Network Working Group
Request for Comments: 2631
Category: Standards Track

E. Rescorla
RTFM Inc.

June 1999

Метод согласования ключей Diffie-Hellman

Diffie-Hellman Key Agreement Method

Статус документа

Этот документ содержит спецификацию протокола Internet, предлагаемого сообществу Internet в качестве стандартного, и служит приглашением к дискуссии в целях совершенствования протокола. Текущее состояние стандартизации и статус протокола можно узнать из документа Internet Official Protocol Standards (STD 1). Документ может распространяться свободно.

Авторские права

Тезисы

Этот документ стандартизует один из вариантов метода Диффи-Хеллмана (Diffie-Hellman или DH), основанный на предварительном стандарте ANSI X9.42, разработанном группой ANSI X9F1. Метод DH представляет собой алгоритм согласования ключей, используемых двумя сторонами для согласования общего секрета. Обеспечивается также алгоритм преобразования общего секрета в произвольный объем ключевого материала. Полученный в результате ключевой материал применяется в качестве симметричного ключа шифрования. Описанный здесь вариант DH требует наличия сертификата у получателя, а отправитель может пользоваться статической (открытый ключ помещается в сертификат) или эфемерной парой ключей.

Оглавление

1. Введение

В работе [DH76] Диффи (Diffie) и Хеллман (Hellman) описали метод, с помощью которого две стороны могут согласовать общий секрет, недоступный для перехвата. Этот секрет можно преобразовать в криптографический ключевой материал для других (симметричных) алгоритмов. Существует множество слегка различающихся вариантов этого метода. Данный документ описывает вариант, основанный на спецификации ANSI X9.42.

1.1. Уровни требований

Ключевые слова **необходимо** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **следует** (SHOULD), **не следует** (SHOULD NOT), **возможно** (MAY) в данном документе должны интерпретироваться в соответствии с [RFC2119].

2. Обзор метода

Согласование ключей по методу DH требует наличия ключевых пар у отправителя и получателя сообщения. Путем комбинирования секретного ключа одной стороны и открытого ключа другой, обе стороны могут рассчитать одинаковое секретное число. Полученное таким образом секретное значение может быть преобразовано в криптографический ключевой материал. Этот материал обычно применяется в качестве ключа шифрования ключей (KEK[1]), служащего для

[1]Key-encryption key.

шифрования (wrap) ключа шифрования содержимого (CEK[1]), который, в свою очередь, применяется для шифрования данных в сообщении.

2.1. Согласование ключей

На первом этапе процесса согласования ключей рассчитывается общее секретное значение, которое обозначим ZZ. При использовании одной и той же пары секретного и открытого ключей инициатор и отвечающий получат одинаковые значения ZZ. Это значение ZZ преобразуется в общий симметричный криптографический ключ. Когда инициатор применяет статическую пару открытого и секретного ключей, введение открытого случайного значения обеспечивает в каждом случае согласования получение разных симметричных ключей.

2.1.1. Генерация ZZ

X9.42 определяет, что общий секрет ZZ генерируется по формуле

```
ZZ = g ^ (xb * xa) mod p
```

Отметим, что стороны выполняют на практике несколько отличающиеся расчеты с одинаковыми результатами

```
ZZ = (yb ^ xa)  mod p = (ya ^ xb)  mod p
```

где ^ означает возведение в степень,

ya - открытый ключ стороны a и

```
ya = g ^ xa mod p
```

yb - открытый ключ стороны b и

```
yb = g ^ xb mod p
```

xa - секретный ключ стороны a

xb - секретный ключ стороны b

p - большое простое число

q - большое простое число

```
g = h^{(p-1)/q} mod p
```

где

h - любое целое число 1 < h < p-1, такое что

```
h{(p-1)/q} mod p > 1
```

(g имеет порядок q mod p; т. е., g^q mod p = 1, если g != 1)

j - большое простое число, такое что p = qj + 1

(см. параграф 2.2, где описаны критерии для ключей и параметров).

В [CMS] ключ получателя идентифицирутся с помощью CMS RecipientIdentifier, указывающего сертификат получателя. Открытый ключ отправителя идентифицирутся полем OriginatorIdentifierOrKey, которое указывает сертификат получателя или просто включает его открытый ключ.

2.1.2. Генерация ключевого материала

X9.42 определяет алгоритм генерации произвольного объема ключевого материала из ZZ. Здесь предлагается другой алгоритм, основанный на X9.42, но добавляющий отдельные необязательные поля и опускающий некоторые поля X9.42.

```
KM = H ( ZZ || OtherInfo)
```

H представляет собой функцию хеширования сообщений SHA-1 [FIPS-180], а ZZ - общий секрет, рассчитанный в соответствии с параграфом 2.1.1. Ведущие нули ZZ сохраняются, поэтому размер ZZ в октетах совпадает с размером p. Например, если p имеет 1024 бита, размер ZZ будет равен 128 байтам. Значение OtherInfo является DER-представлением приведенной ниже структуры.

```
OtherInfo ::= SEQUENCE {
  keyInfo KeySpecificInfo,
  partyAInfo [0] OCTET STRING OPTIONAL,
  suppPubInfo [2] OCTET STRING
}

KeySpecificInfo ::= SEQUENCE {
  algorithm OBJECT IDENTIFIER,
  counter OCTET STRING SIZE (4..4) }
```

Отметим, что в этих определениях ASN.1 используются **явные** теги (в ASN.1 используется **явное** тегирование, если явно не задано **неявное**).

algorithm

Идентификатор объекта ASN.1 (OID) для алгоритма CEK, с которым данный ключ KEK будет использоваться. Отметим, что это **не** AlgorithmIdentifier, а просто **идентификатор объекта**. Параметры не используются.

counter

32-битовое целое число с сетевым порядком байтов. Начальное значение счетчика равно 1 для любого ZZ (шестнадцатеричное представление - 00 00 00 01) и увеличивается на единицу при каждом использовании указанной выше функции генерации ключа для данного KEK.

[1]Content-encryption key.

partyAInfo

Случайная строка, предоставляемая отправителем. В CMS эта строка представляется, как параметр в поле UserKeyingMaterial (в формате OCTET STRING). При наличии поля partyAInfo его размер **должен** быть равен 512 батам.

suppPubInfo

Размер создаваемого ключа KEK (в битах), представленный 32-битовым целым числом с сетевым порядком байтов. Например, для алгоритма 3DES он будет представляться последовательностью байтов 00 00 00 C0.

Для создания KEK генерируется один или множество блоков KM (с соответствующим увеличением счетчика), пока не будет получен достаточный объем материала. Блоки KM объединяются слева направо - KM(counter=1) || KM(counter=2)...

Отметим, что единственным источником энтропии для секрета является этот расчет ZZ. Даже при генерации строки длиннее ZZ эффективное пространство ключей KEK ограничено размером ZZ в дополнение к любым, связанным с безопасностью вопросам, вносимым значениями параметров p и q. Однако, если partyAInfo различается для каждого сообщения, для этих сообщений будут генерироваться и разные ключи KEK. Отметим, что параметр partyAInfo **должен** использоваться в режиме Static-Static и **может** применяться в режиме Ephemeral-Static.

2.1.3. Расчет KEK

Для каждого алгоритма шифрования требуется ключ определенного размера (n). Ключ KEK создается путем отображения n старших (слева) байтов KM на ключ. Для алгоритма 3DES, требующего 192 бита ключевого материала, алгоритм должен применяться дважды — один раз со значением счетчика 1 (для генерации K1', K2' и первых 32 битов K3'), а второй раз - 2 (для генерации оставшихся 32 битов K3'[1]). Затем для K1',K2' и K3' устанавливается четность для генерации 3 ключей DES — K1, K2 и K3. Для алгоритма RC2-128, требующего 128 битов ключевого материала, алгоритм применяется один раз со значением счетчика 1 и 128 битов слева напрямую преобразуются в ключ RC2. Аналогично для RC2-40, которому нужно 40 битов ключевого материала, алгоритм применяется один раз с counter=1 и 40 битов слева используются в качестве ключа.

2.1.4. Размеры ключей

Ниже приведены размеры ключей KEK для некоторых распространенных алгоритмов.

3 ключа 3DES	192 бита
RC2-128	128 битов
RC2-40	40 битов

Эффективный размер ключей RC2 совпадает с реальным размером этих ключей.

2.1.5. Проверка открытого ключа

Ниже приведен алгоритм, который **может** использоваться для проверки полученного открытого ключа y. Ключ считается проверенным, если выполняются оба условия:[2]

1. значение y должно лежать в интервале [2,p-1];
2. должно выполняться условие (y^q mod p) == 1.

Основной целью проверки открытых ключей является предотвращение атаки small subgroup (небольшая подгруппа) [LAW98] на ключевую пару отправителя. В режиме Ephemeral-Static проверка может не требоваться. Дополнительную информацию о проверке открытых ключей можно найти в [P1363].

Отметим, что эта процедура может быть запатентована.

2.1.6. Пример 1

ZZ размером 20 байтов

```
 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13
```

Для шифрования ключей (key wrap) применяется алгоритм 3DES-EDE.

Параметр partyAInfo не используется.

При первом вызове SHA-1 входные данные (ZZ) имеют вид

```
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 ; ZZ

30 1d
   30 13
      06 0b 2a 86 48 86 f7 0d 01 09 10 03 06          ; OID алгоритма 3DES
      04 04
         00 00 00 01                                  ; счетчик
   a2 06
      04 04
      00 00 00 c0                                     ; размер ключа
```

На выходе будет 20-байтовая последовательность

```
a0 96 61 39 23 76 f7 04 4d 90 52 a3 97 88 32 46 b6 7f 5f 1e
```

При втором вызове SHA-1 входные данные (ZZ) имеют вид

```
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 ; ZZ
30 1d
```

[1] В оригинале ошибочно указано K3. *Прим. перев.*

[2] В оригинале алгоритм проверки описан не вполне однозначно. *Прим. перев.*

```
      30 13
         06 0b 2a 86 48 86 f7 0d 01 09 10 03 06          ; OID алгоритма 3DES
         04 04
            00 00 00 02                                  ; счетчик
      a2 06
         04 04
         00 00 00 c0                                     ; размер ключа
```

На выходе будет 20-байтовая последовательность

```
   f6 3e b5 fb 5f 56 d9 b6 a8 34 03 91 c2 d3 45 34 93 2e 11 30
```

В результате получим

```
   K1'=a0 96 61 39 23 76 f7 04
   K2'=4d 90 52 a3 97 88 32 46
   K3'=b6 7f 5f 1e f6 3e b5 fb
```

Примечание. Четность для этих ключей не устанавливалась.

2.1.7. Пример 2

ZZ размером 20 байтов

```
    00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13
```

Для шифрования ключей (key wrap) применяется алгоритм RC2-128, которому нужно 128 битов (16 байтов) ключевого материала.

Параметр partyAInfo имеет размер 64 байта

```
   01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 01
   01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 01
   01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 01
   01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 01
```

При вызове SHA-1 входные данные (ZZ) имеют вид

```
   00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 ; ZZ
   30 61
      30 13
         06 0b 2a 86 48 86 f7 0d 01 09 10 03 07          ; OID алгоритма RC2
         04 04
            00 00 00 01                                  ; счетчик
      a0 42
         04 40
            01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 01 ; partyAInfo
            01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 01
            01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 01
            01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 01
      a2 06
         04 04
            00 00 00 80                                  ; размер ключа
```

На выходе будет 20-байтовая последовательность

```
   48 95 0c 46 e0 53 00 75 40 3c ce 72 88 96 04 e0 3e 7b 5d e9
```

В результате получим

```
   K=48 95 0c 46 e0 53 00 75 40 3c ce 72 88 96 04 e0
```

2.2. Требования к ключу и параметрам

X9.42 требует, чтобы группа параметров была связана уравнением p=jq + 1, где q - большое простое число размера m, а j >= 2. Алгоритм генерации простых чисел, удовлетворяющих этому уравнению (основан на алгоритмах FIPS PUB 186-1 [FIPS-186] и [X942]), показан в параграфе 2.2.1[1].

X9.42 требует принадлежности секретного ключа интервалу [2, (q - 2)]. Значение x следует выбирать из этого интервала случайным образом. После этого y рассчитывается, как (g^x mod p). В соответствии с требованиями этого документа параметр m **должен** быть менее 160 битов[2] (следовательно, и размер q **должен** быть по меньшей мере 160 битов). При использовании симметричных шифров, более строгих по сравнению с DES, могут быть желательные большие значения m[2]. Размер параметра p должен быть не менее 512 битов.

2.2.1. Генерация группы параметров

Агентам **следует** генерировать параметры домена (g,p,q), используя описанный ниже алгоритм, созданный на основе [FIPS-186] и [X942]. Корректность созданных с помощью этого алгоритма параметров можно проверить с помощью алгоритма, описанного в параграфе 2.2.2.

2.2.1.1. Генерация p, q

Этот алгоритм создает пару параметров p и q, где q имеет размер m, а p - размер L.

1. Устанавливается m' = m/160, где знак / означает целочисленное деление с округлением вверх (например, 200/160 = 2).

[1] В оригинале ошибочно указано «Приложение A». *Прим. перев.*

[2] В оригинале ошибочно говорится о размере, а не о значении m, хотя из первого абзаца этого параграфа явно следует иное. *Прим. перев.*

2. Устанавливается L' = L/160.
3. Устанавливается N' = L/1024.
4. Выбирается произвольная битовая строка seed (затравка), размером не менее m.
5. Устанавливается U = 0.
6. В цикле по i от 0 до m' - 1 выполняется расчет

   `U = U + [SHA1(seed + i) XOR SHA1((seed + m' + i) mod 2^{seedlen})] * 2^{160 * i}`[1]
   Отметим, что для m = 160, это выражение сводится к алгоритму [FIPS-186]

   `U = [SHA1(seed) XOR SHA1((seed + 1) mod 2^{seedlen})]`
   где seedlen указывает двоичный размер затравки seed.
7. Формируется значение q на основе U путем расчета (U mod (2^m)) и установки для младшего и старшего (2^(m-1)-й) битов результата значения 1. В терминах логических операций q = U OR 2^(m-1) OR 1. Отметим, что 2^(m-1) < q < 2^m[2].
8. С помощью надежного алгоритма проверяется принадлежность q к простым числам.
9. Если q не является простым числом процедура повторяется с п 4[3].
10. Устанавливается counter = 0.
11. Устанавливается R = seed + 2*m' + (L' * counter).
12. Устанавливается V = 0.
13. В цикле по i от 0 до L'-1 выполняется расчет

    `V = V + SHA1(R + i) * 2^(160 * i)`
14. Устанавливается W = V mod 2^L.
15. Устанавливается X = W OR 2^(L-1).

    Отметим, что 0 <= W < 2^(L-1) и, следовательно, X >= 2^(L-1).
16. Устанавливается p = X - (X mod (2*q)) + 1.
17. Если p > 2^(L-1), с помощью надежного алгоритма проверяется принадлежность p к простым числам.
18. Если p является простым числом, возвращаются значения p, q, seed, counter и процедура завершается.
19. Значение счетчика увеличивается на 1 (counter = counter + 1).
20. Если counter < (4096 * N), процедура повторяется с п 11[4].
21. Возвращается «отказ» (failure).

Примечание. К надежным относятся алгоритмы проверки простых чисел, которые обеспечивают вероятность ошибки не более 2^-80. Подходящие алгоритмы приведены в [FIPS-186] и [X942].

2.2.1.2. Генерация g

Ниже описан алгоритм генерации значений g, созданный на основе [FIPS-186]).

1. Пусть j = (p - 1)/q.
2. В качестве h выбирается любое целое число из диапазона 1 < h < p — 1, отличающееся от использованных в предыдущих попытках.
3. Устанавливается g = h^j mod p.
4. Если g = 1, процедура повторяется с п. 2.

2.2.2. Проверка группы параметров

Представление ASN.1 для ключей DH в [PKIX] содержит элементы j и validation-Parms, которые **могут** быть использованы получателями ключей для проверки корректности генерации параметров. Возможны две проверки, перечисленные ниже.

1. Проверка выполнения условия p = qj + 1 показывает, соответствуют ли параметры критериям X9.42.
2. Проверка того, что для генерации значений p и q использовалась процедура из Приложения 2 к [FIPS-186] с затравкой seed и значение p было получено при counter = pgenCounter.

Успешное прохождение тестов говорит, что параметры были выбраны случайным образом и не имеют специальной формы.

---

[1] В оригинале это и следующее выражение содержали ошибки. См. https://www.rfc-editor.org/errata_search.php?eid=2506. *Прим. перев.*

[2] В оригинале этот пункт имеет номер 5 и далее продолжается ошибочная нумерация до конца списка. *Прим. перев.*

[3] Со сменой затравки. *Прим. перев.*

[4] В оригинале ошибочно указан переход к п. 8 — должен быть указан п. 9 в ошибочной нумерации оригинала или п. 11 в исправленной нумерации перевода. См. также ftp://ftp.iks-jena.de/mitarb/lutz/standards/ansi/X9/x942-05-21-98.pdf (стр. 25). *Прим. перев.*

2.3. Режим «эфемерный-статический»

В режиме Ephemeral-Static получатель имеет статическую (и сертифицированную) пару ключей, а отправитель генерирует новую пару для каждого передаваемого сообщения с использованием originatorKey. Если отправитель генерирует новый ключ для каждого сообщения, общие секреты ZZ также будут различаться для каждого сообщения и параметр partyAInfo **можно** опустить, поскольку он служит исключительно для «развязки» множества ключей KEK, генерируемых с одним набором парных ключей. Однако при использовании одного эфемерного ключа для множества сообщений (например, при кэшировании для повышения производительности) **должны** применяться разные значения partyAInfo для каждого сообщения. Все реализации данного стандарта **должны** поддерживать режим Ephemeral-Static.

Для обеспечения устойчивости к атакам small subgroup получателю **следует** выполнять проверку, описанную в параграфе 2.1.5. если друга сторона не может определить результат (успех или неудача) операции расшифровки у получателя, последний **может** опустить такую проверку. Метод генерации ключей, не подверженный атакам small subgroup, описан в [LL97].

2.4. Режим «статический-статический»

В режиме Static-Static отправитель и получатель имеют статические (и сертифицированные) пары ключей. Поскольку в этом случае ключи отправителя и получателя совпадают для каждого сообщения, значения ZZ также будет одинаковы для всех сообщений. Поэтому параметр partyAInfo **должен** применяться (и различаться для каждого сообщения) для того, чтобы обеспечить использование разных ключей KEK для разных сообщений. Реализации **могут** поддерживать режим Static-Static.

Для предотвращения атак small subgroup отправителю и получателю **следует** выполнять проверку, описанную в параграфе 2.1.5, или проверять достоверность ключа в удостоверяющем центре CA. Метод генерации ключей, не подверженный атакам small subgroup, описан в [LL97].

Благодарности

Описанный здесь метод согласования ключей (Key Agreement) основан на результатах работы группы ANSI X9F1. Автор выражает им свою признательность.

Автор также хочет поблагодарить Stephen Henson, Paul Hoffman, Russ Housley, Burt Kaliski, Brian Korver, John Linn, Jim Schaad, Mark Schertler, Peter Yee и Robert Zuccherato за консультации и отзывы.

Литература

[CMS] Housley, R., "Cryptographic Message Syntax", RFC 2630[1], June 1999.

[FIPS-46-1] Federal Information Processing Standards Publication (FIPS PUB) 46-1, Data Encryption Standard, Reaffirmed 1988 January 22 (supersedes FIPS PUB 46, 1977 January 15).

[FIPS-81] Federal Information Processing Standards Publication (FIPS PUB) 81, DES Modes of Operation, 1980 December 2.

[FIPS-180] Federal Information Processing Standards Publication (FIPS PUB) 180-1, "Secure Hash Standard", 1995 April 17.

[FIPS-186] Federal Information Processing Standards Publication (FIPS PUB) 186, "Digital Signature Standard", 1994 May 19.

[P1363] "Standard Specifications for Public Key Cryptography", IEEE P1363 working group draft, 1998, Annex D.

[PKIX] Housley, R., Ford, W., Polk, W. and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile", RFC 2459, January 1999.

[LAW98] L. Law, A. Menezes, M. Qu, J. Solinas and S. Vanstone, "An efficient protocol for authenticated key agreement", Technical report CORR 98-05, University of Waterloo, 1998.

[LL97] C.H. Lim and P.J. Lee, "A key recovery attack on discrete log-based schemes using a prime order subgroup", B.S. Kaliski, Jr., editor, Advances in Cryptology — Crypto '97, Lecture Notes in Computer Science, vol. 1295, 1997, Springer-Verlag, pp. 249-263.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[X942] "Agreement Of Symmetric Keys Using Diffie-Hellman and MQV Algorithms", ANSI draft, 1998.

Вопросы безопасности

Вся защита предлагаемой системы основана на секретности ключевого материала. При утечке секретного ключа отправителя или получателя все сообщения, переданные или принятые с использованием этого ключа, подвергаются риску раскрытия. Потеря (утрата) секретного ключа ведет к невозможности расшифровки соответствующих сообщений.

Статические ключи Diffie-Hellman уязвимы для атак на небольшие подгруппы (small subgroup), описанных в [LAW98]. На практике проблемы могут возникать на обеих сторонах для режима Static-Static и на стороне получателя в режиме Ephemeral-Static. В параграфах 2.3 и 2.4 указаны меры по предотвращению таких атак. Дополнительную защиту может обеспечить генерация ключей с использованием устойчивого к таким атакам метода, как описано в [LL97].

Уровень защиты, обеспечиваемой этими методами, зависит от нескольких факторов. Защита зависит от размера симметричных ключей (уровень защиты 2^l для ключа размером l битов), размера простого числа q (уровень 2^{m/2}) и простого числа p (уровень защиты экспоненциально увеличивается с ростом размера числа в битах). Хорошим практическим решением будет сбалансированная система, где все три упомянутых уровня защиты будут иметь близкие значения. Если из одной пары простых чисел p и q создается множество ключей, имеет смысл повысить уровень защиты этих чисел. В любом случае общий уровень защиты определяется слабейшим из трех упомянутых.

[1] Этот документ признан устаревшим и заменен RFC 3369 (заменен RFC 3852) и RFC 3370. *Прим. перев.*

Адрес автора

Eric Rescorla

RTFM Inc.

30 Newell Road, #16

East Palo Alto, CA 94303

EMail: ekr@rtfm.com

Перевод на русский язык

Николай Малых

nmalykh@gmail.com

Полное заявление авторских прав

Подтверждение

Финансирование функций RFC Editor обеспечено Internet Society.