



**Wydział
Fizyki**

POLITECHNIKA WARSZAWSKA

Podsumowanie wyników uzyskanych podczas laboratorium z KiBIdF

Wykonali:

**Maciej Czarnecki
Denys Morokov**

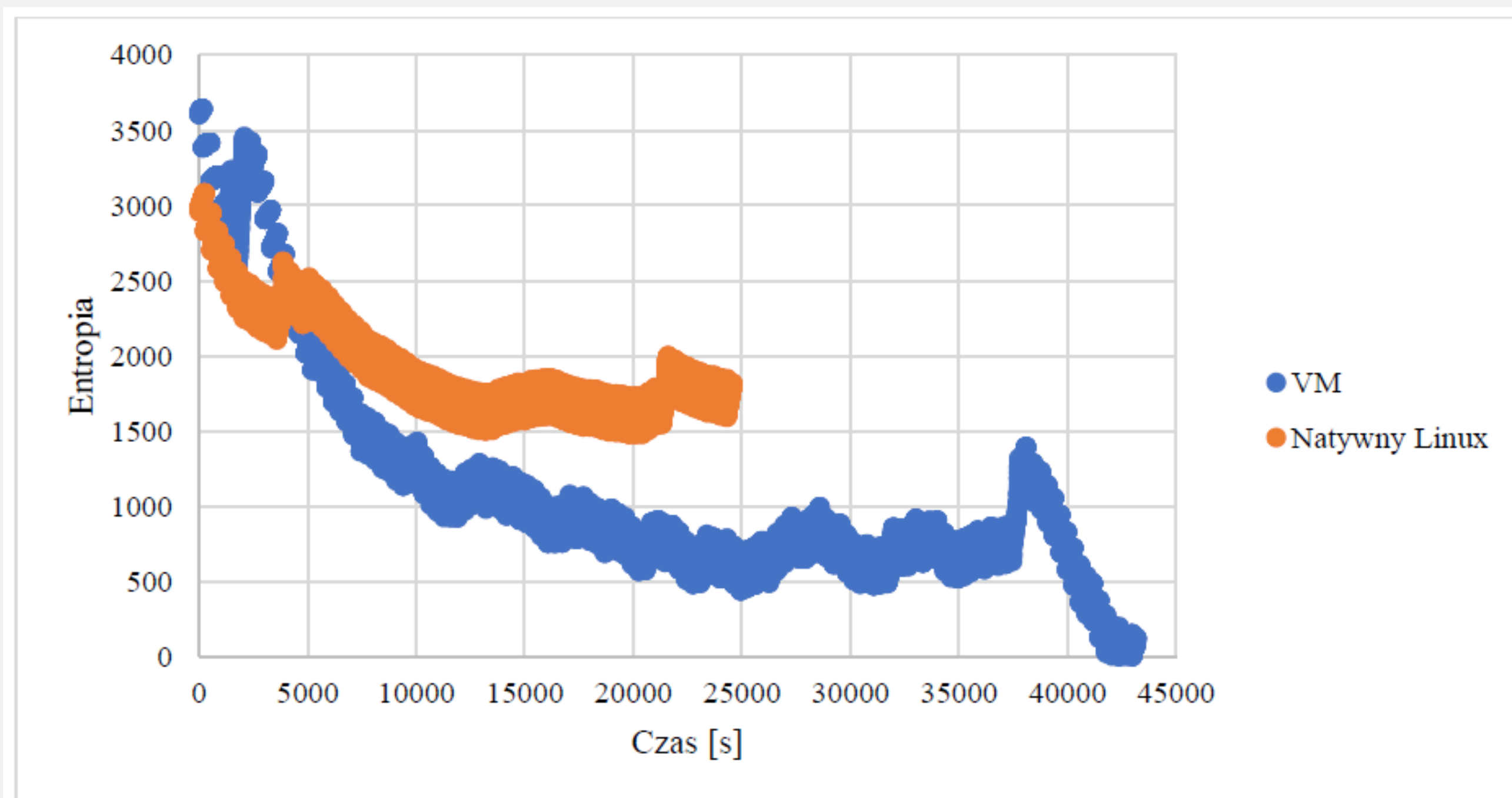
Fizyka techniczna, II stopień, EDiMI

**Politechnika
Warszawska**



Zadanie 1. Zmienność dostępnej entropii programowej.

- **Cel zadania:** zbadanie zmienności dostępnej entropii programowej poprzez napisanie skryptu do jej sprawdzenia.
- **Co zrobiono:** napisano skrypt w Python do sprawdzenia entropii. Pomiar wykonano na dwóch komputerach, gdzie na jednym system operacyjny Linux jest zainstalowany natywnie, natomiast na drugim – przez użycie wirtualnej maszyny.
- **Wyniki:** na rysunku 1.



Rys. 1. Pomiar entropii programowej w ciągu do 12h (odczyt co 3 s)

- **Podsumowanie:** korzystanie z komputera powoduje przyrost entropii. Dla obu systemu zaobserwowano spadek entropii spowodowany odczytem pliku `entropy_avail`, gdyż istnieje mechanizm pompowania entropii z `input_pool` do puli wyjściowej, która obsługuje `random` i `urandom` (`entropy_avail` ocenia entropię `input_pool`).

Zadanie 2. Metryki losowości, testy FIPS 140-2 generatorów liczb pseudolosowych.

- **Cele zadania:** 1) oprogramowanie liniowego generatora kongruentnego, a następnie wykonanie testu FIPS na wygenerowanych liczbach; 2) zbadać jak poprawić entropię systemowego generatora od chwili startu.
- **Co zrobiono:** oprogramowano liniowy generator kongruentny w języku Python. Skrypt generuje 10 ciągów losowych o długości 4096 bitów, które zapisano do pliku txt. Następnie w celu wykonania testu FIPS zainstalowano pakiet Rng-tools. Znaleziono w literaturze 3 sposoby na poprawienie ilości entropii systemowej.
- **Wyniki:** Rysunek 2a przedstawia wynik testu otrzymany na maszynie wirtualnej, a rysunek 2b na natywnym Linuxie.

Sposoby na poprawianie entropii systemowej

Po pierwsze, można użyć specjalnego sprzętu (TPM, Trusted Platform Module) lub instrukcji procesora, takich jak RDRAND (dostępny w procesorach Intel IvyBridge i Haswell). Można wykorzystać „rngd” z pakietu rng-tools, który odczytuje entropię z TPM i wypełnia tę entropią pulę entropii jądra.

Po drugie, można skorzystać z pakietu Haveged. Ten pakiet wykorzystuje algorytm Havage, który generuje entropię na podstawie liczników i stanów procesora. Ze względu na złożoną, wielopoziomową konstrukcję procesorów ten sam kod jest zawsze wykonywany z różną prędkością, a ta niespójność jest podstawą algorytmu Havage.

Trzecim sposobem jest wykorzystanie BitFolk, który ma kilka kluczy Entropy Electronics Simtec podłączonych do różnych hostów. Klucze generują entropię z losowego przepływu elektronów, a demon odczytuje entropię z każdego klucza przez USB i podaje go na porcie TCP.

```
master@master-VirtualBox: ~/Pulpit/krypto/KiBIdF/lab2
Plik Edycja Widok Wyszukiwanie Terminal Pomoc
master@master-VirtualBox:~/Pulpit/krypto/KiBIdF/lab2$ rngtest < krypto.txt
rngtest 5
Copyright (c) 2004 by Henrique de Moraes Holschuh
This is free software; see the source for copying conditions. There is NO warra
nty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

rngtest: starting FIPS tests...
rngtest: entropy source drained
rngtest: bits received from input: 98720
rngtest: FIPS 140-2 successes: 0
rngtest: FIPS 140-2 failures: 4
rngtest: FIPS 140-2(2001-10-10) Monobit: 4
rngtest: FIPS 140-2(2001-10-10) Poker: 4
rngtest: FIPS 140-2(2001-10-10) Runs: 4
rngtest: FIPS 140-2(2001-10-10) Long run: 0
rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
rngtest: input channel speed: (min=3.725; avg=5.731; max=9.313)Gibits/s
rngtest: FIPS tests speed: (min=32.493; avg=34.901; max=36.469)Mibits/s
rngtest: Program run time: 2287 microseconds
master@master-VirtualBox:~/Pulpit/krypto/KiBIdF/lab2$
```

Rys. 2a. Wynik testu FIPS dla 10 ciągów losowych otrzymany na maszynie wirtualnej


```
maciej@maciej:~$ rngtest -c 1000 < ~/KlBIdF/lab2/krypto.txt
rngtest 5
Copyright (c) 2004 by Henrique de Moraes Holschuh
This is free software; see the source for copying conditions.  There is NO warra
nty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

rngtest: starting FIPS tests...
rngtest: entropy source drained
rngtest: bits received from input: 98664
rngtest: FIPS 140-2 successes: 0
rngtest: FIPS 140-2 failures: 4
rngtest: FIPS 140-2(2001-10-10) Monobit: 4
rngtest: FIPS 140-2(2001-10-10) Poker: 4
rngtest: FIPS 140-2(2001-10-10) Runs: 4
rngtest: FIPS 140-2(2001-10-10) Long run: 0
rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
rngtest: input channel speed: (min=4.657; avg=4.967; max=6.209)Gibits/s
rngtest: FIPS tests speed: (min=35.852; avg=37.090; max=37.769)Mibits/s
rngtest: Program run time: 2221 microseconds
```

Rys. 2b. Wynik testu FIPS dla 10 ciągów losowych otrzymany na natywnym Linuxie

- **Podsumowanie:** generator nie przeszedł testów FIPS, ponieważ generator jest liniowy i liczba pseudolosowa w kroku i generowana przez generator zależy od liczby losowanej w kroku $i-1$.

Zadanie 3. Konsekwencje braku losowości: generacja kluczy RSA.

- **Cel zadania:** napisanie prostego skryptu do ściągania kluczy publicznych wybranych domen oraz porównywania tych kluczy. Lista wykorzystanych domen znajduje się w pliku .XML.
- **Co zrobiono:** został napisany w języku Python z wykorzystaniem biblioteki OpenSSL do ściąganie tych kluczy i ich porównanie.
- **Wyniki:** rysunek 3a – lista domen, rysunek 3b – przykładowe klucze publiczne.
- **Podsumowanie:** nie wykryto duplikatów klucza publicznego w większości domen. Wyjątkiem był jedynie przypadek `www.google.pl` i `www.youtube.pl`. Prawdopodobnie spowodowane jest to tym że youtube jest częścią googla i obie witryny korzystają z tych samych kluczy.


```
e3_domens.xml X
<data>
  <domeny>
    <domena id="www.sinoptik.pl"></domena>
    <domena id="www.fizyka.pw.edu.pl"></domena>
    <domena id="www.w3schools.com"></domena>
    <domena id="www.stackoverflow.com"></domena>
    <domena id="www.the-flow.ru"></domena>
    <domena id="www.habr.com"></domena>
    <domena id="www.podatki.gov.pl"></domena>
    <domena id="www.pkobp.pl"></domena>
    <domena id="www.gov.pl"></domena>
    <domena id="www.mon.gov.ua"></domena>
    <domena id="www.github.com"></domena>
    <domena id="www.nawa.gov.pl"></domena>
    <domena id="www.wp.pl"></domena>
    <domena id="www.onet.pl"></domena>
    <domena id="www.kwejk.pl"></domena>
    <domena id="www.udemy.com"></domena>
    <domena id="www.youtube.com"></domena>
    <domena id="www.google.com"></domena>
    <domena id="www.facebook.com"></domena>
    <domena id="www.yandex.ru"></domena>
  </domeny>
</data>
```

Rys. 3a. Wybrane domeny

```
www.udemy.com
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtmUC3p/7EypE+Dz+gNC
lvpiw/myFFNwB1VRvCZ7ZEGXXLIg2XQnJdjAcc8dGgJcivDvVEWLSzXZyi4tBL02
deIXDYRmHrUrSCLB449EjjAeCrqhC904BIUDbVV6+caK5y0gTsSidetCAy1G1BMt
6INDdjBRfsYntvhLGMLL5uyZ38dinGfwmBz+nisgQ1TcQNaec5gsLb0Ku0/RB3+b
ciOMDghhjkSyfacEaiYfdIB3NnWu9uXK/C68KN6440q9gujv9oQ6fFrGhiu1bT1T
8sTmjV5o0SuzI9Q8xpxJ4Iiq4L+4JUWdtNRLLx+V2RSTrNgGPLfqLEcX02cuPdZI
uwIDAQAB
-----END PUBLIC KEY-----

www.youtube.com
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAzWJP5cMThJgMBEtvRKK1
7N6ZcZAbKDVAtnBNnRhIgSItXxCzKtt9rp2RHkLn76oZjdNO25EPp+QgMiWU/rkk
B00Y180ahw5fi8s+K9dRv6i+gS0iv2j1IeW/S0h0swUUDH0JXFkEPKILzp15ML7w
dp5kt93vHxa7Hsw0tAxEz2WtxMdez3Cg03s1s20w13W03iI+kCt7HyvhGy2aRP
LhJfeABpQr0Uku3q6mtomy2cgFawekN/X/aH8KknX799MPcuWutM2q88mtUEBsuZ
my2nsjk9J7/yhhCRDz0V/yY8c5+1/u/rWuwwkZ21gzGp4xBBfhXdr6+m9kmwWCUm
9QIDAQAB
-----END PUBLIC KEY-----

www.google.com
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAzWJP5cMThJgMBEtvRKK1
7N6ZcZAbKDVAtnBNnRhIgSItXxCzKtt9rp2RHkLn76oZjdNO25EPp+QgMiWU/rkk
B00Y180ahw5fi8s+K9dRv6i+gS0iv2j1IeW/S0h0swUUDH0JXFkEPKILzp15ML7w
dp5kt93vHxa7Hsw0tAxEz2WtxMdez3Cg03s1s20w13W03iI+kCt7HyvhGy2aRP
LhJfeABpQr0Uku3q6mtomy2cgFawekN/X/aH8KknX799MPcuWutM2q88mtUEBsuZ
my2nsjk9J7/yhhCRDz0V/yY8c5+1/u/rWuwwkZ21gzGp4xBBfhXdr6+m9kmwWCUm
9QIDAQAB
-----END PUBLIC KEY-----
```

Rys. 3b. Ściągnięte klucze

Zadanie 4. Faktoryzacja kluczy RSA – wg. Bernstein, Heringer, Lange

- **Cel zadania:** napisanie prostego crawler'a do kluczy, który dla zadanej listy domen ściągnie ich klucze publiczne (otrzymane w poprzednim zadaniu) i sprawdzi metodą batchGCD czy da się odtworzyć klucze prywatne.
- **Co zrobiono:** algorytm batchGCD (<http://facthacks.cr.yp.to/batchgcd.html>) został zaimplementowany w Sage. Jako wejście skrypt przyjmuje plik tekstowy ze sprasowanymi kluczami publicznymi.
- **Wyniki:** rysunek 4.
- **Podsumowanie:** nie wykryto duplikatów klucza publicznego w większości domen. Wyjątkiem był jedynie przypadek `www.google.pl` i `www.youtube.pl`. Prawdopodobnie spowodowane jest to tym że youtube jest częścią googla i obie witryny korzystają z tych samych kluczy.

```
SageMath version 8.1, Release Date: 2017-12-07  
Type "notebook()" for the browser-based notebook interface.  
Type "help()" for help.
```

```
sage: load("lab.py")  
[1, 1, 1, 1, 1, 1, 1, 1, 1]  
sage: □
```

Rys. 4. Wynik zastosowania algorytmu

- **Podsumowanie:** wynik w postaci listy dziewięciu 1., świadczy czy to o braku wspólnych GCD, więc nie udało się odtworzyć private key.

Ciekawostka

W trakcie wykonania zadania natrafiono na jedną prezentację z konferencji DEF CON 26 (<https://www.youtube.com/watch?v=Z7cLRE6t1Q8&t=4s>), podczas której została przedstawiona strona firmy, na której można przetestować klucz publiczny na wrażliwość do GCD i ROCA - <https://keylookup.kudelskisecurity.com>

Informacje ze strony:

„Zbieramy klucze publiczne RSA z różnych źródeł i analizujemy, czy któryś z tych kluczy ma wspólne czynniki.

Najnowsze wyniki pokazują, że na 1300 jest 1 para kluczy, których bezpieczeństwo jest zagrożone, ponieważ dzieli wspólny czynnik z inną publicznie dostępną parą kluczy.

Nasza baza danych kluczy publicznych zawiera ponad 340 milionów unikalnych modułów RSA. Testowanie nowych kluczy w naszym pełnym zestawie danych zajmuje tylko kilka minut.

Wrażliwe klucze, które zidentyfikowaliśmy, pochodzą z kluczy SSH gitlab.com, kluczy PGP, certyfikatów stron HTTPS X.509 i innych źródeł.”

keylookup.kudelskisecurity.com/submit

keylookup

Submit key

About us

Help

KUDELSKI
SECURITY

Key container type: x509

Keys in container: 1

Status:

RSA keys

Key #	Key type	Key size	Vulnerable to GCD	Vulnerable to ROCA
1	rsa	2048	Unknown - Your key has been added to the processing queue. Please check again later.	False

© Copyright 2018, Nagravision SA, made by Kudelski Security.

Rys. 4a. Wynik dla klucza publicznego z www.fizyka.pw.edu.pl

Zadanie 5. Kryptografia krzywych eliptycznych

- **Cel zadania:** napisanie prostego programu szyfrującego zawartość przy użyciu krzywych eliptycznych z użyciem wybranej biblioteki kryptograficznej. Program ma umożliwiać ustawienie własnych parametrów dla EC. Należy ustawić parametry inne niż domyślne dla biblioteki ale prawidłowe dla EC.
- **Co zrobiono:** krótki wstęp o kryptografii na krzywych eliptycznych, wyjaśniono dlaczego EC bezpieczniejsze od innych szyfrów; znaleziono w literaturze i opracowano kilka skryptów do kryptografii na krzywych eliptycznych.
- **Wyniki:** ewentualnie pokazać sprawozdanie z zadania.
- **Podsumowanie:** Zaimplementowano skrypt do podpisu wiadomości dla wybranej EC, opisano w skrócie zasadę działania kryptografii krzywych eliptycznych (algorytmy ECDH i ECDSA).
- **Polecamy:** <https://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/>

Dlaczego EC są bezpieczniejsze od innych szyfrów kryptograficznych

Problem logarytmu dyskretnego.

Jeśli znamy P i Q , to ile powinno wynosić k , żeby $Q = kP$. ECC jest interesujące pod tym względem, że obecnie problem dyskretnego logarytmu dla krzywych eliptycznych wydaje się „bardziej skomplikowany” w porównaniu z innymi podobnymi zadaniami stosowanymi w kryptografii. Oznacza to, że potrzebujemy mniej bitów dla k , aby całość mogła uzyskać taki sam poziom ochrony, jak w innych systemach kryptograficznych.

Dobrze to pokazuje tabela poniżej, która porównuje rozmiar klucza RSA z rozmiarem klucza EC (w bitach), tabela udostępniona przez NIST.

Rozmiar klucza RSA [bit]	Rozmiar klucza EC [bit]
1024	160
2048	224
3072	256
7680	384
15360	521

Rys. 5. Porównanie rozmiarów kluczy

Jak widać, nie ma liniowej zależności między rozmiarem klucza RSA a kluczem EC (innymi słowy: jeśli podwoimy rozmiar klucza RSA, nie będziemy musieli podwajać rozmiaru klucza EC). Tabela mówi nam, że **EC** nie tylko **zużywa mniej pamięci**, ale także generowanie kluczy z logowaniem w niej **jest znacznie szybsze**

Zadanie 6. Kryptografia post-kwantowa, algorytm New Hope

- **Cel zadania:** uruchomienie dowolnej biblioteki realizującej algorytm kryptografii postkwantowej i zademonstrowanie zaszyfrowania i odszyfrowania komunikatu.
- **Co zrobiono:** utworzono skrypt w Pythonie z wykorzystaniem biblioteki PyNewHope, zademonstrowano zaszyfrowanie i odszyfrowanie przykładowego komunikatu.
- **Wyniki:** działający skrypt. 😊

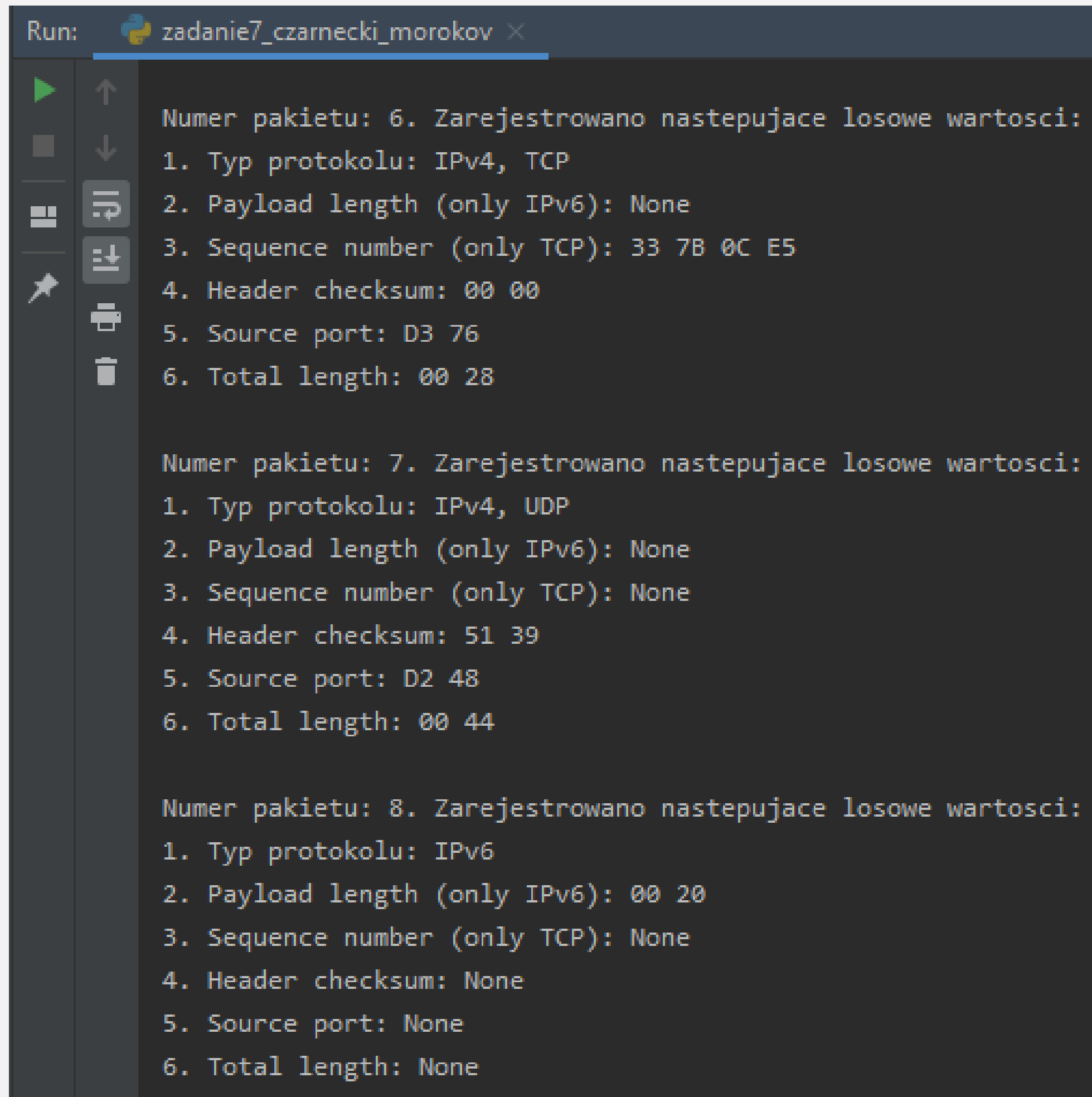
Zadanie 7. Pakiety sieciowe Ethernet jako źródło sygnału losowego

- **Cel zadania:** napisanie programu śledzącego ruch z wybranego hosta i rejestrujący kolejne wartości losowych bitów z pól nagłówek sieciowych.
- **Co zrobiono:** utworzono skrypt w Pythonie z wykorzystaniem biblioteki Scapy, w którym rejestruje wybrane w następujący sposób liczby losowe.

Najpierw sprawdzano typ protokołu internetowego (IPv4 lub IPv6).

- a. Dla **IPv4** dla dwóch typów protokołów TCP i UDP wyznaczono następujące wartości losowe:
 - i. **TCP:** source port (34-35 bajt), header checksum (24-25 bajt), sequence number (38-41 bajt), total length (16-17 bajt).
 - ii. **UDP:** source port (34-35 bajt), header checksum (24-25 bajt), total length (16-17 bajt).
- b. Dla **IPv6** wyznaczono następujące wartości losowe: source port, checksum, transaction ID, payload length (w skrypcie z powodu skomplikowanego określenia konkretnego typu protokołu IPv6 wyznaczano tylko payload length).

Rys. 6. Wybrane parametry losowe do rejestracji



```
Run: zadanie7_czarnecki_morokov X

Numer pakietu: 6. Zarejestrowano nastepujace losowe wartosci:
1. Typ protokolu: IPv4, TCP
2. Payload length (only IPv6): None
3. Sequence number (only TCP): 33 7B 0C E5
4. Header checksum: 00 00
5. Source port: D3 76
6. Total length: 00 28

Numer pakietu: 7. Zarejestrowano nastepujace losowe wartosci:
1. Typ protokolu: IPv4, UDP
2. Payload length (only IPv6): None
3. Sequence number (only TCP): None
4. Header checksum: 51 39
5. Source port: D2 48
6. Total length: 00 44

Numer pakietu: 8. Zarejestrowano nastepujace losowe wartosci:
1. Typ protokolu: IPv6
2. Payload length (only IPv6): 00 20
3. Sequence number (only TCP): None
4. Header checksum: None
5. Source port: None
6. Total length: None
```

Rys. 6a. Wynik działania skryptu

Zadanie 8. Pasywna identyfikacja systemów operacyjnych na podstawie własności pakietów sieciowych

- **Cel zadania:** zidentyfikowanie systemu operacyjnego w obserwowanym ruchu sieciowym na podstawie dowolnych parametrów pakietu lub zawartości i znaleźć i uruchomić program wyszukujący i identyfikujący OS hostów w sieci lokalnej.
- **Co zrobiono:** zarejestrowano ruch sieciowy w Wireshark w ciągu 5 minut. Następnie na 2 sposoby zidentyfikowano systemy operacyjne:
 - Napisano skrypt w Pythonie, który na podstawie parametru TTL (Time To Life) określano system operacyjny urządzenia, z którego dany pakiet został wysłany.
 - Użyto pasywny skaner sieciowy p0f.
- **Wyniki:** rysunek 7a i rysunek 7b.

```

Run: zadanie8_czarnecki_morokov X
Calkowita liczba pakietow: 6450
Liczba zidentyfikowanych OS: 2482

Cisco: 92.
Windows: 2296.
Unix: 94.
Android/iOS: 0.
Lumia: 0.

Process finished with exit code 0

```

Rys. 7a. Identyfikacja ze skryptu na podstawie TTL

```

master@master-VirtualBox:~/Pulpit$ p0f -r ruch_sieciowy.pcapng | grep os | sort | uniq -c
1 .-[ 10.2.11.20/51424 -> 52.114.77.33/443 (host change) ]-
1 .-[ 10.2.11.20/51425 -> 104.18.16.5/443 (host change) ]-
1 .-[ 10.2.11.20/51438 -> 52.114.77.33/443 (host change) ]-
1 .-[ 10.2.11.20/51439 -> 104.244.42.129/443 (host change) ]-
1 .-[ 10.2.11.20/51451 -> 5.135.104.110/443 (host change) ]-
1 [+] Closed 1 file descriptor.
25 | os      = ???
6 | os      = Linux 3.x
3 | os      = Windows 7 or 8
26 | os      = Windows NT kernel
8 | os      = Windows NT kernel 5.x
1 | raw_sig = 1:?Cache-Control,Connection=[Keep-Alive],Accept=[*/*],?If-Modified-Si
Accept-Charset,Keep-Alive:Microsoft-CryptoAPI/10.0
1 | raw_sig = 1:Connection=[Keep-Alive],Accept=[*/*],?If-Modified-Since,?If-None-Ma
Accept-Charset,Keep-Alive:Microsoft-CryptoAPI/10.0
2 | raw_sig = 1:Host,Connection=[keep-alive],Upgrade-Insecure-Requests=[1],DNT=[1],
pplication/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.
anguage=[ru,pl;q=0.9]:Accept-Charset,Keep-Alive:Mozilla/5.0 (Windows NT 10.0; Win64; x64)
945.88 Safari/537.36
5 | reason  = os_diff
master@master-VirtualBox:~/Pulpit$

```

Rys. 7b. Identyfikacja na podstawie p0f

Dziękujemy za uwagę!