Internet Research Task Force (IRTF)
Request for Comments: 8032
Category: Informational
ISSN: 2070-1721

S. Josefsson
SJD AB
I. Liusvaara
Independent

January 2017

Edwards-Curve Digital Signature Algorithm (EdDSA)

Алгоритм цифровой подписи на основе кривой Эдвардса

Аннотация

В этом документе описана схема подписи на основе алгоритма цифровой подписи по кривым Эдвардса (Edwards-curve Digital Signature Algorithm или EdDSA). Алгоритм задан с использованием рекомендуемых параметров для кривых edwards25519 и edwards448. Представлены примеры реализации и тестовые векторы.

Статус документа

Документ не относится к категории Internet Standards Track и публикуется с информационными целями.

Документ является результатом работы IRTF[1]. IRTF публикует результаты исследований и разработок, связанных с Internet. Эти результаты могут оказаться не подходящими для внедрения. В данном RFC представлена согласованная точка зрения исследовательской группы Crypto Forum в составе IRTF. Документ был одобрен для публикации IRSG[2] и не претендует на статус стандарта Internet (см. раздел 2 в RFC 7841).

Информацию о текущем статусе документа, ошибках и способах обратной связи можно найти по ссылке http://www.rfc-editor.org/info/rfc8032.

Оглавление

[1] Internet Research Task Force - комиссия по исследованиям Internet.
[2] Internet Research Steering Group - руководящая группа по исследовательским разработкам Internet.

1. Введение

Алгоритм цифровой подписи по кривой Эдвардса (EdDSA) является вариантом системы подписи Шнорра (Schnorr) с кривыми Эдвардса (возможно, скрученными). Для работы EdDSA требуется задать ряд параметров и в этом документе описаны некоторые рекомендуемые варианты.

Для упрощения внедрения EdDSA в сообществе Internet этот документ описывает схему подписи в ориентированной на реализацию форме и содержит пример кода, а также тестовые векторы.

Ниже указаны преимущества EdDSA.

1. Высокая производительность на различных платформах.
2. Не требуется использовать уникальное случайное число для каждой подписи.
3. Более высокая стойкость к атакам по побочным каналам.
4. Небольшой размер открытых ключей (32 или 57 байтов) и подписей (64 или 114 байтов) для Ed25519 и Ed448, соответственно.
5. «Полнота» формул, т. е. пригодность для всех точек кривой без исключений. Это избавляет EdDSA от дорогостоящей проверки точек по недоверенным публичным значениям.
6. Устойчивость к коллизиям, означающая, что конфликты хэш-значений не нарушают работу системы (справедливо лишь для PureEdDSA).

Дополнительные сведения приведены в статье [EDDSA] и обобщённой версии «EdDSA for more curves» [EDDSA2], а в RFC 7748 [RFC7748] обсуждаются конкретные кривые, включая Curve25519 [CURVE25519] и Ed448-Goldilocks [ED448].

Кривая Ed25519 предназначена для работы на уровне безопасности около 128 битов, а Ed448 - 224 битов. Достаточно большой квантовый компьютер мог бы взломать оба варианта. Исходя из разумных оценок возможностей классических компьютеров, можно считать Ed25519 абсолютно надёжным вариантом. Ed448 предназначается для приложений со меньшими требованиями к производительности, где нужна страховка от аналитических атак на эллиптические кривые.

2. Обозначения и соглашения

p
: Простое число, задающее базовое поле.

GF(p)
: Конечное поле с p элементами.

x^y
: x, умноженное на себя y раз (x в степени y).

B
: Генератор рассматриваемой группы или подгруппы.

[n]X
: X, сложенное с собой n раз (X, умноженное на n).

h[i]
: i-й октет строки октетов.

h_i
: i-й бит h.

a || b
: Конкатенация битовых строк a и b.

a <= b
: a меньше или равно b (не больше).

a >= b
: a больше или равно b (не меньше).

i+j
: Сумма i и j.

i*j
: Произведение i и j.

i-j
: Вычитание j из i (разность).

i/j
: Деление i на j (частное).

i x j
: Декартово произведение i и j.

(u,v)
Точка эллиптической кривой с x-координатой u и y-координатой v.

SHAKE256(x, y)
y первых октетов вывода SHAKE256 [FIPS202] для входного значения x.

OCTET(x)
Октет со значением x.

OLEN(x)
Число октетов в строке x.

dom2(x, y)
Пустая строка октетов при проверке и подписании Ed25519. В остальных случаях строка октетов "SigEd25519 no Ed25519 collisions" || octet(x) || octet(OLEN(y)) || y, где x имеет значение 0-255, а y - строка октетов размером не более 255 октетов. Строка "SigEd25519 no Ed25519 collisions" использует кодировку ASCII (32 октета).

dom4(x, y)
Строка октетов "SigEd448" || octet(x) || octet(OLEN(y)) || y, где x имеет значение 0-255, а y - строка октетов размером не более 255 октетов. Строка "SigEd448" использует кодировку ASCII (8 октетов).

Круглые скобки (т. е. (и)) служат для группировки выражений, чтобы избежать описания порядка выполнения.

Битовые строки преобразуются в строки октетов побитово слева направо с переносом в октет от младшего бита к старшему и переходом к следующему октету при заполнении. Преобразование строки октетов в битовую строку выполняется в обратном порядке. Ниже представлен пример преобразования 16-битовой строки

```
b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12 b13 b14 b15
```

которая преобразуется в два октета x0 и x1 (в указанном порядке) как

```
x0 = b7*128+b6*64+b5*32+b4*16+b3*8+b2*4+b1*2+b0
x1 = b15*128+b14*64+b13*32+b12*16+b11*8+b10*4+b9*2+b8
```

При кодировании little-endian биты располагаются слева направо от младшего к старшему. В сочетании описанным выше преобразованием строки битов в строку октетов это ведёт к кодированию октетов little-endian (если размер не кратен 8, старшие биты последнего октета не используются).

Ключевые слова **необходимо** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **нужно** (SHALL), **не следует** (SHALL NOT), **следует** (SHOULD), **не нужно** (SHOULD NOT), **рекомендуется** (RECOMMENDED), **не рекомендуется** (NOT RECOMMENDED), **возможно** (MAY), **необязательно** (OPTIONAL) в данном документе интерпретируются в соответствии с [RFC2119].

3. Алгоритм EdDSA

EdDSA представляет собой систему цифровых подписей с 11 параметрами. Базовая система EdDSA не предназначена для непосредственной реализации. Выбор всех параметров имеет решающее значение для безопасной и эффективной работы. Однако можно выбрать определённые параметры EdDSA (например, Ed25519 or Ed448), иногда слегка обобщённые, для создания кода, который подойдёт для Ed25519 и Ed448. Поэтому точное объяснение базового алгоритма EdDSA мало полезно для разработчиков. Здесь приведено краткое описание базового алгоритма для справки и полноты изложения. Определение таких параметров, как n и c, может помочь при объяснении некоторых этапов алгоритма, которые могут оказаться интуитивно непонятными. Описание полностью соответствует [EDDSA2].

EdDSA имеет 11 параметров.

1. Нечётная степень простого числа p. В EdDSA применяется эллиптическая кривая над конечным полем GF(p).
2. Целое число b, для которого 2^(b-1) > p. Открытые ключи EdDSA имеют в точности b битов, а подписи EdDSA - в точности 2*b битов. Рекомендуется выбирать значение b, кратное 8, чтобы открытые ключи и подписи занимали целое число октетов.
3. Применяется (b-1)-битовое кодирование элементов конечного поля GF(p).
4. Криптографическая хэш-функция H даёт на выходе 2*b битов. Рекомендуются консервативные (не имеющие коллизий) хэш-функции, не оказывающие существенного влияния на суммарные издержки EdDSA.
5. Целое число c со значением 2 или 3. Секретные скаляры EdDSA кратны 2^c. Целое число c - это двоичный логарифм так называемого сомножителя (cofactor).
6. Целое число n, такое что c <= n < b. Секретные скаляры EdDSA имеют в точности n + 1 битов, причём старший бит (позиция 2^n) всегда установлен (1), а младший - сброшен (0).
7. Неквадратичный элемент d из GF(p). Обычно рекомендуется принимать его значение близким к 0, что обеспечивает подходящую кривую.
8. Ненулевой квадратичный элемент a из GF(p). Обычно рекомендуется для наилучшей производительности устанавливать a = -1, если p mod 4 = 1, и a = 1 при p mod 4 = 3.
9. Элемент B != (0,1) из множества E = { (x,y) } элементов GF(p) x GF(p), таких, что a * x^2 + y^2 = 1 + d * x^2 * y^2 .
10. Нечётное простое число L такое, что [L]B = 0 и 2^c * L = #E. Число #E (количество точек на кривой) является частью стандартных данных, предоставляемых для эллиптической кривой, или вычисляется как cofactor * order.
11. Функция предварительного хэширования (prehash) PH. PureEdDSA означает EdDSA, где PH - функция тождества, т. е. PH(M) = M. HashEdDSA означает EdDSA, где функция PH имеет короткий вывод, независимо от размера сообщения, например, PH(M) = SHA-512(M).

Точки кривой образуют группу относительно сложения (x3, y3) = (x1, y1) + (x2, y2) с формулами

$$x3 = \frac{x1 * y2 + x2 * y1}{1 + d * x1 * x2 * y1 * y2}, \quad y3 = \frac{y1 * y2 - a * x1 * x2}{1 - d * x1 * x2 * y1 * y2}$$

Нейтральным элементом группы является (0,1).

В отличие от многих кривых, применяемых в криптографии, эти формулы являются «полными» и действительны для всех точек кривой без исключений. В частности, знаменатель отличен от 0 для всех входных точек. Имеются более эффективные формулы, которые также являются полными и используют однородные координаты для предотвращения дорогостоящих операций обращения по модулю p (см. [Faster-ECC] и [Edwards-revisited]).

3.1. Кодирование

Целое число 0 <= S <= L - 1[1] кодируется в формате little-endian как строка из b битов ENC(S).

Элемент (x,y) из E кодируется как строка из b битов, называемая ENC(x,y) и представляющая собой (b-1)-битовое кодирование y в конкатенации с одним битом, имеющим значение 1 при отрицательном x и 0 - в ином случае.

Кодирование GF(p) применяется для определения «отрицательных» элементов GF(p), в частности, x является отрицательным, если (b-1)-битовое представление x лексикографически больше (b-1)-битового представления -x.

3.2. Ключ

Секретный ключ EdDSA это b-битовая строка k. Пусть hash H(k) = (h_0, h_1, ..., h_(2b-1)) определяет целое число s, которое равно 2^n плюс сумма m = 2^i * h_i для всех целочисленных i из c <= i < n, a s определяет кратное A = [s]B. Открытый ключ EdDSA - это ENC(A). Биты h_b, ..., h_(2b-1) далее используются для подписи.

3.3. Подпись

Подпись EdDSA для сообщения M с секретным ключом k определяется как подпись PureEdDSA PH(M). Иными словами EdDSA просто использует PureEdDSA для подписи PH(M).

Подпись PureEdDSA для сообщения M с секретным ключом k - это 2*b-битовая строка ENC(R) || ENC(S). Значения R и S выводятся следующим образом. Сначала определяется значение r = H(h_b || ... || h_(2b-1) || M), интерпретируя 2*b-битовую строку в формате little-endian как целые числа из множества {0, 1, ..., 2^(2*b) - 1}. Тогда R = [r]B, a S = (r + H(ENC(R) || ENC(A) || PH(M)) * s) mod L. Значение s взято из предыдущего параграфа.

3.4. Проверка

Проверка подписи PureEdDSA ENC(R) || ENC(S) для сообщения M с секретным ключом ENC(A) выполняется следующим образом. Входные данные разбираются так, что A и R являются элементами E, a S - элементом множества {0, 1, ..., L-1}. Рассчитывается h = H(ENC(R) || ENC(A) || M) и проверяется групповое уравнение [2^c * S] B = [2^c] R + [2^c * h][2] A в E. Подпись отклоняется при неудачном разборе (включая выход S из диапазона) или при невыполнении группового уравнения.

Проверка EdDSA для сообщения M определяется как проверка PureEdDSA для PH(M).

4. PureEdDSA, HashEdDSA и именование

Одним из параметров EdDSA является функция предварительного хэширования. Это может быть тождество, приводящее к алгоритму PureEdDSA, или стойкая к коллизиям хэш-функция, такая как SHA-512, что даёт алгоритм HashEdDSA. Выбор используемого варианта зависит от того, какое свойство считается более важным - 1) стойкость к коллизиям или 2) однопроходный интерфейс для создания подписей. Стойкость к коллизиям означает, что EdDSA будет безопасным даже при возможности коллизий хэш-функции. Наличие однопроходного интерфейса означает, что для создания подписи нужен лишь один проход по входному сообщению (для PureEdDSA нужны 2 прохода). Многие существующие API, протоколы и среды предполагают, что алгоритму подписи нужен лишь один проход по входным данным, иначе могут возникать проблемы с API или пропускной способностью.

Отметим, что однопроходная проверка в большинстве случаев невозможна, независимо от применяемого алгоритма подписи. Это связано с тем, что в большинстве случаев невозможно обработать сообщение до проверки подписи, которая требует прохода через все сообщение.

В этом документе заданы параметры, приводящие к вариантам Ed25519ph и Ed448ph для HashEdDSA и вариантам Ed25519 и Ed448 для PureEdDSA.

5. Экземпляры EdDSA

В этом разделе представлен базовый алгоритм EdDSA для кривых edwards25519 и edwards448 с вариантами PureEdDSA и HashEdDSA, а также расширение схемы Ed25519 с учётом контекста (всего 5 наборов параметров).

5.1. Ed25519ph, Ed25519ctx, Ed25519

Ed25519 - это экземпляр EdDSA с указанными ниже параметрами.

Таблица 1. Параметры Ed25519.

Параметр	***Значение***
p	p из edwards25519 в [RFC7748] (т. е. 2^255 - 19)
b	256
Кодирование GF(p)	255 битовое кодирование little-endian из {0, 1, ..., p-1}
H(x)	SHA-512(dom2(phflag,context)||x) [RFC6234]
c	Двоичный логарифм cofactor из edwards25519 в [RFC7748] (т. е. 3)
n	254
d	d из edwards25519 в [RFC7748] (т. е. -121665/121666 = 37095705934669439343138083508754565189542113879843219016388785533085940283555)
a	-1

[1] В оригинале ошибочно указано 0 < S < L - 1. См. https://errata.rfc-editor.org/eid5968/. *Прим. перев.*

[2] В оригинале ошибочно указано [2^c * S] B = 2^c * R + [2^c * h]. См. https://errata.rfc-editor.org/eid6348/. *Прим. перев.*

B	(X(P),Y(P)) из edwards25519 в [RFC7748] (т. е. (15112221349535400772501151409588531511454012693041857206046113283949847762202, 46316835694926478169428394003475163141307993866256225615783033603165251855960))
L	order из edwards25519 в [RFC7748] (т. е. 2^252+27742317777372353535851937790883648493)
PH(x)	x (т. е. функция тождества)

Строка dom2(f,c) для Ed25519 пуста, phflag не имеет значения, а context (при наличии) **должен** быть пустым. Это делает схему идентичной опубликованной ранее схеме Ed25519.

Для Ed25519ctx phflag=0, значению context **не следует** быть пустым. Для Ed25519ph phflag=1, а PH = SHA512, т. е. перед созданием подписи Ed25519 на входе выполняется хэширование SHA-512.

Значение context задаёт подписывающая и проверяющая сторона (не более 255 октетов, по умолчанию пустая строка, за исключением вариант Ed25519, где контекста не может быть) и для успешного прохождения проверки значения должны совпадать в каждом октете.

Используемая кривая эквивалентна Curve25519 [CURVE25519] при изменении координат и это означает, что что сложность задачи дискретного алгоритма совпадает со случаем Curve25519.

5.1.1. Арифметические операции по модулю

Рекомендации по эффективной и безопасной реализации арифметических операций по модулю p = 2^255 - 19 приведены в Curve25519 [CURVE25519]. Для обращения (инверсии) по модулю p рекомендуется применять уравнение x^-1 = x^(p-2) (mod p). Инвертирование 0 не должно возникать, поскольку для него требуется ввод недействительных данных, которые можно было обнаружить ранее, иначе это приведёт к ошибке в расчётах.

Для декодирования точки или «декомпрессии» требуется извлекать квадратный корень по модулю p. Его можно вычислить по алгоритму Tonelli-Shanks или для особого случая p = 5 (mod 8). Чтобы найти квадратный корень из a сначала рассчитывается кандидат x = a^((p+3)/8) (mod p). Здесь возможны три варианта:

```
x^2 = a (mod p) - x является квадратным корнем;
x^2 = -a (mod p) - 2^((p-1)/4) * x является квадратным корнем;
a не имеет квадратного корня по модулю p.
```

5.1.2. Кодирование

Все значения кодируются в строки октетов. Для целых чисел применяется порядок битов little-endian, т. е. 32-октетная строка h h[0],...h[31] представляет целое число h[0] + 2^8 * h[1] + ... + 2^248 * h[31].

Точка кривой (x,y) с координатами из диапазона 0 <= x,y < p кодируется следующим образом. Сначала координата y представляется строкой из 32 октетов (little-endian), старший бит последнего октета всегда имеет значение 0. Для формирования представления точки младший бит координаты x копируется в старший бит последнего октета.

5.1.3. Декодирование

Декодирование точки, заданной строкой в 32 октета описано ниже.

1. Сначала строка интерпретируется как целое число в форме little-endian. Бит 255 в этом числе является младшим битом координаты x и обозначается x_0. Координата y восстанавливается простым сбросом этого бита. Если результат >= p, декодирование завершается отказом.
2. Для восстановления координаты x используется уравнение кривой, предполагающее x^2 = (y^2 - 1) / (d y^2 + 1) (mod p). Знаменатель всегда отличен от 0 по модулю p. Пусть u = y^2 - 1 и v = d y^2 + 1. Для расчёта квадратного корня из (u/v) сначала вычисляется кандидат x = (u/v)^((p+3)/8). Это можно сделать с помощью указанного ниже выражения, используя 1 возведение в степень по модулю для инверсии v и квадратного корня.

```
                        (p+3)/8            (p-5)/8
     x = (u/v)      = u (u v)          (mod p)[1]
```

3. Возможны 3 варианта:
    1. если v x^2 = u (mod p), x является квадратным корнем;
    2. если v x^2 = -u (mod p), устанавливается x <-- x * 2^((p-1)/4) и это будет квадратный корень;
    3. в иных случаях квадратного корня по модулю p не существует и декодирование завершается отказом.
4. Бит x_0 служит для выбора нужного квадратного корня. Если x = 0 и x_0 = 1, декодирование завершается отказом. Если x_0 != x mod 2, устанавливается x <-- p - x. Возвращается декодированная точка (x,y).

5.1.4. Сложение точек

Для сложения точек рекомендуется описываемый здесь метод. Точка (x,y) представляется в расширенных однородных координатах (X, Y, Z, T), где x = X/Z, y = Y/Z, x * y = T/Z. Нейтральной точкой является (0,1), что эквивалентно (0, Z, Z, 0) при любых Z, отличных от 0.

Ниже приведены выражения для сложения двух точек (x3,y3) = (x1,y1)+(x2,y2) на скрученных кривых Эдвардса с a=-1, корнем из a и неквадратичным d, как описано в параграфе 3.1 [Edwards-revisited] и в [EFD-TWISTED-ADD]. Выражения полны, т. е. работают для любой пары действительных точек-слагаемых.

```
A = (Y1-X1)*(Y2-X2)
B = (Y1+X1)*(Y2+X2)
C = T1*2*d*T2
D = Z1*2*Z2
E = B-A
F = D-C
G = D+C
H = B+A
```

[1] В оригинале выражение содержало ошибку. См. https://errata.rfc-editor.org/eid5758/. *Прим. перев.*

```
X3 = E*F
Y3 = G*H
T3 = E*H
Z3 = F*G
```

Для удвоения точки (x3,y3) = (x1,y1)+(x1,y1) можно просто указать одинаковые точки в описанном выше сложении (полнота сложения допускает такую замену) и отметить, что четыре операции умножения превращаются в возведение в квадрат. Однако при использовании формул, описанных в параграфе 3.3[1] [Edwards-revisited] и в [EFD-TWISTED-DBL] число операций сокращается.

```
A = X1^2
B = Y1^2
C = 2*Z1^2
H = A+B
E = H-(X1+Y1)^2
G = A-B
F = C+G
X3 = E*F
Y3 = G*H
T3 = E*H
Z3 = F*G
```

5.1.5. Генерация ключа

Секретный ключ имеет 32 октета (256 битов в соответствии с b) криптографически безопасных случайных данных. Случайные числа обсуждаются в [RFC4086].

Генерация 32-байтового открытого ключа описана ниже.

1. Хэшируется 32-байтовый секретный ключ с помощью SHA-512, результат сохраняется в буфере h размером 64 октета. Для генерации открытого ключа применяются только младшие 32 байта.
2. Сбрасываются (0) 3 младших бита первого октета буфера, старший бит последнего октета и устанавливается (1) второй по старшинству бит последнего октета.
3. Содержимое буфера интерпретируется как целое число в формате little-endian, формируя секретный скаляр s. Выполняется скалярное умножение с фиксированным основанием [s]B.
4. Открытый ключ A - это кодированная точка [s]B. Сначала кодируется координата y (0 <= y < p) как строка little-endian из 32 октетов. Старший бит последнего октета всегда имеет значение 0. При кодировании точки [s]B младший бит координаты x копируется в старший бит последнего октета. Результатом будет открытый ключ.

5.1.6. Подпись

Для процесса подписания входными данными служат секретный ключ (строка из 32 октетов) и сообщение M произвольного размера. Для Ed25519ctx и Ed25519ph дополнительно передаётся контекст C размером до 255 октетов и флаг F (0 для Ed25519ctx и 1 для Ed25519ph).

1. Хэшируется секретный ключ (32 октета) с помощью SHA-512, давая в результате дайджест h. Создаётся секретный скаляр s из первой половины h и соответствующий открытый ключ A, как описано в предыдущем параграфе. Пусть prefix обозначает вторую половину дайджеста (h[32],...,h[63]).
2. Рассчитывается значение SHA-512(dom2(F, C) || prefix || PH(M)) для подписываемого сообщения M. 64-октетный дайджест интерпретируется как целое число r в формате little-endian.
3. Вычисляется точка [r]B. Для повышения эффективности сначала сокращается значение r по модулю L, что соответствует порядку группы B. Пусть строка R указывает представление этой точки.
4. Рассчитывается значение SHA512(dom2(F, C) || R || A || PH(M)) и 64-октетный дайджест интерпретируется как целое число k в формате little-endian.
5. Рассчитывается S = (r + k * s) mod L. Для повышения эффективности сначала сокращается k по модулю L.
6. Подпись формируется конкатенацией R (32 октета) и представления S в формате little-endian (32 октета, где три старших бита последнего октета всегда имеют значение 0).

5.1.7. Проверка[2]

1. Для проверки подписи сообщения M с открытым ключом A, F = 0 для Ed25519ctx или 1 для Ed25519ph при использовании Ed25519ctx и Ed25519ph с контекстом C подпись сначала делится на две части по 32 октета. Первая часть декодируется как точка R', вторая - как целое число S (0 <= S < L). Открытый ключ A декодируется как точка A'. При отказе в любой из операций декодирования (включая выход S из диапазона) подпись считается недействительной.
2. Рассчитывается значение SHA512(dom2(F, C) || R || A || PH(M)), интерпретируемое как 64-октетное целое число k в формате little-endian.
3. Проверяется групповое уравнение [8][S]B = [8]R' + [8][k]A'. Взамен можно (но не требуется) проверить [S]B = R' + [k]A'.

5.2. Ed448ph и Ed448

Ed448 - это экземпляр EdDSA с указанными ниже параметрами.

[1] В оригинале ошибочно указан параграф 3.2. См. https://errata.rfc-editor.org/eid8197/. *Прим. перев.*
[2] В оригинале этот параграф содержал ряд ошибок. См. https://errata.rfc-editor.org/eid6306/. *Прим. перев.*

Таблица 2. Параметры Ed448.

Параметр	*Значение*
p	p из edwards448 в [RFC7748] (т. е. 2^448 - 2^224 - 1)
b	456
Кодирование GF(p)	455-битовое кодирование little-endian из {0, 1, ..., p-1}
H(x)	SHAKE256(dom4(phflag,context)||x, 114)
phflag	0
c	Двоичный логарифм cofactor из edwards448 в [RFC7748] (т. е. 2)
n	447
d	d из edwards448 в [RFC7748] (т. е. -39081)
a	1
B	(X(P),Y(P)) из edwards448 в [RFC7748] (т. е. (224580040295924300187604334099896036246789641632564134246125461686950415467406032909029192869357953282578032075146446173674602635247 71, 298819210078481492676017930443930673437544040154080242095928241372331506189835876003536878655418784733982303233503462500531545062832660))
L	order из edwards448 в [RFC7748] (т. е. 2^446 - 138180668098951153520073867485154268803366924748821786098945475038 85).
PH(x)	x (т. е. функция тождества)

Ed448ph отличается применением в качестве PH функции SHAKE256(x, 64) и phflag = 1, т. е. хэшированием входных данных перед созданием подписи Ed448 с изменённой константой хэширования.

Значение контекста устанавливает подписывающая и проверяющая сторона (до 255 октетов, по умолчанию пустая строка) и они должны совпадать в каждом октете для успешного прохождения проверки.

Кривая эквивалентна Ed448-Goldilocks при изменении базовой точки, что сохраняет сложность задачи дискретного логарифма.

5.2.1. Арифметические операции по модулю

Рекомендации по эффективной и безопасной реализации арифметических операций по модулю p = 2^448 - 2^224 - 1 приведены в [ED448]. Для обращения (инверсии) по модулю p рекомендуется применять уравнение x^-1 = x^(p-2) (mod p). Инвертирование 0 не должно возникать, поскольку для него требуется ввод недействительных данных, которые можно было обнаружить ранее, иначе это приведёт к ошибке в расчётах.

Для декодирования точки или «декомпрессии» требуется извлекать квадратный корень по модулю p. Его можно вычислить, рассчитав сначала кандидат x = a ^ (p+1)/4 (mod p), а затем проверяя условие x^2 = a. При выполнении условия x будет квадратным корнем, невыполнение говорит об отсутствии корня.

5.2.2. Кодирование

Все значения кодируются в строки октетов. Для целых чисел применяется порядок битов little-endian, т. е. 57-октетная строка h h[0],...h[56] представляет целое число h[0] + 2^8 * h[1] + ... + 2^448 * h[56].

Точка кривой (x,y) с координатами из диапазона 0 <= x,y < p кодируется следующим образом. Сначала координата y представляется строкой из 57 октетов (little-endian), последний октет всегда имеет значение 0. Для формирования представления точки младший бит координаты x копируется в старший бит последнего октета.

5.2.3. Декодирование

Декодирование точки, заданной строкой в 57 октетов описано ниже.

1. Сначала строка интерпретируется как целое число в форме little-endian. Бит 455 в этом числе является младшим битом координаты x и обозначается x_0. Координата y восстанавливается простым сбросом этого бита. Если результат >= p, декодирование завершается отказом.
2. Для восстановления координаты x используется уравнение кривой, предполагающее x^2 = (y^2 - 1) / (d y^2 + 1) (mod p). Знаменатель всегда отличен от 0 по модулю p. Пусть u = y^2 - 1 и v = d y^2 - 1. Для расчёта квадратного корня из (u/v) сначала вычисляется кандидат x = (u/v)^((p+1)/4). Это можно сделать с помощью указанного ниже выражения, используя 1 возведение в степень по модулю для инверсии v и квадратного корня.

```
                        (p+1)/4             (p-3)/4
x = (u/v)         =  u (u v)          (mod p)[1]
```

3. Если v * x^2 = u, восстановленным значением координаты x будет x, иначе корня нет и декодирование завершается отказом.
4. Бит x_0 служит для выбора нужного квадратного корня. Если x = 0 и x_0 = 1, декодирование завершается отказом. Если x_0 != x mod 2, устанавливается x <-- p - x. Возвращается декодированная точка (x,y).

5.2.4. Сложение точек

Для сложения точек рекомендуется описываемый здесь метод. Точка (x,y) представляется в проективных координатах (X, Y, Z), где x = X/Z, y = Y/Z. Нейтральной точкой является (0,1), что эквивалентно (0, Z, Z) при любых Z, отличных от 0.

Ниже показано сложение двух точек (x3,y3) = (x1,y1)+(x2,y2) на скрученной кривой Эдвардса (т. е. a=1) с неквадратичным значением d, как описано в разделе 4 [Faster-ECC] и [EFD-ADD]. Сложение является полным, т. е. работает для любой пары действительных входных точек.

```
A = Z1*Z2
B = A^2
C = X1*X2
D = Y1*Y2
```

[1] В оригинале выражение содержало ошибку. См. https://errata.rfc-editor.org/eid5759/. *Прим. перев.*

```
E = d*C*D
F = B-E
G = B+E
H = (X1+Y1)*(X2+Y2)
X3 = A*F*(H-C-D)
Y3 = A*G*(D-C)
Z3 = F*G
```

Подобно описанной выше кривой при удвоении можно подставить две одинаковые точки, превратив 4 операции умножения в возведение в квадрат. Однако это совсем не оптимально и приведённые ниже формулы из раздела 4 в [Faster-ECC] и [EFD-DBL] избавляют от многократного перемножения.

```
B = (X1+Y1)^2
C = X1^2
D = Y1^2
E = C+D
H = Z1^2
J = E-2*H
X3 = (B-E)*J
Y3 = E*(C-D)
Z3 = E*J
```

5.2.5. Генерация ключа

Секретный ключ имеет 57 октетов (456 битов в соответствии с b) криптографически безопасных случайных данных. Случайные числа обсуждаются в [RFC4086].

Генерация 57-байтового открытого ключа описана ниже.

1. Хэшируется 57-байтовый секретный ключ с помощью SHAKE256(x, 114), результат сохраняется в буфере h размером 114 октетов. Для генерации открытого ключа применяются только младшие 57 байтов.
2. Сбрасываются (0) 2 младших бита первого октета буфера, все биты последнего октета и устанавливается (1) второй по старшинству бит последнего октета.
3. Содержимое буфера интерпретируется как целое число в формате little-endian, формируя секретный скаляр s. Выполняется скалярное умножение с фиксированным основанием [s]B.
4. Открытый ключ A - это кодированная точка [s]B. Сначала кодируется координата y (0 <= y < p) как строка little-endian из 57 октетов. Старший бит последнего октета всегда имеет значение 0. При кодировании точки [s]B младший бит координаты x копируется в старший бит последнего октета. Результатом будет открытый ключ.

5.2.6. Подпись

Для процесса подписания входными данными служат секретный ключ (строка из 57 октетов), флаг F (0 для Ed448, 1 для Ed448ph), контекст C размером до 255 октетов и сообщение M произвольного размера

1. Хэшируется секретный ключ (57 октетов) с помощью SHAKE256(x, 114), давая в результате дайджест h. Создаётся секретный скаляр s из первой половины h и соответствующий открытый ключ A, как описано в предыдущем параграфе. Пусть prefix обозначает вторую половину дайджеста (h[57],...,h[113]).
2. Рассчитывается значение SHAKE256(dom4(F, C) || prefix || PH(M), 114) для подписываемого сообщения M. F = 1 для Ed448ph, 0 для Ed448, C - используемый контекст. 114-октетный дайджест интерпретируется как целое число r в формате little-endian.
3. Вычисляется точка [r]B. Для повышения эффективности сначала сокращается значение r по модулю L, что соответствует порядку группы B. Пусть строка R указывает представление этой точки.
4. Рассчитывается значение SHAKE256(dom4(F, C) || R || A || PH(M), 114) и 114-октетный дайджест интерпретируется как целое число k в формате little-endian.
5. Рассчитывается S = (r + k * s) mod L. Для повышения эффективности сначала сокращается k по модулю L.
6. Подпись формируется конкатенацией R (57 октетов) и представления S в формате little-endian (57 октетов, где десять старших битов последних октетов всегда имеют значение 0).

5.2.7. Проверка

1. Для проверки подписи сообщения M с использованием контекста C и открытого ключа A, F = 0 для Ed448 или 1 для Ed448ph подпись сначала делится на две части по 57 октетов. Первая часть декодируется как точка R, вторая - как целое число S (0 <= S < L). Открытый ключ A декодируется как точка A'. При отказе в любой из операций декодирования (включая выход S из диапазона) подпись считается недействительной.
2. Рассчитывается значение SHAKE256(dom4(F, C) || R || A || PH(M), 114), интерпретируемое как 114-октетное целое число k в формате little-endian.
3. Проверяется групповое уравнение [4][S]B = [4]R + [4][k]A'. Взамен можно (но не требуется) проверить [S]B = R + [k]A'.

6. Реализация Ed25519 на языке Python

В этом разделе показан пример реализации Ed25519 на языке Python версии 3.2 или выше. В Приложении A дана полная реализация, а в Приложении B - тестовый драйвер для запуска с некоторыми тестовыми векторами. Приведённый здесь код не предназначен для продуктивного использования, поскольку не проверялась его корректность при всех входных данных и не обеспечивается защита от утечек по побочным каналам.

```
## Некоторые требуемые примитивы
import hashlib

def sha512(s)
    return hashlib.sha512(s).digest()

# Базовое поле Z_p
p = 2**255 - 19

def modp_inv(x)
    return pow(x, p-2, p)

# Константа кривой
d = -121665 * modp_inv(121666) % p

# Порядок группы
q = 2**252 + 27742317777372353535851937790883648493

def sha512_modq(s)
    return int.from_bytes(sha512(s), "little") % q

## Функции для операций с точками.
# Точки представляются кортежами расширенных координат
# (X, Y, Z, T), где x = X/Z, y = Y/Z, x*y = T/Z

def point_add(P, Q)
    A, B = (P[1]-P[0]) * (Q[1]-Q[0]) % p, (P[1]+P[0]) * (Q[1]+Q[0]) % p;
    C, D = 2 * P[3] * Q[3] * d % p, 2 * P[2] * Q[2] % p;
    E, F, G, H = B-A, D-C, D+C, B+A;
    return (E*F, G*H, F*G, E*H);

# Расчёт Q = s * Q
def point_mul(s, P)
    Q = (0, 1, 1, 0)  # Нейтральный элемент
    while s > 0:
        if s & 1:
            Q = point_add(Q, P)
        P = point_add(P, P)
        s >>= 1
    return Q

def point_equal(P, Q)
    # x1 / z1 == x2 / z2  <==>  x1 * z2 == x2 * z1
    if (P[0] * Q[2] - Q[0] * P[2]) % p != 0:
        return False
    if (P[1] * Q[2] - Q[1] * P[2]) % p != 0:
        return False
    return True

## Функции сжатия точек.
# Квадратный корень из -1
modp_sqrt_m1 = pow(2, (p-1) // 4, p)

# Расчёт соответствующей координаты x со знаком в младшем бите
# или возврат None при отказе.
def recover_x(y, sign)
    if y >= p:
        return None
    x2 = (y*y-1) * modp_inv(d*y*y+1)
    if x2 == 0:
        if sign:
            return None
        else:
            return 0

    # Расчёт квадратного корня из x2
    x = pow(x2, (p+3) // 8, p)
    if (x*x - x2) % p != 0:
        x = x * modp_sqrt_m1 % p
    if (x*x - x2) % p != 0:
        return None

    if (x & 1) != sign:
        x = p - x
    return x

# Базовая точка
g_y = 4 * modp_inv(5) % p
g_x = recover_x(g_y, 0)
G = (g_x, g_y, 1, g_x * g_y % p)

def point_compress(P)
    zinv = modp_inv(P[2])
    x = P[0] * zinv % p
    y = P[1] * zinv % p
    return int.to_bytes(y | ((x & 1) << 255), 32, "little")
```

```
def point_decompress(s)
    if len(s) != 32:
        raise Exception("Invalid input length for decompression")
    y = int.from_bytes(s, "little")
    sign = y >> 255
    y &= (1 << 255) - 1

    x = recover_x(y, sign)
    if x is None:
        return None
    else:
        return (x, y, 1, x*y % p)

## Функции для операций с секретным ключом.
def secret_expand(secret)
    if len(secret) != 32:
        raise Exception("Bad size of private key")
    h = sha512(secret)
    a = int.from_bytes(h[:32], "little")
    a &= (1 << 254) - 8
    a |= (1 << 254)
    return (a, h[32:])

def secret_to_public(secret)
    (a, dummy) = secret_expand(secret)
    return point_compress(point_mul(a, G))

## Функция подписи.
def sign(secret, msg)
    a, prefix = secret_expand(secret)
    A = point_compress(point_mul(a, G))
    r = sha512_modq(prefix + msg)
    R = point_mul(r, G)
    Rs = point_compress(R)
    h = sha512_modq(Rs + A + msg)
    s = (r + h * a) % q
    return Rs + int.to_bytes(s, 32, "little")

## Финальная проверка функции.
def verify(public, msg, signature)
    if len(public) != 32:
        raise Exception("Bad public key length")
    if len(signature) != 64:
        raise[1] Exception("Bad signature length")
    A = point_decompress(public)
    if not A:
        return False
    Rs = signature[:32]
    R = point_decompress(Rs)
    if not R:
        return False
    s = int.from_bytes(signature[32:], "little")
    if s >= q: return False
    h = sha512_modq(Rs + public + msg)
    sB = point_mul(s, G)
    hA = point_mul(h, A)
    return point_equal(sB, point_add(R, hA))
```

7. Тестовые векторы

В этом разделе приведены тестовые векторы для Ed25519ph, Ed25519ctx, Ed448ph, Ed25519 и Ed448. Каждый параграф включает последовательность тестовых векторов. Октеты указаны в шестнадцатеричном формате с добавлением пробелов для удобочитаемости. Открытые и секретные ключи Ed25519, Ed25519ctx, Ed25519ph имеют размер 32 октета, а подписи - 64 октета. Открытые и секретные ключи Ed448 и Ed448ph имеют размер 57 октетов, подписи - 114 октетов. Сообщения имеют произвольный размер, непустой контекст представлен 1 - 255 октетами.

7.1. Тестовые векторы для Ed25519[2]

Тестовые векторы взяты из [ED25519-TEST-VECTORS] (с удалением открытых ключей как суффиксов секретного ключа и удалением сообщений из подписи) и [ED25519-LIBGCRYPT-TEST-VECTORS]. Тестовые векторы 100-111 взяты из работы «Taming the many EdDSAs» Konstantinos Chalkias, François Garillot и Valeria Nikolaenko https://eprint.iacr.org/2020/1244

```
-----Тест 1
Алгоритм
Ed25519

Секретный ключ
9d61b19deffd5a60ba844af492ec2cc4
4449c5697b326919703bac031cae7f60
```

[1] В оригинале оператор raise ошибочно не указан. См. https://errata.rfc-editor.org/eid5930/. *Прим. перев.*

[2] В оригинале этот параграф существенно отличался. См. https://errata.rfc-editor.org/eid7031/. *Прим. перев.*

```
Открытый ключ
d75a980182b10ab7d54bfed3c964073a
0ee172f3daa62325af021a68f707511a

Сообщение (0 байтов)

Подпись
e5564300c360ac729086e2cc806e828a
84877f1eb8e5d974d873e06522490155
5fb8821590a33bacc61e39701cf9b46b
d25bf5f0595bbe24655141438e7a100b

-----Тест 2
Алгоритм
Ed25519

Секретный ключ
4ccd089b28ff96da9db6c346ec114e0f
5b8a319f35aba624da8cf6ed4fb8a6fb

Открытый ключ
3d4017c3e843895a92b70aa74d1b7ebc
9c982ccf2ec4968cc0cd55f12af4660c

Сообщение (1 байт)
72

Подпись
92a009a9f0d4cab8720e820b5f642540
a2b27b5416503f8fb3762223ebdb69da
085ac1e43e15996e458f3613d0f11d8c
387b2eaeb4302aeeb00d291612bb0c00

-----Тест 3
Алгоритм
Ed25519

Секретный ключ
c5aa8df43f9f837bedb7442f31dcb7b1
66d38535076f094b85ce3a2e0b4458f7

Открытый ключ
fc51cd8e6218a1a38da47ed00230f058
0816ed13ba3303ac5deb911548908025

Сообщение (2 байта)
af82

Подпись
6291d657deec24024827e69c3abe01a3
0ce548a284743a445e3680d7db5ac3ac
18ff9b538d16f290ae67f760984dc659
4a7c15e9716ed28dc027beceea1ec40a

-----Тест 1024
Алгоритм
Ed25519

Секретный ключ
f5e5767cf153319517630f226876b86c
8160cc583bc013744c6bf255f5cc0ee5

Открытый ключ
278117fc144c72340f67d0f2316e8386
ceffbf2b2428c9c51fef7c597f1d426e

Сообщение (1023 байта)
08b8b2b733424243760fe426a4b54908
632110a66c2f6591eabd3345e3e4eb98
fa6e264bf09efe12ee50f8f54e9f77b1
e355f6c50544e23fb1433ddf73be84d8
79de7c0046dc4996d9e773f4bc9efe57
38829adb26c81b37c93a1b270b20329d
658675fc6ea534e0810a4432826bf58c
941efb65d57a338bbd2e26640f89ffbc
1a858efcb8550ee3a5e1998bd177e93a
7363c344fe6b199ee5d02e82d522c4fe
ba15452f80288a821a579116ec6dad2b
3b310da903401aa62100ab5d1a36553e
06203b33890cc9b832f79ef80560ccb9
a39ce767967ed628c6ad573cb116dbef
efd75499da96bd68a8a97b928a8bbc10
3b6621fcde2beca1231d206be6cd9ec7
aff6f6c94fcd7204ed3455c68c83f4a4
1da4af2b74ef5c53f1d8ac70bdcb7ed1
85ce81bd84359d44254d95629e9855a9
```

```
   4a7c1958d1f8ada5d0532ed8a5aa3fb2
   d17ba70eb6248e594e1a2297acbbb39d
   502f1a8c6eb6f1ce22b3de1a1f40cc24
   554119a831a9aad6079cad88425de6bd
   e1a9187ebb6092cf67bf2b13fd65f270
   88d78b7e883c8759d2c4f5c65adb7553
   878ad575f9fad878e80a0c9ba63bcbcc
   2732e69485bbc9c90bfbd62481d9089b
   eccf80cfe2df16a2cf65bd92dd597b07
   07e0917af48bbb75fed413d238f5555a
   7a569d80c3414a8d0859dc65a46128ba
   b27af87a71314f318c782b23ebfe808b
   82b0ce26401d2e22f04d83d1255dc51a
   ddd3b75a2b1ae0784504df543af8969b
   e3ea7082ff7fc9888c144da2af58429e
   c96031dbcad3dad9af0dcbaaaf268cb8
   fcffead94f3c7ca495e056a9b47acdb7
   51fb73e666c6c655ade8297297d07ad1
   ba5e43f1bca32301651339e22904cc8c
   42f58c30c04aafdb038dda0847dd988d
   cda6f3bfd15c4b4c4525004aa06eeff8
   ca61783aacec57fb3d1f92b0fe2fd1a8
   5f6724517b65e614ad6808d6f6ee34df
   f7310fdc82aebfd904b01e1dc54b2927
   094b2db68d6f903b68401adebf5a7e08
   d78ff4ef5d63653a65040cf9bfd4aca7
   984a74d37145986780fc0b16ac451649
   de6188a7dbdf191f64b5fc5e2ab47b57
   f7f7276cd419c17a3ca8e1b939ae49e4
   88acba6b965610b5480109c8b17b80e1
   b7b750dfc7598d5d5011fd2dcc5600a3
   2ef5b52a1ecc820e308aa342721aac09
   43bf6686b64b2579376504ccc493d97e
   6aed3fb0f9cd71a43dd497f01f17c0e2
   cb3797aa2a2f256656168e6c496afc5f
   b93246f6b1116398a346f1a641f3b041
   e989f7914f90cc2c7fff357876e506b5
   0d334ba77c225bc307ba537152f3f161
   0e4eafe595f6d9d90d11faa933a15ef1
   369546868a7f3a45a96768d40fd9d034
   12c091c6315cf4fde7cb68606937380d
   b2eaaa707b4c4185c32eddcdd306705e
   4dc1ffc872eeee475a64dfac86aba41c
   0618983f8741c5ef68d3a101e8a3b8ca
   c60c905c15fc910840b94c00a0b9d0

   Подпись
   0aab4c900501b3e24d7cdf4663326a3a
   87df5e4843b2cbdb67cbf6e460fec350
   aa5371b1508f9f4528ecea23c436d94b
   5e8fcd4f681e30a6ac00a9704a188a03

   -----Тест SHA(abc)
   Алгоритм
   Ed25519

   Секретный ключ
   833fe62409237b9d62ec77587520911e
   9a759cec1d19755b7da901b96dca3d42

   Открытый ключ
   ec172b93ad5e563bf4932c70e1245034
   c35467ef2efd4d64ebf819683467e2bf

   Сообщение (64 байта)
   ddaf35a193617abacc417349ae204131
   12e6fa4e89a97ea20a9eeee64b55d39a
   2192992a274fc1a836ba3c23a3feebbd
   454d4423643ce80e2a9ac94fa54ca49f

   Подпись
   dc2a4459e7369633a52b1bf277839a00
   201009a3efbf3ecb69bea2186c26b589
   09351fc9ac90b3ecfdfbc7c66431e030
   3dca179c138ac17ad9bef1177331a704
   -----

-----Тест 100
Алгоритм
Ed25519

Сообщение
8c93255d71dcab10e8f379c26200f3c7bd5f09d9bc3068d3ef4edeb4853022b6
```

```
Открытый ключ
c7176a703d4dd84fba3c0b760d10670f2a2053fa2c39ccc64ec7fd7792ac03fa

Подпись
c7176a703d4dd84fba3c0b760d10670f2a2053fa2c39ccc64ec7fd7792ac037a
0000000000000000000000000000000000000000000000000000000000000000

-----Тест 101
Алгоритм
Ed25519

Сообщение
9bd9f44f4dcc75bd531b56b2cd280b0bb38fc1cd6d1230e14861d861de092e79

Открытый ключ
c7176a703d4dd84fba3c0b760d10670f2a2053fa2c39ccc64ec7fd7792ac03fa

Подпись
f7badec5b8abeaf699583992219b7b223f1df3fbbea919844e3f7c554a43dd43
a5bb704786be79fc476f91d3f3f89b03984d8068dcf1bb7dfc6637b45450ac04

-----Тест 102
Алгоритм
Ed25519

Сообщение
aebf3f2601a0c8c5d39cc7d8911642f740b78168218da8471772b35f9d35b9ab

Открытый ключ
f7badec5b8abeaf699583992219b7b223f1df3fbbea919844e3f7c554a43dd43

Подпись
c7176a703d4dd84fba3c0b760d10670f2a2053fa2c39ccc64ec7fd7792ac03fa
8c4bd45aecaca5b24fb97bc10ac27ac8751a7dfe1baff8b953ec9f5833ca260e

-----Тест 103
Алгоритм
Ed25519

Сообщение
9bd9f44f4dcc75bd531b56b2cd280b0bb38fc1cd6d1230e14861d861de092e79

Открытый ключ
cdb267ce40c5cd45306fa5d2f29731459387dbf9eb933b7bd5aed9a765b88d4d

Подпись
9046a64750444938de19f227bb80485e92b83fdb4b6506c160484c016cc1852f
87909e14428a7a1d62e9f22f3d3ad7802db02eb2e688b6c52fcd6648a98bd009

-----Тест 104
Алгоритм
Ed25519

Сообщение
e47d62c63f830dc7a6851a0b1f33ae4bb2f507fb6cffec4011eaccd55b53f56c

Открытый ключ
cdb267ce40c5cd45306fa5d2f29731459387dbf9eb933b7bd5aed9a765b88d4d

Подпись
160a1cb0dc9c0258cd0a7d23e94d8fa878bcb1925f2c64246b2dee1796bed512
5ec6bc982a269b723e0668e540911a9a6a58921d6925e434ab10aa7940551a09

-----Тест 105
Алгоритм
Ed25519

Сообщение
e47d62c63f830dc7a6851a0b1f33ae4bb2f507fb6cffec4011eaccd55b53f56c

Открытый ключ
cdb267ce40c5cd45306fa5d2f29731459387dbf9eb933b7bd5aed9a765b88d4d

Подпись
21122a84e0b5fca4052f5b1235c80a537878b38f3142356b2c2384ebad4668b7
e40bc836dac0f71076f9abe3a53f9c03c1ceeeddb658d0030494ace586687405

-----Тест 106
Алгоритм
Ed25519

Сообщение
85e241a07d148b41e47d62c63f830dc7a6851a0b1f33ae4bb2f507fb6cffec40

Открытый ключ
442aad9f089ad9e14647b1ef9099a1ff4798d78589e66f28eca69c11f582a623
```

```
Подпись
e96f66be976d82e60150baecff9906684aebb1ef181f67a7189ac78ea23b6c0e
547f7690a0e2ddcd04d87dbc3490dc19b3b3052f7ff0538cb68afb369ba3a514

-----Тест 107
Алгоритм
Ed25519

Сообщение
85e241a07d148b41e47d62c63f830dc7a6851a0b1f33ae4bb2f507fb6cffec40

Открытый ключ
442aad9f089ad9e14647b1ef9099a1ff4798d78589e66f28eca69c11f582a623

Подпись
8ce5b96c8f26d0ab6c47958c9e68b937104cd36e13c33566acd2fe8d38aa1942
7e71f98a473474f2f13f06f97c20d58cc3f54b8bd0d272f42b695dd7e89a8c22

-----Тест 108
Алгоритм
Ed25519

Сообщение
9bedc267423725d473888631ebf45988bad3db83851ee85c85e241a07d148b41

Открытый ключ
f7badec5b8abeaf699583992219b7b223f1df3fbbea919844e3f7c554a43dd43

Подпись
ecffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
03be9678ac102edcd92b0210bb34d7428d12ffc5df5f37e359941266a4e35f0f

-----Тест 109
Алгоритм
Ed25519

Сообщение
9bedc267423725d473888631ebf45988bad3db83851ee85c85e241a07d148b41

Открытый ключ
f7badec5b8abeaf699583992219b7b223f1df3fbbea919844e3f7c554a43dd43

Подпись
ecffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ca8c5b64cd208982aa38d4936621a4775aa233aa0505711d8fdcfdaa943d4908

-----Тест 110
Алгоритм
Ed25519

Сообщение
e96b7021eb39c1a163b6da4e3093dcd3f21387da4cc4572be588fafae23c155b

Открытый ключ
ecffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff

Подпись
a9d55260f765261eb9b84e106f665e00b867287a761990d7135963ee0a7d59dc
a5bb704786be79fc476f91d3f3f89b03984d8068dcf1bb7dfc6637b45450ac04

-----Тест 111
Алгоритм
Ed25519

Сообщение
39a591f5321bbe07fd5a23dc2f39d025d74526615746727ceefd6e82ae65c06f

Открытый ключ
ecffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff

Подпись
a9d55260f765261eb9b84e106f665e00b867287a761990d7135963ee0a7d59dc
a5bb704786be79fc476f91d3f3f89b03984d8068dcf1bb7dfc6637b45450ac04
   -----
```

7.2. Тестовые векторы для Ed25519ctx

```
   -----foo
   Алгоритм
   Ed25519ctx

   Секретный ключ
   0305334e381af78f141cb666f6199f57
   bc3495335a256a95bd2a55bf546663f6
```

```
Открытый ключ
dfc9425e4f968f7f0c29f0259cf5f9ae
d6851c2bb4ad8bfb860cfee0ab248292

Сообщение (16 байтов)
f726936d19c800494e3fdaff20b276a8

Контекст
666f6f

Подпись
55a4cc2f70a54e04288c5f4cd1e45a7b
b520b36292911876cada7323198dd87a
8b36950b95130022907a7fb7c4e9b2d5
f6cca685a587b4b21f4b888e4e7edb0d

-----bar
Алгоритм
Ed25519ctx

Секретный ключ
0305334e381af78f141cb666f6199f57
bc3495335a256a95bd2a55bf546663f6

Открытый ключ
dfc9425e4f968f7f0c29f0259cf5f9ae
d6851c2bb4ad8bfb860cfee0ab248292

Сообщение (16 байтов)
f726936d19c800494e3fdaff20b276a8

Контекст
626172

Подпись
fc60d5872fc46b3aa69f8b5b4351d580
8f92bcc044606db097abab6dbcb1aee3
216c48e8b3b66431b5b186d1d28f8ee1
5a5ca2df6668346291c2043d4eb3e90d

-----foo2
Алгоритм
Ed25519ctx

Секретный ключ
0305334e381af78f141cb666f6199f57
bc3495335a256a95bd2a55bf546663f6

Открытый ключ
dfc9425e4f968f7f0c29f0259cf5f9ae
d6851c2bb4ad8bfb860cfee0ab248292

Сообщение (16 байтов)
508e9e6882b979fea900f62adceaca35

Контекст
666f6f

Подпись
8b70c1cc8310e1de20ac53ce28ae6e72
07f33c3295e03bb5c0732a1d20dc6490
8922a8b052cf99b7c4fe107a5abb5b2c
4085ae75890d02df26269d8945f84b0b

-----foo3
Алгоритм
Ed25519ctx

Секретный ключ
ab9c2853ce297ddab85c993b3ae14bca
d39b2c682beabc27d6d4eb20711d6560

Открытый ключ
0f1d1274943b91415889152e893d80e9
3275a1fc0b65fd71b4b0dda10ad7d772

Сообщение (16 байтов)
f726936d19c800494e3fdaff20b276a8

Контекст
666f6f

Подпись
21655b5f1aa965996b3f97b3c849eafb
a922a0a62992f73b3d1b73106a84ad85
e9b86a7b6005ea868337ff2d20a7f5fb
```

```
d4cd10b0be49a68da2b2e0dc0ad8960f
-----
```

7.3. Тестовые векторы для Ed25519ph

```
-----Тест abc
Алгоритм
Ed25519ph

Секретный ключ
833fe62409237b9d62ec77587520911e
9a759cec1d19755b7da901b96dca3d42

Открытый ключ
ec172b93ad5e563bf4932c70e1245034
c35467ef2efd4d64ebf819683467e2bf

Сообщение (3 байта)
616263

Подпись
98a70222f0b8121aa9d30f813d683f80
9e462b469c7ff87639499bb94e6dae41
31f85042463c2a355a2003d062adf5aa
a10b8c61e636062aaad11c2a26083406
-----
```

7.4. Тестовые векторы для Ed448

```
-----Blank
Алгоритм
Ed448

Секретный ключ
6c82a562cb808d10d632be89c8513ebf
6c929f34ddfa8c9f63c9960ef6e348a3
528c8a3fcc2f044e39a3fc5b94492f8f
032e7549a20098f95b

Открытый ключ
5fd7449b59b461fd2ce787ec616ad46a
1da1342485a70e1f8a0ea75d80e96778
edf124769b46c7061bd6783df1e50f6c
d1fa1abeafe8256180

Сообщение (0 байтов)

Подпись
533a37f6bbe457251f023c0d88f976ae
2dfb504a843e34d2074fd823d41a591f
2b233f034f628281f2fd7a22ddd47d78
28c59bd0a21bfd3980ff0d2028d4b18a
9df63e006c5d1c2d345b925d8dc00b41
04852db99ac5c7cdda8530a113a0f4db
b61149f05a7363268c71d95808ff2e65
2600

-----1 октет
Алгоритм
Ed448

Секретный ключ
c4eab05d357007c632f3dbb48489924d
552b08fe0c353a0d4a1f00acda2c463a
fbea67c5e8d2877c5e3bc397a659949e
f8021e954e0a12274e

Открытый ключ
43ba28f430cdff456ae531545f7ecd0a
c834a55d9358c0372bfa0c6c6798c086
6aea01eb00742802b8438ea4cb82169c
235160627b4c3a9480

Сообщение (1 байт)
03

Подпись
26b8f91727bd62897af15e41eb43c377
efb9c610d48f2335cb0bd0087810f435
2541b143c4b981b7e18f62de8ccdf633
fc1bf037ab7cd779805e0dbcc0aae1cb
cee1afb2e027df36bc04dcecbf154336
c19f0af7e0a6472905e799f1953d2a0f
f3348ab21aa4adafd1d234441cf807c0
3a00
```

```
-----1 октет (с контекстом)
Алгоритм
Ed448

Секретный ключ
c4eab05d357007c632f3dbb48489924d
552b08fe0c353a0d4a1f00acda2c463a
fbea67c5e8d2877c5e3bc397a659949e
f8021e954e0a12274e

Открытый ключ
43ba28f430cdff456ae531545f7ecd0a
c834a55d9358c0372bfa0c6c6798c086
6aea01eb00742802b8438ea4cb82169c
235160627b4c3a9480

Сообщение (1 байт)
03

Контекст
666f6f

Подпись
d4f8f6131770dd46f40867d6fd5d5055
de43541f8c5e35abbcd001b32a89f7d2
151f7647f11d8ca2ae279fb842d60721
7fce6e042f6815ea000c85741de5c8da
1144a6a1aba7f96de42505d7a7298524
fda538fccbbb754f578c1cad10d54d0d
5428407e85dcbc98a49155c13764e66c
3c00

-----11 октетов
Алгоритм
Ed448

Секретный ключ
cd23d24f714274e744343237b93290f5
11f6425f98e64459ff203e8985083ffd
f60500553abc0e05cd02184bdb89c4cc
d67e187951267eb328

Открытый ключ
dcea9e78f35a1bf3499a831b10b86c90
aac01cd84b67a0109b55a36e9328b1e3
65fce161d71ce7131a543ea4cb5f7e9f
1d8b00696447001400

Сообщение (11 байтов)
0c3e544074ec63b0265e0c

Подпись
1f0a8888ce25e8d458a21130879b840a
9089d999aaba039eaf3e3afa090a09d3
89dba82c4ff2ae8ac5cdfb7c55e94d5d
961a29fe0109941e00b8dbdeea6d3b05
1068df7254c0cdc129cbe62db2dc957d
bb47b51fd3f213fb8698f064774250a5
028961c9bf8ffd973fe5d5c206492b14
0e00

-----12 октетов
Алгоритм
Ed448

Секретный ключ
258cdd4ada32ed9c9ff54e63756ae582
fb8fab2ac721f2c8e676a72768513d93
9f63dddb55609133f29adf86ec9929dc
cb52c1c5fd2ff7e21b

Открытый ключ
3ba16da0c6f2cc1f30187740756f5e79
8d6bc5fc015d7c63cc9510ee3fd44adc
24d8e968b6e46e6f94d19b945361726b
d75e149ef09817f580

Сообщение (12 байтов)
64a65f3cdedcdd66811e2915

Подпись
7eeeab7c4e50fb799b418ee5e3197ff6
bf15d43a14c34389b59dd1a7b1b85b4a
e90438aca634bea45e3a2695f1270f07
fdcdf7c62b8efeaf00b45c2c96ba457e
b1a8bf075a3db28e5c24f6b923ed4ad7
```

```
47c3c9e03c7079efb87cb110d3a99861
e72003cbae6d6b8b827e4e6c143064ff
3c00

-----13 октетов
Алгоритм
Ed448

Секретный ключ
7ef4e84544236752fbb56b8f31a23a10
e42814f5f55ca037cdcc11c64c9a3b29
49c1bb60700314611732a6c2fea98eeb
c0266a11a93970100e

Открытый ключ
b3da079b0aa493a5772029f0467baebe
e5a8112d9d3a22532361da294f7bb381
5c5dc59e176b4d9f381ca0938e13c6c0
7b174be65dfa578e80

Сообщение (13 байтов)
64a65f3cdedcdd66811e2915e7

Подпись
6a12066f55331b6c22acd5d5bfc5d712
28fbda80ae8dec26bdd306743c5027cb
4890810c162c027468675ecf645a8317
6c0d7323a2ccde2d80efe5a1268e8aca
1d6fbc194d3f77c44986eb4ab4177919
ad8bec33eb47bbb5fc6e28196fd1caf5
6b4e7e0ba5519234d047155ac727a105
3100

-----64 октета
Алгоритм
Ed448

Секретный ключ
d65df341ad13e008567688baedda8e9d
cdc17dc024974ea5b4227b6530e339bf
f21f99e68ca6968f3cca6dfe0fb9f4fa
b4fa135d5542ea3f01

Открытый ключ
df9705f58edbab802c7f8363cfe5560a
b1c6132c20a9f1dd163483a26f8ac53a
39d6808bf4a1dfbd261b099bb03b3fb5
0906cb28bd8a081f00

Сообщение (64 байта)
bd0f6a3747cd561bdddf4640a332461a
4a30a12a434cd0bf40d766d9c6d458e5
512204a30c17d1f50b5079631f64eb31
12182da3005835461113718d1a5ef944

Подпись
554bc2480860b49eab8532d2a533b7d5
78ef473eeb58c98bb2d0e1ce488a98b1
8dfde9b9b90775e67f47d4a1c3482058
efc9f40d2ca033a0801b63d45b3b722e
f552bad3b4ccb667da350192b61c508c
f7b6b5adadc2c8d9a446ef003fb05cba
5f30e88e36ec2703b349ca229c267083
3900

-----256 октетов
Алгоритм
Ed448

Секретный ключ
2ec5fe3c17045abdb136a5e6a913e32a
b75ae68b53d2fc149b77e504132d3756
9b7e766ba74a19bd6162343a21c8590a
a9cebca9014c636df5

Открытый ключ
79756f014dcfe2079f5dd9e718be4171
e2ef2486a08f25186f6bff43a9936b9b
fe12402b08ae65798a3d81e22e9ec80e
7690862ef3d4ed3a00

Сообщение (256 байтов)
15777532b0bdd0d1389f636c5f6b9ba7
34c90af572877e2d272dd078aa1e567c
fa80e12928bb542330e8409f31745041
07ecd5efac61ae7504dabe2a602ede89
```

```
e5cca6257a7c77e27a702b3ae39fc769
fc54f2395ae6a1178cab4738e543072f
c1c177fe71e92e25bf03e4ecb72f47b6
4d0465aaea4c7fad372536c8ba516a60
39c3c2a39f0e4d832be432dfa9a706a6
e5c7e19f397964ca4258002f7c0541b5
90316dbc5622b6b2a6fe7a4abffd9610
5eca76ea7b98816af0748c10df048ce0
12d901015a51f189f3888145c03650aa
23ce894c3bd889e030d565071c59f409
a9981b51878fd6fc110624dcbcde0bf7
a69ccce38fabdf86f3bef6044819de11

Подпись
c650ddbb0601c19ca11439e1640dd931
f43c518ea5bea70d3dcde5f4191fe53f
00cf966546b72bcc7d58be2b9badef28
743954e3a44a23f880e8d4f1cfce2d7a
61452d26da05896f0a50da66a239a8a1
88b6d825b3305ad77b73fbac0836ecc6
0987fd08527c1a8e80d5823e65cafe2a
3d00

-----1023 октетов
Алгоритм
Ed448

Секретный ключ
872d093780f5d3730df7c212664b37b8
a0f24f56810daa8382cd4fa3f77634ec
44dc54f1c2ed9bea86fafb7632d8be19
9ea165f5ad55dd9ce8

Открытый ключ
a81b2e8a70a5ac94ffdbcc9badfc3feb
0801f258578bb114ad44ece1ec0e799d
a08effb81c5d685c0c56f64eecaef8cd
f11cc38737838cf400

Сообщение (1023 байта)
6ddf802e1aae4986935f7f981ba3f035
1d6273c0a0c22c9c0e8339168e675412
a3debfaf435ed651558007db4384b650
fcc07e3b586a27a4f7a00ac8a6fec2cd
86ae4bf1570c41e6a40c931db27b2faa
15a8cedd52cff7362c4e6e23daec0fbc
3a79b6806e316efcc7b68119bf46bc76
a26067a53f296dafdbdc11c77f7777e9
72660cf4b6a9b369a6665f02e0cc9b6e
dfad136b4fabe723d2813db3136cfde9
b6d044322fee2947952e031b73ab5c60
3349b307bdc27bc6cb8b8bbd7bd32321
9b8033a581b59eadebb09b3c4f3d2277
d4f0343624acc817804728b25ab79717
2b4c5c21a22f9c7839d64300232eb66e
53f31c723fa37fe387c7d3e50bdf9813
a30e5bb12cf4cd930c40cfb4e1fc6225
92a49588794494d56d24ea4b40c89fc0
596cc9ebb961c8cb10adde976a5d602b
1c3f85b9b9a001ed3c6a4d3b1437f520
96cd1956d042a597d561a596ecd3d173
5a8d570ea0ec27225a2c4aaff26306d1
526c1af3ca6d9cf5a2c98f47e1c46db9
a33234cfd4d81f2c98538a09ebe76998
d0d8fd25997c7d255c6d66ece6fa56f1
1144950f027795e653008f4bd7ca2dee
85d8e90f3dc315130ce2a00375a318c7
c3d97be2c8ce5b6db41a6254ff264fa6
155baee3b0773c0f497c573f19bb4f42
40281f0b1f4f7be857a4e59d416c06b4
c50fa09e1810ddc6b1467baeac5a3668
d11b6ecaa901440016f389f80acc4db9
77025e7f5924388c7e340a732e554440
e76570f8dd71b7d640b3450d1fd5f041
0a18f9a3494f707c717b79b4bf75c984
00b096b21653b5d217cf3565c9597456
f70703497a078763829bc01bb1cbc8fa
04eadc9a6e3f6699587a9e75c94e5bab
0036e0b2e711392cff0047d0d6b05bd2
a588bc109718954259f1d86678a579a3
120f19cfb2963f177aeb70f2d4844826
262e51b80271272068ef5b3856fa8535
aa2a88b2d41f2a0e2fda7624c2850272
ac4a2f561f8f2f7a318bfd5caf969614
9e4ac824ad3460538fdc25421beec2cc
6818162d06bbed0c40a387192349db67
```

```
a118bada6cd5ab0140ee273204f628aa
d1c135f770279a651e24d8c14d75a605
9d76b96a6fd857def5e0b354b27ab937
a5815d16b5fae407ff18222c6d1ed263
be68c95f32d908bd895cd76207ae7264
87567f9a67dad79abec316f683b17f2d
02bf07e0ac8b5bc6162cf94697b3c27c
d1fea49b27f23ba2901871962506520c
392da8b6ad0d99f7013fbc06c2c17a56
9500c8a7696481c1cd33e9b14e40b82e
79a5f5db82571ba97bae3ad3e0479515
bb0e2b0f3bfcd1fd33034efc6245eddd
7ee2086ddae2600d8ca73e214e8c2b0b
db2b047c6a464a562ed77b73d2d841c4
b349735512577l3b753632efba348169
abc90a68f42611a40126d7cb21b58695
568186f7e569d2ff0f9e745d0487dd2e
b997cafc5abf9dd102e62ff66cba87

Подпись
e301345a41a39a4d72fff8df69c98075
a0cc082b802fc9b2b6bc503f926b65bd
df7f4c8f1cb49f6396afc8a70abe6d8a
ef0db478d4c6b2970076c6a0484fe76d
76b3a97625d79f1ce240e7c576750d29
5528286f719b413de9ada3e8eb78ed57
3603ce30d8bb761785dc30dbc320869e
1a00
-----
```

7.5. Тестовые векторы для Ed448ph

```
-----Тест abc
Алгоритм
Ed448ph

Секретный ключ
833fe62409237b9d62ec77587520911e
9a759cec1d19755b7da901b96dca3d42
ef7822e0d5104127dc05d6dbefde69e3
ab2cec7c867c6e2c49

Открытый ключ
259b71c19f83ef77a7abd26524cbdb31
61b590a48f7d17de3ee0ba9c52beb743
c09428a131d6b1b57303d90d8132c276
d5ed3d5d01c0f53880

Сообщение (3 байта)
616263

Подпись
822f6901f7480f3d5f562c592994d969
360287561448325650560 0bbc281ae38
1f54d6bce2ea911574932f52a4e6cadd
78769375ec3ffd1b801a0d9b3f4030cd
433964b6457ea39476511214f97469b5
7dd32dbc560a9a94d00bff07620464a3
ad203df7dc7ce360c3cd3696d9d9fab9
0f00

-----Тест abc (с контекстом)
Алгоритм
Ed448ph

Секретный ключ
833fe62409237b9d62ec77587520911e
9a759cec1d19755b7da901b96dca3d42
ef7822e0d5104127dc05d6dbefde69e3
ab2cec7c867c6e2c49

Открытый ключ
259b71c19f83ef77a7abd26524cbdb31
61b590a48f7d17de3ee0ba9c52beb743
c09428a131d6b1b57303d90d8132c276
d5ed3d5d01c0f53880

Сообщение (3 байта)
616263

Контекст
666f6f

Подпись
c32299d46ec8ff02b54540982814dce9
a05812f81962b649d528095916a2aa48
```

```
1065b1580423ef927ecf0af5888f90da
0f6a9a85ad5dc3f280d91224ba9911a3
653d00e484e2ce232521481c8658df30
4bb7745a73514cdb9bf3e15784ab7128
4f8d0704a608c54a6b62d97beb511d13
2100
-----
```

8. Вопросы безопасности

8.1. Утечки по побочным каналам

Для реализаций, выполняющих подписи, секретность приватного ключа является основополагающей. Можно обеспечить защиту от атак по побочным каналам за счёт гарантии соблюдения одинаковой последовательности инструкций и обращений к памяти при каждом значении секретного ключа.

Чтобы таким способом предотвратить утечку по побочным каналам в реализации, в арифметике по модулю недопустимо использовать зависящие от данных ветвления (например, связанные с распространением переноса). Сложение точек без возникновения побочных каналов можно реализовать просто за счёт применения унифицированных формул.

Скалярное умножение, т. е. умножение точки на целое число, требует дополнительных усилий для реализации без возникновения побочных каналов. Одним из простых подходов является реализация условного присваивания без побочных каналов, применяемого вместе с двоичным алгоритмом проверки одного бита целого числа за раз.

По сравнению с другими схемами подписи предотвращение зависимых от данных ветвлений проще, поскольку арифметические операции по модулю без побочных каналов проще (с рекомендуемыми кривыми) и имеются полные формулы сложения вместо множества особых случаев.

Отметим, что в примерах реализаций в данном документе отсутствуют меры предотвращения побочных каналов.

8.2. Случайные значения

Подписи EdDSA являются детерминированными. Это обеспечивает защиту от атак, связанных с использованием некачественных случайных значений при подписании, последствия которых, в зависимости от алгоритма, могут быть самыми серьёзными, вплоть до полной компрометации секретного ключа. Создание качественных случайных чисел может оказаться очень сложным и с этим связано множество проблем безопасности.

Очевидно, что при создании секретных ключей требуются случайные значения, но, благодаря хэшированию секретного ключа перед использованием, несколько недостающих битов энтропии не создают проблем безопасности.

Базовая проверка подписи тоже является детерминированной, однако для ускорения одновременной проверки нескольких подписей нужны случайные значения.

8.3. Использование контекста

Контекст может служить для разделения вариантов применения протокола другими протоколами (что очень трудно сделать без этого) или разными вариантами в рамках одного протокола. При использовании контекста **следует** учитывать ряд обстоятельств.

- Контексту **следует** быть строковой константой, заданной использующим контекст протоколом. **Не следует** включать в контекст непостоянные элементы из сообщений.
- Контекст **не следует** применять конъюнктурно, поскольку такое использование часто ведёт к ошибкам. Если контекст применяется, **следует** требовать его поддержки во всех доступных для использования схемах подписи.
- Контекст является дополнительным входным параметром, передаваемым из API, и даже при поддержке контекста схемой подписи он может оказаться недоступным для использования. Эта проблема усугубляется тем, что во многих случаях приложение вызывает функции подписи и её проверки не напрямую, а через тот или иной протокол.

8.4. Возможность подделки подписи

Некоторые системы считают подписи не поддающимися подделке, т. е. полагают, что злоумышленник, имеющий действительную подпись для некого сообщения с неким ключом не сможет создать другую действительную подпись для того же сообщения с тем же ключом.

Подписи Ed25519 и Ed448 не поддаются подделке из-за проверки того, что декодированное значение S меньше l. Без такой проверки можно добавить к скалярной части кратное l значение, для которого подпись пройдёт проверку.

8.5. Выбор примитива подписи

Ed25519 и Ed25519ph имеют номинальную стойкость 128 битов, а Ed448 и Ed448ph - 224. Хотя меньшей стойкости достаточно в обозримом будущем, более высокий уровень обеспечивает некоторую защиту от будущих достижений криптографии. Оба варианта взламываются квантовыми компьютерами с примерно одинаковой лёгкостью.

Варианты Ed25519ph и Ed448ph применяют предварительное хэширование, полезное в основном при взаимодействии с устаревшими API, поскольку в большинстве случаев объем подписываемых данных невелик или протокол может выполнять хэширование более эффективно, нежели прехэш (например, хэширование по дереву или расщепление данных). Предварительное хэширование существенно повышает уязвимость к слабости хэш-функции и его **не следует** применять.

В Ed25519ctx и Ed448 применяется контекст, однако с этим связаны проблемы, отмеченные в параграфе 8.3.

Функция Ed25519 широко распространена и имеет много высококачественных реализаций, другие функции поддерживаются значительно меньше.

Если достаточно уровня безопасности 128 битов, **рекомендуется** Ed25519, в противном случе **рекомендуется** Ed448.

8.6. Смешение разных прехэшей

Описанные здесь схемы разработаны так, чтобы быть устойчивыми к смешению прехэшей. То есть невозможно найти сообщение, которое пройдёт проверку с использованием той же подписи по другой схеме, даже если взять исходное подписанное сообщение. Это позволяет применять одну и ту же пару ключей для Ed25519, Ed25519ctx и Ed25519ph или одну и ту же пару для Ed448 и Ed448ph.

В качестве текстовой строки выбрана константа SigEd25519 no Ed25519 collisions так, чтобы она не декодировалась как точка. Поскольку входные данные подписи Ed25519 всегда начинаются с действительной точки, тривиальную коллизию создать невозможно. В случае хэша с затравкой (seed hash) тривиальные коллизии столь маловероятны, что даже при гораздо более вероятном выборе злоумышленником всех входных данных что-то пойдёт катастрофически не так.

8.7. Подписание большого объёма данных сразу

Следует избегать подписания сразу большого (зависит от ожидаемого способа проверки) объёма данных. В частности, если базовый протокол не требует иного, получатель **должен** буферизовать все сообщение (или достаточную для восстановления информацию, например, сжатое или зашифрованное сообщение), которое будет проверяться. Это нужно потому, что в большинстве случаев обработка непроверенных данных небезопасна, а проверка подписи проходит сообщение целиком, что в конечном итоге ведёт по меньшей мере к двум проходам.

С точки зрения API это означает, что любой интерфейс проверки IUF[1] (Initialize Update Finalize) подвержен злоупотреблениям. Не рекомендуется изменять подписание Ed25519 или Ed448 для создания действительных подписей Ed25519/Ed448 с использованием интерфейса IUF лишь с постоянной буферизацией, поскольку в таком случае практически любая ошибка приведёт к катастрофическому отказу безопасности.

8.8. Умножение на сомножитель при проверке

В приведённых формулах проверки для Ed25519 и Ed448 выполняется перемножение точек с сомножителем (cofactor). Хотя это и не обязательно для безопасности (фактически любая подпись, удовлетворяющая уравнению без умножения, подойдёт и в уравнении с умножением), в некоторых приложениях нежелательно расхождение реализаций во мнении о точном наборе допустимых подписей. Такое несогласие может приводить, например, к атакам с использованием оттисков (fingerprint).

8.9. Применение SHAKE256 в качестве хэш-функции

В Ed448 применяется хэш-функция SHAKE256, даже когда специально указано, что она не является таковой. Первая возможная проблема заключается в том, что более короткие выходные данные являются префиксами более длинных. Это допустимо, поскольку размер выходных данных фиксирован. Вторая возможная проблема связана с несоответствием стандартным представлениям о безопасности хэширования (особенно для прообразов). Однако оценочного уровня безопасности в 256 битов для защиты от коллизий и прообразов достаточно при использовании в паре с эллиптической кривой с уровнем 224 бита.

9. Литература

9.1. Нормативные документы

[FIPS202] National Institute of Standards and Technology, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", FIPS PUB 202, August 2015, <http://dx.doi.org/10.6028/NIST.FIPS.202>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <http://www.rfc-editor.org/info/rfc6234>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748[2], DOI 10.17487/RFC7748, January 2016, <http://www.rfc-editor.org/info/rfc7748>.

9.2. Дополнительная литература

[CURVE25519] Bernstein, D., "Curve25519: new Diffie-Hellman speed records", DOI 10.1007/11745853_14, February 2006, <http://cr.yp.to/ecdh.html>.

[ED25519-LIBGCRYPT-TEST-VECTORS] Koch, W., "Ed25519 Libgcrypt test vectors", July 2014, <http://git.gnupg.org/cgi-bin/gitweb.cgi?p=libgcrypt.git;a=blob;f=tests/t-ed25519.inp;h=e13566f826321eece65e02c593bc7d885b3dbe23;hb=refs/heads/master>.

[ED25519-TEST-VECTORS] Bernstein, D., Duif, N., Lange, T., Schwabe, P., and B. Yang, "Ed25519 test vectors", July 2011, <http://ed25519.cr.yp.to/python/sign.input>.

[ED448] Hamburg, M., "Ed448-Goldilocks, a new elliptic curve", June 2015, <http://eprint.iacr.org/2015/625>.

[1] В оригинале ошибочно указано IFU. См. https://errata.rfc-editor.org/eid6851/. *Прим. перев.*
[2] Документ признан устаревшим. *Прим. перев.*

[EDDSA] Bernstein, D., Duif, N., Lange, T., Schwabe, P., and B. Yang, "High-speed high-security signatures", DOI 10.1007/978-3-642-23951-9_9, September 2011, <http://ed25519.cr.yp.to/ed25519-20110926.pdf>.

[EDDSA2] Bernstein, D., Josefsson, S., Lange, T., Schwabe, P., and B. Yang, "EdDSA for more curves", July 2015, <http://ed25519.cr.yp.to/eddsa-20150704.pdf>.

[Edwards-revisited] Hisil, H., Wong, K., Carter, G., and E. Dawson, "Twisted Edwards Curves Revisited", DOI 10.1007/978-3-540-89255-7_20, December 2008, <http://eprint.iacr.org/2008/522>.

[EFD-ADD] Bernstein, D. and T. Lange, "Projective coordinates for Edwards curves", The 'add-2007-bl' addition formulas, 2007, <http://www.hyperelliptic.org/EFD/g1p/auto-edwards-projective.html#addition-add-2007-bl>.

[EFD-DBL] Bernstein, D. and T. Lange, "Projective coordinates for Edwards curves", The 'dbl-2007-bl' doubling formulas, 2007, <http://www.hyperelliptic.org/EFD/g1p/auto-edwards-projective.html#doubling-dbl-2007-bl>.

[EFD-TWISTED-ADD] Hisil, H., Wong, K., Carter, G., and E. Dawson, "Extended coordinates with a=-1 for twisted Edwards curves", The 'add-2008-hwcd-3' addition formulas, December 2008, <http://www.hyperelliptic.org/EFD/g1p/auto-twisted-extended-1.html#addition-add-2008-hwcd-3>.

[EFD-TWISTED-DBL] Hisil, H., Wong, K., Carter, G., and E. Dawson, "Extended coordinates with a=-1 for twisted Edwards curves", The 'dbl-2008-hwcd' doubling formulas, December 2008, <http://www.hyperelliptic.org/EFD/g1p/auto-twisted-extended-1.html#doubling-dbl-2008-hwcd>.

[Faster-ECC] Bernstein, D. and T. Lange, "Faster addition and doubling on elliptic curves", DOI 10.1007/978-3-540-76900-2_3, July 2007, <http://eprint.iacr.org/2007/286>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <http://www.rfc-editor.org/info/rfc4086>.

Приложение A. Библиотека Python Ed25519/Ed448

Ниже приведён код реализации Ed25519/Ed448 на языке Python (версии 3.2 и выше). Отметим, что код не предназначен для применения в работе. Хотя код даёт корректные результаты при всех входных данных, он работает медленно и не защищён от атак по побочным каналам.

```
import hashlib;
import os;

#Расчет возможного квадратного корня из x по модулю p, где p = 3 (mod 4)
def sqrt4k3(x,p) return pow(x,(p + 1)//4,p)

#Расчет возможного квадратного корня из x по модулю p, где p = 5 (mod 8).
def sqrt8k5(x,p)
    y = pow(x,(p+3)//8,p)
    #Если квадратный корень существует, это y или y*2^(p-1)/4.
    if (y * y) % p == x % p: return y
    else:
        z = pow(2,(p - 1)//4,p)
        return (y * z) % p

#Декодирование представления целого числа шестнадцатеричной строкой
def hexi(s) return int.from_bytes(bytes.fromhex(s),byteorder="big")

#Сдвиг слова x на b позиций влево.
def rol(x,b) return ((x << b) | (x >> (64 - b))) & (2**64-1)

#Из little-endian.
def from_le(s) return int.from_bytes(s, byteorder="little")

#Преобразование состояния SHA-3 в состояние s.
def sha3_transform(s)
    ROTATIONS = [0,1,62,28,27,36,44,6,55,20,3,10,43,25,39,41,45,15,\
                 21,8,18,2,61,56,14]
    PERMUTATION = [1,6,9,22,14,20,2,12,13,19,23,15,4,24,21,8,16,5,3,\
                   18,17,11,7,10]
    RC = [0x0000000000000001,0x0000000000008082,0x800000000000808a,\
          0x8000000080008000,0x000000000000808b,0x0000000080000001,\
          0x8000000080008081,0x8000000000008009,0x000000000000008a,\
          0x0000000000000088,0x0000000080008009,0x000000008000000a,\
          0x000000008000808b,0x800000000000008b,0x8000000000008089,\
          0x8000000000008003,0x8000000000008002,0x8000000000000080,\
          0x000000000000800a,0x800000008000000a,0x8000000080008081,\
          0x8000000000008080,0x0000000080000001,0x8000000080008008]

    for rnd in range(0,24)
        #AddColumnParity (Theta)
        c = [0]*5;
        d = [0]*5;
```

```
        for i in range(0,25) c[i%5]^=s[i]
        for i in range(0,5) d[i]=c[(i+4)%5]^rol(c[(i+1)%5],1)
        for i in range(0,25) s[i]^=d[i%5]
        #RotateWords (Rho)
        for i in range(0,25) s[i]=rol(s[i],ROTATIONS[i])
        #PermuteWords (Pi)
        t = s[PERMUTATION[0]]
        for i in range(0,len(PERMUTATION)-1)
            s[PERMUTATION[i]]=s[PERMUTATION[i+1]]
        s[PERMUTATION[-1]]=t;
        #NonlinearMixRows (Chi)
        for i in range(0,25,5)
            t=[s[i],s[i+1],s[i+2],s[i+3],s[i+4],s[i],s[i+1]]
            for j in range(0,5) s[i+j]=t[j]^((~t[j+1])&(t[j+2]))
        #AddRoundConstant (Iota)
        s[0]^=RC[rnd]

# Повторная интерпретация массива октетов b как массива слов
# и XOR с состоянием s.
def reinterpret_to_words_and_xor(s,b)
    for j in range(0,len(b)//8)
        s[j]^=from_le(b[8*j:][:8])

# Повторная интерпретация массива слов w в массив октетов и его возврат.
def reinterpret_to_octets(w)
    mp=bytearray()
    for j in range(0,len(w))
        mp+=w[j].to_bytes(8,byteorder="little")
    return mp

# (полу)базовая реализация SHA-3
def sha3_raw(msg,r_w,o_p,e_b)
    r_b=8*r_w
    s=[0]*25
    # Обработка целых блоков.
    idx=0
    blocks=len(msg)//r_b
    for i in range(0,blocks)
        reinterpret_to_words_and_xor(s,msg[idx:][:r_b])
        idx+=r_b
        sha3_transform(s)
    # Обработка дополнения последнего блока.
    m=bytearray(msg[idx:])
    m.append(o_p)
    while len(m) < r_b: m.append(0)
    m[len(m)-1]|=128
    # Обработка дополненного последнего блока.
    reinterpret_to_words_and_xor(s,m)
    sha3_transform(s)
    # Вывод.
    out = bytearray()
    while len(out)<e_b:
        out+=reinterpret_to_octets(s[:r_w])
        sha3_transform(s)
    return out[:e_b]

# Реализация функция SHAKE256
def shake256(msg,olen) return sha3_raw(msg,17,31,olen)

# Элемент поля (простых чисел).
class Field:
    # Создание числа x (mod p).
    def __init__(self,x,p)
        self.__x=x%p
        self.__p=p
    # Проверка совпадения полей self и y.
    def __check_fields(self,y)
        if type(y) is not Field or self.__p!=y.__p:
            raise ValueError("Fields don't match")
    # Сложение полей. Поля должны соответствовать.
    def __add__(self,y)
        self.__check_fields(y)
        return Field(self.__x+y.__x,self.__p)
    # Вычитание полей. Поля должны соответствовать.
    def __sub__(self,y)
        self.__check_fields(y)
        return Field(self.__p+self.__x-y.__x,self.__p)
    # Обращение знака поля.
    def __neg__(self)
        return Field(self.__p-self.__x,self.__p)
    # Умножение полей. Поля должны соответствовать.
    def __mul__(self,y)
        self.__check_fields(y)
        return Field(self.__x*y.__x,self.__p)
```

```
    # Деление полей. Поля должны соответствовать.
    def __truediv__(self,y)
        return self*y.inv()
    # Обращение поля (инверсия 0 это 0).
    def inv(self)
        return Field(pow(self.__x,self.__p-2,self.__p),self.__p)
    # Квадратный корень из поля. Приотсутствии возвращается none.
    # В настоящее время не реализовано для p mod 8 = 1.
    def sqrt(self)
        # Расчёт кандидата для квадратного корня.
        if self.__p%4==3: y=sqrt4k3(self.__x,self.__p)
        elif self.__p%8==5: y=sqrt8k5(self.__x,self.__p)
        else: raise NotImplementedError("sqrt(_,8k+1)")
        _y=Field(y,self.__p);
        # Проверка пригодности кандидата.
        return _y if _y*_y==self else None
    # Создание элемента того же поля в другим значением.
    def make(self,ival) return Field(ival,self.__p)
    # Элемент является аддитивным отождествлением?
    def iszero(self) return self.__x==0
    # Элементы поля равны?
    def __eq__(self,y) return self.__x==y.__x and self.__p==y.__p
    # Элементы поля неравны?
    def __ne__(self,y) return not (self==y)
    # Преобразование числа в последовательность b-1 битов.
    def tobytes(self,b)
        return self.__x.to_bytes(b//8,byteorder="little")
    # Создание числа из последовательности битов.
    def frombytes(self,x,b)
        rv=from_le(x)%(2**(b-1))
        return Field(rv,self.__p) if rv<self.__p else None
    # Определение знака числа (0 или 1). Функция знака имеет своства:
    #sign(x) = 1 - sign(-x) if x != 0.
    def sign(self) return self.__x%2

# Точка (скрученной) кривой Эдвардса.
class EdwardsPoint:
    #base_field = None
    #x = None
    #y = None
    #z = None
    def initpoint(self, x, y)
        self.x=x
        self.y=y
        self.z=self.base_field.make(1)
    def decode_base(self,s,b)
        # Проверка корректности размера кодирования точки.
        if len(s)!=b//8: return (None,None)
        # Извлечение бита знака.
        xs=s[(b-1)//8]>>((b-1)&7)
        # Декодирование y. Отказ при невозможности.
        y = self.base_field.frombytes(s,b)
        if y is None: return (None,None)
        # Попытка восстановить x. Отказ при отсутствии или
        # недействительности zero и xs.
        x=self.solve_x2(y).sqrt()
        if x is None or (x.iszero() and xs!=x.sign())
            return (None,None)
        # Смена некорректного знака x.
        if x.sign()!=xs: x=-x
        # Возврат созданной точки.
        return (x,y)
    def encode_base(self,b)
        xp,yp=self.x/self.z,self.y/self.z
        # Кодирование y.
        s=bytearray(yp.tobytes(b))
        # Добавление бита знака x к кодированию.
        if xp.sign()!=0: s[(b-1)//8]|=1<<(b-1)%8
        return s

    def __mul__(self,x)
        r=self.zero_elem()
        s=self
        while x > 0:
            if (x%2)>0:
                r=r+s
            s=s.double()
            x=x//2
        return r
    # Проверка эквивалентности двух точек.
    def __eq__(self,y)
        # Нужно проверить x1/z1 == x2/z2 и аналогично для y, поэтому
        # выполняется несколько проходов, чтобы избежать деления.
        xn1=self.x*y.z
        xn2=y.x*self.z
        yn1=self.y*y.z
```

```
        yn2=y.y*self.z
        return xn1==xn2 and yn1==yn2
    # Проверка неэквивалентности двух точек.
    def __ne__(self,y) return not (self==y)

# Точка на кривой Edwards25519.
class Edwards25519Point(EdwardsPoint)
    # Создание новой точки на кривой.
    base_field=Field(1,2**255-19)
    d=-base_field.make(121665)/base_field.make(121666)
    f0=base_field.make(0)
    f1=base_field.make(1)
    xb=base_field.make(hexi("216936D3CD6E53FEC0A4E231FDD6DC5C692CC76"+\
        "09525A7B2C9562D608F25D51A"))
    yb=base_field.make(hexi("6666666666666666666666666666666666666666"+\
        "6666666666666666666666658"))
    #The standard base point.
    @staticmethod
    def stdbase()
        return Edwards25519Point(Edwards25519Point.xb,\
            Edwards25519Point.yb)
    def __init__(self,x,y)
        # Проверка принадлежности точки кривой.
        if y*y-x*x!=self.f1+self.d*x*x*y*y:
            raise ValueError("Invalid point")
        self.initpoint(x, y)
        self.t=x*y
    # Декодирование представления точки.
    def decode(self,s)
        x,y=self.decode_base(s,256);

    # Кодирование представления точки.
    def encode(self)
        return self.encode_base(256)
    # Создание нейтральной точки для кривой.
    def zero_elem(self)
        return Edwards25519Point(self.f0,self.f1)
    # Решение для x^2.
    def solve_x2(self,y)
        return ((y*y-self.f1)/(self.d*y*y+self.f1))
    # Сложение точек.
    def __add__(self,y)
        # Формулы из EFD.
        tmp=self.zero_elem()
        zcp=self.z*y.z
        A=(self.y-self.x)*(y.y-y.x)
        B=(self.y+self.x)*(y.y+y.x)
        C=(self.d+self.d)*self.t*y.t
        D=zcp+zcp
        E,H=B-A,B+A
        F,G=D-C,D+C
        tmp.x,tmp.y,tmp.z,tmp.t=E*F,G*H,F*G,E*H
        return tmp
    # Удвоение точек.
    def double(self)
        # Формулы из EFD с распространеннным допущением a=-1).
        tmp=self.zero_elem()
        A=self.x*self.x
        B=self.y*self.y
        Ch=self.z*self.z
        C=Ch+Ch
        H=A+B
        xys=self.x+self.y
        E=H-xys*xys
        G=A-B
        F=C+G
        tmp.x,tmp.y,tmp.z,tmp.t=E*F,G*H,F*G,E*H
        return tmp
    # Порядок базовой точки.
    def l(self)
        return hexi("1000000000000000000000000000000014def9dea2f79cd"+\
            "65812631a5cf5d3ed")
    # Логарифм сомножителя.
    def c(self) return 3
    # Старший установленный бит.
    def n(self) return 254
    # Размер кодирования.
    def b(self) return 256
    # Проверка пригодности (для отладки).
    def is_valid_point(self)
        x,y,z,t=self.x,self.y,self.z,self.t
        x2=x*x
        y2=y*y
        z2=z*z
        lhs=(y2-x2)*z2
        rhs=z2*z2+self.d*x2*y2
```

```
        assert(lhs == rhs)
        assert(t*z == x*y)

# Точка на кривой Edwards448.
class Edwards448Point(EdwardsPoint)
    # Создание новой точки на кривой.
    base_field=Field(1,2**448-2**224-1)
    d=base_field.make(-39081)
    f0=base_field.make(0)
    f1=base_field.make(1)
    xb=base_field.make(hexi("4F1970C66BED0DED221D15A622BF36DA9E14657"+\
        "0470F1767EA6DE324A3D3A46412AE1AF72AB66511433B80E18B00938E26"+\
        "26A82BC70CC05E"))
    yb=base_field.make(hexi("693F46716EB6BC248876203756C9C7624BEA737"+\
        "36CA3984087789C1E05A0C2D73AD3FF1CE67C39C4FDBD132C4ED7C8AD98"+\
        "08795BF230FA14"))
    # Стандартная базвая точка.
    @staticmethod
    def stdbase()
        return Edwards448Point(Edwards448Point.xb,Edwards448Point.yb)
    def __init__(self,x,y)
        # Проверка принадлежности токи кривой.
        if y*y+x*x!=self.f1+self.d*x*x*y*y:
            raise ValueError("Invalid point")
        self.initpoint(x, y)
    # Декодирование представления точки.
    def decode(self,s)
        x,y=self.decode_base(s,456);
        return Edwards448Point(x, y) if x is not None else None
    # Кодирование представления точки.
    def encode(self)
        return self.encode_base(456)
    # Создание нейтральной точки на этой кривой.
    def zero_elem(self)
        return Edwards448Point(self.f0,self.f1)
    # Решение для x^2.
    def solve_x2(self,y)
        return ((y*y-self.f1)/(self.d*y*y-self.f1))
    # Сложение точек.
    def __add__(self,y)
        # Формулы из EFD.
        tmp=self.zero_elem()
        xcp,ycp,zcp=self.x*y.x,self.y*y.y,self.z*y.z
        B=zcp*zcp
        E=self.d*xcp*ycp
        F,G=B-E,B+E
        tmp.x=zcp*F*((self.x+self.y)*(y.x+y.y)-xcp-ycp)
        tmp.y,tmp.z=zcp*G*(ycp-xcp),F*G
        return tmp
    # Удвоение точки.
    def double(self)
        # Формулы из EFD.
        tmp=self.zero_elem()
        x1s,y1s,z1s=self.x*self.x,self.y*self.y,self.z*self.z
        xys=self.x+self.y
        F=x1s+y1s
        J=F-(z1s+z1s)
        tmp.x,tmp.y,tmp.z=(xys*xys-x1s-y1s)*J,F*(x1s-y1s),F*J
        return tmp
    # Порядок базовой точки.
    def l(self)
        return hexi("3fffffffffffffffffffffffffffffffffffffffffffffff"+\
            "fffffffff7cca23e9c44edb49aed63690216cc2728dc58f552378c2"+\
            "92ab5844f3")
    # Логарифм сомножителя.
    def c(self) return 2
    # Старший установленный бит.
    def n(self) return 447
    # Размер кодирования.
    def b(self) return 456
    # Проверка пригодности (для отладки).
    def is_valid_point(self)
        x,y,z=self.x,self.y,self.z
        x2=x*x
        y2=y*y
        z2=z*z
        lhs=(x2+y2)*z2
        rhs=z2*z2+self.d*x2*y2
        assert(lhs == rhs)

# Простая самопроверка.
def curve_self_check(point)
    p=point
    q=point.zero_elem()
    z=q
    l=p.l()+1
```

```
    p.is_valid_point()
    q.is_valid_point()
    for i in range(0,point.b())
        if (l>>i)&1 != 0:
            q=q+p
            q.is_valid_point()
        p=p.double()
        p.is_valid_point()
    assert q.encode() == point.encode()
    assert q.encode() != p.encode()
    assert q.encode() != z.encode()

# Простая самопроверка.
def self_check_curves()
    curve_self_check(Edwards25519Point.stdbase())
    curve_self_check(Edwards448Point.stdbase())

# Схема PureEdDSA.
# Обрабатывается только b mod 8 = 0.
class PureEdDSA:
    # Создание нового объекта.
    def __init__(self,properties)
        self.B=properties["B"]
        self.H=properties["H"]
        self.l=self.B.l()
        self.n=self.B.n()
        self.b=self.B.b()
        self.c=self.B.c()
    # Фиксация приватного скаляра.
    def __clamp(self,a)
        _a = bytearray(a)
        for i in range(0,self.c) _a[i//8]&=~(1<<(i%8))
        _a[self.n//8]|=1<<(self.n%8)
        for i in range(self.n+1,self.b) _a[i//8]&=~(1<<(i%8))
        return _a
    # Генерация ключа. При privkey = None создаётся случайный ключ,
    # в иных случаях возвращается пара (privkey, pubkey).
    def keygen(self,privkey)
        # Если данные секретного ключа не указаны, генерируется
        # случайное значение.
        if privkey is None: privkey=os.urandom(self.b//8)

        # Извлечение ключа.
        khash=self.H(privkey,None,None)
        a=from_le(self.__clamp(khash[:self.b//8]))
        # Возврат пары ключей (открытый ключ - A=Enc(aB)).
        return privkey,(self.B*a).encode()
    # Подпись с парой ключей.
    def sign(self,privkey,pubkey,msg,ctx,hflag)
        # Извлечение ключа.
        khash=self.H(privkey,None,None)
        a=from_le(self.__clamp(khash[:self.b//8]))
        seed=khash[self.b//8:]
        # Расчёт r и R (R применяется только в кодированной форме).
        r=from_le(self.H(seed+msg,ctx,hflag))%self.l
        R=(self.B*r).encode()
        # Расчёт h.
        h=from_le(self.H(R+pubkey+msg,ctx,hflag))%self.l
        # Расчёт s.
        S=((r+h*a)%self.l).to_bytes(self.b//8,byteorder="little")
        # Итоговая подпись в форме конкатенации R и S.
        return R+S
    # Проверка подписи с открытым ключом.
    def verify(self,pubkey,msg,sig,ctx,hflag)
        # Размеры для проверки корректности.
        if len(sig)!=self.b//4: return False
        if len(pubkey)!=self.b//8: return False
        # Разделение подписи на R и S, разбор.
        Rraw,Sraw=sig[:self.b//8],sig[self.b//8:]
        R,S=self.B.decode(Rraw),from_le(Sraw)
        # Разбор открытого ключа.
        A=self.B.decode(pubkey)
        # Проверка результатов раздора.
        if (R is None) or (A is None) or S>=self.l: return False
        # Расчёт h.
        h=from_le(self.H(Rraw+pubkey+msg,ctx,hflag))%self.l
        # Расчёт правой и левой части проверочного уравнения.
        rhs=R+(A*h)
        lhs=self.B*S
        for i in range(0, self.c)
            lhs = lhs.double()
            rhs = rhs.double()
        # Проверка равенства
        return lhs==rhs
```

```
def Ed25519_inthash(data,ctx,hflag)
    if (ctx is not None and len(ctx) > 0) or hflag:
        raise ValueError("Contexts/hashes not supported")
    return hashlib.sha512(data).digest()

# Схемы PureEdDSA.
pEd25519=PureEdDSA({\
    "B":Edwards25519Point.stdbase(),\
    "H":Ed25519_inthash\
})

def Ed25519ctx_inthash(data,ctx,hflag)
    dompfx = b""
    PREFIX=b"SigEd25519 no Ed25519 collisions"
    if ctx is not None:
        if len(ctx) > 255: raise ValueError("Context too big")
        dompfx=PREFIX+bytes([1 if hflag else 0,len(ctx)])+ctx
    return hashlib.sha512(dompfx+data).digest()

pEd25519ctx=PureEdDSA({\
    "B":Edwards25519Point.stdbase(),\
    "H":Ed25519ctx_inthash\
})

def Ed448_inthash(data,ctx,hflag)
    dompfx = b""
    if ctx is not None:
        if len(ctx) > 255: raise ValueError("Context too big")
        dompfx=b"SigEd448"+bytes([1 if hflag else 0,len(ctx)])+ctx
    return shake256(dompfx+data,114)

pEd448 = PureEdDSA({\
    "B":Edwards448Point.stdbase(),\
    "H":Ed448_inthash\
})

#Схема EdDSA.
class EdDSA:
    # Создание нового объекта схемы с заданной базовой схемой PureEdDSA
    # и прехэшированием.
    def __init__(self,pure_scheme,prehash)
        self.__pflag = True
        self.__pure=pure_scheme
        self.__prehash=prehash
        if self.__prehash is None:
            self.__prehash = lambda x,y:x
            self.__pflag = False
    # Генерация ключа. Если privkey = none, создаётся случайное значение
    # privkey, иначе применяется указанных ключ.
    # Возвращается пара (privkey, pubkey).
    def keygen(self,privkey) return self.__pure.keygen(privkey)

    # Подписание сообщения msg с заданной парой ключей.
    def sign(self,privkey,pubkey,msg,ctx=None)
        if ctx is None: ctx=b"";
        return self.__pure.sign(privkey,pubkey,self.__prehash(msg,ctx),\
            ctx,self.__pflag)
    # Проверка подписи sig для сообщения msg с открытым ключом pubkey.
    def verify(self,pubkey,msg,sig,ctx=None)
        if ctx is None: ctx=b"";
        return self.__pure.verify(pubkey,self.__prehash(msg,ctx),sig,\
            ctx,self.__pflag)

def Ed448ph_prehash(data,ctx)
    return shake256(data,64)

# Схемы подписи.
Ed25519 = EdDSA(pEd25519,None)
Ed25519ctx = EdDSA(pEd25519ctx,None)
Ed25519ph = EdDSA(pEd25519ctx,lambda x,y:hashlib.sha512(x).digest())
Ed448 = EdDSA(pEd448,None)
Ed448ph = EdDSA(pEd448,Ed448ph_prehash)

def eddsa_obj(name)
    if name == "Ed25519": return Ed25519
    if name == "Ed25519ctx": return Ed25519ctx
    if name == "Ed25519ph": return Ed25519ph
    if name == "Ed448": return Ed448
    if name == "Ed448ph": return Ed448ph
    raise NotImplementedError("Algorithm not implemented")
```

Приложение B. Драйвер библиотеки

Ниже приводится код консольного приложения, использующего приведённую выше библиотеку для интерактивных расчётов или самопроверки.

```
import sys
import binascii

from eddsa2 import Ed25519

def munge_string(s, pos, change)
    return (s[:pos] +
            int.to_bytes(s[pos] ^ change, 1, "little") +
            s[pos+1:])

# Чтение файла в формате http://ed25519.cr.yp.to/python/sign.input
lineno = 0
while True:
    line = sys.stdin.readline()
    if not line:
        break
    lineno = lineno + 1
    print(lineno)
    fields = line.split(":")
    secret = (binascii.unhexlify(fields[0]))[:32]
    public = binascii.unhexlify(fields[1])
    msg = binascii.unhexlify(fields[2])
    signature = binascii.unhexlify(fields[3])[:64]

    privkey,pubkey = Ed25519.keygen(secret)
    assert public == pubkey
    assert signature == Ed25519.sign(privkey, pubkey, msg)
    assert Ed25519.verify(public, msg, signature)
    if len(msg) == 0:
        bad_msg = b"x"
    else:
        bad_msg = munge_string(msg, len(msg) // 3, 4)
    assert not Ed25519.verify(public,bad_msg,signature)
    assert not Ed25519.verify(public, msg, munge_string(signature,20,8))
    assert not Ed25519.verify(public,msg,munge_string(signature,40,16))
```

Благодарности

EdDSA и Ed25519 исходно описаны в статье Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe и Bo-Yin Yang. Кривая Ed448 предложена Mike Hamburg.

Ранние версии этого документа созданы в соавторстве с Niels Moeller.

Отзывы для этого документа получены от Werner Koch, Damien Miller, Bob Bradley, Franck Rondepierre, Alexey Melnikov, Kenny Paterson, Robert Edmonds.

Тестовые векторы Ed25519 дважды проверены Bob Bradley с использованием трёх разных реализаций - одна на основе TweetNaCl и две разных на основе кода из SUPERCOP.

Адреса авторов

Simon Josefsson
SJD AB
Email: simon@josefsson.org
URI: http://josefsson.org/

Ilari Liusvaara
Independent
Email: ilariliusvaara@welho.com

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru