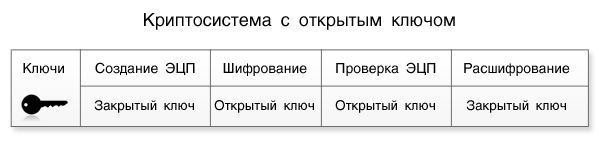
В криптосистеме с открытым ключом, в отличие от симметричной, используются два ключа: открытый и закрытый (закрытый хранится в секрете). **Открытый ключ** используется для проверки ЭЦП и для шифрования сообщений. **Закрытый ключ** – для генерации ЭЦП и для расшифрования сообщений.

Для тех, кто забыл, напомним:  
**ЭЦП** – (Электронная цифровая подпись) – атрибут электронного документа, который получается в результате некоторого криптографического преобразования данных. **ЭЦП** позволяет проверить целостность документов, конфиденциальность документов, а так же идентифицировать владельца документа. Аналог обычной подписи.

[](http://teh-box.ru/wp-content/uploads/2014/11/algoritm-shifrovaniya-rsa-na-palcax-2.jpg)

Стоит отметить, в основе криптографических систем с открытым ключом лежат **односторонние функции** – такие функции, которые обладают следующими свойствами:

1. Пусть известно значение xx, тогда вычислить F(x)F(x)относительно просто.  
2. Пусть известно y=F(x)y=F(x), однако вычислить xx сложно.

Не стоит понимать под этим определением теоретическую однонаправленность, здесь имеется ввиду, что практически невозможно вычислить обратное значение функции за разумное время.  
Необходимо понимать, что криптосистемы с открытым ключом не заменяют симметричные криптосистемы, это связано с тем, что:

1. Скорость работы алгоритмов с открытым ключом намного ниже, чем скорость работы симметричных алгоритмов. Поэтому ассиметричные шифры используются для шифрования небольших по размеру данных, например, ключей.  
2. Длина ключей значительно больше, чем в симметричных криптосистемах.

Конечно, есть и плюсы, например:

1. Удобное распределение открытых ключей, не требует секретности.  
2. В больших сетях число ключей значительно меньше, чем в симметричной криптосистеме.

А теперь перейдем к конкретному примеру ассиметричного шифрования – RSA.

**Описание алгоритма RSA.**

Описание может быть не очень понятным и нудным, наберитесь терпения, когда мы дойдем до примера, все встанет на свои места

В основе RSA лежит задача факторизации произведения двух простых больших чисел. Для шифрования используется простая операция возведения в степень по модулю NN. Для расшифрования же необходимо вычислить функцию Эйлера от числа NN, для этого необходимо знать разложение числа n на простые множители (В этом и состоит задача факторизации).

В RSA открытый и закрытый ключ состоит из пары целых чисел. Закрытый ключ хранится в секрете, а открытый ключ сообщается другому участнику, либо где-то публикуется.

На словах понять тяжело, поэтому предлагаю разобрать все по частям.

**Генерация ключей RSA**

Всё начинается с генерации ключевой пары (открытый, закрытый ключ). Генерация ключей в RSA осуществляется следующим образом:

1. Выбираются два простых числа pp и qq (такие что pp неравно qq).  
2. Вычисляется модуль N=p∗qN=p∗q.  
3. Вычисляется значение функции Эйлера от модуля NN: ϕ(N)=(p−1)(q−1)ϕ(N)=(p−1)(q−1).  
4. Выбирается число ee, называемое открытой экспонентой, число ee должно лежать в интервале  
11<<ee<<ϕ(N)ϕ(N), а так же быть взаимно простым со значением функции ϕ(N)ϕ(N). 5. Вычисляется число dd, называемое секретной экспонентой, такое, что d∗e=1(modϕ(N))d∗e=1(modϕ(N)), то есть является мультипликативно обратное к числу ee по модулю ϕ(N)ϕ(N).

Итак, мы получили пару ключей:

Пара (e,N)(e,N) - открытый ключ.  
Пара (d,N)(d,N) - закрытый ключ.

Если вам нужно сгенерировать пару, используйте [OpenSSL](https://www.openssl.org/" \t "_blank):



|  |  |
| --- | --- |
| 1  2 | $ openssl genrsa -out private\_key.pem 4096  $ openssl rsa -pubout -in private\_key.pem -out public\_key.pem |

А теперь переходим к шифрованию и расшифрованию.

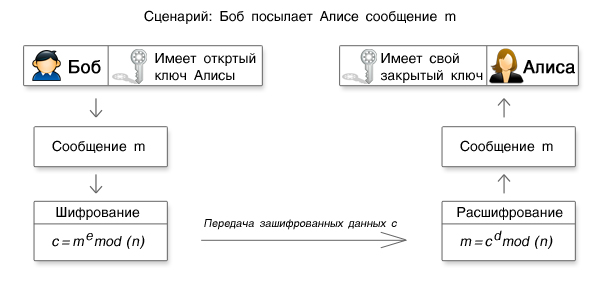
**Шифрование и расшифровывание в RSA**

Есть следующий сценарий: Боб и Алиса переписываются в интернете, но хотят использовать шифрование, чтобы поддерживать переписку в секрете :). Алиса заранее сгенерировала закрытый и открытый ключ, а затем отправила открытый ключ Бобу. Боб хочет послать зашифрованное сообщение Алисе:

**Шифрование:** Боб шифрует сообщение mm, используя открытый ключ Алисы (e,N)(e,N) : C=E(M)=Memod(N)C=E(M)=Memod(N), и отправляет с Алисе.

**Расшифровывание:** Алиса принимает зашифрованное сообщение cc. Используя закрытый ключ (d,N)(d,N), расшифровывает сообщение M=D(C)=Cdmod(N)M=D(C)=Cdmod(N)

Проиллюстрируем то, что описано выше:

[](http://teh-box.ru/wp-content/uploads/2014/11/algoritm-shifrovaniya-rsa-na-palcax-3.jpg)

**Безопасность схемы RSA**

От каких параметров зависит стойкость алгоритма RSA? Представим себе, что к Бобу и Алисе присоединяется Ева, которая хочет узнать, какое сообщение послал Боб. Допустим, что у Евы есть открытый ключ Алисы (e,N)(e,N), для того, чтобы расшифровать сообщение cc, необходимо знать закрытый ключ (d,N)(d,N). Мы знаем, что d∗e=1(modϕ(N))d∗e=1(modϕ(N)), однако Ева не знает ϕ(N)=(p−1)∗(q−1)ϕ(N)=(p−1)∗(q−1), т.е задача сводится к нахождению простых чисел pp и qq (хотя это не всегда так), которые связаны с известным NN следующим образом N=p∗qN=p∗q.

Делаем выводы. Чтобы алгоритм был стойким, необходимо:

1. Выбрать два больших простых случайных числа pp и qq (к примеру, >= 1024 бита каждое), должны быть не слишком различными и быть не слишком близкими  
2. Наибольший общий делитель (p−1)(p−1) и (q−1)(q−1) должен быть небольшим, в лучшем случае равен двум.  
3. Выбрать большое значение открытой экспоненты ee, как правило, выбирают простые числа Ферма: 17, 257, 65537...  
4. Сохранение в секрете закрытого ключа.

Ниже («Почему RSA не так прост?») я покажу, что эти условия необходимые, но не достаточные. Для стойкости шифра RSA необходимы еще некоторые условия.

Далее я хотел бы разобрать пример шифрования и расшифрования с конкретными числами для полного понимания работы алгоритма, показать, какие существуют атаки на RSA и понять, почему использовать алгоритм RSA, который описан выше нельзя.

**Пример шифрования RSA**

Поехали!

1. Выбираем простые числа (небольшие, чтобы упростить вычисления) : p=3p=3 и q=11q=11  
2. Вычисляем модуль n=p∗q=3∗11=33n=p∗q=3∗11=33  
3. Вычисляем функцию Эйлера от модуля NN : ϕ(N)=(p−1)∗(q−1)=2∗10=20ϕ(N)=(p−1)∗(q−1)=2∗10=20.  
4. Выбираем открытую экспоненту e=7e=7  
5. Определяем закрытую экспоненту dd : d∗e?1(modϕ(N))d∗e?1(modϕ(N)) =>=> d=3d=3

Будем шифровать сообщение RSA, пусть букве A соответствует цифра 1, B - 2, C - 3 и т.д (Подобное соответствие вносим для простоты), тогда :

R=18;R=18; S=19;S=19; A=1;A=1;

Открытый ключ : (e,N)=(7,33)(e,N)=(7,33)

C1=(187)mod33=6C1=(187)mod33=6  
C2=(197)mod33=13C2=(197)mod33=13  
C3=(17)mod33=1C3=(17)mod33=1

C("RSA")=6131C("RSA")=6131

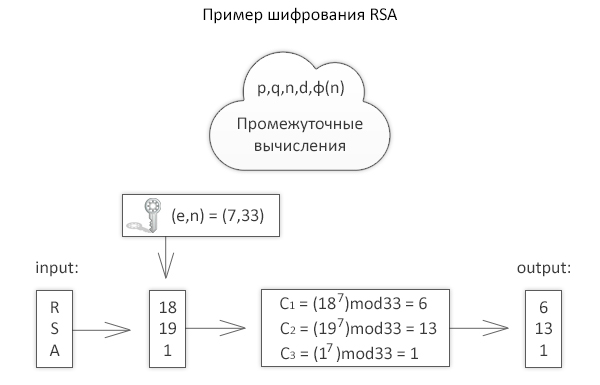
Передача шифртекста происходит в формате hex:



|  |  |
| --- | --- |
| 1  2 | >>> hex(6), hex(13), hex(1)  ('0x6', '0xd', '0x1') |

То есть передается строка вида "0x6 0xd 0x1"

Иллюстрация примера:

[](http://teh-box.ru/wp-content/uploads/2015/01/algoritm-shifrovaniya-rsa-na-palcax-8.jpg)

**Пример расшифрования RSA**

На стороне получателя строка "0x6 0xd 0x1" декодируется:



|  |  |
| --- | --- |
| 1  2 | >>> int("0x6", 16), int("0xd", 16), int("0x1", 16)  (6, 13, 1). |

Используем закрытый ключ : (d,N)=(3,33)(d,N)=(3,33)

M1=(63)mod33=18M1=(63)mod33=18  
M2=(133)mod33=19M2=(133)mod33=19  
M3=(13)mod33=1M3=(13)mod33=1

18=R;18=R; 19=S;19=S; 1=A;1=A;

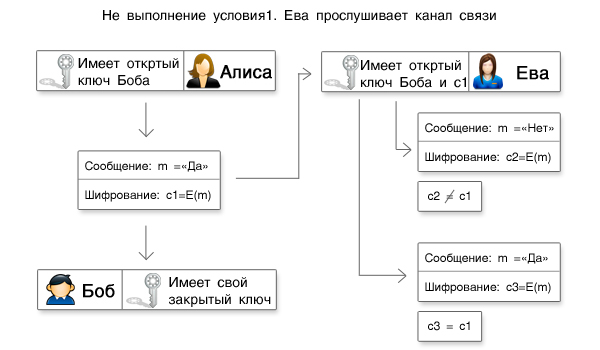
Получаем исходное сообщение - RSA. Иллюстрировать не будем, и так все понятно

**Почему RSA не так прост?**

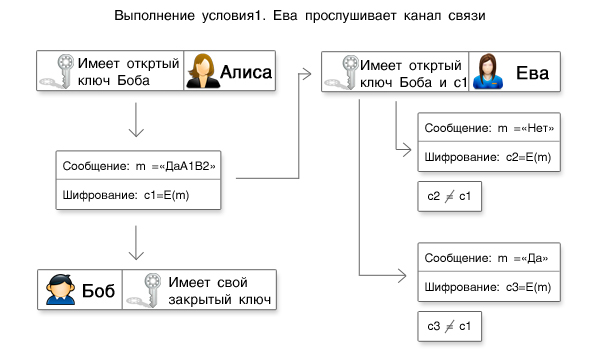
В статье я описал наиболее простую форму RSA, вроде все просто, однако использовать такой алгоритм нельзя. Почему нельзя? Необходимо проверить, удовлетворяет ли данный алгоритм требованиям к ассиметричным криптосистемам.

**Условие 1.**  
Ассиметричный алгоритм шифрования является стойким, если атакующий имеет два открытых текста M1M1 и M2M2, а так же зашифрованный текст CiCi, не может с вероятностью большей, чем 1212 определить к какому из сообщений M1M1 или M2M2 относитсяCiCi

**Проверка условия 1.**  
Перейдем к RSA и проверим условие 1. Вспомним ситуацию с Алисой и Бобом, допустим, канал связи прослушивает Ева. Боб спрашивает у Алисы: «Алиса, мы идем сегодня в кино», причем сообщение не шифруется. Алиса отвечает Бобу, но не хочет, чтобы кто-то знал, поэтому шифрует свой ответ на открытом ключе Боба и отправляет шифротекст Бобу. (Предполагается, что Алиса отвечает Бобу монотонно). Ева перехватывает зашифрованное сообщение и знает, что Алиса ответила либо «Да», либо «Нет». Ева располагает открытым ключом Боба, поэтому последовательно шифрует сообщение «Да» и «Нет», соответственно одно из них совпадет с зашифрованным сообщением Алисы и Ева узнает, пойдет ли Алиса сегодня в кино или нет

[](http://teh-box.ru/wp-content/uploads/2014/11/algoritm-shifrovaniya-rsa-na-palcax-4.jpg)

Из этого видно, что упрощенное описание алгоритма RSA не годится для практического использования. Как решается данная проблема на практике? Решается эта проблема достаточно просто: к сообщению добавляется некоторая случайная величина, а затем полученный текст шифруется. Таким образом, если Ева перехватывает сообщение C1=E("C1=E("ДаA1B2")"), то зашифровав «Да» и «Нет»: C2=E("C2=E("Нет")"), C3=E("C3=E("Да")"), будет видно, что C1C1, C2C2 и C3C3 не совпадут.

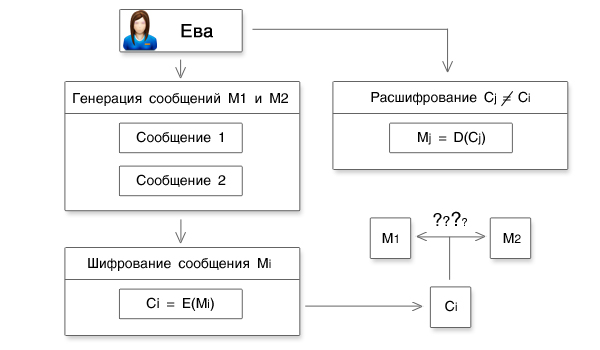
[](http://teh-box.ru/wp-content/uploads/2014/11/algoritm-shifrovaniya-rsa-na-palcax-5.jpg)

Итак, в алгоритме RSA перед тем как зашифровать сообщение, к тексту добавляется некоторая случайная величина, а затем текст проходит процедуру шифрования. Поэтому функция шифрования принимает вид :

C=(M||random)emod(N)C=(M||random)emod(N), вместо C=Memod(N)C=Memod(N)

**Условие 2.**

Допустим, у Евы есть две функции, одна F1F1 шифрует сообщения, вторая F2F2 расшифровывает шифротекст. Затем Ева генерирует два сообщения M1M1 и M2M2. Затем наугад одно из сообщений шифруется функцией F1F1, на выходе функции - шифротекст CiCi. CiCiвозвращается Еве, её задача угадать с вероятностью большей, чем 1212 к какому из сообщений M1M1 или M2M2 принадлежит CiCi. При этом Ева может расшифровать любое сообщение, кроме CiCi(иначе задача лишена смысла). Считается, что криптосистема с открытым ключом стойкая, если злоумышленник не может с вероятностью большей, чем 1212 сказать какому из сообщений соответствует шифротекст.

[](http://teh-box.ru/wp-content/uploads/2014/11/algoritm-shifrovaniya-rsa-na-palcax-6.jpg)

**Проверка условия 2.**

Перейдем к RSA и проверим условие 2. Пусть у Евы есть два открытых сообщения M1M1 и M2M2 и один шифротекст Ci=Me1mod(N)Ci=M1emod(N). Что делает Ева? Она создает сообщение, используя открытый ключ (e,N)(e,N) : C∗=2eC∗=2e CiCi mod(N)mod(N), затем используя функцию F2F2 расшифровыает это сообщение, таким образом M∗=C∗dmod(N)M∗=C∗dmod(N) == 2ed∗Med1mod(N)2ed∗M1edmod(N) == 22 M1M1 mod(N)mod(N), вычисляя M∗2M∗2 Ева получит сообщение M1M1.

Вышесказанное еще раз показывает, что использовать на практике упрощенный алгоритм RSA нельзя. Решается это проблема почти так же как и в случае с условием1, напомню, что в условии1 мы добавляли к сообщению абсолютно случайную и непрогнозируемую информацию, а затем шифровали текст. Теперь накладывам еще одно требование: необходимо, чтобы дополнительные блоки помогали определить, был ли шифротекст получен в результате шифрующей функции или он смоделирован злоумышленником. Для проверки расшифрованных данных на подлинность используются всем известные хеш-функции.

Такая схема в шифровании RSA называется **RSA-OAEP(Optimal asymmetric encryption padding)**, рассмотрим подробнее OAEP на примере.

**RSA-OAEP**

**Шифрование:**  
Для того чтобы зашифровать текст в RSA-OAEP делается следующее:

1. Выбираются две хеш-функции H(x)H(x) и G(x)G(x), такие, что суммарная длина результаов хеш-функии не была больше длины ключа RSA.  
2. Генерируется строка битов LL. Причем последовательность должна быть случайной, а длина не должна превышать длину результата хеш-функции H(x)H(x).  
3. Открытый текст MM разбивается на блоки по kk-бит. Каждому блоку mimi дописывается (p−k)(p−k) нулей, где число pp - длина хеш- функции G(x)G(x).  
4. Определяется набор бит (m||0(p−k)⊕G(L))||(L⊕H(p||0(p−k)⊕G(L))(m||0(p−k)⊕G(L))||(L⊕H(p||0(p−k)⊕G(L))  
5. Биты, полученные на шаге 4, представляются в виде целого числа M1M1  
6. Определяется шифротектс : C=Me1mod(N)C=M1emod(N)

**Расшифрование:**

Для того, что бы расшифровать сообщение делается следующее:

1. Определяется M1:M1=Cdmod(N)M1:M1=Cdmod(N)  
2. В полученной последовательности бит отсекают левую часть (pp левых бит числа M1M1, pp - длина хеш функции G(x)G(x). Пусть эти биты TT : T=(m||0(N−k)⊕G(L))T=(m||0(N−k)⊕G(L))  
3. Определяется H(T)=H(m||0(N−k)⊕G(L))H(T)=H(m||0(N−k)⊕G(L))  
4. Исходя из того, что известно H(T)H(T), т.к. знаем L⊕H(T)L⊕H(T)(правая часть блока), получаем LL.  
5. Находим mm, из условия, что T⊕G(L)T⊕G(L), а TT в свою очередь : T=(m||0(N−k)⊕G(L))T=(m||0(N−k)⊕G(L))  
6. На этом шаге необходимо проверить полученное m, если оно заканчивается (p−k)(p−k)-нулями, то сообщение зашифровано верно, иначе шифротекст некорректен, то есть оно подделано злоумышленником

**Выводы**

Если вы дочитали эту статью до конца, то вы большой молодец :). Что мы вынесли из всего этого.

* Познакомились с криптосистемой с открытым ключом.
* Узнали что такое ЭЦП.
* Познакомились с алгоритмом RSA.
* Узнали, как генерируются ключи в RSA.
* Как происходит шифрование в RSA.
* Разобрали конкретные примеры.
* Узнали о «подводных камнях» упрощенного алгоритма RSA.

Я надеюсь, что данный материал был не только полезен, но и интересен :). А чтобы прокачать свой скилл еще больше предлагаю ознакомиться с материалом, который представлен ниже.

1. В поле *"Каталоги включения"* добавить каталог:
2. D:\Work\mpir-2.7.2\build.vc14\lib\_mpir\_gc\Win32\Debug\

Здесь прредполагается, что:

* + исходные коды библиотеки были помещены в каталог D:\Work\;
  + используется MPIR версии 2.7.2 (mpir-2.7.2);
  + используется среда разработки Visual Studio 2015 (build.vc14);
  + используется статическая версия библиотеки MPIR (lib\_mpir\_gc);
  + настраиваются свойства проекта для платформы *"Win32"* (Win32);
  + настраиваются свойства проекта для конфигурации *"Debug"* (Debug).

1. В поле *"Каталоги библиотек"* добавить тот же самый каталог. Если используется статическая версия библиотеки MPIR, в это поле также необходимо добавить следующий каталог (в предыдущих предположениях):

D:\Work\mpir-2.7.2\build.vc14\lib\_mpir\_cxx\Win32\Debug\