Internet Research Task Force (IRTF)
Request for Comments: 9106
Category: Informational
ISSN: 2070-1721

A. Biryukov
D. Dinu
University of Luxembourg
D. Khovratovich
ABDK Consulting
S. Josefsson
SJD AB

September 2021

Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications

Функция Argon2 для хэширования паролей и приложений Proof-of-Work

Аннотация

Этот документ определяет функцию Argon2 с интенсивным использованием памяти для хэширования паролей и приложений подтверждения работы (proof-of-work). Дано ориентированное на разработчиков описание с тестовыми векторами. Цель заключается в упрощении адаптации Argon2 для протоколов Internet. Документ является результатом работы исследовательской группы Crypto Forum (Crypto Forum Research Group или CFRG) в составе IRTF.

Статус документа

Документ не относится к категории Internet Standards Track и публикуется для информации.

Документ является результатом работы IRTF[1]. IRTF публикует результаты относящихся к Internet исследований и разработок. Эти результаты могут оказаться не пригодными для реализации. Данный RFC представляет согласованное мнение исследовательской группы Crypto Forum в рамках IRTF. Документы, одобренные для публикации IRSG, не претендуют на статус Internet Standard (см. раздел 2 в RFC 7841).

Информацию о текущем статусе документа, ошибках и способах обратной связи можно найти по ссылке https://www.rfc-editor.org/info/rfc9106.

Оглавление

1. Введение....2
1.1. Уровни требований....2
2. Обозначения и соглашения....2
3. Алгоритм Argon2....3
3.1. Входные и выходные данные Argon2....3
3.2. Работа Argon2....3
3.3. Хэш функция с переменным размером H'....4
3.4. Индексирование....4
3.4.1. Расчёт 32-битовых значений J_1 и J_2....4
3.4.1.1. Argon2d....4
3.4.1.2. Argon2i....4
3.4.1.3. Argon2id....4
3.4.2. Отображение J_1 и J_2 на индекс опорного блока [l][z]....4
3.5. Функция сжатия G....5
3.6. Перестановка P....5
4. Выбор параметров....6
5. Тестовые векторы....6
5.1. Тестовые векторы Argon2d....6
5.2. Тестовые векторы Argon2i....7
5.3. Тестовые векторы Argon2id....8
6. Взаимодействие с IANA....8
7. Вопросы безопасности....9
7.1. Безопасность как хэш-функция и KDF....9
7.2. Атаки с компромиссом между расходом времени и памяти....9
7.3. Безопасность для ограниченных по времени защитников....9
7.4. Рекомендации....9
8. Литература....9
8.1. Нормативные документы....9
8.2. Дополнительная литература....9
Благодарности....10
Адреса авторов....10

[1]Internet Research Task Force - комиссия по исследовательским задачам Internet.

1. Введение

Этот документ определяет функцию Argon2 с интенсивным использованием памяти для хэширования паролей и приложений подтверждения работы (proof-of-work). Дано ориентированное на разработчиков описание с тестовыми векторами. Цель заключается в упрощении адаптации Argon2 для протоколов Internet. Документ воответствует хэш-функции Argon2 версии 1.3.

Argon2 - функция с интенсивным использованием памяти [HARD]. Это оптимизированная конструкция, нацеленная на максимальную скорость заполнения памяти и эффективное использование множества расчётных блоков с обеспечением устойчивости к компромиссным (trade-off) атакам. Функция Argon2 оптимизирована для архитектуры x86 и использует организацию кэше и памяти в последних моделях процессоров Intel и AMD. Argon2 имеет 1 основной вариант Argon2id и два дополнительных (Argon2d и Argon2i). В Argon2d применяется зависимый от данных доступ к памяти, что подходит для криптовалют и приложений proof-of-work без угроз атак через побочные каналы. В Argon2i применяется независимый от данных доступ к памяти, который предпочтителен для хэширования паролей и вывода ключей на основе пароля. Argon2id работает как Argon2i для первой половины первого прохода по памяти и как Argon2d в остальное время, что обеспечивает защиту от атак по побочным каналам и экономию при переборе (brute-force) за счёт компромисса между временем и расходом памяти. В Argon2i применяется больше проходов по памяти для защиты от компромиссных атак [AB15].

Функция Argon2id **должна** поддерживаться всеми реализациями этого документа **могут** поддерживаться также Argon2d и Argon2i.

Argon2 - это также режим работы на основе функции сжатия с фиксированным размером ввода (G) и функции хэширования с переменным входным размером (H). Хотя Argon2 можно использовать с произвольной функцией H, которая обеспечивает вывод вплоть до 64 байтов, в документе применяется функций BLAKE2b [BLAKE2].

Дополнительные сведения и обсуждение представлены в статье по Argon2 [ARGON2].

Этот документ представляет согласованный подход исследовательской группы Crypto Forum (CFRG).

1.1. Уровни требований

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 BCP 14 [RFC2119] [RFC8174] тогда и только тогда, когда они выделены шрифтом, как показано здесь.

2. Обозначения и соглашения

x^y
: Умножение целого числа x самого на себя y раз (возведение в степень).

a*b
: Произведение целых чисел a и b.

c-d
: Вычитание целого числа d из целого числа c.

E_f
: Переменная E с подстрочным индексом f.

g / h
: Деление целого числа g на целое число h (результат является рациональным числом).

I(j)
: Значение функции I для аргумента j.

K || L
: Объединение (конкатенация) строки K со строкой L.

a XOR b
: Побитовая рперация исключительное-ИЛИ для битовых строк a и b.

a mod b
: Деление целого числа a по модулю b (результат всегда относится к диапазону [0, b-1]).

a >>> n
: Сдвиг 64-битовой строки a на n битов вправо.

trunc(a)
: 64-битовое значение усечённое до 32 младших битов.

floor(a)
: Наибольшее целое число, не превышающее a

ceil(a)
: Наименьшее целое число, которое не меньше a

extract(a, i)
: i-ый набор из 32 битов битовой строки a, начиная с позиции 0.

|A|
: Число элементов множества A.

LE32(a)
: 32-битовое целое число, преобразованное в строку байтов little endian (например, десятичное число 123456 будет иметь вид 40 E2 01 00).

LE64(a)
: 64-битовое целое число, преобразованное в строку байтов little endian (например, десятичное число 123456 будет иметь вид 40 E2 01 00 00 00 00 00).

int32(s)
: 32-битовая строка s, преобразованная в неотрицательное целое число в формате little endian.

int64(s)
: 64-битовая строка s, преобразованная в неотрицательное целое число в формате little endian.

length(P)
Количество байтов строки P в виде 32-битового целого числа.

ZERO(P)
Строка из P нулевых байтов.

3. Алгоритм Argon2

3.1. Входные и выходные данные Argon2

Ниже перечислены входные параметры Argon2.

- Строка сообщения P, являющаяся паролем для приложений хэширования паролей. Размер строки **должен** быть не более 2^(32)-1 байтов.
- Обноразовое значение S, служащее затравкой для приложений хэширования паролей. Размер затравки **должен** быть не более 2^(32)-1 байтов. Для хэштрования паролей **рекомендуется** размер 16 байтов. Затравку **следует** делать уникальной для каждого пароля.
- Степень распараллеливания p, определяющая число независимых (но синхронизированных) цепочек расчётов (lane), которые могут быть запущены. Это **должно** быть целое число из диапазона 1 - 2^(24)-1.
- Размер тега T **должен** быть целым числом байтов от 4 до 2^(32)-1.
- Размер памяти (в килобайтах) **должен** быть целым числом от 8*p до 2^(32)-1. Фактическое число блоков m' является округлением вниз до ближайшего значения, кратного 4*p.
- Число проходов t (служит для настройки времени работы независимо от объёма памяти), которое **должно** быть целым числом из диапазона 1 - 2^(32)-1.
- Номер версии v **должен** быть 1 байтом со значением 0x13.
- **Необязательное** значение секрета K, которое (при наличии) **должно** иметь размер не более 2^(32)-1 байтов.
- **Необязательные** связанные данные X, размер которых (при наличии) **должен** быть не более 2^(32)-1 байтов.
- Тип y **должен** иметь значение 0 для Argon2d, 1 для Argon2i, 2 для Argon2id.

Выходные данные Argon2 или тег - это строка размером T байтов.

3.2. Работа Argon2

Argon2 использует внутреннюю функцию сжатия G с двумя входными параметрами по 1024 байта, результатом в 1024 байта и внутренней хэш-функцией H^x(), где x - выходной размер в байтах. Функция H^x(), примененная в строке A - это функция BLAKE2b ([BLAKE2], параграф 3.3), которая принимает параметры (d,ll,kk=0,nn=x), где d - это A с заполнением до размера, кратного 128 байтам, а ll - размер d в байтах. Функция сжатия G основана на внутренней перестановке. Применяется также хэш-функция H' с переменным размером, основанная на H. Описание G дано в параграфе 3.5, H' - в параграфе 3.3. Работа функции Argon2 описана ниже.

1. Создаётся 64-байтовое значение H_0, как показано ниже. Если K, X или S имеет нулевой размер, они просто не включаются, но поле размера остаётся.

```
H_0 = H^(64)(LE32(p) || LE32(T) || LE32(m) || LE32(t) ||
        LE32(v) || LE32(y) || LE32(length(P)) || P ||
        LE32(length(S)) || S ||  LE32(length(K)) || K ||
        LE32(length(X)) || X)
```

Рисунок 1. Генерация H_0.

2. Выделяется память как m' блоков по 1024 байта, где m' определяется приведённым ниже выражением.

```
m' = 4 * p * floor (m / 4p)
```

Рисунок 2. Выделение памяти.

Для p цепочек память организуется в форме матрицы B[i][j] блоков с p строк (цепочек) и q = m' / p столбцов.

3. Рассчитывается B[i][0] для всех i от 0 до p-1.

```
B[i][0] = H'^(1024)(H_0 || LE32(0) || LE32(i))
```

Рисунок 3. Первые блоки прохода.

4. Рассчитывается B[i][1] для всех i от 0 до p-1.

```
B[i][1] = H'^(1024)(H_0 || LE32(1) || LE32(i))
```

Рисунок 4. Вторые блоки прохода.

5. Рассчитывается B[i][j] для всех i от 0 до p-1 и всех j от 2 до q-1. Расчёты **должны** выполняться по срезам (slicewise, параграф 3.4) - сначала для всех цепочек среза 0 (с произвольным порядком цепочек), затем для среза 1 и т. д. Индексы блоков l и z определяются для каждой пары i, j по-разному в Argon2d, Argon2i, Argon2id.

```
B[i][j] = G(B[i][j-1], B[l][z])
```

Рисунок 5. Генерация последующих блоков.

6. Если число проходов больше 1, п. 5 повторяется. Рассчитываются B[i][0] и B[i][j] для всех i от 0 до p-1 и всех j от 1 до q-1. Однако блоки рассчитываются по-разному, поскольку старое значение комбинируется с новым (XOR), как показано ниже.

```
B[i][0] = G(B[i][q-1], B[l][z]) XOR B[i][0];
B[i][j] = G(B[i][j-1], B[l][z]) XOR B[i][j].
```

Рисунок 6. Последующие проходы.

7. После t итераций рассчитывается финальный блок C как XOR для последнего столбца

```
C = B[0][q-1] XOR B[1][q-1] XOR ... XOR B[p-1][q-1]
```

Рисунок 7. Финальный блок.

8. Результирующий тег рассчитывается как H'^T(C).

3.3. Хэш функция с переменным размером H'

Пусть V_i - 64-байтовый блок, а W_i - его первые 32 байта. Тогда функция H' определяется, как показано ниже.

```
if T <= 64
    H'^T(A) = H^T(LE32(T)||A)
else
    r = ceil(T/32)-2
    V_1 = H^(64)(LE32(T)||A)
    V_2 = H^(64)(V_1)
    ...
    V_r = H^(64)(V_{r-1})
    V_{r+1} = H^(T-32*r)(V_{r})
    H'^T(X) = W_1 || W_2 || ... || W_r || V_{r+1}
```

Рисунок 8. Функция H' для расчёта тега и начального блока.

3.4. Индексирование

Для параллельного расчёта блока матрица памяти разбита на SL=4 вертикальных среза. Пересечение среза и цепочки называется сегментом и имеет размер q/SL. Сегменты одного среза можно рассчитывать параллельно и они не связаны между собой. Все другие блоки могут быть связаны (ссылаться один на другой).

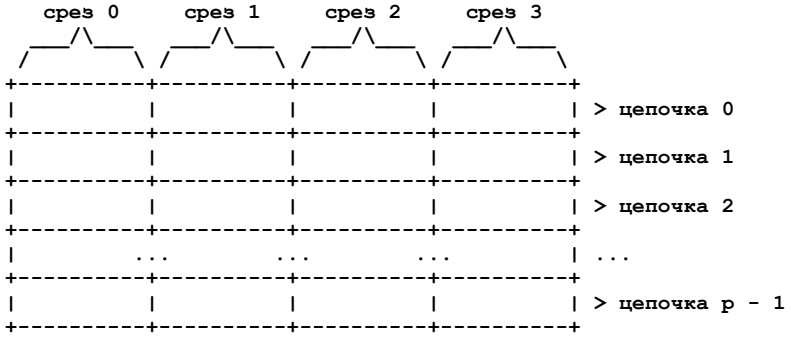

Рисунок 9. Однопроходная функция Argon2 с p цепочек и 4 срезами.

3.4.1. Расчёт 32-битовых значений J_1 и J_2

3.4.1.1. Argon2d

J_1 задаётся первыми 32 битами блока B[i][j-1], а J_2 - следующими 32 битами блока B[i][j-1].

```
J_1 = int32(extract(B[i][j-1], 0))
J_2 = int32(extract(B[i][j-1], 1))
```

Рисунок 10. Расчет J1,J2 в Argon2d.

3.4.1.2. Argon2i

Для каждого сегмента сначала рассчитывается значение Z, ка показано ниже.

```
Z = ( LE64(r) || LE64(l) || LE64(sl) || LE64(m') || LE64(t) || LE64(y) )
```

Рисунок 11. Входные данные для расчета J1,J2 в Argon2i.

где

r - номер прохода;
l - номер цепочки;
sl - номер среза;
m' - общее число блоков памяти;
t - число проходов;
y - тип Argon2 (0 для Argon2d, 1 для Argon2i, 2 для Argon2id).

Затем рассчитываются q/(128*SL) 1024-байтовых значений, как показано ниже.

```
G(ZERO(1024),G(ZERO(1024), Z || LE64(1) || ZERO(968) )),
G(ZERO(1024),G(ZERO(1024), Z || LE64(2) || ZERO(968) )),... ,
G(ZERO(1024),G(ZERO(1024), Z || LE64(q/(128*SL)) || ZERO(968) ))
```

Эти значения делятся на q/(SL) 8-байтовых значений X, которые рассматриваются как X1||X2 и преобразуются в J_1=int32(X1) и J_2=int32(X2). Значения r, l, sl, m', t, y, i представляются как 8 байтов в формате little endian.

3.4.1.3. Argon2id

Для прохода 0 и среза 0 и 1 значения J_1 и J_2 рассчитываются как в Argon2i, в остальных случаях - как в Argon2d.

3.4.2. Отображение J_1 и J_2 на индекс опорного блока [l][z]

Значение l = J_2 mod p задаёт индекс цепочки, из которой берётся блок. Для первого прохода (r=0) и первого среза (sl=0) блок берётся из текущей строки.

Множество W содержит индексы, которые указываются по приведённым ниже правилам.

1. Если l - текущая цепочка, W включает индексы всех блоков из последних SL-1 = 3 рассчитанных и завершенных сегментов, а также из блоков, рассчитанных в текущем сегменте и проходе, кроме B[i][j-1].
2. Если цепочка l не является текущей, W включает индексы всех блоков из последних SL-1 = 3 рассчитанных и завершенных сегментов в цепочке l. Если B[i][j] - первый блок сегмента, последний индекс исключается из W.

Затем берётся блок из W с неоднородным распределением по [0, |W|) с помощью указанного ниже отображения.

```
J_1 -> |W|(1 - J_1^2 / 2^(64))
```

Рисунок 12. Расчёт J1.

Чтобы избежать расчётов с плавающей запятой, применяется показанная ниже аппроксимация.

```
x = J_1^2 / 2^(32)
y = (|W| * x) / 2^(32)
zz = |W| - 1 - y
```

Рисунок 13. Расчёт J1, часть 2.

Затем берётся индекс zz из W. Это будет значение z для индекса «опорного» (reference) блока [l][z].

3.5. Функция сжатия G

Функция сжатия G строится на основе BLAKE2b-преобразования P, принимающего на входе 128 байтов, которые можно рассматривать ка восемь 16-байтовых регистров.

```
P(A_0, A_1, ... ,A_7) = (B_0, B_1, ... ,B_7)
```

Рисунок 14. Функция раунда Блэйка P.

Функция сжатия G(X, Y) работает с двумя блоками по 1024 байта - X и Y. Сначала рассчитывается R = X XOR Y, затем R рассматривается как матрица 8x8 16-байтовых регистров R_0, R_1, ... , R_63. После этого для получения Z применяется преобразование P сначала к каждой строке, затем к каждому столбцу.

```
( Q_0,  Q_1,  Q_2, ... ,  Q_7) <- P( R_0,  R_1,  R_2, ... ,  R_7)
( Q_8,  Q_9, Q_10, ... , Q_15) <- P( R_8,  R_9, R_10, ... , R_15)
                               ...
(Q_56, Q_57, Q_58, ... , Q_63) <- P(R_56, R_57, R_58, ... , R_63)
( Z_0,  Z_8, Z_16, ... , Z_56) <- P( Q_0,  Q_8, Q_16, ... , Q_56)
( Z_1,  Z_9, Z_17, ... , Z_57) <- P( Q_1,  Q_9, Q_17, ... , Q_57)
                               ...
( Z_7, Z_15, Z 23, ... , Z_63) <- P( Q_7, Q_15, Q_23, ... , Q_63)
```

Рисунок 15. Ядро функции сжатия G.

Заключительным этапом G является операция Z XOR R.

```
G: (X, Y) -> R -> Q -> Z -> Z XOR R
```

Рисунок 16. Функция сжатия G в Argon2.

3.6. Перестановка P

Перестановка P основана на круговой (round) функции BLAKE2b. Восемь 16-байтовых входных элементов S_0, S_1, ... , S_7 рассматриваются как матрица 4x4 из 64-битовых слов, где S_i = (v_{2*i+1} || v_{2*i}).

```
 v_0  v_1  v_2  v_3
 v_4  v_5  v_6  v_7
 v_8  v_9 v_10 v_11
v_12 v_13 v_14 v_15
```

Рисунок 17. Маркировка элементов матрицы.

Это работает, как показано ниже.

```
GB(v_0, v_4,  v_8, v_12)
GB(v_1, v_5,  v_9, v_13)
GB(v_2, v_6, v_10, v_14)
GB(v_3, v_7, v_11, v_15)

GB(v_0, v_5, v_10, v_15)
GB(v_1, v_6, v_11, v_12)
GB(v_2, v_7,  v_8, v_13)
GB(v_3, v_4,  v_9, v_14)
```

Рисунок 18. Передача элементов матрицы в GB.

Определение GB(a, b, c, d) приведено ниже.

```
a = (a + b + 2 * trunc(a) * trunc(b)) mod 2^(64)
d = (d XOR a) >>> 32
c = (c + d + 2 * trunc(c) * trunc(d)) mod 2^(64)
b = (b XOR c) >>> 24

a = (a + b + 2 * trunc(a) * trunc(b)) mod 2^(64)
d = (d XOR a) >>> 16
c = (c + d + 2 * trunc(c) * trunc(d)) mod 2^(64)
b = (b XOR c) >>> 63
```

Рисунок 19. Детали GB.

Сложение по модулю в GB сочетается с 64-битовым умножением. Перемножение является единственным отличием от исходной функции BLAKE2b. Это сделано для того, чтобы увеличить глубину схемы и, соответственно, время работы ASIC-реализация, при этом время работы на CPU будет примерно таким же за счёт распараллеливания и конвейеров.

4. Выбор параметров

Функция Argon2d оптимизирована для систем, где злоумышленники не имеют постоянного доступа к системной памяти или CPU, т. е. не могут организовать атаку через побочные каналы на основе данных о времени, а также не способны быстро восстановить пароль с использованием сборки мусора. Такие условия достаточно типичны для backend-серверов и майнинга криптовалют. Для практических ситуаций предлагаются указанные ниже настройки.

- Майнинг криптовалюты, занимающий 0,1 сек на CPU с частотой 2 ГГц и одним ядром, - Argon2d с двумя цепочками и 250 Мбайт ОЗУ.

Функция Argon2id оптимизирована для более реалистичных ситуаций, где злоумышленник может получить доступ к машине, использовать её CPU или организовать атаку с холодной загрузкой (cold-boot). Предлагаемые настройки приведены ниже.

- Аутентификация на backend-сервере, занимающая 0,5 сек на CPU с частотой 2 ГГц и 4 ядрами, - Argon2id с 8 цепочками и 4 Гбайт ОЗУ.
- Вывод ключей для шифрования диска, занимающий 3 сек на CPU с частотой 2 ГГц и 2 ядрами, - Argon2id с 4 цепочками и 6 Гбайт ОЗУ.
- Аутентификация на frontend-сервере, занимающая 0,5 сек на CPU с частотой 2 ГГц и 2 ядрами, - Argon2id с 4 цепочками и 1 Гбайт ОЗУ.

Ниже приведена процедура выбора типа и параметров для практического применение Argon2.

1. Если применим однородно безопасный вариант без адаптации к оборудованию и прилежению, следует выбирать Argon2id с числом итераций t=1, числом цепочек p=4, m=2^(21) (2 Гбайт ОЗУ), 128-битовой затравкой и размером тега 256 битов. Это **первый рекомендуемый** вариант.
2. Если доступен меньший объем памяти однородно безопасным вариантом будет Argon2id и чсилом итераций t=3, числом цепочек p=4, m=2^(16) (64 Мбайт ОЗУ), 128-битовой затравкой и размером тега 256 битов. Это **второй рекомендуемый** вариант.
3. В иных случаях выбор начинается с типа y. Если различия между типами не ясны или предполагается реальная угроза атак по побочным каналам, следует выбирать Argon2id.
4. Выбирается число цепочек p=4.
5. Определяется максимальный объем памяти, доступной для каждого вызова, и транслируется в параметр m.
6. Определяется максимальное число секунд, приемлемое для каждого вызова.
7. Выбирается размер затравки (salt). 128 битов достаточно для всех приложений, но можно сократить размер до 64 битов в случае ограничений по пространству.
8. Выбирается размер тега. 128 битов достаточно для большинства приложений, включая вывод ключей. Если нужны более длинные ключи, размер тега можно увеличить.
9. Если атаки по побочным каналам являются реальной угрозой или нет уверенности по этому вопросу, при вызове библиотеки следует включить опцию очистки памяти (memory-wiping).
10. Запускается схема типа y, с памятью m и числом цепочек lanes с использованием разного числа проходов t. Определяется максимальное значение t при котором время работы не выходит за допустимые пределы. Если это происходит при t = 1, следует уменьшить m.
11. Используется Argon2 с найденными значениями m, p и t.

5. Тестовые векторы

В этом разделе приведены тестовые векторы для Argon2.

5.1. Тестовые векторы Argon2d

Тестовые векторы представлены с полным выводом (теги). Для удобства разработчиков приведены также некоторые внутренние переменные, а именно первый и последний блок памяти в каждом проходе.

```
========================================
Argon2d версии 19
========================================
Размер памяти: 32 Кбайт
Число проходов: 3
Число параллельных цепочек: 4
```

```
Размер тега: 32 байта
Пароль[32]: 01 01 01 01 01 01 01 01
             01 01 01 01 01 01 01 01
             01 01 01 01 01 01 01 01
             01 01 01 01 01 01 01 01
Затравка[16]: 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02
Секрет[8]: 03 03 03 03 03 03 03 03
Связанные данные[12]: 04 04 04 04 04 04 04 04 04 04 04 04
Дайджест предварительного хэширования: b8 81 97 91 a0 35 96 60
                    bb 77 09 c8 5f a4 8f 04
                    d5 d8 2c 05 c5 f2 15 cc
                    db 88 54 91 71 7c f7 57
                    08 2c 28 b9 51 be 38 14
                    10 b5 fc 2e b7 27 40 33
                    b9 fd c7 ae 67 2b ca ac
                    5d 17 90 97 a4 af 31 09

 После прохода 0:
Блок 0000 [  0]: db2fea6b2c6f5c8a
Блок 0000 [  1]: 719413be00f82634
Блок 0000 [  2]: a1e3f6dd42aa25cc
Блок 0000 [  3]: 3ea8efd4d55ac0d1
...
Блок 0031 [124]: 28d17914aea9734c
Блок 0031 [125]: 6a4622176522e398
Блок 0031 [126]: 951aa08aeecb2c05
Блок 0031 [127]: 6a6c49d2cb75d5b6

 После прохода 1:
Блок 0000 [  0]: d3801200410f8c0d
Блок 0000 [  1]: 0bf9e8a6e442ba6d
Блок 0000 [  2]: e2ca92fe9c541fcc
Блок 0000 [  3]: 6269fe6db177a388
...
Блок 0031 [124]: 9eacfcfbdb3ce0fc
Блок 0031 [125]: 07dedaeb0aee71ac
Блок 0031 [126]: 074435fad91548f4
Блок 0031 [127]: 2dbfff23f31b5883

 После прохода 2:
Блок 0000 [  0]: 5f047b575c5ff4d2
Блок 0000 [  1]: f06985dbf11c91a8
Блок 0000 [  2]: 89efb2759f9a8964
Блок 0000 [  3]: 7486a73f62f9b142
...
Блок 0031 [124]: 57cfb9d20479da49
Блок 0031 [125]: 4099654bc6607f69
Блок 0031 [126]: f142a1126075a5c8
Блок 0031 [127]: c341b3ca45c10da5
Тег: 51 2b 39 1b 6f 11 62 97
     53 71 d3 09 19 73 42 94
     f8 68 e3 be 39 84 f3 c1
     a1 3a 4d b9 fa be 4a cb
```

5.2. Тестовые векторы Argon2i

```
========================================
Argon2i версии 19
========================================
Размер памяти: 32 Кбайт
Число проходов: 3
Число параллельных цепочек: 4
Размер тега: 32 байта
Пароль[32]: 01 01 01 01 01 01 01 01
             01 01 01 01 01 01 01 01
             01 01 01 01 01 01 01 01
             01 01 01 01 01 01 01 01
Затравка[16]: 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02
Секрет[8]: 03 03 03 03 03 03 03 03
Связанные данные[12]: 04 04 04 04 04 04 04 04 04 04 04 04
Дайджест предварительного хэширования: c4 60 65 81 52 76 a0 b3
                    e7 31 73 1c 90 2f 1f d8
                    0c f7 76 90 7f bb 7b 6a
                    5c a7 2e 7b 56 01 1f ee
                    ca 44 6c 86 dd 75 b9 46
                    9a 5e 68 79 de c4 b7 2d
                    08 63 fb 93 9b 98 2e 5f
                    39 7c c7 d1 64 fd da a9

 После прохода 0:
Блок 0000 [  0]: f8f9e84545db08f6
Блок 0000 [  1]: 9b073a5c87aa2d97
Блок 0000 [  2]: d1e868d75ca8d8e4
Блок 0000 [  3]: 349634174e1aebcc
...
Блок 0031 [124]: 975f596583745e30
```

```
Блок 0031 [125]: e349bdd7edeb3092
Блок 0031 [126]: b751a689b7a83659
Блок 0031 [127]: c570f2ab2a86cf00

 После прохода 1:
Блок 0000 [  0]: b2e4ddfcf76dc85a
Блок 0000 [  1]: 4ffd0626c89a2327
Блок 0000 [  2]: 4af1440fff212980
Блок 0000 [  3]: 1e77299c7408505b
...
Блок 0031 [124]: e4274fd675d1e1d6
Блок 0031 [125]: 903fffb7c4a14c98
Блок 0031 [126]: 7e5db55def471966
Блок 0031 [127]: 421b3c6e9555b79d

 После прохода 2:
Блок 0000 [  0]: af2a8bd8482c2f11
Блок 0000 [  1]: 785442294fa55e6d
Блок 0000 [  2]: 9256a768529a7f96
Блок 0000 [  3]: 25a1c1f5bb953766
...
Блок 0031 [124]: 68cf72fccc7112b9
Блок 0031 [125]: 91e8c6f8bb0ad70d
Блок 0031 [126]: 4f59c8bd65cbb765
Блок 0031 [127]: 71e436f035f30ed0
Тег: c8 14 d9 d1 dc 7f 37 aa
     13 f0 d7 7f 24 94 bd a1
     c8 de 6b 01 6d d3 88 d2
     99 52 a4 c4 67 2b 6c e8
```

5.3. Тестовые векторы Argon2id

```
========================================
Argon2id версии 19
========================================
Размер памяти: 32 Кбайт
Число проходов: 3
Число параллельных цепочек: 4
Размер тега: 32 байта
Пароль[32]: 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
            01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
Затравка[16]: 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02
Секрет[8]: 03 03 03 03 03 03 03 03
Связанные данные[12]: 04 04 04 04 04 04 04 04 04 04 04 04
Дайджест предварительного хэширования: 28 89 de 48 7e b4 2a e5 00 c0 00 7e d9 25 2f
 10 69 ea de c4 0d 57 65 b4 85 de 6d c2 43 7a 67 b8 54 6a 2f 0a
 cc 1a 08 82 db 8f cf 74 71 4b 47 2e 94 df 42 1a 5d a1 11 2f fa
 11 43 43 70 a1 e9 97

 После прохода 0:
Блок 0000 [  0]: 6b2e09f10671bd43
Блок 0000 [  1]: f69f5c27918a21be
Блок 0000 [  2]: dea7810ea41290e1
Блок 0000 [  3]: 6787f7171870f893
...
Блок 0031 [124]: 377fa81666dc7f2b
Блок 0031 [125]: 50e586398a9c39c8
Блок 0031 [126]: 6f732732a550924a
Блок 0031 [127]: 81f88b28683ea8e5

 После прохода 1:
Блок 0000 [  0]: 3653ec9d01583df9
Блок 0000 [  1]: 69ef53a72d1e1fd3
Блок 0000 [  2]: 35635631744ab54f
Блок 0000 [  3]: 599512e96a37ab6e
...
Блок 0031 [124]: 4d4b435cea35caa6
Блок 0031 [125]: c582210d99ad1359
Блок 0031 [126]: d087971b36fd6d77
Блок 0031 [127]: a55222a93754c692

 После прохода 2:
Блок 0000 [  0]: 942363968ce597a4
Блок 0000 [  1]: a22448c0bdad5760
Блок 0000 [  2]: a5f80662b6fa8748
Блок 0000 [  3]: a0f9b9ce392f719f
...
Блок 0031 [124]: d723359b485f509b
Блок 0031 [125]: cb78824f42375111
Блок 0031 [126]: 35bc8cc6e83b1875
Блок 0031 [127]: 0b012846a40f346a
Тег: 0d 64 0d f5 8d 78 76 6c 08 c0 37 a3 4a 8b 53 c9 d0
 1e f0 45 2d 75 b6 5e b5 25 20 e9 6b 01 e6 59
```

6. Взаимодействие с IANA

Этот документ не требует действий IANA.

7. Вопросы безопасности

7.1. Безопасность как хэш-функция и KDF

Уровни стойкости Argon2 к коллизиям и прообразам эквивалентны уровня базовой хэш-функции BLAKE2b. Для возникновения коллизиии требуется 2^(256) вводов данных, для нахождения прообраза - 2^(512) попыток ввода.

Безопасность KDF определяется длиной ключа и размером внутреннего состояния хэш-функции H'. Чтобы отличить вывод Argon2 с ключом от случайных данных, требуется по меньшей мере (2^(128),2^length(K)) вызовов BLAKE2b.

7.2. Атаки с компромиссом между расходом времени и памяти

Компромисс между расходом времени и памяти позволяет рассчитать интенсивно использующую память функцию с применением меньшего числа блоков за счёт большего числа вызовов внутренней функции сжатия. Преимущество таких атак измеряется коэффициентом сокращени произведения затраченного времени и занятой памяти, где расход памяти и дополнительные ядра функции сжатия вносят вклад в пространство (памяти), а время увеличивается для учёта повторного вычисления пропущенных блоков. Высокий коэффициент сокращения может ускорить поиск прообраза.

В наиболее известной атаке на Argon2i с 1 и 2 проходами (атака с малым размером хранилища), описанной в[CBS16], произведение времени и памяти (с использованием пикового расхода памяти) сокращено в 5 раз. Наиболее сильная атака на Argon2i с 3 и более проходами описана в [AB16] и коэффициент сокращения здесь является функцией от размера памяти и числа проходов (например, при расходе 1 Гбайт и 3 проходах коэффициент составляет 3, при 4 проходах - 2,5, при 6 - 2). При каждом удвоении размера памяти коэффициент возрастает примерно на 0,5. Для полного предотвращения описанных в [AB16] атак число проходов **должно** быть больше двоичного логарифма размера памяти минус 26. Асимптотически наиболее сильная атака на Argon2i с одним проходом описана в [BZ17], при этом преимущества атакующего ограничены сверху значением O(m^(0,233)), где m - число блоков. Эта атака является асимптотически оптимальной, поскольку в [BZ17] показано, что верхняя граница для любой атаки - это O(m^(0,25)).

Наиболее сильной атакой на Argon2d с t проходами является компромиссная атака с ранжированием (ranking trade-off attack), где коэффициент сокращения составляет 1.,3.

Наиболее сильную атаку на Argon2id можно организовать путём дополнения наиболее сильной атаки на Argon2i с 1 проходом наиболее сильной атакой на многопроходный Argon2d. Таким образом, наиболее сильной компромиссной атакой на Argon2id с одним проходом является сочетание атаки с малым объёмом хранилища (для первой половины памяти) и атаки с ранжированием (для второй половины), которая даёт коэффициент сокращения около 2,1. Наиболее сильной атакой на Argon2id с t проходами является компромиссная атака с ранжированием, где коэффициент сокращения имеет значение 1,33.

7.3. Безопасность для ограниченных по времени защитников

Узким местом систем с хэшированием паролей часто является не расход памяти, а задержка, вносимая функцией. Разумная система защиты будет максимально увеличивать расходы атак с перебором (brute-force) для злоумышленников, имеющий список хэшей, затравок и сведений о времени для фиксированного времени расчётов на машине защитника. Оценки стоимости в работе [AB16] показывают, что для Argon2i три прохода почти оптимальны для большинства разумных размеров памяти, для Argon2d и Argon2id один проход максимизирует расходы на атаку при постоянном времени у защитника.

7.4. Рекомендации

Argon2id с t=1 и 2 Гбайт памяти является **первым рекомендуемым** вариантом и предлагается в качестве принятой по умолчанию настройки во всех средах. Эта настройка защищена от атак по побочным каналам и максимизирует расходы злоумышленников на оборудование для перебора (brute-force hardware). Argon2id с t=3 и памятью 64 Мбайт является **вторым рекомендуемым** вариантом в качестве принятого по умолчанию для систем с ограниченной памятью.

8. Литература

8.1. Нормативные документы

[BLAKE2] Saarinen, M-J., Ed. and J-P. Aumasson, "The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)", RFC 7693, DOI 10.17487/RFC7693, November 2015, <https://www.rfc-editor.org/info/rfc7693>.

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

8.2. Дополнительная литература

[AB15] Biryukov, A. and D. Khovratovich, "Tradeoff Cryptanalysis of Memory-Hard Functions", ASIACRYPT 2015, DOI 10.1007/978-3-662-48800-3_26, December 2015, <https://eprint.iacr.org/2015/227.pdf>.

[AB16] Alwen, J. and J. Блокі, "Efficiently Computing Data-Independent Memory-Hard Functions", CRYPTO 2016, DOI 10.1007/978-3-662-53008-5_9, March 2016, <https://eprint.iacr.org/2016/115.pdf>.

[ARGON2] Biryukov, A., Dinu, D., and D. Khovratovich, "Argon2: the memory-hard function for password hashing and other applications", March 2017, <https://www.cryptolux.org/images/0/0d/Argon2.pdf>.

[ARGON2ESP] Biryukov, A., Dinu, D., and D. Khovratovich, "Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications", Euro SnP 2016, DOI 10.1109/EuroSP.2016.31, March 2016, <https://www.cryptolux.org/images/d/d0/Argon2ESP.pdf>.

[BZ17] Блокі, J. and S. Zhou, "On the Depth-Robustness and Cumulative Pebbling Cost of Argon2i", TCC 2017, DOI 10.1007/978-3-319-70500-2_15, May 2017, <https://eprint.iacr.org/2017/442.pdf>.

[CBS16] Boneh, D., Corrigan-Gibbs, H., and S. Schechter, "Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks", ASIACRYPT 2016, DOI 10.1007/978-3-662-53887-6_8, May 2017, <https://eprint.iacr.org/2016/027.pdf>.

[HARD] Alwen, J. and V. Serbinenko, "High Parallel Complexity Graphs and Memory-Hard Functions", STOC '15, DOI 10.1145/2746539.2746622, June 2015, <https://eprint.iacr.org/2014/238.pdf>.

Благодарности

Большое спасибо за подготовку и рецензирование документа Jean-Philippe Aumasson, Samuel Neves, Joel Alwen, Jeremiah Блокі, Bill Cox, Arnold Reinhold, Solar Designer, Russ Housley, Stanislav Smyshlyaev, Kenny Paterson, Alexey Melnikov, Gwynne Raskind.

Описанная в документе работа была выполнена до перехода Daniel Dinu в Intel (пока он учился в университете Люксембурга).

Адреса авторов

Alex Biryukov
University of Luxembourg
Email: alex.biryukov@uni.lu

Daniel Dinu
University of Luxembourg
Email: daniel.dinu@intel.com

Dmitry Khovratovich
ABDK Consulting
Email: khovratovich@gmail.com

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

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru