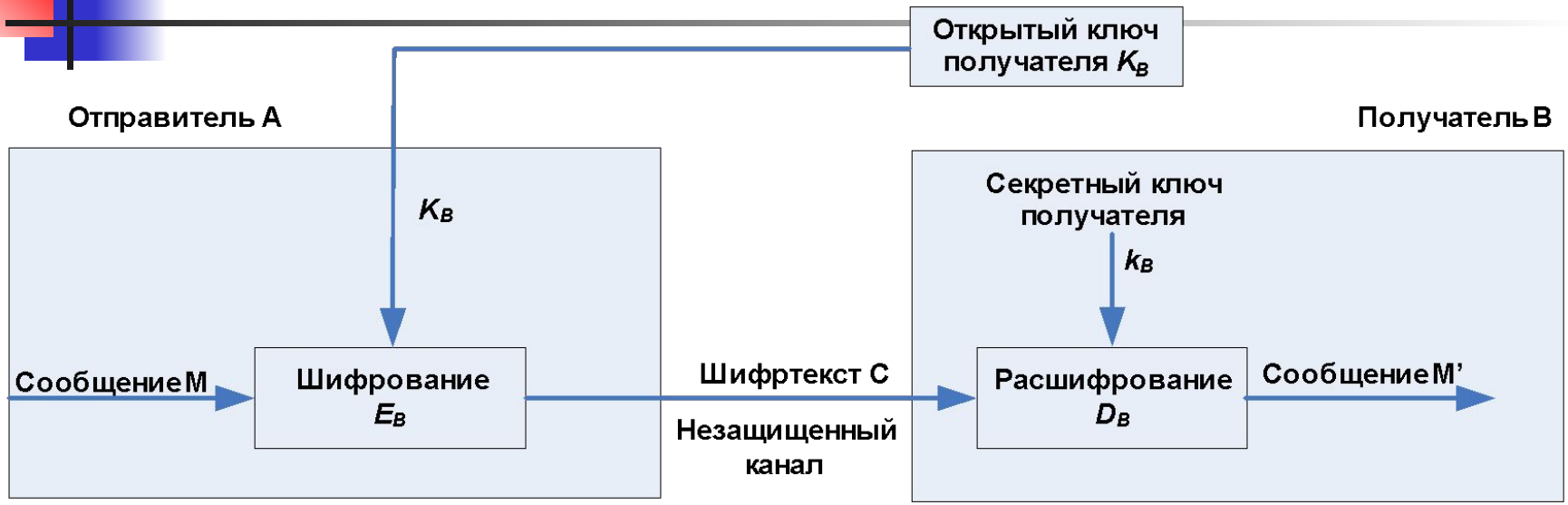
Криптография

Асимметричные криптосистемы

Проблемы симметричного шифрования

- требование защищенности и надежности канала передачи секретного ключа для каждой пары участников информационного обмена
- повышенные требования к службе генерации и распределения ключей
  - при взаимодействии «каждый с каждым»для *n* абонентов требуется *n(n-1)/2* ключей (квадратическая зависимость)

Асимметричная криптосистема



- *открытый ключ* $K_в$
  - используется для шифрования, вычисляется из секретного ключа $k_в$
- *секретный ключ* $k_в$
  - используется для расшифрования, зашифрованной с помощью парного ему открытого ключа $K_в$

Особенности асимметричных криптосистем

- Открытый ключ K_B и криптограмма C могут быть отправлены по незащищенным каналам
- Алгоритмы шифрования и расшифрования являются открытыми

$$E_B : M \to C$$

$$D_B : C \to M$$

Однонаправленные функции

- X, Y – некоторые произвольные множества
- Функция $f:X \rightarrow Y$ однонаправленная, если
  - для всех x *из* X легко вычислить $y=f(x)$, где y *из* Y,
  - но для большинства y достаточно сложно найти x такое, что $f(x)=y$
- Примеры:
  - целочисленное умножение
  - модульная экспонента с фиксированным основанием и модулем

Целочисленное умножение

- Прямая задача
  - вычисление произведения $N=P\times Q$, *где* P и Q - очень большие
- Обратная задача – факторизация
  - нахождение делителей P и Q большого целого числа $N=P\times Q$
  - практически неразрешима при достаточно больших значениях N
  - при $N\approx 2^{664}$ и $P\approx Q$ для разложения числа N потребуется около 10^{23} операций - практически невозможно для современных компьютеров

Модульная экспонента с фиксированным основанием и модулем-1

- Прямая задача:
  - Пусть A и N – целые числа, $1 \le A < N$
  - Модульная экспонента с основанием A и модулем N - функция:

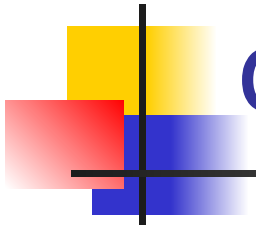
$$f_{A,N}(x) = A^x \pmod{N},$$

где x – целое число, $1 \le x < N$

Модульная экспонента с фиксированным основанием и модулем-2

- Обратная задача - задача нахождения дискретного логарифма:
  - Если $y=A^x$, то $x=log_A(y)$
  - Для известных целых A, N, y поиск целого числа x, такого что $A^x \pmod N = y$
- Алгоритм вычисления дискретного логарифма за приемлемое время пока не найден
- При $A \approx 2^{664}$ и $N \approx 2^{664}$ поиск дискретного логарифма требует ~ 10^{26} операций, что в 1000 раз сложнее задачи факторизации
- При увеличении длины чисел разница в оценках сложности задач возрастает

Однонаправленные функции с секретом



- Функция относится к классу однонаправленных функций с секретом, если
  - является однонаправленной
  - возможно эффективное вычисление обратной функции, если известен секрет
- Секрет:
  - секретное число
  - строка
  - другая информация, ассоциирующаяся с данной функцией

Преимущества асимметричных криптосистем перед симметричными

- Решена проблема распределения ключей между пользователями
- Линейная, а не квадратическая зависимость числа ключей от числа пользователей
  - Для N абонентов используется $2 \times N$ ключей
- Возможность реализации протоколов взаимодействия сторон, которые не доверяют друг другу
  - закрытый ключ должен быть известен только его владельцу

Недостатки асимметричных криптосистем

- пока нет математического доказательства необратимости используемых в асимметричных алгоритмах функций
- асимметричное шифрование существенно медленнее симметричного (используются ресурсоемкие операции)
- необходимо защищать открытые ключи от подмены

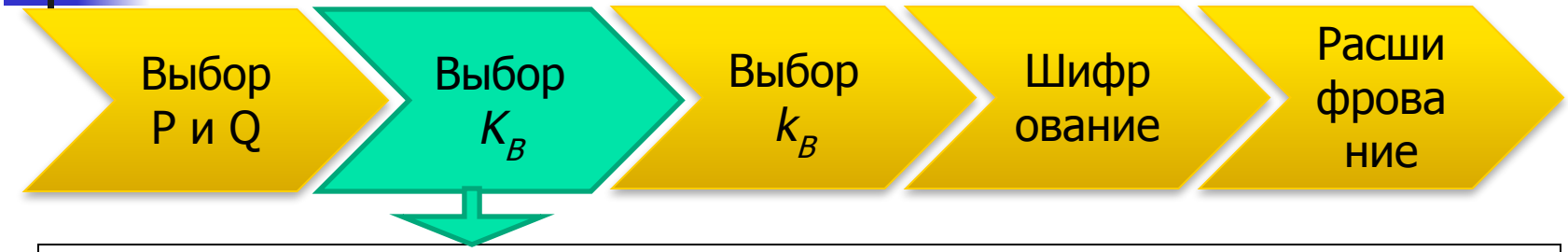
Алгоритм шифрования RSA

- в 1978 г. предложили 3 автора
  - Ron Rivest, Adi Shamir, leonard Adleman
- Режимы работы RSA:
  - шифрование данных
  - электронная цифровая подпись
- Надежность RSA
  - факторизация больших чисел
  - вычисление дискретных логарифмов в конечном поле

Алгоритм RSA. Шаг 1. Выбор P и Q

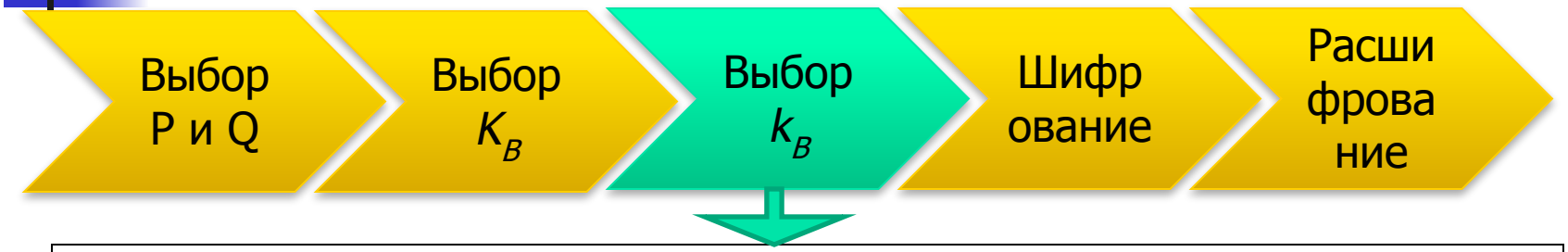
- *Предпосылки:*
  - Z_N={0,1,…,N-1} - множество целых чисел
  - открытый ключ K_B, секретный ключ k_B, сообщение M и криптограмма C принадлежат Z_N
- *Выбор P и Q*
  - случайные большие простые числа
  - равной длины
  - хранятся в секрете
- N – модуль: $N=P\times Q$

Алгоритм RSA.
Шаг 2. Выбор открытого ключа



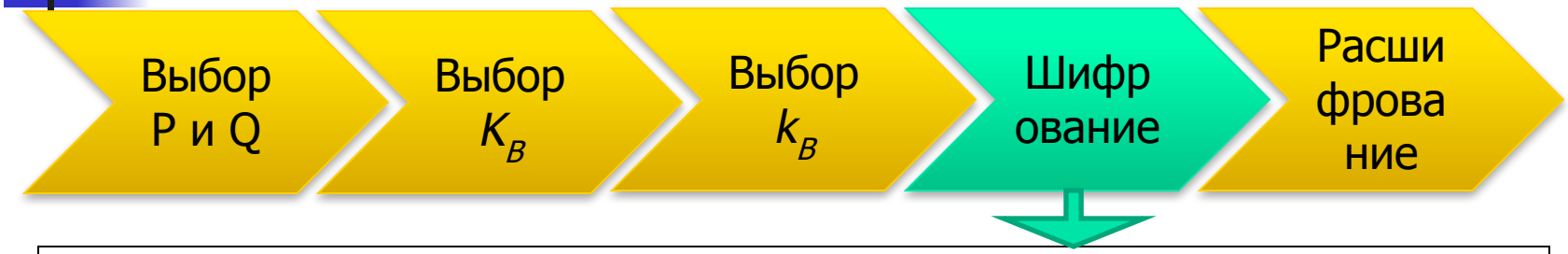
- Условия выбора K_B:
  - K_B - *случайный*
  - $1 < K_B \leq \varphi(N)$
  - НОД(K_B, $\varphi(N)$)=1
  - $\varphi(N)=(P-1)(Q-1)$ – функция Эйлера
    - $\varphi(N)$=количеству положительных целых чисел в интервале от 1 до N, которые взаимно просты с N

Алгоритм RSA.
Шаг 3. Вычисление секретного ключа



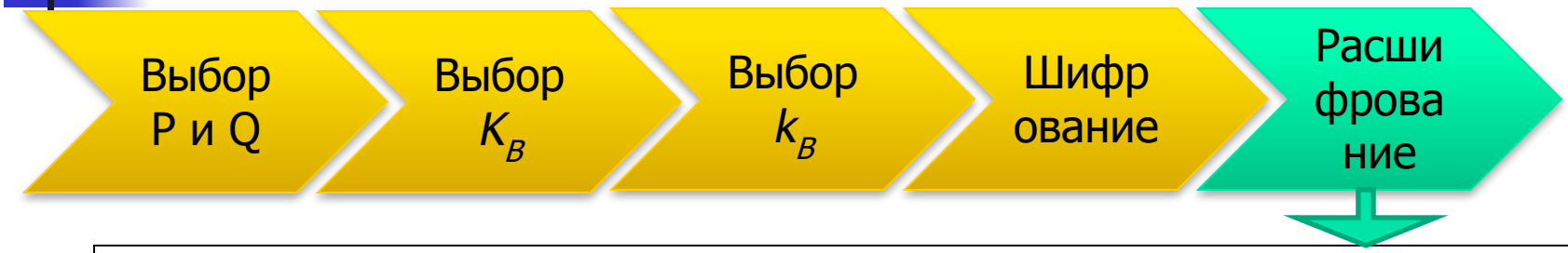
- Предпосылки:
  - получатель *B* знает пару простых чисел *P* и *Q*
  - может вычислить φ(*N*)
- Вычисление k_B (расширенный алгоритм Евклида):
  - $K_B \times k_B \equiv 1 \pmod{\varphi(N)}$
  - k_B и *N* должны быть взаимно простыми

Алгоритм RSA.
Шаг 4. Шифрование блоками



- До шифрования
  - разбивка исходного открытого текста *M* на блоки
  - каждый блок представляется числом M_i=0,1,2,…*N*-1.
- Формула шифрования: $C_i = E_{K_B}(M_i) = M_i^{K_B} (mod\ N)$
- Алгоритм быстрого вычисления значения C_i
  - последовательные возведения в квадрат целого M_i
  - умножения на M_i с приведением по модулю *N*
- Отправка криптограммы $C_1, C_2, \ldots, C_i, \ldots$ получателю

Алгоритм RSA. Шаг 5. Расшифрование



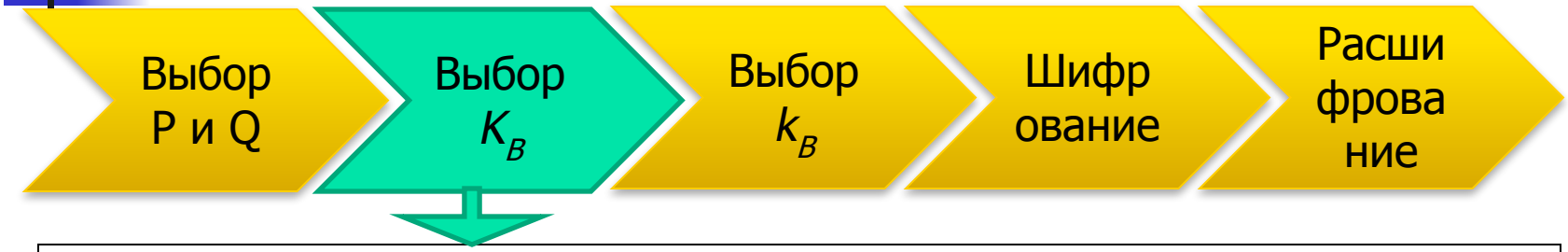
- Расшифрование криптограммы по формуле

$$M_i = D_{k_B}(C_i) = C_i^{k_B} (mod\ N)$$

Пример использования RSA. Шаг 1

- Задача: зашифруем сообщение “CAB”
- Действия получателя сообщения B:
  - Выбор P=3, Q=11
  - Вычисление модуля $N=P\times Q=3\times 11=33$
  - Вычисление значения функции Эйлера для N=33:
  - $\varphi(N)=\varphi(33)=(P-1)(Q-1)=2\times 10=20$

Пример использования RSA. Шаг 2

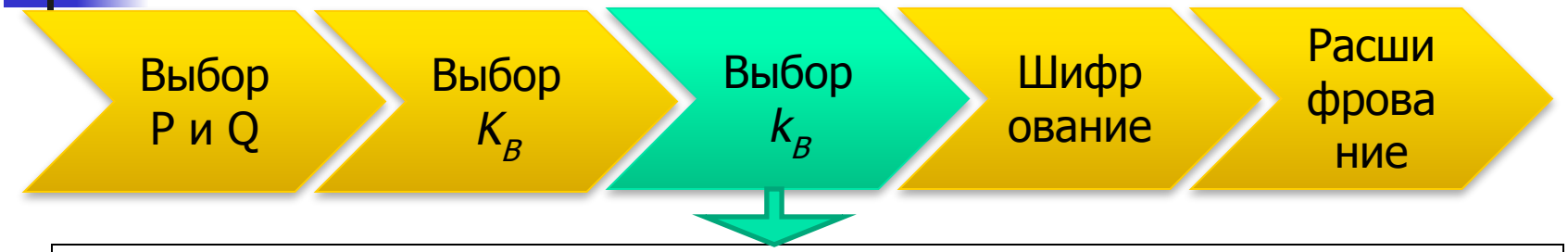


- Выбор в качестве открытого ключа K_B произвольного числа с учетом выполнения условий

$$1 < K_B \leq 20, \text{НОД}(K_B, 20)=1.$$

- Пусть $K_B=7$

Пример использования RSA. Шаг 3



- Вычисление секретного ключа k_B (алгоритм Евклида) при решении сравнения

$$k_B = 7^{-1} (\mathrm{mod}\ 20)$$

- Решение дает $k_B = 3$
- Пересылка пользователю *A* пары чисел (N=33, K_B =7)

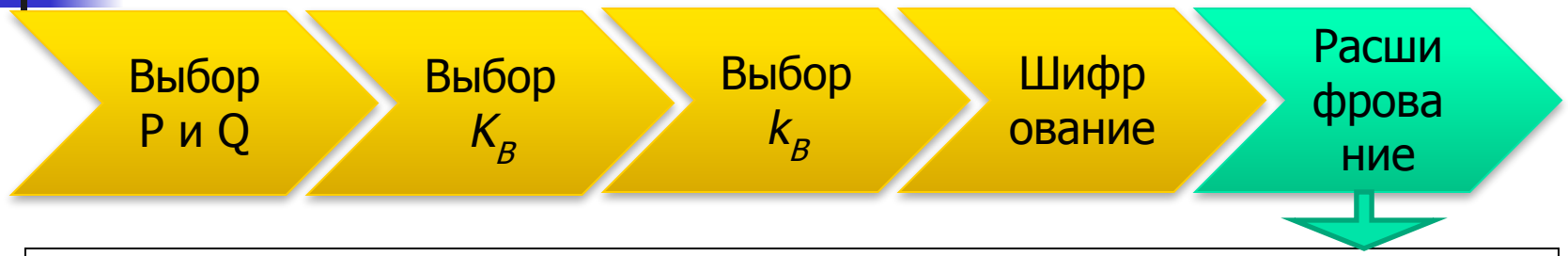
Пример использования RSA. Шаг 4

- *Действия пользователя А*
  - Кодирование - Пусть буква А представляется как число 1, буква В как 2, буква С как 3
  - Тогда сообщение «CAB» представляет последовательность «312», т.е. M_1=3, M_2=1, M_3=2.
  - Шифрование с ключом K_B=7 и N=33 по формуле $C_i = M_i^{\,7}(mod\ 33)$

$$C_1 = 3^7(mod\ 33) = 2187(mod\ 33) = 9$$
$$C_2 = 1^7(mod\ 33) = 1(mod\ 33) = 1$$
$$C_3 = 2^7(mod\ 33) = 128(mod\ 33)$$

  - Отправка пользователю В криптограммы C_1, C_2, C_3=9,1,29.

Пример использования RSA. Шаг 5



- *Действия пользователя B*
  - Расшифровка принятой криптограммы C_1, C_2, C_3 секретным ключом k_B=3 по формуле

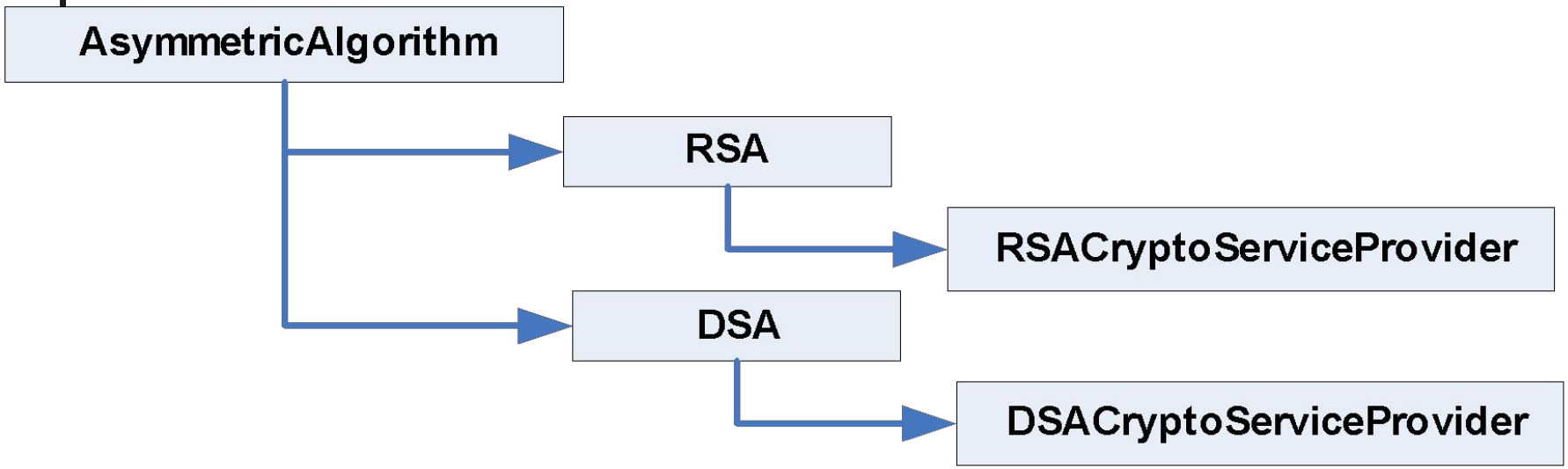
$$M_i = C_i^3 (mod\ 33)$$

  - Получаем

$$M_1 = 9^3 (mod\ 33) = 729 (mod\ 33) = 3$$
$$M_2 = 1^3 (mod\ 33) = 1 (mod\ 33) = 1$$
$$M_3 = 29^3 (mod\ 33) = 24\ 389 (mod\ 33) = 2$$

  - Восстановлено исходное сообщение «312», т.е. «CAB»

Асимметричная криптография в .NET



- RSA, DSA – абстрактные классы
- RSACryptoServiceProvider, DSACryptoServiceProvider – реализуации асимметричного алгоритма RSA, предоставляемого поставщиком служб шифрования (CSP)

Некоторые методы RSACryptoServiceProvider-1

Публичный метод	Описание
Encrypt	Выполняет зашифрование данных с помощью алгоритма RSA public byte[] Encrypt(byte[] rgb, //массив входных данных bool fOAEP //определяет режим дополнения) •Если fOAEP=false, режим дополнения PKCS#1 v1.5 •Если fOAEP=true, то OAEP (Optimal Asymmetric Encryption Padding) – обеспечивает более качественное с точки зрения безопасности дополнение (для ОС выше WinXP, Win2000)
Decrypt	Выполняет расшифрование данных с помощью алгоритма RSA public byte[] Decrypt(byte[] rgb, bool fOAEP)

Некоторые методы RSACryptoServiceProvider-2

Публичный метод	Описание
ExportParameters	Экспортирует параметры алгоритма RSA в структуру RSAParameters public override RSAParameters ExportParameters(bool includePrivateParameters //выгружать закрытый ключ)
ImportParameters	Импортирует параметры алгоритма RSA из структуры RSAParameters public override void ImportParameters(RSAParameters parameters)

Структура RSAParameters

Поле	Описание
Exponent	Представляет параметр Exponent (K_B) для алгоритма RSA
D	Представляет параметр D (k_B) для алгоритма RSA
P	Представляет параметр P для алгоритма RSA
Q	Представляет параметр Q для алгоритма RSA
Modulus	Представляет параметр N для алгоритма RSA

Пример шифрования по методу RSA в .NET

```
class RSACSPSample{
   static void Main()    {
         ASCIIEncoding ByteConverter = new ASCIIEncoding();
         string dataString = "Data to Encrypt";
         byte[] dataToEncrypt = ByteConverter.GetBytes(dataString);
         byte[] encryptedData;          byte[] decryptedData;
         RSACryptoServiceProvider RSAalg = new RSACryptoServiceProvider();
         Console.WriteLine("Original Data: {0}", dataString);
         encryptedData = RSAalg.Encrypt(dataToEncrypt, false);
         Console.WriteLine("Encrypted Data: {0}", ByteConverter.GetString(encryptedData));
          decryptedData = RSAalg.Decrypt(encryptedData, false);
         Console.WriteLine("Decrypted plaintext: {0}",
                              ByteConverter.GetString(decryptedData));
   }
}
```

Сохранение ключей в формате XML

Методы RSACryptoServiceProvider:

Публичный метод	Описание
ToXmlString	Создает и возвращает строку XML, содержащую ключ текущего объекта RSA public override string ToXmlString(bool includePrivateParameters //выгружать закрытый ключ)
FromXmlString	Инициализирует объект RSA, используя данные ключа из строки XML public override void FromXmlString(string xmlString)

Сохранение ключей в файлы XML

```
RSACryptoServiceProvider rsa =      new RSACryptoServiceProvider();

      StreamWriter writer =       new StreamWriter("PublicPrivateKey.xml");
string publicPrivateKeyXML =  rsa.ToXmlString(true);
writer.Write(publicPrivateKeyXML);
writer.Close();


writer =  new StreamWriter("PublicOnlyKey.xml");
string publicOnlyKeyXML =      rsa.ToXmlString(false);
writer.Write(publicOnlyKeyXML);
writer.Close();
```

Дешифрование с помощью закрытого XML-ключа

```
RSACryptoServiceProvider rsa =      new RSACryptoServiceProvider();

      StreamReader reader =       new StreamReader("PublicPrivateKey.xml");
string publicPrivateKeyXML = reader.ReadToEnd();
rsa.FromXmlString(publicPrivateKeyXML);
reader.Close();

      byte[] plainbytes =       rsa.Decrypt(
                  cipherbytes,
                  false); //fOAEP
ConsoleWriteLine(Encoding.UTF8.GetString(plainbytes));
```