

# PIN-kod

Każda karta bankomatowa ma swój 4-cyfrowy numer identyfikacyjny, tzw. PIN (ang. personal identification number). To, czy transakcja z użyciem karty zostanie wykonana, zależy od tego, czy numer karty i jej PIN są zgodne. Zgodność PIN-u i numeru karty sprawdza moduł HSM (ang. hardware security module). Moduł ten otrzymuje trzy parametry: numer karty  $x$ , PIN  $p$  oraz tablicę konwersji  $a$  (16-elementową tablicę liczb od 0 do 9, indeksowaną od 0 do 15). Moduł HSM działa w następujący sposób:

1. szyfruje numer karty  $x$ , otrzymując liczbę  $y$  zapisaną szesnastkowo,
2. z otrzymanej liczby  $y$  pozostawia tylko 4 pierwsze cyfry szesnastkowe,
3. pozostawione cyfry szesnastkowe zamienia na dziesiętne za pomocą tablicy  $a$  (tzn. cyfra  $h$  zamieniana jest na  $a[h]$ , gdzie  $h = A$  jest utożsamiane z 10,  $h = B$  z 11,  $h = C$  z 12,  $h = D$  z 13,  $h = E$  z 14 i  $h = F$  z 15),
4. tak otrzymany 4-cyfrowy numer dziesiętny musi być identyczny z podanym PIN-em.

Standardową tablicą konwersji jest  $a = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5)$ .

Zalóżmy na przykład, że numerem karty jest  $x = 4556\ 2385\ 7753\ 2239$ , a po zaszyfrowaniu uzyskujemy numer szesnastkowy  $y = 3F7C\ 2201\ 00CA\ 8AB3$ . Żeby uzyskać 4-cyfrowy PIN: bierzemy pierwsze 4 cyfry szesnastkowe ( $3F7C$ ) i kodujemy je za pomocą standardowej tablicy konwersji. Wynikiem jest PIN  $p = 3572$ .

Niestety, nieuczciwy pracownik banku lub komputerowy włamywacz może uzyskać dostęp do modułu HSM i próbować odgadnąć PIN manipulując tablicą konwersji.

## Zadanie

Napisz program, który będzie starał się odgadnąć PIN za pomocą zapytań do modułu HSM, przy czym może on zadać co najwyżej 30 zapytań.

## Opis interfejsu

Twój program powinien komunikować się ze „światem zewnętrznym” jedynie poprzez funkcje udostępniane przez moduł `hsm` (`hsm.pas` w Pascalu i `hsm.h` w C/C++). Oznacza to, że nie może on otwierać żadnych plików ani korzystać ze standardowego wejścia/wyjścia.

Przy każdym uruchomieniu Twój program powinien odgadnąć jeden PIN, zgodny z numerem karty znanej modułowi `hsm` przy **standardowej** tablicy konwersji.

Moduł `hsm` udostępnia funkcje: `sprawdz` oraz `wynik`. Ich deklaracje w Pascalu wyglądają następująco:

```
function sprawdz(pin: array of Longint, a: array of Longint): Boolean;  
procedure wynik (pin: array of Longint);
```

## 48 PIN-kod

a w C/C++ następująco:

```
int sprawdz(int pin[], int a[]);  
void wynik(char pin[]);
```

Parametrami funkcji `sprawdz` są: badany PIN (w postaci 4-elementowej tablicy cyfr) oraz tablica konwersji (16-elementowa). Jej wynikiem jest wartość logiczna określająca, czy podany PIN jest zgodny z numerem karty, przy podanej tablicy konwersji. Twój program powinien co najwyżej 30 razy wywoływać funkcję `sprawdz` i dokładnie raz funkcję `wynik`. Wywołanie procedury `wynik` kończy działanie programu. Jej parametrem powinien być PIN zgodny z numerem karty przy standardowej tablicy konwersji.

Żeby przetestować swoje rozwiązanie powinieneś napisać własny moduł `hsm`. Nie jest on jednak elementem rozwiązania i nie należy go przysyłać wraz z programem.

### Przykład

Przykładowa interakcja programu z modulem `hsm` może wyglądać następująco:

```
sprawdz('1111', (0,0,1,1,0,1,0,1,0,0,0,0,1,1,0,1)) = true  
sprawdz('1100', (0,0,0,1,0,1,0,0,0,0,0,0,0,0,1,0,1)) = true  
sprawdz('1100', (0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0)) = false  
sprawdz('1000', (0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0)) = true  
sprawdz('0010', (0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0)) = false  
sprawdz('0001', (0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0)) = true  
wynik('3572')
```

## Rozwiązanie

### Geneza zadania

Historia przedstawiona w treści zadania jest, niestety, prawdziwa. Za godne ubolewania należy uznać, że taki prosty system, jakim jest moduł sprawdzania zgodności PIN-u i numeru karty, zaprojektowano w sposób wprowadzający lukę w ochronie danych. Istotnie, gdyby nie lekkomyślne wzbogacenie interfejsu tego modułu o tablicę konwersji, tylko brutalna siła zdołałaby wydrzeć zeń tajemnicę PIN-u.

Sprawa ta ujrzała światło dzienne całkiem niedawno, kiedy to naukowcy z Uniwersytetu Cambridge (wśród nich były laureat Olimpiady Informatycznej, Piotr Zieliński) zbadali podstawy działania systemów bankomatowych. Odbiło się to szerokim echem w popularnych mediach („Polak złamał PIN-y”, „Niebezpieczne bankomaty”, itp.), nierzadko wyolbrzymiających skalę zagrożenia.

To jest pierwszy przykład na to, że bezpieczeństwo i ochrona danych jest niezwykle trudną do osiągnięcia — bo bardziej subtelną i delikatną — cechą systemu informatycznego. Drugi przykład pochodzi z własnego podwórka. W treści zadania umieszczony został przykład obliczania PIN-u, identyczny z tym zamieszczonym w oryginalnym artykule Piotra Zielińskiego. W konsekwencji uczestnik olimpiady mógł, wpisując w wyszukiwarce internetowej podany w zadaniu ciąg 3F7C 2201 00CA 8AB3, z łatwością odnaleźć ów artykuł i poznać cenne (być może) uwagi, wskazówki i inspiracje.

## Rozwiązanie

Ograniczenie liczby pytań do 30 jasno wskazuje, że metoda brutalnego próbowania wszystkich 10 000 kombinacji PIN-ów prowadzi donikąd. Ale jasno widać po przykładowej interakcji programu z modulem HSM, że kluczem do szybkiego wyznaczenia PIN-u jest podawanie specjalnych tablic konwersji. Istotnie, jeśli użyjemy np. tablicy konwersji o tylko dwóch wartościach 0 i 1, to każda z odpowiedzi TAK i NIE da nam całkiem dużo informacji.

Odgadywanie kodu PIN przypomina bardzo grę Mastermind albo grę „w dwadzieścia pytań”. Za każdym razem zadajemy pytanie typu TAK/NIE, którego odpowiedź zawęży nam zbiór możliwych wyników. Najlepiej, by zadane pytanie dzieliło zbiór potencjalnych wyników na części jak najbardziej zbliżonej wielkości. W pesymistycznym przypadku bowiem (gdy mamy pecha lub gdy moduł grający stosuje tzw. diabelską strategię) odpowiedź na pytanie pozostawi nas z większym zbiorem możliwości.

## Rozwiązanie wzorcowe

W rozwiązaniu wzorcowym szukamy pytania, które dzieli możliwe rozwiązania jak „najbardziej po połówce”. Ograniczamy przy tym się do tablic konwersji zawierających same 0 i 1. Dodatkowo w tablicy a zawsze bierzemy  $a[10+i] = a[i]$ . W ten sposób utożsamiamy A-F z 0-5 i ograniczamy liczbę możliwych tablic do 1024, dzięki czemu możemy przeglądać wszystkie pytania i wybrać z nich najlepsze.

Najprościej byłoby sprawdzić każde pytanie osobno. Mamy 1024 możliwe tablice konwersji i 16 „fałszywych” PIN-ów złożonych z samych 0 i 1, co razem daje  $16 \cdot 1024$  pytań, dla każdego z nich przeglądamy bieżący zbiór możliwych PIN-ów (na początku 10 000).

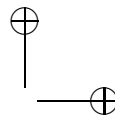
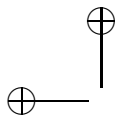
Ale można szybciej: dla każdej tablicy konwersji i każdego prawdziwego PIN-u istnieje dokładnie jeden PIN złożony z 0 i 1, dla którego dostaniemy odpowiedź TAK. Zliczając wystąpienia każdego z tych 16 „fałszywych” PIN-ów, możemy wybrać najlepszy fałszywy PIN do zapytania dla danej tablicy konwersji — taki, dla którego liczba wystąpień jest jak najbliższa połowy. Całe pytanie wybieramy zatem, sprawdzając każdą tablicę konwersji z każdym pozostałym możliwym prawdziwym PIN-em, co upraszcza nam obliczenia o czynnik 16.

Rozwiązanie wzorcowe nie potrzebuje więcej niż dwadzieścia parę pytań dla odgadnięcia któregośkolwiek PIN-u. Najwięcej czasu potrzebuje na zadanie pierwszego pytania, każde następne oblicza coraz szybciej.

## Dalsze optymalizacje

Przed wszystkim zauważmy, że najwięcej obliczeń rozwiązanie wzorcowe wykonuje na samym początku gry, kiedy to zbiór potencjalnych PIN-ów jest największy. Ale te początkowe pytania możemy obliczyć uprzednio i stablicować w kodzie programu. Jeśli ustalimy drzewo gry aż do głębokości  $d$ , to na jego zapamiętanie potrzebujemy  $O(2^d)$  pamięci, ale za to zmniejszamy czas działania o czynnik podobnego rzędu — coś za coś.

Ponadto zachłanna metoda wybierania najlepszego w danej sytuacji pytania nie gwarantuje optymalnej gry. Niemniej jest to wystarczająca technika, by zgadnąć PIN zadając mniej niż 30 pytań.



## 50 *PIN-kod*

### Testy

Rozwiązania były testowane przy użyciu 25 osobnych testów. Każdy test składał się z jednego PIN-u do zgadnięcia. Moduł HSM odpowiadał na pytania zgodnie z tym PIN-em — czyli nie używał „diabelskiej” strategii.

