Kryptoanaliza stosowana 2025

Lista zadań nr 3: funkcje skrótu

Na zajęcia 27 października 2025

Pliki `txt`, `html`, `py` i `ps` występujące w poniższych zadaniach zostały dodane do bieżącego pliku PDF jako załączniki. Na pewno bardziej szpanersko byłoby zamiast tego utworzyć poliglotę PDF/ZIP, ale taki plik byłby mniej wygodny, a może nawet sugerowałby, że nie umiem dodawać załączników do plików PDF.[1]

W wielu zastosowaniach oblicza się kryptograficznie bezpieczne hasze, ale przechowuje i sprawdza jedynie ich prefiksy. W pewnych zastosowaniach (np. jako sumy kontrolne) takie postępowanie jest dozwolone (np. przechowywanie 64, tj. połowy, bitów MD5 jako zamiennika CRC64), w innych absolutnie nie.

Zadanie 1 (1–3 pkt). Oto dwa różne dokumenty w HTML-u:

```
<html>
<head>
<style type="text/css">
   p.fgfgfgfg { font-style: italic; }
</style>
</head>
<body>
<p class="fgfgfgfg">Jest wspaniale!</p>
</body>
</html>
```

```
<html>
<head>
<style type="text/css">
   p.iesfec { font-style: italic; }
</style>
</head>
<body>
<p class="iesfec">Jest fatalnie!</p>
</body>
</html>
```

Autor tych dokumentów tłumaczy się, że padł ofiarą chwilowej zapaści wyobraźni i nie był w stanie wybrać bardziej sensownej nazwy dla klasy akapitu (w przypadku `fgfgfgfg` faktycznie tak było!). Zwróć uwagę, że w obu przypadkach 24-bitowy prefiks skrótu SHA256 to `6c4842`. Czy używając ataku siłowego na prawy plik umiesz znaleźć podobne dokumenty kolidujące na prefiksie długości 28 bitów (2 pkt[2]) lub (często używanym) prefiksie długości 32 bitów (3 pkt[3])? Jeśli nie, to spróbuj za 1 punkt znaleźć dokumenty posiadające wspólny prefiks 24-bitowy, ale zawierające teksty „`Jest bombowo!`” oraz „`Jest kiepsko!`”.

Zadanie 2 (1–3 pkt). Oto dwa różne dokumenty w HTML-u:

```
<html>
<head>
<style type="text/css">
   p { color: #02d5df; }
</style>
</head>
<body>
<p>Jest wspaniale!</p>
</body>
</html>
```

```
<html>
<head>
<style type="text/css">
   p { color: #0b910a; }
</style>
</head>
<body>
<p>Jest wspaniale!</p>
</body>
</html>
```

Zwróć uwagę, że w jednym tekst jest turkusowy, a w drugim zielony, ale w obu przypadkach 32-bitowy prefiks skrótu SHA256 to `a0e36ffe`. Umiesz znaleźć podobne dokumenty kolidujące na prefiksie długości 48 bitów (2 pkt) lub 64 bitów (3 pkt)? Jeśli nie, to spróbuj za 1 punkt znaleźć dwa podobne dokumenty posiadające wspólny prefiks długości 32 bitów, ale zawierające tekst „`Jest bombowo!`” (kolory muszą być oczywiście inne, ale powinny się wyraźnie odróżniać). Tu łatwo za pomocą *time-memory tradeoff* znacznie przyspieszyć znajdowanie kolizji — wystarczy tworzyć słownik przypisujący kolory wyliczonym haszom i sprawdzać, czy świeżo wyliczony hasz jest już w słowniku, a zatem wykorzystać paradoks urodzin. Zwróć uwagę, że dla prefiksu 64-bitowego 2^{24} możliwych kolorów to za mało dla skonstruowania kolizji! *Birthday attack* wymaga tu też pamięci rzędu 2^{32}, nie będzie więc łatwo…

[1] Por. pytania na StackOverflow o tworzenie poliglotów PDF/ZIP od osób, które nie mają pojęcia, że format PDF pozwala na dodawanie dowolnych załączników!

[2] Mój laptop dał radę.

[3] Mój laptop się zgrzał i nie dał rady — w końcu to tylko Celeron 3050.

Zadanie 3 (2 pkt). Paradoks urodzin można też zastosować do zadania 1. Tam przyda się następująca wersja paradoksu: mamy r chłopców i r dziewcząt. Jakie jest prawdopodobieństwo, że istnieje para złożona z chłopca i dziewczynki mająca urodziny tego samego dnia? (Zob. Tony Crilly, Shekhar Nandy, *The birthday problem for boys and girls*, The Mathematical Gazette, 2, **71**(455):19–22, Mar. 1987.) Czy teraz potrafisz znaleźć hasze kolidujace na pierwszych 32, 48, 64 bitach? Zob. też: Gideon Yuval, *How to swindle Rabin*, Cryptologia **3**(3):187–191, 1979.

Zadanie 4 (1 pkt). Opracuj inną niż powyższe oraz inną niż użycie komentarzy „`<!-- -->`" metodę ukrywania ciągów powodujących kolizje w HTML-u. Mogą być to ciągi ograniczone do pewnego podzbioru znaków (jak powyżej). Efektywne metody poszukiwania kolizji zwykle generują ciągi dowolnych bajtów (spoza zakresu 32–127). Zastanów się, jak można ukryć takie ciągi w HTML-u.

Zadanie 5 (1 pkt). GnuPG jako krótkich identyfikatorów kluczy publicznych (tzw. *short ID*) używa 32--bitowych prefiksów ich haszy. Jest to rozszerzenie GnuPG, gdyż standard OpenPGP (RFC 4880) definiuje *key ID* jako dolne 64 bity hasza SHA-1. Jeśli mam w breloku kilkadziesiąt kluczy znajomych, to prawdopodobieństwo przypadkowej kolizji jest znikomo małe i jest to wygodny sposób identyfikowania kluczy (8 cyfr szesnastkowych to chyba maksimum, które daje się ogarnąć jednym spojrzeniem lub zapamiętać). Problemy pojawiają się, gdy takie krótkie prefiksy są nadużywane. Zajrzyj na strony:

`https://github.com/lachesis/scallion`
`https://evil32.com`
`https://www.youtube.com/watch?v=Ow-YcP_KsIw`

i przygotuj krótkie omówienie ataku na *short ID* przedstawionego przez Richarda Klaftera i Erica Swansona na DEFCON 22 w 2014 roku („Evil32: Check your GPG fingerprints"). Zajrzyj też do LKML z 2016 roku:

`https://lkml.org/lkml/2016/8/15/445`

Serwer `pgp.mit.edu` zwykle nie działa, lepiej używać `keyserver.ubuntu.com` lub `keys.openpgp.org`. Fałszywe klucze już zniknęły z serwerów.

Zadanie 6 (2 pkt). Na serwerach HKP (np. `keyserver.ubuntu.com`) znajduje się następujący klucz OpenPGP:

```
pub   rsa4096 2017-01-04 [SC] [expires: 2025-05-13]
      351E 9CC6 2053 BA23 A0C1  2CF7 575C F393 5989 50B2
uid           [  full  ] Piotr Polesiuk <ppolesiuk@cs.uni.wroc.pl>
uid           [  full  ] Piotr Polesiuk <Piotr.Polesiuk@cs.uni.wroc.pl>
uid           [  full  ] Piotr Polesiuk <piotr.polesiuk@gmail.com>
```

Korzystając z oprogramowania Scallion rozważanego w poprzednim zadaniu wygeneruj fałszywy klucz o tych samych *User ID* oraz *short ID* (w tym przypadku `5989 50B2`).

Zadanie 7 (1–2 pkt). Zajrzyj do pracy: Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, Yarik Markov, *The first collision for full SHA-1*, CRYPTO 2017, LNCS 10401, 570–596 oraz na strony:

`https://shattered.io/`
`https://github.com/corkami/collisions`

Przygotuj krótkie omówienie ataku *Shattered* (1 pkt). Za dodatkowy punkt: wygeneruj parę plików PDF o tym samym haszu SHA-1. Pierwszy z nich powinien zawierać bieżącą listę zadań, a drugi — jakieś śmieszne teksty. (Na Githubie jest sporo projektów wykorzystujących wygenerowaną w 2017 roku kolizję).

Zadanie 8 (1–2 pkt). Zajrzyj do pracy: Gaëtan Leurent, Thomas Peyrin, *SHA-1 is a Shambles. First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust*, USENIX Security 2020. Zajrzyj też na stronę

`https://sha-mbles.github.io/`

Przygotuj krótkie omówienie ataku (1 pkt). Skup się głównie na zastosowaniu wyliczonej kolizji do fałszowania podpisów kluczy OpenPGP. Umiesz zaatakować klucz Piotrka Polesiuka? (Dodatkowy punkt!!!)

Zadanie 9 (2 pkt). Pokaż, że CRC32 nie jest funkcją skrótu bezpieczną kryptograficznie. Oczywiście dla 32 bitów jest możliwy nawet atak siłowy, ale pokaż, że można *błyskawicznie* wyznaczać przeciwobrazy oraz kolizje (w tym *chosen prefix*). Nawet więcej — dla dowolnej wiadomości i założonej sumy kontrolnej (np. `0xDEADBEEF`) można łatwo znaleźć taki czterobajtowy sufiks, że wiadomość z dodanym sufiksem ma tę sumę. Można też np. zamienić nie więcej niż 32 bity wiadomości na przeciwne żeby uzyskać założoną sumę.[4] Ta metoda pozwala np. dla danego nieskopresowanego obrazka zastąpić go dowolnym innym, w którym wystarczy uszkodzić jeden piksel (przy 4-bajtowym kodowaniu koloru RGBA), by sumy CRC32 obu obrazków były równe.

[4] Zob. np. Martin Stigge, Henryk Plötz, Wolf Müller, Jens-Peter Redlich, *Reversing CRC – Theory and Practice*, HU Berlin Public Report, SAR-PR-2006-05, May 2006.

Niech

```
msg = b'This is a very important message.'
```

Wówczas dla

```
suffix = b'\xca\xc8\x98\x8c'
```

mamy

```
>>> '{:08X}'.format(zlib.crc32(msg + suffix))
DEADBEEF
```

Funkcja `crc32` pochodzi z wbudowanego modułu `zlib`, więc jej kod źródłowy w Pythonie nie jest dostępny. Przykładowa uproszczona implementacja tej funkcji jest podana w załączniku. Może się przydać jako punkt wyjścia do napisania programu znajdującego sufiksy takie jak powyższy.

Zadanie 10 (2 pkt). Oto dokument w Postscripcie:[5]

```
%!PS-Adobe-1.0
%%BoundingBox: 0 0 596 842
                      (aaa)(bbb)eq{
/Helvetica findfont 12 scalefont setfont
50 792 moveto
(Tekst pierwszy:) show
50 772 moveto
(Jest wspaniale!) show
}{
/Helvetica findfont 12 scalefont setfont
50 792 moveto
(Tekst drugi:) show
50 772 moveto
(Jest fatalnie!) show
}ifelse
showpage
```

Operator `moveto` zdejmuje dwie liczby ze stosu i ustawia kursor w danej pozycji. Początek układu współrzędnych znajduje się lewym dolnym rogu kartki. Miarą są punkty postscriptowe równe 1/72 cala (a zatem `BoundingBox` definiuje rozmiar kartki A4). Nawiasy okrągłe obejmują napisy. Operator `show` zdejmuje napis ze stosu i umieszcza w miejscu kursora. Operator `showpage` drukuje gotową stronę. Operator `ifelse` zdejmuje ze stosu trzy argumenty i przechodzi do wykonania drugiego bądź trzeciego zależnie od tego, czy pierwszy ma wartość `true`, czy `false`. Operator `eq` zdejmuje ze stosu dwa argumenty i je porównuje. Wynik porównania (`true` lub `false`) odkłada na stos.

Początek trzeciego wiersza zawiera 21 spacji tak, by pierwsza litera ciągu `aaa` wypadła na początku 64-bajtowego bloku (plik ma uniksową konwencję znaków nowego wiersza). Korzystając z narzędzi dostępnych w Internecie można w ułamku sekundy wyznaczyć dwublokową kolizję MD5 dla 64-bajtowego prefiksu

```
%!PS-Adobe-1.0
%%BoundingBox: 0 0 596 842
                      (
```

(ostatni wiersz nie jest zakończony znakiem nowego wiersza). Jeśli kolidujące ciągi wstawimy, odpowiednio, zamiast `aaa` i `bbb`, to dokument wyświetli drugi tekst. Jeśli drugi ciąg wstawimy również zamiast `aaa`, to hasz dokumentu się nie zmieni, ale zostanie wyświetlony pierwszy tekst.

Skonstruuj taką parę plików postscriptowych. Przeglądarki (np. `gv`) akceptują w okrągłych nawiasach dosyć dowolne ciągi, byleby nie zawierały nawiasu zamykającego. Jeśli wygenerowana kolizja nieszczęśliwie zawiera nawias zamykający, to musisz np. nieco zmienić prefiks. Zauważ, że raz wygenerowana kolizja pozwala budować pary dokumentów o tym samym haszu zawierające dowolne teksty.

[5] Zobacz: Stefan Lucks, Magnus Daum, *The Story of Alice and her Boss: Hash Functions and the Blind Passenger Attack*, EUROCRYPT 2005 ramp session.

Zadanie 11 (1 pkt). Nawet w czystym pliku tekstowym możemy w niewidoczny sposób sterować wartością hasza. Oto jeden z bardziej znanych wierszy Stanisława Lema, pochodzący z „Cyberiady” („Wyprawa pierwsza A czyli Elektrybałt Trurla”)[6]:

> Nieśmiały cybernetyk potężne ekstrema
> Poznawał, kiedy grupy unimodularne␣
> Cyberiady całkował w popołudnie parne,␣
> Nie wiedząc, czy jest miłość, czy jeszcze jej nie ma.
>
> Precz mi, precz, Laplasjany z wieczora do ranka,␣
> I wersory wektorów z ranka do wieczora!
> Bliżej, przeciwobrazy! Bliżej, bo już pora␣
> Zredukować kochankę do objęć kochanka!
> ␣
> On drżenia wpółmetryczne, które jęk jednoczy,
> Zmieni w grupy obrotów i sprzężenia zwrotne,
> A takie kaskadowe, a takie zawrotne,␣
> Że zwarciem zagrażają, idąc z oczu w oczy!␣
> ␣
> Ty, klaso transfinalna! Ty, silna wielkości!␣
> Nieprzywiedlne continuum! Praukładzie biały!␣
> Christoffela ze Stoksem oddam na wiek cały␣
> Za pierwszą i ostatnią pochodną miłości.
> ␣
> Twych skalarnych przestrzeni wielolistne głębie␣
> Ukaż uwikłanemu w Teoremat Ciała,
> Cyberiado cyprysów, bimodalnie cała
> W gradientach, rozmnożonych na loty gołębie!
>
> O, nie dożył rozkoszy, kto tak bez siwizny
> Ani w przestrzeni Weyla, ani Brouwera
> Studium topologiczne uściskiem otwiera,
> Badając Moebiusowi nie znane krzywizny!
>
> O, wielopowłokowa uczuć komitanto,
> Wiele trzeba cię cenić, ten się dowie tylko,
> Kto takich parametrów przeczuwając fantom,
> Ginie w nanosekundach, płonąc każdą chwilką!
>
> Jak punkt, wchodzący w układ holonomiczności,
> Pozbawiany współrzędnych zera asymptotą,
> Tak w ostatniej projekcji, ostatnią pieszczotą
> Żegnany — cybernetyk umiera z miłości.

Udało mi się spowodować, by hasz SHA256 tego pliku zawierał na początku pięć zer:

```
000009d14328bb69c7156c1a7603e962cc9c6e8e28ff73a44e9450200f30cc63
```

Sztuczka polega na tym, że w wierszach 2, 3, 6, 8, 10, 13, 14, 15, 16, 17, 18, 20 i 21 przed znakiem końca wiersza dodałem niewidoczną[7] spację.[8] Można argumentować, że te spacje powstały przypadkiem podczas pisania tekstu...[9]

Spróbuj w ten sposób osiągnąć 6 lub więcej zer na początku. Rozważ inne hakerskie zastosowania tego pomysłu (pliki źródłowe programów z różnym formatowaniem lub obocznością spacje/tabulacje itp.). Zob. też Gideon Yuval, *How to swindle Rabin*, Cryptologia **3**(3):187–191, 1979 (Appendix B, drugi akapit).

Zadanie 12 (1 pkt). W załączniku `zad12.txt` mamy kolejny plik tekstowy, zawierający tym razem inwokację z „Pana Tadeusza”, w którym w niewidoczny sposób dodano inną wiadomość. Umiesz ją odczytać?

[6] „Elektrybałt” to po naszemu ChatGPT.
[7] Zob. załącznik `zad11.txt`.
[8] Numery wierszy ustalił mój laptop udając koparkę bitcoinów.
[9] W Vim-ie przydaje się komenda `:set list`.

Zadanie 13 (1 pkt). Zajrzyj do pracy: John Kelsey, Tadayoshi Kohno, *Herding Hash Functions and the Nostradamus Attack*, EUROCRYPT 2006, LNCS 4004, 183–200. Opisz na czym polega atak Nostradamusa i w jaki sposób można generować „sensowne" kolizje. Opisz krótko, na czym polega idea zaganiania (*herdingu*). Czy da się ją wykorzystać do kolidowania prefiksów?

Zadanie 14 (1 pkt). Oto krótki program w Pythonie:

```
import hashlib

def hash(s):
    m = hashlib.sha256()
    m.update(s)
    return m.digest()

def xor3(u,v,w):
    return bytes(map(lambda u, v, w: u^v^w, u, v, w))

a =    b'FOO-0x000000003B1BD2039' + b' ' * 53 + bytes.fromhex('dd3ff46f')
b =    b'BAR-0x00000000307238E22' + b' ' * 53 + bytes.fromhex('a80da323')
c = b'FOOBAR-0x000000001BB6C4C9F' + b' ' * 50 + bytes.fromhex('b01d7c21')
xyz = map(hash, (a,b,c))
uvw = map(hash, xyz)
xor3uvw = xor3(*uvw)

print(xor3uvw.hex())
```

Program wypisuje

```
ccb9cef8d7d1c0c67d81536695eeff7e0000000000000000000000000000000000
```

O co chodzi? Przeczytaj: Mellila Bouam, Charles Bouillaguet, Claire Delaplace, Camille Noûs, *Brute-Force Cryptanalysis with Aging Hardware: Controlling Half the Output of SHA-256*, 2021. Przygotuj krótkie omówienie.[10]

[10] Tytuł jest mocno przesadzony, a praca niezbyt głeboka, ale dosyć zabawna.