Praktyka

Kryptografia dla poczty i danych

Lars Packschies

stopień trudności

Termin kryptografia pochodzi od greckich słów: kryptós, ukryte, oraz gráphein, pismo. W ogólności, wyróżniamy dwa rodzaje szyfrów kryptograficznych: symetryczne i asymetryczne. Określenia te związane są ze strukturą klucza. Aby zaszyfrować dane bądź wiadomość potrzebne są informacje o tym, jak szyfrować bądź deszyfrować dane (szyfr), a także klucz – tajny parametr szyfru.

Czy umieściłbyś poufne informacje na kartce pocztowej i wysłał je w ten sposób do znajomych, współpracowników bądź partnerów biznesowych?
Chyba, nie. Dlaczego zatem mielibyśmy umieszczać poufne informacje w e-mailu i wysyłać je przez cały świat? Kryptografia nie tylko bardzo zwiększa bezpieczeństwo komunikacji w Internecie oferując możliwość szyfrowania i/lub podpisywania wiadomości, ale także stanowi gwarancję naszej prywatności. Przykładowo, być może jesteś świadom faktu, że Unia Europejska usankcjonowała przechowywanie przez dostawców Internetu oraz operatorów sieci komórkowych informacji o połączeniach przez przynajmniej 6 miesięcy. W połączeniu z informacjami o kartach kredytowych i premiowych, a także wszystkimi innymi dostępnymi tu i ówdzie informacjami, pozwala to na wygenerowanie kompletnego profilu osobistego nie tylko z podstawowych danych, ale także z algorytmów akwizycji danych. Być może już teraz zebrały one mnóstwo informacji na temat ciebie i twoich nawyków, teraz jednak możesz zacząć coś z tym robić.

Szyfry symetryczne i asymetryczne

Symetryczne klucze kryptograficzne charakteryzują się tym, że klucze: szyfrujący i deszyfrujący są identyczne. Innymi słowy, nadawca i odbiorca wymienianej między nimi wiadomości muszą posiadać ten sam klucz – i muszą wymienić ten klucz przed rozpoczęciem transmisji wiadomości. Była to od zawsze główna wada metod symetrycznych: *problem wymiany klucza.*

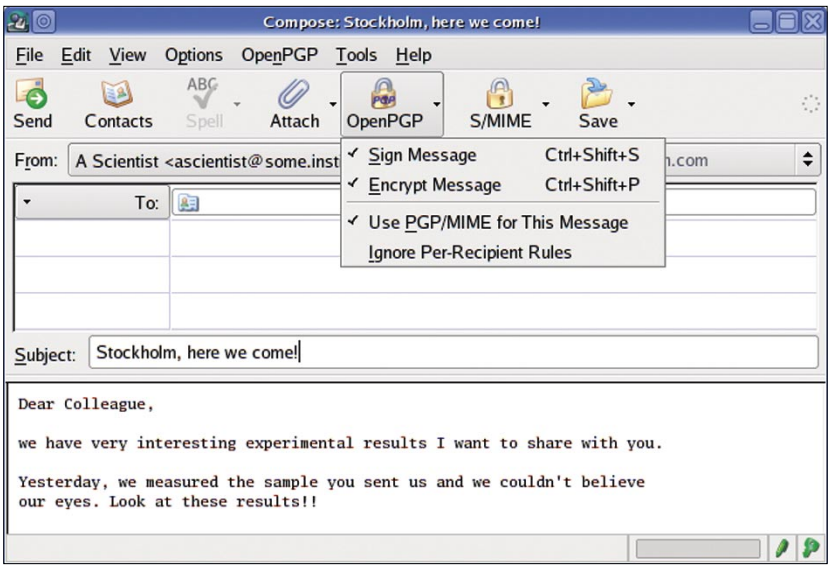
Jeden z pierwszych znanych szyfrów nosi nazwę *szyfru Cezara*. Juliusz Cezar szyfro-

Z artykułu dowiesz się...

- Jak zainstalować i urzyć klucze GnuPG
- Jak szyfrować dane na poziomie systemu plików

Powinieneś wiedzieć...

- Czym są podstawy kryptografii symetrycznej i asymetrycznej
- Jakie są podstawy algorytmów



Rysunek 1. *Enigmail dodaje przycisk OpenPGP, pozwalający na podpisywanie i/lub szyfrowanie poczty*

wał wiadomości zastępując każdą literę w oryginalnej wiadomości literą przesuniętą o trzy miejsca w prawo w alfabecie: A staje się D, B staje się E, a Z staje się C. Algorytmem jest tu zastąpienie każdej litery jawnego tekstu przez literę z przesuniętego alfabetu, kluczem zaś jest 3: alfabet przesunięty został o 3.

Oczywiście metody zastępowania i transponowania liter celem stworzenia bardziej zaawansowanych szyfrów ulegały przez lata ewolucji, niektórych przypadkach wymagały one zastosowania urządzeń mechanicznych. Jednym z ważnych przykładów jest tu ENIGMA, wykorzystywana przez niemieckie wojsko podczas drugiej wojny światowej (bardzo dobry artykuł na temat tego urządzenia oraz jego kryptoanalizy znaleźć można w Wikipedii). Wykorzystywano ponad 200 000 takich maszyn, zaś każdy operator musiał otrzymywać co miesiąc listę kluczy, tak zwaną książkę kodową. Przy okazji: zakończoną powodzeniem kryptoanalizę ENIGMY (można powiedzieć: jej *złamanie*) przeprowadziła grupa naukowców skupionych wokół polskiego matematyka Mariana Rajewskiego oraz Alana Turinga, w Bletchley Park, w Milton Keynes, w Anglii, już w połowie lat 30 ubiegłego wieku. Ogół poinformowano o tym w latach 1970 (nazwano to *ultra sekretem*).

Dzięki wykorzystaniu współczesnym komputerów istnieje obecnie całkiem spora liczba symetrycznych szyfrów, które uważa się za *bezpieczne* – na przykład AES (Advanced Encryption Standard, inaczej Rijndael, stworzony przez Joana Daemena i Vincenta Rijmena), 3DES (potrójny DES, Data Encryption Standard, oparty na pracach Horsta Feistela) czy też IDEA (International Data Encryption Algorithm), wymieniając zaledwie kilka. Wszystkie te nowoczesne symetryczne szyfry zaprojektowano z grubsza później niż w latach 50 ubiegłego wieku. Szyfry stworzone wcześniej określa się szerzej jako *klasyczne*.

Niemniej dopiero w latach siedemdziesiątych kryptografowie rozwiązali problem wymiany klucza (prace Whitfielda Diffiego, Martina Hellmana i Ralpha Merkle'a) oraz powstała oparta na tym pomyśle koncepcja asymetrycznych kluczy, opracowana przez Rona Rivesta, Adiego Shamira i Leonarda Adlemana w 1977 roku.

Metody czy też algorytmy kryptografii asymetrycznej wykorzystują inne klucze do szyfrowania, a inne do deszyfrowania wiadomości. Oba takie klucze nazywane są wspólnie *parą kluczy* (często określaną po prostu jako *klucz* bądź *klucz asymetryczny*). Po wygenerowaniu pary jeden z kluczy trzymany jest w sekrecie, nazywamy go *kluczem prywatnym*, drugi zaś udostępnia się publicznie i jest nazywany *kluczem publicznym*. Jeden z kluczy służy do szyfrowania wiadomości, a dzięki matematycznym podstawom stosowanego rozwiązania do zrekonstruowania oryginalnej wiadomości można wykorzystać tylko drugi klucz. Praktycznie niemożliwe jest wyliczenie klucza prywatnego z klucza publicznego (bądź na odwrót). Ponadto równie niemożliwe są próby deszyfrowania wiadomości przez zastosowanie wszystkich możliwych kluczy (próby tego rodzaju określa się mianem *ataków siłowych*) – wedle współczesnej wiedzy zajęłoby to kilka miliardów lat.

Klucze prywatne i publiczne

Koncepcja kluczy prywatnych i publicznych ogólnie rzecz biorąc pozwala na wykorzystanie ich na dwa sposoby: (1) szyfrowanie/deszyfrowanie oraz (2) generacja i weryfikacja elektronicznych podpisów.

Szyfrowanie/deszyfrowanie

Wyobraźmy sobie dwie osoby, Alice i Boba. Alice generuje parę kluczy (robi to tylko raz, potem klucze można wykorzystywać wielokrotnie) i upublicznia swój klucz publiczny, dzięki czemu Bob może pobrać ten klucz. Teraz Bob może wykorzystać ów klucz do zaszyfrowania wiadomości przeznaczonej dla Alice, ale tylko prywatny klucz Alice jest w stanie rozszyfrować wiadomość od Boba. Tylko właściciel prywatnego klucza, Alice będzie mógł ją przeczytać. Każda osoba mająca dostęp do publicznego klucza Alice może wysłać jej wiadomość, którą tylko ona może odczytać. Gdyby Alice chciała wysłać tajną wiadomość do Boba, mogłaby skorzystać z publicznego klucza wygenerowanego przez tego ostatniego.

Podpis

Drugi sposób wykorzystuje te same dwa klucze Alice, ale w odwrotnej kolejności. Wyobraźmy sobie, że Alice pisze wiadomość i szyfruje ją swoim *kluczem prywatnym*. Teraz każdy, kto posiada dostęp do odpowiedniego klucza publicznego może odczytać tę wiadomość po jej odszyfrowaniu. W przypadku takim odbiorca może być pewien, że wiadomość zaszyfrowano prywatnym kluczem Alice, a co za tym idzie to Alice musiała ją napisać – z definicji Alice jest jedyną osobą posiadającą dostęp do tego klucza prywatnego. Nazywamy to podpisem elektronicznym.

Ogólnie rzecz biorąc, istnieją dwie główne metody asymetryczne, z którymi będziemy mieli do czynienia i które uważane są za *bezpieczne*: RSA (Rivest, Shamir, Adleman; opatentowany) i ElGamal (autorstwa Tahera ElGamala). Istnieje także Digital Signature Algorithm (DSA).

PGP, OpenPGP, S/MIME

Zbierają wszystkie powyższe informacje razem: RSA, ElGamal i DSA to algorytmy bądź szyfry asymetryczne. AES, 3DES bądź IDEA (IDEA również został opatentowany) są szyframi symetrycznymi. Można ich używać po prostu by szyfrować, deszyfrować bądź nawet elektronicznie podpisywać dane – jednak aby naprawdę możliwe było wykorzystywanie tych algorytmów w rzeczywistych zastosowaniach niezbędna jest znaczna wiedza w innych dziedzinach, na przykład: jak przetwarzać dane, jakich algorytmów używać do generacji par kluczy, co zrobić gdy wiadomość ma być zaszyfrowana bądź odszyfrowana i tak dalej.

Aby skomplikować zagadnienie jeszcze bardziej, w nowoczesnych aplikacjach do szyfrowania danych wykorzystuje się nie tylko szyfry asymetryczne. Zaszyfrowanie dużych ilości danych za pomocą szyfru asymetrycznego zajmuje mnóstwo czasu, znacznie dłużej niż ma to miejsce w przypadku szyfrów symetrycznych.

Dlatego ze względów praktycznych dla każdej wiadomości generowany jest symetryczny klucz sesji, za pomocą którego szyfrowane są dane; dopiero klucz symetryczny szyfrowany jest za pomocą asymetrycznej pary kluczy. W efekcie dostajemy dwa komponenty: symetrycznie zaszyfrowany blok danych oraz asymetrycznie zaszyfrowany klucz symetryczny. Następnie odbiorca po prostu korzysta z odpowiedniego klucza asymetrycznego aby wydobyć klucz symetryczny, za pomocą którego to klucza deszyfrowany jest blok danych.

Pierwszą aplikacją implementującą ww. algorytmy po tym, jak zostały one upublicznione, była PGP (Pretty Good Privacy) Phila Zimmermana, opublikowana w 1991 w systemie biuletynowym. Zyskała ona wysoką popularność, ale także stawała się coraz bardziej komercyjna. Nie każda wersja PGP dostępna była w postaci kodu źródłowego. Ponadto niedozwolone było eksportowanie PGP ze Stanów Zjednoczonych w postaci programu komputerowego (istniały specjalne wersje międzynarodowe, 5.0i; były one drukowane na papierze i legalnie eksportowane w postaci książek, zaś poza granicami USA kod był skanowany i przetwarzany programami OCR), a sam program zawierał opatentowane algorytmy. Ze względu na to, że nie zawsze możliwe było społeczne zapoznanie się z kodem źródłowym, wersjom tym nie można było w pełni zaufać – mogły one przykładowo posiadać zaimplementowane bez wiedzy ogółu tylne drzwi bądź algorytmy klucza głównego. Wykorzystanie kodu kryptograficznego to sprawa zaufania. Aby uniknąć problemów z patentami i licencjami, rozpoczęto prace nad GnuPG (autorstwa Wernera Kocha). GnuPG implementuje tak zwany Standard OpenPGP (RFC 2440, często nazywany także PGP/MIME), oparty na PGP (tym, co napisał Phil Zimmerman).

Byłoby jednak zbyt prosto, gdyby istniał tylko jeden standard: istnieje także S/MIME (Secure MIME, RFC 2822). S/MIME wykorzystuje (niektóre) szyfry wykorzystywane także przez OpenPGP, jednak oba standardy posiadają różne formaty kluczy oraz wiadomości i z tego względu są niekompatybilne. Ponadto oba standardy wykorzystują różne modele zaufania: podczas gdy OpenPGP pozwala na stworzenie dużej *sieci zaufania* , S/MIME korzysta z silnie hierarchicznych certyfikatów opartych na X.509 v3 (standard X.509 określa m.in. standardowe formaty certyfikatów kluczy publicznych oraz algorytm walidacji ścieżki certyfikacji – patrz Wikipedia).

Algorytmy skrótów

Protokoły kryptograficzne wykorzystują algorytmy generujące tak zwane *odciski palców*, czy też *wartości skrótu*, danych. Tego rodzaju skrót jest bardzo krótki, nie można zrekonstruować danych z ich wartości skrótu (w przeciwnym wypadku byłyby to najlepsze znane algorytmy kompresji), a wartość skrótu powinna być definitywna. Ponadto musi być niemożliwe (a przynajmniej niemal niemożliwe) wygenerowanie dwóch różnych dokumentów o tej samej wartości skrótu – tak zwana *generacja kolizji.*

Możliwe że widziałeś już kiedyś skróty, sprawdzając poprawność pobranych pakietów z oprogramowaniem (na przykład za pomocą *md5sum* bądź *sha1sum*). Algorytmy: MD5 i SHA1 są powszechnie stosowane w kryptografii, a zwłaszcza w podpisach elektronicznych; z drugiej strony, badacze odkryli sposoby na ograniczenie liczby testów przy poszukiwaniu kolizji o kilka rzędów wielkości. Dla MD5 istnieje przykład, w którym naukowcy wygenerowali dwa różne pliki postscript o takiej samej wartości skrótu MD5. Pierwszy z nich to list rekomendacyjny szefa Alice, drugi zaś – rozkaz rzymskiego imperatora Gajusza Juliusza Cezara.

Z tego względu MD5 należy traktować jako niebezpieczny. Po-

dobnie rzecz dzieje się w przypadku SHA1. Niemniej, MD5 i SHA1 są wciąż w użyciu ze względu na to, iż są one częścią algorytmu DSS. Tak długo, jak będzie to prawdą, MD5 i SHA1 wciąż będą używane na przykład w GnuPG. GnuPG implementuje wprawdzie lepsze algorytmy, ale np. SHA256 korzysta z kluczy RSA, nie DSS. Niestety wygląda na to, że trzeba będzie z tym żyć tak długo, aż oficjalny standard NIST nie pozwoli na obejście tego problemu. Jest jednak możliwe skonfigurowanie kluczy GnuPG tak, by unikały one stosowania MD5. SHA-1 z kolei jest obowiązkowym elementem standardu OpenPGP, można jednak ograniczyć prawdopodobieństwo użycia go poprzez zmianę priorytetów różnych algorytmów skrótu. Wrócimy do tego później.

Generacja kluczy

GnuPG może już być zainstalowany w twoim linuksowym systemie. Spróbuj wywołać `gpg --version`; jeżeli GnuPG jest już dostępny, powinieneś zobaczyć jego numer wersji oraz (skróconą) listę zaimplementowanych w obecnej wersji algorytmów kryptograficznych oraz kompresji:

```
......> gpg (GnuPG) 1.4.2.2
[..]
Home: ~/.gnupg
Supported algorithms:
Pubkey: RSA, RSA-E,
 RSA-S, ELG-E, DSA
Cipher: 3DES, CAST5, BLOWFISH,
 AES, AES192, AES256, TWOFISH
Hash: MD5, SHA1, RIPEMD160,
SHA256, SHA384, SHA512
Compression:
 Uncompressed, ZIP, ZLIB, BZIP2
```

Jesteśmy teraz gotowi do wygenerowania naszej pierwszej pary kluczy GnuPG. Aby rozpocząć proces generacji, wpisz

```
.............> gpg --gen-key
Please select
what kind of key you want:
   (1) DSA and Elgamal (default)
   (2) DSA (sign only)
   (5) RSA (sign only)
Your selection?
```

Tabela 1. *Lista kodów*

Kod	Algorytm
Symetryczne szyfry	
S1	IDEA
S2	3DES
S3	CAST5
S4	BLOWFISH
S7	AES128
S8	AES192
S9	AES256
S10	TWOFISH
Algorytmy skrótów	
H1	MD5
H2	SHA1
H3	RipeMD160
H8	SHA256
H9	SHA384
H10	SHA512
Algorytmy kompresji	
Z1	ZIP
Z2	ZLIB
Z3	BZIP2

Wybierz tu opcję domyślną. Para kluczy DSA (wykorzystywana w podpisach) będzie miała 1024 bity długości, można jednak zmienić rozmiar pary kluczy ElGamal. Na ogół wystarczającą liczbą jest 2048. Od pewnego momentu przestaje mieć sens dalsze wydłużanie klucza, łatwiej bowiem wtedy torturować odpowiednią osobę by zdobyć klucz prywatny, niż próbować go złamać. Niestety użytkownik pozostaje najsłabszym ogniwem łańcucha.

```
DSA keypair will have 1024 bits.
ELG-E keys may be
between 1024 and 4096 bits long.
What keysize do you want? (2048)
Dlatego po prostu wciśnij <Enter>.
Requested keysize is 2048 bits
Please specify how long the key
should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
```

Na ogół wpisujemy tutaj 0. Jeżeli chcesz zmieniać klucz co roku, możesz wpisać tutaj coś innego. Jeżeli jednak korzystasz z tak zwanych serwerów kluczy do dystrybucji swojego klucza bądź kluczy, nieważne klucze będą akumulowane na serwerach – nie można ich zeń kasować, można je co najwyżej odwołać.

Następnym krokiem będzie teraz umieszczenie w kluczu, jeżeli chcemy, trochę informacji osobistych. Jeżeli chcesz brać udział w sieci zaufania i pozwolić innym *podpisywać* twój klucz publiczny, sygnalizując w ten sposób że ci ufają, będzie miało sens umiesz-

czenie w nim adresu e-mail oraz prawdziwego imienia i nazwiska. Ogólnie rzecz biorąc, można tutaj wpisać cokolwiek.

```
You need a user ID to identify your
                      key;
 the software constructs the user ID
from the Real Name,
 Comment and Email Address
in this form:
    "Heinrich Heine (Der Dichter)
 <heinrichh@duesseldorf.de>"
Real name: Alice C
mail address: alice@example.com
Comment:
You selected this USER-ID:
    "Alice C <
alice@example.com>"
Change (N)ame, (C)omment,
 (E)mail or (O)kay/(Q)uit?
```

Naciśnij (O). Teraz wprowadź swoje zdanie kodowe, inaczej nazywane *mantrą*. Powinno być ono tak długie jak tylko jest to możliwe, powinieneś być jednak w stanie je zapamiętać. 30-40 znaków powinno wystarczyć, w miarę możliwości nie stosuj jednak tutaj słów ze słownika bądź zdań z książek. Mantra to ostatni bastion pomiędzy kluczem prywatnym a zewnętrznym światem, niech zatem będzie ona dobra. Jeżeli chcesz ją zapisać, umieść odpowiednią kartkę w sejfie. Może się zdarzyć, że przed rozpoczęciem procesu generacji klucza trzeba będzie wpisać mantrę dwukrotnie. GnuPG informuje cię potem, że dobrym pomysłem będzie poruszanie trochę myszą, porobienie czegoś na klawiaturze i tak dalej. Do generacji klucza GnuPG potrzebuje liczb losowych. Jakość tych liczb jest krytyczna. Współczesne dystrybucje Linuksa stosują generatory liczb losowych, które nadają się do takich potrzeb.

W ten sposób proces generacji klucza kończy się. GnuPG przedstawia zestawienie właściwości klucza oraz jego dane identyfikacyjne, na przykład:

```
pub   1024D/E7318B79 2006-03-17
      Key fingerprint
 = 6DB6 3657 EE80 E74D 164B
 C978 6500 F1EF E731 8B79
uid Alice C <alice@example.com>
sub   2048g/2B381D4B 2006-03-17
```

Linia zaczynająca się od *pub 1024D* informuje nas, że główny klucz ma długość 1024 bitów (klucze DSA zawsze są tej długości), że jest to klucz DSA (oznaczenie D) oraz że jego key-ID to E7318B79. Numer ten będzie identyfikować twój klucz na światowych serwerach kluczy. Następna linijka zawiera odcisk palca naszego klucza. Kiedy klucz twój jest podpisywany przez innych użytkowników, odcisk palca wykorzystywany jest do identyfikacji (zauważ, że identyfikator klucza przypomina cztery ostatnie bajty jego odcisku palca). Linia zaczynająca się od *sub 2048g* informuje nas, że podklucz jest typu ElGamal (g) i ma długość 2048 bitów. Całość, zawierająca potencjalnie dalsze podklucze i inne dane tożsamościowe (np. adresy e-mail itd.), zawsze identyfikowana będzie jako *key-ID E7318B79.*

Generacja certyfikatu odwołania klucza

Jest bardzo ważne, by w następnym kroku wygenerować tak zwany certyfikat odwołania klucza. Pozwala on nam odwołać nasz klucz, co oznacza oznakowanie go jako np. *nieważny* bądź więcej nie używać. Jeżeli klucz twój zostanie w jakiś sposób skompromitowany bądź ukradziony, odwołanie jest jedynym sposobem na przekazanie światu, że nie należy go więcej używać. Z certyfikatem należy być jednak bardzo ostrożnym. Jeżeli zostanie on ukradziony, złodziej może odwołać twój klucz i wysłać go na serwer kluczu, czyniąc go w ten sposób bezużytecznym – a w dodatku nie potrzebuje on do tego ani twojego klucza prywatnego, ani twojej mantry. Po wysłaniu i rozprzestrzenieniu certyfikatu nie jest możliwe usunięcie go z klucza.

Celem wygenerowania osobistego certyfikatu odwołania wywołujemy następujące polecenie:

```
...> gpg --gen-revoke <your key-ID>
```

i podajemy żądane informacje. Na ogół generuje się certyfikat odwołujący klucz bez żadnego szczególnego powodu, można zatem pozostawić tutaj *the key is not used anymore*. Po wpisaniu mantry GnuPG wyświetli certyfikat na standardowym wyjściu. Najlepszą opcją jest teraz zapisanie go na kawałku papieru i umieszczenie w sejfie. Jeżeli chce się go wydrukować, warto być świadomym faktu, że dokument ten może przejść przez serwery drukowania, które mogą składować dane. Możesz także zapisać certyfikat na dysku i także umieścić go w sejfie, ale dyski tracą z wiekiem dane.

W sytuacji gdy wystąpiły jakieś problemy z kluczem (zgubiłeś go, został ukradziony albo po prostu nie chcesz go już używać), po prostu wczytaj ów certyfikat do pęku kluczy publicznych i wyślij go na serwer kluczy. Więcej o importowaniu i eksportowaniu kluczy powiemy poniżej. Certyfikat odwołania można traktować jak dowolny plik z kluczami publicznymi (jednak póki co nie rób tego).

```
 > gpg --import
<rev_certificate_filename>
```

Serwery kluczy

Nasz klucz jest gotów do użytku. Jego publiczna część może być zapisana do pliku, który następnie rozprzestrzenimy pośród przyjaciół, albo umieszczona na międzynarodowych serwerach kluczy. Wysyłanie klucza publicznego na serwer zdecydowanie nie jest jedna zalecane dopóki, dopóty nie zyskasz doświadczenia w pracy z nową parą kluczy. Aby wysłać klucz publiczny na serwer, wydaj poniższe polecenie:

```
 > gpg --send-keys <key-ID>
```

Zaś do pobrania klucza z serwera posłuży nam:

```
 > gpg --recv-keys <key-ID>
```

Może być konieczne podanie adresu serwera kluczy. Werner Koch za-

leca korzystanie z tak zwanych serwerów kluczy SKS, jako że są one w stanie poradzić sobie ze wszystkimi informacjami, jakie może zawierać plik kluczy. Adres serwera określić można za pomocą opcji --keyserver. Przykładowo, w Polsce można skorzystać z *sks.keyserver.penguin.pl*, w Niemczech zaś z *sks.keyserver.penguin.de*. Serwery kluczy wymieniają między sobą informacje, a zatem nasz klucz automatycznie trafi do dystrybucji.

Pęki kluczy

Zbiory albo pęki publicznych i prywatnych kluczy znaleźć można w katalogu *~/.gnupg*. Warto się upewnić, że żaden inny użytkownik nie może czytać plików w tym katalogu.

Import i eksport kluczy

Aby wyeksportować swój klucz publiczny do pliku o nazwie mykey.txt, wywołaj polecenie:

```
> gpg --export --armor <twój key-ID> >
                    mykey.txt
```

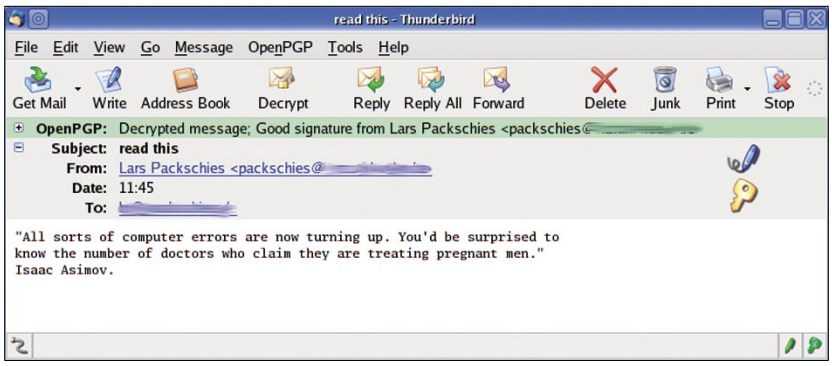
Opcja --armor powoduje, że klucz wypisany zostanie w postaci zrozumiałej dla człowieka. Key-ID może być zastąpiony dowolnymi danymi użytkownika zawartymi w kluczu, na przykład nazwiskiem bądź adresem e-mail.

Aby wczytać klucz innego użytkownika, po prostu weź otrzymany plik i dokonaj jego importu do swojego publicznego pęku. Tak wyglądałoby to w przypadku klucza, który Alice otrzymała od Boba (klucz znajduje się w pliku bobpublic.txt):

```
 > gpg --import bobpublic.txt
gpg: key 20ACB216:
 public key "Bob B
<bob@example.com>"
 imported
gpg: Total number processed: 1
gpg:        imported: 1
```

Edycja, definicja zaufania i podpisywanie kluczy

Istnieje tutaj jeden drobny haczyk: oprócz po prostu rozpoczęcia korzystania z klucza Boba Alice powinna uprzednio wyrazić swoje dlań zaufanie. Do wykonania tego GnuPG zapewnia edytor kluczy, uruchamiany poleceniem `gpg --edit-key <key-ID>`. Również i w tym przypadku `key-ID` może zostać zastąpiony np. przez imię Bob bądź jego adres e-mail.



Rysunek 2. *Zielona linia przedstawia stan podpisu. Wiadomość była zaszyfrowana i podpisana, co pokazują ikony: klucza i pióra po prawej stronie. Można na nich kliknąć, aby uzyskać więcej informacji o wykorzystanych do tego kluczach*

Przegląd poleceń edytora uzyskać można wpisując w jego linii poleceń `help`. Aby ustawić poziom zaufania dla klucza Boba, Alice uruchamia edytor i otwiera ten klucz. Wyświetlone zostaną informacje o kluczu, między innymi o nieznanym poziomie zaufania i ważności.

```
pub  1024D/20ACB216  created:
 2006-03-17  expires:
 never       usage: CS
                    trust:
 unknown       validity: unknown
sub  2048g/6B99CC08  created:
 2006-03-17  expires:
 never       usage: E
[ unknown] (1).
 Bob B <bob@example.com>
```

Bardzo ważne jest by upewnić się, czy klucz ten rzeczywiście należy do osoby, o której Alice myśli jako o Bobie – a na kolejnym etapie, czy Bob rzeczywiście jest osobą, za którą podaje się Alice. Jest możliwe, że podszywająca się pod Boba trzecia osoba, nazwijmy ją tradycyjnie Mallorym, przekaże Alice niewłaściwy klucz. Jeżeli teraz Alice zaufałaby temu kluczowi i szyfrowała nim wiadomości dla Boba, to Mallory mógłby je czytać zamiast Boba. Aby uniknąć tego rodzaju ataku Alice mogłaby poprosić Boba o okazanie dokumentu tożsamości i przekazanie klucza osobiście, mogłaby także sprawdzić odcisk palca klucza i zapytać Boba, czy jest on poprawny:

```
 > gpg --fingerprint bob
[..]
 Key fingerprint
 = 6871 3E47 AEEE 7424 10EF
 B544 3EC0 383B 20AC B216
```

Kiedy tożsamość Boba i poprawność jego klucza zostały już potwierdzone, Alice wydaje polecenie trust:

```
Please decide how far you trust
this user to correctly verify
other users' keys (by looking
at passports, checking
fingerprints from different
sources, etc.)
  1 = I don't know or won't say
  2 = I do NOT trust
  3 = I trust marginally
  4 = I trust fully
  5 = I trust ultimately
  m = back to the main menu
```

GnuPG oczekuje, że Alice oceni doświadczenie Boba w zarządzaniu kluczami oraz na ile jest on według niej wiarygodny. Wartość 5 zarezerwowana jest dla osobistych kluczy, nie może jej mieć żaden klucz innego użytkownika. Alice ufa Bobowi w pełni (4):

```
pub  1024D/20ACB216  created:
 2006-03-17  expires:
 never       usage: CS
                    trust:
 full          validity: unknown
sub  2048g/6B99CC08  created:
 2006-03-17  expires:
 never       usage: E
[ unknown] (1). Bob B <bob@example.com>
Please note that the shown key validity
 is not necessarily correct
unless you restart the program.
```

Mimo wszystko, klucz w ciąż nie jest ważny. Aby móc uczynić go ważnym, Alice ma możliwość podpisania go swoim kluczem prywatnym. Z poziomu edytora kluczy można to uczynić poleceniem `sign`. Klucz może być następnie wyeksportowany do pliku (patrz wyżej) i wysłany z powrotem do Boba, który może wczytać go do swojego publicznego pęku. Podpisywanie kluczy innych użytkowników ma na celu tworzenie *sieci zaufania*. Jeżeli Alice nie chce przekazać podpisanego klucza do Boba, a jedynie chce by był on ważny w jej własnym pęku, może podpisać go tylko lokalnie za pomocą polecenia lsign.

```
 > lsign pub  1024D/20ACB216  created:
 2006-03-17  expires:
 never       usage: CS
  trust: full     validity: unknown
 Primary key fingerprint:
 6871 3E47 AEEE 7424 10EF
 B544 3EC0 383B 20AC B216
  Bob B <bob@example.com>
Are you sure that you want to sign
 this key with your
key "Alice C <alice@example.com>"
 (E7318B79)
The signature will be marked
 as non-exportable.
Really sign? (y/N)
```

Informacja pod koniec wynika z lokalności tworzonego podpisu. Alice wpisuje y aby móc podpisać klucz, następnie zaś musi wpisać swoją mantrę – konieczne jest otwarcie jej klucza prywatnego.

```
You need a passphrase to unlock the
                    secret key for
user: "Alice C <alice@example.com>"
1024-bit DSA key,
 ID E7318B79, created 2006-03-17
Enter passphrase:
```

Po wprowadzeniu poprawnego zdania kodowego (mantry) klucz Boba staje się ważny dla Alice. Może okazać się, że trzeba zrestartować edytor kluczy (użyj polecenia `quit`) aby zaktualizować wewnętrzną bazę zaufania GnuPG.

```
pub  1024D/20ACB216  created:
 2006-03-17  expires:
 never       usage: CS
                    trust:
 full          validity: full
sub  2048g/6B99CC08  created:
 2006-03-17  expires:
 never       usage: E
[  full  ] (1). Bob B <bob@example.com>
```

Jeżeli Bob wczyta i podpisze klucz Alice stosując opisany powyżej schemat (ewentualnie generując nielokalny podpis), mogą oni zacząć wysyłać do siebie tajne wiadomości. Ogólnie rzecz biorąc, wysłanie tajnej wiadomości polega po prostu na napisaniu jej i zaszyfrowaniu kluczem publicznym odbiorcy. Jeżeli korzysta się z rozumiejącego GnuPG programu pocztowego (jak na przykład Mozilla Thunderbird z wtyczką Enigmail), robi to za nas program. Na początek jednak użyjemy interfejsu linii poleceń GnuPG.

Sieć zaufania

Podpisywanie i ufanie kluczom innych użytkowników tworzy sieć zaufania. Wyobraźmy sobie, że chcemy napisać tajną wiadomość do pewnego odbiorcy, którego klucz pobraliśmy z serwera kluczy. Nigdy nie spotkaliśmy tej osoby osobiście, a chcemy wiedzieć, czy dany klucz rzeczywiście do niej należy. My nie przeszliśmy przez procedurę sprawdzania odcisku palca, wysyłanie testowych listów itd., ale mógł to zrobić ktoś inny. Jeżeli rzeczywiście ufamy tej trzeciej osobie, nieznany klucz automatycznie staje się dla nas ważny.

Sieć zaufania opiera się na kilku prostych zasadach. Można nimi nieco manipulować, ale w ogólności wyglądają one następująco. Klucz jest ważny, jeżeli:

- podpisaliśmy go, lub
- został podpisany kluczem, do którego mamy pełne zaufanie, lub
- został podpisany trzema kluczami, którym ufamy marginalnie
- oraz ścieżka pomiędzy naszym kluczem, a kluczem odbiorcy składa się z nie więcej niż pięciu kroków.

Edycja preferencji klucza

Jak wspomnieliśmy powyżej, nasz klucz może zostać skonfigurowany tak, by unikać stosowania szczególnych algorytmów takich jak SHA-1 czy MD5, a przynajmniej dać wyższy priorytet innym algorytmom: możliwe jest także wyłączenie DES i korzystanie z szyfru Blowfish jako preferowanego szyfru symetrycznego, jeżeli tego sobie życzymy.

Również i te ustawienia można zmieniać za pomocą edytora kluczy. Alice na przykład uruchomiłaby edytor za pomocą następującego polecenia:

```
 > gpg --edit-key alice
Secret key is available.
pub  1024D/E7318B79  created:
 2006-03-17  expires:
 never       usage: CS
                    trust:
 ultimate      validity: ultimate
sub  2048g/2B381D4B  created:
 2006-03-17  expires:
 never       usage: E
[ultimate] (1).
Alice C <alice@example.com>
```

Wykorzystywane przez dany klucz algorytmy można wyświetlić za pomocą poleceń edytora: `showpref` i `pref`. Pierwsze wyświetla parametry klucza w nieco obszerniejszej formie, drugie zastępuje nazwy algorytmów ich kodami. Klucz Alice jest skonfigurowany następująco:

```
Command> showpref
pub  1024D/E7318B79  created:
 2006-03-17  expires:
 never       usage: CS
  trust: ultimate   validity: ultimate
[ultimate] (1).
Alice C <alice@example.com>
     Cipher: AES256, AES192,
 AES, CAST5, 3DES
     Digest: SHA1, RIPEMD160,
 SHA256, MD5
   Compression: ZLIB, BZIP2, ZIP,
 Uncompressed
  Features: MDC, Keyserver no-modify
```

Jak łatwo zauważyć, SHA1 jest pierwszym algorytmem w linii *Digest*, nie można jednak ustawiać preferencji stosując nazwy algorytmów – trzeba zastąpić je odpowiednimi kodami. Polecenie pref wyświetla dokładnie linię kodów danego klucza:

```
Command> pref
pub  1024D/E7318B79  created:
 2006-03-17  expires:
 never     usage: CS
   trust: ultimate    validity:
                      ultimate
[ultimate] (1).
Alice C <alice@example.com>
     S9 S8 S7 S3 S2 H2 H3
 H8 H1 Z2 Z3 Z1
 [mdc] [no-ks-modify]
```

S to kody algorytmów symetrycznego szyfrowania, H – algorytmów skrótów, Z zaś – algorytmów kompresji.

Do ustawienia preferencji służy polecenie `setpref`, przyjmujące na wejściu linię kodów. Jeżeli Alice chce się pozbyć MD5, ale chce zachować pozostałe ustawienia nienaruszone, skopiuje ona i linię kodów z przedstawionego powyżej wyjścia polecenia pref, pomijając jedynie *H1* i przesuwając *H2* na koniec listy algorytmów skrótów.

```
Command> setpref S9 S8 S7 S3 S2
H3 H8 H2 Z2 Z3 Z1
Set preference list to:
  Cipher: AES256, AES192,
 AES, CAST5, 3DES
 Digest: RIPEMD160, SHA256, SHA1
  Compression: ZLIB, BZIP2, ZIP,
 Uncompressed
  Features: MDC, Keyserver no-modify
Really update the preferences? (y/N)
Say yes here;
You need a passphrase to unlock
 the secret key for user:
"Alice C <alice@example.com>"
1024-bit DSA key,
 ID E7318B79, created 2006-03-17
Enter passphrase:
```

Po wprowadzeniu poprawnego zdania kodowego atrybuty klucza są aktualizowane. Rezultaty tej operacji pokazuje polecenie `showpref`:

```
Command> showpref
pub  1024D/E7318B79  created:
 2006-03-17  expires: never
                      usage: CS
   trust: ultimate      validity:
                      ultimate
[ultimate] (1).
Alice C <alice@example.com>
     Cipher: AES256, AES192,
AES, CAST5, 3DES
    Digest: RIPEMD160,
SHA256, SHA1
     Compression: ZLIB, BZIP2, ZIP,
 Uncompressed
   Features: MDC, Keyserver no-modify
```

Jak widać MD5 został wyłączony, zaś SHA1 trafił na koniec linii ale nie można go całkowicie wyłączyć. Pokazuje to użytkownikowi klucza Alice, że woli ona korzystać z RIPEMD160 bądź SHA256, niż z SHA1. W większości przypadków wystarcza to do uniknięcia korzystania z SHA1.

Konfiguracja preferencji może być również ważna gdy znajdziemy się w potrzebie wczytania klucza PGP do GnuPG bądź *vice versa*. Aby na przykład wyeksportować klucz GnuPG tak, by można było go użyć w PGP (zauważ przy tym, że w przypadku PGP nie można korzystać z kluczy ElGamal), preferencje należy ustawić na S9 S8 S7 S3 S2 S10 H2 H3 Z1 Z0.

Uwaga: niektóre wersje GnuPG używają polecenia `updpref` do uaktywnienia ustawień zmienionych za pomocą `setpref`. Przyjrzyj się wyjściu polecenia edytora `help`. Aby zakończyć bieżącą sesję, wyjdź z edytora poleceniem `quit`.

Szyfrowanie i deszyfrowanie danych

Wyobraź sobie, że Alice chce wysłać do Boba wiadomość zawierającą poufne informacje. Może napisać ją w pliku (secret.txt), a następnie zaszyfrować go za pomocą polecenia:

```
.  > gpg --recipient bob
 --encrypt --armor secret.txt
```

Wygeneruje ono plik secret.txt.asc. Teraz nawet Alice nie jest w stanie rozszyfrować wiadomości, chociaż oczywiście wciąż posiada ona oryginał. Z drugiej strony, możliwe jest wygenerowanie zaszyfrowanego pliku, które będzie mogło rozkodować dwóch bądź więcej użytkowników. W przypadku takim plik musi być zaszyfrowany z więcej niż jednym odbiorcą. Alice mogłaby to zrobić następująco:

```
 > gpg --recipient bob
 --encrypt --recipient alice --armor
                        secret.txt
Lub w skróconej postaci,
 > gpg -r bob -r alice -e -a secret.txt
```

Co się tutaj dzieje? Oryginalna wiadomość jest szyfrowana kluczem sesji, zaś tenże klucz sesji jest następnie szyfrowany w osobnych kopiach kluczami publicznymi Alice i Boba. Wszystkie te dane są następnie umieszczane razem w pliku secret.txt.asc.

Alice może wysłać ten plik do Boba, który jest w stanie go odszyfrować za pomocą opcji `decrypt`. Jego prywatny klucz zostanie wykorzystany automatycznie, ale Bob musi wprowadzić swoją mantrę, by go odblokować.

```
 > gpg --decrypt secret.txt.asc
You need a passphrase to unlock
 the secret key for
user: "Bob B <bob@example.com>"
2048-bit ELG-E key, ID 6B99CC08,
```

```
 created 2006-03-17
(main key ID 20ACB216)
Enter passphrase:
Bob enters his passphrase here.
gpg: encrypted with 2048-bit ELG-E key,
 ID 2B381D4B, created 2006-03-17
  "Alice C <alice@example.com>"
gpg: encrypted with 2048-bit ELG-E key,
 ID 6B99CC08, created 2006-03-17
  "Bob B <bob@example.com>"
Hi Bob, come to the willow tree tonight
 at 8. we have to talk, Alice.
```

W tym szczególnym przypadku ostatnia linijka to oryginalna wiadomość od Alice. Istnieje ponadto możliwość wykorzystywania GnuPG do szyfrowania danych za pomocą szyfrów symetrycznych; program wykorzystuje do tego opcję `conventional`. Można także wybrać algorytm szyfrujący. Aby zaszyfrować plik `mail.tgz` za pomocą AES256, po prostu wpisz

```
 > gpg --cipher-algo aes256
 -c mail.tgz
Enter passphrase:
Repeat passphrase:
```

Zdanie kodowe trzeba wpisać dwukrotnie, aby uniknąć literówek. Jeżeli nie określisz za pomocą opcji -o pliku wyjściowego, GnuPG skorzysta z nazwy pliku wejściowego wydłużonej o przyrostek *.gpg*. Aby rozszyfrować te dane, wystarczy wpisać

```
 > gpg -o mail2.tgz mail.tgz.gpg
gpg: AES256 encrypted data
Enter passphrase:
```

Plik mail2.tgz zawierać będzie oryginalne dane.

Podpisy i ich walidacja

Bob może teraz przeczytać wiadomość i wie, że wiadomość z pewnością była przeznaczona dla niego (została zaszyfrowana jego kluczem publicznym), nie ma jednak pewności, czy rzeczywiście została ona napisana i wysłana przez *Alice* – wiadomość nie posiada bowiem podpisu. Aby takowy dodać, Alice może zaszyfrować wiadomość podając jednocześnie dodatkową opcję: tutaj skorzystała ona z opcji `--sign`. Umieszcza ona podpis i podpisany tekst w jednym pliku, o nazwie secret.txt.asc. Zakładając, że Bob zaufał kluczowi Alice oraz go podpisał, polecenie `gpg --decrypt secret.txt.asc` zwróci mu, co następuje:

```
2048-bit ELG-E key,
ID 6B99CC08,
created 2006-03-17
(main key ID 20ACB216)
gpg: encrypted with 2048-bit ELG-E key,
 ID 2B381D4B, created 2006-03-17
  "Alice C <alice@example.com>"
gpg: encrypted with 2048-bit ELG-E key,
 ID 6B99CC08, created 2006-03-17
  "Bob B <bob@example.com>"
Hi Bob, come to the willow
tree tonight at 8. we have to talk,
                    Alice.
gpg: Signature made
Fri 17 Mar 2006 04:05:26 PM CET
 using DSA key ID E7318B79
gpg: Good signature from
 "Alice C <alice@example.com>"
```

Teraz Bob może być pewien, że Alice napisała tę wiadomość – można bowiem zweryfikować jej podpis.

Szyfruj swoją pocztę: Thunderbird i Enigmail

Korzystanie z GnuPG w przedstawiony powyżej sposób może być irytujące, zwłaszcza gdy chce się wykorzystywać regularnie funkcje kryptograficzne do podpisywania bądź szyfrowani poczty. W przypadku takim każda wiadomość musiałaby zostać zapisana w systemie plików, przetworzona przez GnuPG, a następnie ponownie otwarta w programie pocztowym i wysłana.

Aby uprościć i uprzyjemnić użytkownikom życie, niemal wszystkie klienckie programy pocztowe albo implementują funkcje kryptograficzne, albo dają dostęp do odpowiednich programów przez specjalne wtyczki – np. KMail, Mutt, Pine, Sylpheed, Emacs czy Balsa, żeby wymienić zaledwie kilka.

Jedną z najpotężniejszych i niezawodnych kombinacji jest w tej dziedzinie Thunderbird (samodzielny klient poczty z projektu Mozilla) wspólnie z wtyczką dla GnuPG o nazwie Enigmail. Enigmail dodaje do klienta poczty menu *OpenPGP*; jest ona dostępna pod Linuksa, Mac OS X oraz Windows, wymagana jest zainstalowana instancja GnuPG. GnuPG dla twojej platformy znajdziesz pod adresem *http://www.gnupg.org/download*. Jeżeli chcesz korzystać z GnuPG pod Windows rzuć okiem na plik GnuPG.README.Windows w folderze GnuPG w menuStart; gpg można używać z poziomu linii poleceń Windows albo za pośrednictwem Windows Privacy Tray WinPT.

Enigmail pobrać można spod adresu *http://enigmail.mozdev.org/*. Aby zainstalować tę wtyczkę wejdź do menu *Narzędzia* i wybierz *Rozszerzenia*, a następnie *Instaluj*. Wskaż przeglądarce świeżo pobrany plik wtyczki Enigmail i wybierz Instaluj. Enigmail będzie dostępna po restarcie Thunderbirda.

Enigmail musi być uaktywniona osobno dla każdej tożsamości e-mail, dla której chcesz jej używać. Czyni się to w menu *Edycja*, *Ustawienia kont* (w menu *Narzędzia* w przypadku Windows). Trzeba zaznaczyć pole *Uruchom obsługę OpenPGP* (*Enigmail*) *dla tej tożsamości*. Jeżeli Enigmail ma problem z określeniem identyfikatora domyślnego klucza na podstawie adresu e-mail, okno to pozwala wybrać odpowiedni identyfikator ręcznie. Przycisk *Zaawansowane* otwiera dialog *Preferencje OpenPGP*; można się do niego dostać także z menu *OpenPGP* (pozycja *Preferencje*). Może zaistnieć konieczność podania w zakładce *Podstawy* ścieżki pliku binarnego gpg, jeżeli nie została ona ustawiona automatycznie. Ponadto ważne jest sprawdzenie, czy zaznaczone jest pole *Szyfruj do siebie* w zakładce *Wysyłanie*, w przeciwnym razie nie bylibyśmy w stanie czytać wysłanej przez nas zaszyfrowanej poczty.

Inną wartą uwagi opcją jest *Zawsze korzystaj z PGP/MIME* w zakładce PGP/MIME; jeżeli nie jest ona aktywna Enigmail korzysta z tak zwanego formatu inline PGP, w którym nie są szyfrowane załączniki.

Aby zaszyfrować bądź podpisać list, naciśnij przycisk OpenPGP w oknie kompozycji. Możesz wybrać tam *Podpisz wiadomość, Zaszyfruj wiadomość* bądź i jedno, i drugie. Jeżeli chcemy podpisać wiadomość, Enigmail wyświetli dialog z prośbą o zdanie kodowe aby uzyskać dostęp do naszego klucza prywatnego. Następnie GnuPG przetwarza dane, zanim nasz klient poczty wyśle je w świat. Jeżeli chcesz zaszyfrować wiadomość, GnuPG musi znać klucz publiczny jej adresata.

Po otrzymaniu zaszyfrowanej i przeznaczonej dla nas wiadomości, jesteśmy proszeni o wprowadzenie zdania kodowego. Thunderbird sprawdza także podpis (jeżeli takowy występuje) i informuje, czy jest on poprawny.

Zaszyfrowane pliki-kontenery i systemy plików

Kryptografia to nie tylko GnuPG i szyfrowanie plików bądź wiadomości. Obok wielu innych jej potencjalnych zastosowań, takich jak zabezpieczanie komunikacji w Internecie (ssh, scp) bądź serwerów poczty, WWW itd. (nie pokazane tutaj), Linux pozwala także na dość łatwe szyfrowanie plików-kontenerów oraz systemów plików. Pokrótce przedstawimy tutaj LoopAES oraz DM-Crypt, istnieją jednak oczywiście inne opcje, a ponadto możliwe jest wykonywanie podobnych operacji pod innymi systemami operacyjnymi. Poniżej przedstawimy dwa przykłady.

Szyfrowanie partycji dzięki LoopAES

LoopAES korzysta z linuksowego urządzenia loopback (z włączonymi rozszerzeniami kryptograficznymi) by stworzyć system plików wewnątrz kontenera bądź na partycji jako urządzeniu. Zamierzamy tutaj pokazać szybki i prosty scenariusz, w którym wykorzystamy wolną partycję na dysku (w naszym przypadku noszącą nazwę /dev/sdc3) by stworzyć system plików na zaszyfrowanym urządzeniu loopback. System plików będzie szyfrowany i deszyfrowany w locie, ale by uzyskać do niego dostęp potrzebny będzie klucz Właściwy klucz będzie przechowywany w symetrycznie zaszyfrowanym pliku. Brzmi to nieco skomplikowanie, jednak zapoznając się s przykładem zobaczysz, jak to rozwiązanie działa.

Być może nie będzie ono działać na wszystkich systemach, jednak z powodzeniem udało nam się to sprawdzić pod Fedora Core 4. Niektóre systemy potrzebować będą zmodyfikowanej wersji urządzenia loopback; trochę wskazówek na ten temat znaleźć można pod adresem *http://loop-aes.sourceforge.net/.*

Po pierwsze wybieramy partycję. Może ona znajdować się na pen drive'ie USB bądź na zewnętrznym twardym dysku, może też być partycją na dysku wbudowanym – musi jednak być pusta.

Po drugie, utwórz losowy klucz. W naszym przypadku pobierzemy 2925 bajtów z /dev/random, dokonamy ich konwersji do formatu base64 (za pomocą narzędzia `uuencode`, na ogół dostępnego w pakiecie *sharutils*) i skorzystamy z *head* i *tail*, by wybrać z tego losowego bloku 65 linijek. Na koniec przy pomocy GnuPG szyfrujemy te liczby algorytmem AES256:

```
 > head -c 2925 /dev/random |
uuencode -m | head -n 66 | tail -n
                        65  |
gpg --symmetric -cipher-algo
 aes256 -a > keyfile.gpg
```

W zależności od ilości entropii dostępnej w systemie operacja ta może zająć dość dużo czasu (do generowania liczb losowych za pomocą /dev/random potrzeba dużo entropii!). Teraz można umieścić plik kluczy np. na dysku USB bądź SmartCard'zie. Od tego momentu twój zaszyfrowany system plików będzie dostępny tylko wtedy, gdy podłączysz doń to fizyczne urządzenie.

Kolejnym krokiem jest inicjacja partycji danych. Wypełnimy ją raz pseudolosowymi liczbami, korzystając z /dev/zero by wygenerować strumień zer, które zostaną zaszyfrowane przez szyfrujące urządzenie loopback. Robi się to tylko raz:

```
 > head -c 15 /dev/urandom | uuencode
                          -m - |
head -n 2 | tail -n 1 | losetup -p 0 -e
                          aes256
/dev/loop3 /dev/sdc3
```


O autorze

Doktor Lars Packschies jest pracownikiem naukowym oraz administratorem oprogramowania i ochrony danych w środowisku Linux, SunOS/Solaris, IRX i AIX Regionalnego Centrum Rachunkowego (Regionales Rechenzentrum, RRZ) na Uniwersytecie w Koloni. Kontakt z autorem: *packschies@rrz.uni-koeln.de.*

W Sieci

- *http://downlode.org/Etext/alicebob.html*
- *http://www.gnupg.org*
- *http://rfc2440.x42.com – OpenPGP, RFC 2440*
- *http://rfc2822.x42.com – RFC 2822, S/MIME*
- *http://www.cits.rub.de/MD5Collisions – Historia Alice i jej szefa*
- *http://www.heise.de/newsticker/meldung/56624 – komentarze Wernera Kocha, po niemiecku*
- *http://www.gnupg.org/(de)/documentation/faqs.html*
- *http://www.stud.uni-hannover.de/~twoaday/winpt.html*

Tworzymy w ten sposób urządzenie loopback /dev/loop3 korzystające z partycji /dev/sdc3, zainicjowanej pewną ilością losowych liczb oraz korzystającej z AES256. Od tej chwili wszystko, co zapiszemy do /dev/loop3 będzie zaszyfrowane. W ten sposób strumień zer stanie się długą listą pseudolosowych liczb; korzystamy z tego sposobu, ponieważ jest on po prostu dużo szybszy niż generacja liczb losowych. Operację tę przeprowadza się tylko raz.

```
> dd if=/dev/zero of=/dev/loop3 bs
=4k conv=notrunc 2>/dev/null
```

Inicjacja zostanie zakończona, gdy urządzenie loopback jest zwalniane:

```
> losetup -d /dev/loop3
```

Teraz musimy zainicjować na naszej partycji system plików:

```
> losetup -K /path/to/your/keyfile.gpg
                        -e
AES256 /dev/loop3 /dev/sdc3
```

i

```
> mkfs -t ext2 /dev/loop3
```

Aby ponownie zwolnić urządzenie, wywołamy

```
> losetup -d /dev/loop3
```

Za każdym razem, gdy chcemy skorzystać z ten partycji, tworzymy urządzenie loopback i montujemy je w systemie plików. Jeżeli dodamy do pliku /etc/fstab następującą linię (wszystko to powinno znaleźć się w jednej linijce):

```
/dev/sdc3 /mnt/loopdev ext2
  defaults,noauto,loop=/dev/loop3,
encryption=AES256,gpgkey=naszplikkluczy
                        0 0
```

operacja ta stanie się całkiem prosto. Wystarczy wywołać:

```
> mount /mnt/loopdev
Password: zdanie kodowe pliku kluczy
```

Kontener danych zaszyfrowany przez DM-Crypt

Kolejnym przykładem, który chcę tutaj przedstawić, jest korzystanie z pliku-kontenera (czyli po prostu bloku losowych danych na twardym dysku) do przechowywania weń zaszyfrowanych danych. Wykorzystamy tutaj DM-Crypt, który powinien być dostępny w twoim systemie, pod warunkiem że korzystasz z architektury z jądrem 2.6. Jeżeli w twoim systemie narzędzie to nie działa, zajrzyj pod adres `http://www.saout.de/misc/dm-crypt`.

DM-Crypt to napisany przez Christophe'a Saouta cel dla Device Mappera służący do szyfrowania danych. Od wersji 2.6.4 jądra DM-Crypt zastępuje Cryptoloop. Device Mapper administruje wirtualnymi urządzeniami blokowymi, które z kolei mogą korzystać z fizycznych urządzeń takich jak twarde dyski czy też partycje. Istnieje całkiem spora liczba celów dla Device Mappera, na przykład ten zapewniający rozdzielanie danych pomiędzy kilkoma urządzeniami. Równie dobrze można tę swego rodzaju warstwę pośrednią wyposażyć w funkcje kryptograficzne: DM-Crypt.

Upewnij się, że w twoim jądrze aktywne są funkcje *Device mapper support* oraz *Crypt target support* (można je znaleźć pod *Device Drivers/Multi-Device-Support (RAID and LVM)*). Ponadto aktywne powinny być także *Device Drivers/Block Devices/Loopback device support* oraz *Cryptographic Options/AES cipher algorithms*.

Jeżeli korzystasz z dystrybucji Fedora Core, zainstaluj pakiet *device-mapper*; użytkownikom Debiana potrzebne będą pakiety *dmcrypt* oraz *cryptsetup*. Oprócz tego należy załadować kilka modułów jądra. Użytkownicy Red Hata bądź Fedory mogą dodać następujące linijki do pliku /etc/rc.local:

```
modprobe aes
modprobe dm_mod
modprobe dm_crypt
```

Jeżeli korzystasz z Debiana, użyj narzędzia `modconf` i wybierz *kernel/drivers/md* oraz *kernel/crypto*. Wykorzystamy kontener o rozmiarze 200 MB. Stworzymy go następująco:

```
> dd if=/dev/urandom
of=container bs=1024k count=200
```

Teraz superużytkownik może podłączyć kontener do Device Mappera poprzez DM-Crypt, w naszym przypadku korzystając z urządzenia /dev/loop4.

```
> losetup /dev/loop4 container
> cryptsetup -y create secret /dev/
                        loop4
```

Aby uniknąć skutków ewentualnych literówek przy wpisywaniu zdania kodowego, dzięki opcji -y zostaniemy o nie zapytane dwukrotnie. `Container` to nazwa pliku kontenera (może być konieczne podanie pełnej ścieżki), `secret` zaś to nazwa pliku Device Mappera (można równie dobrze użyć innej nazwy); znajdziesz go potem pod `/dev/mapper/secret`. Urządzenie nie zawiera póki co systemu plików, stworzymy go następująco:

```
> mkfs.ext2 /dev/mapper/secret
```

po czym możemy je zamontować:

```
> mount /dev/mapper/secret /mnt/secret
```

Po zakończeniu korzystania z kontenera, wpisujemy

```
> umount /mnt/secret
> cryptsetup remove secret
> losetup -d /dev/loop4
```

DM-Crypt może obsługiwać nie tylko kontenery danych, ale także całe partycje; można także łatwo zaszyfrować partycje wymiany. Powyższy przykład pokazał tylko szyfrowanie kontenera, więcej informacji o DM-Crypt znaleźć można na jego stronie domowej pod adresem *http://www.saout.de/misc/dm-crypt/* ●