Лекція 11. Криптографія з відкритим ключем та функції хешування.

Основные требования к алгоритмам асимметричного шифрования

Создание алгоритмов асимметричного шифрования является величайшим и, возможно, единственным революционным достижением в истории криптографии.

Алгоритмы шифрования с открытым ключом разрабатывались для того, чтобы решить две наиболее трудные задачи, возникшие при использовании симметричного шифрования.

Первой задачей является распределение ключа. При симметричном шифровании требуется, чтобы обе стороны уже имели общий ключ, который каким-то образом должен быть им заранее передан. Диффи, один из основоположников шифрования с открытым ключом, заметил, что это требование отрицает всю суть криптографии, а именно возможность поддерживать всеобщую секретность при коммуникациях.

Второй задачей является необходимость создания таких механизмов, при использовании которых невозможно было бы подменить кого-либо из участников, т.е. нужна цифровая подпись. При использовании коммуникаций для решения широкого круга задач, например в коммерческих и частных целях, электронные сообщения и документы должны иметь эквивалент подписи, содержащейся в бумажных документах. Необходимо создать метод, при использовании которого все участники будут убеждены, что электронное сообщение было послано конкретным участником. Это более сильное требование, чем аутентификация.

Диффи и Хеллман достигли значительных результатов, предложив способ решения обеих задач, который радикально отличается от всех предыдущих подходов к шифрованию.

Сначала рассмотрим общие черты алгоритмов шифрования с открытым ключом и требования к этим алгоритмам. Определим требования, которым должен соответствовать алгоритм, использующий один ключ для шифрования, другой ключ – для дешифрования, и при этом вычислительно невозможно определить дешифрующий ключ, зная только алгоритм шифрования и шифрующий ключ.

Кроме того, некоторые алгоритмы, например RSA, имеют следующую характеристику: каждый из двух ключей может использоваться как для шифрования, так и для дешифрования.

Сначала рассмотрим алгоритмы, обладающие обеими характеристиками, а затем перейдем к алгоритмам открытого ключа, которые не обладают вторым свойством.

При описании симметричного шифрования и шифрования с открытым ключом будем использовать следующую терминологию. Ключ, используемый в симметричном шифровании, будем называть секретным ключом. Два ключа, используемые при шифровании с открытым ключом, будем называть открытым ключом и закрытым ключом. Закрытый ключ держится в секрете, но называть его будем закрытым ключом, а не секретным, чтобы избежать путаницы с ключом, используемым в симметричном шифровании. Закрытый ключ будем обозначать KR, открытый ключ – KU.

Будем предполагать, что все участники имеют доступ к открытым ключам друг друга, а закрытые ключи создаются локально каждым участником и, следовательно, распределяться не должны.

В любое время участник может изменить свой закрытый ключ и опубликовать составляющий пару открытый ключ, заменив им старый открытый ключ.

Диффи и Хеллман описывают требования, которым должен удовлетворять алгоритм шифрования с открытым ключом.

1. Вычислительно легко создавать пару (открытый ключ KU, закрытый ключ KR).
2. Вычислительно легко, имея открытый ключ и незашифрованное сообщение M, создать соответствующий зашифрованное сообщение:

$$C = E_{KU}[M]$$

3. Вычислительно легко дешифровать сообщение, используя закрытый ключ:

$$M = D_{KR}[C] = D_{KR}[E_{KU}[M]]$$

4. Вычислительно невозможно, зная открытый ключ KU, определить закрытый ключ KR.
5. Вычислительно невозможно, зная открытый ключ KU и зашифрованное сообщение C, восстановить исходное сообщение M.

Можно добавить шестое требование, хотя оно не выполняется для всех алгоритмов с открытым ключом:

6. Шифрующие и дешифрующие функции могут применяться в любом порядке:

$$M = E_{KU}[D_{KR}[M]]$$

Это достаточно сильные требования, которые вводят понятие односторонней функции с люком. Односторонней функцией называется такая функция, у которой каждый аргумент имеет единственное обратное значение, при этом вычислить саму функцию легко, а вычислить обратную функцию трудно.

- $Y = f(X)$ – легко
- $X = f^{-1}(Y)$ – трудно

Обычно "легко" означает, что проблема может быть решена за полиномиальное время от длины входа. Таким образом, если длина входа имеет n битов, то время вычисления функции

пропорционально n^a, где a – фиксированная константа. Таким образом, говорят, что алгоритм принадлежит классу полиномиальных алгоритмов P. Термин "трудно" означает более сложное понятие. В общем случае будем считать, что проблему решить невозможно, если усилия для ее решения больше полиномиального времени от величины входа. Например, если длина входа n битов, и время вычисления функции пропорционально 2^n, то это считается вычислительно невозможной задачей. К сожалению, тяжело определить, проявляет ли конкретный алгоритм такую сложность. Более того, традиционные представления о вычислительной сложности фокусируются на худшем случае или на среднем случае сложности алгоритма. Это неприемлемо для криптографии, где требуется невозможность инвертировать функцию для всех или почти всех значений входов.

Вернемся к определению односторонней функции с люком, которую, подобно односторонней функции, легко вычислить в одном направлении и трудно вычислить в обратном направлении до тех пор, пока недоступна некоторая дополнительная информация. При наличии этой дополнительной информации инверсию можно вычислить за полиномиальное время. Таким образом, односторонняя функция с люком принадлежит семейству односторонних функций f_k таких, что

- $Y = f_k(X)$ – легко, если k и X известны
- $X = f_k^{-1}(Y)$ – легко, если k и Y известны
- $X = f_k^{-1}(Y)$ – трудно, если Y известно, но k неизвестно

Мы видим, что разработка конкретного алгоритма с открытым ключом зависит от открытия соответствующей односторонней функции с люком.

Криптоанализ алгоритмов с открытым ключом

Как и в случае симметричного шифрования, алгоритм шифрования с открытым ключом уязвим для лобовой атаки. Контрмера стандартная: использовать большие ключи.

Криптосистема с открытым ключом применяет определенные неинвертируемые математические функции. Сложность вычислений таких функций не является линейной от количества битов ключа, а возрастает быстрее, чем ключ. Таким образом, размер ключа должен быть достаточно большим, чтобы сделать лобовую атаку непрактичной, и достаточно маленьким для возможности практического шифрования. На практике размер ключа делают таким, чтобы лобовая атака была непрактичной, но в результате скорость шифрования оказывается достаточно медленной для использования алгоритма в общих целях. Поэтому шифрование с открытым ключом в настоящее время в основном ограничивается приложениями управления ключом и подписи, в которых требуется шифрование небольшого блока данных.

Другая форма атаки состоит в том, чтобы найти способ вычисления закрытого ключа, зная открытый ключ. Невозможно математически доказать, что данная форма атаки исключена для конкретного алгоритма открытого ключа. Таким образом, любой алгоритм, включая широко используемый алгоритм RSA, является подозрительным.

Наконец, существует форма атаки, специфичная для способов использования систем с открытым ключом. Это атака вероятного сообщения. Предположим, например, что посылаемое сообщение состоит исключительно из 56-битного ключа сессии для алгоритма симметричного шифрования. Противник может зашифровать все возможные ключи, используя открытый ключ, и может дешифровать любое сообщение, соответствующее передаваемому зашифрованному тексту. Таким образом, независимо от размера ключа схемы открытого ключа, атака сводится к лобовой атаке на 56-битный симметричный ключ. Защита от подобной атаки состоит в добавлении определенного количества случайных битов в простые сообщения.

Основные способы использования алгоритмов с открытым ключом

Основными способами использования алгоритмов с открытым ключом являются шифрование/дешифрование, создание и проверка подписи и обмен ключа.

Шифрование с открытым ключом состоит из следующих шагов:

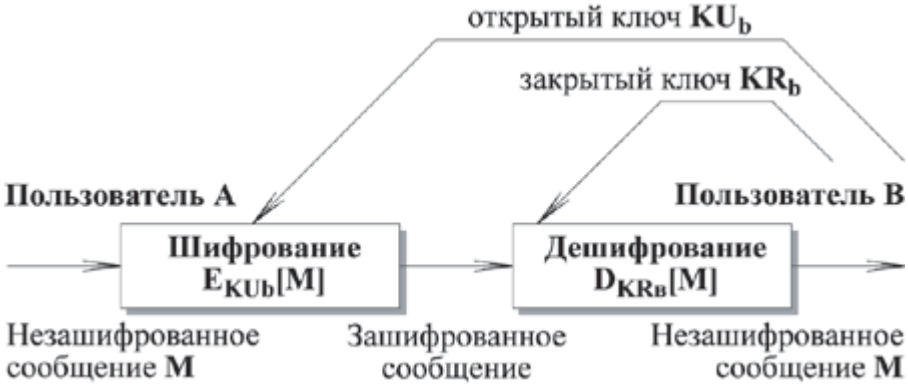

Рис. 7.1. Шифрование с открытым ключом

1. Пользователь *B* создает пару ключей KU_b и KR_b, используемых для шифрования и дешифрования передаваемых сообщений.

2. Пользователь *B* делает доступным некоторым надежным способом свой ключ шифрования, т.е. открытый ключ KU_b. Составляющий пару закрытый ключ KR_b держится в секрете.

3. Если *A* хочет послать сообщение *B*, он шифрует сообщение, используя открытый ключ *B* KU_b.

4. Когда *B* получает сообщение, он дешифрует его, используя свой закрытый ключ KR_b. Никто другой не сможет дешифровать сообщение, так как этот закрытый ключ знает только *B*.

Если пользователь (конечная система) надежно хранит свой закрытый ключ, никто не сможет подсмотреть передаваемые сообщения.

Создание и проверка подписи состоит из следующих шагов:

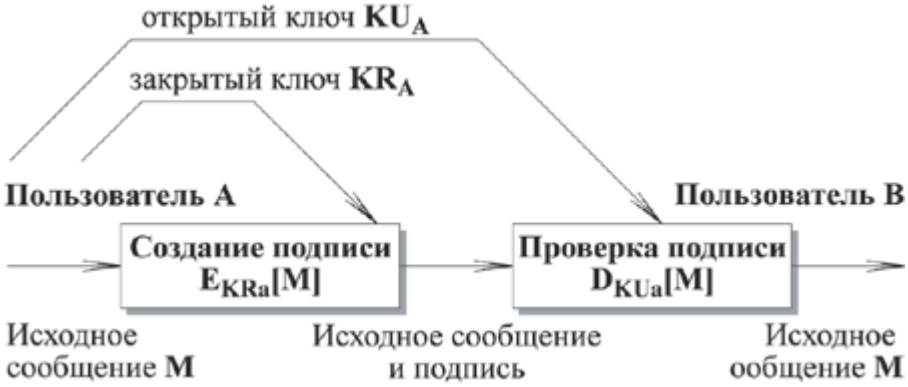

Рис. 7.2. Создание и проверка подписи

1. Пользователь *A* создает пару ключей KR_A и KU_A, используемых для создания и проверки подписи передаваемых сообщений.

2. Пользователь *A* делает доступным некоторым надежным способом свой ключ проверки, т.е. открытый ключ KU_A. Составляющий пару закрытый ключ KR_A держится в секрете.

3. Если А хочет послать подписанное сообщение *B*, он создает подпись $E_{KRa}[M]$ для этого сообщения, используя свой закрытый ключ KR_A.

4. Когда *B* получает подписанное сообщение, он проверяет подпись $D_{KUa}[M]$, используя открытый ключ *A* KU_A. Никто другой не может подписать сообщение, так как этот закрытый ключ знает только *A*.

До тех пор, пока пользователь или прикладная система надежно хранит свой закрытый ключ, их подписи достоверны.

Кроме того, невозможно изменить сообщение, не имея доступа к закрытому ключу *A*; тем самым обеспечивается аутентификация и целостность данных.

В этой схеме все сообщение подписывается, причем для подтверждения целостности сообщения требуется много памяти. Каждое сообщение должно храниться в незашифрованном виде для использования в практических целях. Кроме того, копия сообщения также должна храниться в зашифрованном виде, чтобы можно было проверить в случае необходимости подпись. Более эффективным способом является шифрование небольшого блока битов, который является функцией от сообщения. Такой блок, называемый аутентификатором, должен обладать свойством невозможности

изменения сообщения без изменения аутентификатора. Если аутентификатор зашифрован закрытым ключом отправителя, он является цифровой подписью, с помощью которой можно проверить исходное сообщение. Далее эта технология будет рассматриваться в деталях.

Важно подчеркнуть, что описанный процесс создания подписи не обеспечивает конфиденциальность. Это означает, что сообщение, посланное таким способом, невозможно изменить, но можно подсмотреть. Это очевидно в том случае, если подпись основана на аутентификаторе, так как само сообщение передается в явном виде. Но даже если осуществляется шифрование всего сообщения, конфиденциальность не обеспечивается, так как любой может расшифровать сообщение, используя открытый ключ отправителя.

Обмен ключей: две стороны взаимодействуют для обмена ключом сессии, который в дальнейшем можно использовать в алгоритме симметричного шифрования.

Некоторые алгоритмы можно задействовать тремя способами, в то время как другие могут использоваться одним или двумя способами.

Перечислим наиболее популярные алгоритмы с открытым ключом и возможные способы их применения.

Алгоритм	Шифрование / дешифрование	Цифровая подпись	Обмен ключей
RSA	Да; непригоден для больших блоков	Да	Да
DSS	Нет	Да	Нет
Диффи-Хеллман	Нет	Нет	Да

Алгоритм RSA

Диффи и Хеллман определили новый подход к шифрованию, что вызвало к жизни разработку алгоритмов шифрования, удовлетворяющих требованиям систем с открытым ключом. Одним из первых результатов был алгоритм, разработанный в 1977 году Роном Ривестом, Ади Шамиром и Леном Адлеманом и опубликованный в 1978 году. С тех пор алгоритм Rivest-Shamir-Adleman (RSA) широко применяется практически во всех приложениях, использующих криптографию с открытым ключом.

Алгоритм основан на использовании того факта, что задача факторизации является трудной, т.е. легко перемножить два числа, в то время как не существует полиномиального алгоритма нахождения простых сомножителей большого числа.

Алгоритм RSA представляет собой блочный алгоритм шифрования, где зашифрованные и незашифрованные данные являются целыми между 0 и $n-1$ для некоторого n.

Описание алгоритма

Алгоритм, разработанный Ривестом, Шамиром и Адлеманом, использует выражения с экспонентами. Данные шифруются блоками, каждый блок рассматривается как число, меньшее некоторого числа n. Шифрование и дешифрование имеют следующий вид для некоторого незашифрованного блока M и зашифрованного блока C.

$$C = M^e \bmod n$$

$$M = C^d \bmod n = \left(M^e\right)^d \bmod n = M^{ed} \bmod n$$

Как отправитель, так и получатель должны знать значение n. Отправитель знает значение e, получатель знает значение d. Таким образом, открытый ключ есть $KU = \{e, n\}$ и закрытый ключ есть $KR = \{d, n\}$. При этом должны выполняться следующие условия:

1. Возможность найти значения e, d и n такие, что $M^{ed} = M \bmod n$ для всех $M < n$.
2. Относительная легкость вычисления M^e и C^d для всех значений $M < n$.
3. Невозможность определить d, зная e и n.

Сначала рассмотрим первое условие. Нам необходимо выполнение равенства:

$$M^{ed} = M \bmod n$$

Рассмотрим некоторые математические понятия, свойства и теоремы, которые позволят нам определить e, d и n.

1. Если $(a \cdot b) \equiv (a \cdot c) \bmod n$, то $b \equiv c \bmod n$, если a и n взаимно простые, т.е $\gcd(a, n) = 1$.

2. Обозначим Z_p – все числа, взаимно простые с p и меньшие p. Если p – простое, то Z_p – это все остатки. Обозначим w^{-1} такое число, что $w \cdot w^{-1} \equiv 1 \bmod p$.

Тогда $\forall\ w \in Z_p\ \exists\ z:\ w \cdot z \equiv 1 \bmod p$

Доказательство этого следует из того, что т.к. w и p взаимно простые, то при умножении всех элементов Z_p на w остатками будут все элементы Z_p, возможно, переставленные. Таким образом, хотя бы один остаток будет равен 1.

3. Определим функцию Эйлера следующим образом: $\Phi(n)$ – число положительных чисел, меньших n и взаимно простых с n. Если p – простое, то $\Phi(p) = p - 1$.

Если p и q – простые, то $\Phi(p \cdot q) = (p-1) \cdot (q-1)$.

В этом случае $Z_{p \cdot q} = \{0, 1, \ldots, (p \cdot q - 1)\}$.

Перечислим остатки, которые не являются взаимно простыми с $p \cdot q$:

$\{p, 2 \cdot p, \ldots, (q-1) \cdot p\}$

$\{q, 2 \cdot q, \ldots, (p-1) \cdot q\}$

0

Таким образом $\Phi(p \cdot q) = p \cdot q - [(q-1) + (p-1) + 1] = p \cdot q - (p+q) + 1 = (p-1) \cdot (q-1)$.

4. Теорема Ферма.

$a^{n-1} \equiv 1 \bmod n$, если n – простое и a – положительное целое число, которое не делится на n.

Если все элементы Z_n умножить на a по модулю n, то в результате получим все элементы Z_n, быть может, в другом порядке. Кроме того, $a \cdot 0 \equiv 1 \bmod n$. Рассмотрим следующие $(n-1)$ чисел:

$\{a \bmod n, 2 \cdot a \bmod n, j, (n-1) \cdot a \bmod n\}$, которые являются числами $\{1, 2, j, (n-1)\}$, быть может, в некотором другом порядке. Теперь перемножим по модулю n все эти числа:

$[(a \bmod n) \cdot (2 \cdot a \bmod n) \cdot \ldots \cdot (n-1) \cdot a \bmod n] \bmod n \equiv (n-1)! \bmod n$.

Но $(a) \cdot (2 \cdot a) \cdot \ldots \cdot (n-1) \cdot a \equiv (n-1)! \cdot a^{n-1}$.

Поэтому $(n-1)! \cdot a^{n-1} \equiv (n-1)! \bmod n$

Здесь можно сократить на $(n-1)!$, поскольку n и $(n-1)!$ являются взаимно простыми, следовательно, $a^{n-1} \equiv 1 \bmod n$.

Альтернативная формулировка теоремы: если n является простым и a является положительным целым числом, то

$$a^n \equiv a \bmod n$$

5. Теорема Эйлера.

$a^{\Phi(n)} \equiv 1 \bmod n$ для любых взаимно простых a и n.

Это верно, если n – простое, т.к. в этом случае $\Phi(n) = n - 1$. Рассмотрим множество положительных целых значений, меньших n и взаимно простых с n: $R = \{x_1, x_2, \ldots, x_{\Phi(n)}\}$. Теперь умножим по модулю n каждый элемент этого множества на a. Получим множество $S = \{a \cdot x_1 \bmod n, a \cdot x_2 \bmod n, \ldots, a \cdot x_{\Phi(n)} \bmod n\}$. Это множество является перестановкой множества R по следующим причинам.

- Так как a, как и x_i, является взаимно простым с n, то $a \cdot x_i$ также должно быть взаимно простым с n. Таким образом, S – это множество целых, меньших n и взаимно простых с n.
- В S нет дублей, т.к. если $a \cdot x_i \bmod n = a \cdot x_j \bmod n \Rightarrow x_i = x_j$.

Следовательно, перемножив элементы множеств S и R, получим:

$$\prod_{i=1}^{\Phi(n)}(a\cdot x_i \bmod n)=\prod_{i=1}^{\Phi(n)} x_i ,$$

$$\prod_{i=1}^{\Phi(n)} a\cdot x_i \equiv \prod_{i=1}^{\Phi(n)} x_i \bmod n ,$$

$$a^{\Phi(n)}\cdot\prod_{i=1}^{\Phi(n)} x_i \equiv \prod_{i=1}^{\Phi(n)} x_i \bmod n ,$$

$$a^{\Phi(n)} \equiv 1 \bmod n .$$

Полезной оказывается и следующая альтернативная формулировка теоремы:

$$a^{\Phi(n)+1} \equiv a \bmod n .$$

Из теоремы Эйлера можно вывести следствие, с помощью которого демонстрируется эффективность алгоритма RSA. Для любых двух простых чисел p и q и целых чисел $n=p\cdot q$ и M (здесь $0<M<n$) выполняется следующее условие:

$$M^{\Phi(n)+1}=M^{(p-1)(q-1)+1}\equiv M \bmod n$$

Полезной будет и следующая альтернативная форма того же следствия:

$$\left[M^{\Phi(n)}\right]^k \equiv 1 \bmod n ,$$
$$M^{k\Phi(n)} \equiv 1 \bmod n,$$
$$M^{k\Phi(n)+1}=M^{k(p-1)(q-1)+1}\equiv M \bmod n .$$

Теперь рассмотрим сам **алгоритм RSA**. Пусть p и q – простые.

$n=p\cdot q$.

Таким образом, следует выбрать e и d такие, что $e\cdot d=k\cdot\Phi(n)+1$. Это эквивалентно следующим соотношениям:

$$e\cdot d\equiv 1 \bmod \Phi(n),$$
$$d\equiv e^{-1} \bmod \Phi(n)$$

e и d являются взаимно обратными по умножению по модулю $\Phi(n)$. Заметим, что в соответствии с правилами модульной арифметики, это верно только в том случае, если d (и следовательно, e) являются взаимно простыми с $\Phi(n)$. Таким образом, $\gcd(\Phi(n),d)=1$.

Теперь рассмотрим все **элементы алгоритма RSA**.

p, q – два простых целых числа	– закрыты, выбираемы.
$n=p\cdot q$	– открыто, вычисляемо.
такое e, что $\gcd(\Phi(n),e)=1$; $1<e<\Phi(n)$	– открыто, выбираемо.
$d\equiv e^{-1} \bmod \Phi(n)$	– закрыто, вычисляемо.

Закрытый ключ состоит из $\{d,n\}$, открытый ключ состоит из $\{e,n\}$. Предположим, что пользователь *A* опубликовал свой открытый ключ, и что пользователь *B* хочет послать пользователю *A* сообщение M. Тогда *B* вычисляет $C=M^e \bmod n$ и передает C. При получении этого зашифрованного текста пользователь *A* дешифрует вычислением $M=C^d \bmod n$.

Суммируем алгоритм RSA:

Создание ключей

Выбрать простые p и q

Вычислить $n=p\cdot q$

Вычислить $\Phi(n)=(p-1)\cdot(q-1)$

Выбрать e, $\gcd(\Phi(n),e)=1$ и $1<e<\Phi(n)$

Вычислить d, $d\equiv e^{-1}\bmod\Phi(n)$

Открытый ключ $KU=\{e,n\}$

Закрытый ключ $KR=\{d,n\}$

Шифрование

Незашифрованный текст: $M<n$

Зашифрованный текст: $C=M^e \bmod n$

Дешифрование

Зашифрованный текст: C

Незашифрованный текст: $M=C^d \bmod n$

Рассмотрим конкретный **пример**:

Выбрать два простых числа: $p=7$, $q=17$.

Вычислить $n=p\cdot q=7\cdot 17=119$.

Вычислить $\Phi(n)=(p-1)\cdot(q-1)=6\cdot 16=96$.

Выбрать e, взаимно простое с $\Phi(n)=96$ и меньшее, чем $\Phi(n)$: в данном случае $e=5$.

Определить d так, чтобы $d\cdot e=1\bmod 96$ и $d<96$. Соответствующим значение будет $d=77$, так как $77\cdot 5=385=4\cdot 96+1$

Результирующие ключи открытый $KU=\{5,119\}$ и закрытый $KR=\{77,119\}$.

Например, требуется зашифровать сообщение $M=19$.

При шифровании 19 возводится в пятую степень, что в результате дает 2476099. В результате деления на 119 определяется остаток, равный 66. Следовательно, $19^5\equiv 66\bmod 119$, и поэтому шифрованным текстом будет 66.

Для дешифрования вычисляется $66^{77}=1.27\ldots\cdot 10^{140}$, что после деления на 119 дает $1.06\ldots\cdot 10^{138}$ и остаток 19, т.е. $66^{77}\equiv 19\bmod 119$. Итак, получаем дешифрованный текст – 19.

Вычислительные аспекты

Рассмотрим сложность вычислений в алгоритме RSA при создании ключей и при шифровании/дешифровании.

Шифрование/дешифрование

Как шифрование, так и дешифрование включают возведение целого числа в целую степень по модулю n. При этом промежуточные значения будут громадными. Для того чтобы частично этого избежать, используется следующее свойство модульной арифметики:

$$[(a \bmod n)\cdot(b \bmod n)]\bmod n=(a\cdot b)\bmod n$$

Другая оптимизация состоит в эффективном использовании показателя степени, так как в случае RSA показатели степени очень большие. Предположим, что необходимо вычислить x^{16}. Прямой подход требует 15 умножений. Однако можно добиться того же конечного результата с помощью только четырех умножений, если использовать квадрат каждого промежуточного результата: x^2, x^4, x^8, x^{16}.

Создание ключей

Создание ключей включает следующие задачи:

1. Определить два простых числа p и q.
2. Выбрать e и вычислить d.

Прежде всего, рассмотрим проблемы, связанные с выбором p и q. Так как значение $n = p \cdot q$ будет известно любому потенциальному противнику, для предотвращения раскрытия p и q эти простые числа должны быть выбраны из достаточно большого множества, т.е. p и q должны быть большими числами. С другой стороны, метод, используемый для поиска большого простого числа, должен быть достаточно эффективным.

В настоящее время неизвестны алгоритмы, которые создают произвольно большие простые числа. Процедура, которая используется для этого, выбирает случайное нечетное число из требуемого диапазона и проверяет, является ли оно простым. Если число не является простым, то опять выбирается случайное число до тех пор, пока не будет найдено простое.

Были разработаны различные тесты для определения того, является ли число простым. Это тесты вероятностные, то есть тест показывает, что данное число вероятно является простым. Несмотря на это они могут выполняться таким образом, что сделают вероятность близкой к 1. Если n "проваливает" тест, то оно не является простым. Если n "пропускает" тест, то n может как быть, так и не быть простым. Если n пропускает много таких тестов, то можно с высокой степенью достоверности сказать, что n является простым. Это достаточно долгая процедура, но она выполняется относительно редко: только при создании новой пары (KU, KR).

На сложность вычислений также влияет то, какое количество чисел будет отвергнуто перед тем, как будет найдено простое число. Результат из теории чисел, известный как теорема простого числа, говорит, что простых чисел, расположенных около n в среднем одно на каждые $\ln(n)$ чисел. Таким образом, в среднем требуется проверить последовательность из $\ln(n)$ целых, прежде чем будет найдено простое число. Так как все четные числа могут быть отвергнуты без проверки, то требуется выполнить приблизительно $\ln(n)/2$ проверок. Например, если простое число ищется в диапазоне величин 2^{200}, то необходимо выполнить около $\ln(2^{200})/2 = 70$ проверок.

Выбрав простые числа p и q, далее следует выбрать значение e так, чтобы $\gcd(\Phi(n), e) = 1$ и вычислить значение d, $d = e^{-1} \bmod \Phi(n)$. Существует единственный алгоритм, называемый расширенным алгоритмом Евклида, который за фиксированное время вычисляет наибольший общий делитель двух целых и если этот общий делитель равен единице, определяет инверсное значение одного по модулю другого. Таким образом, процедура состоит в генерации серии случайных чисел и проверке каждого относительно $\Phi(n)$ до тех пор, пока не будет найдено число, взаимно простое с $\Phi(n)$. Возникает вопрос, как много случайных чисел придется проверить до тех пор, пока не найдется нужное число, которое будет взаимно простым с $\Phi(n)$. Результаты показывают, что вероятность того, что два случайных числа являются взаимно простыми, равна 0.6.

Обсуждение криптоанализа

Можно определить четыре возможных подхода для криптоанализа алгоритма RSA:

1. Лобовая атака: перебрать все возможные закрытые ключи.
2. Разложить n на два простых сомножителя. Это даст возможность вычислить $\Phi(n) = (p-1) \cdot (q-1)$ и $d = e^{-1} \bmod \Phi(n)$.
3. Определить $\Phi(n)$ непосредственно, без начального определения p и q. Это также даст возможность определить $d = e^{-1} \bmod \Phi(n)$.
4. Определить d непосредственно, без начального определения $\Phi(n)$.

Защита от лобовой атаки для RSA и ему подобных алгоритмов состоит в использовании большой длины ключа. Таким образом, чем больше битов в e и d, тем лучше. Однако, так как вычисления необходимы как при создании ключей, так и при шифровании/дешифровании, чем больше размер ключа, тем медленнее работает система.

Большинство дискуссий о криптоанализе RSA фокусируется на задаче разложения n на два простых сомножителя. В настоящее время неизвестны алгоритмы, с помощью которых можно было бы разложить число на два простых множителя для очень больших чисел (т.е. несколько сотен десятичных цифр). Лучший из известных алгоритмов дает результат, пропорциональный:

$$L(n) = e^{\sqrt{\ln(n)\ln(\ln(n))}}$$

Пока не разработаны лучшие алгоритмы разложения числа на простые множители, можно считать, что величина n от 100 до 200 цифр в настоящее время является достаточно безопасной. На современном этапе считается, что число из 100 цифр может быть разложено на множители за время порядка двух недель. Для дорогих конфигураций (т.е. порядка $10 млн) число из 150 цифр может быть разложено приблизительно за год. Разложение числа из 200 цифр находится за пределами

вычислительных возможностей. Например, даже если вычислительный уровень в 10^{12} операций в секунду достижим, что выше возможностей современных технологий, то потребуется свыше 10 лет для разложения на множители числа из 200 цифр с использованием существующих алгоритмов.

Для известных в настоящее время алгоритмов задача определения $\Phi(n)$ по данным e и n, по крайней мере, сопоставима по времени с задачей разложения числа на множители.

Для того чтобы избежать выбора значения n, которое могло бы легко раскладываться на сомножители, на p и q должно быть наложено много дополнительных ограничений:

- p и q должны друг от друга отличаться по длине только несколькими разрядами. Например, оба значения p и q должны быть от 10^{75} до 10^{100}.
- Оба числа $(p-1)$ и $(q-1)$ должны содержать в своих разложениях достаточно большой простой сомножитель.
- $\gcd(p-1, q-1)$ должен быть достаточно малым.

Алгоритм обмена ключа Диффи-Хеллмана

Первая публикация данного алгоритма открытого ключа появилась в статье Диффи и Хеллмана, в которой вводились основные понятия криптографии с открытым ключом и в общих чертах упоминался алгоритм обмена ключа Диффи-Хеллмана.

Цель алгоритма состоит в том, чтобы два участника могли безопасно обменяться ключом, который в дальнейшем может использоваться в каком-либо алгоритме симметричного шифрования. Сам алгоритм Диффи-Хеллмана может применяться только для обмена ключами.

Алгоритм основан на трудности вычислений дискретных логарифмов. Дискретный логарифм определяется следующим образом. Вводится понятие первообразного корня простого числа Q как числа, чьи степени создают все целые от 1 до $Q-1$. Это означает, что если A является первообразным корнем простого числа Q, тогда числа

$$A \bmod Q,\ A^2 \bmod Q,\ \ldots,\ A^{Q-1} \bmod Q$$

являются различными и состоят из целых от 1 до $Q-1$ с некоторыми перестановками. В этом случае для любого целого $B < Q$ и первообразного корня A простого числа Q можно найти единственный показатель степени X, такой, что

$$B = A^X \bmod Q, \text{ где } 0 \le X \le (Q-1)$$

Показатель степени X называется дискретным логарифмом, или индексом B по основанию $A \bmod Q$. Это обозначается как

$$ind_{A,Q}(B).$$

Теперь опишем **алгоритм обмена ключей Диффи-Хеллмана**.

Общеизвестные элементы	
Q	простое число
A	$A < Q$ и A является первообразным корнем Q

Вычисление ключа пользователем A	
Выбор случайного числа X_A (закрытый ключ)	$X_A < Q$
Вычисление числа Y_A (открытый ключ)	$Y_A = A^{X_A} \bmod Q$
Вычисление ключа пользователем B	
Выбор случайного числа X_B (закрытый ключ)	$X_B < Q$
Вычисление числа Y_B (открытый ключ)	$Y_B = A^{X_B} \bmod Q$

Вычисление секретного ключа пользователем A
$K = (Y_B)^{X_A} \bmod Q$
Вычисление секретного ключа пользователем B

$$K = (Y_A)^{X_B} \bmod Q$$

Предполагается, что существуют два известных всем числа: простое число Q и целое A, которое является первообразным корнем Q. Теперь предположим, что пользователи **A** и **B** хотят обменяться ключом для алгоритма симметричного шифрования. Пользователь **A** выбирает случайное число $X_A < Q$ и вычисляет $Y_A = A^{X_A} \bmod Q$. Аналогично пользователь **B** независимо выбирает случайное целое число $X_B < Q$ и вычисляет $Y_B = A^{X_B} \bmod Q$. Каждая сторона держит значение X в секрете и делает значение Y доступным для другой стороны. Теперь пользователь **A** вычисляет ключ как $K = (Y_B)^{X_A} \bmod Q$, и пользователь **B** вычисляет ключ как $K = (Y_A)^{X_B} \bmod Q$. В результате оба получат одно и то же значение:

$$\begin{aligned}
&K = (Y_B)^{X_A} \bmod Q = \\
&= \left(A^{X_B} \bmod Q\right)^{X_A} \bmod Q = \\
&= \left(A^{X_B}\right)^{X_A} \bmod Q = && \text{(по правилам арифметики в классах вычетов)} \\
&= A^{X_B \cdot X_A} \bmod Q = \\
&= \left(A^{X_A}\right)^{X_B} \bmod Q = \\
&= \left(A^{X_A} \bmod Q\right)^{X_B} \bmod Q = \\
&= (Y_A)^{X_B} \bmod Q.
\end{aligned}$$

Таким образом, две стороны обменялись секретным ключом. Так как X_A и X_B являются закрытыми, противник может получить только следующие значения: Q, A, Y_A и Y_B. Для вычисления ключа атакующий должен взломать дискретный логарифм, т.е. например вычислить (для определения ключа пользователя **B**)

$$X_B = ind_{A,Q}(Y_B).$$

После этого он сможет вычислить ключ K точно так же, как это делает пользователь **B**.

Безопасность обмена ключа в алгоритме Диффи-Хеллмана вытекает из того факта, что, хотя относительно легко вычислить экспоненты по модулю простого числа, очень трудно вычислить дискретные логарифмы. Для больших простых чисел задача считается неразрешимой.

Следует заметить, что данный алгоритм уязвим для атак типа "man-in-the-middle". Если противник может осуществить активную атаку, т.е. имеет возможность не только перехватывать сообщения, но и заменять их другими, он может перехватить открытые ключи участников Y_A и Y_B, создать свою пару открытого и закрытого ключа (X_C, Y_C) и послать каждому из участников свой открытый ключ. После этого каждый участник вычислит ключ, который будет общим с противником, а не с другим участником. Если нет контроля целостности, то участники не смогут обнаружить подобную подмену.