GraFPE – bezpieczny schemat szyfrowania zachowujący format danych*

Filip Zagórski[1]

Oktawave

Abstract. Schemat szyfrowania zachowujący format danych (format-preserving encryption (FPE)) jest takim rodzajem szyfrowania, że kryptogramy mają ten sam format co teksty jawne. FPE najczęściej wykorzystuje się w systemach, w których chce się wprowadzić szyfrowanie danych, ale z uwagi na złożoność systemu nie jest wskazana modyfikacja np. definicji pól w bazie danych. Za pomocą schematów FPE szyfruje się numery kart kredytowych, numery identyfikacyjne (takie jak np. pesel), ale można też zaszyfrować dowolne wyrażenie regularne.

Większość schematów FPE bazuje na jednym z następujących podejść:

1. projektuje się efektywny schemat tasowania, który jest oblivious shuffle (tj. można obliczyć trajektorię pojedyńczego elementu bez obliczania trajektorii pozostałych elementów). Przykładem takich schematów są np. Thorp-shuffle, Swap-or-not, Mix-and-cut, Cycle-slicer.
2. wykorzystuje się sieć Feistela obliczaną nad liczbami całkowitymi. To podejście zastosowano m. in. w standardach NIST'u FF1 i FF3.

W tej pracy opisujemy inne podejście. Schemat wykorzystuje własności ekspanderów. Wewnętrznym stanem jest graf regularny, w celu zaszyfrowania tekstu jawnego należy wykonać kilka spacerów losowych po tym grafie. Opisujemy tu zmodyfikowany schemat GraFPE [16] – spacer wykonywany jest wyłącznie po grafach skierowanych, a nie jak to zostało zaproponowane w GraFPE po nieskierowanych ekspanderach. Zmiana ta powoduje, że schemat jest efektywniejszy.

Słowa kluczowe: FPE, szyfrowanie zachowujące format danych, łańcuch Markowa, FF1, FF3

1 Wstęp

Szyfrowanie zachowujące format danych (*format-preserving encryption*, FPE) jest takim szyfrowaniem, które tekst jawny X szyfrują

* Wyniki badań były finansowane przez Regionalny Program Operacyjny Województwa Mazowieckiego na lata 2014-2020, umowa RPMA.01.02.00-14-5767/16-02

jako kryptogram Z, który ma ten sam format co *format* X. Klasycznym przykładem jest szyfrowanie numerów kart kredytowych: kryptogramem 16-cyfrowego ciągu z poprawną sumą kontrolną Luhn'a jest również 16-cyfrowy ciąg z poprawną sumą kontrolną Luhn'a.

FPE wykorzystywane są często w systemach, które działają od wielu lat, a w których wprowadza się zabezpieczenia np. poprzez szyfrowanie rekordów bazy danych. Wykorzystanie schematów, które są CPA/CCA-secure często nie wchodzi w grę, gdyż w każdym takim schemacie długość kryptogramu jest większa niż długość tekstu jawnego. Tak więc wprowadzenie takiego szyfrowania musiałoby wiązać się ze zmianą długości rekordów czy też formatów przetwarzanych danych. FPE jest wolne od tego typu problemów – długość i format kryptogramu jest taki sam jak danych wejściowych.

Każde tasowanie, które jest oblivious (oblivious shuffling) może być wykorzystane jako schemat FPE. Przykładowe schematy, wykorzystujące to podeście to:Thorp-shuffle [13], Swap-or-not [9], Mix-and-cut [15], Cycle-slicer [12].

Morris, Stegers i Rogaway pokazali w [13] jak wykorzystać Thorp-shuffle jako schemat FPE i udowodnili jego (nie adaptatywne) bezpieczeństwo PRP (non-adaptive PRP-security) dla liczby rund (schematu Thorp) $r \geq 2\log_2^2 n$ dla adwersarzy zadających nie więcej niż $n/(8\log_2 n)$ zapytań. Wynik ten pokazuje, że potrzeba około 1800 rund do zaszyfrowania 9-cyfrowego numeru SSN (social security number) oraz około 5650 rund do szyfrowania 16-cyfrowego numeru karty kredytowej. Aby uzyskać adaptatywne bezpieczeństwo, liczba rund musi być zwiększona dwókrotnie.

Na konferencji Eurocrypt 2014 Morris i Rogaway zaproponowali [14] schemat, który potrzebuje średnio $O(\log n)$ kroków. Autorzy szacują, że należy wykonać około 1200 wywołań AES dla poziomu bezpieczeństwa $\varepsilon = 10^{-10}$ (a w najgorszym przypadku, 18239 wywołań AES). Jednakże, gdy przyjęte zostaną bardziej praktyczne parametry bezpieczeństwa np. $\varepsilon = 2^{-128}$ to oczekiwana liczba wywołań AES rośnie do 2070, a dla $\varepsilon = 2^{-256}$ wynosi 3300.

Powyższe przykłady pokazują nieefektywność konstrukcji bazujących na oblivious-shuffle.

Innym sposobem na uzyskanie schematu FPE jest wykorzystanie sieci Feistela [7]. Podejście to jest wykorzystywane m. in. przez standardy ustalone przez NIST [6]: FF1 i FF3. Niestety, w 2016 i 2017

zostały zademonstrowane ataki [2,5] zarówno na FF1 jak i na FF3. Na konferencji Crypto 2018 Hoang, Tessaro i Trien zaprezentowali technikę [8], która skutecznie atakuje wszystkie używane schematy FPE: FF1, FF2 oraz rozwiązania Cisco – schemat FNR [4], oraz DTP [11] firmy Protegrity.

2 Grafy losowe

Przyjmujemy następujące oznaczenia: $[n] := \{0, \ldots, n-1\}$, a $\mathbb{G}_{n,d}$ oznacza zbiórd-regularnych grafów na wierzchołkach $[n]$. Przez losowy graf d-regularny rozumiemy graf wybrany ze zbioru $\mathbb{G}_{n,d}$ zgodnie z pewnym rozkładem (niekoniecznie jednostajnym).

2.1 Configuration model

Niech $\mathcal{G}_{n,d}$ będzie rozkładem jednostajnym na $\mathbb{G}_{n,d}$. Sposób losowania z rozkładu jednostajnego z $\mathcal{G}_{n,d}$ został zaproponowany przez Bollobasa w [3] i nosi nazwę *configuration model* (bądź pairing model). Niech nd będzie parzyste. Niech $C_0, \ldots, C_{n-1}$ będą partycjami nd punktów zdefiniowanych następująco (nazywamy je komórkami): $C_i = \{i, n+i, \ldots, (d-1)n+i\}$. Dla $x \in \{0, \ldots, nd-1\}$ definiujemy

$$\varphi(x) = x \bmod n$$

jako liczbę komórek, do których należy x tzn. $x \in C_{\varphi(x)}$. Niech $\mathcal{M}$ będzie partycją $\{0, 1, \ldots, nd-1\}$ na $nd/2$ par (czyli $\mathcal{M}$ jest skojarzeniem). Definiujemy $\kappa(\mathcal{M})$ jako (multi)graf na wierzchołkach $[n] = \{0, \ldots, n-1\}$ i o krawędziach $\{(\varphi(x), \varphi(y)) : (x, y) \in \mathcal{M}\}$.

Jeżeli matching $\mathcal{M}$ jest losowy to odpowiadający mu graf $\kappa(\mathcal{M})$ jest grafem prostym (bez krawędzi wielokrotnych i pętli) z prawdopodobieństwem $(1 + o(1))exp\left(\frac{1-d^2}{4}\right)$ (porównaj (2.1) w [10]) Do więcej, uzyskany tak graf jest spójny z dużym prawdopodobieństwem. Tak więc procedura samplowania $G \sim \mathcal{G}_{n,d}$ jest oczywista: należy wylosować skojarzenie doskonałe na wierzchołkach $\{0, \ldots, nd-1\}$ (w czasie $nd/2$) i zaakceptować uzyskany tak graf jeżeli jest on grafem prostym. W przeciwnym przypadku należy losować nowe skojarzenie. Niestety taka procedura ma oczekiwany czas działania $nde^{(d^2-1)/4}$ (dla $d \leq n^{1/3}$). Przykładowe skojarzenie dla $d = 4, n = 8$

jest przedstawiony na Rysunku 1a, a odpowiadający mu graf na Rysunku 1b.

2.2 The permutation model

Teraz opiszemy inny sposób losowania grafu z $\mathbb{G}_{n,d}$ (z nieco innym rozkładem niż jednostajny). Ten sposób jest wykorzystywany w schemacie szyfrowania GraFPE. Sposób ten nosi nazwę *permutation model* i polega na tym, że wybiera się $d/2$ losowych permutacji (dla parzystego d), a następnie z permutacji tworzy się wynikowy graf.

Niech $\{P\}_{i\geq 1}$ będzie zbiorem losowych permutacji na $[n]$.

Definicja 1 *Zbiór m permutacji $\mathcal{P}_m = \{P_1, \ldots, P_m\}$ nazywamy* valid *jeżeli zachodzą poniższe warunki:*

1. $\forall_i \forall_x P_i(x) \neq x$ *(brak pętli własnych),*
2. *if* $P_i(x) = y$ *then* $\forall_j P_j(y) \neq x$ *(brak ścieżek długości 2),*
3. *if* $P_i(x) = y$ *then* $\forall_{j\neq i} P_j(x) \neq y$ *(brak krawędzi wielokrotnych).*

Procedura sprawdzająca czy zbiór permutacji $\{P_1, \ldots, P_m\}$ na $[n]$ jest *valid* jest oznaczany jako $\mathsf{ValidSet}(\langle P_1, \ldots, P_m \rangle, n)$.
Now, instead of a random matching (as in configuration model), we use a set of valid permutations to create a graph.

Definicja 2 *Dla zbioru permutacji $\mathcal{P}_k = \{P_1, \ldots, P_k\}$ na $[n]$, który jest valid definiujemy indukowany graf $\gamma(\mathcal{P}_k) = (V, E)$, gdzie $V = [n]$*

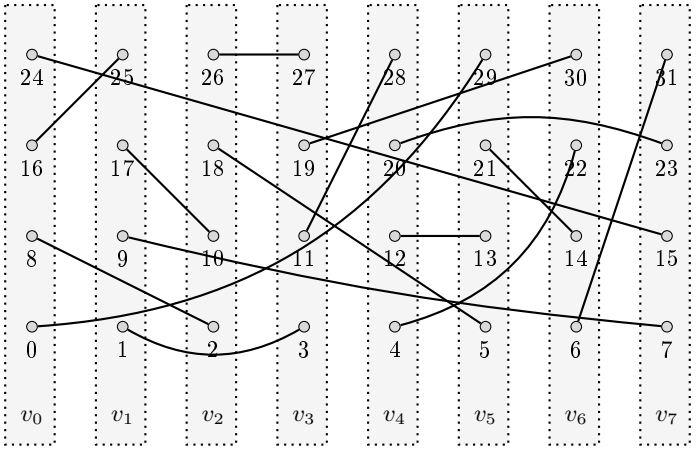
$\gamma_u(\mathcal{P}_k)$ **nieskierowany stopnia** $2k$**:**
$E = \{(x, y) : x, y \in V, (P_i(x) = y \vee P_i(y) = x), i = 1, \ldots, k\}$,
$\gamma_d(\mathcal{P}_k)$ **skierowany stopnia** k**:**
$E = \{(x, y) : x, y \in V, P_i(x) = y, i = 1, \ldots, k\}$.

Przykład grafu indukowanego jest przedstawiony na Rysunku 2.

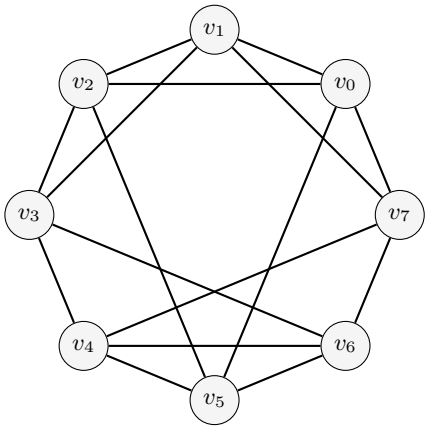
Niestety jeżeli zbiór $\mathcal{P}_{d/2}$ jest zbiorem *jednostajnych permutacji* n-elementowych, otrzymany graf $\gamma(\mathcal{P}_{d/2})$ nie musi być jednostajny na $\mathbb{G}_{n,d}$. Niech $\mathcal{G}^{\mathcal{P}}_{n,d}$ oznacza rozkład tak generowanych grafów $\gamma(\mathcal{P}_{d/2})$.

2.3 Generowanie losowych permutacji

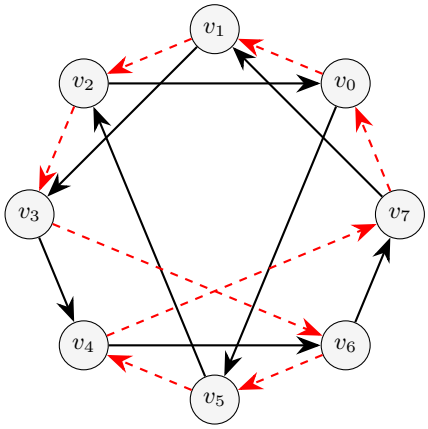
Chcąc wygenerować permutację n elementów możemy pomyśleć o wykonaniu tasowania n kart. Rozpatrzmy następujący schemat tasowania – każdy jego krok to jednen krok odpowiadającego mu łańcucha Markowa.



(a) Skojarzenie $\mathcal{M}$



(b) Graf $G = (V, E)$ indukowany ze skojarzenia $\mathcal{M}$. $V = \{0, \ldots, 7\}$, dla każdej krawędzi $(i, j) \in \mathcal{M}$ istnieje odpowiadająca jej krawędź $(v_{i \bmod 8}, v_{j \bmod 8}) \in E$.



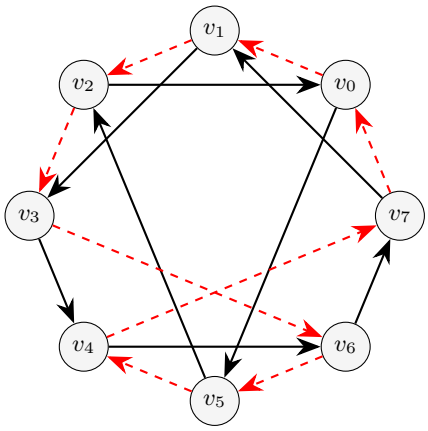
(c) Permutacje P_1 (czerwony), P_2 (czarny) i indukowany przez nie graf G.

Fig. 1: Przykładowe generowanie grafów regularnych G ze zbioru permutacji o własności valid i z doskonałego skojarzenia $\mathcal{M}$.

$$P_1 = \begin{pmatrix} 0\,1\,2\,3\,4\,5\,6\,7 \\ 5\,3\,0\,4\,6\,2\,7\,1 \end{pmatrix}$$

$$P_2 = \begin{pmatrix} 0\,1\,2\,3\,4\,5\,6\,7 \\ 1\,2\,3\,6\,7\,4\,5\,0 \end{pmatrix}$$

(a) Zbiór permutacji $\mathcal{P}_2 = \{P_1, P_2\}$.



(b) Indukowany graf 4-regularny $\gamma(\mathcal{P}_2)$, otrzymany z permutacji P_1 (czerwona) i P_2 (czarna)

Fig. 2: Przykład wygenerowanego grafu 4-regularnego ze zbioru 2-elementowego permutacji.

> *Dla każdej karty w talii rzucamy symetryczną modetą i przypisujemy jej 0 gdy wypadła reszka i 1 gdy wypadł orzeł. Karty, którym przypisano 1 są przekładane (z zachowaniem relatywnej kolejności – tasowanie stabilne) na górę talii.*

Powyżej zdefiniowany sposób tasowania jest *odwróconym w czasie* tasowaniem Riffle Shuffle. Następujące reguła jest SST dla tego tasowania (zaproowana przez Aldous'a i Diaconis'a w [1]):

> *Dla każdej karty zachowujemy przypisywane w kolejnych krokach bity (ciąg tych bitów nazywamy trajektorią). Kończymy tasowanie gdy wszystkie trajektorie są różne.*

Po zatrzymaniu procesu tasowania zgodnie z powyższą zasadą, otrzymana permutacja jest samplem z rozkładu jednostajnego (gdyż jest to proces stacjonarny dla tego procesu). Oczekiwana liczba kroków w tym tasowaniu wynosi $2\log_2 n$ kroków. Powyższy opis stanowi nieformalny opis wyidealizowanego modelu matematycznego. W naszym schemacie idealna losowość (symetryczna moneta) zostaje zastąpiona przez bity generowane przez generator pseudolosowy.

Procedura jest przestawiona jako Algorytm 1.

Algorithm 1 RiffleShuffle(n, Rng)

Require: n – rozmiar permutacji, Rng – stateful PRNG
1: $\pi = \langle 1, \ldots, n \rangle$
2: **for** $i = 0$ to $n - 1$ **do**
3: **for** $j = 1$ to $i - 1$ **do**
4: $M[i, j] = 0$
5: **end for**
6: **end for**
7: **while** $\exists_{0 \leq i \leq n-1} \exists_{1 \leq j < i} M[i, j] == 0$ **do**
8: $\mathcal{S}_0, \mathcal{S}_1 = \emptyset$
9: $\mathbb{B} \leftarrow \mathsf{Rng}.getBits(n)$ {pobranie n bitów z Rng}
10: **for** $w := 0$ to $n - 1$ **do**
11: $b = \mathbb{B}[w]$
12: $\mathcal{S}_b = \mathcal{S}_b \uplus \pi[w]$
13: **end for**
14: **for** $i \in \mathcal{S}_0$ **do**
15: **for** $j \in \mathcal{S}_1$ **do**
16: $M[\max(i, j), \min(i, j)] = 1$
17: **end for**
18: **end for**
19: $\pi = \mathcal{S}_0 \uplus \mathcal{S}_1$
20: **end while**
21: **return** π

2.4 Generowanie zbiorów permutacji

Generowanie zbioru permutacji, który jest valid tj. spełnia Definicję 1 przebiega w następujący sposób. Wybieramy losowo pierwszą permutację P_1, jeżeli nie posiada punktów stałych i cykli długości 2 to ją dodajemy do zbioru. Jeżeli nie spełnia warunków Definicji 1 to ją odrzucamy i próbujemy ponownie. Kolejne permutacje dorzucamy do zbioru, jeżeli własność *valid* jest zachowana po dodaniu nowej permutacji do zbioru. Procedura ta jest opisana jako Algorytm 3 w pracy [16].

Algorithm 2 GenGraph(n, d, Rng)

Require: d – parzyste ($k = d/2$), Rng – stateful PRNG
1: **for** $i = 1 \ldots k$ **do**
2: **repeat**
3: $P_i \leftarrow \mathsf{RiffleShuffle}(n, \mathsf{Rng})$ {generujemy nową permutację}
4: **until** $\mathsf{ValidSet}(\langle P_1, \ldots, P_i \rangle, n) == 1$
5: **end for**
6: **return** $\langle P_1, \ldots, P_k \rangle$

Algorytm nosi nazwę GenGraf dlatego, że wygenerowany przez niego zbiór permutacji $P = \langle P_1, \ldots, P_k \rangle$ gwarantuje, że graf $\gamma(P)$ jest n-wierzchołkowym grafem d-regularnym.

Generowanie grafu skierowanego
Zmodyfikowany schemat GraFPE, proponowany w niniejszym opracowaniu różni się od wersji oryginalnej sposobem generowania grafu. Gdy chcemy wygenerować graf skierowany stopnia k (dla k parzystego, w praktyce najoptymalniej jest wykorzystywać k będące potęgą liczby 2), możemy wykorzystać do tego dwa podejścia:

1. wygenerować zbiór P_k składający się z k permutacji, które są valid i otrzymać indukowany graf $\gamma_d(\mathsf{P}_k)$,
2. wygenerować zbiór P_{k-1} składający się z $k-1$ permutacji, które są valid. Następnie dodać do zbioru permutację identycznościową P_k: $\mathsf{P}_k = \mathsf{P}_{k-1} \cup P_k$. Wynikowy graf jest uzyskiwany jako $\gamma_d(\mathsf{P}_k)$

Algorithm 3 GenGraph(n, d, Rng)

Require: d – parzyste ($k = d/2$), Rng – stateful PRNG
1: $P_1 \leftarrow id_n$
2: **for** $i = 2 \ldots k$ **do**
3: **repeat**
4: $P_i \leftarrow$ RiffleShuffle(n, Rng) {generujemy nową permutację}
5: **until** ValidSet$(\langle P_1, \ldots, P_i \rangle, n) == 1$
6: **end for**
7: **if** $\gamma_d(\langle P_1, \ldots, P_k \rangle)$ nie jest grafem spójnym **then**
8: GOTO 1
9: **end if**
10: **return** $\langle P_1, \ldots, P_k \rangle$

Powyższa procedura ma następujące przewagi nad oryginalnym schematem: (1) gwarantuje, że spacer losowy jest wykonywany po grafie spójnym, (2) z uwagi na dodanie do permutacji (krok 1 w algorytmie GenGraph) permutacji identycznościowej, gwarantujemy, że rozkładem stacjonarnym spaceru losowego jest rozkład jednostajny, (3) jak pokazują wyniki eksperymentalne przedstawione w Tabeli 1, spacer losowy po grafie skierowanym szybciej zbiega do rozkładu stacjonarnego.

3 GraFPE

W tym rodziale zaprezentowany jest schemat szyfrowania $\mathsf{F} = \mathsf{F}_{n,d,\varepsilon}$. Parametr bezpieczeństwa oznaczamy przez ε i wpływa on wraz z parametrami n i d na czas działania algorytmów szyfrujących i deszyfrujących.

Przestrzenią wiadomości jest $\mathsf{F.Dom} = \{0, \ldots, n_1 - 1\} \times \ldots \times \{0, \ldots, n_t - 1\}$ dla dowolnego ciągu $n_i \geq 3$, jednakże dla przejrzystości opisu przyjmujemy, że dziedziną jest $\mathsf{F.Dom} = \{0, \ldots, n-1\}^t$ (czyli $n_i = n$).

Przestrzeń kluczy oznaczamy przez $\mathsf{F.Keys}$, a przestrzeń "tweak'ów" oznaczana jest jako $\mathsf{F.Twk}$. Podobnie jak inne schematy szyfrowania typu FPE, GraFPE wykorzystuje szyfr blokowy Ciph (w implementacji wykorzystujemy AES w trybie GCM) tak więc w tym przypadku przestrzeń kluczy $\mathsf{F.Keys} = \mathsf{AES.Keys}$, np. $\{0,1\}^{256}$.

Schemat ma dodatkową fazę $\mathsf{F.GenGraph}(n, d)$ zależy od przyjętego alfabetu/dziedziny $\{0, \ldots, n-1\}$, parametru d (domyślnie mała potęga 2). Rezultatem tej fazy jest zbiór $d/2$ permutacji $\mathcal{P} = (P_1, \ldots, P_{d/2})$ (Algorytm 3) i odpowiadający jemu graf G. Faza ta nie zależy ani od tekstu jawnego ani od tweaka. Może (ale nie musi) zależeć od klucza).

Scheme 1 (GraFPE) *Schemat szyfrowania GraFPE* $\mathsf{F} = \mathsf{F}_{n,d,\epsilon}$ *nad przestrzenią wiadomości* $\mathsf{F.Dom} = \{0, \ldots, n-1\}^t$, *przestrzenią kluczy* $\mathsf{F.Keys}$, *przestrzenią tweaków* $\mathsf{F.Twk}$, *parametrem bezpieczeństwa* ϵ, *całkowitą liczbą parzystą* d *i schematem szyfrowania wspierającym "associated data" (np. AES-GCM)* Ciph *to czwórka algorytmów* $\langle \mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec} \rangle$*:*

- $\mathsf{F.GenGraph}(d, n)$ *zwraca:* $\mathcal{P} = \mathsf{GenGraph}(n, d, \cdot)$ *(opisany jako Algorytm 3) zbiór permutacji* $\mathcal{P} = \left\langle P_1, \ldots, P_{\frac{d}{2}} \right\rangle$, *które jednoznacznie wyznaczają* d*-regularny graf na* n *wierzchołkach.*
- $\mathsf{F.Gen}(d, n, s)$ *zwraca: klucz* K *wybrany jednostajnie losowo z* $\mathsf{F.Keys}$,
- $\mathsf{F.Enc}(G, \mathcal{P}, K, T, l, \langle x_1, \ldots, x_t \rangle) = \langle z_1, \ldots, z_t \rangle$, *opisany jako Algorytm 7.*
- $\mathsf{F.Dec}(G, \mathcal{P}, K, T, l, \langle z_1, \ldots, z_t \rangle) = \langle x_1, \ldots, x_t \rangle$, *opisany jako Algorytm 8.*

Lemat 1 *W zależności od rozmiaru* n *grafu* G*, jego stopnia wierzhołka* d *i parametru bezpieczeństwa* s*, długość spaceru losowego powinna*

wynosić: $l = \lceil \frac{d}{d-2} \log_{d-1} n + \gamma\sqrt{\log n} \rceil$, *gdzie* $\gamma = (\Lambda + o(1))\Phi^{-1}(1/2^s)$, *aby szyfrowanie i deszyfrowanie zapewniało poziom bezpieczeństwa* $1/2^s$.

3.1 Spacer losowy

Mając dany graf G, zbiór permutacji $\mathcal{P}$, punkt startowy $x \in V = [n]$ oraz ciąg bitów losowych B, $RW(G, \mathcal{P}, x, l, B)$ jest l-krokowym spacerem losowym na G startującym w x. Procedura jest przedstawiona jako Algorytm 6.

Algorithm 4 $RW(G, \mathcal{P}, x, l, B)$

Require: $d = 2^f$ regularny graf G na wierzchołkach $[n]$, $x \in [n]$, $B = [b_1, \ldots, b_{lf}], b_i \in \{0, 1\}$.

$x_0 := x$
for $i = 1 \ldots l$ **do**
bity od $b_{(i-1)f+1}$ do $b_{(i-1)f+f}$, tj.

$$t_i = b_{(i-1)f+1} b_{(i-1)f+2} \ldots, b_{(i-1)f+f-1}$$

wyznaczają numer permutacji względem której przechodzimy do sąsiadującego wierzchołka, natomiast $b_{if} = b_{(i-1)f+f}$ określa, czy idzie się względem P_{t_i} czy $P_{t_i}^{-1}$, dokładniej:

$$x_i = \begin{cases} P_{t_i}(x_{i-1}) & \text{if } b_{if} = 0, \\ P_{t_i}^{-1}(x_{i-1}) & \text{if } b_{if} = 1. \end{cases}$$

end for
return x_l

Należy zauważyć, że operacja RW jest *odwracalna* tj. mając dane $G, \mathcal{P}, B$ i punkt końcowy spaceru y możemy jednoznacznie znaleźć punkt początkowy x tj. punkt taki, że $y = RW(G, \mathcal{P}, x, l, B)$ (przedstawione jako Algorytm 5).

Algorithm 5 $RW^{-1}(G, \mathcal{P}, y, l, B)$

Require: $d = 2^f$ regularny graf G na wierzchołkach $[n]$, $y \in [n]$, $B = [b_1, \ldots, b_{lf}], b_i \in \{0, 1\}$.

$y_l := y$
for $i = l \ldots 1$ **do**
bity $t_i = b_{(i-1)f+1} b_{(i-1)f+2}, \ldots, b_{(i-1)f+f-1}$ wyznaczają, która permutacja jest wykorzystana, natomiast bit $b_{(i-1)f+f} = b_{lf}$ okrela czy spacer jest zgodny z $P_{t_i}^{-1}$ czy P_{t_i}, dokładniej:

$$y_{i-1} = \begin{cases} P_{t_i}(y_{i-1}) & \text{if } b_{(i-1)f+f} = 1, \\ P_{t_i}^{-1}(y_{i-1}) & \text{if } b_{(i-1)f+f} = 0. \end{cases}$$

end for
return y_0

Spacer po grafie skierowanym

W zmodyfikowanej wersji algorytmu, gdy spacer jest wykonywany po grafie skierowanym nigdy nie idziemy względem P_i^{-1}, dlatego też w każdym kroku oszczędzany jest dodatkowo jeden bit (bo nie musimy już określać "zwrotu" permutacji). Biorąc pod uwagę fakt, że generowane grafy są budowane za pomocą 4/8/16 permutacji, oszczędność wynosi 33% dla $d = 4$ (zamiast 3 bitów na jeden krok spaceru: 2, które określają numer permutacji i jednego, który określa zwrot potrzebne są 2 bity), 25% dla $d = 8$ i 20% dla $d = 16$.

Algorithm 6 $RW(G, \mathcal{P}, x, l, B)$

Require: $d = 2^f$ regularny graf **skierowany** G na wierzchołkach $[n]$, $x \in [n]$, $B = [b_1, \ldots, b_{lf}], b_i \in \{0, 1\}$.

$x_0 := x$
for $i = 1 \ldots l$ **do**
$t_i = b_{(i-1)f+1} b_{(i-1)f+2} \ldots, b_{(i-1)f+f}$
$x_i = P_{t_i}(x_{i-1})$
end for
return x_l

3.2 Szyfrowanie

Szyfrowanie przedstawione jako Algorytm 7 działa w sposób następujący: wykonuje $2t$ spacerów pseudo-losowych (zależnych m. in. od klucza i tweaka). W pierwszej fazie (absorbing phase – terminologia zapożyczona z konstrukcji sponge), zaczyna w punkcie $x_1, x_2, x_3, \ldots, x_t$ i kończy w punkcie $y_1, \ldots, y_t$. W drugiej fazie (squeezing phase) zaczyna w punkcie: $y_1, y_2, y_3, \ldots, y_t$ i kończy w $z_1, z_2, \ldots, z_t$. Ostatecznie, dla tekstu jawnego $x_1 \ldots x_t$ odpowiadającym kryptogramem jest $z_1 \ldots z_t$.

Punkty pośrednie i końcowe zależą od wszystkich punktów początkowych. Zaczynając w x_i, losowość wykorzystana do spaceru zależy od $y_1, \ldots, y_{i-1}$ oraz od $x_{i+1}, \ldots, x_t$. Analogicznie, zaczynając spacer y_i losowość zależy od $z_1, \ldots, z_{i-1}$ oraz od $y_{i+1}, \ldots, y_t$. Wizualizacja procesu szyfrowania jest przedstawiona na Rysunku 3, a opis jako Algorytm 7.

Algorithm 7 $\mathsf{F.Enc}(G, \mathcal{P}, K, T, l, \langle x_1, \ldots, x_t \rangle)$

1: $f = \lg d$
   {Faza I (absorbing phase)}
2: **for** $j = 1 \ldots t$ **do**
3: $\quad \mathbb{B} \leftarrow \mathsf{Ciph}_K(IV = T||y_1|| \ldots ||y_{j-1}||x_{j+1}|| \ldots ||x_t||j||"a", A = \emptyset, P = 0^{fl})$
4: $\quad y_j := RW(G, \mathcal{P}, x_j, l, \mathbb{B})$
5: **end for**
   {Faza II (squeezing phase)}
6: **for** $j = 1 \ldots t$ **do**
7: $\quad \mathbb{B} \leftarrow \mathsf{Ciph}_K(IV = T||z_1|| \ldots ||z_{j-1}||y_{j+1}|| \ldots ||y_t||j||"s", A = \emptyset, P = 0^{fl})$
8: $\quad z_j := RW(G, \mathcal{P}, y_j, l, \mathbb{B})$
9: **end for**
10: **return** $z_1, \ldots, z_t$

3.3 Deszyfrowanie

Podczas deszyfrowania (Algorytm 8) wykonuje $2t$ *odwróconych w czasie* spacerów losowych po G. Rozpoczyna spacer w punkcie z_t, losowość spaceru zależy od $z_1, \ldots, z_{t-1}$ (i klucza tajnego). Proces następnie jest wykonywany dla $z_{t-1}, \ldots z_1$ i otrzymywane są wartości $y_{t-1}, \ldots, y_1$. Gdy wszystkie y_i są obliczone, wartości x_i są otrzymywane w sposób analogiczny jak y_i były obliczane z z_i. Wizualizacja procesu deszyfrowania jest przedstawiona na Rysunku 4.

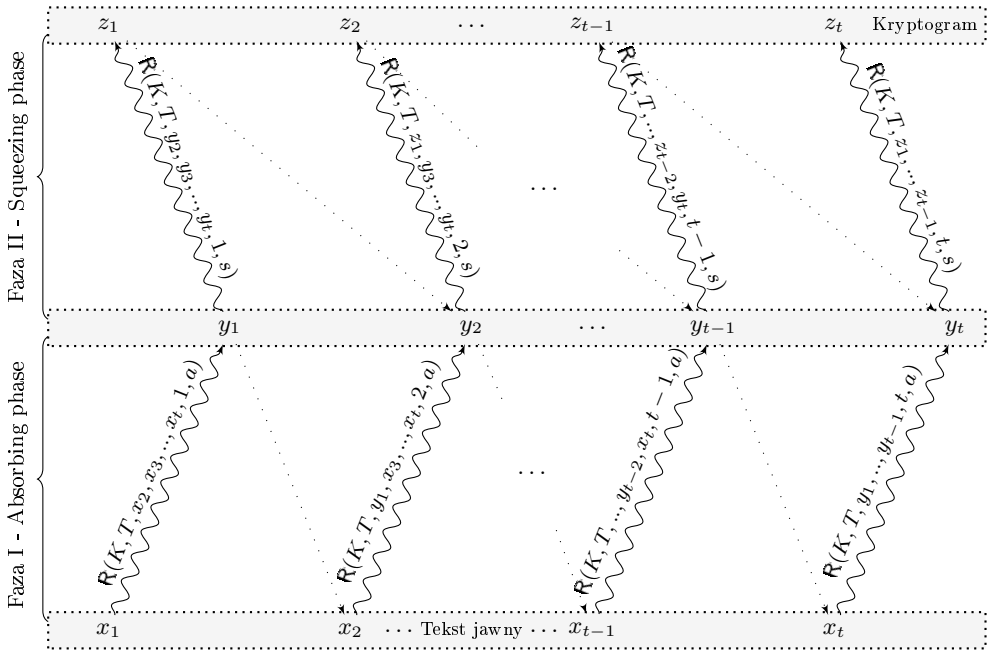


Fig. 3: Szyfrowanie. $\mathsf{R}(\cdot)$ – losowość użyta przez spacer losowy RW, zależna od wartości przypisanych w liniach 3 i 7 Algorytmu 7)

Algorithm 8 $\mathsf{F.Dec}(G, \mathcal{P}, K, T, l, \langle z_1, \ldots, z_t \rangle)$

1: $f = \lg d$
{odwracanie Fazy II (squeezing phase)}
2: **for** $j = t \ldots 1$ **do**
3: $\quad \mathbb{B} \leftarrow \mathsf{Ciph}_K(IV = T||z_1|| \ldots ||z_{j-1}||y_{j+1}|| \ldots ||y_t||j||"s", A = \emptyset, P = 0^{fl})$
4: $\quad y_j := RW^{-1}(G, \mathcal{P}, z_j, l, \mathbb{B})$
5: **end for**
{odwracanie Fazy I (absorbing phase)}
6: **for** $j = t \ldots 1$ **do**
7: $\quad \mathbb{B} \leftarrow \mathsf{Ciph}_K(IV = T||y_1|| \ldots ||y_{j-1}||x_{j+1}|| \ldots ||x_t||j||"a", A = \emptyset, P = 0^{fl})$
8: $\quad x_j := RW^{-1}(G, \mathcal{P}, y_j, l, \mathbb{B})$
9: **end for**
10: **return** $x_1, \ldots, x_t$

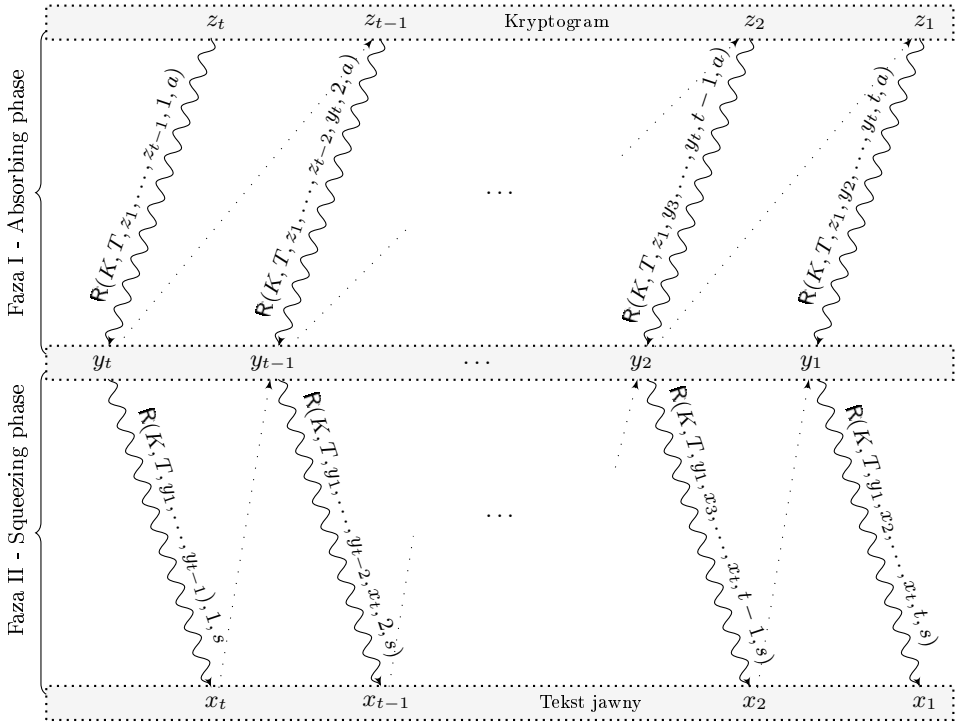


Fig. 4: Deszyfrowanie. $\mathsf{R}(\cdot)$ jest losowością wykorzystywaną przez spacer losowy RW^{-1}.

4 Gwarancje bezpieczeństwa

W tym rozdziale przedstawiamy analizę bezpieczeństwa rozwiązania. Rozważania ograniczamy do grafów o rozmiarach, które najczęściej wykorzystywane są w praktyce, a więc $n = 100, 1\,000, 10\,000$. Wyniki dla innych wartości można przybliżyć wykorzystując Twierdzenie **??**.

W szczególności prezentujemy wyniki dla tzw. przeklętych domen ("cursed domains", które doprowadziły do złamania standardów FF1/FF3), gdzie $n = 100$. W takich przypadkach możemy dla danego grafu G o n wierzchołkach policzyć dokładnie odległość (total variation distance) pomiędzy rozkładem łańcucha startującego w wierzchołku j po k krokach a rozkładem stacjonarnym (jednostajnym) $\pi(\cdot) = 1/n$,

$$d_{TV}(\mathbf{P}^k(j,\cdot),\pi) = \sum_{i=0}^{n-1} \left| \mathbf{P}^k(j,i) - \frac{1}{n} \right|.$$

Będziemy chcieli zminimalizować wartość maksimum odległości po wszystkich stanach

$$\rho(k) = \max_{j\in\{0,\ldots,n-1\}} \left\{ d_{TV}(\mathbf{P}^k(j,\cdot),\pi) \right\}.$$

Dzięki temu będziemy mieli gwarancję, że po k krokach, dla każdego początkowego stanu j zachodzi

$$d_{TV}(\mathbf{P}^k(j,\cdot),\pi) \leq \frac{1}{2^{\tau(k)}}, \text{ gdzie } \tau(k) := \log_2\left(\frac{1}{\rho(k)}\right).$$

W Tabeli 1 przedstawione są wartości $\tau(k)$ dla $n = 100$ wierzchołków i różnych stopni d grafu. Ponadto analogiczne wyniki są przedstawione też dla grafu skierowanego.

5 Podsumowanie

Zaprezentowany shcemat FPE jest efektywniejszy niż złamane niedawno standardy FF1 i FF3. Z uwagi na to, że interesujące dziedziny są stosunkowo małe (np. $n = 100/1000$) to można dla wygenerowanych parametrów systemu (wygenerowanych grafów) policzyć dokładnie jakie są gwarancje bezpieczeństwa.

$n = 100$, Undirected graph			
steps	degree d		
	$d = 4$	$d = 8$	$d = 16$
4	0.781	2.282	4.372
8	1.801	5.075	9.119
16	3.828	10.554	18.331
32	7.767	21.464	36.684
64	—	—	—

$n = 100$, Directed graph			
steps	degree d		
	$d = 4$	$d = 8$	$d = 16$
4	0.184	1.720	3.733
8	1.845	5.711	9.270
16	5.760	13.849	20.678
32	13.412	29.842	42.964
64	28.399	52.714	—

Table 1: Logarytm dwójkowy wartości $\tau(k)$ dla grafów skierowanych i nieskierowanych (w obu przypadkach dla $n = 100$) i różnych wartości stopnia d. Wartości w tabeli odpowiadają gwarancjom bezpieczeństwa w zależności od długości (*steps*) spaceru losowego wykonywanego podczas procedury RW w procesie szyfrowania/deszyfrowania

References

1. David Aldous and Persi Diaconis. Shuffling cards and stopping times. *American Mathematical Monthly*, 93(5):333–348, 1986.
2. Mihir Bellare, Viet Tung Hoang, and Stefano Tessaro. Message-Recovery Attacks on Feistel-Based Format Preserving Encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*, pages 444–455, New York, New York, USA, 2016. ACM Press.
3. Béla Bollobás. Random Graphs. pages 215–252. Springer New York, 1998.
4. Sashank Dara and Scott Fluhrer. FNR: Arbitrary Length Small Domain Block Cipher Proposal. pages 146–154. Springer, Cham, oct 2014.
5. F. Betül Durak and Serge Vaudenay. Breaking the FF3 Format-Preserving Encryption Standard over Small Domains. pages 679–707. Springer, Cham, aug 2017.
6. Morris Dworkin. Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption. *NIST Special Publication*, pages 800–38.
7. Viet Tung Hoang and Phillip Rogaway. On Generalized Feistel Networks. (1).
8. Viet Tung Hoang, Stefano Tessaro, and Ni Trieu. The Curse of Small Domains: New Attacks on Format-Preserving Encryption. In *CRYPTO 2018*. 2018.
9. VT T Hoang, Ben Morris, and Phillip Rogaway. An enciphering scheme based on a card shuffle. *Advances in Cryptology–CRYPTO 2012*, pages 1–13, 2012.
10. Eyal Lubetzky and Allan Sly. Cutoff phenomena for random walks on random regular graphs. *Duke Mathematical Journal*, 153(3):475–510, jun 2010.
11. Ulf T Mattsson. FORMAT-CONTROLLING ENCRYPTION USING DATATYPE-PRESERVING ENCRYPTION. Technical report.
12. Sarah Miracle and Scott Yilek. Cycle Slicer: An Algorithm for Building Permutations on Special Domains. pages 392–416. 2017.
13. Ben Morris, Philip Rogaway, and Till Stegers. How to Encipher Messages on a Small Domain Deterministic Encryption and the Thorp Shuffle. In *CRYPTO*, pages 1–19, 2009.
14. Ben Morris and Phillip Rogaway. Sometimes-Recurse Shuffle Almost Random Permutations in Logarithmic Expected Time. 2013.

15. Thomas Ristenpart and Scott Yilek. The mix-and-cut shuffle: small-domain encryption secure against N queries. In *Advances in Cryptology–CRYPTO 2013*, pages 392–409. Springer Berlin Heidelberg, 2013.
16. Adam Budziak, Paweł Lorek, Marcin Słowik, Filip Zagórski. GraFPE – a tweakable PRP-secure format preserving encryption scheme (secure also over cursed domains). preprint, 2019.