Szyfrowanie plików

Nie ulega wątpliwości, że współczesny system operacyjny powinien oferować użytkownikowi jak największe bezpieczeństwo danych, które są za jego pomocą przetwarzane. Także po zakończeniu pracy i wyłączeniu komputera dane te powinny być dalej chronione. Celem tego zadania jest rozbudowanie serwera *mfs*, obsługującego system plików MINIX (MINIX File System), o prototypowe szyfrowanie treści plików.

W poniższym opisie słowo *plik* jest używane w znaczeniu zwykłego pliku, nie katalogu.

(A) Szyfrowanie treści plików

Algorytm szyfrowania jest bardzo prosty. Kluczem szyfrującym jest 1-bajtowa liczba. Zaszyfrowanie ciągu bajtów polega na dodaniu (modulo 256) do każdego z nich wartości klucza. Odszyfrowanie ciągu bajtów polega na odjęciu (modulo 256) od każdego z nich wartości klucza. Oczywiście, zdecydowanie nie jest to bezpieczny algorytm szyfrowania, ale będzie wystarczający na potrzeby naszej prototypowej implementacji.

Szyfrowaniu podlega tylko treść plików: ich zawartość powinna być szyfrowana podczas operacji zapisu i odszyfrowywana podczas operacji odczytu. Szyfrowaniu nie podlegają katalogi (m.in. struktura katalogów na dysku) ani metadane pliku (m.in. nazwa, rozmiar, uprawnienia).

Użytkownik klucz szyfrujący podaje poprzez zapisanie go do pliku o nazwie `KEY`, który jest umieszczony w głównym katalogu partycji. Wartość ta nie jest jednak faktycznie zapisywana w treści pliku, a jedynie odpowiednio interpretowana przez zmodyfikowany serwer *mfs*. Zmianie nie powinny ulegać także metadane pliku `KEY` (m.in. rozmiar oraz czas modyfikacji). Podany klucz jest wykorzystywany do szyfrowania treści plików danej partycji do czasu odmontowania tej partycji lub do czasu podania przez użytkownika innego klucza. Zapis do pliku `KEY` większej ilości danych niż 1 bajt powinien zakończyć się błędem `EINVAL`. Natomiast próba odczytu treści pliku `KEY` powinna zakończyć się błędem `EPERM`.

Serwer *mfs* nie sprawdza poprawności klucza: każda podana 1-bajtowa wartość jest traktowana jako poprawny klucz. Zmieniając (ręcznie) klucze, użytkownik może szyfrować różne pliki różnymi kluczami. Gdy klucz ma wartość 0, to dane przed i po zaszyfrowaniu wyglądają tak samo.

Przykład działania:

```
# cd <główny katalog partycji>
# touch KEY
# touch plik
# ls
KEY  plik
# printf '\x2A' > ./KEY
# echo 'ALA MA KOTA' > ./plik
# cat ./plik
ALA MA KOTA
# cat ./KEY
cat: ./KEY: Operation not permitted
# printf '\x0' > ./KEY
# cat ./plik
kvkJwkJuy~k4#
```

(B) Blokowanie partycji

Aby zapobiec przypadkowemu odczytowi lub zapisowi pliku zanim użytkownik poda klucz szyfrujący, bezpośrednio po zamontowaniu partycja jest *zablokowana*: próba odczytania lub zapisania treści pliku powinna zakończyć się błędem `EPERM` (za wyjątkiem pseudo-zapisu do pliku `KEY`). Partycja jest samoczynnie odblokowywana, gdy użytkownik poda klucz.

Blokowaniu nie podlegają wszystkie inne operacje na plikach (m.in. tworzenie, usuwanie, zmiana nazwy) oraz katalogach.

Przykład działania:

```
# cd <główny katalog właśnie zamontowanej partycji>
# ls
KEY  plik
# echo 'ala ma kota' | tee ./plik
tee: ./plik: Operation not permitted
ala ma kota
# cat ./plik
cat: ./plik: Operation not permitted
# printf '\x2A' > ./KEY
# echo 'ala ma kota' | tee ./plik
ala ma kota
# cat ./plik
ala ma kota
#
```

(C) Dezaktywacja szyfrowania

Jeśli użytkownik nie chce szyfrować danej partycji, w jej głównym katalogu może umieścić plik lub katalog o nazwie `NOT_ENCRYPTED`. Gdy taki plik lub katalog jest obecny, treść plików nie jest szyfrowana podczas zapisów i odszyfrowywana podczas odczytów, a partycja nie jest blokowana. Nie jest możliwa wtedy jednak zmiana klucza: próba zapisu do pliku `KEY`, który jest umieszczony w głównym katalogu partycji, powinna zakończyć się błędem `EPERM`.

Plik lub katalog `NOT_ENCRYPTED` możne być tworzony (dezaktywacja szyfrowania) i usuwany (aktywacja szyfrowania) przez użytkownika w dowolnym momencie. Po jego usunięciu partycja jest szyfrowana dalej takim kluczem, jaki był ustawiony wcześniej lub zablokowana, jeśli użytkownik nie ustawił wcześniej żadnego klucza.

Przykład działania:

```
# cd <główny katalog partycji>
# ls
KEY  plik
# touch ./NOT_ENCRYPTED
# echo 'ala ma kota' > ./plik
# cat ./plik
ala ma kota
# printf '\x2A' | tee ./KEY
tee: ./KEY: Operation not permitted
*# rm ./NOT_ENCRYPTED
# printf '\x2A' | tee ./KEY
*# cat ./plik
7B7C7AEJ7#
```

Wymagania i niewymagania

1. Wszystkie pozostałe operacje na plikach, inne niż opisane powyżej, powinny działać bez zmian.
2. Modyfikacje serwera nie mogą powodować błędów w systemie plików: ma być on zawsze poprawny i spójny.
3. Dyski przygotowane i używane przez niezmodyfikowany serwer *mfs* powinny być poprawnymi dyskami także dla zmodyfikowanego serwera.
4. Modyfikacje mogą dotyczyć tylko serwera *mfs* (czyli mogą dotyczyć tylko plików w katalogu `/usr/src/minix/fs/mfs`).
5. Podczas działania zmodyfikowany serwer nie może wypisywać żadnych dodatkowych informacji na konsolę ani do rejestru zdarzeń (ang. *log*).

6. Specjalne znaczenie mają tylko plik o nazwie `KEY` i plik lub katalog o nazwie `NOT_ENCRYPTED` umieszczone w głównym katalogu partycji. Natomiast szyfrowaniu podlegają wszystkie pliki umieszczone na danej partycji (także w podkatalogach).
7. Można założyć, że w testowanych przypadkach użytkownik będzie miał wystarczające uprawnienia do wykonania wszystkich operacji.
8. Można założyć, że w testowanych przypadkach w systemie plików będą tylko zwykłe pliki (nie łącza, nie pseudo-urządzenia itp.) i katalogi.
9. Rozwiązanie nie musi być optymalne pod względem prędkości działania. Akceptowane będą rozwiązania, które działają bez zauważalnej dla użytkownika zwłoki.
10. Szyfrowaniu i deszyfrowaniu nie musi podlegać dostęp do treści pliku przy mapowaniu go do przestrzeni adresowej procesu (`mmap()`): proces taki może widzieć bezpośrednio zaszyfrowane dane. Z takiego mapowania korzysta m.in. polecenie `cp` podczas kopiowania małych plików, zatem nie jest wymagane, aby ich kopiowanie poleceniem `cp` z lub do zaszyfrowanej partycji działało poprawnie.

Wskazówki

1. Aby skompilować i zainstalować zmodyfikowany serwer *mfs*, należy wykonać `make; make install` w katalogu `/usr/src/minix/fs/mfs`. Takimi poleceniami będzie budowane i instalowane oddane rozwiązanie.
2. Każde zamontowane położenie (ich listę wyświetli polecenie `mount`) obsługiwane jest przez nową instancję serwera *mfs*. Położenia zamontowane przed instalacją nowego serwera będą obsługiwane nadal przez jego starą wersję, więc, aby przetestować na nich zmodyfikowany serwer, należy je odmontować i zamontować ponownie lub zrestartować system.
3. Aby zmodyfikowany serwer obsługiwał też korzeń systemu plików (`/`), należy wykonać dodatkowe kroki, ale radzimy nie testować na nim (i nie wymagamy tego) zmodyfikowanego serwera *mfs*.
4. Do MINIX-a uruchomionego pod *QEMU* można dołączać dodatkowe dyski twarde (i na nich testować swoje modyfikacje). Aby z tego skorzystać, należy:

   A. Na komputerze-gospodarzu stworzyć plik będący nowym dyskiem, np.: `qemu-img create -f raw extra.img 1M`.

   B. Podłączyć ten dysk do maszyny wirtualnej, dodając do parametrów, z jakimi uruchamiane jest *QEMU*, parametry `-drive file=extra.img,format=raw,index=1,media=disk`, gdzie parametr `index` określa numer kolejny dysku (0 to główny dysk – obraz naszej maszyny).

   C. Za pierwszym razem stworzyć na nowym dysku system plików mfs: `/dev/c0d<numer kolejny dodanego dysku>`, np. `/sbin/mkfs.mfs /dev/c0d1`.

   D. Stworzyć pusty katalog (np. `mkdir /root/nowy`) i zamontować do niego podłączony dysk: `mount /dev/c0d1 /root/nowy`.

   E. Wszystkie operacje wewnątrz tego katalogu będą realizowane na zamontowanym w tym położeniu dysku.

   F. Aby odmontować dysk, należy użyć polecenia `umount /root/nowy`.
5. Tablica z funkcjami obsługiwanymi przez serwer *mfs* znajduje się w pliku `table.c`.
6. W MINIX-ie małe ilości informacji przekazuje się między procesami poprzez wiadomości (zob. Laboratorium 7), natomiast większe porcje danych poprzez niskopoziomową pamięć dzieloną – tzw. granty (zob. punkt 4.2 w Laboratorium 7).
7. Dane między procesami powinny być kopiowane przez jądro w sposób bezpieczny (zob. Laboratorium 11).
8. I-węzeł głównego katalogu partycji ma stały numer (zob. `const.h`):

```
#define ROOT_INODE   ((ino_t) 1)    /* inode number for root directory */
```

Rozwiązanie

Poniżej przyjmujemy, że *ab123456* oznacza identyfikator studenta rozwiązującego zadanie. Należy przygotować łatkę (ang. *patch*) ze zmianami. Plik o nazwie `ab123456.patch` uzyskujemy za pomocą polecenia `diff -rupN`, tak jak w zadaniu 3. Łatka będzie aplikowana przez umieszczenie jej w katalogu `/` nowej kopii MINIX-a i wykonanie polecenia `patch -p1 < ab123456.patch`. Należy zadbać, aby łatka zawierała tylko niezbędne różnice. Na Moodle należy umieścić tylko łatkę ze zmianami.

Ocenianie

Oceniana będą zarówno poprawność, jak i styl rozwiązania. Podstawą do oceny rozwiązania będą testy automatyczne sprawdzające poprawność implementacji oraz przejrzenie kodu przez sprawdzającego. Za poprawną i w dobrym stylu implementację funkcjonalności opisanych w punkcie (A) rozwiązanie otrzyma 3 pkt. Za implementację funkcjonalności opisanych w punktach (B) i (C) po 1 pkt. Rozwiązanie, w którym łatka nie nakłada się poprawnie, które nie kompiluje się lub powoduje *kernel panic* podczas uruchamiania otrzyma 0 pkt.

Pytania

Pytania do zadania należy zadawać w przeznaczonym do tego wątku Moodle.