Kurs administrowania systemem Linux

Zajęcia nr $8\frac{1}{2}$: Szyfrowanie dysków

Instytut Informatyki Uniwersytetu Wrocławskiego

4 maja 2022

Zagrożenia poufności danych zapisanych na nośnikach trwałych

Dostęp fizyczny

- Trwałe przejęcie przez osoby niepowołane:
  - kradzież lub zagubienie sprzętu przenośnego,
  - kradzież z włamaniem sprzętu stacjonarnego.
- Tymczasowy, nieujawniony, często wielokrotny, dostęp osób niepowołanych do sprzętu pozostawionego bez opieki:
  - komputer pozostawiany w biurze w pracy,
  - laptop pozostawiony w pokoju hotelowym,
  - macierz dyskowa w serwerowni.
  - Tzw. *evil maid attack* — możliwość wielokrotnego odczytu i modyfikacji danych.
- Trwałe lub tymczasowe przejęcie przez osoby powołane:
  - kasacja starych nośników i przekazanie do utylizacji,
  - sprzedaż sprzętu używanego,
  - naprawa bądź wymiana sprzętu uszkodzonego.

Dostęp zdalny

- Włamanie poprzez sieć (Internet, Ethernet, WiFi, Bluetooth, itp.) do działającego systemu korzystającego z nośników.

Sposoby ochrony poufności danych zapisanych na nośnikach trwałych

Dostęp fizyczny

- Ochrona fizyczna
  - Zamki, kraty, pracownicy ochrony (głównie serwerownie).
  - Sejfy (nośniki backupowe: taśmy, dyski przenośne itp.).
- Usuwanie danych z wycofanych nośników
  - Zamazywanie nośników (uwaga na ograniczenia, zob. Gutman):
    - firmware'owe: np. SECURE ERASE (ATA) lub FORMAT UNIT (SCSI),
    - programowe: np. dd (aka Disk Destroyer) — nie dla nośników COW.
  - *Shreddery* i *degaussery* — niszczenie wycofanych nośników (wada: zniszczenie samego nośnika).
- Kryptografia
  - Bardzo dobra ochrona w razie kradzieży (ale nie chroni wartości samych nośników).
  - Bardzo łatwy sposób „usunięcia" danych z nośnika — *cryptoshredding*.

Dostęp zdalny

- Kontrola dostępu, firewalle, *intrusion prevention* itp.
- Zasada minimalnych uprawnień, ograniczanie *attack surface*.

Model zagrożeń (*threat model*)

Audyt bezpieczeństwa

- Audyt danych
  - Jakie dane przechowujemy?
  - Kto nie powinien a może uzyskać do nich dostęp?
  - Jakie straty (w tym koszty) wywoła ich a) ujawnienie, b) utrata?
- Audyt zagrożeń
  - W jaki sposób osoby niepowołane mogą uzyskać dostęp do chronionych danych?
  - W jaki sposób dane są narażone na uszkodzenie/utratę?
  - Jakie jest ryzyko powyższych zdarzeń.
- Audyt metod ochrony
  - Jakie są dostępne metody ochrony?
  - Jak kosztowne (czas, pieniądze) są te metody?

Problemy

- Ocena wrażliwości danych jest często trudna
  - Czy `/var/` zawiera jakieś wrażliwe dane?
  - Czy ja kilka lat temu nie zapisałem przypadkiem gdzieś na tym dysku pliku z aktualnym hasłem do banku?

Ochrona danych

Wybór i wdrożenie odpowiednich mechanizmów ochrony danych jest możliwe dopiero *po* wykonaniu audytu bezpieczeństwa i opracowaniu modelu zagrożeń.

Przykłady

- Kupiłem nowy bardzo drogi laptop:
  - chcę na nim przechowywać wrażliwe dane,
  - jest na gwarancji, więc nie można go otworzyć w celu np. wyjęcia dysku,
  - zamierzam nosić go ze sobą w miejscach publicznych.

  Co się stanie, jeśli:
  - płyta główna ulegnie awarii i trzeba go będzie oddać do naprawy,
  - zostanie skradziony?
  - A jeśli dodatkowo zamierzam go zostawiać bez dozoru np. w pokoju hotelowym?
- Mam w domu komputer stacjonarny:
  - nie jest na gwarancji, mogę swobodnie wymieniać jego komponenty,
  - ryzyko kradzieży z włamaniem do mieszkania oceniam jako znikomo małe,
  - służy głównie do przeglądania Internetu, nie trzymam na nim wrażliwych danych.

Jakie mechanizmy ochrony danych w spoczynku oferuje kryptografia?

Szyfrowanie plików

- Niektóre formaty plików oferują natywne szyfrowanie:
  - PDF: AES-128 (w trybie OFB?)
  - Archiwa: ZIP, RAR, 7z (uwaga — niektóre są podatne!)
  - Uwaga na tzw. *export-grade cryptography* (np. PDF: 40-bit)!
- Szyfrowanie dowolnych plików:
  - `openssl`
  - GnuPG

Szyfrowanie drzewa katalogów

- ext4
- ZFS — szyfrowanie datasetów

Szyfrowanie urządzeń blokowych (partycji)

- Linux Device Mapper crypt module (dm-crypt), także w Dragonfly BSD
- Veracrypt
- BSD mają własne rozwiązania, np. GBDE i GELI we FreeBSD (ale też Veracrypt)
- Microsoft: Bitlocker

Szyfrowanie plików: GnuPG

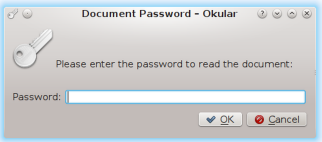
```
gpg --symmetric --verbose --verbose
    --cipher-algo AES256
    --s2k-digest-algo SHA512 --s2k-count 65011712
    --output zaszyfrowany.gpg jawny.txt
```

`--symmetric`	zaszyfruj plik szyfrem symetrycznym
`--verbose --verbose`	wypisuj dużo komunikatów diagnostycznych
`--cipher-algo AES256`	użyj szyfru AES256
`--s2k-digest-algo SHA512`	w PBKDF2 używaj algorytmu SHA512
`--s2k-count 65011712`	wykonaj 65011712 iteracji funkcji mieszającej (maksymalna wartość; domyślnie 65536)
`--output zaszyfrowany.gpg`	nazwa pliku zaszyfrowanego

Szyfrowanie szczególnych rodzajów plików

PDF

```
$ pdftk jawny.pdf output tajny.pdf user_pw PROMPT
Please enter the user password to use on the output PDF.
   It can be empty, or have a maximum of 32 characters:
student.8
```



TXT

- Można użyć dowolnego dobrego programu szyfrującego, np. gpg.
- Wtyczki do vim-a i emacs-a pozwalają otwierać takie pliki wprost w edytorze, który nie zapisuje wówczas tekstu jawnego na dysku.

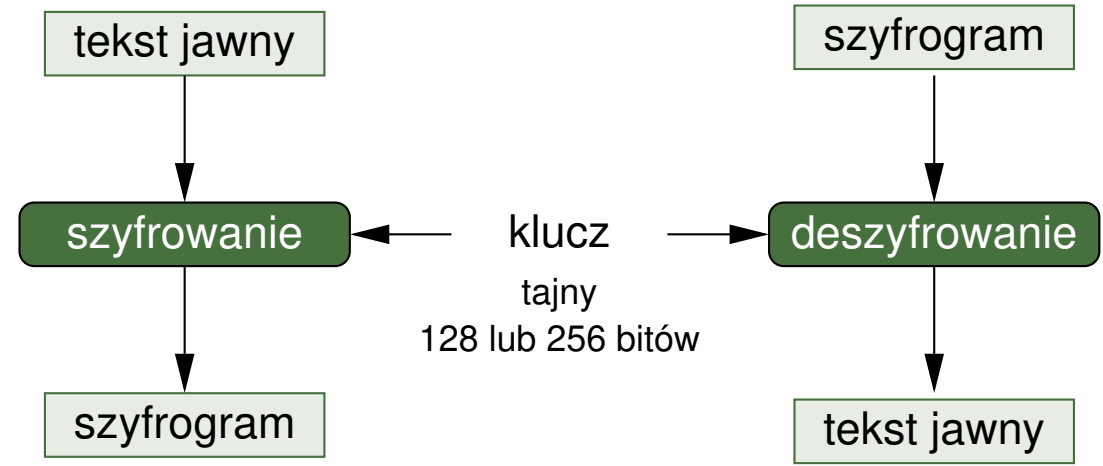
Korzystanie z szyfrowania plików

- Tworzymy plik zaszyfrowany. Plik `jawny.txt` pozostaje na dysku!
- Nie używajcie `rm` żeby go usunąć!
- Lepiej: `shred -vzun0 jawny.txt` (ale nie na dysku SSD!)
- Plik zaszyfrowany można przechowywać na dysku, pendrive itp. Jeśli wpadnie w niepowołane ręce, jego zawartość będzie niedostępna.
- Wady
  - Każdy plik trzeba szyfrować osobno.
  - Wiele programów przetwarzających pliki tworzy kopie zapasowe. Trzeba pamiętać, żeby je też szyfrować lub usuwać.

  Dlatego lepiej szyfrować całe systemy plików lub całe dyski.
- Plik `zaszyfrowany.gpg` można przesłać jako załącznik w poczcie. Hasło należy dostarczyć adresatowi niezależnym kanałem (osobiście, w rozmowie telefonicznej, za pomocą SMS itp.)
  Uwaga: najsłabszym ogniwem jest hasło — musi być silne. To jest kłopot...
  (Dlatego do transmisji danych mamy lepsze metody).

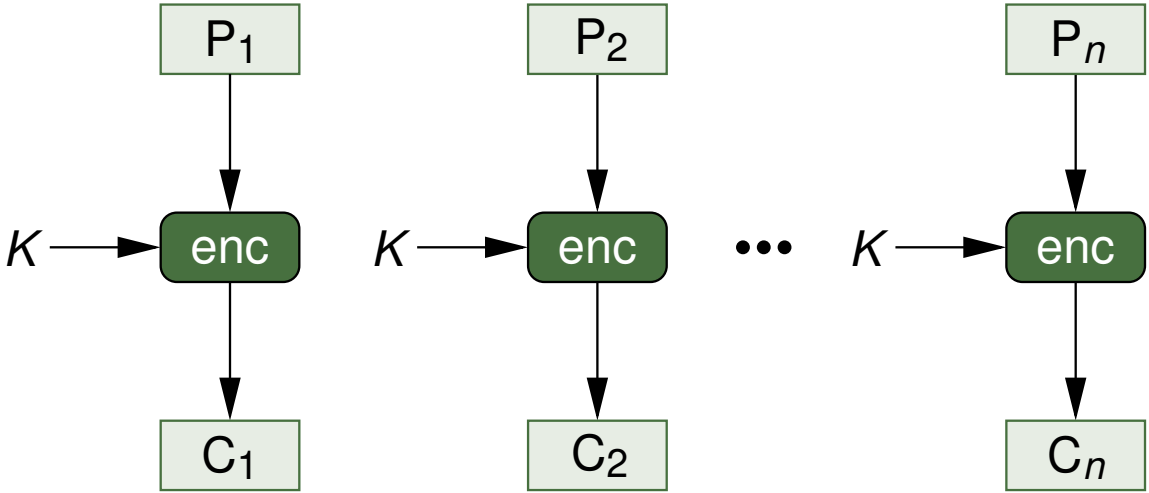
Szyfry blokowe

blok 128 bitów = 16 bajtów



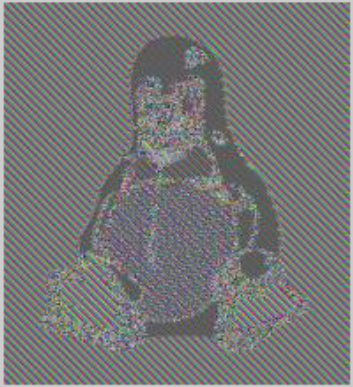
blok tej samej długości

Jak zaszyfrować więcej danych?

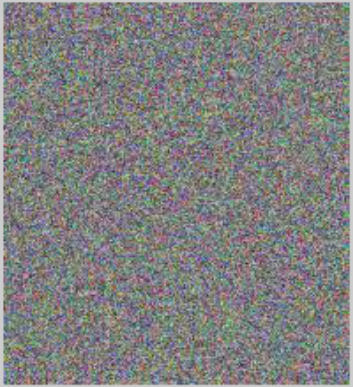


Tryb łączenia bloków

Plaintext



ECB



CBC

Tryby łączenia bloków

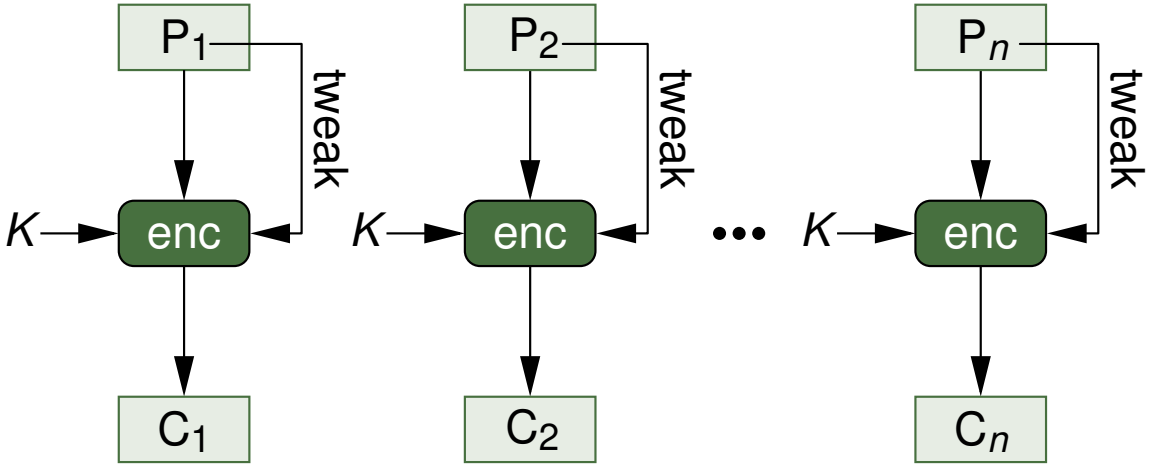
Tryb ECB (Electronic Code Book)

- Jeśli dwa bloki tekstu jawnego mają tę samą zawartość, to ich szyfrogramy też są równe.
- Nie wolno (!) używać, jeśli istnieje ryzyko, że dwa bloki tekstu jawnego mają tę samą zawartość!

Szyfry modyfikowalne (*tweakable ciphers*)

- Inicjalizacja algorytmu szyfrowania/deszyfrowania dla danego klucza jest kosztowna: wiele bloków trzeba szyfrować tym samym kluczem.
- Szyfr modyfikowalny ma dodatkowy argument, który dla każdego bloku może być inny.

Szyfr modyfikowalny w trybie ECB



Współczesne szyfry i tryby wiązania bloków

AES (Advanced Encryption Standard, Rijndael, 1998)

- Jedyny obecnie dopuszczony przez NIST do szyfrowania dysków.
- Blok: 128 bitów (16 bajtów).
- Klucz: 128, 192 (rzadko), 256 bitów.

Szyfr modyfikowalny na bazie AES: tryb AES-XTS

- XEX (xor-encrypt-xor, Rogaway 2004): $C = T \oplus E_K(T \oplus P)$
- XTS (XEX-based tweaked-codebook with ciphertext stealing, NIST 2010):

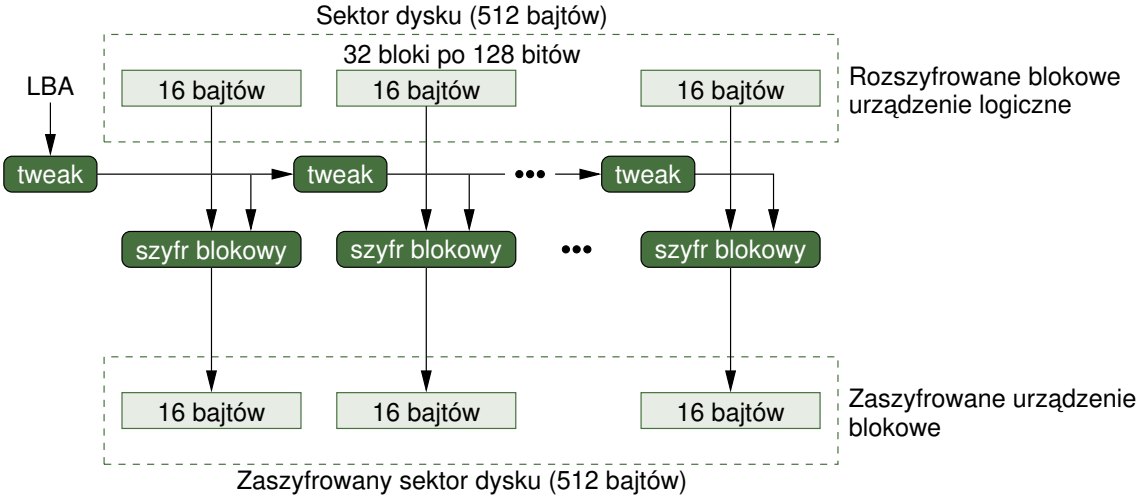
$$\begin{aligned} M_i &= E_{K'}(IV) \cdot \alpha^i \\ C_i &= M_i \oplus E_K(M_i \oplus P_i) \end{aligned}$$

- Obliczenie M_i jest wykonane w ustalonym przez standard ciele Galois. Parametr α jest ustalony przez standard.
- Zwykle IV to 64-bitowy numer sektora, zaś i to numer bloku w sektorze (0–31 dla sektorów 512 B).

Szyfrowanie dysków

- Szyfruje się całe urządzenie blokowe (np. partycję).
- Zwykle szyfruje się sektory 512 B (ale można też większe, np. 4 KiB).
- Każdy sektor jest szyfrowany oddzielnie za pomocą AES-XTS.
- Klucze K i K' (długości 128 lub 256 bitów każdy) są wspólne dla wszystkich sektorów.
- IV to zwykle 64-bitowy numer sektora (LBA).

Szyfrowanie sektora



Własności dobrego szyfrowania

Zasada Kerckhoffsa (1883)

- Bezpieczeństwo systemu kryptograficznego powinno w całości zależeć od tajności klucza.
- Zasada działania, algorytmy i szczegóły implementacji systemu nie muszą być tajne.
- Ich ujawnienie nie może w żaden sposób pomóc w złamaniu szyfru.
- Przeciwnie: projekt systemu powinien być dostępny publicznie (możliwość audytu i wskazania luk).
- Lokalizacja sekretu: jest nim tylko klucz (np. 512 bitów).

Konfuzja i dyfuzja (Claude Shannon 1945)

- Konfuzja: każdy bit szyfrogramu powinien zależeć od wielu bitów klucza.
- Dyfuzja: zmiana pojedynczego bitu tekstu jawnego powinna spowodować zmianę około połowy bitów szyfrogramu.
- AES-XTS świetnie spełnia te warunki (w ramach pojedynczego bloku).

Szyfrowanie dysków w trybie AES-XTS

Zalety i wady AES-XTS

- Zmiana/uszkodzenie pojedynczego bitu powoduje „losową” zmianę całego 16-bajtowego bloku.
- Zmiany są zlokalizowane w ramach jednego bloku (możliwość monitorowania zmian na dysku z granulacją 16-bajtową).

Szyfry z szerokim blokiem (obecnie w fazie eksperymentów)

- Szyfry o bloku długości równej sektorowi.
- Nie wymagają łączenia bloków podczas szyfrowania sektora.
- Zmiana jednego bitu tekstu jawnego powoduje zmianę około połowy bitów całego zaszyfrowanego sektora.

Odzyskiwanie danych w razie uszkodzenia dysku

- Zwykle sektor daje się odczytać w całości lub wcale, więc szyfrowanie jest „przezroczyste”.
- *Bit-rotting* — zjawisko rzadkie.

Szyfrowanie dysków w Linuksie

Usługa jądra: dm-crypt

- Tworzy urządzenie blokowe `/dev/dm-`*n*.
- Link symboliczny: `/dev/mapper/`*nazwa*.
- Klucz szyfrujący może być przechowywany:
  - w sterowniku urządzenia w jądrze,
  - w breloku kluczy jądra (bezpieczniej).
- Niskopoziomowe narzędzie do konfigurowania: `dmsetup`.

Narzędzie konfiguracyjne w przestrzeni użytkownika: cryptsetup

- Konfiguruje urządzenie `dm-crypta`.
- Tryby pracy:
  - plain (klucz sesyjny podany jawnie jako argument),
  - LUKS (Linux Unified Key Setup), ver. 1 i 2,
  - kompatybilności z loop-aes,
  - kompatybilności z Veracrypt.

Szyfrowanie partycji: cryptsetup

Setup

- Wybierz nazwę *cryptname* dla urządzenia zaszyfrowanego.
- Dla wygody ta sama nazwa w trzech znaczeniach:
  - etykieta partycji,
  - nazwa urządzenia odszyfrowanego,
  - nazwa punktu montażowego.
- Nadaj nazwę *cryptname* partycji, którą chcesz zaszyfrować:
  `parted name /dev/`*zlokalizuj-właściwą-partycję cryptname*
- `cryptsetup luksFormat LABEL=`*cryptname*
- `cryptsetup open LABEL=`*cryptname cryptname*
- `mkfs.ext4 /dev/mapper/`*cryptname*
- `cryptsetup close` *cryptname*
- `mkdir -m0 /media/`*cryptname*
- Do `/etc/fstab` dodaj wiersz:
  `/dev/mapper/`*cryptname* `/media/`*cryptname* `ext4 defaults 0 0`

Szyfrowanie partycji: cryptsetup

Montowanie

```
cryptsetup open LABEL=cryptname cryptname
mount /media/cryptname
```

Odmontowanie

```
umount /media/cryptname
cryptsetup close cryptname
```

Warto oskryptować!

Idea:

```
CRYPTNAME=$1
if ! cryptsetup open LABEL=$CRYPTNAME $CRYPTNAME
then
   echo "Nie można odszyfrować urządzenia $CRYPTNAME"
   exit 1
fi
if ! fsck /dev/mapper/$CRYPTNAME || ! mount /media/$CRYPTNAME
then
   echo "Nie można zamontować urządzenia $CRYPTNAME"
   cryptsetup close $CRYPTNAME
   exit 2
fi
```

Dobra rada: jednoznaczne nazwy urządzeń

- Problem: nazwy urządzeń nadawane przez jądro (np. `/dev/sdb`) mogą się zmieniać.
- Każda partycja GPT ma swój unikatowy UUID i może mieć nadany LABEL.
- Co, jeśli chcemy zaszyfrować cały dysk, bez podziału na partycje?
  - Nagłówek LUKS ma swój unikatowy UUID.
- Co, jeśli chcemy zaszyfrować cały dysk, bez podziału na partycje i z *detached header*?
  - Linki symboliczne w katalogu `/dev/disk/by-id/`, zwykle:
    `/dev/disk/by-id/ata-`*`Model_Number-Serial_Number`*
    `/dev/disk/by-id/wwn-`*`Logical_Unit_WWN_Device_Identifier`*
    gdzie *`Model_Number`*, *`Serial_Number`*, *`Logical_Unit_WWN_Device_Identifier`* są odczytane z dysku (zob. `hdparm -I` *`dysk`*).
  - WWN: World Wide Number — uogólnienie MAC adresu na dyski Fibre Channel, ATA i SAS; ma 8 lub 16 bajtów.
- W razie problemów pomagają: `blkid(8)` i `findfs(8)`. Przydatny podczas konfiguracji: `lsblk(8)`.

Detached header

Jak to działa?

- `cryptsetup open --header=`*`plik-z-nagłówkiem-LUKS dysk nazwa`*
- Uwaga: system nie ma jak sprawdzić, że *dysk* jest zaszyfrowany zgodnie z nagłówkiem!

Typowe zastosowanie

- Dysk przenośny.
- W razie zgubienia/kradzieży oferuje pełną odporność szyfru blokowego. Bezpieczeństwo nie jest skorumpowane słabym hasłem.
- Nagłówek przechowywany na każdym komputerze, na którym chcemy go montować.
- To nie jest ograniczenie: dyski zaszyfrowane można montować tylko na zaufanych urządzeniach, nad którymi mamy kontrolę.
- Alternatywa: tryb *plain* z kluczem przechowywanym na każdym komputerze.
- Koniecznie oskryptować — łatwo o pomyłkę!

Linux Unified Key Setup (LUKS)

Nagłówek LUKS

- Klucz szyfrujący jest przechowywany w zaszyfrowanej postaci w nagłówku.
- Nagłówek ma domyślnie 2 MiB (LUKS1) i 16 MiB (LUKS2).
- Można skofigurować domyślnie do 8 haseł odblokowujących klucz.
- PBKDF2 i Argon2 (tylko LUKS2).
- *Key-stretching* wzmacnia hasła.
- *Anti-forensic stripes* (domyślnie 4000).
- Nagłówek jest umieszczony domyślnie na początku urządzenia blokowego.
- Nagłówek może być umieszczony poza urządzeniem blokowym (*detached header*).

Nagłówek na urządzeniu COW

- Główna cecha nagłówka: pozwala na wycofanie hasła bez potrzeby przeszyfrowania całego urządzenia blokowego.
- Usunięcia hasła z nagłówka: nadpisanie zgodnie z Gutmanem (1995).
- Nie ma sensu na urządzeniach COW (np. SSD), choć AFS nieco pomaga.
- Wniosek: na dyskach innych niż CMR lepiej umieszczać nagłówek poza dyskiem!

Bezpieczeństwo szyfru a bezpieczeństwo klucza

Zaszyfrowany *payload* dysku

- Zawartość dysku zaszyfrowana za pomocą AES-XTS (klucz 256+256 bitów).
- Klucz wygenerowany losowo (uwaga na jakość generatora losowego!).
- Takie szyfrowanie oferuje najwyższej klasy bezpieczeństwo (także *post-quantum* itd.).

Klucz szyfrujący zabezpieczony w nagłówku LUKS

- Podatność zależy od jakości hasła.
- Hasła są zwykle słabe.
- Warto je wzmacniać za pomocą *key stretching* i ograniczać możliwości ataku za pomocą Argon2.
- Jeśli mamy na urządzeniu nagłówek, w miarę możności należy używać losowo wygenerowanego bardzo długiego hasła.

Jak trudne powinny być hasła?

Entropia źródła haseł

- Dane jest źródło generujące hasła.
- Zawartość informacyjna hasła h wygenerowanego przez źródło: $I(h) = -\log_2(\Pr(h))$.
- Entropia źródła haseł — wartość oczekiwana zawartości informacyjnej generowanych haseł: $H = E(I(h))$.
- Jeśli hasła są wybierane losowo, niezależnie, z jednakowym prawdopodobieństwem ze zbioru X, to $H = \log_2 |X|$.
- Jeśli hasłem jest ciąg n losowych bitów, to entropia tego źródła haseł wynosi n.
- Aby dokonać ataku *brute force* na hasło wygenerowane przez źródło o entropii n, trzeba wykonać przeciętnie 2^{n-1} prób.

Przykłady

- Słownictwo czynne przeciętnego Polaka: kilkanaście tysięcy słów.
- Słowa wybierane „losowo" z takiego słownika mają około 14 bitów entropii.
- Trójki takich słów: 42 bity.
- Ciągi drukowalnych znaków ASCII (94 różne znaki): 6.55 bita/znak.

Kiedy hasła powinny być „trudne”?

Atak on-line

- Atakujący próbuje logować się do systemu, który może ograniczać tempo i liczbę wprowadzanych haseł.
- Przestrzeń haseł może być mała (np. 4-cyfrowe PIN-y do karty płatniczej).

Atak off-line

- Atakujący ma dostęp do zaszyfrowanych danych i może uruchomić własną weryfikację haseł.
- Tempo sprawdzanych haseł może być wielokrotnie większe niż u legalnego użytkownika.

Wniosek

- Jeśli atak off-line nie jest możliwy, hasła mogą być bardzo proste.

Przykład: hasło do konta w GMail

Zagrożenia

- Trojan (keylogger) na urządzeniu użytkownika — przechwytuje samo hasło (trudność nieistotna).
- Atakujący próbuje różne hasła w portalu GMail — atak off-line niemożliwy.
- Przesyłanie hasła od użytkownika do serwera GMail — szyfrowane za pomocą TLS — możliwy atak off-line, ale na TLS, a nie hasło użytkownika.
- Wyciek bazy danych haseł GMaila — możliwy atak off-line, ale zupełnie nieprawdopodobny.

Wnioski

- Atak off-line na hasło do GMaila jest praktycznie niemożliwy.
- Zatem hasło do GMaila może być proste!

Uwagi

- Co innego, gdyby hasło do GMaila było przesyłane np. za pomocą CRAM. OAUTH2?
- Co innego, jeśli chodzi o pocztę w gorszym serwisie!

Hasła przechowywane na dysku

Przykłady

- Hasło do klucza prywatnego SSH lub GPG umieszczonego w pliku na dysku.
- Hasło odblokowujące klucz szyfrujący w nagłówku LUKS.

Zagrożenia

- Zawartość pliku lub dysku zostają skradzione.
- Atakujący może atakować hasła off-line.

Wnioski

- Hasła do nagłówka LUKS powinny być bardzo trudne.
- Najlepiej, jeśli byłyby tak trudne, jak atakowanie samego szyfru blokowego.
- W miarę możliwości należy w ogóle unikać takich haseł.

Na czym polega atak *brute force*

Próba pojedynczego hasła

- Należy wykonać *key setup* dla tego hasła do momentu sprawdzenia jego poprawności.
- W LUKS1/2:
  - KDF (PBKDF2/Argon2),
  - odszyfrowanie AF Stripes i utworzenie klucza sesyjnego,
  - KDF na kluczu sesyjnym,
  - porównanie z haszem zapisanym w nagłówku.
- *Key stretching*: liczba iteracji haszy w KDF — rzędu miliona — dodaje około 20 bitów entropii.
- Koszt pojedynczego *key setup* jest równoważny pewnej liczbie wywołań SHA-256.

Koszt *brute-forcingu*

- Koszt: $2^{H-1} \cdot S$, gdzie H-entropia źródła haseł, S — koszt pojedynczej próby.
- Ile kosztuje obliczenie SHA-256?

Cena obliczenia haszy Bitcoina ($2 \times$ SHA-256)

Dane według https://bitinfocharts.com/bitcoin/ (3 maja 2022 16:00 CEST)

- Hash rate: $231.371\,\mathrm{EH/s} = 231.371 \cdot 10^{18}\,\mathrm{H/s}$
- Blocks average per hour: 6
- Reward per block: $6.25 + 0.07471\,\mathrm{BTC}$
- Bitcoin price: $38525.19\,\mathrm{USD}$

Liczba haszy na BTC

$$\frac{231.371 \cdot 10^{18}\,\mathrm{H/s} \times 600\,\mathrm{s}}{6.25 + 0.07471\,\mathrm{BTC}} = 0.569737\,\mathrm{EH/BTC}$$

Cena wyliczenia haszy w USD (*mining profitability*)

$$\frac{(6.25 + 0.07471\,\mathrm{BTC}) \cdot 38525.19\,\mathrm{USD/BTC}}{231.371 \cdot 10^{18}\,\mathrm{H/s} \times 600\,\mathrm{s}} = 1.75519\,\mathrm{USD/EH}$$

Cena 2^n haszy (albo: co to jest skala logarytmiczna)

n	koszt
56	¢ 12.65
60	$ 2.02
64	$ 32.38
70	$ 2072
77	$ 243.66 tys. — koszt jednego bloku ($6.25 + 0.07471$ BTC)
80	$ 2.12 mln.
90	$ 2.17 mld.
100	$ 2.22 bln.
103	PKB USA w 2020 ($20.94 bln.)
105	PKB Świata w 2020 ($84.71 bln.)
128	$ $597 \cdot 10^{18} = 7.05$ mln. PKB Świata

Wnioski

- Uwaga: łamanie haseł LUKS-a, to nie tylko liczenie SHA-256. Koszt może być, powiedzmy, 1000 razy większy (przesuwa entropię o 10 bitów).
- Szacunkowy koszt łamania hasła złożonego z 3 „losowych" słów: 42 bity entropii plus 10 bitów narzutu plus 20 bitów *key stretching*, razem: 72 bity, tj. około $ 8000.
- Dobre hasła powinny mieć nie mniej niż 80 bitów entropii (co najmniej 13 losowych drukowalnych znaków ASCII).
- *Key stretching* w LUKS-ie zwiększa koszt łamania około milion razy.
- Używamy żałośnie słabych haseł...
- Tam, gdzie można, należy unikać uzależniania bezpieczeństwa kryptografii od haseł!
- Rozwiązanie: *detached header*!

Rozruch komputera

1. *Hardware bootloader* zapisany w pamięci *flash* (*firmware*) płyty głównej (BIOS/UEFI):
   - POST
   - inicjalizacja magistral i urządzeń (nie, gdy *fast boot*)
   - uruchomienie programu zapisanego na ustalonym urządzeniu blokowym (*boot device*)

Rozruch komputera

1. *Hardware bootloader* zapisany w pamięci *flash* (*firmware*) płyty głównej (BIOS/UEFI):
   - POST
   - inicjalizacja magistral i urządzeń (nie, gdy *fast boot*)
   - uruchomienie programu zapisanego na ustalonym urządzeniu blokowym (*boot device*)

2. Jądro systemu operacyjnego z RAM-dyskiem (`vmlinuz` + `initrd`)
   - inicjalizuje przestrzeń użytkownika
   - pyta o hasła, odszyfrowuje i montuje dyski (również sieciowe)

Rozruch komputera

1. *Hardware bootloader* zapisany w pamięci *flash* (*firmware*) płyty głównej (BIOS/UEFI):
   - POST
   - inicjalizacja magistral i urządzeń (nie, gdy *fast boot*)
   - uruchomienie programu zapisanego na ustalonym urządzeniu blokowym (*boot device*)

1a. *Software bootloader* (Syslinux, LILO, Grub)

2. Jądro systemu operacyjnego z RAM-dyskiem (`vmlinuz` + `initrd`)
   - inicjalizuje przestrzeń użytkownika
   - pyta o hasła, odszyfrowuje i montuje dyski (również sieciowe)

Rozruch komputera

1. *Hardware bootloader* zapisany w pamięci *flash* (*firmware*) płyty głównej (BIOS/UEFI):
   - POST
   - inicjalizacja magistral i urządzeń (nie, gdy *fast boot*)
   - uruchomienie programu zapisanego na ustalonym urządzeniu blokowym (*boot device*)

1a. *Software bootloader* (Syslinux, LILO, Grub)

2. Jądro systemu operacyjnego z RAM-dyskiem (`vmlinuz` + `initrd`)
   - inicjalizuje przestrzeń użytkownika
   - pyta o hasła, odszyfrowuje i montuje dyski (również sieciowe)

Jądro z systemem plików `initrd` i bootloadery są na *nieszyfrowanej* partycji — nie można zapewnić ich integralności!

Evil Maid Attack

Bootkit

Bootkit

Złośliwe oprogramowanie instalowane na komputerze, które modyfikuje proces rozruchu i umożliwia np. podsłuchanie hasła do dysku.

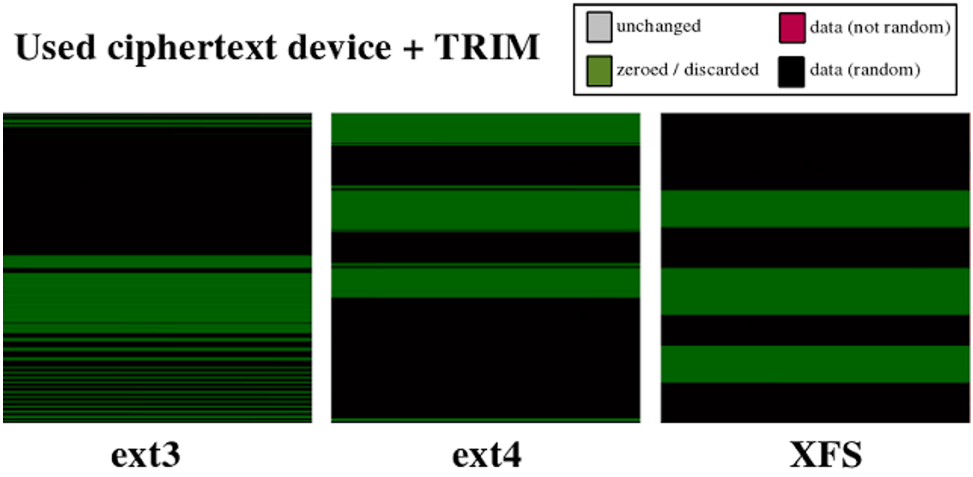
- Klasyczny Evil Maid Attack.
- Joanna Rutkowska, *Evil Maid goes after TrueCrypt!*, October 16, 2009, blog post.
- Podatne: TrueCrypt, PGP Whole Disk Encryption, dm-crypt, BitLocker i inne.

Sposoby *częściowej* ochrony:

- Partycja rozruchowa na pendrivie.
- Hasła do BIOS/UEFI.
- Zabezpieczenia fizyczne obudowy (*anti-tampering*).
- Secure boot, szczególnie w połączeniu z TPM.

Dalsze problemy: zamazywanie dysków SSD a trimming

Milan Broz's blog: TRIM & dm-crypt ... problems?



Hardware-based full disk encryption

Self-Encrypting Drives

- Trusted Computing Group, *Ten Reasons to Buy Self-Encrypting Drives*, September 2010.
- Gunnar Alendal, Christian Kison, modg, *got HW crypto? On the (in)security of a Self-Encrypting Drive series*, Sept. 28th, 2015.

Czy szyfrowanie spowalnia? Mój laptop i stacja robocza:

PASSMARK® SOFTWARE

CPU Benchmarks

Over 600,000 CPUs Benchmarked

	Intel Core i3-3217U @ 1.80GHz	Intel Core2 Duo E8400 @ 3.00GHz
Socket Type	BGA1023	LGA775
CPU Class	Laptop	Desktop
Clockspeed	1.8 GHz	3.0 GHz
Turbo Speed	Not Supported	Not Supported
# of Physical Cores	2 (2 logical cores per physical)	2
Max TDP	17W	65W
First Seen on Chart	Q2 2012	Q4 2008
# of Samples	720	2227
Single Thread Rating	895	1257
CPU Mark	**2296**	**2170**

https://www.cpubenchmark.net/

Benchmarki dm-crypt: Core 2 Duo E8400 (Wolfdale 2008)

```
linux# cryptsetup benchmark
# Tests are approximate using memory only (no storage IO).
PBKDF2-sha1       819200 iterations per second
PBKDF2-sha256     520126 iterations per second
PBKDF2-sha512     368179 iterations per second
PBKDF2-ripemd160  492751 iterations per second
PBKDF2-whirlpool  160627 iterations per second
#  Algorithm | Key |  Encryption |  Decryption
     aes-cbc   128b   158.0 MiB/s   182.6 MiB/s
 serpent-cbc   128b    59.8 MiB/s   229.6 MiB/s
 twofish-cbc   128b   154.9 MiB/s   203.4 MiB/s
     aes-cbc   256b   123.6 MiB/s   137.4 MiB/s
 serpent-cbc   256b    59.8 MiB/s   229.6 MiB/s
 twofish-cbc   256b   154.8 MiB/s   203.3 MiB/s
     aes-xts   256b   183.0 MiB/s   179.8 MiB/s
 serpent-xts   256b   208.6 MiB/s   213.7 MiB/s
 twofish-xts   256b   189.6 MiB/s   191.5 MiB/s
     aes-xts   512b   138.1 MiB/s   136.0 MiB/s
 serpent-xts   512b   208.5 MiB/s   213.9 MiB/s
 twofish-xts   512b   189.5 MiB/s   191.3 MiB/s
```

Benchmarki dm-crypt: i3-3217U (Ivy Bridge 2012)

```
linux# cryptsetup benchmark
# Tests are approximate using memory only (no storage IO).
PBKDF2-sha1       587766 iterations per second
PBKDF2-sha256     387786 iterations per second
PBKDF2-sha512     261099 iterations per second
PBKDF2-ripemd160  350459 iterations per second
PBKDF2-whirlpool  120470 iterations per second
#  Algorithm | Key |  Encryption |  Decryption
     aes-cbc   128b   110.0 MiB/s   127.1 MiB/s
 serpent-cbc   128b    44.5 MiB/s   151.9 MiB/s
 twofish-cbc   128b    94.2 MiB/s   181.1 MiB/s
     aes-cbc   256b    86.1 MiB/s    94.9 MiB/s
 serpent-cbc   256b    46.3 MiB/s   153.2 MiB/s
 twofish-cbc   256b    95.6 MiB/s   181.2 MiB/s
     aes-xts   256b   127.8 MiB/s   124.8 MiB/s
 serpent-xts   256b   152.7 MiB/s   149.8 MiB/s
 twofish-xts   256b   176.5 MiB/s   179.2 MiB/s
     aes-xts   512b    96.2 MiB/s    93.4 MiB/s
 serpent-xts   512b   158.0 MiB/s   151.1 MiB/s
 twofish-xts   512b   178.0 MiB/s   179.1 MiB/s
```

Benchmarki dm-crypt: Celeron 3050 (Braswell 2015)

```
linux# cryptsetup benchmark
# Tests are approximate using memory only (no storage IO).
PBKDF2-sha1       330989 iterations per second for 256-bit key
PBKDF2-sha256     431157 iterations per second for 256-bit key
PBKDF2-sha512     254015 iterations per second for 256-bit key
PBKDF2-ripemd160  264258 iterations per second for 256-bit key
PBKDF2-whirlpool  203527 iterations per second for 256-bit key
#  Algorithm | Key |  Encryption |  Decryption
     aes-cbc   128b   273.8 MiB/s   402.5 MiB/s
 serpent-cbc   128b    31.8 MiB/s    91.2 MiB/s
 twofish-cbc   128b    72.7 MiB/s    83.4 MiB/s
     aes-cbc   256b   219.0 MiB/s   313.6 MiB/s
 serpent-cbc   256b    36.7 MiB/s    91.3 MiB/s
 twofish-cbc   256b    79.9 MiB/s    83.8 MiB/s
     aes-xts   256b   355.9 MiB/s   356.9 MiB/s
 serpent-xts   256b    86.5 MiB/s    86.6 MiB/s
 twofish-xts   256b    78.2 MiB/s    78.1 MiB/s
     aes-xts   512b   283.8 MiB/s   285.5 MiB/s
 serpent-xts   512b    86.8 MiB/s    86.7 MiB/s
 twofish-xts   512b    78.1 MiB/s    78.2 MiB/s
```

Intel® AES New Instructions

`AESENC` This instruction performs a single round of encryption. The instruction combines the four steps of the AES algorithm — ShiftRows, SubBytes, MixColumns & AddRoundKey into a single instruction.

`AESENCLAST` Instruction for the last round of encryption. Combines the ShiftRows, SubBytes, & AddRoundKey steps into one instruction.

`AESDEC` Instruction for a single round of decryption. This combines the four steps of AES — InvShiftRows, InvSubBytes, InvMixColumns, AddRoundKey into a single instruction.

`AESDECLAST` Performs last round of decryption. It combines InvShiftRows, InvSubBytes, AddRoundKey into one instruction.

`AESKEYGENASSIST` is used for generating the round keys used for encryption.

`AESIMC` is used for converting the encryption round keys to a form usable for decryption using the Equivalent Inverse Cipher.

Phoronix

Michael Larabel http://www.phoronix.com

The Performance Impact of Linux Disk Encryption

Program	Miara	Bez	Z	Strata
Enemy Territory v2.60 800×600	[Avg. FPS]	100.0	99.4	0.6%
Doom 3 v1.3.1. 1280x1024 HQ	[Avg. FPS]	56.2	54.8	2.5%
ET: Quake Wars v1.4 1280×1024 HQ	[Avg. FPS]	31.2	28.3	10.2%
LAME Encoding v3.97 81.3 MB WAV to MP3	[s]	53.71	55.87	4.0%
Gzip Compression 745MB	[s]	53.19	57.48	8.0%
Flash Drive to HDD Transfer 1.3GB in 364 files	[s]	66.952	71.933	7.4%

- AMD Athlon 64 X2 4200+ AM2 processor
- 2GB of A-DATA DDR2-800 memory
- 160GB Western Digital SATA hard drive
- Abit NF-M2 nView motherboard
- NVIDIA GeForce 6600GT 128MB graphics card with the 169.12 driver.

Published on 16 March 2008 by Phoronix