ~/Desktop/PGP Bootcamp
$ cat mensaje_final.txt
Estamos todos de acuerdo, el ascenso será el mes que viene, agosto, si no hay sorpresas.

d_gar@MINI-Gaming-G1 MINGW64 ~/Desktop/PGP Bootcamp
$ cat FINAL.asc
-----BEGIN PGP MESSAGE-----
o^%o)@5oP@CooYooo/okogjo4oMo
obtuoZooQ0698tsoopcacllo/o-oooozBfSoo
d0oofooo%ooo^ooo^|?Fo oTc@ojo \-oooKooj|s3;oo;oo%"oo0raAqpIyoo-$[oPoos3oVc4oo^'ooooo,ooo5okoooRq
o7dlo5cu4<i./o>o`oo'`oo
ooo!o/"0ooadob&o4ooo:o6Yovoooo\x%ovk<oo2oooao o=ocoooo6pooooCbojoo~ooooo o\o:+Uoooo
o^o|o
oow
-----END PGP MESSAGE-----
```

→ Despues por checkear como siempre que está funcionando, he procedido a verificar.

```
d_gar@MINI-Gaming-G1 MINGW64 ~/Desktop/PGP Bootcamp
$ gpg --decrypt FINAL.asc
gpg: encrypted with cv25519 key, ID 7C1A46EA20B0546F, created 2022-06-26
    "RRHH <RRHH@RRHH>"
gpg: encrypted with cv25519 key, ID 25D6D0294035B650, created 2022-06-26
    "Pedro Pedrito Pedro <pedro.pedrito.pedro@empresa.com>"
```

Estamos todos de acuerdo, el ascenso será el mes que viene, agosto, si no hay sorpresas.

- Nuestra compañía tiene un contrato con una empresa que nos da un servicio de almacenamiento de información de videollamadas. Para lo cual, la misma nos envía la clave simétrica de cada videollamada cifrada usando un RSA-OAEP. El hash que usa el algoritmo interno es un SHA-256.

El texto cifrado es el siguiente:

b72e6fd48155f565dd2684df3ffa8746d649b11f0ed4637fc4c99d18283b32e1709b30c
96b4a8a20d5dbc639e9d83a53681e6d96f76a0e4c279f0dff76a329d04e3d3d4ad629
793eb00cc76d10fc00475eb76bfbc1273303882609957c4c0ae2c4f5ba670a4126f2f14
a9f4b6f41aa2edba01b4bd586624659fca82f5b4970186502de8624071be78cce573d
896b8eac86f5d43ca7b10b59be4acf8f8e0498a455da04f67d3f98b4cd907f27639f4b1
df3c50e05d5bf63768088226e2a9177485c54f72407fdf358fe64479677d8296ad38c6f

177ea7cb74927651cf24b01dee27895d4f05fb5c161957845cd1b5848ed64ed3b0372

2b21a526a6e447cb8ee

Las claves pública y privada las tenemos en los ficheros clave-rsa-oaep-publ.pem y clave-rsaoaep-priv.pem.

Si has recuperado la clave, vuelve a cifrarla con el mismo algoritmo. ¿Por qué son diferentes los textos cifrados?

- ➔ Clave simétrica recuperada:
e2cff885901a5449e9c448ba5b948a8c4ee377152b3f1acfa0148fb3a426db72
- ➔ Longitud: 32 bytes

```
=====
2. VOLVIENDO A CIFRAR...
-----
[OK] Re-cifrado funciona
Texto re-cifrado:
5ddbf23ee87f961d3208ab47f6b7516adc7e1b7d207eb8da763e91dd24e6924b0aec8b7bdfcf0e5fe06c215b23e5ca4889798194b59f8e5805452ccfab5c954884bccea2d2ef708e699c272a191ea2e589168d6d436661fd94e217d735f6c20d45d71765f
8fa9fee9eff5ff6c8a09e293b85b8e530d92e8c599cb14c6aa0e992464a461f1ecf9b1af2fa24821960e63415db34abbab6f066ab3a8ec104f121b2477def4c7b3ab892ce08c118bd8e809833835964eaee3bb5ed29716314ab033c018e9595839d999911
1519c87028caeae5a4fea0bead83c419a1dfb6508f5bed1ab56d19b3250e22d94a46e4143a9a8a04c92029ebb6fb674a5ae88751c...
Longitud: 256 bytes

3. COMPARACIÓN:
Original:
b726fd4d8155f565dd2684d3fffa8746d49b11f0ed4637fc4c99d18283b32e1789b30c96b4a8a20d5dbc639e9d83a53681e6d96f76a0e4c279f0dff76a329d04e3d3d4ad629793eb00cc760f00475eb76bfb1273303882609957c4c0ae2c4f5ba670a
4126f2f14af4fb6f41aa2edba01b4db5662465fca82f5b4970186502d8624071be78ccf573d899fb8eac6f5d3ca7b1b0b59e04ac8f8e0498a455da04f67d3f98b4cd987f27639f4b1df3c50e05d5bf63768088226e2a9177485c54f72407fdf358fe64
479677d8296d38f6c177ea7cb74927651c24b01dee27895d4f05fb5c161957845cd1b5848ed64ed3b83722b21a526a6e447cb8ee...
Nuevo:
5ddbf23ee87f961d3208ab47f6b7516adc7e1b7d207eb8da763e91dd24e6924b0aec8b7bdfcf0e5fe06c215b23e5ca4889798194b59f8e5805452ccfab5c954884bccea2d2ef708e699c272a191ea2e589168d6d436661fd94e217d735f6c20d45d71765f
8fa9fee9eff5ff6c8a09e293b85b8e530d92e8c599cb14c6aa0e992464a461f1ecf9b1af2fa24821960e63415db34abbab6f066ab3a8ec104f121b2477def4c7b3ab892ce08c118bd8e809833835964eaee3bb5ed29716314ab033c018e9595839d999911
1519c87028caeae5a4fea0bead83c419a1dfb6508f5bed1ab56d19b3250e22d94a46e4143a9a8a04c92029ebb6fb674a5ae88751c...
◆Son iguales? NO
```

- ➔ Los textos cifrados son diferentes porque el sistema RSA-OAEP incluye componentes aleatorios cada vez que hace un cifrado por lo que cada vez que se genera uno nuevo siempre será diferente al anterior generado.

12. Nos debemos comunicar con una empresa, para lo cual, hemos decidido usar un algoritmo como el AES/GCM en la comunicación. Nuestro sistema, usa los siguientes datos en cada comunicación con el tercero:

Key:E2CFF885901B3449E9C448BA5B948A8C4EE322152B3F1ACFA0148FB3A42

6DB74

Nonce:9Yccn/f5nJJhAt2S

¿Qué estamos haciendo mal?

- ➔ Inicialmente, según el texto, expone que la empresa esta “usando estos mismos datos en cada comunicación”, el problema viene en que el nonce debe ser

aleatorio y no usarse en mas de una ocasión, ya que si se usa más veces podría ser vulnerable si se descubre.

Cifra el siguiente texto:

He descubierto el error y no volveré a hacerlo mal

Usando para ello, la clave, y el nonce indicados. El texto cifrado presentalo en hexadecimal y en base64.

- Este es el cifrado que se pide con sus variantes, como siempre incluyo al final el descifrado para comprobar que está funcionando correctamente:

```

1  # Ejercicio 12 - AES/GCM
2
3  from Crypto.Cipher import AES
4  import base64
5
6  # Datos que vamos a usar
7  CLAVE_HEX = "E2CF885901B3449E9C448BA5B948A8C4EE322152B3F1ACFA0148FB3A426DB74"
8  clave_bytes = bytes.fromhex(CLAVE_HEX)
9
10 nonce_b64 = "9Yccn/f5nJhAt2S"
11 nonce_bytes = base64.b64decode(nonce_b64)
12
13 texto_original = "He descubierto el error y no volveré a hacerlo mal"
14
15 print("== CIFRADO AES/GCM ==")
16 print(f"Key: {CLAVE_HEX}")
17 print(f"Nonce: {nonce_b64} -> {nonce_bytes.hex()} ({len(nonce_bytes)} bytes)")
18 print(f"Texto original: {texto_original}")
19
20 # Cifrado GCM
21 cipher = AES.new(clave_bytes, AES.MODE_GCM, nonce=nonce_bytes)
22 texto_cifrado, tag = cipher.encrypt_and_digest(texto_original.encode('utf-8'))
23
24 print(f"\n RESULTADOS:")
25 print(f"Texto cifrado (hex): {texto_cifrado.hex()}")
26 print(f"Texto cifrado (Base64): {base64.b64encode(texto_cifrado).decode()}")
27 print(f"Tag autenticación (hex): {tag.hex()}")
28
29 # Descifrado para verificar que funciona a la inversa
30 try:
31     cipher_verif = AES.new(clave_bytes, AES.MODE_GCM, nonce=nonce_bytes)
32     texto_descifrado = cipher_verif.decrypt_and_verify(texto_cifrado, tag)
33     print("\n Descifrado funciona: {texto_descifrado.decode('utf-8')}")
34 except Exception as e:
35     print(f"\n [!]Error: {e}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

[Running] python -u "c:\Users\d_gar\Desktop\GitHub Cripto\criptografia\tempCodeRunnerFile.py"

```

== CIFRADO AES/GCM ==
Key: E2CF885901B3449E9C448BA5B948A8C4EE322152B3F1ACFA0148FB3A426DB74
Nonce: 9Yccn/f5nJhAt2S -> f5871cff7f99c926102dd92 (12 bytes)
Texto original: He descubierto el error y no volveré a hacerlo mal

RESULTADOS:
Texto cifrado (hex): 5dccb6261dfba29ce39431e9a013b34cbca2a4e04bb2d90149d61f4af0d4d65e2abdd9d84bba6eb8307095f5078fbfc16256d
Texto cifrado (Base64): Xcu0Jh0Puin0OUlMmgE7NvKKk4Euy2QFJ1h9K/QTMXiq92dhLum64MHCV9QePv8FiVt
Tag autenticación (hex): 6120e37aa4c3ecfd9261640dcc46410d

Descifrado funciona: He descubierto el error y no volveré a hacerlo mal

```

13. Se desea calcular una firma con el algoritmo PKCS#1 v1.5 usando las claves contenidas en los ficheros clave-rsa-oaep-priv y clave-rsa-oaep-publ.pem del mensaje siguiente:

El equipo está preparado para seguir con el proceso, necesitaremos más recursos.

¿Cuál es el valor de la firma en hexadecimal?

→ Firma:

```
a4606c518e0e2b443255e3626f3f23b77b9d5e1e4d6b3dcf90f7e118d6063950a23885c6dec
e92aa3d6eff2a72886b2552be969e11a4b7441bdeadc596c1b94e67a8f941ea998ef08b2cb3
a925c959bc当地2ca9e6e60f95b989c709b9a0b90a0c69d9eacc863bc924e70450ebbbb8736
9d721a9ec798fe66308e045417d0a56b86d84b305c555a0e766190d1ad0934a1befbbe03185
3277569f8383846d971d0daf05d023545d274f1bdd4b00e8954ba39dacc4a0875208f36d3c9
207af096ea0f0d3baa752b48545a5d79cce0c2ebb6ff601d92978a33c1a8a707c1ae1470a09
663acb6b9519391b61891bf5e06699aa0a0dbae21f0aaaa6f9b9d59f41928d
```

→

→ Código usado:

```
C:\Users\d_gar\Desktop>Python VS Code > Ejercicio_13.py ...
1 #Ejercicio 13
2
3 from Crypto.PublicKey import RSA
4 from Crypto.Signature.pkcs1_15 import PKCS115_SigScheme
5 from Crypto.Hash import SHA256
6 import binascii
7 import os
8
9 #Cargamos la clave PRIVADA porque generaremos una firma
10 my_path = os.path.abspath(os.getcwd())
11 #path_file_priv = my_path + "/claveprivada-RSA_desc_qaep.pem" # < Tengo problema al cargar el archivo de la firma esta es la ruta que yo tengo "C:\Users\d_gar\Desktop\clave-rsa-qaep-priv.pem"
12 path_file_priv = r'C:\Users\d_gar\Desktop\GitHub\Cripto\Criptografia\Practica\clave-rsa-qaep-priv.pem'
13 keypriv = RSA.importKey(open(path_file_priv).read())
14
15 with open(path_file_priv, 'rb') as f:
16     keypriv = RSA.import_key(f.read())
17
18
19 mensaje_bytes = bytes("El equipo está preparado para seguir con el proceso, necesitaremos más recursos.", "utf-8")
20 hash = SHA256.new(mensaje_bytes)
21
22 firmador=PKCS115_SigScheme(keypriv) ## Generamos un Signer
23 firma = firmador.sign(hash)
24 print("Firma: ", firma.hex())
```

```
Firma:
a4606c518e0e2b443255e3626f3f23b77b9d5e1e4d6b3dcf90f7e118d6063950a23885c6dec
e92aa3d6eff2a72886b2552be969e11a4b7441bdeadc596c1b94e67a8f941ea998ef08b2cb3
a925c959bc当地2ca9e6e60f95b989c709b9a0b90a0c69d9eacc863bc924e70450ebbbb8736
9d721a9ec798fe66308e045417d0a56b86d84b305c555a0e766190d1ad0934a1befbbe03185
3277569f8383846d971d0daf05d023545d274f1bdd4b00e8954ba39dacc4a0875208f36d3c9
207af096ea0f0d3baa752b48545a5d79cce0c2ebb6ff601d92978a33c1a8a707c1ae1470a09
663acb6b9519391b61891bf5e06699aa0a0dbae21f0aaaa6f9b9d59f41928d
```

Calcula la firma (en hexadecimal) con la curva elíptica ed25519, usando las claves ed25519priv y ed25519-pub.

```

    Ejercicio_13_Parte2.py X Ejercicio_13_Parte2.py
C: > Users > d_gar > Desktop > Python VS Code > Ejercicio_13_Parte2.py > ...
6
7     path_priv = r"C:\Users\d_gar\Desktop\GitHub_Cripto\criptografia\Practica\ed25519-priv"
8     with open(path_priv, "rb") as f:
9         private_bytes = f.read()
10
11    # Verificar si ha cargado bien [...] Aquí siempre tenemos problemas! <- Revisamos también tamaño correcto
12    print(f"Clave leída: {len(private_bytes)} bytes")
13
14    #private_key = ed25519.Ed25519PrivateKey.from_private_bytes(private_bytes)
15
16    # Tomar solo los primeros 32 bytes
17    clave_priv_32_bytes = private_bytes[:32]
18
19    private_key = ed25519.Ed25519PrivateKey.from_private_bytes(clave_priv_32_bytes)
20
21    # Mensaje
22    mensaje = "El equipo está preparado para seguir con el proceso, necesitaremos más recursos."
23    firma = private_key.sign(mensaje.encode('utf-8'))
24
25    print("Firma Ed25519:", firma.hex())

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 47 Filter Code

[Running] python -u "c:\Users\d_gar\Desktop\Python VS Code\Ejercicio_13_Parte2.py"
Clave leída: 64 bytes
Firma Ed25519: bf32592dc235a26e31e231063a1984bb75ffd9dc5550cf30105911ca4560dab52abb40e4f7e2d3af828abac1467d95d668a80395e0a71c51798bd54469b7360d

Alternativa 2? ¿i

```

    Ejercicio_13_Parte2.2.py X
C: > Users > d_gar > Desktop > Python VS Code > Ejercicio_13_Parte2.2.py > ...
1     #Ejercicio13_Parte2.2
2
3     from cryptography.hazmat.primitives.asymmetric import ed25519
4     import nacl.signing
5     import hashlib
6
7     #Cargar
8
9     path_priv = r"C:\Users\d_gar\Desktop\GitHub_Cripto\criptografia\Practica\ed25519-priv"
10    with open(path_priv, "rb") as f:
11        private_bytes = f.read()
12
13    #private_key = ed25519.Ed25519PrivateKey.from_private_bytes(private_bytes)
14
15    # Solo los primeros 32 bytes
16    private_key = ed25519.Ed25519PrivateKey.from_private_bytes(private_bytes[:32])
17
18    #signedKey = ed25519.SigningKey(private_key)
19
20    #myhash = hashlib.sha256()
21    #myhash.update(bytes("El equipo está preparado para seguir con el proceso, necesitaremos más recursos.", "utf8"))
22    #msg_hasheado= myhash.digest()
23
24    # Crean hash SHA256 del mensaje
25    mensaje = "El equipo está preparado para seguir con el proceso, necesitaremos más recursos."
26    hash_obj = hashlib.sha256(mensaje.encode('utf-8'))
27    msg_hasheado = hash_obj.digest()
28
29    #signature = signedKey.sign(msg_hasheado, encoding='hex')
30    signature = private_key.sign(msg_hasheado)
31
32
33    print("Firma Generada (64 bytes):", signature.hex())
34

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 32 Filter Code

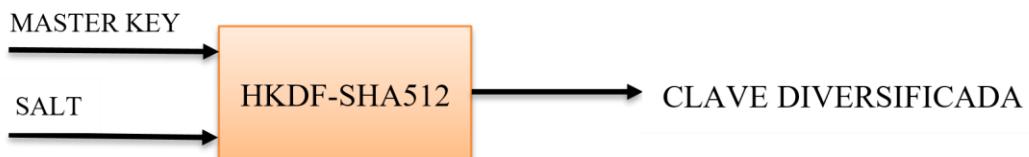
[Running] python -u "c:\Users\d_gar\Desktop\Python VS Code\Ejercicio_13_Parte2.2.py"
Firma Generada (64 bytes): fc7d764f6006aa3c31cc4ad445d094f38ae8f3597091e1be644cdcb1358c0b83d3cc60c3e8024981d6985c9495dff8de0eb289927f5214182c99e85250770c

[Done] exited with code=0 in 0.136 seconds

14. Necesitamos generar una nueva clave AES, usando para ello una HKDF (HMACbased Extractand-Expand key derivation function) con un hash SHA-512. La clave maestra requerida se encuentra en el keystore con la etiqueta "cifrado-sim-aes-

256". La clave obtenida dependerá de un identificador de dispositivo, en este caso tendrá el valor en hexadecimal:

```
e43bb4067cbcfab3bec54437b84bef4623e345682d89de9948fbb0afedc461a3
```



¿Qué clave se ha obtenido?

Last build: 4 months ago - Version 10 is here! Read about the new features [here](#)

Recipe

From Hex

Delimiter: None

Derive HKDF key

Salt: e43bb4067cbcfab3b ... HEX

Hashing function: SHA512

Extract mode: with salt

L (number of output octets): 32

Input

A2CFF885901A5449E9C448BA5B948A8C4EE377152B3F1ACFA0148FB3A426DB72|

Output

REC 64 = 1

e716754c67614c53bd9bab176022c952a08e56f07744d6c9edb8c934f52e448a|

```
C: > Users > d_gar > Desktop > Python VS Code > # Ejercicio 14.py > ...
1  # Ejercicio 14
2
3  from Crypto.Protocol.KDF import HKDF
4  from Crypto.Hash import SHA512
5
6  # Datos usados
7  salt = bytes.fromhex("e43bb4067cbcfa3bec54437b84bef4623e345682d89de9948fb0afedc461a3")
8  master = bytes.fromhex("A2CFF885901A5449E9C448BA5B948A8C4EE377152B3F1ACFA0148FB3A426DB72")
9
10 aes_key = HKDF(master, 32, salt, SHA512, 1)
11
12 print("Clave final: ", aes_key.hex())
13 expected_hex = "e716754c67614c53bd9bab176022c952a08e56f07744d6c9edb8c934f52e448a"
14 aes_key_hex = aes_key.hex()
15 print("Coincide con CyberChef?", aes_key_hex == expected_hex)

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS ... Filter Code
Running python -u C:\Users\d_gar\Desktop\Python VS Code\# Ejercicio 14.py
Clave final: e716754c67614c53bd9bab176022c952a08e56f07744d6c9edb8c934f52e448a
❖Coincide con CyberChef? True

[Done] exited with code=0 in 0.129 seconds
```

15. Nos envían un bloque TR31:

D0144D0AB00S000042766B9265B2DF93AE6E29B58135B77A2F616C8D515ACDB
E6A5626F79FA7B4071E9EE1423C6D7970FA2B965D18B23922B5B2E5657495E0
3CD857FD37018E111B

Donde la clave de transporte para desenvolver (unwrap) el bloque es:

¿Con qué algoritmo se ha protegido el bloque de clave?

➔ AES Key Derivation Binding Method

¿Para qué algoritmo se ha definido la clave?

→ AFS

¿Para qué modo de uso se ha generado?

→ Cifrado y descifrado

¿Es exportable?

- Sí, es exportable bajo clave no confiable

¿Para qué se puede usar la clave?

- D0 = Data Encryption Key (Generic) | Clave de cifrado de datos genérica

¿Qué valor tiene la clave?