



تطبيق محاكاة بنظام الأندرويد يستخدم تقنية التشفير

طرف لطرف

(م شروع فصلي)

إعداد الطلاب:

يوسف مالك ديب

حيدر محمد علي

يوشع بهاء الدين أسعد

بإشراف الدكتور:

بسيم برهوم

2017-2016

المقدمة:

إن التقدم التكنولوجي الكبير وتطور وسائل التواصل والاتصال المتنوعة ، وانفتاح العالم على بعضه ، واعتماده على إرسال شتى أنواع البيانات خلال الشبكات، كل ذلك أدى إلى إحداث خطر على تسرّب هذه البيانات ووصولها للأشخاص الخاطئين أو المنافسين ،وبالتالي أصبح أمن المعلومات الحاجة الملحة للحفاظ على السرية .

ولعل أبرز العلوم التي تبحث في سياق أمن المعلومات والحفاظ على السرية هو علم التشفير الذي كان ولا يزال العنوان الأبرز للأبحاث والدراسات وطراً عليه الكثير من التطورات التي ساهمت في تطور مفهوم أمن المعلومات ووسائله.

الهدف من المشروع:

يهدف المشروع بشكل أساسي إلى بناء تطبيق محادثة يعتمد على تقنية التشفير طرف لطرف *end to end encryption*، وتطبيق هذه التقنية من خلال خوارزميتي *AES "Advanced Encryption Standard"* و *RSA*، إضافة إلى تقنية *Node. Js* التي ظهرت حديثاً وتقوم بتحويل الحاسب إلى مخدم *Server* لتأمين عملية الاتصال بين الأجهزة التي تستخدم هذا التطبيق .

وإضافةً إلى تطبيق المحادثة سنقوم بتشفير ملفات نصية *txt* وملفات وورد *docx* وذلك لإيضاح مبدأ عمل خوارزمية التشفير *AES* .

الصفحة	الفصل
2	المقدمة
3	الهدف من المشروع
4	الفهرس
7	فهرس الرسوم والأشكال
10	فهرس الجداول
11	الدراسة المرجعية
12	الأول: التشفير
13	1.1 تعريف التشفير
13	1.1.1 خوارزمية التشفير Encryption Algorithm
14	2.1.1 خوارزمية فك التشفير Decryption Algorithm
15	2.1 التشفير طرف لطرف End to End Encryption
17	3.1 أنواع خوارزميات التشفير
17	1.3.1 التشفير باتجاهين
17	1.1.3.1 التشفير المتناظر Symmetric Cryptography
18	2.1.3.1 التشفير غير المتناظر Asymmetric Cryptography
19	3.1.3.1 مقارنة بين التشفير المتناظر وغير المتناظر
21	2.3.1 التشفير باتجاه واحد
22	الثاني: خوارزميات التشفير وتقنية Node.js
23	1.2 خوارزمية AES
25	1.1.2 خوارزمية توليد المفاتيح الفرعية
26	1.1.1.2 حساب الكلمة Wi
27	2.1.2 مبدأ عمل خوارزمية AES
28	3.1.2 خوارزمية فك التشفير
29	4.1.2 الخطوات الأساسية لخوارزمية AES

29	1.4.1.2 عملية الاستبدال <i>SubByte</i>
31	2.4.1.2 الإزاحة <i>shiftRows</i>
32	3.4.1.2 مرحلة المزج <i>Mixcolumns</i>
34	4.4.1.2 إضافة المفتاح الخاص بالتكرار
36	2.2 خوارزمية <i>RSA</i>
36	1.2.2 خطوات خوارزمية <i>RSA</i>
37	1.1.2.2 توليد المفاتيح <i>Key Generation</i>
37	2.1.2.2 نشر المفاتيح <i>Key Distribution</i>
38	3.1.2.2 التشفير <i>Encryption</i>
38	4.1.2.2 فك التشفير <i>Decryption</i>
39	3.2 تقنية <i>Node.js</i>
40	1.3.2 <i>io socket</i>
40	2.3.2 الخطوات الأساسية لإنشاء ملف <i>Js</i>
46	الثالث : الأكواد البرمجية للمشروع
47	1.3 الأكواد البرمجية لخوارزمية <i>AES</i>
47	1.1.3 التابع <i>computeWord</i>
48	2.1.3 التابع <i>Rcn</i>
48	3.1.3 التابع <i>StringToKey</i>
49	4.1.3 التابع <i>toState</i>
50	5.1.3 التابع <i>addRoundKey</i>
50	6.1.3 التابع <i>subByte</i>
50	7.1.3 التابع <i>shiftRows</i>
51	8.1.3 التابع <i>prod</i>
52	9.1.3 التابع <i>mixcolumns</i>

52	10.1.3 التابع <i>do_AES</i>
53	11.1.3 التابع <i>AES_Encryption</i>
54	12.1.3 التابع <i>subByteInv</i>
55	13.1.3 التابع <i>shiftRows</i>
55	14.1.3 التابع <i>prodInv</i>
56	15.1.3 التابع <i>MixColumnsInv</i>
57	16.1.3 التابع <i>do_AES_Decryption</i>
58	17.1.3 التابع <i>AES_DecryptionS</i>
58	18.1.3 التابع <i>stateToArray</i>
59	2.3 التوابع الخاصة بخوارزمية <i>RSA</i>
59	1.2.3 التابع <i>exp_mod</i>
61	الرابع: التطبيق العملي
62	1.4 الواجهة الأساسية للتطبيق عند الأجهزة الطرفية
71	2.4 الواجهة الخاصة بخوارزمية <i>AES</i>
71	1.2.4 تشفير ملف <i>word</i> بصيغة <i>docx</i> يحوي صورة ونص
76	2.2.4 فك تشفير ملف <i>word</i> بصيغة <i>docx</i>
78	3.2.4 تشفير ملف نصي <i>txt</i>
80	4.2.4 فك تشفير ملف نصي <i>txt</i>
81	5.2.4 تشفير صورة
83	6.2.4 فك تشفير صورة
84	النتائج والتوصيات والمقترحات
85	المراجع

فهرس الرسوم والأشكال:

الصفحة	الشكل
17	(١) أنواع التشفير
18	(٢) التشفير المتناظر
19	(٣) التشفير اللامتناظر
24	(٤) المخطط العام لخوارزمية AES
26	(٥) توليد المفاتيح الفرعية
27	(٦) ثوابت التكرار
30	(٧) صندوق التعويض <i>S_Box</i>
30	(٨) عملية الاستبدال بال <i>S_Box</i>
31	(٩) الصندوق <i>S_Box_Dec</i>
31	(١٠) عمليتي <i>shiftRows</i> و <i>invShiftRows</i>
32	(١١) عملية <i>mixcolumns</i>
33	(١٢) عملية المزج
35	(١٣) عملية إضافة المفتاح الفرعي
35	(١٤) آلية عمل خوارزمية AES
41	(١٥) عملية تهيئة <i>package.json</i>
42	(١٦) محتويات الملف <i>package.json</i>
43	(١٧) محتويات الملف <i>index.js</i>
45	(١٨) تنصيب ال <i>socket.io</i>
47	(١٩) التابع <i>ComputeWord</i>
48	(٢٠) التابع <i>Rcn</i>
49	(٢١) التابع <i>StringToKey1</i>
49	(٢٢) التابع <i>toState</i>
50	(٢٣) التابع <i>addRoundKey</i>
50	(٢٤) التابع <i>subByte</i>
51	(٢٥) التابع <i>shiftRows</i>
51	(٢٦) التابع <i>prod</i>

52	التابع <i>MixColumns</i> (٢٧ ٣)
53	التابع <i>do_AES</i> (٢٨ ٣)
54	التابع <i>AES_Encryption</i> (٢٩ ٣)
55	التابع <i>subByteInv</i> (٣٠ ٣)
55	التابع <i>shiftRowsInv</i> (٣١ ٣)
56	التابع <i>prodInv</i> (٣٢ ٣)
56	التابع <i>MixColumnsInv</i> (٣٣ ٣)
57	التابع <i>do_AES_Decryption</i> (٣٤ ٣)
58	التابع <i>AES_DecryptionS</i> (٣٥ ٣)
58	التابع <i>stateToArray</i> (٣٦ ٣)
59	التابع <i>exp_mod</i> (٣٧ ٣)
62	(٣٨ ٤) الواجهة الأساسية للتطبيق عند الأجهزة الطرفية
62	(٣٩ ٤) قيام السيرفر بعملية <i>listening</i>
63	(٤٠ ٤) اتصال أول مستخدم بالسيرفر
63	(٤١ ٤) اتصال ثاني مستخدم بالسيرفر
64	(٤٢ ٤) واجهة المرسل
64	(٤٣ ٤) واجهة المستقبل
65	(٤٤ ٤) تحديد p و q
66	(٤٥ ٤) تبادل المفتاح العام $\{e,n\}$
67	(٤٦ ٤) تحديد مفتاح التشفير
68	(٤٧ ٤) تبادل مفتاح التشفير
69	(٤٨ ٤) آلية إرسال كلمة <i>hello</i>
71	(٤٩ ٤) واجهة توضح عمل خوارزمية <i>AES</i>
72	(٥٠ ٤) ملف بصيغة <i>docx</i>
72	(٥١ ٤) عملية اختيار ملف <i>docx</i>
73	(٥٢ ٤) يبين مسار الملف
73	(٥٣ ٤) إدخال مفتاح التشفير لخوارزمية <i>AES</i>
74	(٥٤ ٤) إتمام عملية التشفير

75	(٤ ٥٥) محتويات الملف المشفر
75	(٤ ٥٦) النزر الخاص بعرض المفاتيح الفرعية
76	(٤ ٥٧) عرض المفاتيح الفرعية
77	(٤ ٥٨) آلية اختيار ملف لفك تشفيره
77	(٤ ٥٩) رسالة توضح إتمام فك التشفير
78	(٤ ٦٠) محتويات الملف بعد فك تشفيره
78	(٤ ٦١) ملف نصي <i>txt</i>
79	(٤ ٦٢) اختيار ملف نصي ومفتاح تشفير
79	(٤ ٦٣) رسالة تؤكد إتمام عملية التشفير
79	(٤ ٦٤) محتويات الملف المشفر
80	(٤ ٦٥) محتويات الملف الذي تم فك تشفيره
81	(٤ ٦٦) آلية تشفير صورة
82	(٤ ٦٧) خرج أول دورة لخوارزمية <i>AES</i>
82	(٤ ٦٨) خرج المرحلة 5 من خوارزمية <i>AES</i>

فهرس الجداول:

الصفحة	الجدول
16	(١ +) مقارنة بين محاسن ومساوئ التشفير طرف لطرف
20	(١ ٢) مقارنة بين مزايا التشفير المتناظر والتشفير الالامتناظر
21	(١ ٣) مقارنة بين عيوب التشفير المتناظر والتشفير الالامتناظر
60	(3-4) قيم المتحولات أثناء تنفيذ خوارزمية <i>RSA</i>

الدراسة المرجعية:

فيما يخص التشفير طرف لطرف *end to end Encryption* فقد كان مشروع فصلي سابق عن خوارزمية *AES* ومنها جاءت فكرة المشروع إذ قمنا باستخدام هذه الخوارزمية مع خوارزمية *RSA* للحصول على تقنية التشفير طرف لطرف *end to end Encryption* والتي تضمن لنا سرية عالية للمعلومات المتناقلة بين الأجهزة المتصلة إلى مخدم *server* واحد .

ومن خلال بحثنا تبين لنا مايلي إذ هنالك العديد من المشاريع المفيدة المتعلقة بأمن المعلومات ووسائلها وأدواته وأهميتها في الحفاظ على المعلومات، ووجدنا العديد من الأبحاث فيما يخص التشفير وأهميته وطريقة عمله وأهم الخوارزميات المتبعة في ذلك،

واستطعنا في الفصول اللاحقة تضمين كل من تقنية *NodeJs* وخوارزمية *RSA* ضمن التطبيق لم تتطرق له الأبحاث والمشاريع الأخرى إضافة إلى استخدامنا بيئة *Android Studio* والتي لم نجا في المشاريع المتعلقة بهذه الأمور من استخدامها.

الفصل الأول:

التشفير: *Encryption*

في هذا الفصل سنتعرف على مفهوم التشفير والهدف من استخدامه مع شرح بسيط لجوانب علم التشفير الأساسية.

سنستعرض في هذا الفصل أنواع خوارزميات التشفير ونركز منها على التشفير المتناظر والتشفير اللامتناظر، كما سنتطرق إلى الحديث عن التشفير طرف لطرف *End to End Encryption*.



1.1 تعريف التشفير:

علم التشفير (*Encryption*) هو دراسة تقنيات الرياضيات والمنطقية واستخدامها في الجوانب المختلفة المتعلقة بأمن المعلومات بغرض تحقيق مجموعة من الأهداف.

يستخدم التشفير في الحفاظ على سرية المعلومات عن طريق تحويلها إلى رموز عشوائية غير مفهومة وبالتالي لن يتمكن المخترق من الاطلاع على محتواها حتى وإن تمكن من الحصول عليها ، كما يستخدم في ضمان تكامل وسلامة المعلومات عن طريق التحكم في الوصول لهذه المعلومات وحصره في الشخص المصرح له فقط (أي يملك هوية صالحة) لمنع التعديل على المعلومة من قبل أشخاص غير مخولين بذلك في حال حدوث خلل وتمكن المخترق من الوصول إلى المعلومات وتعديلها ، يجب أن نتمكن من اكتشاف أن هذه البيانات قد تم تعديلها وليست على حالتها الأصلية التي يجب أن تكون عليها.

كذلك من الأهداف عدم الإنكار (*Non_Repudiation*) أي إذا أثبتنا أن الرسالة أو المعلومة صحيحة ولم يتم التلاعب بها وأنها صادرة فعلاً عن الشخص المعني ، هذا الشخص ملزم بها ولا يستطيع إنكارها ، مثلاً يطلب العميل من المصرف الذي يعمل به إلكترونياً أن يقوم بتحويل مبلغ مالي كبير من حسابه إلى حساب شخص آخر ، لا يستطيع هذا العميل إنكار ذلك وإلقاء المسؤولية على المصرف مادامنا نستطيع باستخدام علم التشفير إثبات صحة هذه الرسالة وأنها صادرة منه شخصياً وهنا يجب أن نكون ملمين بالمصطلحات التالية:

النص الأصلي (*Plain Text*) هو البيانات (أو الرسالة) المراد تشفيرها باستخدام خوارزمية التشفير.

1.1.1 خوارزمية التشفير (*Encryption Algorithm*):

هي الخوارزمية التي تقوم بتشفير النص الأصلي باستخدام المفتاح السري والنتيجة هو النص المشفر. المفتاح السري (*Secret Key*) هو عدد (أو سلسلة من البتات العشوائية) تستخدمها خوارزمية التشفير لتشفير النص الأصلي ولا بد من الحفاظ على سرية لأن يمكن كشف سرية المعلومات التي تم تشفيرها باستخدامه.

2.1.1 خوارزمية فك التشفير (Decryption Algorithm):

فك التشفير هو عملية عكسية ولا بد من استخدام خوارزمية فك التشفير لإرجاع النص المشفر لحالته الأصلية ، عادة خوارزميات التشفير عكسية ، أي يتم إعطاؤها النص الأصلي والمفتاح وينتج عن ذلك النص المشفر مخرجات والعكس صحيح، أدخل النص المشفر والمفتاح فيسترجع النص الأصلي .

جوانب علم التشفير الأساسية:

يضم علم التشفير جوانب متعددة ويقسم من حيث استخدام المفاتيح ونوعها إلى ثلاثة أقسام رئيسية :

القسم المتعلق بالمفتاح غير المتناظر أي المفتاح العلني (Public Key) ويضم ثلاثة جوانب:

التشفير

التوقيع

التحقق من الهوية

القسم المتعلق بالمفتاح المتناظر (Symmetric Key) ويضم الجوانب التالية :

1- تشفير الكتلة (Block Cipher)

2- التشفير المتسلسل (Stream Cipher)

3- توابع الاختزال (Hash Functions) التي تستخدم للتحقق من صحة الرسالة (MAC)

4- التوقيع الرقمي

5- الأعداد (البتات) شبه العشوائية

6- التحقق من الهوية

7- قسم لا يوجد لديه ارتباط بالمفاتيح ويضم :

8- توابع الاختزال "الهش"

9- عملية الإبدال

10- الأعداد (البتات) العشوائية

2.1 التشفير طرف لطرف *End To End Encryption*:

هو تقنية للاتصال يكون فيه المستخدمون المتصلون مع بعضهم فقط القادرون على قراءة الرسائل ، بشكل أساسي يمنع هذا النظام إمكانية (اختلاس السمع) من قبل شركات الاتصالات ومزودات خدمات الإنترنت أو أي شخص يقوم باختراقهما والوصول لمفاتيح التشفير الضروري لفك تشفير المحادثة والتلاعب بها.

الأنظمة مصممة لتமானع محاولات الاعتداء على أنظمة المراقبة لديها لكي لا يكون هناك طرفاً ثالثاً (*man in the middle*) قادر على فك تشفير البيانات التي يتم تراسلها أو تخزينها.

مثلاً الشركات التي تستخدم التشفير طرف لطرف تكون غير قادرة على تسليم محتوى رسائل المستخدمين لمنظمات أخرى كما سرب إدوارد سنودن عام 2013 أن تطبيق سكايب لديه ثغرات سرية تسمح لشركة مايكروسوفت بتقديم رسائل المستخدمين لوكالة الأمن القومي الأمريكية .

في نظام التشفير طرف لطرف مفاتيح التشفير يجب أن تكون معروفة لأطراف الاتصال فقط . لتحقيق هذه الغاية النظام يمكنه تشفير البيانات مستخدماً سلاسل من الرموز متفق عليها سلفاً والتي يسمي السر المشارك سلفاً.

يمكن أن يكون هناك تطبيقات ويب لا تستخدم التشفير طرف لطرف لكن بروتوكول *https* يضمن ذلك.

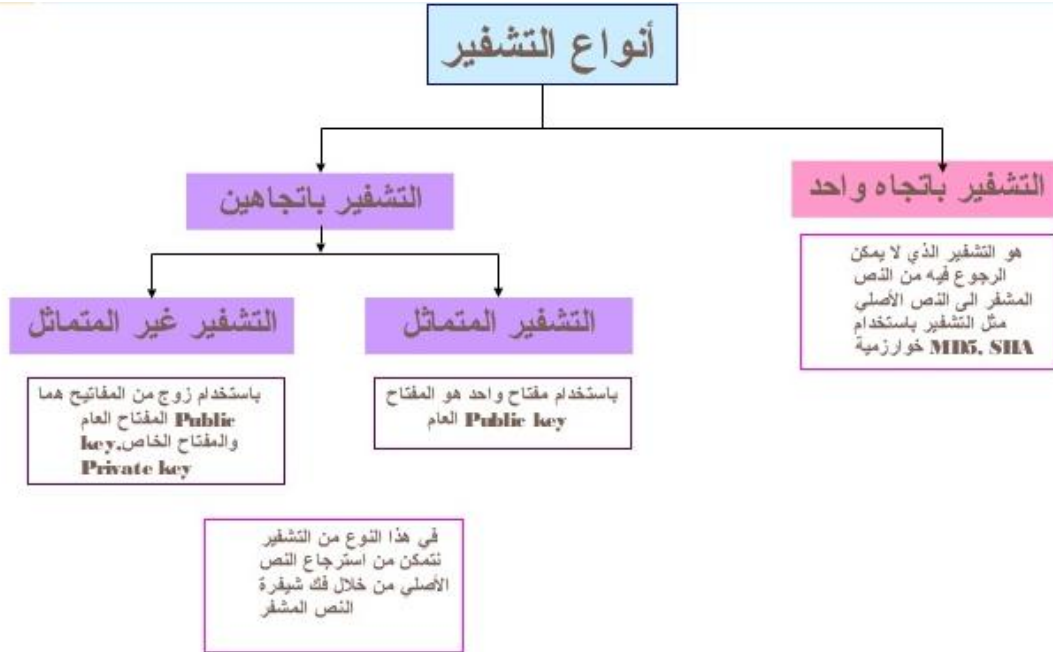
يتم استخدام هذه الطريقة في برامج المراسلة الآمنة عبر الإيميل مثل (*Secure /Multipurpose Internet Mail Extensions PGP and S/MIME*) وبرامج المحادثة الفورية مثل *imessage* و *OTR, telegram and Whatsapp*، وبرامج التخزين السحابي مثل *Tresorit and MEGA* وفي تقنية الاتصال اللاسلكي *TETRA*

وفيما يلي مقارنة بين محاسن ومساوئ التشفير طرف لطرف :

<i>End To End Encryption Disadvantages</i>	<i>End To End Encryption Advantages</i>
سيئة واحدة فقط هي أن معلومات التوجيه والعناوين والنهايات لا تكون محمية لكونها غير مشفرة أصلاً	خاصية اختلار البيانات المراد تشفيرها من التي لا تحتاج إلى تشفير
	أحجام ملفات صغيرة نسبياً مما يساعد على تنزيلها بسهولة
	معالجة خفيفة لا تحتاج لعتاديات ضخمة وتكلفة بسيطة
	مرونة تغيير المفاتيح

الجدول (1-1) مقارنة بين محاسن ومساوئ التشفير طرف لطرف

3.1 أنواع خوارزميات التشفير:



الشكل (1-1) يبين أنواع التشفير

1.3.1 التشفير باتجاهين:

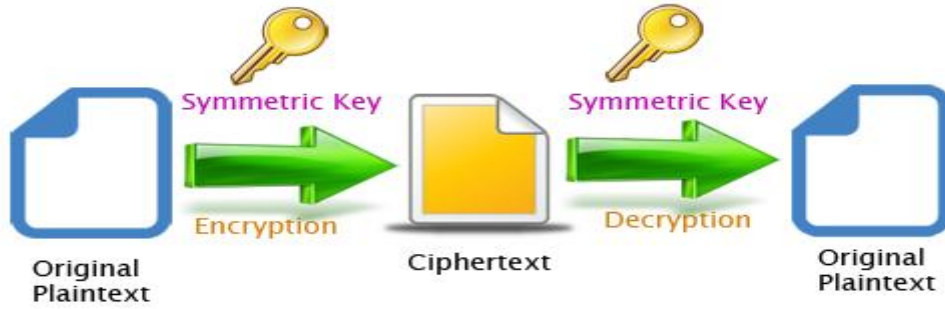
تستخدم هذه الطريقة من التشفير عندما نكون بحاجة لاستعادة المعلومات التي قمنا بتشفيرها أي إعادتها إلى النص الأصلي.
أنواعه:

1.1.3.1 التشفير المتماثل (المتناظر) (Symmetric Cryptography):

سمي بالمتناظر لأن مفتاح التشفير وفك التشفير متناظران أو بعبارة أخرى مفتاح التشفير المستخدم في تشفير البيانات هو نفسه المستخدم في فك تشفير هذه البيانات ، إذا لم يتم استخدام نفس المفتاح فبالطبع سيكون الناتج مختلفاً عن البيانات الأصلية (Plain Text).
هناك خوارزميات كثيرة ومشهورة من هذا النوع لعل أشهر خوارزمية معيار تشفير البيانات DES وهي أول معيار عالمي في التشفير ظهرت في منتصف سبعينيات القرن الماضي ، وانتشر

استخدامها بشكل واسع ، ثم ظهرت في العام 2001 خوارزمية معيار التشفير المتقدم *AES* كمعيار عالمي جديد.

التشفير المتناظر هو الأكثر استخداماً ، حيث يوفر مستوى جيد من الأمان مع مراعاة أن يكون المفتاح بالطول والعشوائية المطلوبين مع مستوى مقبول من حيث سرعة الأداء ، ولكن ما يعيبه هو صعوبة توزيع المفاتيح ، حيث أنه لا يمكن فك التشفير إلا بنفس المفتاح.



الشكل (1-2) يبين التشفير المتناظر

2.1.3.1 التشفير غير المتماثل (غير المتناظر) (*Asymmetric Cryptography*):

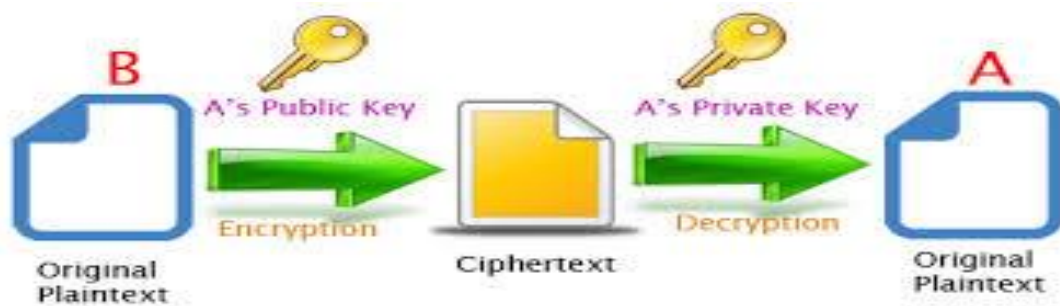
في العام 1976 قدم العالمان *Martin Hellman*، *Whitfield Diffie* ورقة بحث علمي في مجال التشفير اقترحا فيها استخدام التشفير غير المتناظر أي استخدام زوجين من المفاتيح أحدهما للتشفير (يكون علنياً ومعروفاً للجميع) والآخر لفك التشفير (على صاحبه أن يقيه سرّاً) مما يوفر حلاً لمشكلة توزيع المفاتيح لأن مفتاح التشفير علني ويمكن توزيعه بسهولة ، ثم ظهرت بعد ذلك في السنوات اللاحقة عدة خوارزميات عملية تطبق هذا المفهوم ومن أشهرها *RSA* التي تستخدم للتشفير والتوقيع الرقمي ، وخوارزمية *DSA* التي تستخدم للتوقيع .

وفي هذا النوع من التشفير كل شخص لديه مفتاحان أحدهما خاص و (سري) يستخدم لفك التشفير أو لتوقيع الرسائل والآخر علني يستخدم للتشفير (ويمكن توليده من المفتاح الخاص والعكس غير صحيح) أو للتأكد من صحة التوقيع ، إذا أردت إرسال رسالة نصية لشخص ما فكل ما عليك هو الحصول على مفتاحه العلني وتشفيرها به ثم إرسال الرسالة المشفرة له ولن يتمكن شخص آخر من الاطلاع عليها لأن مفتاح فك التشفير (المفتاح الخاص) سري ولا يعرفه إلا صاحبه.

النقطة المهمة هنا أن المخترق لا يمكنه فك التشفير حتى ولو عرف المفتاح العلني والذي هو غير سري أي باستطاعة كل شخص الوصول إليه.

ثم بناءً على هذه التقنية تطور مفهوم التوقيع الرقمي.

من أهم عيوب هذه التقنية هي الحاجة لاستخدام مفاتيح ذات طول كبير نسبياً 1024 بت أو أكثر مما يجعل التشفير أبطأ.



الشكل (1-3) يبين التشفير اللامتناظر

أغلب أنظمة التشفير الحديثة تستخدم كلا التقنيتين للاستفادة من ميزات كليهما.

3.1.3.1 مقارنة بين التشفير المتناظر وغير المتناظر:

كلا النوعين له مميزات وعيوبه ومن الجدير بالذكر أن المقارنة لن تكون حول مستوى أو قوة التشفير لأن كلا النوعين (باستخدام البروتوكولات والخوارزميات القياسية) يوفران مستوى جيداً من التشفير.

سيتم إجراء مقارنة بينهما لتوضيح مزايا وعيوب كل نوع:

مزايا التشفير المتناظر	مزايا التشفير غير المتناظر
يقدم سرعة عالية من الأداء) وقد تصل إلى عدة ميغا بتات في الثانية الواحدة).	المفتاح الخاص فقط هو الذي يجب أن يبقى طي الكتمان أما المفتاح العلني فيمكن نشره.
المفاتيح المستخدمة في هذا النوع أقصر بكثير من النوع الآخر بالتالي يسهل الاحتفاظ بها ونقلها.	عملية إدارة وتوزيع المفاتيح في هذا النوع أسهل بكثير.
مفاتيح هذا النوع يسهل توظيفها في تقنيات أخرى مهمة كتوليد الأرقام شبه العشوائية وتوابع الاختزال.	زوج المفاتيح الخاص والعلني يمكن الاحتفاظ بهما لفترة طويلة قد تصل لعدة سنوات.
مفاتيح هذا النوع يمكن تركيبها ودمجها لتوفير مستوى أعلى من التشفير والتعمية.	التوقيع الرقمي المعتمد على هذا النوع عملياً أفضل حيث أن المفاتيح المستخدمة في عمليتي التوقيع والتحقق من صحة التوقيع تكون أقصر بكثير.
التشفير بالمفتاح المتناظر له تاريخ طويل ويضم الكثير من الخوارزميات المشهورة التي قام الكثير من العلماء والباحثين بتجربتها واختبارها وبالتالي تقدم مستوى عالي من الموثوقية.	عند إجراء الاتصالات عبر الشبكات سيكون عدد المفاتيح المطلوب أقل عند استخدام هذا النوع من التشفير.

الجدول (1-2) مقارنة بين مزايا التشفير المتناظر والتشفير غير المتناظر

عيوب التشفير المتناظر	عيوب التشفير غير المتناظر
عند إجراء اتصال بين طرفين على الشبكة ،لابد من إبقاء المفاتيح السرية للمحافظة على سرية المحادثة	التشفير بالمفتاح العلني أبطأ كثيراً من التشفير بالمفتاح المتناظر.
في الشبكات الواسعة ستكون عملية إدارة وتوزيع المفاتيح مسألة شاقة،هذا في الواقع أكبر مساوئ التشفير المتناظر.	التشفير بالمفتاح العلني يستخدم مفاتيح أكبر بكثير من النوع المستخدم في النوع الآخر وبالتالي ستكون عملية حفظها والتعامل معها أصعب .
الحاجة إلى تغيير المفاتيح بشكل دوري،حيث أن استخدام نفس المفتاح لفترة طويلة قد يعرض المعلومات السرية للخطر .	عدد الخوارزميات المستخدمة في التشفير غير المتناظر التي أثبتت كفاءتها وإمكانية تطبيقها عملياً أقل بكثير مقارنةً بالنوع الآخر.

الجدول (1-3) مقارنة بين عيوب التشفير المتناظر وغير المتناظر

2.3.1 التشفير باتجاه واحد:

عملية يتم بموجبها تشفير المعلومات باستخدام خوارزمية التشفير ولكن لا يوجد خوارزمية لفك تشفير الرسالة الناتجة،مثل خوارزمية MD5.

الفصل الثاني

خوارزميات التشفير وتقنية *Node.js*

سنستعرض في هذا الفصل إلى الحديث عن خوارزمية *AES* وخوارزمية *RSA*، بالإضافة إلى الحديث عن تقنية *Node.js*.

١.٢ خوارزمية AES :

هي عبارة عن خوارزمية معيارية حديثة، وكانت تسمى بـ "خوارزمية *Rijndael*" نسبةً إلى أسماء مصمميها (*Rijmen & Daemen*) ولكن أطلق عليها فيما بعد اسم *AES* اختصاراً لـ *Advanced Encryption Standard* حيث اعتمدت عالمياً لأن تكون المعيار التجاري كخوارزمية للحماية من قبل *NIST* في الولايات المتحدة الأمريكية ،

وتم التوصية أوروبياً من أجل استخدامها في عمليات التشفير وفك التشفير وذلك من قبل *NESSIE* " *New European Schemes for Signature Integrity and encrypt* " وكذلك من قبل *CRYPTRE* الهيئة المسؤولة في اليابان.

خوارزمية *AES* خوارزمية كتلية *Block Cipher* تعمل بطول مفاتيح متغير (*128 bits* و *192bits* و *256 bits*) وطول الكتلة (*16 byte*) *128 bits* ويمكن أن تكون متغيرة الطول وتعتمد شكلياً المصفوفات في خطوات عملها وتقوم على تكرار عدة خطوات عدداً من المرات (كما سنرى لاحقاً) حيث أن عدد التكرارات يكون حسب طول المفتاح أي :

إذا كان طول المفتاح *128 bit* فسيكون لدينا 10 تكرارات

إذا كان طول المفتاح *192 bit* فسيكون لدينا 12 تكرار

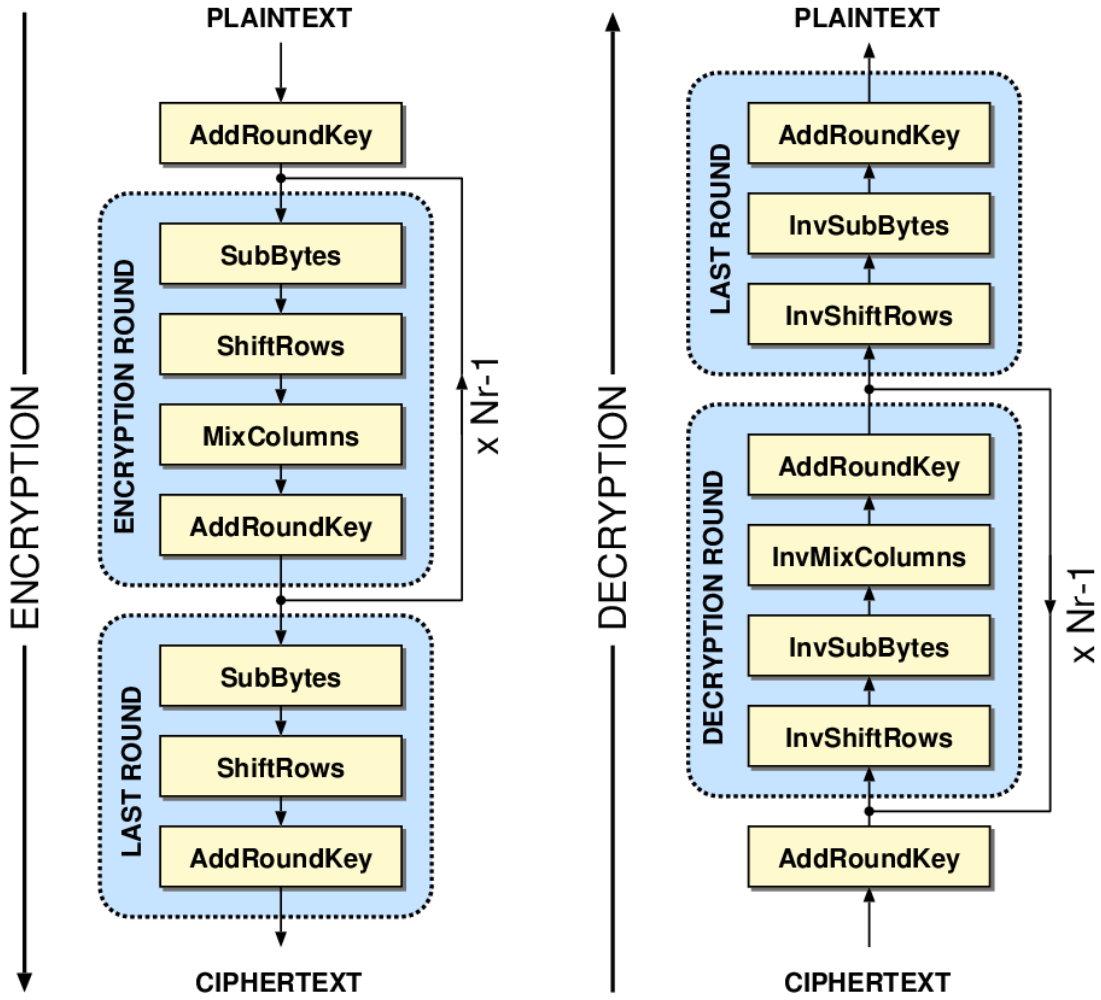
إذا كان طول المفتاح *256 bit* فسيكون لدينا 14 تكرار

وسنستخدم في مشروعنا خوارزمية *AES* بطول مفتاح *128 bit* .

بشكل عام تتكون خوارزمية *AES* من قسمين أساسيين :

معالجة عملية توليد المفاتيح المستخدمة في مراحل الخوارزمية من المفتاح الأساسي
معالجة ثمانية النص (عمليات استبدال و إزاحة ومزج استبدالي وإضافة المفتاح الخاص بالمرحلة).

ما يميز خوارزمية *AES* عن غيرها من الخوارزميات المتناظرة الأخرى أنها تعتمد على مفهوم المصفوفات في عملها ولم تعتمد في تكرارها على مبدأ *Feistel Network* المعتمد في أغلب خوارزميات التشفير المتناظرة



الشكل (2-4) يوضح المخطط العام للخوارزمية

القسم الأول: يتعلق بتوليد المفاتيح الفرعية من المفتاح الأساسي *Keys schedule*.

بفرض لدينا مفتاح أساسي مكون من 128 bit (مثله بمصفوفة 4×4) ونريد توليد 10 مفاتيح فرعية لاستخدامها في مراحل الخوارزمية.

1.1.2 خوارزمية توليد المفاتيح الفرعية :

في البداية يكون لدينا مفتاح التشفير ب $128 \text{ bit} (16 \text{ byte})$ نمثله بمصفوفة ثنائية $4*4$ وبعد ذلك نقوم بتوليد 10 مفاتيح فرعية أخرى لاستخدامها في مراحل الخوارزمية لذا نتبع الخطوات التالية :

1- نحري على العمود الأخير من المفتاح الأساسي إزاحة دورانية نحو الأعلى بمقدار موقع واحد.

نشر بنات العمود الناتج عن الخطوة رقم 1 في صندوق تعويض S_Box من خلال استبدال كل عنصر من العمود بعنصر موافق له (سطراً وعموداً) في الصندوق S_Box .

نجمع ثنائياً (أي نقوم بعملية XOR) العمود الناتج عن الخطوة رقم 2 مع العمود الأول (من اليسار) من المفتاح الأساسي .

نضع ناتج الخطوة 3 كعمود أول من اليسار في المفتاح الفرعي الأول .

للحصول على العمود الثاني من المفتاح الفرعي الأول نجمع ثنائياً ناتج الخطوة 4 مع العمود الثاني من المفتاح الأساسي .

نجمع ثنائياً ناتج الخطوة رقم 5 مع العمود الثالث من المفتاح الأساسي .

نجمع ثنائياً ناتج الخطوة رقم 6 مع العمود الرابع من المفتاح الأساسي .

وبذلك نكون قد حصلنا على المفتاح الفرعي الأول ، ولحساب المفتاح الفرعي الثاني نكرر نفس الخطوات مع استبدال المفتاح الأساسي بالمفتاح الفرعي الأول وهكذا.....

وعند الانتهاء من توليد المفاتيح يكون قد نتج لدينا 10 مفاتيح إضافة إلى المفتاح الأساسي وكل مفتاح مكون من 4 كلمات (كلمة w تعني عمود في مصفوفة المفتاح).

أي المفتاح الأساسي مكون من الكلمات $w0 \ w1 \ w2 \ w3$

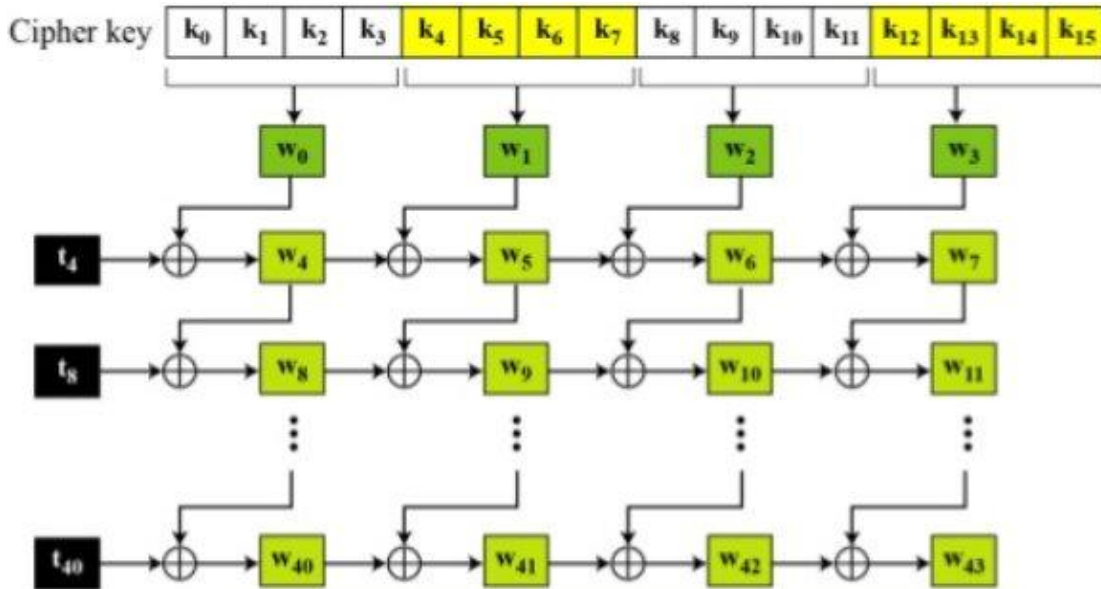
والمفتاح الفرعي الأول مكون من الكلمات $w4 \ w5 \ w6 \ w7$

والمفتاح الفرعي الثاني مكون من الكلمات $w8 \ w9 \ w10 \ w11$

.....

والمفتاح الفرعي العاشر مكون من الكلمات $w_{40} w_{41} w_{42} w_{43}$

الشكل التالي يوضح عملية توليد المفاتيح



الشكل (2-5) يوضح توليد المفاتيح الفرعية

1.1.1.2 حساب الكلمة W_i :

نقوم بحساب الكلمة W_i بحيث $i=4 \rightarrow 43$ وفق العلاقة التالية:

$$W_i = \begin{cases} W_{i-1} \text{ XOR } W_{i-4} & : i \% 4 \neq 0 \\ t_i \text{ XOR } W_{i-4} & : i \% 4 = 0 \end{cases}$$

حيث أن :

$$t_i = \text{subword}(\text{Rotword}(W_{i-1})) \text{ XOR } Rcon(i/4)$$

وال $Rcon(i/4)$ تسمى ثوابت التكرارات وكل ثابت مؤلف من 4 byte كما هو موضح بالجدول التالي:

Rcon Constants (Base 16)			
Round	Constant(Rcon)	Round	Constant(Rcon)
1	01 00 00 00	6	20 00 00 00
2	02 00 00 00	7	40 00 00 00
3	04 00 00 00	8	80 00 00 00
4	08 00 00 00	9	1B 00 00 00
5	10 00 00 00	10	36 00 00 00

الشكل (2-6) يبين ثوابت التكرار

القسم الثاني: معالجة بتات الكتلة

من أجل كل كتلة نقوم بتمثيلها بمصفوفة 4×4 تسمى *state* حيث أن كل تكرار ملئون من 4 مراحل وهي:

مرحلة الاستبدال *subByte*.

مرحلة إزاحة الأسطر *shiftRows*.

مرحلة نشر الأعمدة *mixcolumns*.

مزج الكتلة مع المفتاح الفرعي *addRoundKey*.

2.1.2 مبدأ عمل خوارزمية AES :

تعمل خوارزمية AES كما يلي :

نجمع ثنائياً المفتاح الأساسي *round key[0]* مع كتلة النص *state*.

نقوم بعملية *subByte* للكتلة الناتجة عن المرحلة رقم 1 حيث نسند لكل عنصر فيها بما يقابله

في صندوق التعويض *S_Box*.

نقوم بعملية *shiftRows* أي إزاحة دورانية نحو اليسار لأسطر المصفوفة الناتجة عن الخطوة رقم

2 وذلك حسب رقم السطر.

نقوم بمرحلة *mixcolumns* حيث نأخذ المصفوفة الناتجة عن الخطوة رقم 3 وذلك بضرب كل

عمود بالمصفوفة المربعة التالية:

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

إضافة مفتاح التكرار *addRoundKey* وذلك بالجمع الثنائي لعناصر المصفوفة الناتجة عن الخطوة رقم 4 مع عناصر المفتاح الفرعي الأول *addRoundKey[0]* .

تكرر الخطوات من 2 إلى 5 تسع مرات مع إضافة المفتاح المناسب لكل تكرار .

نقوم بعملية الاستبدال *subByte* للمصفوفة الناتجة عن الخطوة رقم 6 .

نقوم بعملية *shiftRows* للمصفوفة الناتجة عن الخطوة 7.

نقوم بإضافة المفتاح الفرعي الأخير *addRoundKey[10]* على خرج الخطوة رقم 8 .

وبذلك نحصل على الكتلة المشفرة الموافقة للكتلة الأصلية،

نلاحظ أن الخطوة الأخيرة لا تحوي على مرحلة *mixcolumns*

3.1.2 خوارزمية فك التشفير :

بفرض لدينا الكتلة المشفرة كمصفوفة 4×4 ولدينا المفاتيح من $10 \rightarrow 0$ ، عندئذ تكون الخطوات الأساسية لفك التشفير هي:

نقوم بعملية *addRoundKey[10]* أي بعملية *XOR* بين الكتلة المشفرة والمفتاح الفرعي العاشر.

نقوم بعملية *invShiftRows* وهي العملية المعاكسة لـ *shiftRows* حيث أننا نقوم بإزاحة دورانية نحو اليمين عدد من المرات يساوي دليل السطر للمصفوفة.

نقوم بعملية *invSubByte* حيث أننا نأخذ كل عنصر من الكتلة الناتجة عن الخطوة رقم 2 ونأخذ ما يقابله من صندوق اسمه *S_Box_Dec*.

نقوم بعملية *addRoundKey* مع المصفوفة الناتجة عن الخطوة رقم 3 .

نقوم بعملية *invMixColumns* حيث نقوم بضرب كل عمود من المصفوفة الناتجة مع المصفوفة الناتجة من الخطوة رقم 4 بالمصفوفة التالية :

$$\begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix}$$

نكرر الخطوات 5->2 تسع مرات .

نقوم بعملية *invShiftRows* للمصفوفة الناتجة عن المصفوفة رقم 6 .

نقوم بعملية *invSubByte* للمصفوفة الناتجة عن الخطوة رقم 7 .

نقوم بعملية *addRoundKey[0]* أي عملية *XOR* بين خرج الخطوة 8 والمفتاح الأساسي

4.1.2 الخطوات الأساسية للخوارزمية:

1.4.1.2 عملية الاستبدال *subByte* :

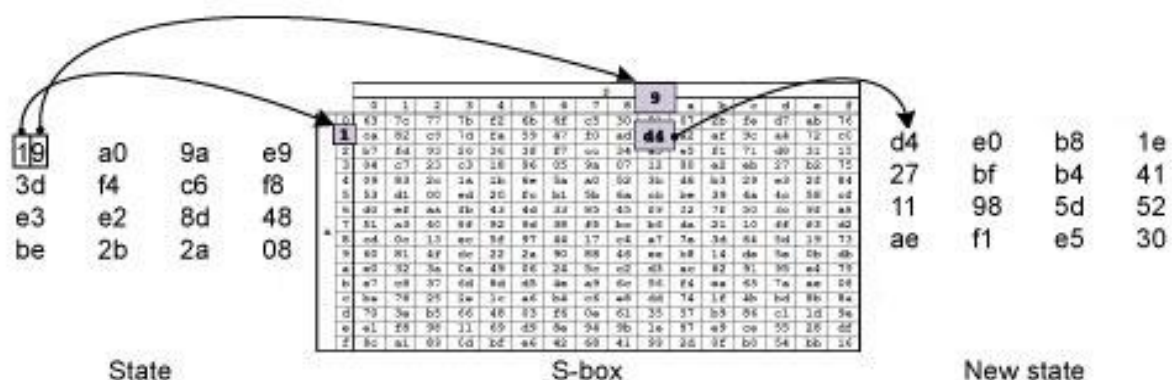
هي عملية غير خطية تستخدم *S_Box* والذي يعمل على استبدال المدخلات بمخرجات مختلفة حسب الشكل الذي سنعرضه،

تقوم عملية الاستبدال على مستوى البايت في المصفوفة فهذه العملية تستقبل بايت وتستبدله ببايت آخر باستخدام *S_Box* لكل بايت وبما أنه لدينا *128 bit* أي ما يعادل *16 byte* نحتاج إلى *16 S_Box* في هذه العملية .

	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
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

الشكل (2-7) يوضح صندوق التعويض S_Box

الشكل التالي يوضح كيف يتم استخدام الجدول لاستبدال البايت في المصفوفة مع العلم أن هذه العملية لا تأخذ بعين الاعتبار البايتات الأخرى في المصفوفة



الشكل (2-8) يوضح عملية الاستبدال بال S_Box

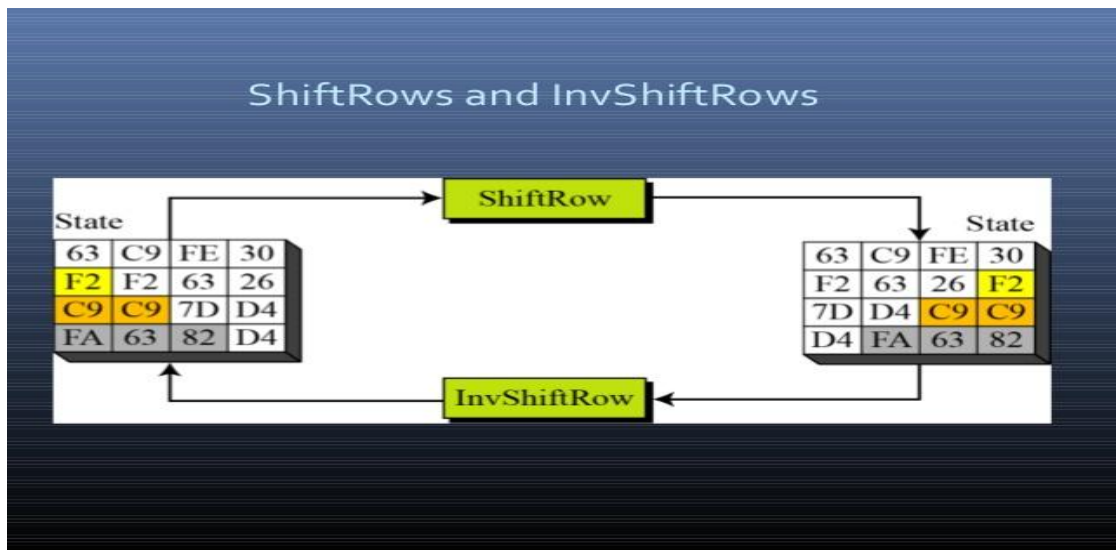
أما في مرحلة inverse sub byte فإننا نقوم بالاستبدال مع صندوق آخر S_Box_Dec له الشكل التالي:

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1x	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2x	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3x	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4x	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5x	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6x	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7x	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8x	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9x	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
ax	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
bx	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
cx	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
dx	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
ex	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
fx	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

الشكل (2-9) يوضح الصندوق S_Box_Dec

2.4.1.2 الإزاحة $shiftRows$:

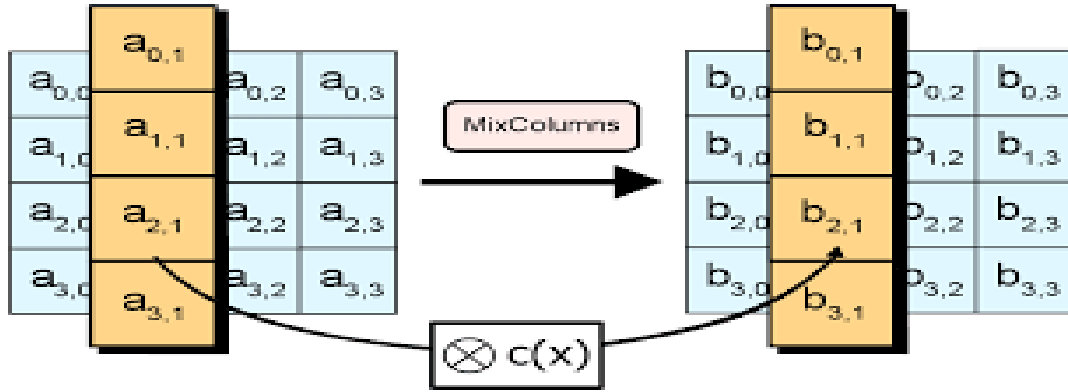
أي نقوم بتغيير أماكن البايت بناءً على رقم الصف الذي ينتمي إليه علماً أن ترتيب الصفوف يبدأ من الرقم 0 وعلى ذلك فإن الصف 0 لا نغير فيه أي شيء والصف الأول نغير مكان أول بايت من اليسار ليصبح مكانه أول بايت من اليمين ، والصف الثاني نكرر نفس الطريقة ل 2 بايت وأخيراً الصف الثالث نكرر نفس الطريقة ل 3 بايت ، أما في مرحلة $invShiftRows$ نقوم بتغيير مكان البايت من اليمين ليصبح في اليسار كما في الشكل التالي:



الشكل (2-10) يوضح عمليتي $shiftRows$ و $invShiftRows$

3.4.1.2 مرحلة المزج Mixcolumns:

توفر هذه العملية خاصية إخفاء العلاقة بين النص والنص المشفر ، حيث أنها تعمل على ضرب كل عمود من أعمدة المصفوفة مع مصفوفة ثابتة، وفي هذه المرحلة نكون قد تعمقنا في التشفير إلى مستوى البت، كما أنها أول عملية تأخذ بعين الاعتبار البايت المجاورة للبايت المراد تشفيره، كما في الصورة التالية:



الشكل (2-11) يوضح عملية mixcolumns

والمصفوفة التي نقوم بضرب الأعمدة بها في مرحلة التشفير هي من الشكل :

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

			d4				02	03	01	01		04
			bf				01	02	03	01		66
e0	b8	1e	5d				01	01	02	03		81
b4	41	27	30				03	01	01	02		e5
52	11	98										
ae	f1	e5										

الشكل (2-12) تبين عملية المزج

أما بالنسبة لعملية الجداء فتتلخص كما يلي:

في حال جداء أي عدد سداسي عشر مع 01 يبقى الناتج كما هو فمثلاً:

$$01 * 5D = 00000001 * 01011101 \\ = 01011101 = 5D$$

في حال جداء أي عدد سداسي عشر مع 02 نجد العلاقة:

$$X * 02 = \begin{cases} b6 \ b5 \ b4 \ b3 \ b2 \ b1 \ b0 \ 0 & \text{if } b7=0 \\ (b6 \ b5 \ b4 \ b3 \ b2 \ b1 \ b0 \ 0) \oplus 00011011 & \text{if } b7=1 \end{cases}$$

أي ننظر للبت الثامن من العدد السداسي عشر X أي ننظر إلى b7 فإذا كان مساوياً للصفر فإننا نقوم بإزاحة دورانية نحو اليسار بمقدار بت واحد فقط،

أما إذا كان البت الثامن مساوياً للواحد فإننا نقوم بإزاحة دورانية نحو اليسار وبعد ذلك نقوم بعملية XOR مع 1B (00011011).

فمثلاً:

$$57 * 02 = \underline{0}1010111 * 000000010 \\ = 1010111\underline{0} = AE$$

$$\begin{aligned}
 AE*02 &= \underline{1}0101110 * 00000010 \\
 &= 01011100 \oplus 00011011 \\
 &= 01000111 = 47
 \end{aligned}$$

في حال كان الجداء مع 03 :

$$\begin{aligned}
 X*03 &= (X*01) \oplus (X*02) \\
 &= X \oplus (X*02)
 \end{aligned}$$

وهذه هي عملية *Mixcolumns*، أما في مرحلة فك التشفير فأن الجداء يكون بالمصفوفة التالية:

$$\begin{pmatrix}
 0E & 0B & 0D & 09 \\
 09 & 0E & 0B & 0D \\
 0D & 09 & 0E & 0B \\
 0B & 0D & 09 & 0E
 \end{pmatrix}$$

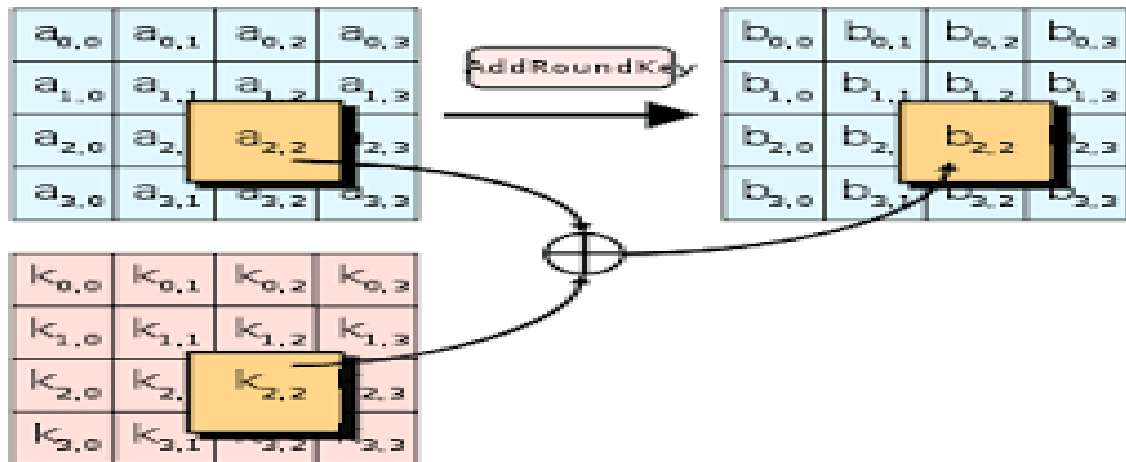
ويوجد طريقة خاصة لمعالجة عملية جداء أي عدد بأحد أعداد المصفوفة تعتمد على إيجاد صناديق

خاصة بكل من 09, 0D, 0E, 0B شبيهة بال *S_Box*.

4.4.1.2 إضافة المفتاح الخاص بالتكرار:

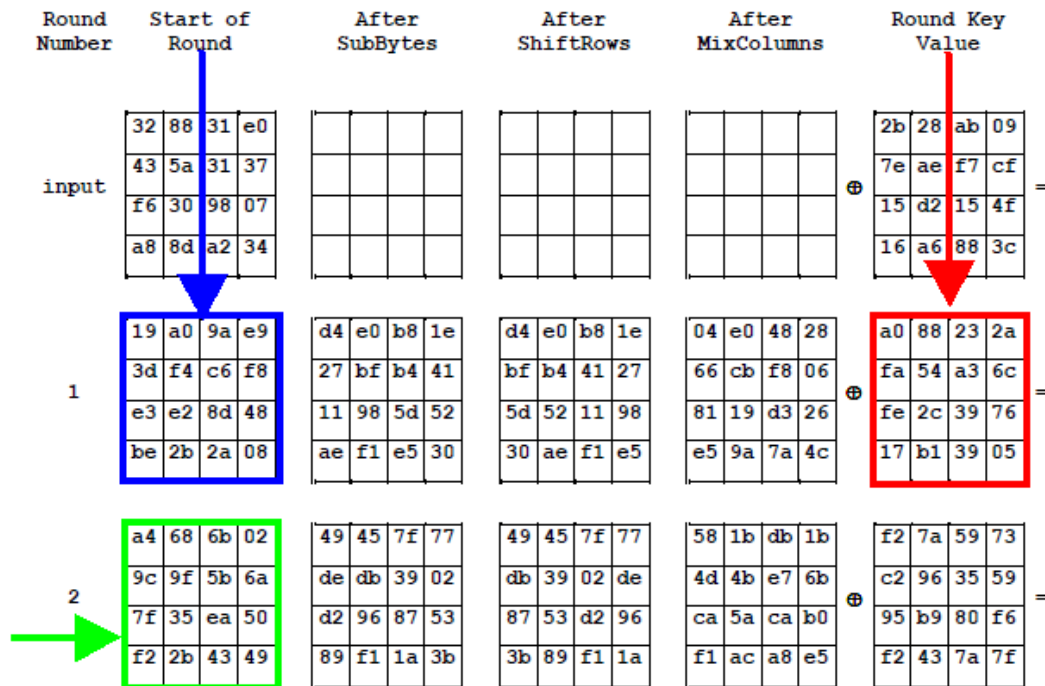
وهي أهم عملية في مقياس التشفير المتقدم *AES*، وتكمن أهمية عملية إضافة المفتاح الخاص بالدورة إلى أن جميع العمليات السابقة هي معروفة ومعلنة لدى الجميع ، فسرية التشفير هي من سرية المفتاح المستخدم .

تقوم هذه العملية بجمع مفتاح الدورة القادم من مولد المفاتيح مع المصفوفة، وعملية جمع المصفوفات هي جمع في مجال $(GF 2^8)$ أي نستخدم XOR ما بين مفتاح الدورة وبما أن الجمع والطرح في مجال $(GF 2^8)$ كلاهما يستخدم XOR فإن عملية إضافة مفتاح الدورة هي معكوسة لنفسها، أي تستخدم نفس العملية في فك التشفير.



الشكل (2-13) يوضح عملية إضافة المفتاح الفرعي

الشكل التالي يوضح خطوات خوارزمية AES على بلوك واحد من أجل تكرارين:



الشكل (2-14) يوضح آلية عمل خوارزمية AES

٢.٢ خوارزمية RSA :

خوارزمية الـ *RSA* هي واحدة من أوائل خوارزميات التشفير ذات المفتاح العام وشائعة الاستخدام كوسيلة لنقل الملفات بطريقة آمنة . حيث في أنظمة التشفير المماثلة، مفتاح التشفير يكون عام ويختلف عن مفتاح فك التشفير الذي يبقى سرياً . في هذه الخوارزمية عدم التناظر بين المفتاحين يعتمد على الصعوبة العملية في الحصول على معاملات الضرب لعددتين أوليين كبيرين .

RSA هي اختصار *Ron Rivest, Adi Shamir, Leonard Adleman* أسماء العلماء الذين قاموا بالتوصيف العام للخوارزمية في عام 1977 . *Clifford Cocks* الرياضي البريطاني قام بتطوير نظام مماثل في عام 1973 لكن لم يتم تصنيفه للعام 1997 .

مستخدم الخوارزمية ينشئ مفتاح عام ثم يقوم بنشره وذلك اعتماداً على عددتين أوليين كبيرين . الأعداد الأولية يجب أن تبقى سرية . أي شخص قادر على استخدام المفتاح العام لتشفير رسائله ، وتبعاً لكبر المفتاح العام فقط الشخص الذي يملك الأعداد الأولية يقدر على فك تشفير الرسالة .

الخوارزمية تصنف من الخوارزميات البطيئة ولذلك هي لا تستخدم لتشفير بيانات المستخدم المباشرة ، غالباً تستخدم لتشفير المفتاح المشترك للخوارزميات المتناظرة حيث تكون أكثر سرعة، وقد قمنا باستخدام الخوارزمية لتشفير مفتاح خوارزمية *AES* المتناظرة .

كما ذكرنا سابقاً الخوارزمية لها مفتاح عام هو الثنائية (e, n) الذي يستخدم لتشفير الرسالة والمفتاح الخاص الذي هو عبارة عن الثنائية (d, n) ويستخدم لفك التشفير .

1.2.2 خطوات خوارزمية RSA :

إن خطوات الخوارزمية تتلخص بأربع خطوات رئيسية :

- توليد المفاتيح (*Key Generation*) .

- نشر المفاتيح (*Key Distribution*) .

- التشفير (Encryption).

- فك التشفير (Decryption).

1.1.2.2 توليد المفاتيح (Key Generation) :

أولا نقوم باختيار رقمين أوليين مميزين p و q الأعداد يجب أن تكون متشابهة بالإشارة ومختلفة بعدد الخانات لجعل عملية اكتشافهما صعبة جدا ويمكن توليد قائمة أعداد أولية كبيرة وتخزينها ثم اختيار عددين بشكل عشوائي. نقوم بحساب (n) حيث $n = p * q$ ، نستخدم كقياس لكلا المفتاحين العام والخاص.

نقوم بحساب فاي $\lambda(n)$ حيث :

$\lambda(n) = \text{lcm}(\lambda(p), \lambda(q)) = \text{lcm}(p - 1, q - 1)$ حيث lcm هو المضاعف المشترك الأصغر هذه القيمة تحفظ سرية.

نقوم باختيار عدد e حيث $1 < e < \lambda(n)$ والقاسم المشترك الأكبر للعددين $\lambda(n)$ و e هو الواحد أي $\text{gcd}(e, \lambda(n)) = 1$.

حساب d التي هي تحقق $d \equiv e^{-1} \pmod{\lambda(n)}$ أي بحث عن قيمة تحقق $d * e \equiv 1 \pmod{\lambda(n)}$.

2.1.2.2 نشر المفاتيح (Key Distribution) :

لنفترض أنه لدينا شخص ما يريد إرسال رسالة لشخص آخر يجب على المرسل أن يعرف المفتاح العام للمستقبل والمستقبل يستخدم مفتاحه الخاص لفك تشفير الرسالة أي المستقبل بداية يرسل (e, n) المفتاح العام للمرسل ولكن أبدا d لا ينشر فهو يبقى المفتاح الخاص الذي لا يعرفه أحد سوى صاحبه.

3.1.2.2 التشفير (Encryption):

بعد تلقي المرسل المفتاح العام يمكنه الآن التشفير. لنفرض أن الرسالة قبل التشفير رمزها M يقوم المرسل بالحصول على الحرف الأول من الرسالة ولنرمز له m ويأخذ مقابله الرقمي وفقاً لجدول ASCII مثلاً حيث يكون $0 < m < n$ ثم يقوم بالتشفير وفقاً للعلاقة $c \equiv m^e \pmod{n}$ أي c هو تشفير m وهكذا نأخذ حرف تلو الآخر ونشفره حتى تنتهي الرسالة وتصبح جاهزة للإرسال ولنفتض رمز الرسالة المشفرة C .

4.1.2.2 فك التشفير (Decryption):

المستقبل يمكنه فك تشفير الرسالة والحصول على الرسالة الأصلية M باستخدام مفتاحه الخاص d وذلك ببساطة وفقاً للعلاقة $d \equiv m \pmod{n}$ حيث $c^d \equiv m \pmod{n}$ وهذا يعطي m وتجميع الحروف يعطي الرسالة الأصلية M .

مثال:

نختار $p=61, q=53$ وليكن

نحسب n حيث $n=61*53=3233$

نحسب فاي (n) حيث $\lambda(n)=p-1*q-1=60*52=780$

نختار e حيث $1 < e < 780$ والعددان e و فاي أوليان فيما بينهما ولتكن $e=17$

نحسب d التي توافق e بالقياس $\lambda(n)$ أي $d=413$

أي تحقق $\lambda(n)=1 \pmod{d}$ ونتحقق من ذلك $413*17 \pmod{780}=1$

أصبح لدينا المفتاح العام هو $(n=3233, e=17)$

نقوم بالتشفير الآن وفقاً للعلاقة $c(m)=m^{17} \pmod{3233}$

المفتاح الخاص هو $(n=3233, d=413)$ ونقوم بفك التشفير وفقاً للعلاقة

$$m(c)=c^{413} \bmod 3233$$

لنأخذ فرضاً $m=65$ عندها التشفير يكون:

$$c=65^{17} \bmod 3233 = 2790$$

لفك تشفير $c=2790$ نقوم بما يلي:

$$m=2790^{413} \bmod 3233 = 65$$

نلاحظ أن المشفر كان 65 وفك التشفير أعطى 65.

3.2 تقنية Node.JS :

NodeJs: هو نظام برامج مصمم لكتابة تطبيقات انترنت قابلة للتوسع كخوادم للوب، وتم اختياره بواسطة *infoworld* لجائزة تقنية العام في 2012 حيث أن *node.js* أنشئت على يد *Ryan Dahl* ابتداءً من 2009.

تكتب برامج *node.js* بلغة *javascript* والإدخال والإخراج فيها غير متزامن ذلك للحد من النفقات (زمن الانتظار وزمن التخدم) ولتحقيق أكبر قدر من قابلية التوسع.

فمثلاً:

ليكن لدينا سيرفر يقوم باستلام طلبات المستخدمين ويستجلب معلومات من قاعدة البيانات ويرسلها للزبائن فسير العملية بدون تقنية *Node.js* سيكون كما يلي :

عندما يقوم المستخدم الأول بإرسال طلب للسيرفر فإن السيرفر يقوم باستلام الطلب ومعالجته وبقية المستخدمين المرسلين لطلبات عليهم الانتظار في رتل الانتظار حتى يفرغ السيرفر من عمله أي أن السيرفر يقوم بتخدم طلب واحد في الزمن الواحد إذا لم يكن يعتمد على تقنية *multithreading* السيئة عتادياً وهذا الأمر يصبح معقداً جداً في حال وجود ملايين الطلبات كموقع فيسبوك مثلاً.

ومع تقنية *Node.js* يصبح سير العملية كما يلي:

يقوم السيرفر باستلام طلب مستخدم ما ولا ينتظر حتى يتم تقديمه بل في هذا الوقت يذهب لاستلام طلبات أخرى وبذلك لا يوجد وقت ضائع.

مميزات *node.js* :

- ١ - سرعة الأداء مع المحافظة على الفاعلية مع عدد أقل من العمليات غير الناجحة .
- ٢ - استغلال أفضل للموارد حيث أنك تستدعي المكتبات التي تعمل عليها في الوقت الذي تريده.
- ٣ - تعتمد على *single threading* وبالتالي لا تسبب إجهاداً للعتاد.

1.3.2 *io socket* :

ال *socket* هي الحل لمعظم أنظمة المحادثة في الزمن الحقيقي وتوفر قناة اتصال ثنائية الاتجاه بين الزبون و المخدم ، هذا يعني أن المخدم *server* يمكن أن يدفع الرسائل إلى الزبائن والزبون أيضاً يقوم بإرسال رسالة إلى المخدم ومن ثم يرسلها المخدم إلى للزبائن الأخرى

2.3.2 الخطوات الأساسية لإنشاء ملف *js* :

أولاً: نقوم بتنصيب *Node.js* في الحاسب حسب الخطوات التالية:

ثانياً: سوف نقوم بإنشاء لإطار عمل يسمى *express* لذلك يجب أولاً إنشاء ملف *package.json* يحوي المعلومات التالية:

```
{  
  "name" : "socket_chat_example",  
  "version" : "0.0.1",  
  "description" : "first socket.io app",  
  "dependencies" : { } }
```


حيث أننا نقوم بتحقيق هذه الخطوة من خلال الذهاب إلى ال *CMD* ونختار مسار معين مثلاً *C:\users\user\final* .

بعد ذلك نكتب التعليم *npm init* حيث أن الأداة *npm* تثبت بشكل تلقائي عند تنصيب *Node.js*، ثم نقوم بملء البيانات كما ذكرناها سابقاً



```
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

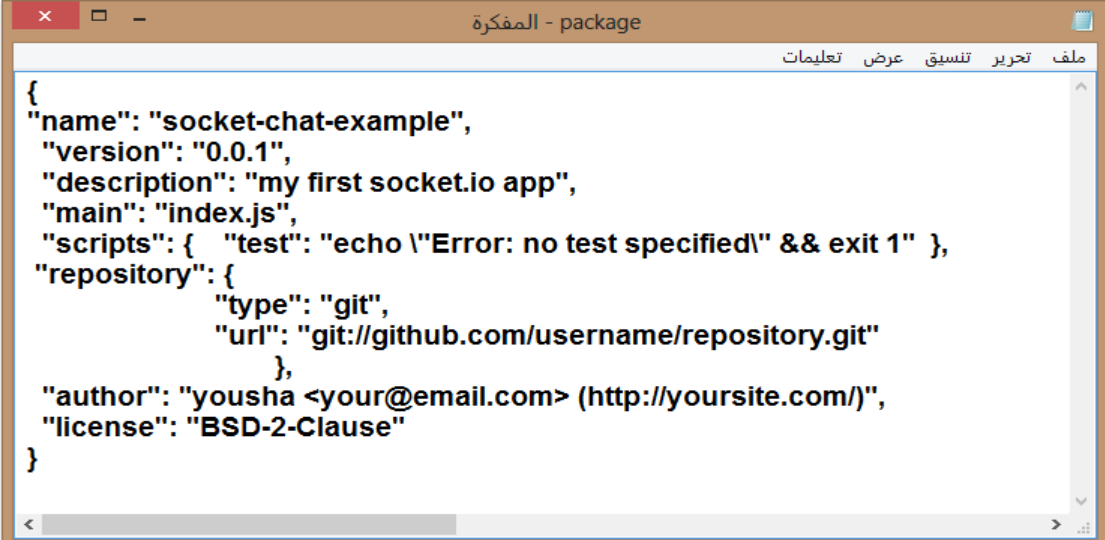
C:\Users\user>cd final
C:\Users\user\final>npm init
```



```
Press ^C at any time to quit.
name: <HM> socket_chat_example
version: <1.0.0> 0.0.1
description:
```

الشكل (2-15) يوضح عملية تهيئة *Package.json*

وعند الانتهاء نؤكد عملية التهيئة وذلك بكتابة *yes* في ال *cmd* فنلاحظ أنه تم إنشاء ملف اسم *package.json* في المسار الذي حددناه سابقاً ومحتوياته مبينة في الشكل التالي:



```
{
  "name": "socket-chat-example",
  "version": "0.0.1",
  "description": "my first socket.io app",
  "main": "index.js",
  "scripts": { "test": "echo \\\"Error: no test specified\\\" && exit 1" },
  "repository": {
    "type": "git",
    "url": "git://github.com/username/repository.git"
  },
  "author": "yousha <your@email.com> (http://yoursite.com/)",
  "license": "BSD-2-Clause"
}
```

الشكل (2-16) يوضح محتويات الملف *package.json*

ثالثاً: نقوم بإنشاء ملف نصي نسميه مثلاً *node.js* يحوي على التعليمات التالية

```

var app = require('express')();
var http = require('http').Server(app);
var io = require('socket.io')(http);

app.get('/',function(req,res){
    res.sendFile(__dirname+'/index.html');
})
io.on('connection',function(socket){
    console.log('one user connected '+socket.id);

    socket.on('message',function(data){
        var sockets =Object.keys( io.sockets.sockets);
        console.log(data);
        sockets.forEach(function(sock){
            if(sock.id != socket.id)
            {
                io.to(sock).emit('message',data)
            }
        })
    })

    socket.on('disconnect',function(){
        console.log('one user disconnected '+socket.id);
    })
})

http.listen(5000,function(){
    console.log('server listening on port 5000');
})

```

الشكل (2-17) يبين محتويات الملف *index.js*

```
io.on('connection',function(socket){
  console.log('one user connected '+socket.id);
```

هذا المقطع ينفذ عند اتصال مستخدم إلى السيرفر ،فبمجرد اتصال مستخدم فسيظهر على ال *cmd* الخاص بالسيرفر الرسالة *one user connected* متبوعة بال *id* الخاص بالسوكيت التي أنشأت اتصال بين المستخدم والسيرفر .

```
socket.on('message',function(data){
  var sockets =Object.keys( io.sockets.sockets);
  console.log(data);
  sockets.forEach(function(sock){
    if(sock.id != socket.id)
    {
      io.to(sock).emit('message',data)
    }
  }) })
```

هذا المقطع سيتفعل عند قيام مستخدم بإرسال رسالة (الرسالة هذه طبعاً ستكون مشفرة بخوارزمية *AES*) إلى السيرفر، فيقوم بطباعة هذه الرسالة على ال *cmd* الخاص بالسيرفر للدلالة على أنه قام باستقبالها ومن ثم يقوم بإرسالها إلى كل مستخدم متصل من خلال التعليمة *.emit*.

```
socket.on('disconnect',function(){
  console.log('one user disconnected '+socket.id);
})
```

هذا المقطع يتفعل عندما يقطع المستخدم الاتصال عن السيرفر عندها يظهر رسالة *one user disconnected* متبوعة بال *id* الخاص بالسوكيت التي أنشأت اتصال بين المستخدم والسيرفر .

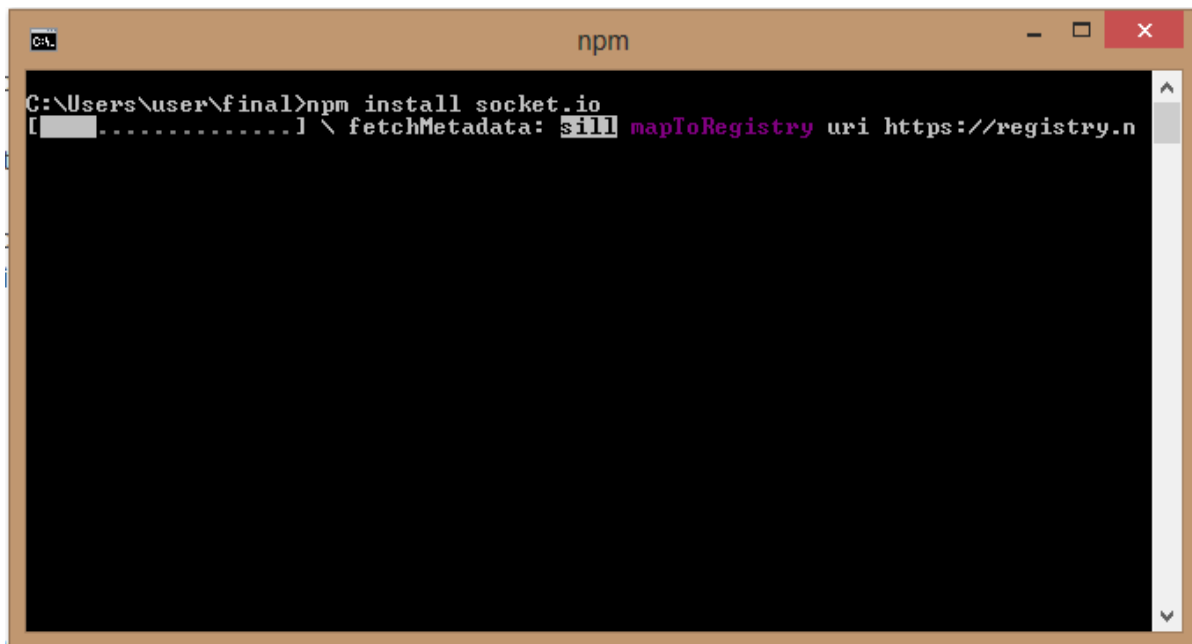
```
http.listen(5000,function(){
  console.log('server listening on port 5000');
})
```

هذا المقطع هو المسؤول عن قيام السيرفر بعملية *listening* (تنصت) على المنفذ (*port*)
الممرر كوسيط لهذا التابع وهنا اخترنا ال *port* رقم 5000 ،وبذلك تظهر رسالة على ال *cmd*
الخاص بالسيرفر *server is listening on port 5000*.

❖ *Socket.IO* مكونة من قسمين :

- ١ - *Socket.io* : هي من جانب السيرفر الذي يتكامل مع *Node.js http server*.
- ٢ - *Socket.io_client* : هي مكتبة خاصة بالزبون والتي هي من جانب الجهاز الذي يتصل بالسيرفر.

ومن أجل ذلك يجب تثبيت *socket.io* من خلال التعليمة
npm install socket.io



```
C:\Users\user\final>npm install socket.io
[.....] \ fetchMetadata: sill mapToRegistry uri https://registry.n
```

الشكل (2-18) يوضح تنصيب ال *socket.io*

والآن تبدأ عملية تثبيت ال *socket.io*

الفصل الثالث

الأكواد البرمجية للمشروع

في هذا الفصل سننتقل إلى ذكر التوابع الأساسية لخوارزميتي AES و RSA وعمل كل منها .

١.٣ الأكواد البرمجية لخوارزمية AES :

١.١.٣ التابع *computeWord* :

الذي يقوم بتوليد الكلمات التي تشكل المفاتيح الفرعية العشرة بدءاً من المفتاح

الرئيسي حيث أنه يحسب الكلمات من 43->4 :

```
//this function compute the ith word
static void computeWord(String [][]w,int i){
    String []arr=new String[4];
    String []wi1=new String [4];
    String []wi4=new String[4];
    for(int j=0;j<4;j++){
        wi1[j]=w[j][i-1];
        wi4[j]=w[j][i-4];
    }

    if(i%4==0){ //compute subword(Rotword(wi-1))XOR Rcon(i/4)
        String []t=new String[4];
        shiftTOLeft(wi1, 1);//rotWord
        wi1=subBytes(wi1); //subword
        Mathematical.XORBetween2Word(wi1, Rcn(i/4), t);
        Mathematical.XORBetween2Word(t, wi4, arr);
        for(int j=0;j<4;j++)
            w[j][i]=arr[j];
    }
    else
    { //compute Wi-1 XOR Wi-4
        Mathematical.XORBetween2Word(wi1, wi4, arr);
        for(int j=0;j<4;j++)
            w[j][i]=arr[j];
    }
}
```

الشكل (3-19) يوضح تابع *ComputeWord*

٢.١.٣ : التابع *Rcn*

الذي يقوم بحساب ثوابت التكرار المستخدمة من أجل حساب الكلمات.

```
//compute constant of replication Rcn(i/4)
static String [] Rcn(int i){
    String []res=new String [4];

    switch(i){
        case 1: res[0]="01" ;break;
        case 2: res[0]="02" ;break;
        case 3: res[0]="04" ;break;
        case 4: res[0]="08" ;break;
        case 5: res[0]="10" ;break;
        case 6: res[0]="20" ;break;
        case 7: res[0]="40" ;break;
        case 8: res[0]="80" ;break;
        case 9: res[0]="1B" ;break;
        case 10: res[0]="36" ;break;
    }

    //fill the next three cell with 00
    for(int j=1;j<4;j++){
        res[j]="00";
    }
    return res;
}
```

الشكل (3-20) يوضح التابع *Rcn*

٣.١.٣ : التابع *StringToKey*

الذي يقوم بتحويل مفتاح التشفير ب *16 byte* إلى مصفوفة 4×4 (تحتوي

الكلمات الأربعة الأولى *W0 W1 W2 W3*):


```

//this function convert Encryption key with length 16 byte to array 4*4
static void StringToKey1(String s,String [][]key){
    char [][]c=new char[4][4];

    if(s.length()==32){
        int index=0;
        for(int i=0;i<4;i++){
            for(int j=0;j<4;j++){
                key[j][i]=s.charAt(index)+" "+s.charAt(index+1);
                index+=2;
            }
        }
    }
    else
        if(s.length()>32){
            String str="";
            for(int i=0;i<32;i++){
                str+=s.charAt(i);
                StringToKey1(str, key);
            }
        }
}

```

الشكل (3-21) يوضح التابع StringToKey1

٤.١.٣ : التابع toState

يقوم بتحويل كتلة النص ذات طول 16 byte إلى مصفوفة 4 × 4 تسمى state .

```

//this function converts input String with 16 byte
// to array 4*4
static void toState(String s1,String [][]state){
    char [][]c=new char[4][4];
    String s=s1;
    int index=0;
    for(int i=0;i<4;i++){
        for(int j=0;j<4;j++){
            c[j][i]=s.charAt(index);
            index++;
        }
    }
    //convert every byte in the the array to its coordinate hexadecimal
    for(int i=0;i<4;i++){
        for(int j=0;j<4;j++){
            state[i][j]=Mathimatical.AsciiToHex((int)c[i][j]);
        }
    }
}

```

الشكل (3-22) يوضح التابع toState

٥.١.٣ : التابع *addRoundKey*

الذي يقوم بعملية XOR بين ال *state* والمفتاح الفرعي الذي رقمه *start*

```
//this function do the step add round key to state
static String[][] addRoundKey(String [][] stat,String [][]k,int start){
    String [][]temp=new String [4][4];
    temp=Mathimatical.XORBetween2Array(stat,k,start);//do XOR between state and the partial key his number =start
    return temp;
}
```

الشكل (3-23) يوضح التابع *addRoundKey*

٦.١.٣ : التابع *SubByte*

الذي يقوم بعملية استبدال كل عنصر من الكتلة بما يقابلها من الصندوق

:*S_Box*

```
//this function do subByte phase to array w
static void subByte(String [][]w){
    for(int i=0;i<4;i++)
        for(int j=0;j<4;j++)
        {
            int i0=Mathimatical.charToint(w[i][j].charAt(0));//obtain the first digit from the hexadecimal number w[i][j]
            int i1=Mathimatical.charToint(w[i][j].charAt(1));//obtain the second digit from the hexadecimal number w[i][j]
            w[i][j]=AES_Sub_Box.s_Box[i0][i1];//find the element in S_Box with row i0 and columns i1
        }
}
```

الشكل (3-24) يوضح التابع *subByte*

٧.١.٣ : التابع *shiftRows*

الذي يقوم بعملية إزاحة دورانية للكتلة نحو اليسار وذلك حسب دليل السطر:

```
//this function do shiftRows step to the array w due to index of row
static void shiftRows(String [][]w){
    String []ar=new String[4];
    for(int i=1;i<4;i++){
        for(int j=0;j<4;j++){
            ar[j]=w[i][j];
            keyExpansion .shiftTOLeft(ar, i); //shift the iTh row in ar i time to the left
        }
        for(int j=0;j<4;j++){
            w[i][j]=ar[j];
        }
    }
}
```

الشكل (3-25) يوضح التابع *shiftRows*

٨.١.٣ التابع *prod*:

الذي يقوم بعملية جداء رقم سداسي عشر مع 01 أو 02 أو 03، وهذا التابع

سيستدعى في المرحلة اللاحقة والتي هي *mixcolumns*

```
/* this function compute product any hexadecimal number with 01 ,01 or 03
 * where we can use this function in mixColumns phase in AES
 */
static String prod(String b ,String a){
    switch(b){
        case "01": return a;
        case "02": a=Mathimatical.hexToBinary(a);//convert hexadecimal number to binary
            if(a.charAt(0)=='0'){ //check if the 8th bit =0
                a=a.substring(1);
                a=a.concat("0");
                a=Mathimatical.binaryTohex(a);
            }
            else
            {
                a=a.substring(1);
                a=a.concat("0");

                a=Mathimatical.XOR(a,Mathimatical. hexToBinary("1B")); //compute XOR between a and 1B
                a=Mathimatical.binaryTohex(a);}
            return a;

        case "03": String d=Mathimatical.hexToBinary(prod("02",a));
            String e=Mathimatical.hexToBinary(a);
            return Mathimatical.binaryTohex(Mathimatical.XOR(d, e));
    }
}
return "a";
}
```

الشكل (3-26) يوضح التابع *prod*

٩.١.٣ التابع *mixcolumns* الذي يقوم بعملية المزج كما شرحناها سابقاً:

```
//this function do mixcolumns step
static String[][] MixColumns(String [][]b,String [][]a){
    String [][]res=new String [4][4];
    String temp;

    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            temp =prod(b[i][0], a[0][j]) ; //call the function prod to compute product b[i][0] with 01,02 or 03

            for (int k = 1; k < 4; k++)
            {
                String t=Mathimatical.hexToBinary(prod(b[i][k],a[k][j]));

                temp=Mathimatical.XOR(Mathimatical.hexToBinary(temp),t );
                temp=Mathimatical.binaryTohex(temp);
            }

            res[i][j] = temp;
        }
    }
    return res;
}
```

الشكل (3-27) يوضح التابع *MixColumns*

١٠.١.٣ التابع *do_AES*:

الذي يقوم بتطبيق خوارزمية *AES* فقط من أجل كتلة واحدة *16 byte* وهذا التابع سنستخدمه في التابع الرئيسي للخوارزمية وذلك لتطبيقه على كامل كتل النص.

```

//this function do AES to the message with 16 byte and the encryption key
static String do_AES(String message,String key){
    String [][]roundKey=new String[4][44];
    keyExpansion.StringToKey1(key,roundKey);

    for(int i=4;i<44;i++){
        keyExpansion.computWord(roundKey,i);
    }
    if(roundKey[0][0]!=null){
        String [][]states=new String [4][4];
        toState(message, states);//call function tostate
        states=addRoundKey(states, roundKey,0);//call function addroundKey

        for(int i=1;i<10;i++){
            subByte(states);    //subByte step
            shiftRows(states);  //shift step
            states=MixColoumns(mixCol,states); // Mixcolumns step
            states=addRoundKey(states, roundKey, i); // addRoundStep
        }
        subByte(states);
        shiftRows(states);
        states=addRoundKey(states, roundKey, 10);

    }

    return stateToArray(states);
}
else
    return "error";
}
}

```

الشكل (3-28) يوضح التابع `do_AES`

١١.١.٣ : التابع `AES_Encryption`

الذي يقوم بتطبيق خوارزمية `AES` على كامل النص الأصلي `plain text` مع ملاحظة أن التابع يقوم بإكمال آخر بلوك إلى `16 byte` في حال كان عدد محارفه أقل من `16 byte` حيث يقوم بحشوه بالفراغات.

```

// this function do AES to all Text
static String AES_Encryption(String message,String key){

    //fill the last block with ' ' to 16 byte if its 0<length<16
    int insertedBits=0;
    if(message.length()%16!=0){
        int x=message.length()/16;
        for(int i=message.length();i<(x+1)*16;i++){
            message+=' ';
            insertedBits++;}

    int block=1;
    String ciepher="";
    message+="$";
    String temp=message.charAt(0)+"";
    for(int i=1;i<message.length();i++){
        if(temp.length()%16==0){
            ciepher+=do_AES(temp, key);
            temp="";
            temp+=message.charAt(i);
            block++;
        }
        else
            temp+=message.charAt(i);
    }
    return ciepher;
}

```

الشكل (3-29) يوضح التابع *AES_Encryption*

١٢.١.٣ التابع *subByteInv*:

المعاكس لتابع *subByte* وهو يقوم باستبدال كل عنصر من الكتلة بما يقابلها من

الصندوق *S_Box_Dec*:

```
//this function compute inverse sub Byte
static void subByteInv(String [][]w){
    for(int i=0;i<4;i++){
        for(int j=0;j<4;j++){
            int i0=Mathimatical.charToint(w[i][j].charAt(0)); //get first digit form the hexadecimal number
            int i1=Mathimatical.charToint(w[i][j].charAt(1)); //get second digit form the hexadecimal number
            /*get hexadecimal number from S_Box_Dec
            which its row =i0 and column=i1
            */
            w[i][j]=AES_sub_Box_Decryption.s_Box_Decryption[i0][i1];
        }
    }
}
```

الشكل (3-30) يوضح التابع *subByteInv*

١٣.١.٣ :التابع *shiftRowsInv*

الذي يقوم بعملية إزاحة دورانية نحو اليمين للكتلة وذلك بإزاحة لكل سطر عدد من المرات يساوي دليل هذا السطر :

```
//compute inverse shift rows to array w
static void shiftRowsInv(String [][]w){
    String []ar=new String[4];
    for(int i=1;i<4;i++){
        for(int j=0;j<4;j++){
            ar[j]=w[i][j];
        }
        shiftTORight(ar, i);
        for(int j=0;j<4;j++){
            w[i][j]=ar[j];
        }
    }
}
```

الشكل (3-31) يوضح التابع *shiftRowsInv*

١٤.١.٣ : التابع *prodInv*

الذي قوم بإيجاد جداء أي رقم سداسي عشر ب 09 أو 0B أو 0D أو 0E وهذا

التابع سيستخدم في مرحلة *Invmixcolumn* :

```

//compute inverse product between hexadecimal number and 09,0B,0D or 0E
static String prodInv(String a,String b){
    char x=a.charAt(1);
    switch(x){
        case '9':return replaceValue(b,prod09);
        case 'B':return replaceValue(b,prod0B);
        case 'D':return replaceValue(b,prod0D);
        case 'E':return replaceValue(b,prod0E);
    }

    return "ERROR REPLACE";
}

//replace w with its equalivat in table
static String replaceValue(String w,String [][]table){
    int i0=Mathimatical.charToint(w.charAt(0));
    int i1=Mathimatical.charToint(w.charAt(1));
    return table[i0][i1];
}

```

الشكل (3-32) يوضح التابع *prodInv*

١٥.١.٣ التابع *MixColumnsInv*:

الذي يقوم بالعملية العكسية لعملية *MixColumns*.

```

//compute mix columns inverse
static String[][] MixColoumnsInv(String [][]b,String [][]a){
    String [][]res=new String [4][4];
    String temp;

    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            temp =prodInv(b[i][0], a[0][j]) ;//cal prodInv function

            for (int k = 1; k < 4; k++)
            {
                String t=Mathimatical.hexToBinary(prodInv(b[i][k],a[k][j]));

                temp=Mathimatical.XOR(Mathimatical.hexToBinary(temp),t );
                temp=Mathimatical.binaryTohex(temp);
            }

            res[i][j] = temp;
        }
    }
    return res;
}

```

الشكل (3-33) يوضح التابع *MixColumnsInv*

١٦.١.٣ التابع *do_Aes_Decryption*

الذي يقوم بتنفيذ خوارزمية فك التشفير على بلوك 16 byte .

```

/*this function do AES Decryption to message with 16 hexadecimal number
and keyE is Key of Decryption
*/
static String do_AES_Decryption(String message,String keyE){
    String key=keyE;
    String [][]states=new String [4][4];
    String [][]roundKey=new String[4][44];
    keyExpansion.StringToKey1(key,roundKey);//convert key to array 4*4

    if(roundKey[0][0]!=null){
        for(int i=4;i<44;i++){
            keyExpansion.computeWord(roundKey,i); //compute partial keys by computing words 4-> 43
            toStateAr(message,states);
            //add round key [10] to state
            states=JavaApplication18.addRoundKey(states, roundKey, 10);
            for(int i=9;i>=1;i--){//repeat 9 times
                shiftRowsInv(states);
                subByteInv(states);
                states=JavaApplication18.addRoundKey(states, roundKey, i);
                states=MixColoumnsInv(mixColDec,states);
            }
            shiftRowsInv(states);
            subByteInv(states);
            states=JavaApplication18.addRoundKey(states, roundKey, 0);//add round key[0]
        }
        String [][]res=new String[4][4];
        for(int i=0;i<4;i++){
            for(int j=0;j<4;j++){
                states[i][j]=(char)Mathimatical.hexToAscii(states[i][j])+"";//convert final block to Ascii
            }
        }
        return stateToString(states);}

```

الشكل (3-34) يوضح التابع *do_Aes_Decryption*

١٧.١.٣ التابع *AES_DecryptionS*

الذي يقوم بتطبيق خوارزمية فك التشفير على كامل الرص المشفر.

```
//do AES Decryption for all plain text
static String AES_DecryptionS(String message,String key){
String plain="";
message+='$';
String temp=message.charAt(0)+"";
for(int i=1;i<message.length();i++)
    if(temp.length()%32==0){
        plain+=do_AES_Decryption(temp, key);
        temp="";
        temp+=message.charAt(i);
    }
else
    temp+=message.charAt(i);

return plain;
}
}
```

الشكل (3-35) يوضح التابع *AES_DecryptionS*

١٨.١.٣ التابع *stateToArray*:

الذي يقوم بتحويل البلوك (4×4) الناتج من عملية فك التشفير إلى سلسلة التي تمثل النص الأصلي.

```
//convert array 4*4 to String with 16 byte
static String stateToArray(String [][]state){
String res="";
for(int i=0;i<4;i++)
    for(int j=0;j<4;j++)
        res+=state[j][i];

return res;
}
```

الشكل (3-36) يوضح التابع *stateToArray*

٢.٣ التتابع الخاصة بخوارزمية RSA :

١.٢.٣ التتابع exp_mod :

الذي يقوم بعملية حساب $(a^e \bmod n)$ وذلك اعتماداً على طريقة بسيطة هي حساب الباقي لكل مرحلة من مراحل حساب القوة.

```
/*method to compute  $a^k \bmod n$  with a simple repetition
way and use mod every repetition for reduce memory needed for it more and more
*/

public static long exp_mod(long a, long k, long n)
{
    long d=1;
    long aa=a;

    while(k>0)
    {
        if(k%2==1)
        {
            d=(d*aa)%n;
        }
        k=(k-(k%2))/2;
        aa=(aa*aa)%n;
    }

    return d;
}
```

الشكل (3-37) يوضح التتابع exp_mod

لأهمية هذا التتابع سيتم توضيحه بمثال:

ليكن لدينا الأرقام $(3,3,5)$ هي الوسطاء الممرة لهذا التتابع أي استدعاؤه هكذا

$exp_mod(a=3, k=3, n=5)$ فإن تنفيذه سيكون على الشكل التالي:

سندخل في حلقة *while* شرط التوقف فيها أن يكون $k \leq 0$ ولدينا الجدول يوضح
تغير قيم المتحولات خلال تنفيذ التابع والنتيجة التي يعطيها.

التكرارات	$k=3$	$aa=3$	$d=1$
التكرار الأول	1	4	3
التكرار الثاني	0	1	2

الجدول (3-4) يوضح قيم المتحولات أثناء التنفيذ

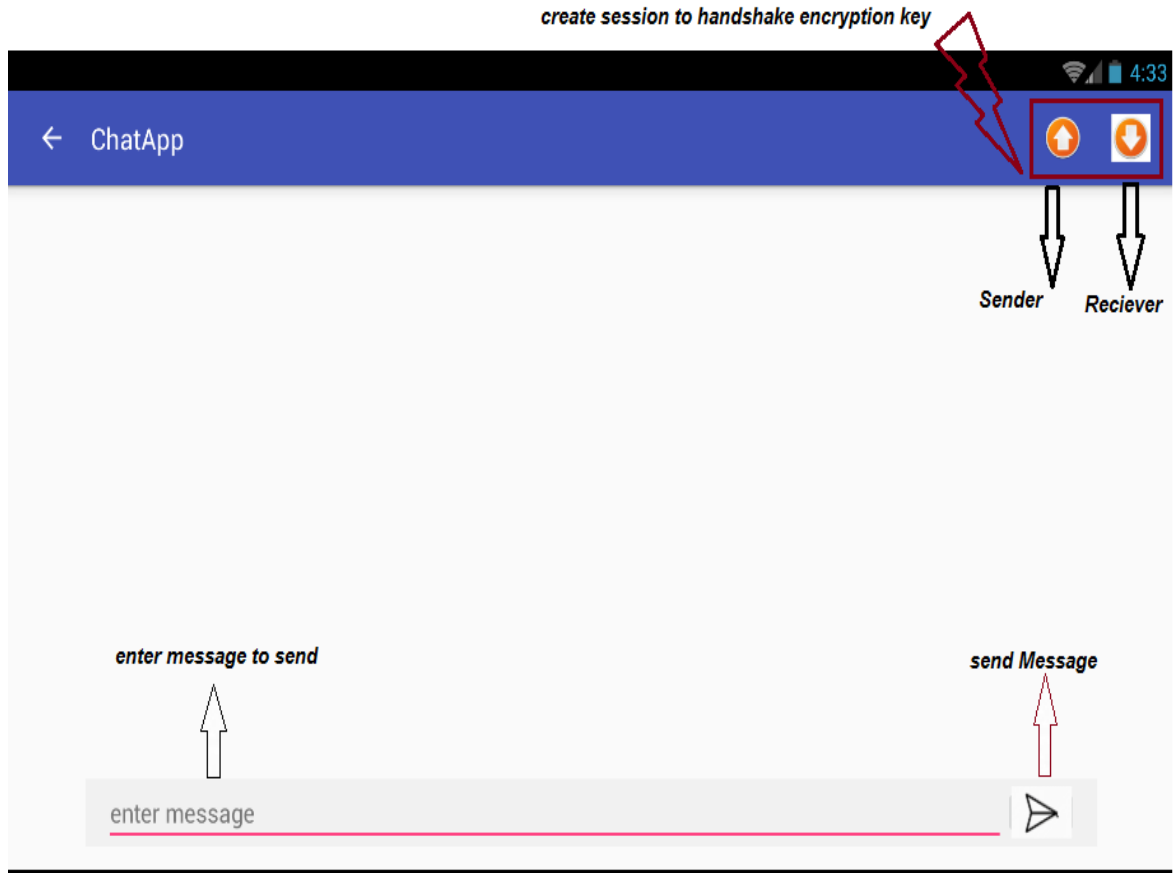
التابع يقوم بإرجاع قيمة d والتي لدينا تساوي 2 وهذا صحيح لأن
 $3^3 \bmod 5 = 27 \bmod 5 = 2$ أهمية هذا التابع تكمن في قدرته على حساب قوى
كبيرة دون أخطاء متعلقة بالذاكرة المستخدمة لتخزين الناتج الكبير نسبياً.

الفصل الرابع

التطبيق العملي

في هذا الفصل سنتطرق إلى كيفية عمل تطبيق المحادثة الذي يستخدم تقنية *Node.js* بالإضافة إلى توضيح آلية عمل خوارزمية *AES* من خلال تشفير ملفات *txt* و *docx* بالإضافة إلى تشفير صورة.

١.٤ الواجهة الأساسية للتطبيق عند الأجهزة الطرفية:



الشكل (4-38) يوضح الواجهة الأساسية للتطبيق عند الأجهزة الطرفية

ولمعرفة آلية عمل التطبيق نتبع الخطوات الخطوات التالية:

١ - في ال `cmd` نكتب التعليمة `node index.js` فتظهر لدينا النافذة التالية:

```
node index.js - موجه الأوامر
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.
C:\Users\user>cd final
C:\Users\user\final>node index.js
server listening on port 5000
```

الشكل (4-39) يوضح قيام السيرفر بعملية `listening`

وهنا يبدأ السيرفر بعملية التنصت *listening* وينتظر اتصال المستخدمين .

٢ - تشغيل التطبيق في الجهاز الأول فنلاحظ في ال *cmd* مايلي:



```
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Users\user>cd final
C:\Users\user\final>node index
server listening on port 5000
one user connected 6S0ZXJdzzRff8JOQAAAA
```

الشكل (4-40) يوضح اتصال أول مستخدم بالسيرفر

٣ - تشغيل التطبيق في الجهاز الثاني فنلاحظ أيضاً في ال *cmd* مايلي:

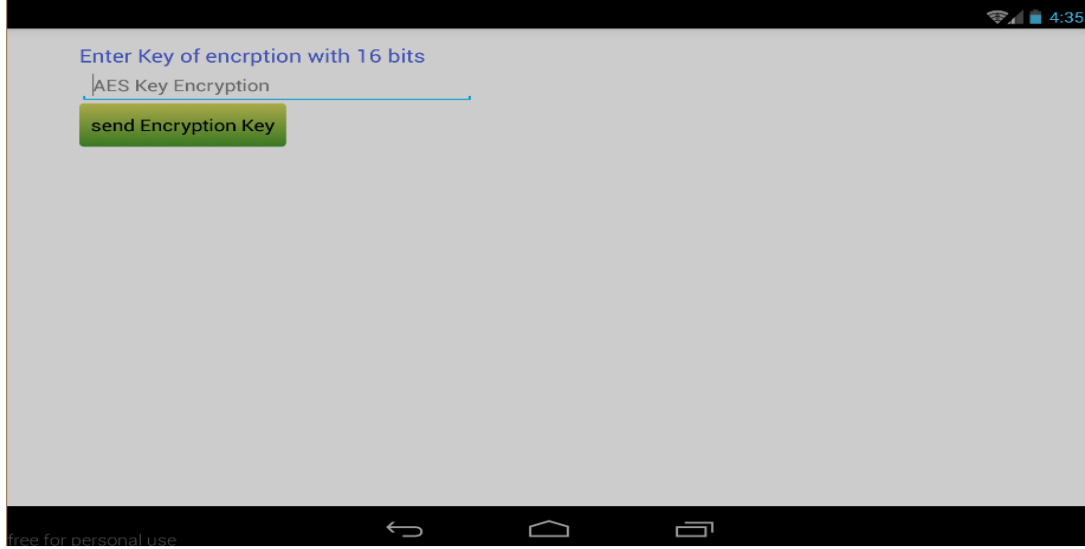


```
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

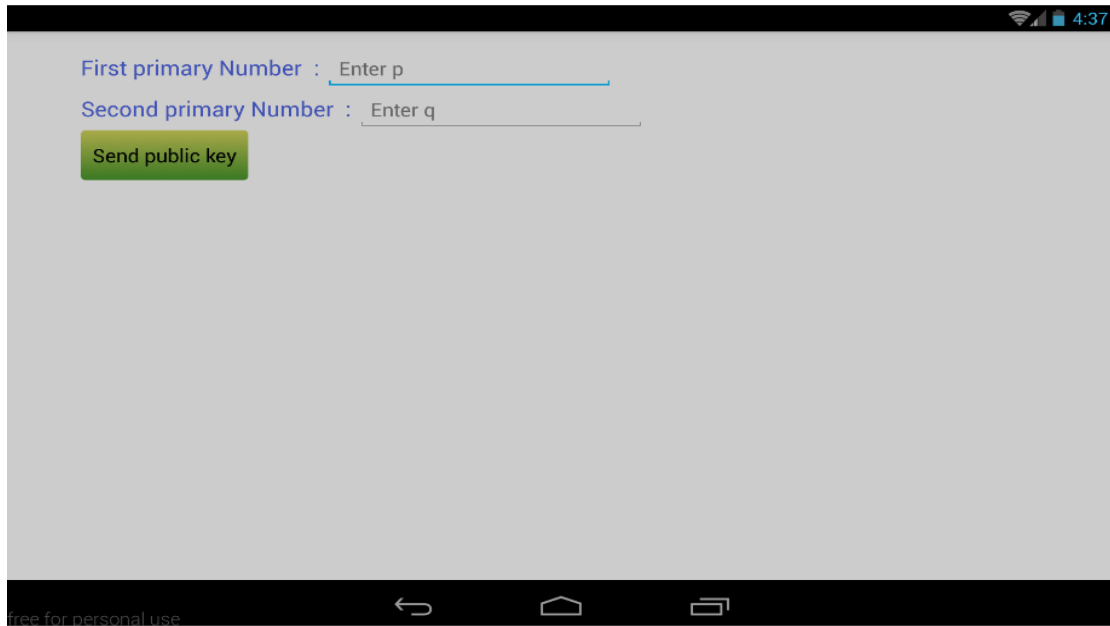
C:\Users\user>cd final
C:\Users\user\final>node index
server listening on port 5000
one user connected 6S0ZXJdzzRff8JOQAAAA
one user connected rrRyeeahv8XoXJn2AAAA
```

الشكل (4-41) يبين اتصال المستخدم الثاني بالسيرفر

- ٤ - قبل البدء بالمحادثة يجب على كلا الجهازين إنشاء جلسة لتحديد مفتاح التشفير الذي سيستخدم لتشفير الرسائل المتبادلة بينهما حيث أن الجهاز الأول يعمل كمرسل لمفتاح التشفير والثاني يعمل كمستقبل للمفتاح:

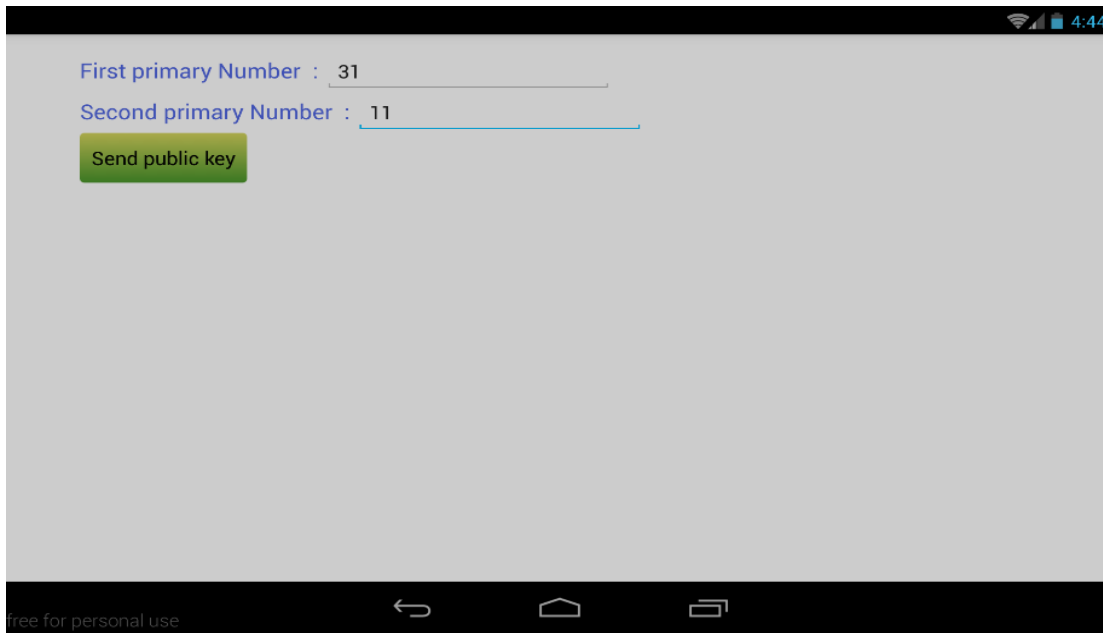


الشكل (4-42) يبين واجهة المرسل



الشكل (4-43) يبين واجهة المستقبل

٥ - يجب على المستقبل تحديد قيمتي p و q اللذان سيستخدمان في توليد المفتاح العام $\{e, n\}$ لخوارزمية RSA .



الشكل (4-44) يوضح تحديد p و q

وعند الضغط على الزر *send public key* يتم توليد المفتاح العام $\{e, n\}$ والذي سيرسل إلى السيفرر والسيفرر بدوره يرسله إلى الجهاز الأول (المرسل)

```
node index
server listening on port 5000
one user connected jeygyIwI-eB6gkDBAAAA
one user connected e7r_Rz3GozqFKHVOAAAB
{ text: '7 341' }
```

server recieved e and n

First primary Number : Enter p

Second primary Number : Enter q

Send public key

e= 7 , n=341

Enter Key of encrption with 16 bits

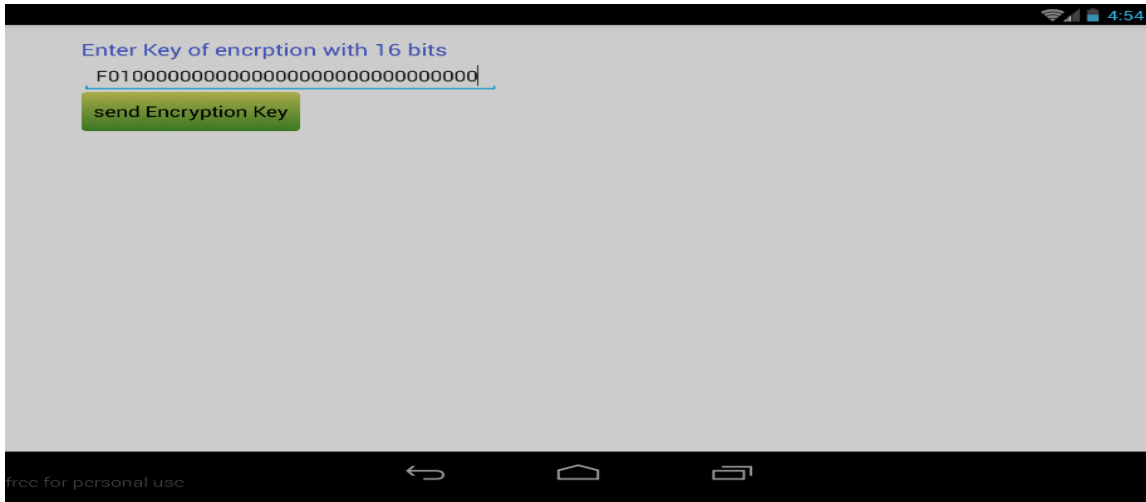
AES Key Encryption

send Encryption Key

e= 7 , n=341

الشكل (4-45) يوضح تبادل المفتاح العام e & n

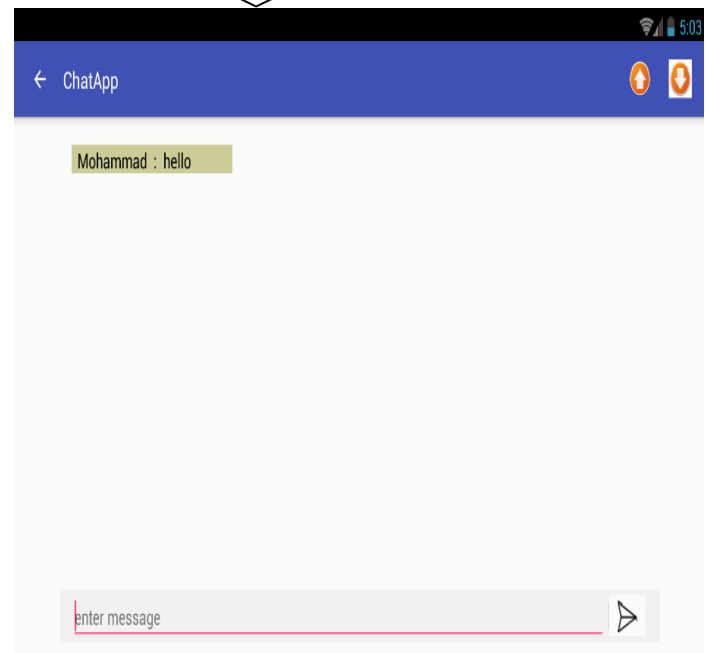
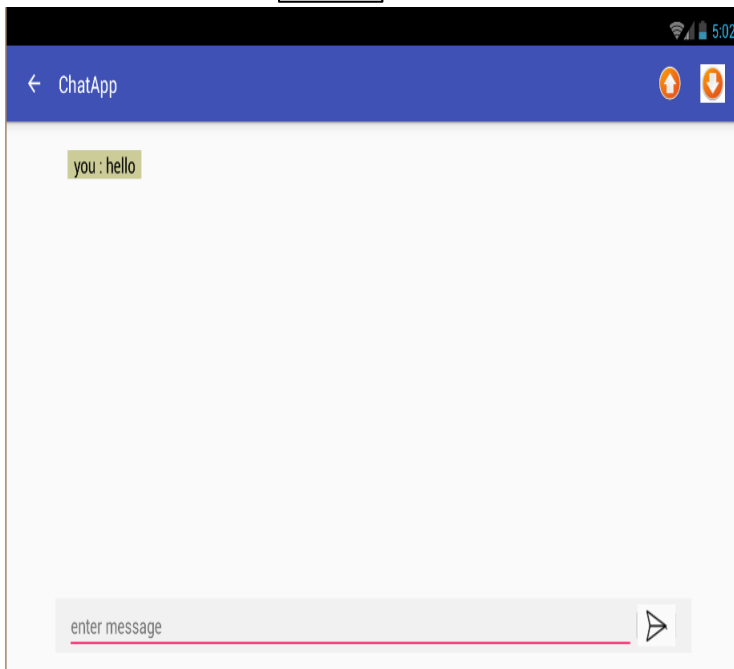
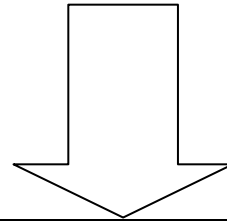
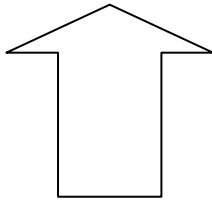
٦ - الآن على الجهاز الأول (المرسل) تحديد مفتاح التشفير كما يلي :



الشكل (4-46) يبين تحديد مفتاح التشفير

وعند الضغط على الزر *send Encryption key* يتم تشفيره بخوارزمية *RSA* ومن ثم إرساله إلى السيرفر والذي بدوره يرسله إلى الجهاز الثاني (المستقبل) ومن ثم المستقبل يستخدم مفتاحه الخاص $\{d, n\}$ لفك تشفير المفتاح وبالتالي يصبح مفتاح التشفير موجود عند كل من المرسل والمستقبل أما عند السيرفر فيبقى مشفراً.

٧ - الآن تبدأ عملية التراسل الآمن بين الجهازين وسنوضح ذلك من خلال قيام الجهاز الأول (المرسل) بإرسال كلمة *hello* إلى الجهاز الثاني (المستقبل).

[illegible]

الشكل (4-48) يوضح آلية إرسال كلمة *hello*

في الشكل السابق نجد مايلي:

- عندما يريد المرسل إرسال كلمة *hello* فبدائيةً يتم تشفير الكلمة بخوارزمية *AES* فيكون تشفير هذه الكلمة هي السلسلة *39 FB C2 C1 7B 29 47 63 CF CC E2 BE*
.FF 06 A2 30

- بعد تشفير الكلمة يتم تحويل كل رقم سداسي عشر إلى مقابله بال *ASCII* ومن ثم يتم إرسالها إلى السيفر فمثلاً

$$(39)_{16} = (57)_{10} = (9)_{ASCII}$$

$$(FB)_{16} = (251)_{10} = (\hat{u})_{ASCII}$$

$$(C2)_{16} = (194)_{10} = (\hat{A})_{ASCII}$$

وهكذا

وفي النهاية ينتج لدينا السلسلة *0 9ûÂÁ{)Gcİİâ¾4ÿ 0*

- بعد تلقي السيفر للرسالة المشفرة فإنه يقوم بإرسالها إلى الجهاز الثاني دون القيام بأي تعديل عليها أو أي فهم لمحتواها .

- يقوم الطرف الثاني باستقبال الرسالة المشفرة ومن ثم يقوم بتحويل كل رمز إلى الرقم السداسي عشر المقابل له ومن ثم يقوم بفك تشفيرها باستخدام خوارزمية *AES* وبالتالي يحصل على كلمة *.hello*

❖ وبذلك يستطيع الطرفان تبادل الرسائل بسرية شبه تامة بينهما، و حتى لو تمكن طرف ثالث من الوصول إلى الرسائل المتناقلة فإنه لا يستطيع فك تشفيرها ومعرفة فحواها لعدم امتلاكه مفتاح فك التشفير. أما الآن فسننتقل إلى الجزء الآخر من المشروع وهو التطبيق العملي لخوارزمية *. AES*

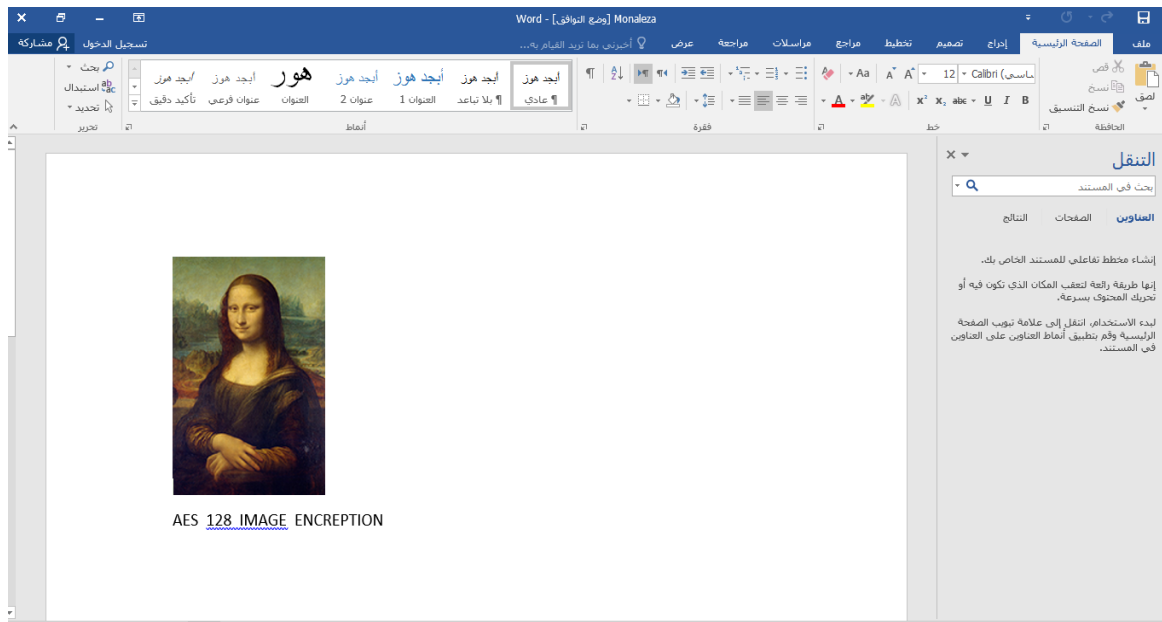
٢.٤ الواجهة الخاصة بخوارزمية AES :

The screenshot shows a web application titled "AES_128 Algorithm" in red. It features several input fields and buttons. At the top left, there is a text input field labeled "Enter Path Of File" with a file selection icon. Below it is another text input field labeled "Enter Key Of Encryption". To the right of these fields is a large empty box labeled "Algorithm Rounds for each block". Below the key input field are two buttons: "Encrypt" and "Decrypt". Further down, there is a text input field labeled "Enter Round Number" next to a small square icon, and a button labeled "Get Round Picture". At the bottom, there is a large empty box labeled "Key Generation :" in green, and a button labeled "show generated key".

الشكل (4-49) واجهة توضيح عمل خوارزمية AES

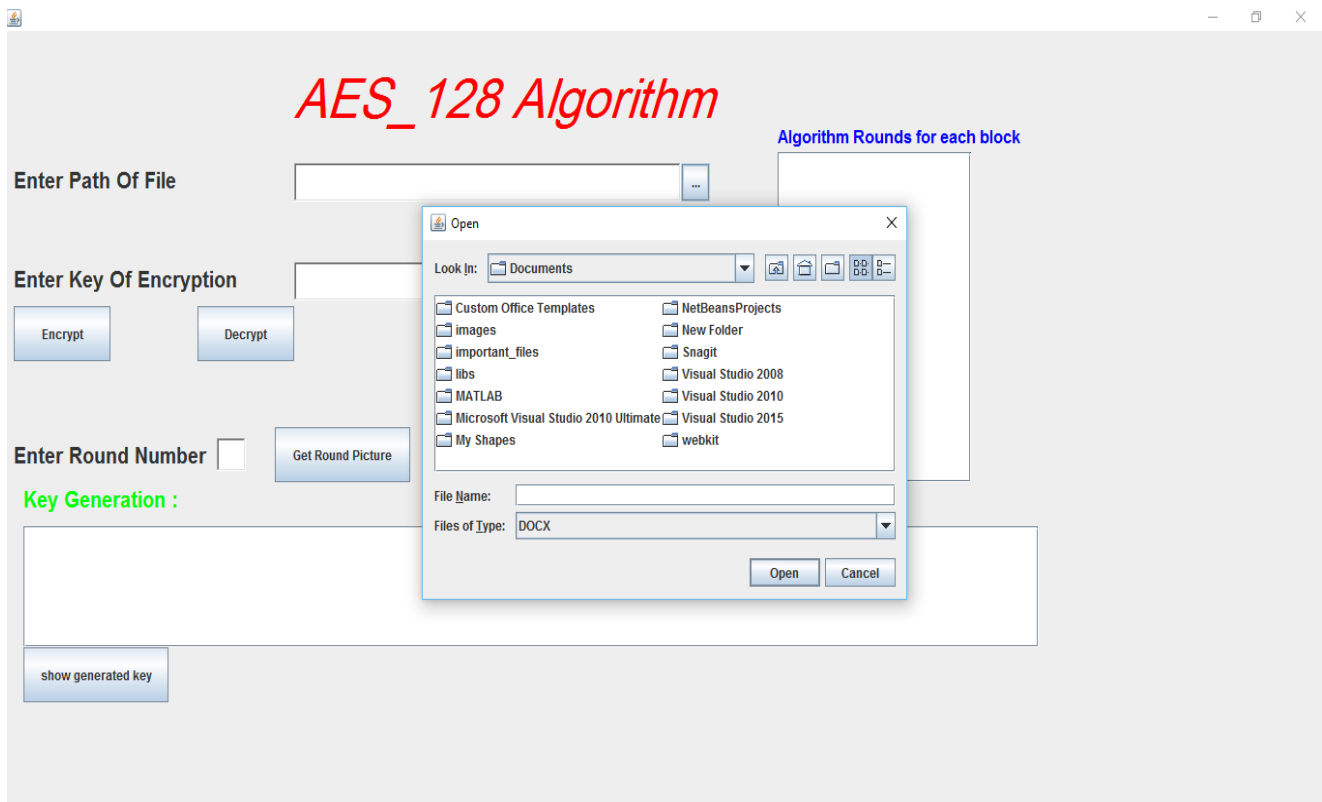
1.2.4 تشفير ملف word بصيغة docx يحوي صورة ونص :

أولاً: ليكن لدينا ملف word اسمه monaliza.docx محتوياته موضحة بالشكل



الشكل (4-50) يوضح ملف بصيغة docx

ثانياً: لتشفير هذا الملف أولاً نقوم بإدخال مسار هذا الملف في المكان المخصص في الواجهة



الشكل (4-51) يوضح عملية اختيار ملف docx

الملف

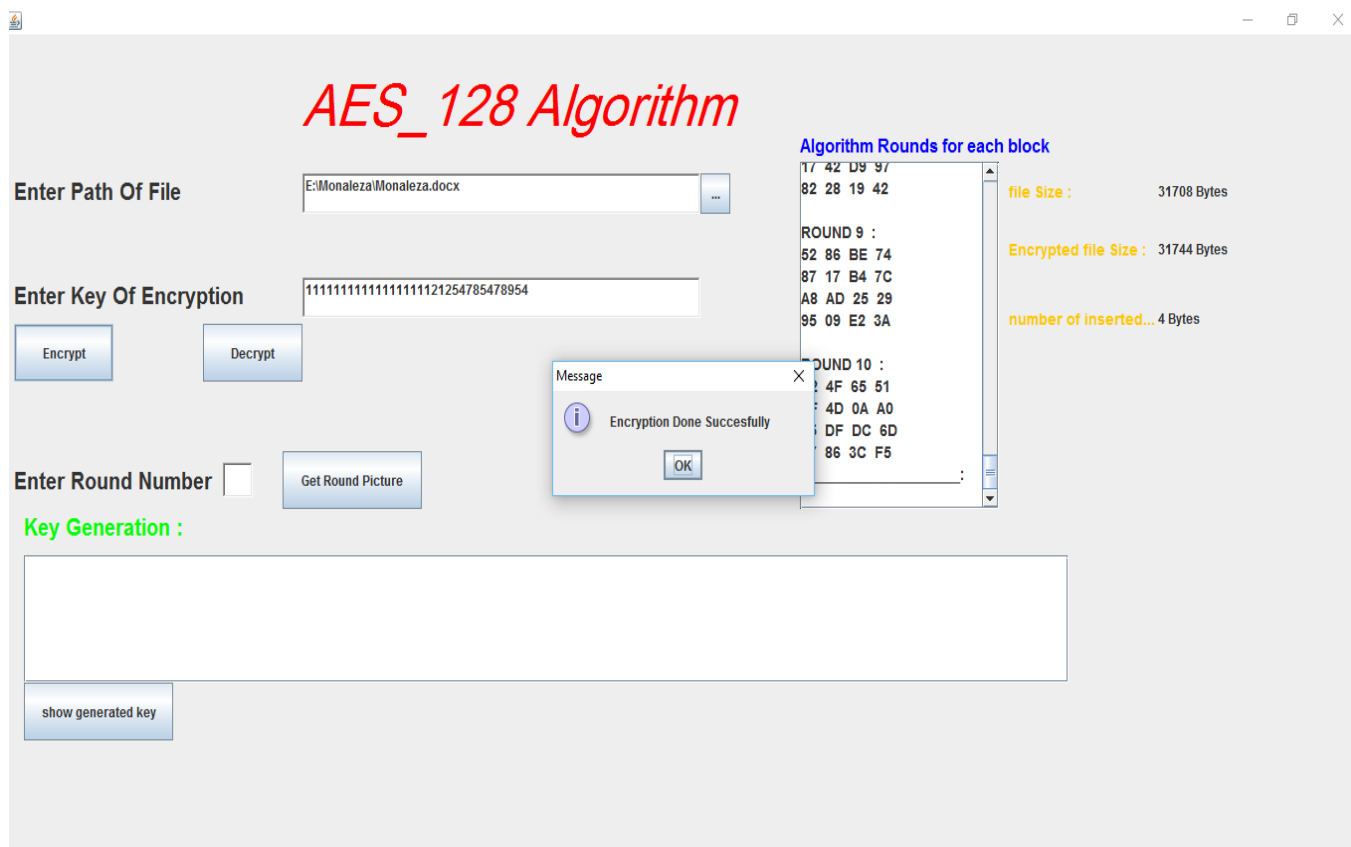
AES_128 Algorithm

Algorithm Rounds for each block

[illegible]

الشكل (4-53) يبين إدخال مفتاح التشفير لخوارزمية AES

رابعاً: بعد تحديد الملف وإدخال مفتاح التشفير بقي علينا الضغط على زر *Encrypt* لبدء عملية التشفير وبعد أن تتم العملية يظهر ما يلي :



الشكل (4-54) يوضح إتمام عملية التشفير

في الجانب الأيمن للواجهة نلاحظ ظهور ثلاثة عبارات

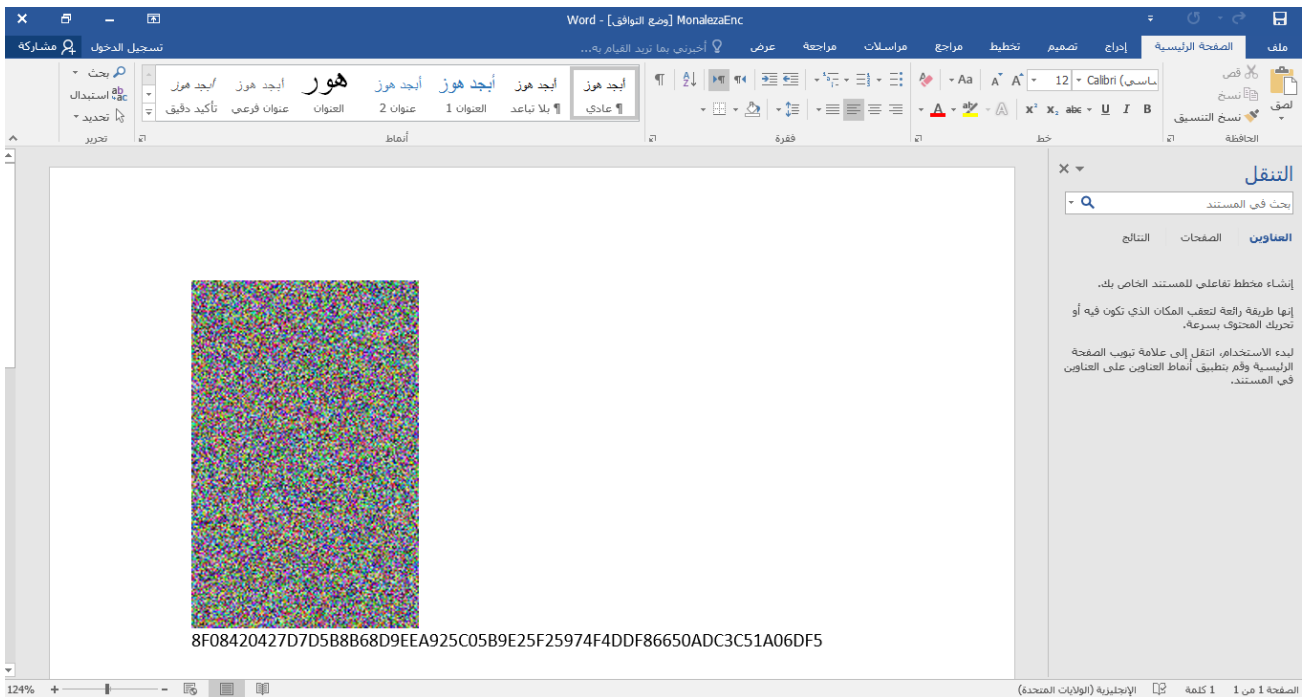
العبرة الأولى تظهر لنا حجم الملف الأصلي أي قبل عملية التشفير وهنا لدينا حجم الملف قبل التشفير 31708 byte .

العبرة الثانية تظهر لنا حجم الملف المشفّر وهو 31744 byte .

العبرة الثالثة تبين لنا مقدار الحشو الحاصل على آخر كتلة (أي عدد البايتات اللازم إضافته لآخر $block$ حتى يصبح حجمها 16 byte) فمثلاً في الملف السابق الكتلة الأخير طولها 12 byte وبالتالي يجب إضافة 4 byte لبدء تشفيرها مع الملاحظة أن هذه البايتات هي محرف الفراغ ($space$).

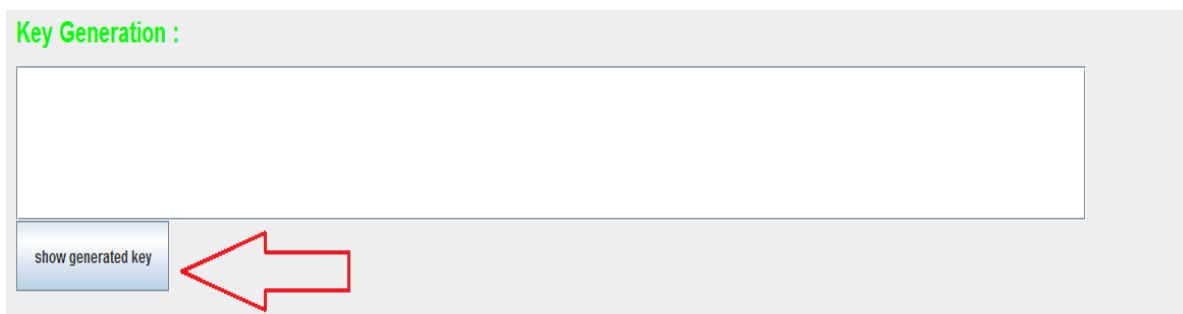
أما في المربع على يسار هذه العبارات الثلاثة فقد وضع فيه قيمة كل *block* عند كل خطوة من الخوارزمية أي وضعنا قيمة ال *block* بعد مرحلة *add round key* و *shift rows* و *sub byte* و *mix columns* كل على حدى من أجل كل مرحلة للخوارزمية.

خامساً : نفتح الملف المشفر فنشاهد محتوياته كما يلي



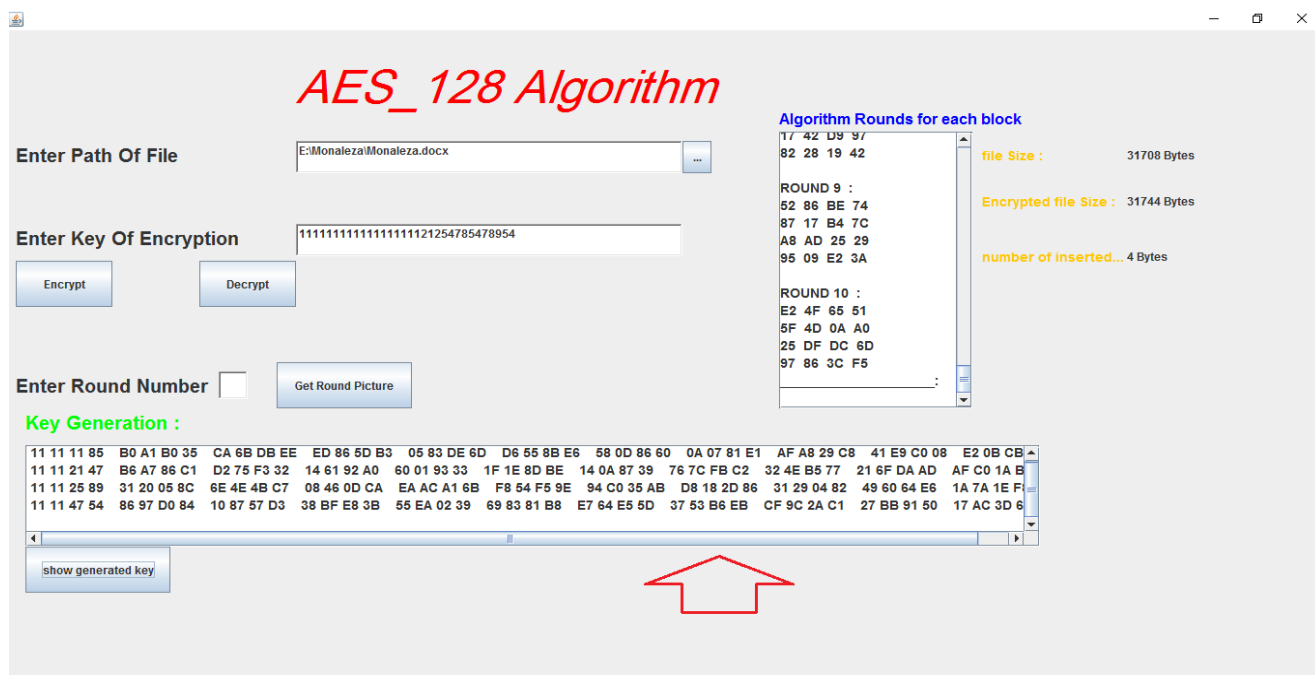
الشكل (4-55) يوضح محتويات الملف المشفر

- لرؤية المفاتيح المستخدمة في الخوارزمية فليس علينا إلا أن نضغط على الزر *show generated key*



الشكل (4-56) يبين الزر الخاص بعرض المفاتيح الفرعية

وعند الضغط عليه يظهر ما يلي :



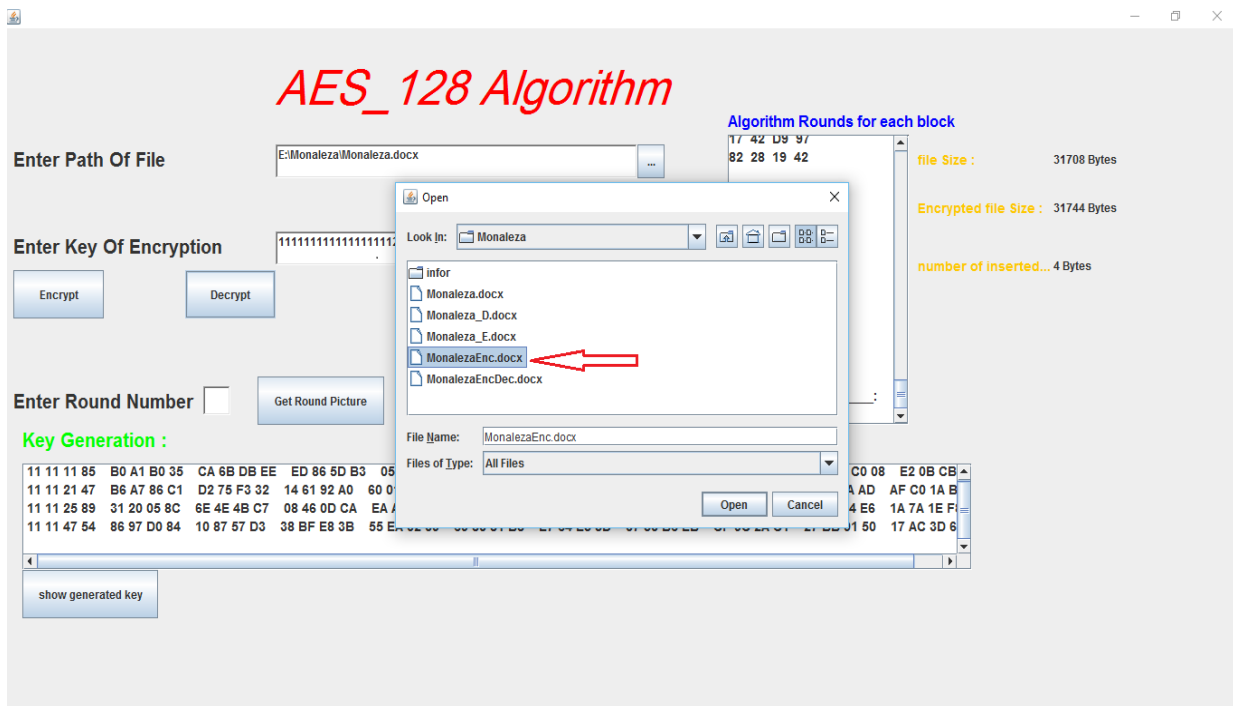
الشكل (4-57) يوضح عرض المفاتيح الفرعية

كما نرى فقد ظهر لدينا في هذا المكان المفاتيح الفرعية المستخدمة في مراحل الخوارزمية حيث أن المفتاح الأول هو المفتاح الأساسي الذي قمنا بإدخاله والمفاتيح العشرة الباقية قمنا بتوليدها اعتماداً على خوارزمية توليد المفاتيح الفرعية.

2.2.4 فك تشفير ملف *word* بصيغة *docx* :

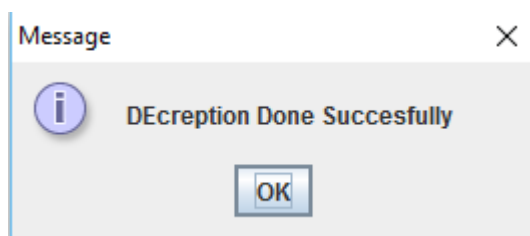
سنطبق خوارزمية فك التشفير على نفس الملف الذي قمنا بتشفيره في الخطوة السابقة:

أولاً: نقوم باختيار الملف *MonalizaEnc.docx*



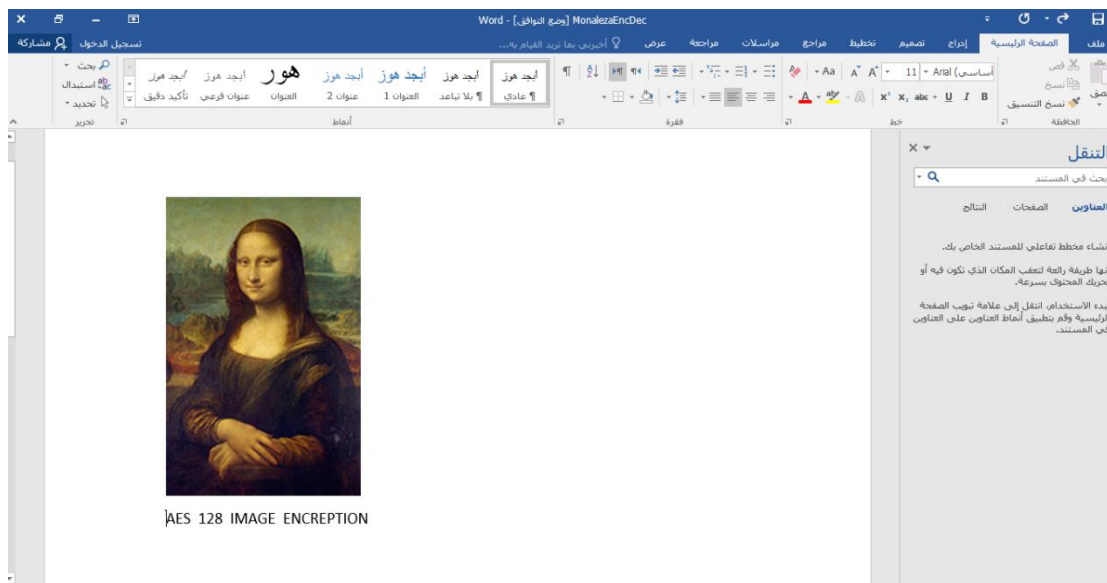
الشكل (4-58) يوضح آلية اختيار الملف لفك تشفيره

ثانياً: نضغط زر *decrypt* فتتم عملية فك تشفير الملف وتظهر لدينا الرسالة التالية:



الشكل (4-59) رسالة توضح إتمام فك التشفير

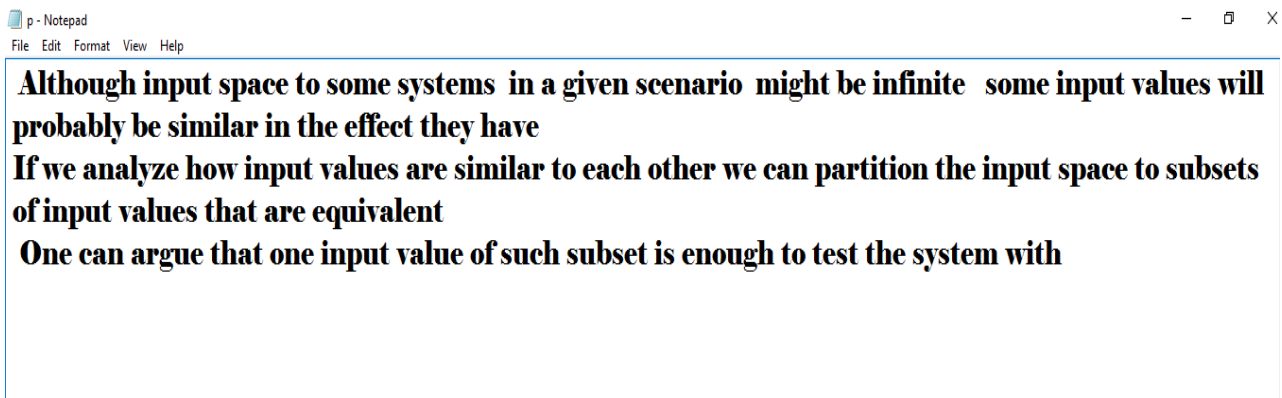
فيتولد ملف في نفس المسار اسمه *MonaleazaEncDec.docx* فنلاحظ أن محتوياته هي نفس محتويات الملف الأصلي والتي هي عبارة عن صورة ونص.



الشكل (4-60) يوضح محتويات الملف بعد فك تشفيره

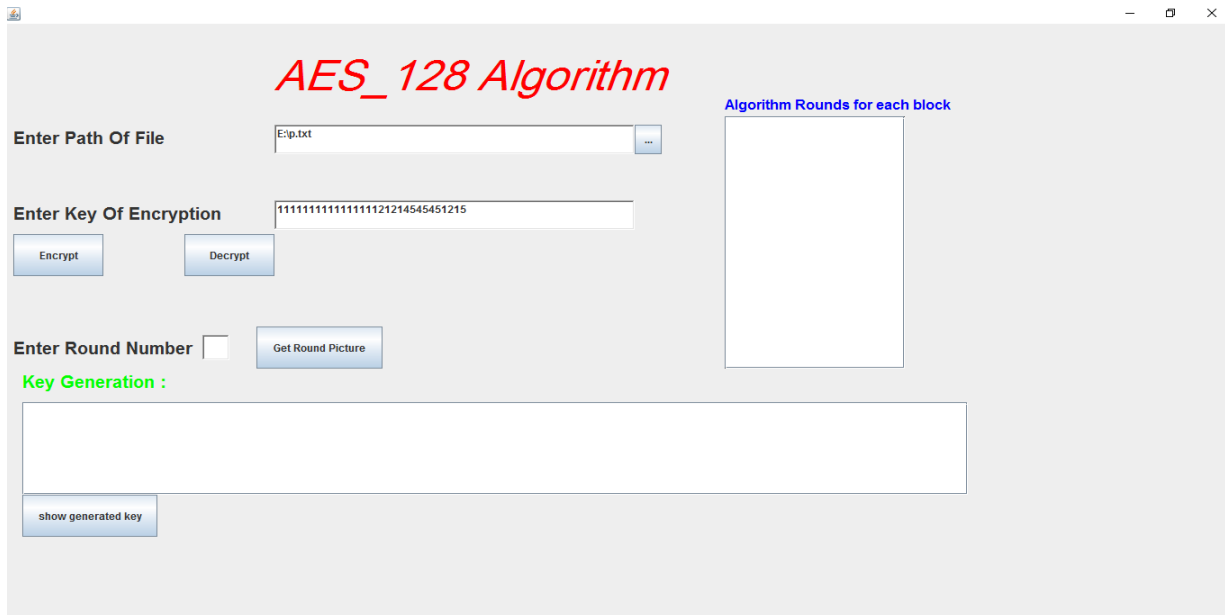
3.2.4 تشفير ملف نصي *txt* :

أولاً: ليكن لدينا الملف النصي التالي :



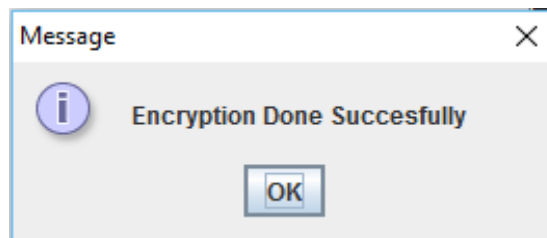
الشكل (4-61) ملف نصي

ثانياً: نختار مسار الملف في الواجهة الأساسية للخوارزمية ونختار مفتاح التشفير المكون من 16 byte أي 32 حرف.



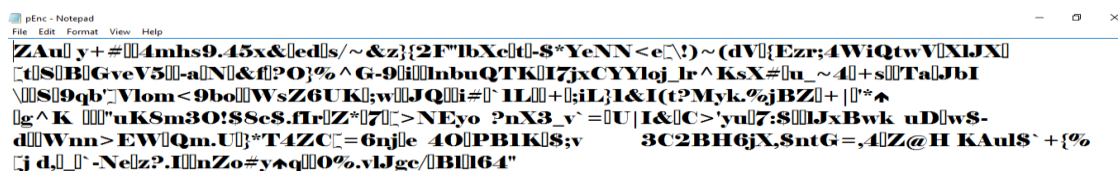
الشكل (4-62) توضيح اختيار ملف نصي ومفتاح تشفير

ثالثاً: نقوم بالضغط على زر *Encrypt* فتم عملية التشفير وتظهر الرسالة التالية:



الشكل (4-63) رسالة تؤكد إتمام عملية التشفير

رابعاً: يتولد ملف نصي في نفس المسار يحتوي على النص المشفر وعند فتح الملف نرى محتوياته كما يلي:



الشكل (4-64) يوضح محتويات الملف المشفر

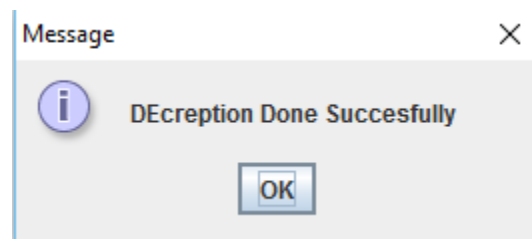
4.2.4 فك تشفير ملف نصي *txt* :

لفك التشفير نختار الملف المشفر كما في الخطوات السابقة أي ندخل المسار

ومفتاح فك التشفير ونضغط على زر *Decrypt*

فيتم فك تشفير الملف ويظهر لدينا ملف ومحتوياته هي نفس محتويات الملف المشفر

وتظهر لدينا الرسالة التالية :



نقوم بفتح الملف فنرى محتوياته كما يلي :

pEncDec - Notepad
File Edit Format View Help

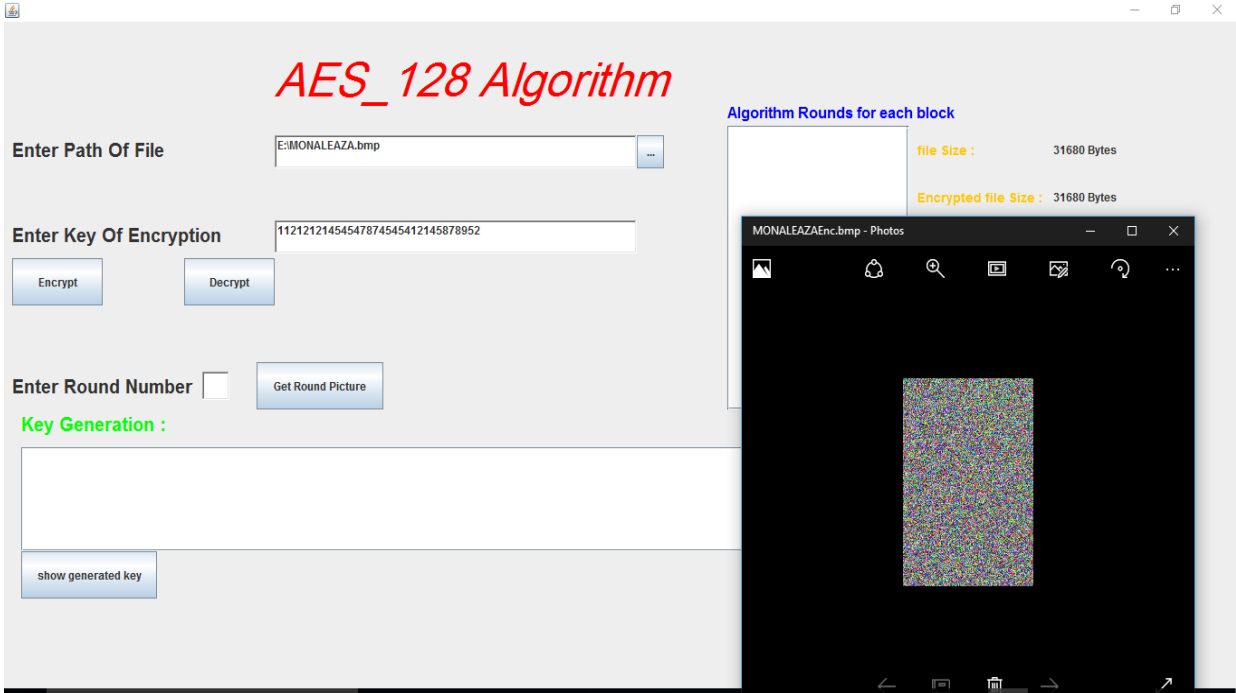
Although input space to some systems in a given scenario might be infinite some input values will probably be similar in the effect they have
If we analyze how input values are similar to each other we can partition the input space to subsets of input values that are equivalent
One can argue that one input value of such subset is enough to test the system with

الشكل (4-65) يوضح محتويات الملف الذي تم فك تشفيره

وبالتالي تمت عملية فك التشفير بنجاح.

5.2.4 تشفير صورة :

نتبع نفس خطوات التشفير السابقة على الواجهة حيث نقوم باختيار مسار الصورة المراد تشفيرها كما نختار مفتاح التشفير ونضغط على زر Encrypt فتم عملية تشفير الصورة فيظهر لدينا مايلي :



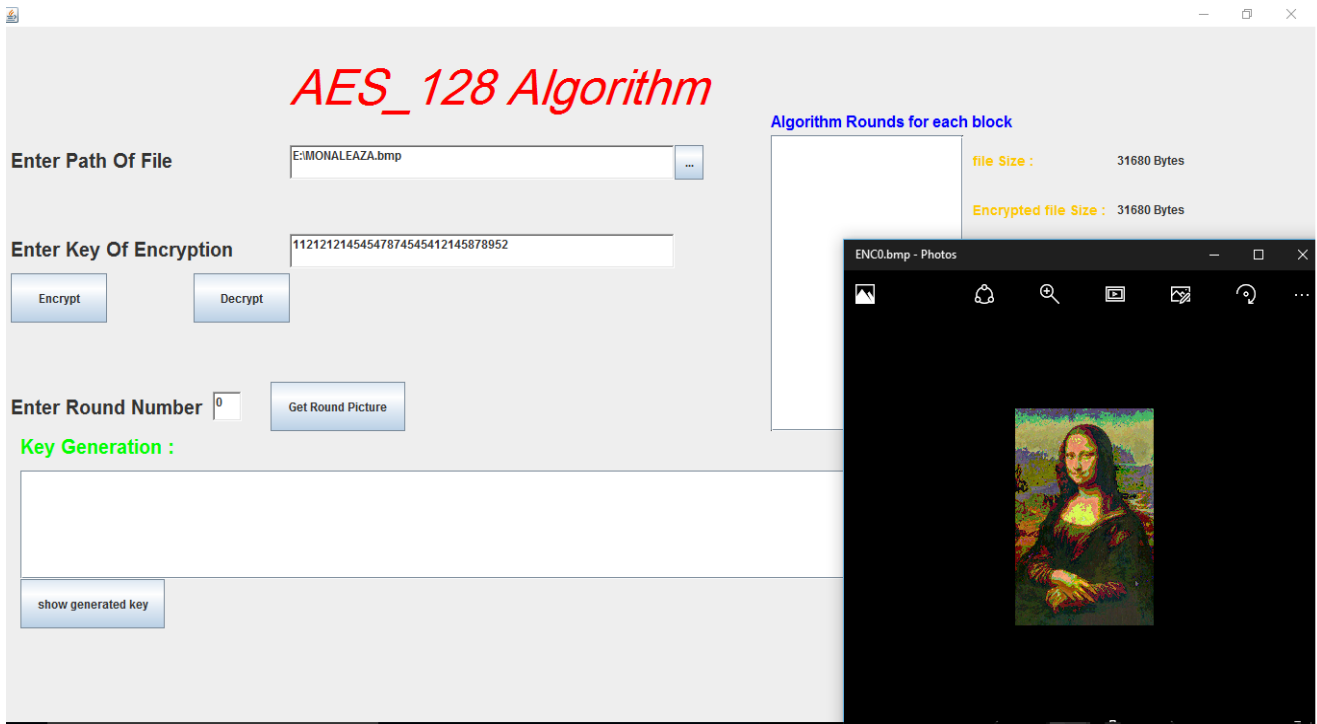
الشكل (4-66) يوضح آلية تشفير صورة

حيث أن الصورة التي ظهرت تمثل الصورة المشفرة .

- إضافة إلى الصورة المشفرة النهائية يمكننا أيضاً رؤية الصورة في أي مرحلة من مراحل عمل الخوارزمية وللقيام بذلك نقوم بإدخال رقم الدورة في الخانة المقابلة لـ *Enter Round Number* وبعد ذلك نضغط الزر *Get Round Picture* فتظهر لدينا الصورة في الناتجة عن المرحلة المدخلة.

فمثلاً لرؤية الصورة الناتجة عن المرحلة 0 نقوم بإدخال 0 في الخانة الخاصة ثم نضغط الزر

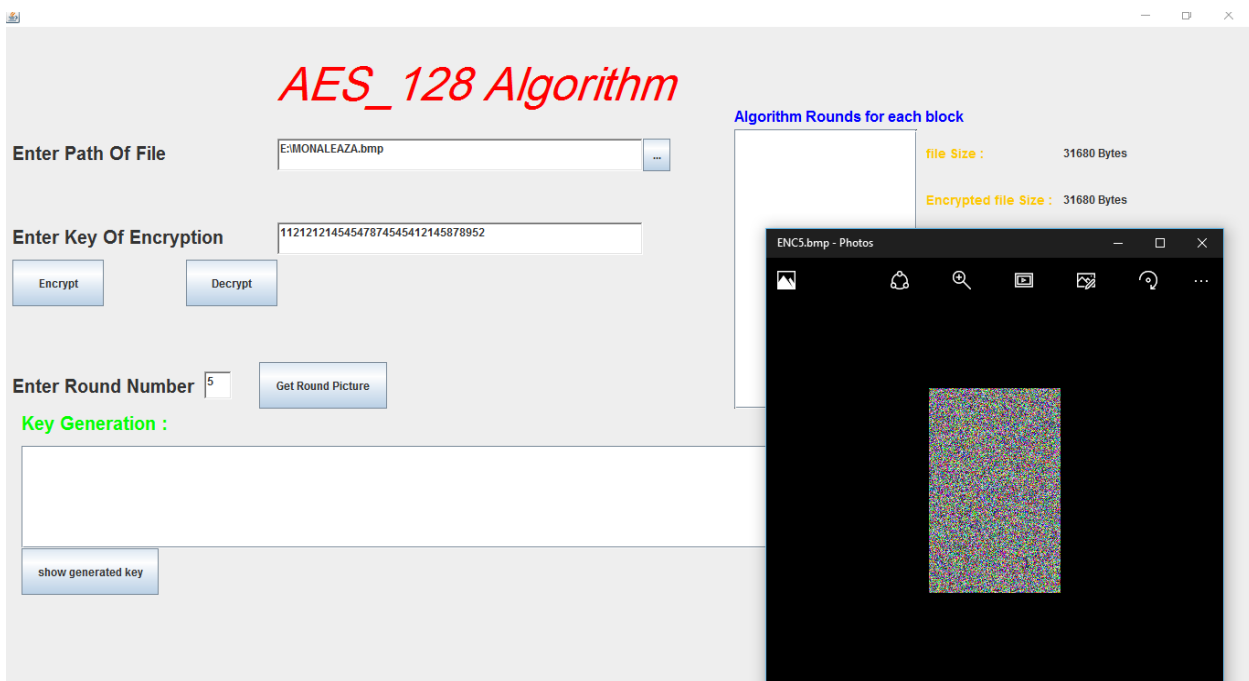
Get Round Picture فنلاحظ مايلي :



الشكل (4-67) خرج أول دورة من خوارزمية AES

ومثلاً لرؤية الصورة الناتجة عن المرحلة الخامسة نقوم بإدخال 5 ثم نضغط زر

Get Round Picture فنرى مايلي:



الشكل (4-68) يوضح خرج المرحلة 5 من خوارزمية AES

6.2.4 فك تشفير صورة:

عملية فك التشفير تتم بان ندخل مسار الصورة المشفرة الأساسية ونضغط على زر *Decrypt* فتظهر لدينا رسالة (فك التشفير تم بنجاح) .
نفتح الصورة الناتجة فنلاحظ أنها نفس الصورة المشفرة تماماً أي عمليتي التشفير وفك التشفير ناجحتين.

نتائج والتوصيات والمقترحات :

- ١ التطبيق يحقق فكرة التشفير طرف لطرف والتي هي الغاية الأساسية من بناء التطبيق كما أنه يحقق مقدار عالي من السرية والأمن الذي يطلبه المستخدمون.
- ٢ التشفير طرف لطرف تقنية حديثة نسبياً مما يجعل فكرة التطبيق فريدة عن الأفكار السابقة التي كانت تحقق فكرة التشفير كفكرة مجردة دون استخدامها بشكل واضح في التطبيقات الخدمية.
- ٣ استخدام تقنية *Node.js* التي تمكن المبرمج من جعل حاسوبه لسيرفر وذلك لاختبار التطبيقات التي تحتاج سيرفر وهي تقنية جديدة أيضاً.
- ٤ قمنا ببناء توابع مستقلة لكل تقنية مستخدمة في التطبيق وبالتالي يمكن استخدام هذه التوابع في أي مشاريع مستقبلية واستخدامها كتطبيقات جاهزة.
- ٥ يمكن تطوير هذا التطبيق وتشغيله على عتاد ذو إمكانيات فائقة وطرحه للاستخدام بعد إجراء التحسينات الممكنة.
- ٦ كنا نريد العمل أكثر على بعض الأمور مثل واجهة التطبيق وأن نجعل التطبيق قادر على إرسال واستقبال الملفات غير الصور والمراسلات العادية .

المراجع:

- ١ - الدكاك أميمة ، النجدي حاتم ، 2006 - التعمية التطبيقية . الطبعة الأولى ، الجمعية المعلوماتية السورية ، سوريا ، 894 صفحة .
- ٢ - حسن محمد ، شفا عمري معتصم ، برهوم بسيم ، 2007 - دراسة تحليلية ومقارنة لأشهر خوارزميات التعمية المتناظرة. مجلة جامعة تشرين للدراسات والبحوث العلمية .

- 3- BEHAOUZA,2008-Introduction to Cryptography and Network Security , McGrawHill,1st Ed ,United States , 721.
- 4- Bruce Schneier ,Applied cryptography, second Eition,662.
- 5- Joachim Rosenthal Cryptography ,
- 6- Joan Daemen , Vincent Rijmen ,2001-The Design of Rijndael ,Germany,128.
- 7-Eveny Melanov ,2009- The RSA Algorithm .
- 8- William Stallings 2005-Cryptography and Network Security Principles and Practices, 4th Ed , 592.

وتم الاستعانة ببعض المواقع :

- 1- <https://socket.io/get-started/chat>
- 2- http://mawdoo3.com/%D8%A8%D8%AD%D8%AB_%D8%B9%D9%86_%D8%A3%D9%85%D9%86_%D8%A7%D9%84%D9%85%D8%B9%D9%84%D9%88%D9%85%D8%A7%D8%AA
- 3-<https://www.youtube.com/channel/UC7OGxluGpaD5NHMNoKj33pw?spfreload=10>.
- 4- <https://ar.wikipedia.org/wiki/Node.js>.
- [Advanced_Encryption_Standard](#).
- [RSA](#).