ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Пост-квантовый алгоритм шифрования ThreeBears основан на методе Любашевского-Пейкерта-Регева и криптосистеме Диня с обучением с ошибками в кольце (RLWE). Также он основан на алгоритмах Kyber и NewHope. Алгоритм Kyber использует в задачу модульного обучения с ошибками (MLWE). Создатели алгоритма заменили кольцо полиномов, лежащее в основе этого модуля, целыми числами по модулю обобщенного числа Мерсена, превратив его в целочисленное модульное обучение с ошибками (далее I-MLWE).

Целью создания алгоритма ThreeBears является поощрение исследования потенциально желательных, но менее традиционных решений. Именно поэтому ThreeBears использует I-MLWE вместо MLWE, закрытый ключ является только стартовым параметром, создатели алгоритма не использовали хэш для подтверждения открытого текста.

ОПИСАНИЕ АЛГОРИТМА

Обозначения

Пусть $\mathbb{Z}$ обозначает целые числа, а $\mathbb{Z}/N$ — кольцо целых чисел по модулю некоторого целого числа N. Для элемента $x \in \mathbb{Z}/N$ за $res(x)$ обозначим его значение в виде целого числа из множества $\{0,...,N-1\}$.

Для вещественного числа r $\lfloor r \rfloor$(«округление r вниз») — наибольшее целое число $\leq r$; $\lceil r \rceil$(«округление r вверх») — наименьшее целое число $\geq r$; и $\lceil r \rfloor$ — округление r до ближайшего целого числа.

Пусть T^n обозначает множество последовательностей из n элементов, каждый элемент которых имеет тип T. Для одной такой последовательности используется обозначение $a,b,...,z$ или $S_i \ {}^{n-1}_{i=0}$. Если S — последовательность из n элементов и $0 \leq i \leq n$, тогда S_i — ее i-ый элемент.

Код, исправляющий ошибки, ниже описывается в терминах битовых последовательностей, то есть элементов вида $\{0,1\}^n$. Пусть $a \oplus b$ – операция побитового исключающего или для двух битовых последовательностей. Если a и b имеют разную длину, короткая последовательность дополняется нулевыми элементами до длины другой последовательности. Операция побитого исключающего или для множества последовательностей обозначается как $\oplus S$.

Кодирование

Пусть B обозначает множество байт, то есть множество $\{0,...,255\}$. Открытые ключи, закрытые ключи и шифртексты хранятся и передаются в виде последовательностей байтов фиксированной длины. То есть, как элементы B^n с некоторым значением n, которое зависит от параметров системы.

В данном алгоритме последовательности байт в основном представлены в порядке от младшего к старшему разряду и обладают постоянной длиной, что означает, что первый элемент последовательности всегда будет представлять младший разряд. Кроме того, количество элементов в последовательности всегда задается типом элементов и набором параметров, так что никакое дополнительное изменение длин последовательной не используется.

Элемент z из $\mathbb{Z}/N$ кодируется как последовательность байт B, упорядоченная от младшего к старшему, длиной $\texttt{bytelength}(N) := \lceil \log_{256} N \rceil$, такая, что

$$\sum_{i=0}^{\texttt{bytelength}(N)-1} B_i \cdot 256^i = res(z)$$

Для декодирования вычисляются $B_i \cdot 256^i \bmod N$ без проверки того, что закодированный остаток меньше N.

Зашифрованные с помощью ThreeBears ключи содержат последовательность 4-битных полубайт (nibbles в описаниях алгоритмов), то есть элементов множества $\{0,...,15\}$. Эта последовательность кодируется с помощью объединения по два полубайта и формирования множества $byte^1$, в котором каждая последовательность упорядочена от младшего к старшему разряду. Последовательность полубайт s кодируется как

$$s_{2\cdot i} + 16 \cdot s_{2\cdot i+1} \quad {}^{\lceil length(s)/2 \rceil}_{i=0}$$

Параметры

Данный экземпляр алгоритма ThreeBears имеет множество параметров. Около половины из них — это длины различных стартовых параметров, зафиксированные в соответствии с требованиями безопасности. Список параметров приведен в таблице 1. Эти параметры затрагивают действия каждой функции в данной курсовой работе.

Таблица 1. Общие параметры алгоритма.

Описание	Название	Значение
Независимые параметры		
Версия спецификации	`version`	1
Длина секретного ключа (байт)	`privateKeyBytes`	40
Длина стартового параметра матрицы (байт)	`matrixSeedBytes`	24
Длина стартового параметра кодирования (байт)	`encSeedBytes`	32
Длина вектора инициализации (байт)	`ivBytes`	0

Длина разделяемого секрета (байт)	`sharedSecretBytes`	32
Количество бит на представление числа	`lgx`	10
Размерность кольца	D	312
Размерность модуля	d	$[2;4]$
Дисперсия шума	σ^2	$[0,25;1]$
Точность округления при шифровании	l	4
Кол-во бит, используемых для прямого исправления ошибок	`fecBits`	18
Защита от CCA	`cca`	$[0;1]$
Производные параметры		
Основание системы счисления	x	$2^{\mathtt{lgx}}$
Модуль	N	$x^D - x^{D/2} - 1$
Уточняющий элемент	`clar`	$x^{D/2} - 1$

Параметры для рекомендованных экземпляров приведены в таблице 2. Каждая система имеет варианты обмена ключами с защитой от CPA и защитой от CCA. Для каждой системы создатели алгоритма оценили вероятность сбоя, сложность атаки экземпляра с помощью решетчатых атак и сложность атаки по выбранному шифртексту с помощью квантового компьютера.

Таблица 2. Рекомендованные параметры ThreeBears.

Система	d	`cca`	σ^2	Вероятность ошибки	Стойкость против решетчатых атак		
					Обычная	Квантовая	Класс
BabyBear	2	0	1	$\approx 2^{-58}$	168	153	II
		1	9/16	$< 2^{-156}$	154	140	II
MamaBear	3	0	7/8	$\approx 2^{-51}$	262	238	V
		1	13/32	$< 2^{-206}$	235	213	IV

PapaBear	4	0	3/4	$\approx 2^{-52}$	351	318	V
		1	5/16	$< 2^{-256}$	314	280	V

Также создатели алгоритма определили два набора «игрушечных» параметров, показанных в таблице 3.

Таблица 3. "Игрушечные" параметры ThreeBears.

Система	lgx	D	d	cca	σ^2	Вероятность ошибки	Стойкость
TeddyBear	9	240	1	1	3/4	$\approx 2^{-58}$	60
DropBear	10	312	2	1	2	$\approx 2^{-6}$	184

Инкапсуляция

Пусть $\texttt{seq}_b(n)$ — последовательность бит целого числа n длины b в порядке от младшего к старшему разряду. Для бита a и последовательности B пусть будет

$$a \cdot B := a \cdot B_i \quad {}^{length(B)-1}_{i=0}$$

Для битовых последовательностей R и s длиной $b+1$ и b соответственно, пусть

$$\texttt{step}(R,s) := (s \oplus (s_0 \cdot R))_i \quad {}^{b}_{i=1}$$

Пусть $\texttt{step}^i(R,s)$ обозначает i-ую итерацию $\texttt{step}(R,\cdot)$, примененную к s. Тогда $\texttt{FecEncode}_{18}$ добавляет 18 бит проверочной информации таким образом, как показано на рисунке 1.

Algorithm 3: Melas FEC encode

Function $\texttt{syndrome}_{18}(B)$ **is**
  input : Bit sequence of length n
  output: Syndrome, a bit sequence of length 18.
  $P \leftarrow \texttt{seq}_{18+1}(\texttt{0x46231})$;
  $s \leftarrow 0$;
  for $i = 0$ **to** $n-1$ **do** $s \leftarrow \texttt{step}(P, s \oplus [\![B_i]\!])$;
  return s;
end

Function $\texttt{FecEncode}_{18}(B)$ **is**
  return $B \parallel \texttt{syndrome}_{18}(B)$
end

Рисунок 1. FEC кодирование.

Декапсуляция

Декодирование осуществляется сложнее, потому что для обнаружения двух ошибок необходимо решить квадратное уравнение. Пусть $Q := \mathtt{seq}_{9+1}(0x211)$. Для 9-разрядных последовательностей a и b определим следующую 9-разрядную последовательность:

$$a \odot b := \bigoplus_{i=0}^{8} (b_{8-i} \cdot \mathtt{step}^i(Q, a))$$

Операции $\oplus$ и $\odot$ определяют поле из 2^9 элементов с нейтральным по сложению элементом 0 и нейтральным по умножению элементом $\mathtt{seq}_9(0\mathtt{x100})$. Знаком $\odot$ обозначается умножение с помощью алгоритма Монтгомери. Определим $a^{\odot n}$ как n-ую степень a, полученную с помощью операции умножения $\odot$.

Остальная часть алгоритма декодирования показана на рисунке 2.

Algorithm 4: Melas FEC decode

Function $\mathtt{FecDecode}_{18}(B)$ **is**

input : Encoded bit sequence B of length n, where $18 \le n \le 511$

output: Decoded bit sequence of length $n - 18$

// Form a quadratic equation from syndrome.

$s \leftarrow \mathtt{syndrome}_{18}(B)$;

$Q \leftarrow \mathtt{seq}_{9+1}(\mathtt{0x211})$;

$c \leftarrow \mathtt{step}^9(Q, s) \odot \mathtt{step}^9(Q, \mathtt{reverse}(s))$;

$r \leftarrow \mathtt{step}^{17}\left(Q,\ c^{\odot 510}\right)$;

$s_0 \leftarrow \mathtt{step}^{511-n}(Q, s)$;

// Solve quadratic for error locators using half-trace

$\mathtt{halfTraceTable} \leftarrow [\![36, 10, 43, 215, 52, 11, 116, 244, 0]\!]$;

$\mathtt{halfTrace} \leftarrow \bigoplus_{i=0}^{8} (r_i \cdot \mathtt{seq}_9(\mathtt{halfTraceTable}_i))$;

$(e_0, e_1) \leftarrow (s_0 \odot \mathtt{halfTrace},\ (s_0 \odot \mathtt{halfTrace}) \oplus s_0))$;

// Correct the errors using the locators

for $i = 0$ **to** $n - 18 - 1$ **do**

  if $\mathtt{step}^i(Q, e_0) = \mathtt{seq}_9(1)$ **or** $\mathtt{step}^i(Q, e_1) = \mathtt{seq}_9(1)$ **then**

    $B_i \leftarrow B_i \oplus 1$;

  end

end

return $[\![B_i]\!]_{i=0}^{n-18-1}$;

end

Рисунок 2. FEC декодирование.

Генерация ключей.

Генерация ключей определяется так, чтобы секретный ключ представлял собой байтовую строку, заполненную случайными, равномерно распределенными значениями. Реализации онлайн-обмена ключами могут кэшировать промежуточные значения, такие, как секретный вектор или матрица, но алгоритм ThreeBears достаточно быстр, чтобы в этом не было необходимости.

Algorithm 5: Keypair generation

Function `GetPubKey`(sk) **is**
  input : Uniformly random private key sk of length `privateKeyBytes`
  output: Public key pk

  // Generate the private vector
  for $i = 0$ **to** $d - 1$ **do** $a_i \leftarrow \mathtt{noise}_1(\mathrm{sk}, i)$;

  // Generate a random matrix, multiply and add noise
  matrixSeed $\leftarrow H_1(\mathrm{sk}, \mathtt{matrixSeedLen})$;
  for $i, j = 0$ **to** $d - 1$ **do** $M_{i,j} \leftarrow \mathtt{uniform}(\mathrm{matrixSeed}, i, j)$;
  for $i = 0$ **to** $d - 1$ **do**
    $A_i \leftarrow \mathtt{noise}_1(\mathrm{sk}, d + i) + \sum_{j=0}^{d-1} M_{i,j} \cdot a_j \cdot \mathtt{clar}$
  end

  // Output
  pk $\leftarrow$ (matrixSeed, $[\![A_i]\!]_{i=0}^{d-1}$);
  return pk;
end

Function `Keypair`() **is**
  sk $\leftarrow$ `RandomBytes`(`privateKeyBytes`);
  return (sk, `GetPubKey`(sk));
end

Рисунок 3. Генерация ключей.

Шифрование

Функция шифрования показана на рисунке 4. Она включает в себя детерминированную версию, которая используется для дешифрования с защитой от CCA. Как и в случае с функцией генерации ключей функция инкапсуляции просто передает случайное начальное значение и вектор инициализации в функцию EncapsDet.

В реализации шифрования с защитой от CCA шум получается из стартового параметра, и этот же стартовый параметр используется в качестве открытого текста, как того требует преобразование Фудзисаки-Окамото. В ином же варианте реализации и шум, и открытый текст получаются из стартового параметра путем применения хэш-функции H_2.

Algorithm 6: Encapsulation

Function `EncapsDet`(pk, seed, iv) **is**

- **input** : Public key pk
- **input** : Uniformly random seed of length `encSeedBytes`
- **input** : Uniformly random iv of length `ivBytes`
- **output**: Shared secret; capsule

// Parse the public key

$(\text{matrixSeed}, [\![A_i]\!]_{i=0}^{d-1}) \leftarrow \text{pk}$;

// Generate ephemeral private key and make I-MLWE instance

for $i = 0$ **to** $d-1$ **do** $b_i \leftarrow \texttt{noise}_2(\text{matrixSeed}||\text{seed}||\text{iv}, i)$;

for $i, j = 0$ **to** $d-1$ **do** $M_{i,j} \leftarrow \texttt{uniform}(\text{matrixSeed}, i, j)$;

for $i = 0$ **to** $d-1$ **do**

- $B_i \leftarrow \texttt{noise}_2(\text{matrixSeed}||\text{seed}||\text{iv}, d+i) + \sum_{j=0}^{d-1} M_{j,i} \cdot b_j \cdot \texttt{clar}$;

end

// Form plaintext; encrypt using approximate shared secret

$C \leftarrow \texttt{noise}_2(\text{matrixSeed}||\text{seed}||\text{iv}, 2 \cdot d) + \sum_{j=0}^{d-1} A_j \cdot b_j \cdot \texttt{clar}$;

if `CCA` **then** pt $\leftarrow$ seed;

else pt $\leftarrow H_2(\text{matrixSeed}||\text{seed}||\text{iv}, \texttt{encSeedBytes})$;

encpt $\leftarrow \texttt{FecEncode}$(pt as a sequence of bits);

for $i = 0$ **to** length(encpt) $- 1$ **do**

- $\text{encr}_i \leftarrow \texttt{extract}_\ell(C, i) + 8 \cdot \text{encoded\_seed}_i \bmod 16$;

end

// Output

shared_secret $\leftarrow H_2(\text{matrixSeed}||\text{pt}||\text{iv}, \texttt{sharedSecretBytes})$;

capsule $\leftarrow \left([\![B_j]\!]_{j=0}^{d-1},\ \text{nibbles}\ [\![\text{encr}_i]\!]_{i=0}^{\text{length(pt)}-1}, \text{iv}\right)$;

return (shared_secret, capsule);

end

Function `Encapsulate`(pk) **is**

- (seed, iv) $\leftarrow$(`RandomBytes(encSeedBytes)`, `RandomBytes(ivBytes)`);
- **return** `EncapsDet`(pk, seed, iv);

end

Рисунок 4. Шифрование.

Дешифрование

Алгоритм дешифрования, Decapsulate, принимает в качестве входных параметров закрытый ключ *sk* и шифртекст. Он возвращает либо разделенный секрет, либо символ сбоя ⊥, как показано на рисунке 5.

Algorithm 7: Decapsulation

Function `Decapsulate`(sk, capsule) **is**
  input : Private key sk, capsule, implicit or explicit rejection
  output: Shared secret or ⊥
  `// Unpack private key and capsule`
  for $i = 0$ **to** $d - 1$ **do** $a_i \leftarrow \texttt{noise}_1(\text{sk}, i)$;
  $\left([\![B_j]\!]_{j=0}^{d-1},\ \text{nibbles}\ [\![\text{encr}_i]\!],\ \text{iv}\right) \leftarrow$ capsule;
  `// Calculate approximate shared secret and decrypt seed`
  $C \leftarrow \sum_{j=0}^{d-1} B_j \cdot a_j \cdot \texttt{clar}$;
  for $i = 0$ **to** length(encr) **do**
    $\text{encoded\_seed}_i \leftarrow \left\lfloor \frac{2 \cdot \text{encr}_i - \texttt{extract}_{\ell+1}(C,i)}{2^\ell} \right\rceil$
  end
  seed $\leftarrow$ `FecDecode`(encoded_seed);
  if `CCA` **then**
    `// Re-encapsulate to check that capsule was honest`
    $(\text{shared\_secret}, \text{capsule}') \leftarrow \texttt{EncapsDet}(\texttt{GetPubKey}(\text{sk}), \text{seed}, \text{iv})$;
    if capsule$'$ = capsule **then return** shared_secret;
    else if implicitReject **then**
      prfk $\leftarrow H_1(\text{sk}||[\![\texttt{0xFF}]\!], \texttt{privateKeyBytes})$;
      return $\leftarrow H_3(\text{prfk}||\text{capsule}, \texttt{sharedSecretBytes})$
    end
    else return ⊥;
  else
    `// Don't check: just calculate the shared secret`
    matrixSeed $\leftarrow H_1(\text{sk}, \texttt{matrixSeedLen})$;
    shared_secret $\leftarrow H_2(\text{matrixSeed}||\text{seed}||\text{iv}, \texttt{sharedSecretBytes})$;
    return shared_secret
  end
end

Рисунок 5. Дешифрование.

ВЫБОР ПАРАМЕТРОВ

Стартовые параметры

Размеры стартовых параметров в данном алгоритме рассчитаны на общее пространство поиска 2^{256} или больше. Стартовые параметры для шифрования и передаваемые ключи составляют 256 бит. Создатели алгоритма не считают, что многоцелевые атаки по восстановлению ключей будут являться существенной проблемой, поскольку на обычных вычислительных устройствах для восстановления одного из T ключей методом перебора потребовалось бы $2^{256}/T$ времени, и данный метод не допускает значительного квантового ускорения. Но защита генерации ключей практически беззатратна и осуществляется использованием секретных ключей длины 320 бит (40 байт). Это означает, что классическая многоцелевая атака с целью восстановления ключей на 2^{64} ключа будет иметь сложность 2^{256}.

Так как стартовый параметр шифрования составляет 256 бит, существует многоцелевая атака, смысл которой состоит в получении множества шифртекстов, зашифрованных с помощью одного ключа. Чтобы осложнить применение такой атаки, нужно прикрепить вектор инициализации (также IV) к каждому зашифрованному тексту. Однако параметры, рекомендованные создателями алгоритма, устанавливают длину IV равной 0 байт (т.е. вектор не используется), потому что многоцелевая атака методом полного перебора, по их мнению, не представляет реальной угрозы.

Длина стартового параметра для матрицы составляет 192 бита. Это значение выбрано для того, чтобы стартовые параметры матрицы почти наверняка никогда не совпадали даже при генерации 2^{64} ключей. И даже если они совпадут, это даст противнику весьма малое преимущество.

Модуль

Значение N было выбрано простым, чтобы исключить атаки, основанные на подкольцах. Было бы прекрасно, если бы N было простым числом Ферма, но простых чисел Ферма нужного размера не существует. Следующим очевидным выбором могли бы стать простые числа Мерсенна $2^p - 1 = 2^k \cdot x^D - 1$, где в лучшем случае k может быть ± 1: оно не может быть 0, потому что p — простое число, но $D \cdot \lg x$ является составным. Следовательно, уменьшение по модулю простого числа Мерсенна по крайней мере удвоит амплитуду шума и увеличит его дисперсию в четыре раза.

На данный момент лучший вариант — это простое число, основанное на принципе «золотого сечения»: $x^D - x^{D/2} - 1$. Умножение на $\texttt{clar} := x^{D/2} - 1$ и уменьшение по модулю этого простого числа увеличит дисперсию на $3/2$ в числах, находящихся в зоне середины диапазона. При таком усилении требуется выбрать $x \geq 2^{10}$ для приемлемой вероятности отказа и $D \geq 256$ для передачи 256-битного ключа. Это оставляет варианты:

$2^{2600} + 2^{1300} - 1$ и $2^{3120} - 2^{1560} - 1$

Если небольшая дисперсия шума алгоритма ThreeBears вызывает беспокойство, возможно использовать тот же большой модуль с $D = 260$ и $x = 2^{12}$, что увеличит значение шума. Это будет полезно, если комбинаторные атаки станут серьезной угрозой. Но согласно текущим оценкам атаковать алгоритм с параметрами $D = 312$ и $x = 2^{10}$ гораздо сложнее.

Точность округления

Параметр точности округления l при шифровании — это компромисс. Большее значение l увеличивает размер шифртекста, но уменьшает вероятность сбоя. Это позволяет добавлять больше шума, что повышает безопасность. По оценкам безопасности, произведенным создателями алгоритма, наилучший компромисс между уровнем безопасности и размером шифртекста достигается при $l = 3$, но при $l = 4$ ситуация лишь немного хуже. Параметр l был выбран равным 4, потому что это упрощает реализацию.

Дисперсия

Дисперсия шума была выбрана как простая двоичная дробь. Создатели алгоритма стремились установить вероятность отказа для экземпляров, защищенных от CPA, ниже 2^{-50}. Для экземпляров с защитой от CCA уровень шума устанавливается таким образом, чтобы частота отказов составляла около $2^{-\lambda}$, где λ — предполагаемый уровень битовой стойкости. Для обычного злоумышленника никакая атака с использованием одного ключа не может привести к сбою за ожидаемое время, меньшее, чем $1/\delta > 2^{\lambda}$. Известные атаки с ограниченными запросами намного слабее, чем эта, даже при использовании квантового компьютера.

Для BabyBear δ устанавливается ближе к λ в надежде, что, поскольку как оценка защиты от решетчатых атак, так и оценка защиты от CCA занижены, BabyBear может фактически достигнуть уровня безопасности III. Создатели алгоритма также хотели сохранить его значение значительно ниже 2^{-128}, чтобы атака с отказом была совершенно невозможной, даже с использованием квантового компьютера и значительным улучшением стратегии атаки.

Для других систем δ устанавливается чуть выше целевого уровня безопасности. Это связано с философией снижения рисков. На самом деле никто не беспокоится о злоумышленнике, выполняющем 2^{192} операции, но все беспокоятся о возможных прорывах, которые сократят работу до посильного уровня. Атаки CCA имеют меньше возможностей для улучшения, чем решетчатые атаки, и поэтому с меньшей вероятностью повлияют на практическую безопасность ThreeBears. В данном случае необходимо было только убедиться, что частота отказов не является подтвержденной слабостью.

Есть некоторые недостатки в использовании такой малой дисперсии, такие как возможность применения гибридных атак. Но Миччанчо и Пейкерт предполагают, что даже двоичный шум должен быть безопасным до тех пор, пока доступное количество LWE выборок для противника невелико. В данном случае противник видит только $d+1$ образец колец размерности D, которые, по крайней мере, достаточно малы, чтобы применение известных атак не было возможным.

АНАЛИЗ СТОЙКОСТИ

Задача I-MLWE

Безопасность ThreeBears основана на сложности задачи обучения с ошибками по модулю целого числа (I-MLWE). Было доказано, что асимптотически целочисленные задачи RLWE и полиномиальные задачи RLWE имеют одинаковую сложность, и это доказательство должно быть перенесено непосредственно на I-MLWE.

Как это часто бывает с уменьшением защиты от решетчатых атак, это доказательство является асимптотическим и неприменимо к практическим параметрам. Но также нет причин для того, чтобы предполагать, что задача I-MLWE проще, чем P-MLWE, поэтому создатели алгоритма ожидают, что эти две проблемы будут схожи по практической сложности.

Преобразование CCA

При анализе алгоритма с использованием модели случайного оракула может быть доказано, что для противника A класса IND-CCA, который выполняет q квантовых запросов на глубине d к cSHAKE в качестве квантово-доступного случайного оракула, существует квантовый алгоритм B, использующий лишь немного больше ресурсов, чем A, такой, что

$$Adv_{IND-CCA}(A) \le \sqrt[4]{2(d+1)\cdot(Adv_{I-MLWE}(B)+q/2^{8\cdot \texttt{encryptionSeedBytes}-3})} + \sqrt[4]{qd/2^{8\cdot \texttt{privateKeyBytes}}} + 16qd\delta + negl.$$

Где

• $Adv_{IND-CCA}(A)$ является преимуществом различения против метода шифрования ключа для A.

• $Adv_{I-MLWE}(B)$ — это преимущество различения B против $I-MLWE_{(d+1)\times d}$.

• q — это количество раз, когда злоумышленник вызывает cSHAKE.

• δ — вероятность отказа.

• $negl.$ — число намного меньшее, чем другие переменные, по крайней мере, для рекомендуемых параметров.

Создатели алгоритма не моделировали атаки, которые увеличивают вознаграждение противника (например, путем взлома многих ключей в пакете). Они также не анализировали многоцелевые I-MLWE атаки ни в одной из этих моделей. Существует некоторая работа по пакетным атакам для поиска коротких векторов решеток в кольце, но ничего более, что могло бы обеспечить пакетную атаку на ThreeBears.

СЛОЖНОСТЬ РЕАЛИЗАЦИИ

Время

Обе процедуры генерации ключей и шифрования в алгоритме ThreeBears требуют выборки случайной матрицы размерности $d \times d$ и умножения ее на вектор. Для используемого в данном алгоритме N подходит умножение Карацубы, поэтому эти операции занимают приблизительно $O(d^2 \cdot (\log N)^{\log_2 3}/b^2)$ времени на процессоре с b-разрядным умножителем. Это сопоставимо с шифрованием RSA с небольшой открытой экспонентой. Для шифрования и дешифрования требуется точечное векторное произведение длины d, что в d раз быстрее. Кроме того, генерация ключей и шифрование требуют выборки размером $2\cdot d$ и $2\cdot d+1$ элементов шума соответственно.

Чтобы измерить конкретную производительность, создатели алгоритма провели сравнительный анализ кода алгоритма на нескольких разных платформах, что показано в таблице 4.

Предполагается, что реализации, оптимизированные по скорости, вероятно, будут обменивать память на время дешифрования, кэшируя некоторые компоненты открытого и закрытого ключей. Объем информации, подлежащей кэшированию, зависит от ограничений приложения. Однако предоставленные реализации ничего не кэшируют.

Создатели алгоритма считают, что реализация на Skylake и Cortex-A53 достаточно близка к оптимальной, но, возможно, тщательная настройка алгоритма умножения может снизить время еще на 25%. Для реализации на Cortex-A8 можно оптимизировать алгоритм умножения и обеспечить значительное улучшение. Для реализации на Cortex-M4 возможности оптимизации тщательно не изучались.

В ходе изучения создатели обнаружили, что FEC прибавил от 0,1% до 2% времени к полученному результату. Фактически, более значительные накладные расходы от добавления FEC заключаются в том, что он создает больший шум, что может привести к большему количеству итераций в функции шума.

Таблица 4. Время работы основных функций алгоритма в циклах.

Система	Защита от CPA			Защита от CCA		
	KeyGen	Enc	Dec	KeyGen	Enc	Dec
Skylake						
BabyBear	41k	62k	28k	41k	60k	101k
MamaBear	84k	103k	34k	79k	90k	156k
PapaBear	124k	153k	40k	118k	145k	211k
Cortex-A53						
BabyBear	153k	211k	80k	154k	210k	351k
MamaBear	302k	377k	11k	297k	369k	566k
PapaBear	500k	594k	141k	492k	582k	840k
Cortex-A8						
BabyBear	344k	501k	176k	345k	495k	810k
MamaBear	729k	943k	260k	720k	931k	1379k
PapaBear	1234k	1511k	319k	1225k	1502k	2134k
Cortex-M4 (высокая скорость)						
BabyBear	644k	841k	273k	644k	824k	1299k
MamaBear	1266k	1521k	381k	1257k	1494k	2174k
PapaBear	2095k	2409k	488k	2082k	2378k	3272k
Cortex-M4 (малые затраты памяти)						
BabyBear	744k	1039k	273k	744k	1022k	1495k
MamaBear	1564k	1967k	381k	1548k	1929k	2609k
PapaBear	2691k	3201k	488k	2663k	3150k	4044k

Размер кода

Был измерен общий размер кода на каждой платформе для реализации MamaBear. Итоговые размеры приведены в таблице 7. Реализации для разных параметров могут отличать по размеру.

Таблица 5. Размер выходных данных алгоритма в байтах.

Система	Закрытый ключ	Открытый ключ	Шифртекст
BabyBear	40	804	917
MamaBear	40	1194	1307
PapaBear	40	1584	1697

Таблица 6. Размер кода для MamaBear в байтах.

Компонент	Skylake	Cortex-A53	Cortes-A8	Cortex-M4	
				Скорость	Память
Вычисления	2194	1892	1424	930	930
FEC Меласа	655	541	431	417	417
cSHAKE	1488	900	866	777	777
Основная система	3206	2843	2133	2187	1951
Всего	7543	6176	4854	4311	4075

Использование памяти

С помощью специальных инструментов было измерено использование стековой памяти каждой функции верхнего уровня для реализаций на Skylake и Cortex-M4. Эти измерения включали память, используемую алгоритмом ThreeBears во время работы, включая хэш-контексты и вызовы функций, но не входные или выходные данные. Результаты приведены в таблице 7, их следует рассматривать как приблизительные.

Таблица 7. Использование памяти алгоритмом в байтах, исключая входные и выходные данные.

Система	Защита от CPA			Защита от CCA		
	Keygen	Enc	Dec	Keygen	Enc	Dec
Skylake (высокая скорость)						
BabyBear	6216	6632	4232	6216	6632	8184
MamaBear	9112	9528	4632	9112	9560	11512
PapaBear	12856	13272	5048	12856	13304	15672

Skylake (малые затраты памяти)						
Все реализации	2392	2424	2168	2392	2424	3080
Cortex-M4 (высокая скорость)						
BabyBear	2760	2832	2080	2760	2832	4944
MamaBear	3256	3312	2080	3256	3320	5904
PapaBear	3736	3800	2080	3736	3800	6864
Cortex-M4 (малые затраты памяти)						
Все реализации	2288	2352	2080	2288	2352	3024

ЗАКЛЮЧЕНИЕ

В данной пояснительной записке было приведено подробное описание алгоритма Three Bears, участника конкурса национального института стандартов и технологий NIST по выбору квантово-устойчивых криптоалгоритмов для стандартизации. Также были описаны рекомендуемые параметры кода и известные атаки.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, EUROCRYPT 2010, volume 6110 of LNCS, pages 1–23. Springer, Heidelberg, May / June 2010.
2. Mike Hamburg. Ed448-goldilocks, a new elliptic curve. Cryptology ePrint Archive, Report 2015/625, 2015.
3. Joppe Bos, Leo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, and Damien Stehle. CRYSTALS – kyber: a CCA-secure module-lattice based KEM. Cryptology ePrint Archive, Report 2017/634, 2017.
4. Erdem Alkim, Leo Ducas, Thomas Poppelmann, and Peter Schwabe. Post-quantum key exchange - a new hope. Cryptology ePrint Archive, Report 2015/1092, 2015.
5. Jintai Ding, Xiang Xie, and Xiaodong Lin. A simple provably secure key exchange scheme based on the learning with errors problem. Cryptology ePrint Archive, Report 2012/688, 2012.