Metody kryptografii w analizie danych

LEDApkc

Arkadiusz Wołk

2023

1 Kryptografia z użyciem kodów korekcyjnych

1.1 Kody korekcyjne

Odwzorowanie:

$$C_{n,k} : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2^n \tag{1}$$

, gdzie $\mathbb{F}_2$ to binarne ciało skończone z operacjami dodawania i mnożenia, odpowiednio jako **XOR** i **AND**, nazywamy kodem t-korekcyjnym jeżeli dla dowolnego $u \in \mathbb{F}_2^k$ jesteśmy w stanie z wektora $\tilde{c} = C_{n,k}(u) + e$ odzyskać u i e, gdzie $e \in F_2^n$ to losowy wektor o wadze Hamminga równej t.

1.2 Liniowe kody korekcyjne

Liniowy kod korekcyjny jesteśmy w stanie zapisać przy użyciu macierzy $G \in F_2^{k \times n}$:

$$C_{n,k}(x) = xG \tag{2}$$

1.3 Systematyczne kody korekcyjne

Kod systematyczny to liniowy kod, dla którego wektory wyjściowe zawierają wektor wejściowy. Najczęściej takie kody przedstawia się jako konkatenację $n - k$ bitów do wektora wejściowego, odpowiadająca macierz G może przyjąć formę:

$$G = [I_k|P] \tag{3}$$

, gdzie I_k to macierz jednostkowa o wymiarze $k \times k$, a P to macierz binarna o wymiarze $k \times (n-k)$.

1.4 Macierz parzystości

Możemy zdefinować macierz parzystości $H \in F_2^{(n-k)\times n}$ tak, że jej iloczyn z wektorem o długości n (nazywany syndromem tego wektora) jest równy wektorowi zerowemu, wtedy i tylko wtedy gdy jest to jedno z zaszyfrowanych słów możliwych do uzyskania z $C_{n,k}$:

$$\forall_{c \in F_2^n} Hc^T = \mathbf{0} \tag{4}$$

Aby móc wyznaczyć H należy rozwiązać następujące równanie:

$$\forall_{u \in F_2^n} \mathbf{0}_{(n-k)\times 1} = HG^T u^T \tag{5}$$

$$\mathbf{0}_{(n-k)\times k} = HG^T \tag{6}$$

Dla kodów systematycznych uzyskamy:

$$G = [I_k|P]$$
$$H = [P^T|I_r]$$
$$HG^T = P^T + P^T = \mathbf{0}$$

2 McEliece

Pierwszy kryptosystem oparty o liniowe kody korekcyjne, którego omówienie jest konieczne to zrozumienia kolejnych rozdziałów.

2.1 Generowanie kluczy

Dla zadanych n i k wybierana jest macierz G, która generuje kod t-korekcyjny. Ponieważ znajomość tej macierzy jest równoważna z możliwością deszyfrowania dowolnego szyfru dla tego kodu to jest to część naszego klucza prywatnego.
Kluczem publicznym musi być też macierz generująca, lecz musi ona uniemożliwiać odzyskanie z niej klucza prywatnego. Z tego powodu wybierane są dodatkowe dwie macierze S (gęsta macierz mieszająca $k \times k$) i P (macierz permutacji $n \times n$), które też będą częścią klucza prywatnego. Wtedy klucz publiczny ma postać:

$$G' = SGP$$

2.2 Szyfrowanie

Aby zaszyfrować słowo binarne u o długości k przy pomocy klucza publicznego G' musimy wylosować wektor błędu o długości n i wadze Hamminga równej t i wtedy zaszyfrowane słowo ma postać:

$$x = uG' + e = c + e$$

2.3 Deszyfrowanie

Aby odszyfrować przesłane słowo x należy zauważyć, że:

$$\begin{aligned} x &= uG' + e = uSGP + e \\ x' = xP^{-1} &= uSG + eP^{-1} = u'G + e' \\ P^{-1} &= P^T \end{aligned}$$

Mamy więc słowo x', które jest zaszyfrowanym słowem $u' = uS$ przy pomocy kodu G z błędem $e' = eP^{-1}$. Po wydobyciu z x' słowa u' jesteśmy w stanie odzyskać $u = u'S^{-1}$.

3 LEDA

3.1 Blokowy kwazicykliczny kod korekcyjny z macierzą parzystości o niskiej gęstości

Kod blokowy to liniowy kod korekcyjny, dla którego rozmiar słowa wejściowego to pk_0, a słowo wyjściowe ma rozmiar $n_0 p$, gdzie n_0 to podstawowa długość bloku. W takim kodzie macierze G i H są złożone z podmacierzy o rozmiarach $p \times p$.

Kwazicykliczność określa postać podmacierzy w macierzy generującej i macierzy parzystości. Macierz kwazicykliczna ma następującą postać:

$$A = \begin{bmatrix} a_0 & a_1 & a_2 & \dots & a_{v-1} \\ a_{v-1} & a_0 & a_1 & \dots & a_{v-2} \\ a_{v-2} & a_{v-1} & a_0 & \dots & a_{v-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_1 & a_2 & a_3 & \dots & a_0 \end{bmatrix}$$

Pierścień macierzy kwazicyklicznych nad ciałem F_2 o rozmiarze $v \times v$ jest izomorficzny do pierścienia wielomianów $F_2[x]/\langle x^v + 1 \rangle$ z następującym przekształceniem:

$$A \leftrightarrow a(x) = \sum_{i=0}^{v-1} a_i x^i \tag{7}$$

Natomiast niska gęstość macierzy parzystości oznacz, że liczba wartości niezerowych d_v w kolumnie H jest zdecydowanie mniejsza od jej długości i rośnie z nią subliniowo.

3.2 LEDApkc

LEDApkc to kryptosystem oparty o blokowe kwazicykliczne kody korekcyjne z macierzą parzystości o niskiej gęstości, gdzie $n = pn_0$ i $k = p(n_0 - 1)$, a n_0 jest małą liczbą większą od 2 i p jest liczbą pierwszą. Oznacza to, że mamy tylko jeden blok redundancji o wymiarze $p \times p$.

3.3 Generowanie kluczy

3.3.1 KLucz prywatny

W LEDApkc kluczem prywatnym jest macierz parzystości H ($1 \times n_0$ bloków $p \times p$) i dodatkowa macierz transformacji Q ($n_0 \times n_0$ bloków $p \times p$).
Mamy zadany parametr d_v (od 15 do 25), który określa liczbę niezerowych elementów w kolumni-/wierszu każdego z n_0 kwazicyklicznych bloków wchodzących w skład H. Ponieważ są to macierze kwazicykliczne to wystarczy wygenerować jeden wiersz dla każdego bloku.
Podobnie mamy parametr $m = [m_0, m_1, ..., m_{n_0-1}]$, który określa liczbę niezerowych elementów w każdym z n_0 bloków w pierwszym wierszu kwazicyklicznej macierzy kwazicyklicznych bloków Q. W tym przypadku również losujemy pozycje na, których wystąpią niezerowe elementy dla każdego z bloków w określonej liczbie przez parametry m_i.
Odpowiedni dobór parametru m jest konieczny do poprawnego działania szyfru, co autorzy udowadniają w udostępnionej specyfikacji.

Implementacja referencyjna

W referencyjnej implementacji kluczem prywatnym jest prawdziwie losowa liczba o odpowiednio dobranej długości LEN_{TRNG} (tak aby zapewnić odpowiedni poziom bezpieczeństwa), która następnie jest używana jako ziarno dla generatora liczb pseudolosowych, który jest używany do generowania macierzy H i Q.

3.3.2 Klucz publiczny

Najpierw należy obliczyć macierz L:

$$L = HQ = [L_0|L_1|L_2|...|L_{n_0-1}]$$
$$L_i = \sum_{j=0}^{n_0-1} H_j Q_{j,i}$$

Odpowiedni dobór parametrów m i d_v zagwarantował nam, że macierz L_{n_0-1} ma odwrotność. Dzięki temu jesteśmy w stanie obliczyć macierz M_l, która jest naszym kluczem publicznym:

$$M = L_{n_0-1}^{-1} L = [M_0|M_1|M_2|...|M_{n_0-2}|I_p] = [M_l|I_p]$$

3.4 Szyfrowanie

Szyfrowanie przebiega tak samo jak w McELiece z tym, że macierz G' ma postać:

$$G' = [I_k|M_l^T]$$
$$k = (n_0 - 1)p$$

3.4.1 Implementacja referencyjna

Implementacja referencyjna jest bardziej rozbudowana niż opis w specyfikacji, dlatego zostanie tu także opisana.

Wektor błędu

W implementacji wektor błędu nie jest losowany lecz jest uzyskiwany poprzez użycie kodowania "constant weight", które pozwala na bezpieczne zaszyfrowanie na pn_0 bitach i t (liczba błędów) niezerowych wartościach słowa o długości LEN_{CW} (parametr szyfru).

Mkasymalny rozmiar

Teraz jesteśmy w stanie określić maksymalny rozmiar danych, który może szostać zaszyfrowany dla zadanych parametrów szyfru:

$$LEN_{MAX} = (n_0 - 1)p + LEN_{CW} - 8 * LEN_{HASH} - 2$$

, gdzie $(n_0-1)p$ jest liczbą bitów możliwą do zakodowania przy użyciu kodu korekcyjnego, LEN_{CW} liczba bitów możliwa do zaszyfrowania w błędzie, LEN_{HASH} liczba bajtów potrzebna do przesłania dodatkowych danych potrzebnych do odwórcenia zastosowanych transformacji użytych do ukrycia postaci jawnej słowa wejściowego, a 2 bity są wymagane do poprawnego działania dopełnienia.

Przygotowanie danych

Jak da się zauwyażyć kod publiczny jest kodem systematycznym, więc prefiks zaszyfrowanego słowa jest słowem jawnym, aby się przed tym zabezpieczyć przekazane dane nie są bezpośrednio szyfrowane lecz odpowiednio transformowane.
Tworzony jest bufor o długości $(n_0 - 1)p + LEN_{CW}$ bitów, który ma postać:

$$y = [\mathbf{0}_{LEN_{HASH}}|x|1|0000...|1]$$

, gdzie x to słowo wejściowe, a za nim dołożone jest dopełnienie.
Następnie losowana jest prawdziwie losowa liczba o rozmiarze w bajtach równym LEN_{TRNG} (parametr szyfru). Liczba ta jest następnie używana w deterministycznym generatorze liczb pseudolosowych to wygenerowania sekwencji bitów o długości równej $LEN_y - LEN_{HASH}$, a potem wykonywana jest opracja:

$$y[LEN_{HASH} :] = y[LEN_{HASH} :] \oplus prngSequence$$

Ta operacja ukrywa oryginalną postać słowa wejściowego, dzięki czemu po zastosowaniu szyfru jego prefiksem nie będzie słowo wejściowe.
Jednak aby móc odzyskać oryginalne dane musimy przesłać wylosowane ziarno. To tego jest wykorzystywane początkowe $8 * LEN_{HASH}$ bitów z bufora y, do którego jest zapisywane ziarno, a następnie xorowane ze skrótem $y[LEN_{HASH} :]$.

Szyfrowanie

Pierwsze LEN_{CW} bitów z bufora y jest zamieniane na wektor błędu przy użyciu kodowania "constant weight", a pozostałe są szyfrowane tak jak opisano to na początku rozdziału przy użyciu klucza publicznego. Na koniec do zaszyfrowanego wektora jest dodawany wektor błędu.

3.5 Deszyfrowanie

Deszyfrowanie wygląda podobnie jak w McEliece:

$$s^T = HQx^T = HQ(uG' + e)^T = HQe^T = H(eQ^T)^T$$

Z tego wynika, że syndrom całego zaszyfrowanego słowa możemy traktować jak syndrom wektora błędu $e' = eQ^T$. Dla takiego przypadku istnieje wydajny algorytm deszyfrujący (Q-Decoder) pozwalający na odzyskanie wektora e, a co za tym idzie i x (słowo poddane szyfrowaniu przez kod korekcyjny).

$$x + e = u[I_k|M_l^T]$$

Zaszyfrowane dane możemy odzyskać poprzez wyciągnięcie $p(n_0 - 1)$ bitów z wektora $x + e$.

3.5.1 Q-Decoder

Jako wejście przyjmuje syndrom s i macierze H i Q. Działa poprzez wykonanie maksymalnie l_{max} iteracji, gdzie każda wymaga na wejściu $s^{(l-1)}$ i $e^{(l-1)}$, a na wyjściu zwraca $s^{(l)}$ i $e^{(l)}$. Przyjmujemy, że $s^{(0)} = s$ i $e^{(0)} = \mathbf{0}$. Każda iteracja składa się z następujących kroków:

1. Obliczenie $\Sigma^{(l)} = [\sigma_1^{(l)}, \sigma_2^{(l)}, ..., \sigma_{pn_0}^{(l)}]$, gdzie kolejne wartości oznaczają liczbę niespełnionych równań parzystości powiązanych z tą pozycją. Pozwala to na oszacowanie prawdopodobieństwa, że w wektorze błędu e' na tej pozycji występuje 1.

2. Obliczenie $R^{(l)} = [p_1^{(l)}, p_2^{(l)}, ..., p_{pn_0}^{(l)}]$, gdzie kolejne wartości zonaczają korelację między kolejnymi wartościami z $\Sigma^{(l)}$, a wierszami Q^T. Ponieważ e i Q są rzadkie, a e' można zapisać jako sumę wierszy macierzy Q^T ($e' = eQ^T$) to nakładanie się na siebie wartości niezerowych przy mnożeniu jest mało prawdopodobne (a co za tym idzie, jest mało prawdopodobne, że kilka pozycji z oryginalnego wektora błędu e ma wpływ na niespełnienie tego samego równania parzystości), a więc duża wartość korelacji oznacza wysokie prawdopodobieństwo, że dany v wiersz macierzy Q^T składa się na wektor e', a co za tym idzie oryginalny wektor e ma na pozycji v wartość 1.

3. Wyznaczenie progu korelacji b_l, takiego, że jeżeli pozycja v w R ma wartość większą to należy dokonać zmiany wartości na tej samej pozycji w e.

4. Wyznaczenie $e^{(l)} = e^{(l-1)} + \sum_{v:p_v > b^{(l)}} q_v$, gdzie q_v to v-ty rząd macierzy Q^T

5. Wyznaczenie $s^{(l)} = s + e^{(l)}H^T$

6. Jeżeli waga Hamminga wektora s jest równa 0 to pomyślnie zdekodowano i należy zwrócić $e^{(l)}$

Jeżeli w l_{max} iteracjach nie uda nam się odkodować błędu to nastąpuję błąd dekodowania i nalęzy poinformować o tym drugą stronę.

3.5.2 Implementacja referencyjna

W implementacji referencyjnej po odzyskaniu zaszyfrowanych danych musimy wykonać odpowiednie operacje, aby odwrócić transformacje z procesu szyfrowania i odzyskać słowo wejściowe.
Obliczamy w jako wartość po dekodowaniu "constant weight" wektora e, a nstępnie tworzymy bufor $y = w|u$ jako konkatenacje wektora błędu i odzyskanych zaszyfrowanych danych. Pierwsze $8 * LEN_{HASH}$ stanowi obfuskowane ziarno, które odzyskujemy poprzez:

$$seed = y[: 8 * LEN_{HASH}] \oplus HASH(y[8 * LEN_{HASH}])$$

Następnie podobnie jak przy szyfrowaniu inicjalizujemy generator liczb pseudolosowych tym ziarnem tak, aby w wyniku uzyskać identyczną losową sekwencję i odzyskać słowo wejściowe:

$$u = y[8 * LEN_{HASH}] \oplus prngSequence$$

Na koniec wystarczy usunąć dopełnienie i otrzymujemy słowo wejściowe.