УДК 004.056.5 **DOI:10.30837/rt.2023.2.213.02**

С.О. КАНДІЙ

АНАЛІЗ ПРОЦЕСІВ ГЕНЕРАЦІЇ ПСЕВДОВИПАДКОВИХ ЧИСЕЛ В ЕП CRYSTALS-DILITHIUM

Вступ

Здебільшого під час аналізу електронних підписів (ЕП) у літературі вважається, що є певне ідеалізоване джерело випадковості і не існує атак, що використовують властивості цього джерела [13]. Проте на практиці джерело випадковості може бути неідеальним та містити вразливості.

ЕП Crystals-Dilithium є фіналістом у конкурсу NIST PQC з постквантової криптографії [8]. Ця робота присвячена аналізу процесів генерації випадкових чисел у еталонній реалізації ЕП Crystals-Dilithium.

ЕП Crystals-Dilithium використовує для генерації випадкових змінних shake128/256 [9] та (варіативно) генератор псевдовипадкових чисел (DRNG) на основі AES в режимі лічильника (CTR) [1, 8]. Останній є варіацією стандартизованого CTR_DRBG. Оскільки PRNG на основі AES у режимі лічильника (CTR) є популярним вибором для багатьох практичних застосувань, то для повноти картини у цій роботі проведено аналіз стандартизованої версії – генератора CTR_DRBG [1] з AES-256 у якості блочного шифру (надалі – AES_CTR_PRNG). Аналіз виконано відповідно до останньої версії стандарту AIS 31 [2], що регламентує вимоги до генераторів ПВП.

Додатково проведено аналіз безпеки shake128/256 для генерації випадкових поліномів у ЕП Crystals-Dilithium. Отримано практичні рекомендації щодо параметрів компіляції в залежності від умов використання.

Роботу організовано наступним чином. В розд. 1 наведено необхідні відомості з криптографії, наведено опис генераторів. В розд. 2 побудовано та в розд. 3 проаналізовано формальну модель для AES_CTR_PRNG. В розд. 4,5 наведено аналіз використання shake128/256 в ЕП Crystals-Dilithium.

1. Попередні відомості

1.1. Стандарт AIS 31

Стандарт AIS 31 класифікує генератори випадкових чисел наступним чином [2]:

- генератори псевдовипадкових чисел (PRNG). Усі безпечні PRNG діляться на три функціональні класи: DRG.2, DRG.3, DRG.4;
- фізичні генератори випадкових чисел (PTRNG). Усі безпечні PTRNG діляться на два функціональні класи: PTG.2, PTG.3;
- нефізичні генератори випадкових чисел (NPTRNG). Для NPTRNG стандарт визначає тільки один функціональний клас – NTG.1.

Кожен PRNG, згідно з AIS 31, може бути описано за допомогою кортежу параметрів $(S, S_{req}, R, A, I, \phi, \phi_{req}, \phi_0, \psi)$, де

- S – множина допустимих внутрішніх станів;
- S_{req} – множина допустимих (тимчасових) станів запитів;
- R – множина допустимих вихідних значень;
- A – множина допустимих додаткових вхідних значень;
- I – множина допустимих розмірів запитів;
- $\phi : S \times A \to S$ – функція переходу станів;
- $\phi_{req} : S \times A \to S_{req}$ – функція генерації стану запиту;

- $\phi_0 : S_{req} \times A \to S_{req}$ – функція переходу стану запиту;
- $\psi : S_{req} \to R$ – вихідна функція.

Вважається, що для обробки запиту на $p \in I$ біт генерується m внутрішніх випадкових чисел. Причому, виконується наступний псевдокод для кожного запиту з додатковими вхідними даними a та внутрішнім станом s:

$$
\begin{aligned}
& s_{req} = \phi_{req}(s, a) \\
& for(j = 1; m;\ j{+}{+}) \\
& \quad r_j = \psi(s_{req}) \\
& \quad s_{req} = \phi_0(s_{req}) \\
& s = \phi(s, a)
\end{aligned}
\quad .
$$

Процес ініціалізації PRNG описується кортежем $(SM, PS, S, \phi_{seed}, \phi_{reseed})$, де SM – множина допустимих строк ініціалізації; PS – множина допустимих строк персоналізації; S – множина допустимих внутрішніх станів; ϕ_{seed} – функція ініціалізації; ϕ_{reseed} – функція повторної ініціалізації.

Повний опис функціональних класів виходить за межі даної роботи, проте для подальшого аналізу необхідно описати вимоги до класу DRG.3.

Генератор псевдовипадкових чисел належить до класу DRG.3 у разі виконання наступних вимог:

DRG.3.1 – строка ініціалізації (seed), повинен бути отриманий з DRG.3 або DRG.4 сумісного PRNG або безпечного TRNG;

DRG.3.2 – між викликами процедури ініціалізації та повторної ініціалізації повинно відбутися не більше 2^{48} запитів на генерацію ПВП. Кожен запит повинен бути не більше 2^{19} біт;

DRG.3.3 – ефективний внутрішній стан (критична для безпеки частина внутрішнього стану) повинен мати не менше 252 біт ентропії;

DRG.3.4 – початковий ефективний внутрішній стан повинен мати не менше 250 біт ентропії (або 240 біт мінімальної ентропії);

DRG.3.5 – має бути обчислювально важко дізнатися наступне випадкове число, знаючи деяку кількість попередніх (forward secrecy);

DRG.3.6 – має бути обчислювально важко дізнатися попереднє випадкове число, знаючи деяку кількість наступних (backward secrecy);

DRG.3.7 – якщо поточний внутрішній стан дізнається зловмисник, то в нього не має бути змоги дізнатися попередній внутрішній стан (enhanced backward secrecy);

DRG.3.8– додаткові вхідні дані не повинні зменшувати безпеку;

DRG.3.9 – функція переходу станів ϕ та вихідна функція ψ мають бути криптографічними;

DRG.3.10 – мають бути свідчення того, що статистичні тести не зможуть відрізнити на практиці псевдовипадкову послідовність від випадкової.

1.2. Генератор ПВП CTR_DRBG

На рис. 1 наведено псевдокод генератора CTR_DRBG, що є частиною стандарту SP800-90A [1]. Зауважимо, що у межах цієї роботи розглядається варіант CTR_DRBG без функції виведення. Втім, аналіз може бути легко адаптований для цього варіанта за необхідності.

CTR-DRBG update

Require: $provided\_data, K, V$
Ensure: K, V
$temp \leftarrow \varepsilon$; $m \leftarrow \lceil (\kappa + \ell)/\ell \rceil$
For $j = 1, \ldots, m$
  $V \leftarrow (V + 1) \bmod 2^{\ell}$; $Z \leftarrow \mathsf{E}(K, V)$
  $temp \leftarrow temp \,\|\, Z$
$temp \leftarrow \text{left}(temp, (\kappa + \ell))$
$temp \leftarrow temp \oplus provided\_data$
$K \leftarrow \text{left}(temp, \kappa)$
$V \leftarrow \text{right}(temp, \ell)$
Return K, V

CTR-DRBG next

Require: $S = (K, V, cnt), \beta, addin$
Ensure: $R, S' = (K', V', cnt')$
1. If $cnt > reseed\_interval$
2. Return reseed_required
3. If $addin \neq \varepsilon$
4. If derivation function used then
5. $addin \leftarrow \mathsf{CTR\text{-}DRBG\_df}(addin, (\kappa + \ell))$
6. Else if $len(addin) < (\kappa + \ell)$ then
7. $addin \leftarrow addin \,\|\, 0^{(\kappa+\ell-len(addin))}$
8. $(K, V) \leftarrow \mathsf{update}(addin, K, V)$
9. Else $addin \leftarrow 0^{\kappa+\ell}$
10. $temp \leftarrow \varepsilon$; $n \leftarrow \lceil \beta/\ell \rceil$
11. For $j = 1, \ldots, n$
12. $V \leftarrow (V + 1) \bmod 2^{\ell}$; $r \leftarrow \mathsf{E}(K, V)$
13. $temp \leftarrow temp \,\|\, r$
14. $R \leftarrow \text{left}(temp, \beta)$
15. $(K', V') \leftarrow \mathsf{update}(addin, K, V)$
16. $cnt' \leftarrow cnt + 1$
17. Return $R, (K', V', cnt')$

Рис. 1. Генератор ПВП CTR_DRBG

У межах роботи розглядається варіант цього генератора для AES256 [3] (хоча більша частина аналізу підходить до довільного блочного шифру). Позначимо як $c = AES256(m, key)$ шифротекст AES-256 повідомлення m на ключі key. Режим роботи лічильника (CTR) для AES-256 наведено на рис. 2.

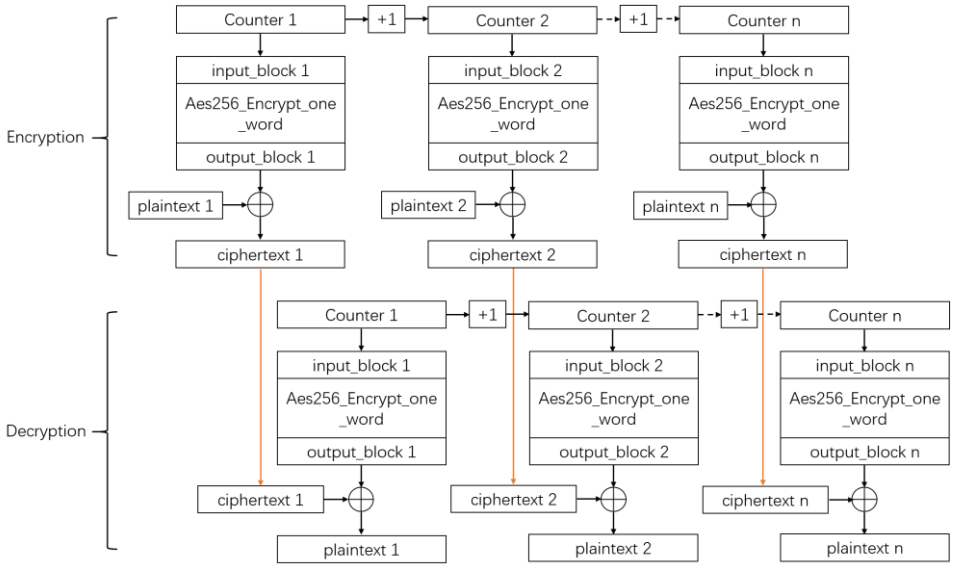

Рис. 2. Режим лічильника (CTR)

Для режиму лічильника введемо позначення $c = AES_{CTR}^{IV}(m, key)$. Передбачається, що поточний стан v при шифруванні має форму $v = v_{info} \,\|\, ctr$, де v_{info} – довільна інформаційна частина, ctr – лічильник.

1.3. shake128/256

shake128 та shake256, за визначенням, є XOF (eXtendable output function). XOF є узагальненням геш функцій [9]. Криптографічна геш функція, за визначенням, є стискаючим відображенням, що відображає строки довільного розміру у множину строк (меншого) фіксова-

ного розміру. XOF робить відображення строк довільного розміру у множину строк довільного розміру, що заданий аргументом функції. При цьому зберігаються властивості криптографічних геш функцій. Це дає змогу використовувати XOF у якості геш функції, потокового шифру, DRNG, MAC-коду та інших симетричних криптопримітивів!

В основі XOF shake128/256 лежить sponge-конструкція, що зображена на рис. 3.

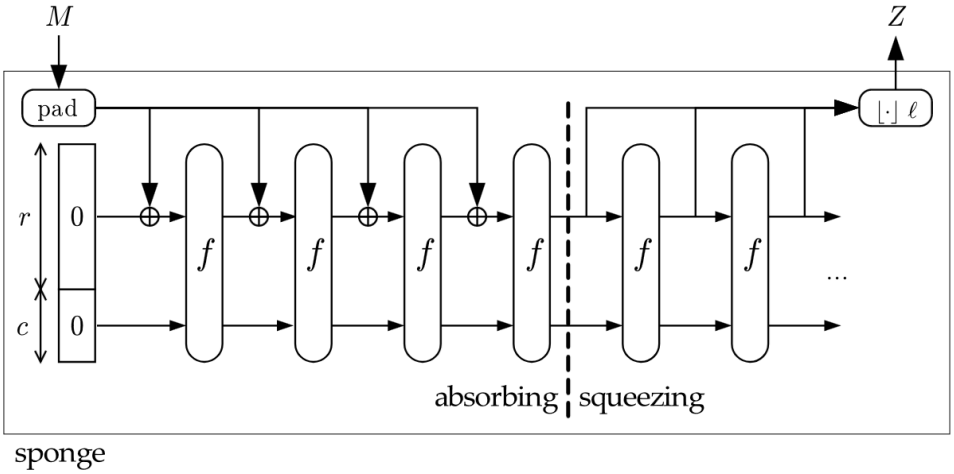

Рис. 3. Sponge-конструкція

Sponge-конструкція використовує випадкову перестановку $f:\{0,1\}^{b=r+c}\to\{0,1\}^b$, де b – довжина входу, r – бітрейт, c – ємність. Робота Sponge-конструкції складається з двох фаз – поглинання (absorbing) та стиснення (squeezing).

На фазі поглинання Sponge-конструкція застосовує до повідомлення M схему відступу pad, розбиває повідомлення на r-бітні блоки та побітово складає останні з поточним станом.

На фазі стиснення Sponge-конструкція ітераційно повертає перші r біт внутрішнього стану та обрізає отриману послідовність до ℓ біт, як зображено на рис. 3.

Введемо позначення $Z = SPONGE[f, pad, r](M, \ell)$ для Sponge-конструкції, що використовує випадкову перестановку f, схему відступу pad, має бітрейт r та повертає ℓ-бітну бітову строку Z для повідомлення M.

XOF shake128/256 є Sponge-конструкціями. Їх параметри зведено у табл. 1.

Таблиця 1

Параметри shake128/256

	f	pad	r	c
shake128	KECCAK-p[1600, 24]	pad10*1	1344	256
shake256			1088	512

Повна специфікація KECCAK-p[1600, 24] та pad10*1 наведена в стандарті FIPS 202 [2]. Стандарт FIPS 202 доволі добре проаналізований міжнародною спільнотою, тож надалі вважається, що не існує ефективних атак, що використовують властивості f чи pad10*1.

На Sponge-конструкцію існують загальні атаки, що використовують саме її властивості. Надалі останні c біт стану називатимемо внутрішнім станом. Загальні атаки можна класифікувати як атаки:

на пошук внутрішніх колізій (колізій внутрішнього стану);

пошук шляху до внутрішнього стану;

пошук циклів;

відновлення внутрішнього стану.

Загальні атаки детально проаналізовано у [11]. Спираючись на [11], розглянемо кожен клас загальних атак більш детально.

Відповідно до визначень, внутрішній стан s є бітовою строкою довжини $b=r+c$. Його можливо представити як конкатенацію бітових строк s_{outer} та s_{inner} довжини r та c біт відповідно.

Сутність атак на пошук внутрішніх колізій полягає у тому, щоб знайти два повідомлення M_1, M_2, для яких стани (після фази поглинання) s^1, s^2 матимуть $s^1_{inner} = s^2_{inner}$. Позначимо цю подію як W_{IC}. Нехай N – кількість звернень до перестановки f. Якщо f обчислювально складно відрізнити від випадкової перестановки, то з простого комбінаторного аналізу [4, розд.5.4] випливає оцінка атаки

$$\Pr(W_{IC}) \approx \frac{N(N+1)}{2^{c+1}} - \frac{N(N-1)}{2^{r+c+1}}. \quad (1)$$

Атаки на пошук шляху до внутрішнього стану фактично полягають у знаходженні повідомлення M для відомого внутрішнього стану $s^1 = s^1_{outer} \| s^1_{inner}$, для якого $s^2 = s^2_{outer} \| (s^2_{inner} = s^1_{inner}) = absorb(M)$. Позначимо цю подію як W_{Path}. Якщо f обчислювально складно відрізнити від випадкової перестановки, то з простого комбінаторного аналізу [4, розд. 5.5] випливає оцінка

$$\Pr(W_{Path}) \approx \frac{N(N+4)}{2^{c+2}} - \frac{N^2}{2^{r+c+2}}. \quad (2)$$

Атака на пошук циклів полягає у знаходженні повідомлення M, для якого у послідовності проміжних станів $s^1, s^2, \ldots$ існують такі i, j, що $s^i = s^j$. Позначимо цю подію як W_{Cycle}. Якщо f обчислювально складно відрізнити від випадкової перестановки, то ймовірність такої події становить

$$\Pr(W_{Cycle}) \approx \frac{N}{2^{r+c}}. \quad (3)$$

Атаки на відновлення внутрішнього стану є найбільш актуальними для аналізу генерації випадкових чисел у реалізаціях ЕП. Сутність атак на відновлення стану полягає для заданої бітової строки Z у знаходженні s, для якого виконується $Z = squeeze(s, |Z|)$. Позначимо цю подію як W_{SR}.

Аналіз цієї атаки є дещо складнішим за раніше розглянуті атаки. Оцінки безпеки отримано в роботах [11, 12]. Введемо необхідну термінологію.

Нехай $|Z| = mr$. Строку Z, за визначенням, можливо розбити на послідовність блоків $Z_0, Z_1, \ldots, Z_{m-1}$. Прямим розбиттям блоків $B_f(Z)$ є множина підмножин індексів $i, 0 \le i < m-1$. В одну підмножину входять індекси $i_1, i_2, \ldots$, для яких значення блоків є однаковими $Z_{i_1} = Z_{i_2} = \ldots$. Прямою кратністю $m_f(Z,r)$ є потужність найбільшої підмножини у $B_f(Z)$. Зворотне розбиття блоків $B_b(Z)$ і зворотна кратність $m_b(Z,r)$ визначені аналогічно, тільки для індексів у діапазоні $0 < i \le m-1$. Повна кратність визначена як

$$m(Z,r) = \max\{m_f(Z,r), m_b(Z,r)\}. \quad (4)$$

Згідно з [4, 5] для заданого $Z = squeeze(s, |Z|)$ ймовірність знаходження s складає

$$\Pr(W_{SR} \mid \mathrm{Z}) \approx \Pr(W_{SR}) \approx \frac{m(Z,r)\,\mathrm{N}}{2^c}. \quad (5)$$

1.4. Відомості з теоретичної криптографії

Псевдовипадкові функції (PRF) є функціями, які обчислювально важко відрізнити від випадкових функцій. Псевдовипадкові перестановки (PRP), відповідно, є перестановками, які важко відрізнити від дійсно випадкових перестановок. Точні формальні визначення можливо знайти, наприклад, у [4]. PRNG, PRF, PRP тісно пов'язані між собою. У межах роботи нас цікавлять два результати, що отримані у [5, 6].

Т е о р е м а 1 . Для всіх алгоритмів, що роблять не більше M запитів на обчислення функції в точці та працюють за час, не більший за t, для будь-якої родини перестановок P довжини L має місце твердження

$$Adv^{P}_{PRF}(t,M) \le Adv^{P}_{PRP}(t,M) + M^2 2^{-L-1}.$$

Тут і надалі Adv позначає перевагу у відрізненні відповідного псевдовипадкового об'єкта від випадкового. З теореми видно, що будь-яка псевдовипадкова перестановка може бути розглянута як псевдовипадкова функція.

Сімейство бієктивних псевдовипадкових перестановок (функцій) може бути розглянуто як блочний шифр. Наступна теорема встановлює зв'язок між захищеністю шифру (у режимі лічильника) від атак з підібраним відкритим текстом (CPA).

Т е о р е м а 2 . Припустимо, що E є сімейством псевдовипадкових функцій з розміром образів L. Тоді, для будь-яких t та M має місце

$$Adv^{CPA}_{CTR[E]}(t,M) \le 2 \cdot Adv^{PRF}_{E}(t,M).$$

Тут $CTR[E]$ позначає блочний шифр E у режимі лічильника.

2. Побудова формальної моделі

У цьому розділі будуть використовуватися дві допоміжні функції - $rm(str, bitlen)$ та $lm(str, bitlen)$, які обрізають бітову строку до довжини $bitlen$ справа та зліва. Функція $len(\cdot)$ повертає довжину строки в бітах. $seedlen$ – довжина строки ініціалізації.

Для аналізу AES_CTR_PRNG побудуємо формальний опис у вигляді кортежу $(S, S_{req}, R, A, I, \phi, \phi_{req}, \phi_0, \psi)$. Нехай $S_K = \{0,1\}^{256}$ – множина усіх допустимих ключів та $S_B = \{0,1\}^{128}$ – множина усіх допустимих шифротекстів/відкритих текстів. Тоді, множину усіх допустимих внутрішніх станів можливо описати як

$$S = S_B \times S_K \times \square_{2^{48}}. \quad (6)$$

Кожен допустимий внутрішній стан задається трійкою (v, key, rc), де v – значення лічильника, key – ключ шифрування, rc – кількість зроблених запитів до генератора ПВП.

Ефективним внутрішнім станом є друга компонента внутрішнього стану – ключ шифрування key. Внутрішній стан запиту можливо визначити як

$$S_{req} = S_B \times S_K. \quad (7)$$

Відповідно до попередніх домовленостей маємо визначення множини додаткових входів та допустимих вихідних значень:

$$A = \{0,1\}^{\le seedlen}, \quad (8)$$

$$R = S_B . \tag{9}$$

Для завершення опису наведемо декомпозицію генератору ПВП на функції $\phi, \phi_{req}, \phi_0, \psi$. Вихідна функція $\psi : S_{req} \to R$ задається як

$$\psi(s_{req} = (v, key)) = AES256(v, key) . \tag{10}$$

Функція генерації внутрішнього стану запиту $\phi_{req} : S \times A \to S_{req}$ задається як

$$\begin{aligned} &\phi_{req}(s = (v, key, rc), a) = \\ &rm(f(a, key, v), 256), lm(f(a, key, v), 128) = (v_{req}, key_{req}) \end{aligned}, \tag{11}$$

де f визначена наступним чином:

$$f(a, key, v) = AES^{v}_{CTR}(a \,||\, 0^{seedlen - len(a)}, key) . \tag{12}$$

Функція оновлення внутрішнього стану запиту $\phi_0 : S_{req} \times A \to S_{req}$ визначена наступним чином:

$$\phi_0(v = v_{info} \,||\, ctr, key, a) = (v_{info} \,||\, ctr + 1, key) . \tag{13}$$

Для опису функції оновлення внутрішнього стану $\phi : S \times A \to S$ зробимо наступну декомпозицію:

$$\phi = \phi_A \circ (\phi_B \times id) , \tag{14}$$

де функція $\phi_A : S \times A \to S$ визначена як

$$\begin{aligned} &\phi_A(s = (v, key, rc), a) = \\ &rm(f(a, key), 256), lm(f(a, key), 128), rc = (v_{new}, key_{new}, rc) \end{aligned}. \tag{15}$$

І функція $\phi_B : S \times A \to S$ визначена як

$$\begin{aligned} &\phi_B(s = (v, key, rc), a) = \\ &rm(f(a, key), 256), lm(f(a, key), 128), rc = (v_{new}, key_{new}, rc + 1) \end{aligned}. \tag{16}$$

Також, для аналізу процесів ініціалізації наведемо формальний опис у термінах кортежу $(SM, PS, S, \phi_{seed}, \phi_{reseed})$. Множина допустимих внутрішніх станів була визначена раніше (формула (1)). Множини допустимих строк ініціалізації (SM) та персоналізуючих строк (PS) задані наступним чином:

$$SM = \{0,1\}^{seedlen} , \tag{17}$$

$$PS = \{0,1\}^{\le seedlen} . \tag{18}$$

Функції $\phi_{seed} : SM \times PS \to S$ та $\phi_{reseed} : S \times SM \times PS \to S$ задані наступним чином:

$$\begin{aligned} &\phi_{seed}(seed, personal) = \\ &rm(f(seed \oplus personal, 0^{256}), 256), \\ &lm(f(seed \oplus personal, 0^{256}), 128), 1 = (v, key, rc) \end{aligned} \tag{19}$$

$$\begin{aligned}&\phi_{reseed}(s=(v,key),seed,personal)=\\&rm(f(seed\oplus personal,key,v),256),\\&lm(f(seed\oplus personal,key,v),128),1=(v,key,rc)\end{aligned}. \quad (20)$$

3. Аналіз формальної моделі

Покажемо, що AES_CTR_PRNG задовільняє вимогам класу безпеки DRG.3.

Властивість DRG.3.2 (обмеження на кількість біт за один запит та кількість біт між ініціалізаціями) виконується за визначенням AES_CTR_PRNG.

Властивість DRG3.3 (ефективний внутрішній стан повинен мати щонайменше 252 біта ентропії) виконується, оскільки ефективним внутрішнім станом є секретний ключ *key* та вектор ініціалізації v, які отримані як результат застосування CTR[AES256] до строки ініціалізації. З властивостей AES256 випливає, що отримані бітові строки будуть мати достатньо велику ентропію. Властивість DRG3.4 (початковий ефективний внутрішній стан повинен мати щонайменше 250 біт ентропії) виконується з тих самих причин.

Доведемо, що AES_CTR_PRNG задовільняє властивості DRG.3.5 – знання випадкових чисел $r_i,...,r_j$, що отримані за допомогою AES_CTR_PRNG, не дозволяє визначити r_{j+1}. Для DRNG це твердження є еквівалентним наступному твердженню: послідовність $r_i,...,r_j,r_{j+1}$ неможливо відрізнити від послідовності дійсно випадкових чисел.

Припустимо, що задано деякий алгоритм D, що робить не більше q запитів на генерацію випадкових чисел та може відрізнити послідовність $r_i,...,r_j,r_{j+1}$ від послідовності дійсно випадкових чисел. Розглянемо гру $Game^b_{RNG}(D,q)$ між тестувальником та супротивником D, яка для параметра $b\in\{0,1\}$ визначена наступним чином. Тестувальник на початку гри володіє станом AES_CTR_PRNG - $S_0=(v,key,rc)$. Тестувальник генерує бітові послідовності $r^0_0,r^0_1,...,r^1_M$ та $r^1_0,r^1_1,...,r^1_M$ для деякого M, де послідовність $r^0_0,r^0_1,...,r^1_M$ отримана за допомогою AES_CTR_PRNG, а $r^1_0,r^1_1,...,r^1_M$ є дійсно випадковою послідовністю Результатом гри є рішення супротивника $d\leftarrow D(r^b_0,r^b_1,...,r^b_M)$. Якщо супротивник вважає, що послідовність є дійсно випадковою, то повертає 1, інакше повертає 0. Перевага супротивника D у $Game^b_{RNG}(D)$ визначена наступним чином:

$$\begin{aligned}&Adv_{AES\_CTR\_PRNG}(D,q)=\\&|\Pr(Game^1_{PRG}(D,q)=1)-\Pr(Game^0_{PRG}(D,q)=1)|\end{aligned}. \quad (21)$$

Ця нотація узагальнюється для довільного алгоритму, що працює не довше за заданий час t, наступним чином:

$$Adv_{AES\_CTR\_PRNG}(t,q)=\max_D\{Adv_{AES\_CTR\_PRNG}(D,q)\}, \quad (22)$$

де D береться з множини усіх алгоритмів, що працюють менше за заданий час t.

З визначення функцій ψ,ϕ_0 випливає, що послідовність $r_i,...,r_j$ має вигляд $AES256(v_{info}\,||\,ctr+i,key)$, $AES256(v_{info}\,||\,ctr+i+1,key)$,..., $AES256(v_{info}\,||\,ctr+j,key)$, що є визначенням режиму лічильника для AES. Отже, за теореми 2 випливає:

$$\begin{aligned}&Adv^{CTR[AES256]}_{CPA}(t,M)\le Adv^{AES256}_{PRF}(t,M)\\&\le 2\cdot(Adv^{AES256}_{PRP}(t,M)+M^2 2^{-L-1})\end{aligned}. \quad (23)$$

Відповідно маємо оцінку безпеки для одного запита:

$$Adv_{AES\_CTR\_PRNG}(t,1) \le Adv_{ROR-CPA}^{CTR[AES256]}(t,M). \quad (24)$$

Оскільки AES256 є добре вивченим шифром, то можемо вважати, що $Adv_{PRP}^{AES256}(t,M)=0$, звідки випливає оцінка

$$Adv_{AES\_CTR\_PRNG}(t,1) \le \frac{M^2}{2^{L+1}}. \quad (25)$$

Припустимо, що супротивник є обчислювально необмеженим. Верхню оцінку для $Adv_{AES\_CTR\_PRNG}(q)$ можливо отримати, якщо припустити, що ймовірність перемогти у грі $Game_{RNG}^{b}(D,q)$ не залежить від кількості спроб, тоді маємо:

$$\begin{aligned} &Adv_{AES\_CTR\_PRNG}(\cdot,q) \le 1-(1-Adv_{AES\_CTR\_PRNG}(\cdot,1))^q \\ &\approx q \cdot Adv_{AES\_CTR\_PRNG}(\cdot,1) \end{aligned} \quad (26)$$

Звідки випливає верхня оцінка

$$Adv_{AES\_CTR\_PRNG}(\cdot,q) \le \frac{qM^2}{2^{L+1}}. \quad (27)$$

Оскільки максимальна кількість блоків за один запит складає $2^{19}/2^7 = 2^{12}$, то ймовірність знаходження r_{j+1} з $r_i,...,r_j$ є меншою за $q \cdot 2^{-104}$. Враховуючи те, що після кожного запиту ключ шифрування змінюється, то можливо вважати $q=1$, оскільки ефективних багатоключових атак на AES256 невідомо.

Властивість DRG.3.6 (з $r_i,...,r_j$ важко отримати r_{j-1}) аналогічно доводиться для AES_CTR_PRNG. Оскільки послідовність псевдовипадкових бітів важко відрізнити від дійсно випадкових бітів, то важко отримати r_{j-1}.

Властивість DRG.3.7 полягає у тому, що якщо супротивник дізнався поточний стан S_i, то він не зможе дізнатися попередній стан S_{i-1}. Ця властивість випливає з визначення функції ϕ_B. Оскільки у кінці кожного запиту новий ключ шифрування отримується як результат криптографічної операції на старому ключі шифрування, то, якщо супротивник дізнається S_{i-1}, він порушить безпеку AES256, оскільки зможе розшифрувати шифротекст, не знаючи ключа шифрування, на якому останній був отриманий.

Властивість DRG.3.8 полягає у тому, що додаткові вхідні дані не мають зменшувати безпеку генератора, якщо зловмисник може контролювати їх. У AES_CTR_PRNG додаткові вхідні дані фактично зашифровуються AES256 у режимі лічильника. Якщо супротивник може за допомогою зміни вхідних даних впливати на безпеку генератора, то це означало б вдалу атаку з підібраними відкритими текстами на AES256 у режимі лічильника, що вважається неможливим згідно з модельними припущеннями безпеки AES256.

Властивість DRG.3.9 вимагає, щоб функції ϕ,ψ були криптографічними. Причому, функція ϕ має бути односторонньою. З визначень функцій ϕ,ψ видно, що це виконується, оскільки результатом роботи обох функцій є застосування AES256 до вхідних аргументів.

Властивість DRG.3.10 вимагає, щоб були сильні свідчення того, що статистичні тести не можуть відрізнити псевдовипадкові біти від дійсно випадкових. Вище було показано, що

ймовірність цього складає не більше $q \cdot 2^{-104}$, де q – кількість запитів на вироблення випадкових бітів.

Остання вимога, яку необхідно розглянути є DRG.3.1, вимагає, щоб матеріал для ініціалізації генератора був отриманий за допомогою TRNG (допустимі класи безпеки PTG.2,PTG.3,NTG.1) або іншого PRNG класу DRG.3. Виконання цієї вимоги залежить від конкретної реалізації. Наприклад, генератор випадкових чисел /dev/random у ядрах Linux версії 5.6–5.17 задовільняє вимогам DRG.3 і може бути використаний для отримання матеріалу для ініціалізації.

Тож, усі вимоги виконано.

4. Генерація ПВП у Crystals-Dilithium

Розглянемо детально процеси генерації випадкових об'єктів в реалізації ЕП Crystals-Dilithium. Для зручності наведемо псевдовод Crystals-Dilithium з специфікації [8] на рис. 4.

Gen

$\rho \leftarrow \{0,1\}^{256}$
$K \leftarrow \{0,1\}^{256}$
$(\mathbf{s}_1, \mathbf{s}_2) \leftarrow S_\eta^\ell \times S_\eta^k$
$\mathbf{A} \in R_q^{k \times \ell} := \mathsf{ExpandA}(\rho)$ // $\mathbf{A}$ is stored in NTT Domain Representation
$\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
$(\mathbf{t}_1, \mathbf{t}_0) := \mathsf{Power2Round}_q(\mathbf{t}, d)$
$tr \in \{0,1\}^{384} := \mathsf{CRH}(\rho \parallel \mathbf{t}_1)$
return $(pk = (\rho, \mathbf{t}_1), sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0))$

Sign(sk, M)

$\mathbf{A} \in R_q^{k \times \ell} := \mathsf{ExpandA}(\rho)$ // $\mathbf{A}$ is stored in NTT Domain Representation
$\mu \in \{0,1\}^{384} := \mathsf{CRH}(tr \parallel M)$
$\kappa := 0$, $(\mathbf{z}, \mathbf{h}) := \perp$
while $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
  $\mathbf{y} \in S_{\gamma_1 - 1}^\ell := \mathsf{ExpandMask}(K \parallel \mu \parallel \kappa)$
  $\mathbf{w} := \mathbf{A}\mathbf{y}$
  $\mathbf{w}_1 := \mathsf{HighBits}_q(\mathbf{w}, 2\gamma_2)$
  $c \in B_{60} := \mathsf{H}(\mu \parallel \mathbf{w}_1)$
  $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$
  $(\mathbf{r}_1, \mathbf{r}_0) := \mathsf{Decompose}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)$
  if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$ or $\mathbf{r}_1 \neq \mathbf{w}_1$, then $(\mathbf{z}, \mathbf{h}) := \perp$
  else
    $\mathbf{h} := \mathsf{MakeHint}_q(-c\mathbf{t}_0, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0, 2\gamma_2)$
    if $\|c\mathbf{t}_0\|_\infty \geq \gamma_2$ **or** the # of 1's in $\mathbf{h}$ is greater than ω, then $(\mathbf{z}, \mathbf{h}) := \perp$
  $\kappa := \kappa + 1$
return $\sigma = (\mathbf{z}, \mathbf{h}, c)$

Verify$(pk, M, \sigma = (\mathbf{z}, \mathbf{h}, c))$

$\mathbf{A} \in R_q^{k \times \ell} := \mathsf{ExpandA}(\rho)$ // $\mathbf{A}$ is stored in NTT Domain Representation
$\mu \in \{0,1\}^{384} := \mathsf{CRH}(\mathsf{CRH}(\rho \parallel \mathbf{t}_1) \parallel M)$
$\mathbf{w}_1' := \mathsf{UseHint}_q(\mathbf{h}, \mathbf{A}\mathbf{z} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2)$
return $[\![\|\mathbf{z}\|_\infty < \gamma_1 - \beta]\!]$ **and** $[\![c = \mathsf{H}(\mu \parallel \mathbf{w}_1')]\!]$ **and** $[\![\#$ of 1's in $\mathbf{h}$ is $\leq \omega]\!]$

Рис. 4. Псевдокод ЕП Crystals-Dilithium

4.1. Генерація ключової пари

Відповідно до специфікації виконується наступна послідовність викликів до засобів генерації псевдовипадкових послідовностей:

- генерується випадкова строка seed за допомогою функції randombytes, що робить запит до криптографічного API Windows або до Linux RNG (в залежності від операційної системи). Надалі вважатимемо, що криптографічне API задовольняє необхідному рівню безпеки;
- обчислюється rho||rhoprime||key = shake256(seed);

- за допомогою генератора ПВП stream128 (буде розглянуто далі) та rho генерується матриця поліномів A. Кожен поліном генерується з унікальним nounce, який є його індексом у матриці;
- За допомогою генератора ПВП stream256 (буде розглянуто далі) генеруються вектори поліномів s_1, s_2. На відміну від поліномів у матриці, до поліномів додатково застосовується вибірка з відхиленням;
- shake256 в кінці генерації використовується як криптографічна геш функція: tr=shake256(rho||t1).

4.2. Вироблення підпису

Відповідно до специфікації виконується наступна послідовність викликів до засобів генерації псевдовипадкових послідовностей:

- shake256 використовується як криптографічна геш функція: mu=shake256(m||tr);
- shake256 використовується як криптографічна геш функція: rhoprime=shake256(key||mu), як і при генерації ключів;
- за допомогою генератора ПВП stream256 та rho генерується матриця поліномів A, як і при генерації ключів;
- за допомогою генератора ПВП stream256 генерується вектор поліномів y. Зауважимо, що в реалізації також підтримується режим, коли вектор генерується на основі випадкового значення з функції randombytes, що робить запит до криптографічного API. Ввімкнути цей режим можливо параметром компіляції DILITHIUM_RANDOMIZED_SIGNING;
- обчислюється c_seed для полінома c як shake256(mu||w1) та безпосередньо поліном c з shake256(c_seed).

4.3. Перевірка підпису

Відповідно до специфікації виконується наступна послідовність викликів до засобів генерації псевдовипадкових послідовностей:

- обчислення shake256(shake256(rho|| t), M);
- обчислення c2 з відновленого вектора поліномів w, аналогічно до обчислення оригінального полінома.

Для реалізації stream128/256 в залежності від параметрів компіляції існує два варіанти:

- використання shake128/256. Генерація гамми відбувається аналогічно до попередніх використань;
- використання AES128/256 в режимі лічильника. Генерація відбувається згідно з [3]. Цей генератор ПВП вже був проаналізований в [1]. Щоб ввімкнути використання цього варіанта, необхідно встановити параметр компіляції DILITHIUM_USE_AES.

5. Аналіз генерації випадкових чисел в ЕП Crystals-Dilithium

З опису можна зробити декілька висновків:

- в ЕП основним засобом генерації є shake256, проте для генерації матриці A використовується shake128. Це пов’язано з оптимізацією часу виконання. Матриця A є публічною, і оскільки Module-LWE є безпечним «у середньому» для випадкових матриць A, то для безпеки необхідно лише щоб у матриці A не було явно заданої алгебраїчної структури, що може зменшити складність Module-LWE. Тому використання shake128 не зменшує безпеки;
- усю роботу з випадковими числами можливо представити у вигляді графа. На рис. 5 зображено граф обчислень. Публічно відомі змінні позначено зеленим кольором, таємні змінні позначено червоним кольором. Цікавим є те, що у ЕП Crystals-Dilithium вироблення підпису є детермінованим. Випадковий вектор y генерується з випадковості у секретному ключі. У класичному випадку вектор y повинен генеруватися з незалежного джерела випадковості.

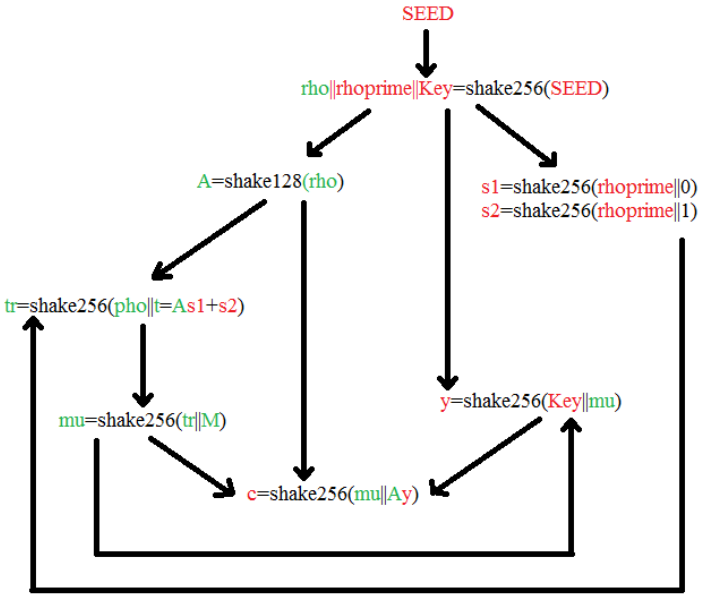

Рис. 5. Граф обчислень

Для змінних tr та mu shake256 виступає у якості криптографічної геш функції. Для shake256 вартість пошуку колізії складає $\min(d/2, 256)$, де d – довжина вихідної послідовності. Для обох випадків для параметрів Crystals-Dilithium маємо $\min(384/2, 256) = 192$ для всіх рівнів безпеки в специфікації ЕП.

У всіх інших випадках shake256 можливо розглядати як PRNG. Для s_1, s_2, y є невідомими як аргумент shake256, так і вихідне значення. Їх аргументи залежать від результату роботи shake256(seed), для якого частково є відомим значення вихідних даних (rho), отже, у цьому випадку задача криптоаналізу зводиться до атаки та пошуку внутрішнього стану. Стійкість до цієї атаки описується формулою (6). Оскільки r є доволі великим, то $m(Z, r) \approx 1$ і маємо оцінку:

$$\Pr(seed \mid rho) \approx \frac{\mathrm{N}}{2^{256}}, \tag{28}$$

де N – кількість запитів супротивника до функції стискання shake256. Тож, складність такої атаки фактично близька до перебору. Важливо також зауважити, що в результаті обчислень маємо послідовність

$$\text{Shake256(shake256(…shake(256(…)…)))}$$

У загальному випадку можуть існувати атаки саме на таку конструкцію, проте аналіз таких атак виходить за межі даної роботи.

Висновки

У роботі показано, що AES_CTR_PRNG задовільняє вимогам функціонального класу DRG.3. Проте, варто зауважити, останні дослідження [7] показують, що AES_CTR_PRNG є вразливим до атак на реалізацію у багатьох випадках. Тому, перед використанням AES_CTR_PRNG необхідно детально дослідити середовище виконання.

Окремо варто підкреслити, що модель безпеки AIS 31 передбачає використання незалежних та рівномірно розподілених випадкових змінних для ініціалізації генератора. У випадку AES_CTR_PRNG це є особливо критичним, оскільки це дає змогу зловмиснику за певних умов реалізовувати атаки на зв'язаних ключах.

Використання XOF shake256 є загальноприйнятим прийомом для реалізації асиметричної криптографії через її гнучкість та гарну безпеку.

Оскільки генератор на основі AES більш вразливий до атак по побічним каналам, то не рекомендовано використовувати флаг DILITHIUM_USE_AES, якщо немає додаткових гарантій захисту машини від атак по побічним каналам.

Якщо є довіра до криптографічного API операційної системи, то рекомендовано використовувати флаг компіляції DILITHIUM_RANDOMIZED_SIGNING. У системах з низькою довірою до криптографічного API операційної системи можна використовувати варіант з детермінованим виробленням підпису.

Список літератури:

1. NIST SP 800-90A. Recomendation for Random Number Generation Using Deterministic Random Bit Generation, June 2015.
2. BSI AIS 31. A Proposal for Functionality Classes for Random Number Generators, September 2022
3. FIPS 197. Advanced Encryption Standart, 2001.
4. Introduction to Modern Cryptography: Principles and Protocols / Katz, Jonathan; Lindell, Yehuda // Chapman & Hall/CRC Cryptography and Network Security Series, CRC Press, 2014.
5. Bellare M, Desai A., Jokipii E., Rogaway P. A concrete security treatment of symmetric encryption. FOCS, 1997.
6. M. Campagna. Security bounds for the NIST codebook-based deterministic random bit generator. Cryptology ePrint Archive. URL: https://eprint.iacr.org/2006/379
7. Hoang V., Shen Y. Security Analysis of NIST CTR-DRBG. Cryptology ePrint Archive. URL: https://eprint.iacr.org/2020/619
8. Lyubachevsky V., Ducas L., Kiltz E. CRYSTALS-Dilithium Techn. rep. NIST, 2017. [Electronic resource]. – Access mode: https://pq-crystals.org/dilithium/index.shtml
9. FIPS 202. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, 2015.
10. Goldreich O. Foundations of Cryptography: Volume 2. Cambridge University Press, 2000. 392 p.
11. Bertoni G., Daemen J., Peeters M., Van Assche G. Cryptographic sponge functions. URL: https://keccak.team/files/CSF-0.1.pdf
12. Sponge-based pseudo-random number generators, CHES (S. Mangard and F.-X. Standaert, eds.), Lecture Notes in Computer Science, vol. 6225, Springer, 2010, pp. 33–47.
13. Горбенко І. Д., Горбенко Ю. І. Прикладна криптологія. Теорія. Практика. Застосування: монографія. Харків : Форт, 2012. 880 с.

Надійшла до редколегії 10.06.2023

Відомості про автора:

Кандій Сергій Олегович – Харківський національний університет імені В. Н. Каразіна, аспірант кафедри безпеки інформаційних систем і технологій, факультет комп'ютерних наук; АТ «Інститут Інформаційних Технологій», технік-конструктор, Україна; e-mail: sergeykandy@gmail.com; ORCID: https://orcid.org/0000-0003-0552-8341