Wydział Fizyki i Informatyki Stosowanej
Akademia Górniczo-Hutnicza w Krakowie

Systemy peracyjne

„Poufność danych oraz PKI: OpenSSL”

laboratorium: 06

Opracowali:
Tomasz Juszkiewicz
Adrian Kałuziński

Kraków, 2022

Spis treści

1. Wiadomości wstępne

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące szyfrowania, **PKI** (ang*. Public Key Infrastructure)*. Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej, drugiej części laboratorium.

1.1. Tematyka laboratorium

Laboratorium ma za zadanie zaprezentowanie podstawowych pojęć z zakresu poufności danych: metod szyfrowania, wyjaśnienie znaczenia i zastosowania kluczy kryptograficznych, opisania infrastruktury klucza publicznego (PKI). Celem autorów jest ponadto przedstawienie praktycznego zastosowania protokołu **SSL** poprzez aplikację OpenSSL, systemu certyfikatów oraz podpisów cyfrowych w aplikacji GnuPG.

1.2. Zagadnienia do przygotowania

Przed przystąpieniem do realizacji laboratorium należy zapoznać się z zagadnieniami dotyczącymi podstaw modelu ISO/OSI oraz podstaw zastosowania kryptografii.

- Znajomość pojęcia modelu ISO/OSI [1,2]
- Elementarna znajomość algorytmów DES/RSA [3,4,6]
- [Opcjonalnie] Znajomość standardu TSL [7]

Literatura:

[1] Andrew Stuart Tanenbaum *Sieci komputerowe*. Helion 2004.
[2] Michal Zalewski *Cisza w sieci*, Helion 2005
[3] http://pl.wikipedia.org/wiki/Kryptografia
[4] Jon Erickson *Hacking. Sztuka penetracji*. Helion 2008
[5] *ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf* [dla ambitnych]
[6] Praca Zbiorowa *Hack Proofing. Your Network.* Helion 2002
[7] IETF RFC 4366 (TSL)

1.3. Opis laboratorium

Laboratorium koncentruje się na najwyższych warstwach modelu OSI - prezentacji i aplikacji. Zostaną wyjaśnione podstawowe pojęcia obligatoryjne do zrozumienia tematu. W oparciu o aplikacje GnuPG i OpenSSL wyjaśnimy praktyczne zastosowanie metod kryptograficznych w celu uzyskania gwarantowanej poufności naszych danych przesyłanych przez kanały ograniczonego zaufania takie jak Internet.

1.4. Szyfrowanie

Kryptograficzny algorytm szyfrujący (szyfr) jest wykorzystywany do utajniania tekstu jawnego lub jego odtajnienia. Zazwyczaj jedna funkcja wykorzystywana jest do szyfrowania, a inna do deszyfrowania wiadomości (istnieją wyjątki takie jak ROT-13). Wiadomość przed zaszyfrowaniem nazywana jest tekstem jawnym, zaś wiadomość zaszyfrowaną nazywamy szyfrogramem. Proces zamiany tekstu jawnego na szyfrogram nazywamy szyfrowaniem.
Wyróżniamy dwie kategorie algorytmów kryptograficznych z kluczem:

1. Algorytmy **symetryczne** – klucz szyfrujący może być wyznaczony z klucza deszyfrującego oraz klucz deszyfrujący może być wyznaczony z klucza szyfrującego (klucz szyfrujący jest tym samym, co deszyfrujący). Przykładem takiego szyfru jest **AES** (ang. *advanced encryption standard*).
2. Algorytmy **asymetryczne** – klucz szyfrujący (publiczny) jest inny niż klucz deszyfrujący (prywatny), z klucza prywatnego można obliczyć klucz publiczny, oraz nie można wyznaczyć klucza prywatnego z publicznego. Na szyfrowaniu asymetrycznym opiera się popularny w szyfrowaniu certyfikatów algorytm **RSA** (od nazwisk autorów R. Rivesta, A. Shamira oraz L. Adlemana.)

Współczesne aplikacje takie jak OpenSSL i GnuPG wykorzystują kombinację obu tych rodzajów szyfrowania znaną pod pojęciem szyfru hybrydowego.

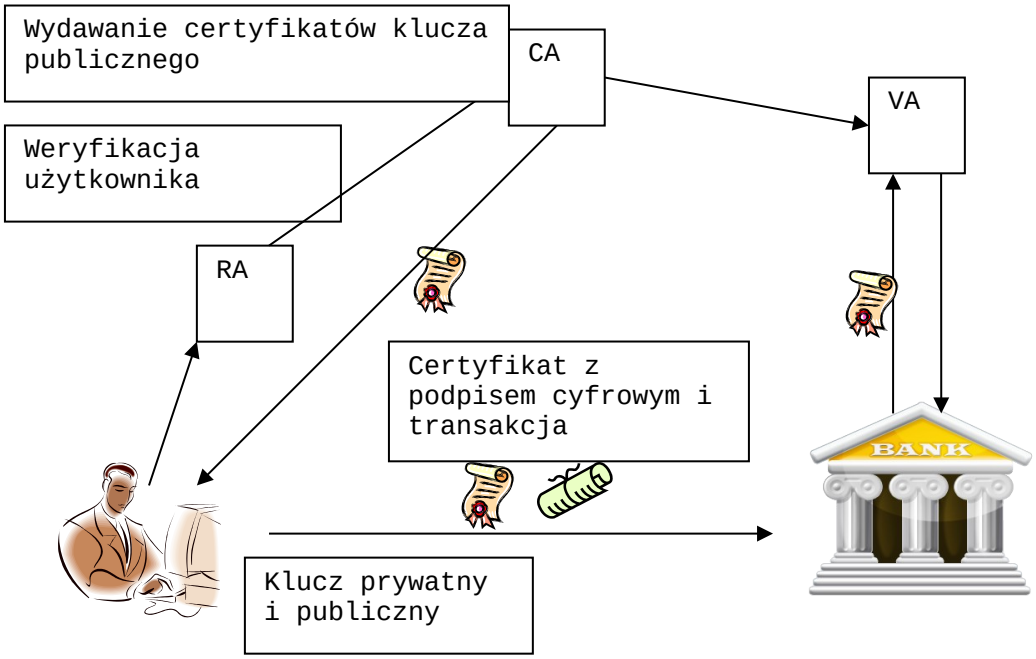
1.5. Public-key infrastructure

Jest to abstrakcyjny model urządzeń, aplikacji, polityki, użytkowników oraz procedur mający na celu tworzenie, przechowywanie, zarządzanie i rozprowadzanie cyfrowych certyfikatów klucza publicznego. Jest to kryptosystem, który składa się z urzędów certyfikacji (CA - ang. *certificate authority)*, urzędów rejestracyjnych (RA – ang. *registration autorithy*), urzędów walidacyjnych (VA- ang. *validation autority*), użytkowników certyfikatów klucza publicznego oraz software/hardware. Cała infrastruktura tworzy bezpieczną i zaufaną strukturę, której kluczowym elementem jest certyfikat. Najpopularniejszym standardem certyfikatów PKI jest X.509 w wersji trzeciej.

Do podstawowych funkcji PKI należą:

1) Weryfikacja tożsamości subskrybentów
2) Wymiana kluczy kryptograficznych

3) Wystawianie i weryfikacja certyfikatów
4) Podpisywanie i szyfrowanie przekazu
5) Potwierdzanie tożsamości



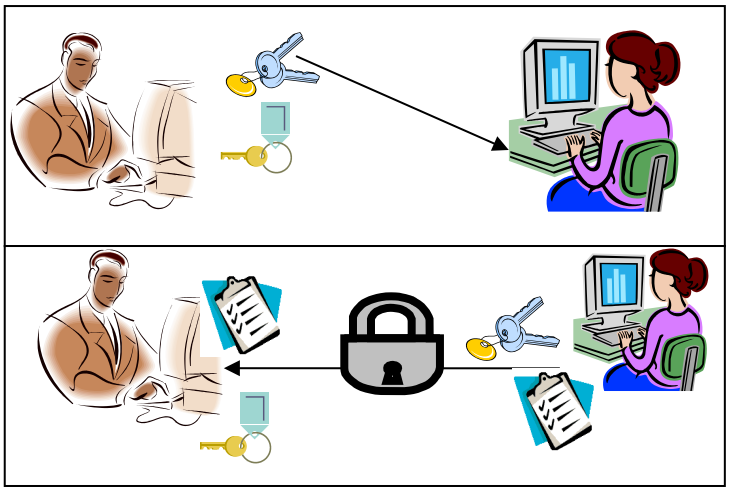
Rys 1. Model PKI

1.6. Opis elementów PKI

1.6.1. Klucz publiczny i prywatny

Kryptografia asymetryczna (klucza publicznego) - rodzaj kryptografii, w którym używa się dwóch lub więcej powiązanych ze sobą kluczy, umożliwiających wykonywanie operacji kryptograficznych. Jeden z kluczy może być udostępniony publicznie bez zagrożenia bezpieczeństwa danych zabezpieczanych tym kryptosystemem.

Kryptografii asymetrycznej – szyfrowanie i podpisy cyfrowe zakłada istnienie 2 kluczy – **prywatnego** i **publicznego**, przy czym klucza prywatnego nie da się łatwo odtworzyć na podstawie publicznego. W niektórych zastosowaniach kluczy może być więcej. Algorytmy mające zastosowanie w kryptografii asymetrycznej wykorzystują operacje jednokierunkowe - takie, które da się łatwo przeprowadzić w jedną stronę a bardzo trudno w drugą. Np. mnożenie ma niską złożoność obliczeniową, a faktoryzacja wysoką (patrz. algorytm RSA [6]). We wszystkich kryptosystemach uzyskanie klucza prywatnego na podstawie publicznego musi być obliczeniowo trudne i praktycznie niemożliwe w sensownym czasie.



Rys 2. Mirek wysyła do Kasi swój klucz publiczny.
Kasia szyfruje wiadomość kluczem publicznym.
Mirek odszyfrowuje wiadomość kluczem prywatny.

1.6.2. Podpis cyfrowy

Podpis cyfrowy (*ang. digital signature*) - jest to algorytmiczny sposób potwierdzania autentyczności cyfrowego dokumentu. Istnieją różne standardy **podpisów cyfrowych** (np. X.509, PGP), obecnie jednak najpopularniejszym jest schemat podpisu dokumentów cyfrowych w systemach kryptograficznych z kluczem publicznym i jednokierunkową **funkcją skrótu** (ang. *hash*) - w systemie tym do oryginalnej wiadomości dołączany jest skrót dokumentu, zaszyfrowany prywatnym kluczem nadawcy. Potwierdzenie autentyczności wiadomości jest możliwe po odszyfrowaniu skrótu kluczem publicznym nadawcy i porównaniu go z obliczonym skrótem odebranego dokumentu. Daje nam to pewność, iż nie została ona zmodyfikowana na którymś z węzłów pośrednich w sieci.

1.6.3. Certyfikat klucza publicznego

Certyfikatem klucza publicznego nazywamy informację o kluczu publicznym podmiotu, która dzięki podpisaniu przez zaufaną trzecią stronę jest niemożliwa do podrobienia. Certyfikat klucza publicznego zawiera trzy podstawowe informacje:

1. klucz publiczny podmiotu,
2. opis tożsamości podmiotu,
3. podpis cyfrowy złożony przez zaufaną trzecią stronę na dwóch powyższych strukturach.

Certyfikat klucza publicznego realizuje przede wszystkim funkcję autentyczności w stosunku do klucza publicznego podmiotu. Autentyczny klucz może być następnie używany do realizacji kolejnych funkcji bezpieczeństwa.

1.6.4. Urząd certyfikacji

Urzędem certyfikacji jest jednostka wydająca elektroniczne certyfikaty tożsamości dla innych osób czy instytucji. Jest przykładem zaufanej trzeciej strony. CA występują w wielu schematach PKI. CA jest zazwyczaj firma, która za pewną opłatą wydaje certyfikat klucza publicznego, który poświadcza tożsamość właściciela klucza. Zadaniem urzędu jest sprawdzenie tożsamości osoby, której wydaje certyfikat kryptograficzny. Uczestnicy transakcji mogą sprawdzić tożsamość właściciela certyfikatu, opierając się na zaufaniu do urzędu, który wydał certyfikat.

1.7. TLS/SSL

TLS (ang. *Transport Layer Security*) – przyjęte jako standard w Internecie rozwinięcie protokołu **SSL** (ang. *Secure Socket Layer*) działający na poziomie warstwy prezentacji i zabezpiecza tym samym warstwę aplikacji. TLS zapewnia poufność i integralność transmisji danych, a także uwierzytelnienie serwera, jak również klienta poprzez system certyfikatów. Opiera się na szyfrowaniu asymetrycznym oraz certyfikatach X.509. SSL nie jest nowym algorytmem szyfrującym. To ustandaryzowany zestaw wcześniej znanych algorytmów, technik i schematów wykorzystywanych do zapewnienia bezpieczeństwa transmisji. SSL jest najczęściej związany z protokołem HTTP jako HTTPS, lecz może także służyć do zabezpieczania innych popularnych protokołów takich jak Telnet, SMTP, POP, IMAP czy FTP, gdyż protokoły te same w sobie nie zapewniają szyfrowania połączenia.

TLS/SSL składa się z następujących pod-protokołów:

1. SSL Handshake – definiuje metody negocjowania parametrów bezpiecznej sesji, czyli algorytmów szyfrowania danych, algorytmów uwierzytelniania i integralności informacji
2. SSL Change Cipher – protokół zmiany specyfikacji szyfru SSL
3. SSL Alert Protocol – protokół alarmowy SSL
4. SSL Record – definiuje format przesyłanych pakietów danych

SSL w wersji trzeciej dopuszcza m.in. następujące algorytmy i protokoły: DES, 3DES, IDEA, RC2, RC4, RSA, DSS, Diffiego-Hellmana. Kluczowym parametrem określającym siłę szyfrowania SSL jest długość użytych kluczy. Im dłuższy klucz, tym trudniej jest go złamać, a przez to odszyfrować transmisję. Dla kluczy asymetrycznych, długością sugerowaną jest obecnie 2048 bitów. Powszechnie używa się kluczy 128-bitowe.

1.7.1. Zasada działania połączenia SSL

1. **Klient → Serwer: ClientHello**
   Klient wysyła do serwera zgłoszenie zawierające m.in. obsługiwaną wersję protokołu SSL, dozwolone sposoby szyfrowania i kompresji danych oraz identyfikator sesji.

Komunikat ten zawiera również liczbę losową używaną następnie przy generowaniu kluczy.

2. **Klient ← Serwer: ServerHello**
   Serwer odpowiada analogiczm komunikatem w którym zwraca klientowi wybrane parametry połączenia: wersję protokołu SSL, rodzaj algorytmu szyfrowania i kompresji, oraz podobną liczbę losową.
3. **Klient ← Serwer: Certificate**
   Serwer wysyła swój certyfikat pozwalając klientowi na sprawdzenie swojej tożsamości (ten etap jest opcjonalny, ale w większości przypadków występuje)
4. **Klient ← Serwer: ServerKeyExchange**
   Serwer wysyła informację o swoim kluczu publicznym. Rodzaj i długość tego klucza jest określony przez typ algorytmu przesłany w poprzednim komunikacie.
5. **Klient ← Serwer: ServerHelloDone**
   Serwer informuje, że klient może przejść do kolejnej fazy zestawiania połączenia.
6. **Klient → Serwer: ClientKeyExchange**
   Klient wysyła serwerowi wstępny klucz sesji, zaszyfrowany za pomocą klucza publicznego serwera. Na podstawie ustalonych w poprzednich komunikatach dwóch liczb losowych oraz ustalonego wstępnego klucza sesji obie strony generują klucz sesji używany do faktycznej wymiany danych. Wygenerowany klucz jest kluczem algorytmu symetrycznego (zazwyczaj DES).
7. **Klient → Serwer ChangeCipherSpec**
   Klient zawiadamia, że serwer może przełączyć się na komunikację szyfrowaną...
8. **Klient → Serwer: Finished**
   ... oraz że jest gotowy do odbierania danych zakodowanych.
9. **Klient ← Serwer: ChangeCipherSpec**
   Serwer zawiadamia, że wykonał polecenie – od tej pory wysyłał będzie tylko zaszyfrowane informacje...
10. **Bezpieczne połączenie zostało nawiązane.**

1.7.2. Uwierzytelnianie klienta

W protokole SSL standard zakłada wyłącznie uwierzytelnianie serwera. Istnieją jednak metody pozwalające na uwierzytelnienie klienta. W tym celu korzysta się z trzech dodatkowych komunikatów:

1. **Klient ← Serwer CertificateRequest**
   Po przesłaniu swojego certyfikatu serwer zawiadamia, że chciałby otrzymać certyfikat klienta.
2. **Klient → Serwer Certificate**
   Po otrzymaniu komunikatu ServerHelloDone klient odsyła swój certyfikat.
3. **Klient → Serwer CertificateVerify**

Klient musi potwierdzić, że faktycznie posiada klucz prywatny odpowiadający wysłanemu certyfikatowi. W tym celu klient podpisuje swoim kluczem prywatnym skrót wszystkich dotychczas ustalonych danych o połączeniu i wysyła go korzystając z tego komunikatu.

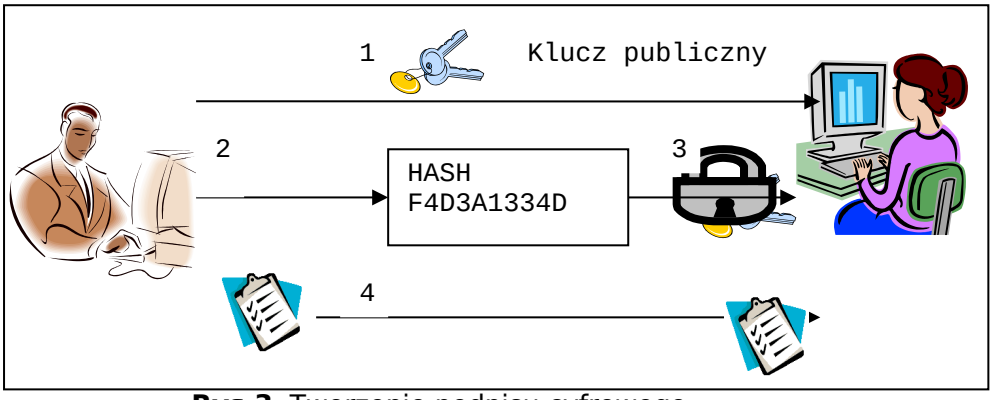
Na podstawie podpisanego skrótu serwer jest w stanie uwierzytelnić klienta.

1.8. Tworzenie i weryfikacja podpisu cyfrowego

I.

Tworzenie podpisu cyfrowego, który zostanie następnie wykorzystany przy podpisywaniu dokumentów, podpisywanie danych) wygląda następująco:

1. Nadawca generuje klucz publiczny i prywatny i wysyła odbiorcy klucz publiczny.
2. Nadawca za pomocą funkcji haszującej oblicza hash.
3. Nadawca szyfruje ten hash korzystając z wygenerowanego przez siebie klucza prywatnego. Zaszyfrowany hash jest podpisem elektronicznym.
4. Nadawca przekazuje odbiorcy podpis cyfrowy wraz z niezaszyfrowaną wiadomością.



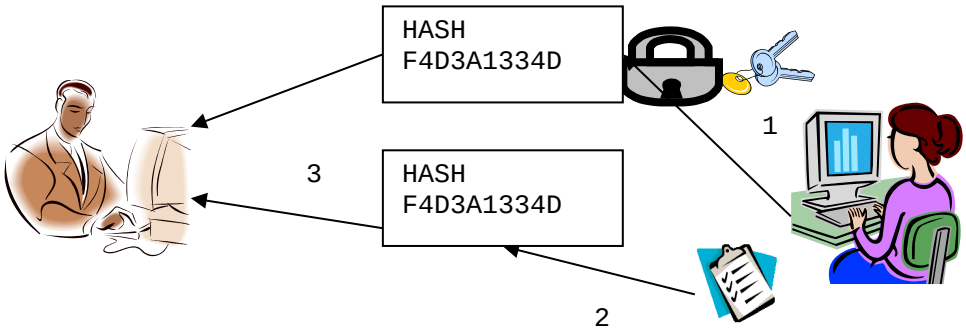
Rys 3. Tworzenie podpisu cyfrowego

II.

Następnie ma miejsce **weryfikacja podpisu cyfrowego**:

1. Odbiorca odszyfrowuje hash korzystając z klucz publicznego nadawcy.
2. Odbiorca z przekazanej mu wiadomości tworzy hash.
3. Odbiorca porównuje oba hasze (skróty).

Jeżeli oba są sobie równe, to mamy zagwarantowane autentyczność, integralność oraz niezaprzeczalność nadawcy.



Rys 3. Weryfikacja podpisu cyfrowego

autentyczność - dane wysłał właściwy nadawca. Podpis cyfrowy gwarantuje, że nikt się nie podszyje pod nadawcę (atak *man in the middle*), bo tylko nadawca dysponuje prywatnym kluczem nadawcy. Owszem ktoś może próbować się podszyć, ale wtedy sygnatura wysłana przez takiego napastnika będzie błędna Odbiorca oczywiście to wykryje, gdyż porównując oba hashe, obliczy, że są różne.

integralność **(**ang. *data integrity*) - dane nie zostały zmienione przez osoby trzecie. Jeżeli ktoś zmieniłby treść wiadomości, to odbiorca haszując taką zmienioną wiadomość otrzyma inny hash niż powinien – porównane hashów będą się różnić, gdyż hashowane były różniące się wiadomości.

niezaprzeczalność (ang. *non-repudiation*) - nadawca nie wyprze się, że wysłał dane - tylko nadawca wiadomości zna swój klucz prywatny. Jeżeli więc weryfikacja podpisu cyfrowego się powiodła, to znaczy, że tylko właściwy nadawca ja wysłał. Dzięki temu podpis cyfrowy jest prawnie równoważny podpisowi fizycznemu.

2. Przebieg laboratorium

Druga część instrukcji zawiera zadania do praktycznej realizacji, które demonstrują zastosowanie technik z omawianego zagadnienia.

2.1. Przygotowanie laboratorium

Do pełnego zrozumienia załączonych programów wymagana jest elementarna znajomość języka C++. Do kompilacji programów należy użyć programu g++ z biblioteką STL bądź dowolnego innego kompilatora tego języka. W kernelach opartych na Debianie pobranie kompilatora następuje poprzez komendę:

```
$ sudo apt-get install g++
```

2.2. Zadanie 1. Proste aplikacje kryptograficzne

2.2.1. Szyfrowanie XOR

W pierwszym zadaniu zostanie przedstawiony w praktyce jeden z najprostszych i najpopularniejszych algorytmów szyfrujących kodujący dane dzięki jednej z własności *alternatywy wykluczającej* (XOR) - przemienności. Tablica prawdy tego działania logicznego wygląda następująco:

P	Q	p XOR q
0	0	0
0	1	1
1	0	1
1	1	0

Operacja ta jest składową tak skomplikowanych algorytmów jak DES. Samodzielnie zastosowana nie gwarantuje wysokiego poziomu bezpieczeństwa, jednak jej zaletą jest łatwość implementacji oraz możliwość odszyfrowania danych za pomocą klucza szyfrującego:

tekst **XOR** klucz = szyfrogram

szyfrogram **XOR** klucz = tekst
(tekst **XOR** klucz) XOR klucz = tekst

Szyfrowanie plików tekstowych na dysku będzie możliwe po zapisaniu znaków ASCII w postaci dwójkowej.

```
std::ifstream plik_wejsciowy(argv[1], std::ios::binary);
std::string klucz = argv[2];
std::stringstream bufor;
unsigned int i = 0;
char znak = plik_wejsciowy.get();

while (plik_wejsciowy.good()) {
    bufor << static_cast<char>(znak ^ klucz[i]);
    i = (i+1) % klucz.length();
    znak = plik_wejsciowy.get();
}
```

W pętli *while* ma miejsce bitowe wykonanie operacji XOR: `znak ^ klucz[i]` na każdym ze znaków z pliku, bit po bicie.
W pliku *xor.cpp* znajduje się kompletny kod źródłowy przykładowej implementacji tego algorytmu zaprogramowanej w C++.
Użycie aplikacji:

```
#g++ xor.cpp –o xor
#xor [plik wejsciowy][klucz szyfrujący][plik wynikowy]
```

Otrzymanie oryginalnego tekstu następuje przez uruchomienie aplikacji XOR na pliku wynikowym przy użyciu tego samego klucza.

Zadania:
Krótkie i zwięzłe odpowiedzi należy umieścić we sprawozdaniu.

1. Czy algorytm zadziała gdy klucz będzie krótszy niż plik wejściowy?
2. Jak oceniasz bezpieczeństwo tego algorytmu?
3. Czy i w jaki sposób można złamać szyfrogram nie dysponując kluczem?
4. W pliku szyfr.cpp znajduje się implementacja jednego z klasycznych i popularnych algorytmów przestawieniowych. Dokonaj analizy funkcji szyfr() napisz na czym polega jego działanie.

2.2.2. Funkcje skrótu MD5/SHA-x

Ważnym elementem certyfikatu jest element wyznaczania funkcji skrótu podpisu. Najczęściej dokonuje się to poprzez algorytm SHA1(160-bit), SHA-2(256-bit) bądź MD5(128-bit). Kryptograficzna funkcja skrótu teoretycznie powinna wykazywać stałą długość i wysoką losowość niezależnie od danych wejściowych, oraz uniemożliwiać otrzymanie oryginalnej wiadomości. Zmiana zaledwie jednego bitu strumienia wejściowego indukuje zmianę niemal całego klucza na zupełnie odmienny. Główną ideą w wykorzystaniu funkcji hashujacych jest porównywanie dwóch ciągów bitów. Jeżeli oba ciągi mają identyczne skróty, to z prawdopodobieństwem bliskim 100% możemy stwierdzić, że zawierają tę samą informację. Obliczenia funkcji skrótu pliku MD5 w systemach unixowych dokonujemy za pomocą polecenia `$md5sum [nazwa_pliku]`, zaś w języku PHP dzięki funkcji `md5($zmienna)`. Istotną własnością funkcji hashującej jest **odporność na kolizje** (szansa na taki sam hash dla 2 różnych wiadomości) i przykładowo dla algorytmu MD5 wynosi $1/2^{64}$, natomiast dla SHA-1 $1/2^{80}$

Zadania:

1. W katalogu z instrukcją znajduje się skrypt md5.php służący do wygenerowania N skrótów dla kolejnych liczb naturalnych i wyszukaniu czy któreś liczby nie mają powtarzających się początkowych znaków. Uruchom skrypt i spróbuj znaleźć dwie liczby mające takie same 3-4 początkowe znaki.
2. Zmień w md5.php funkcję hashującą z md5() na sha1() lub crypt() [algorytm DES] i powtórz czynność z poprzedniego zadania.
3. Wielokrotnie w bazach danych użytkowników wykorzystuje się skróty MD5 do zakodowania haseł użytkowników. Jak oceniasz bezpieczeństwo tak przechowywanych poufnych danych wiedząc, że komputer domowy może przeprowadzając atak brute-force może sprawdzić miliardy haseł w ciągu kilku sekund?
4. Napisz skrypt/program w dowolnym języku łamiący metodą brute-force skrót **e00cf25ad42683b3df678c61f42c6bda** wiedząc, że jest on skrótem MD5 6 - znakowego hasła alfanumerycznego.

2.3. OpenSSL

OpenSSL – oprogramowanie będące implementacją protokołów SSL (2.0, 3.0) i TLS (1.0), oraz algorytmów kryptograficznych ogólnego przeznaczenia. Zawiera zbiór bibliotek implementujących powyższe standardy oraz mechanizmy kryptograficzne, jak i również zestaw narzędzi konsolowych służących do tworzenia kluczy czy certyfikatów, zarządzania certyfikatami i urzędem certyfikacji, (de)szyfrowania, obliczania podpisów cyfrowych i innych). Program nie wymaga instalacji, gdyż jest preinstalowany w systemach GNU/Linux. Podstawowe polecenia programu:

Polecenie	Opis
Ca	Zarządzanie certyfikatami
Dgst	Generowanie skrótów plików
Enc	Szyfrowanie/deszyfrowanie
genrsa	Generowanie kluczy RSA
passwd	Generowanie hashowanych haseł
s_client	Wbudowany klient SSL
s_server	Wbudowany serwer SSL
Speer	Przeprowadzenie benchmarków
Req	Zarządzanie żądaniami certifikatów
Rsa	Zarządzanie kluczami RSA
Rsautl	Narzędzie do podpisywania, szyfrowania, deszyfrowania danych przez RSA
Verify	Weryfikacja i walidacja certyfikacji x509
version	Wyświetlenie wersji programu

W celu wyświetlenia wszystkich poleceń należy włączyć opcję -h. Aby wyświetlić listę wszystkich opcji dla danego polecenia stosujemy analogicznie:

```
$openssl polecenie –h
```

Ogólny schemat korzystania z programu jest następujący:

```
$openssl polecenie [opcje].
```

Zadania.:
Sprawozdanie z laboratorium powinno zawierać listę wykonanych komend realizujących poniższe polecenia oraz odpowiednie wnioski.

1) (De)szyfrowanie plików za pomocą algorytmów symetrycznych. (polecenie enc)
    a. Utwórz plik tekstowy i wypełnij go dowolną treścią.
    b. Zaszyfruj go korzystając z algorytmu AES w trybie CBC z 256-bitowym kluczem.
    c. Spróbuj otworzyć zaszyfrowany plik edytorem tekstu. Czy to możliwe? Dlaczego?

   d. Odszyfruj zaszyfrowaną wersję tekstu do pliku o innej nazwie i sprawdź, czy teksty są takie same.
   e. Jaki fakt podczas wykonywania powyższych poleceń świadczy o tym, że do szyfrowania wykorzystany był algorytm symetryczny?
2) Generowanie skrótów MD5 i SHA1 plików. (polecenie `dgst`)
   a. Utwórz parę plików tekstowych z dowolną zawartością, różniącą się jednym znakiem, i porównaj między sobą wygenerowane dla nich skróty MD5 i SHA1.
3) Czy jest możliwa kolizja skrótów? Do czego może to doprowadzić?

4) Kryptografia asymetryczna. (polecenia `genrsa, rsa, rsautl`)
   a. Wygeneruj parę kluczy RSA o rozmiarze 2048 bitów.
   b. Wyciągnij z uzyskanego pliku klucz publiczny i wyświetl go.
   c. Zaszyfruj plik tekstowy o dowolnej treści otrzymanym kluczem publicznym.
   d. Odszyfruj plik.

5) Zarządzanie certyfikatami. (polecenia ca, req)
   a. Wykonaj żądanie nowego certyfikatu z kluczem RSA o rozmiarze 2048 bitów i zapisz go do pliku.
   b. Wyświetl zawartość utworzonego certyfikatu.
   c. Skonwertuj certyfikat do formatu DER i wyświetl go.
   d. Zweryfikuj certyfikat.
   e. Unieważnij go.

2.4. GnuPG

Bardzo popularną i powszechnie wykorzystywaną aplikacją kryptograficzną jest, wykorzystujący algorytm RSA, GnuPG. Jest to open-source’owa implementacja standardu PGP służącego do szyfrowania plików i poczty e-mail. Domyślnie aplikacja ta jest zainstalowana na systemach Linux. Jeśli jednak tak nie jest, zainstaluj ją za pomocą:

```
$apt-get install gnupg
```

Przed pierwszym uruchomieniem programu GnuPG należy wykonać polecenie tworzące podkatalog tej aplikacji w naszym katalogu domowym.

```
mkdir ~/.gnupg
```

Klucze algorytmów DSA/RSA/ElGamal kreujemy po użyciu komendy:

```
$ gpg --gen-key
```

Postępujemy zgodnie z komunikatami pojawiającymi się na ekranie. Najkorzystniej jest skorzystać z domyślnych opcji oferowanych przez program tj. DSA/2048-bit/nielimitowana ważność klucza.

Uwaga! Istnieje spore prawdopodobieństwo, że program poinformuje, iż brakuje mu entropii losowości do dokończenia czynności tworzenia klucza. W takiej sytuacji

poruszaj myszką, pisz na klawiaturze, korzystaj z sieci oraz I/O HDD i odczekaj kilka chwil.
Hasło należy utworzyć jak najbardziej skomplikowane, gdyż każdy dodatkowy znak zwiększa bezpieczeństwo wykładniczo. Powinniśmy otrzymać komunikat:

```
pub 1024D/630BE83F 2012-05-25 Imie i nazwisko nasz@email.com
Odcisk klucza = 6F92 BCA7 0F6B 2464 909C 1A85 6585 0076 630B E83F
sub 1024g/624564B4 2012-05-25
```

Kluczamy zarządzamy poniższymi komendami:

```
#gpg [--opcje]
```

Parametr/opcja	**Opis**
--list-keys	Lista kluczy
–export fadres@email.pl >/katalog/plik	Eksport klucza gpg do pliku binarnego
--armour --output /tmp/klucz2 --export adres@email.pl	Eksport klucza gpg do pliku tekstowego
--import /tmp/klucz	Import klucza
--edit-key >sign >check >trust	Edycja klucza - podpisanie w skali (0-3) - sprawdzenie podpisów - ustalenie zaufania (1-5)
--output /tmp/wiadomosc.gpg --encrypt /tmp/wiadomosc.txt	Szyfrowanie pliku wiadomosc.txt do formatu binarnego wiadomosc.gpg
--armour --output /tmp/wiadomosc.gpg --encrypt /tmp/wiadomosc.txt	Szyfrowanie pliku wiadomosc.txt do formatu tekstowego wiadomosc.gpg
[--armour] –output /tmp/wiadomosc.sig --sign /tmp/wiadomosc.txt	Podpisywanie cyfrowe pliku [tekstowego].
--decrypt /tmp/wiadomosc2.gpg	Odszyfrowanie i sprawdzenie podpisu cyfrowego
[--armour] --output /tmp/wiadomosc-sym.gpg --symmetric /tmp/wiadomosc.txt	Szyfrowanie symetryczne – bez wykorzystania klucza publicznego
--verify /tmp/wiadomosc.sig	Sprawdzenie podpisu cyfrowego

Zadania:

1) Utwórz klucz publiczny i prywatny. Sprawdź ich stopień zaufania opisany w skali (1-5).
2) Zaszyfruj wiadomość tekstową binarnie i tekstowo.
    a. [Opcjonalnie] Podpisz wiadomość cyfrowo.
3) Utwórz kopię pliku zaszyfrowanego tekstowo, zmień jeden znak i spróbuj go odszyfrować. Umieść w sprawozdaniu informację jaki błąd otrzymał(e/a)ś.
4) Wyślij zaszyfrowaną gpg wiadomość e-mail do drugiego członka zespołu [lub do siebie jeśli niemożliwe]. Możesz to zrobić:
    a. Zaszyfrować symetrycznie, lub
    b. Zaszyfrować asymetrycznie i wymienić się kluczami publicznymi
5) Np. snifferem sprawdź czy można odczytać wiadomość na trasie pośredniej.
6) Odszyfruj wiadomość kolegi [lub swoją].

2.5. Opracowanie i sprawozdanie

Realizacja laboratorium pt. *„Poufność danych oraz PKI: OpenSSL”* polega na wykonaniu wszystkich zadań programistycznych podanych w drugiej części tej instrukcji. Wynikiem wykonania powinno być sprawozdanie w formie wydruku papierowego dostarczonego na kolejne zajęcia licząc od daty laboratorium, kiedy zadania zostały zadane.

Sprawozdanie powinno zawierać:

- opis metodyki realizacji zadań (system operacyjny, język programowania, biblioteki, itp.),
- algorytmy wykorzystane w zadaniach (zwłaszcza, jeśli zastosowane zostały rozwiązania nietypowe),
- opisy napisanych programów wraz z opcjami,
- trudniejsze kawałki kodu, opisane tak, jak w niniejszej instrukcji,
- uwagi oceniające ćwiczenie: trudne/łatwe, nie/realizowalne podczas zajęć, nie/wymagające wcześniejszej znajomości zagadnień (wymienić jakich),
- wskazówki dotyczące ewentualnej poprawy instrukcji celem lepszego zrozumienia sensu oraz treści zadań.

