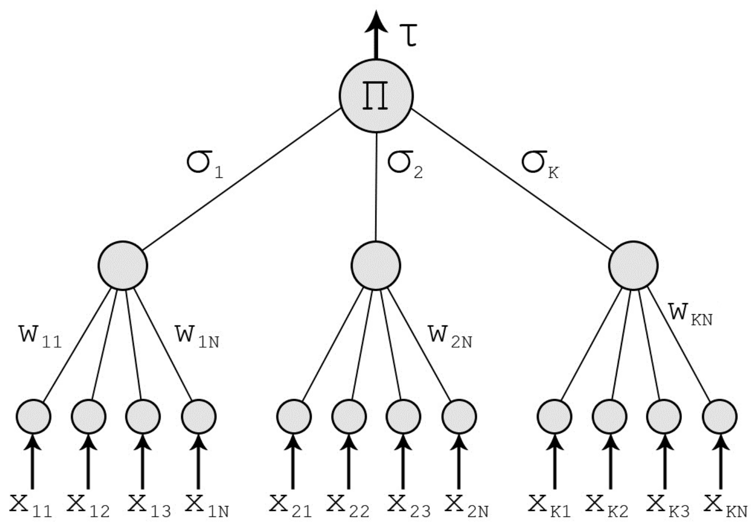
Kryptografia – Parity Machines

Łukasz Dubiel, Maksymilian Wojnar

1 Opis algorytmów Parity Machines

Algorytmy z rodziny Parity Machines służą do wymiany klucza prywatnego. Ich działanie opiera się na wymianie komunikatów prowadzącej do synchronizacji odpowiedniego wektora, z którego następnie tworzony jest klucz prywatny. Ogólny schemat Parity Machines znajduje się na rysunku 1. Algorytmy konstruują specjalną sieć neuronowej, która ma jedno wyjście i jedną warstwę ukrytą, a wagi mają wartości całkowitoliczbowe.



Źródło: Wikipedia, *Neural cryptography*

Rysunek 1: Schemat konstrukcji Parity Machines

2 Tree Parity Machine

2.1 Parametry

Parametrami algorytmu Tree Parity Machine (TPM) są:

- $k \in \mathbf{N}_+$ – liczba neuronów w warstwie ukrytej,
- $n \in \mathbf{N}_+$ – liczba wejść każdego neuronu warstwy ukrytej (co daje wektor wejściowy o długości $k \cdot n$),
- $l \in \mathbf{N}_+$ – głębokość synaptyczna, parametr opisujący zbiór możliwych wartości wag sieci, tj. każda waga $w_{ij} \in \{-l, ..., 0, ..., l\}$.

2.2 Opis działania

Najpierw wagi sieci są losowo inicjalizowane wartościami z $\{-l, ..., 0, ..., l\}$. Następnie, aż do momentu osiągnięcia synchronizacji, wykonywane są następujące kroki. Jedna instancja losuje wartości warstwy widocznej x ze zbioru $\{-1, 0, 1\}$ i przekazuje je drugiej instancji. Następnie obliczane są aktywacje neuronów warstwy ukrytej zgodnie z następującym wzorem $\sigma_i = sgn(\sum_{j=1}^{n} w_{ij} x_{ij})$, gdzie sgn to funkcja signum. Wyjście sieci obliczane jest w następujący sposób $\tau = \Pi_{i=1}^{k} \sigma_i$. Obie instancje wymieniają się wartościami τ. Jeśli wartości są równe, sieci aktualizują wagi jedną z poniższych metod:

- Hebbian learning rule: $w_{ij}^{+} = g(w_{ij} + \sigma_i x_{ij} \Theta(\sigma_i, \tau))$,
- Anti-Hebbian learning rule: $w_{ij}^{+} = g(w_{ij} - \sigma_i x_{ij} \Theta(\sigma_i, \tau))$,
- Random walk: $w_{ij}^{+} = g(w_{ij} + x_{ij} \Theta(\sigma_i, \tau))$,

gdzie g to deterministyczna funkcja ograniczająca wartości wag do zbioru $\{-l, ..., 0, ..., l\}$ (np. $g(w_{ij}) = min(l, max(w_{ij}, -l)))$, natomiast Θ to funkcja zwracająca 1, jeśli oba argumenty są tej samej wartości, 0 w przeciwnym przypadku.

Instancje są zsynchronizowane, jeśli wszystkie odpowiadające wagi są tej samej wartości. W celu stwierdzenia czy tak jest, możliwe jest zastosowanie wymiany hashy wektora wag sieci, co generowałoby dodatkowy duży narzut komunikacyjny. Alternatywą jest ustalenie pewnej liczby synchronizacji z rzędu, po której można uznać, że instancje są zsynchronizowane, co jednak może powodować nadmiarowe wydłużenie synchronizacji. Doświadczalnie sprawdzono, że po zaobserwowaniu $n+k+l$ aktualizacji z rzędu instancje są prawie zawsze zsynchronizowane. Taki eksperyment wykonano dla wartości $k \in [10, 20)$, $n \in [k + 10, k + 30)$ oraz $l \in [10, 30)$, po 10 powtórzeń dla każdej kombinacji. Na 80 000 prób synchronizacji nastąpiło 27 niepowodzeń.

2.3 Źródła

- Wikipedia – Tree Parity Machine
- „Synchronization of Tree Parity Machines using non-binary input vectors"
- Przykładowa implementacja

3 Permutation Parity Machine

3.1 Parametry

Parametrami algorytmu Permutation Parity Machine (PPM) są:

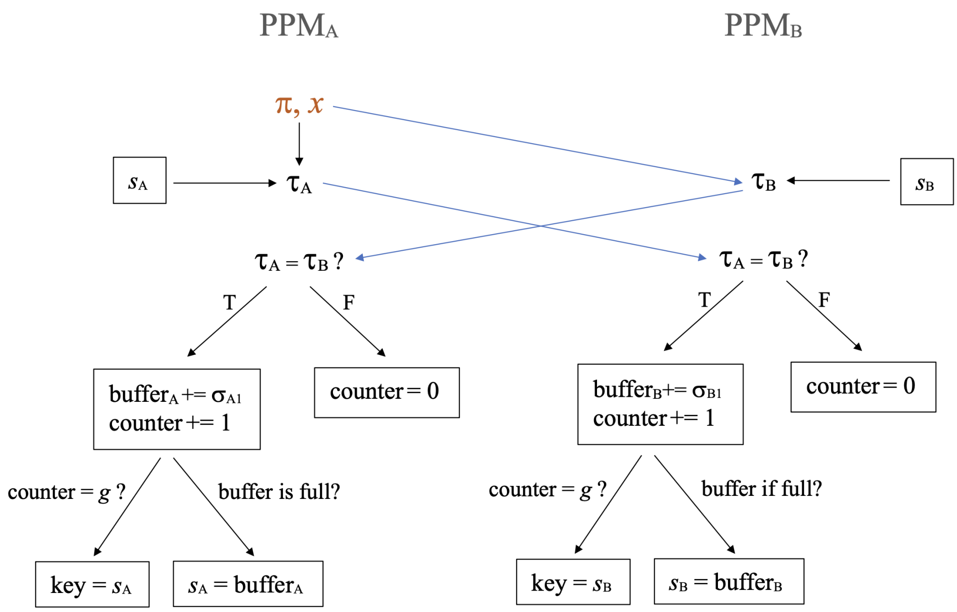
- $k \in \mathbf{N}_{+}$ – liczba neuronów w warstwie ukrytej,
- $n \in \mathbf{N}_{+}$ – liczba wejść każdego neurona warstwy ukrytej (co daje wektor wejściowy o długości $k \cdot n$),
- $g \in \mathbf{N}_{+}, g \geqslant k \cdot n$ – długość generowanego klucza w bitach.

3.2 Opis działania

PPM posiada wewnętrzny stan, na który składają się wektor s oraz bufor. Pierwszy z nich ma długość g i jest inicjowany losowo wartościami binarnymi, drugi zaś jest początkowo pusty i działa na zasadzie bufora cyklicznego o maksymalnej pojemności g. Wejścia x_{ij} oraz wagi sieci w_{ij} mają wartości binarne $\{0,1\}$. Wartość neuronów w warstwie ukrytej jest obliczana jako $\sigma_i = \theta_n \left(\sum_{j=1}^{n} w_{ij} \oplus x_{ij} \right)$, gdzie $\theta_n(x)$ przyjmuje wartość 1, jeśli $x \leqslant n/2$, a w przeciwnym przypadku 0. Wartość wyjścia jest obliczana jako $\tau = \oplus_{i=1}^{k} \sigma_i$.

Wagi sieci są losowane z wektora s w każdej iteracji algorytmu. Jedna ze stron generuje i udostępnia losowy wektor wejściowy $x \in \{0,1\}^{k \cdot n}$ oraz wektor permutacji π o długości $k \cdot n$ z wartościami ze zbioru $\{0,1,2,...,g-1\}$, który wyznacza macierz wag (`W = s[pi].reshape(k, n)`). Następnie obie strony obliczają wyjścia sieci τ_A oraz τ_B i wymieniają się tymi wartościami. Jeśli $\tau_A = \tau_B$, obie strony dokonują aktualizacji wag, która polega na dodaniu do bufora aktywacji pierwszego neuronu σ_1. Gdy bufor się zapełni, jego wartość przepisywana jest do wektora s, a kolejne aktualizacje wpisują wartości od początku bufora. Jeżeli przez kolejne g razy wyjścia sieci są identyczne, to PPM zsynchronizowały się i wektor s po obu stronach powinien mieć mieć taką samą wartość. Kluczem prywatnym jest liczba, której kolejne bity to wartości wektora s.

Na rysunku 2 przedstawiono działanie oraz komunikację pomiędzy stronami PPM w trakcie jednej iteracji algorytmu. Na pomarańczowo zaznaczono losowe wektory generowane w każdej iteracji, w czarnej ramce znajduje się stan PPM, czarna czcionka bez ramki oznacza wartości obliczane według wzorów podanych w poprzednim akapicie, niebieskie strzałki wskazują przesyłanie wartości pomiędzy stronami komunikacji, a czarne strzałki zależności i instrukcje warunkowe.



Rysunek 2: Schemat jednej iteracji Permutation Parity Machine

3.3 Źródła

- Wikipedia – Permutation Parity Machine
- „Key Exchange Protocol Using Permutation Parity Machines”

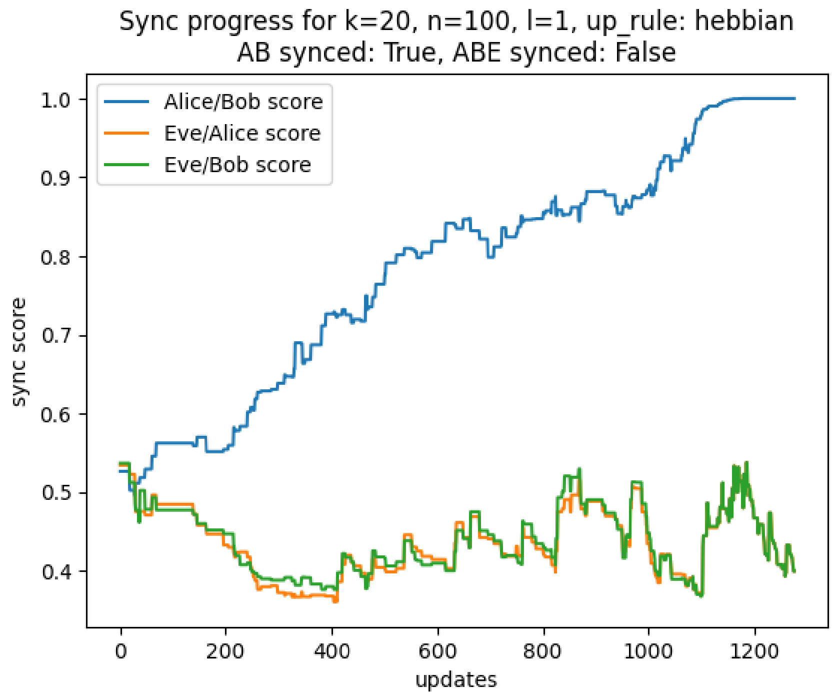
4 Analiza algorytmów

Bezpieczeństwo Parity Machines opiera się na tym, że do wykonania aktualizacji konieczne jest uzyskanie tego samego wyjścia τ dla danego wektora x. Sprawia to, że adwersarz podsłuchujący kanał komunikacyjny pomiędzy dwoma stronami ma znacznie mniej szans na wykonanie aktualizacji stanu maszyny, niż komunikujące się podmioty (aby adwersarz wykonał aktualizację, musi zachodzić warunek $\tau_A = \tau_B = \tau_E$, gdzie τ_E jest wyjściem sieci podsłuchującego adwersarza).

4.1 Szybkość synchronizacji instancji oraz adwersarza

Wykresy 3 i 4 przedstawiają poziom synchronizacji w zależności od liczby aktualizacji instancji TPM. Dla większych wartości parametrów uzyskujemy większe bezpieczeństwo, ponieważ poziom synchronizacji instancji Eve nie sięga niebezpiecznych wartości, natomiast w przypadku niskich wartości parametrów jest większa szansa, że Eve również się zsynchronizuje, co widać na wykresie 3. Ponadto dla małych wartości synchronizacja jest bardziej niejednostajna – częściej występują fluktuacje poziomu synchronizacji.

Rysunek 3: Wykres poziomu synchronizacji instancji Alice i Bob razem z atakującym Eve dla parametrów o małej wartości



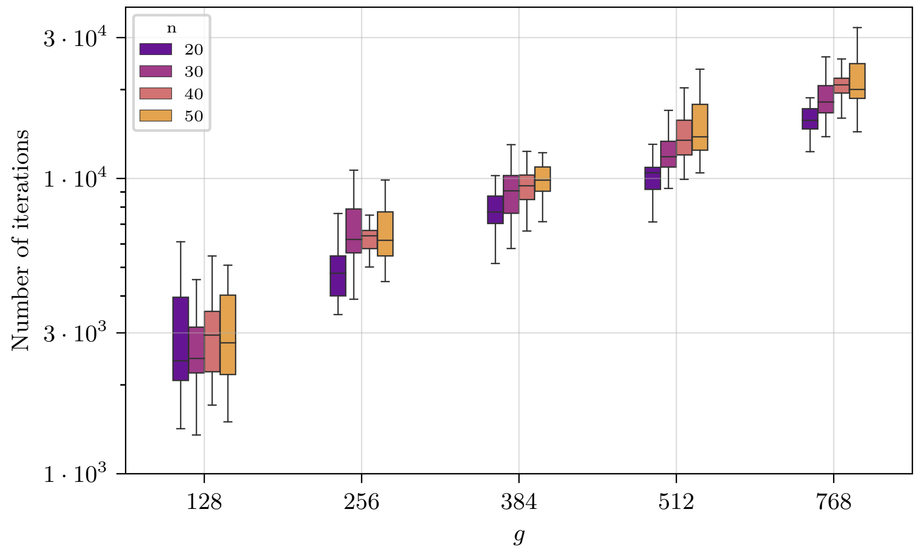
Rysunek 4: Wykres poziomu synchronizacji instancji Alice i Bob razem z atakującym Eve dla parametrów o dużej wartości

4.2 Liczba iteracji

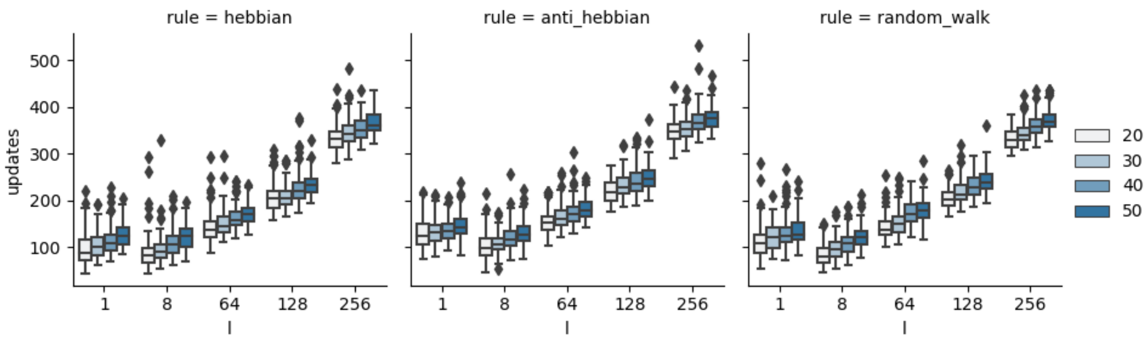
Rysunek 5 przedstawia liczbę iteracji PPM dla ustalonego $k = 2$ w zależności od parametrów n oraz g. Na każdy punkt na przypada 25 niezależnych uruchomień algorytmu z wyłączeniem wartości odstających (w których liczba iteracji przekraczała 10^5, co sporadycznie zdarza się w przypadku wylosowania „nieszczęśliwych" wag).

Na liczbę iteracji konieczną do zsynchronizowania drzew ma wpływ parametr n – im więcej neuronów w warstwie wejściowej, tym więcej iteracji jest konieczne do skutecznej wymiany klucza. Najbardziej istotnym czynnikiem determinującym liczbę iteracji jest jednak oczekiwana długość klucza w bitach, czyli parametr g. Jest to jedna z głównych wad Parity Machines – do wymiany klucza o długości 512 bitów wymagana jest liczba iteracji rzędu 10^4 (dodatkowo, w przypadku PPM, każda iteracja składa się z przesłania wektorów π i x, a także wartości τ_A oraz τ_B, co generuje duży narzut komunikacyjny algorytmu).

Rysunek 6 zawiera zestawienie liczby iteracji w zależności od parametrów n oraz l dla ustalonego $k = 2$ oraz reguł aktualizacji przy 100 próbach dla każdej kombinacji parametrów. Zestawienie pokazuje, że reguła aktualizacji nie wpływa istotnie na czas synchronizacji, może natomiast wpływać na bezpieczeństwo.



Rysunek 5: Liczba iteracji Permutation Parity Machine



Rysunek 6: Porwónanie metod aktualizacji przy ustalonym k=2 w zależności od parametrów n oraz l

5 Rozszerzenia oraz ataki na Parity Machines

Na algorytmy z rodziny Parity Machines zostało przeprowadzonych wiele ataków, niektóre z nich są skuteczne. Zaprezentowano również wiele modyfikacji i ulepszeń dla tych algorytmów. Poniżej prezentujemy listę publikacji poruszających temat kryptoanalizy oraz rozszerzeń algorytmów Parity Machines:

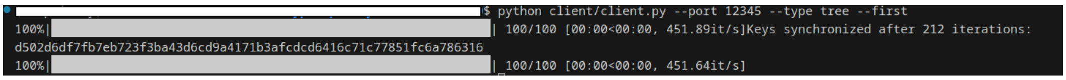
- „Key Exchange Using Tree Parity Machines: A Survey”,
- „Neural Synchronization and Cryptography”,
- „Neural Cryptography”,
- „Analysis of Neural Cryptography”,
- „Successful attack on permutation-parity-machine-based neural cryptography”.

6 Dokumentacja klienta

Program działa w architekturze klient-serwer, jedna instancja nasłuchuje na przychodzące żądanie synchronizacji (tzw. instancja *first*). Komunikacja następuje przez gniazda TCP. Aplikacja napisana jest jako skrypt w języku Python, który przyjmuje następujące argumenty:

- `--first` – program z tym argumentem jest instancją *first* i nasłuchuje na przychodzące żądanie synchronizacji,
- `--host` – adres IP instancji *first* (istotne dla instancji, która *NIE JEST first*),
- `--port` – port, na którym nasłuchuje instancja *first* (istotne dla instancji, która *NIE JEST first*),
- `--type` – typ używanej Parity Machine, może być „permutation” lub „tree”.

Podczas procesu synchronizacji, w konsoli wyświetlany jest postęp w postaci „progress bar”. Po zakończeniu synchronizacji wyświetlany jest ustalony tajny klucz w formacie heksadecymalnym (rysunek 7).



Rysunek 7: Przykład działania aplikacji