Podstawy kryptografii

Kryptografia symetryczna

email: contact@cyberskiller.com
cyberskiller.com

Podstawy kryptografii

Kryptografia symetryczna

Spis Treści

email: contact@cyberskiller.com
cyberskiller.com

Wiadomości podstawowe

Celem kryptografii jest zapewnienie poufności informacji lub danych, tak aby dostęp do nich miały tylko osoby lub procesy będące w posiadaniu pewnego sekretu. Sekretem tym jest najczęściej jeden lub więcej kluczy szyfrujących.

Kryptografia symetryczna wykorzystuje symetryczne algorytmy kryptograficzne do szyfrowania i deszyfrowania danych. W najprostszym ujęciu, szyfrowanie to przekształcenie zbioru uporządkowanych danych zwanych tekstem jawnym w inny uporządkowany zbiór danych (tekst zaszyfrowany lub szyfrogram) wykorzystując przy tym tak zwany klucz szyfrujący. Samo przekształcenie określone jest poprzez algorytm szyfrujący. Jeśli przez M oznaczymy tekst jawny, przez C szyfrogram, a przez K klucz szyfrujący, to proces szyfrowania można zapisać symbolicznie w następujący sposób:

$$E(M,K) \rightarrow C$$

Z praktycznego punktu widzenia algorytm szyfrujący jest najczęściej jawny i ogólnie znany. Bezpieczeństwo zaszyfrowanych danych zależy przede wszystkim od bezpieczeństwa klucza i złożoności samego algorytmu szyfrującego. Złożoność algorytmu można zbadać i określić metodami naukowymi. Bezpieczeństwo klucza szyfrującego jest związane z jego długością i tajnością. Przyjmuje się, że im dłuższy i bardziej złożony klucz szyfrujący, tym trudniej złamać dany kryptosystem.

Istotą kryptografii symetrycznej jest fakt, że do szyfrowania i odszyfrowania wykorzystywany jest **`ten sam klucz szyfrujący`**. Proces deszyfrowania można zapisać symbolicznie następująco:

$$D(C,K) \rightarrow M$$

Poniżej przedstawiono wynik szyfrowania zawartości pliku tekstowego z wykorzystaniem algorytmu **`AES-256-ECB`**.

Zawartość pliku **`file.txt`** z tekstem jawnym:

```
00000000  41 6c 61 20 6d 61 20 6b  6f 74 61 2e              |Ala ma kota.|
```

Zawartość plik **`file.enc`** z tekstem zaszyfrowanym:

```
00000000  68 70 76 02 ae 23 02 6b  64 a9 d6 3d 4f 39 45 f9
|hpv.?#.kd??=O9E?|
```

Użyty klucz szyfrujący (256 bitowy) ma postać:

```
87bf78f4fde0cdb4f5f657d3c8faff16ce190b4a81c11431866995d9d1a17bf0
```

Szyfrowanie wykonano wykorzystując następujące polecenie:

```
openssl enc -e -aes-256-ecb -in file.txt -out file.enc -K
87bf78f4fde0cdb4f5f657d3c8faff16ce190b4a81c11431866995d9d1a17bf0
```

Po parametrze **`-K`** pojawia się 256 bitowy klucz szyfrujący.

Dla powyższego przykładu polecenie służące do odszyfrowania wygląda następująco:

```
openssl enc -d -aes-256-ecb -in file.enc -out file_2.txt -K
87bf78f4fde0cdb4f5f657d3c8faff16ce190b4a81c11431866995d9d1a17bf0
```

Zwróćmy uwagę na ponowne wykorzystanie dokładnie tego samego klucza do odszyfrowania.

$$D(E(M,K)) \rightarrow M$$

Szyfrowanie i deszyfrowanie z wykorzystaniem biblioteki openssl

Biblioteka kryptograficzna OpenSSL implementuje szeroki zakres algorytmów kryptograficznych używanych w różnych standardach internetowych. Usługi świadczone przez tę bibliotekę są wykorzystywane przez implementacje TLS i CMS OpenSSL, a także przez wiele innych produktów i protokołów innych firm.

Dostępne funkcjonalności obejmują: szyfrowanie symetryczne, kryptografię klucza publicznego, uzgadnianie kluczy, obsługę certyfikatów, kryptograficzne funkcje skrótu, kryptograficzne generatory liczb pseudolosowych, kody uwierzytelniania wiadomości (MAC), funkcje wyprowadzania kluczy (KDF) i różne narzędzia.

Prymitywy kryptograficzne, takie jak skrót SHA256 czy szyfrowanie AES, są w OpenSSL określane mianem “algorytmów”. Każdy algorytm może mieć wiele implementacji, z których można korzystać.

Różne algorytmy można pogrupować według ich przeznaczenia. Na przykład istnieją algorytmy szyfrowania i różne algorytmy służące do generowania skrótów danych. Te różne grupy są w OpenSSL nazywane “operacjami”. Każda operacja ma inny zestaw funkcji z nią związanych. Dostęp do określonej grupy operacji wymaga podania odpowiedniej komendy. Wybrane komendy wraz z ich krótkim opisem przedstawione zostały poniżej.

Komenda	Opis
dgst	perform digest operations
enc	symmetric cipher routines
genrsa	generate an RSA private key
rsa	RSA key processing tool
rsautl	RSA utility
rand	generate pseudo-random bytes

Pełny opis dotyczący biblioteki OpenSSL można znaleźć pod adresem: https://www.openssl.org.

Przed przystąpieniem do szyfrowania należy dokonać wyboru algorytmu, który zostanie użyty. Listę dostępnych algorytmów można uzyskać za pomocą następującej komendy:

```
openssl enc --list
```

```
-aes-128-cbc
-aes-128-cfb
-aes-128-cfb1
-aes-128-cfb8
-aes-128-ctr
-aes-128-ecb
-aes-128-ofb
-aes-192-cbc
-aes-192-cfb
-aes-192-cfb1
-aes-192-cfb8
-aes-192-ctr
-aes-192-ecb
-aes-192-ofb
-aes-256-cbc
-aes-256-cfb
-aes-256-cfb1
-aes-256-cfb8
-aes-256-ctr
-aes-256-ecb
-aes-256-ofb
-aes128
-aes128-wrap
-aes192
-aes192-wrap
-aes256
-aes256-wrap
-aria-128-cbc
-aria-128-cfb
-aria-128-cfb1
-aria-128-cfb8
-aria-128-ctr
-aria-128-ecb
-aria-128-ofb
-aria-192-cbc
-aria-192-cfb
-aria-192-cfb1
-aria-192-cfb8
-aria-192-ctr
-aria-192-ecb
-aria-192-ofb
-aria-256-cbc
-aria-256-cfb
-aria-256-cfb1
-aria-256-cfb8
-aria-256-ctr
-aria-256-ecb
-aria-256-ofb
-aria128
-aria192
-aria256
-bf
-bf-cbc
-bf-cfb
-bf-ecb
-bf-ofb
-blowfish
-camellia-128-cbc
-camellia-128-cfb
-camellia-128-cfb1
-camellia-128-cfb8
-camellia-128-ctr
-camellia-128-ecb
-camellia-128-ofb
-camellia-192-cbc
-camellia-192-cfb
-camellia-192-cfb1
-camellia-192-cfb8
-camellia-192-ctr
-camellia-192-ecb
-camellia-192-ofb
-camellia-256-cbc
-camellia-256-cfb
-camellia-256-cfb1
-camellia-256-cfb8
-camellia-256-ctr
-camellia-256-ecb
-camellia-256-ofb
-camellia128
-camellia192
-camellia256
-cast
-cast-cbc
-cast5-cbc
-cast5-cfb
-cast5-ecb
-cast5-ofb
-chacha20
-des
-des-cbc
-des-cfb
-des-cfb1
-des-cfb8
-des-ecb
-des-ede
-des-ede-cbc
-des-ede-cfb
-des-ede-ecb
-des-ede-ofb
-des-ede3
-des-ede3-cbc
-des-ede3-cfb
-des-ede3-cfb1
-des-ede3-cfb8
-des-ede3-ecb
-des-ede3-ofb
-des-ofb
-des3
-des3-wrap
-desx
-desx-cbc
-id-aes128-wrap
-id-aes128-wrap-pad
-id-aes192-wrap
-id-aes192-wrap-pad
-id-aes256-wrap
-id-aes256-wrap-pad
-id-smime-alg-CMS3DESwrap
-rc2
-rc2-128
-rc2-40
-rc2-40-cbc
-rc2-64
-rc2-64-cbc
-rc2-cbc
-rc2-cfb
-rc2-ecb
-rc2-ofb
-rc4
-rc4-40
-seed
-seed-cbc
-seed-cfb
-seed-ecb
-seed-ofb
-sm4
-sm4-cbc
-sm4-cfb
-sm4-ctr
-sm4-ecb
-sm4-ofb
```

W przedstawionych przykładach skupimy się na algorytmach z rodziny AES (Advanced Encryption Standard)

AES jest szyfrem blokowym, co oznacza, że sam w sobie nadaje się tylko do bezpiecznego przekształcania kryptograficznego (szyfrowania lub deszyfrowania) jednej grupy bitów o stałej długości, zwanej blokiem. Bloki takie mają rozmiar zazwyczaj od 128 do 256 bitów. Ponieważ tekst jawny może mieć dowolny rozmiar, zachodzi konieczność dzielenia go na bloki danych o wymaganej długości. Jednakże, szyfrowanie wielu bloków za pomocą tego samego klucza szyfrującego znacznie zwiększa podatność na ataki oparte o kryptoanalizę. Stąd powstałą konieczność wprowadzenia tak zwanych trybów działania (ang. mod of operations). Tryb działania opisuje, w jaki sposób wielokrotnie zastosować operację szyfrowania pojedynczego bloku do bezpiecznego przekształcania ilości danych większych niż blok. W większości trybów do każdej operacji szyfrowania wymagany jest unikatowy ciąg binarny, często nazywany wektorem inicjującym (IV). Musi on być unikalny (nie mogący się powtarzać), a w niektórych trybach także losowy. Wektor inicjujący jest używany do zapewnienia, że powstają różne szyfrogramy, nawet jeśli ten sam tekst jawny jest szyfrowany wiele razy niezależnie przy użyciu tego samego klucza. Szyfry blokowe mogą operować na więcej niż jednym rozmiarze bloku, ale podczas transformacji rozmiar bloku jest zawsze stały. Tryby pracy szyfrów blokowych działają na pełnych blokach i wymagają, aby ostatnia, często niepełna, część danych była dopełniania do pełnego bloku. Istnieją jednak tryby, które nie wymagają wypełniania, ponieważ efektywnie wykorzystują szyfr blokowy jako szyfr strumieniowy.

Wybrane tryby pracy:

- Electronic CodeBook (ECB) - tryb elektronicznej książki kodowej. Najprostszy i już nie stosowany tryb szyfrowania. Wiadomość jest dzielona na bloki, a każdy z nich jest szyfrowany osobno.
- Cipher block chaining (CBC) - tryb wiązania bloków. Każdy blok tekstu jawnego przed zaszyfrowaniem jest XOR-owany z poprzednim blokiem szyfrogramu. W ten sposób każdy blok szyfrogramu zależy od wszystkich bloków tekstu jawnego przetworzonych do tego momentu. Aby każda wiadomość była unikalna, w pierwszym bloku musi zostać użyty wektor inicjujący.
- Cipher feedback (CFB) - tryb szyfru ze sprzężeniem zwrotnym. W swojej najprostszej postaci wykorzystuje całe wyjście szyfru blokowego. W tej odmianie jest on bardzo podobny do szyfru CBC i przekształca szyfr blokowy w samosynchronizujący się szyfr strumieniowy.
- Output feedback (OFB) - tryb wyjściowego sprzężenia zwrotnego. Przekształca szyfr blokowy w synchroniczny szyfr strumieniowy. Generuje on bloki strumienia kluczy, które są następnie XOR-owane z blokami tekstu jawnego w celu uzyskania szyfrogramu.

Warto tu wspomnieć że kryptosystem AES jest następcą DES'a (Data Encryption Standard), który po próbach ratowania poprzez standard 3DES ostatecznie porzucony na rzecz silniejszych algorytmów. Standard 3DES to nic innego jak algorytm wykorzystujący trzykrotne szyfrowanie startym algorytmem DES w celu zwiększenia bezpieczeństwa. Biblioteka openssl pozwala na użycie obu standardów w miarę potrzeb. Na liście możliwych do wykorzystania algorytmów szyfrujących mamy dostępne:

- algorytmy oparte na DES:

```
des-cbc             DES in CBC mode
des                 Alias for des-cbc
des-cfb             DES in CFB mode
des-ofb             DES in OFB mode
des-ecb             DES in ECB mode
```

- algorytmy oparte na 3DES

```
des-ede-cbc         Two key triple DES EDE in CBC mode
des-ede             Two key triple DES EDE in ECB mode
des-ede-cfb         Two key triple DES EDE in CFB mode
des-ede-ofb         Two key triple DES EDE in OFB mode
des-ede3-cbc        Three key triple DES EDE in CBC mode
des-ede3            Three key triple DES EDE in ECB mode
des3                Alias for des-ede3-cbc
```

Wiadomości podstawowe

Jak wspomniano wcześniej algorytmy kryptografii symetrycznej wymagają klucza szyfrującego. Jego długość jest zależna od specyfiki algorytmu i dla systemu AES może wynosić 128, 192 lub 256 bitów. Z praktycznych powodów, z punktu widzenia użytkownika o wiele łatwiej zapamiętać jest hasło (nierzadko krótsze niż sam klucz szyfrujący). Biblioteka openssl pozwala na użycie hasła, na podstawie którego generowany jest klucz szyfrujący. Za generowanie klucza odpowiedzialne są algorytmy zwane PBKDF (**P**assword **B**ased **K**ey **D**erivation Function) Należy zwrócić uwagę podczas szyfrowania lud deszyfrowania czy podajemy jawnie klucz czy hasło:

- klucz: **`0cf41548ffb7a1d8ee1679bbc0cf0872`**

```
openssl enc -e -aes-128-ecb -K 0cf41548ffb7a1d8ee1679bbc0cf0872 \
-in file.txt -out file.enc
```

- hasło: **`p2-s7-91-0L`**

```
openssl enc -e -aes-128-ecb -k p2-s7-91-0L -in file.txt -out file.enc
```

Algorytm AES może pracować w kilku trybach: ECB, CBC , … W trybie AES-256-ECB (Electronic Code Book) do szyfrowania wystarczy sam klucz - nie jest potrzebne podawanie wektora inicjującego (IV). Jeśli chcielibyśmy zmienić tryb pracy algorytmu na CBC (Cipher Block Chaining), wówczas należałoby podać także wektor inicjujący w jawnej postaci:

- klucz: **`0cf41548ffb7a1d8ee1679bbc0cf0872`**, wektor inicjujący: **`ac0b8cca6065c7a2`**

```
openssl enc -e -aes-128-cbc -K 0cf41548ffb7a1d8ee1679bbc0cf0872 \
-iv ac0b8cca6065c7a2 -in file.txt -out file.enc
```

Nie musimy podawać jawnie wektora inicjującego jeśli zamiast klucza podamy hasło:

- hasło: **p3-6k-1a-Zb**

```
openssl enc -e -aes-128-cbc -k p3-6k-1a-Zb -in file.txt -out file.enc
```

Funkcja PBKDF

Funkcje **PBKDF** i **PBKDF2** (Password-Based Key Derivation Function 1 i 2) pozwalają na wyodrębnienie (wygenerowanie) klucza szyfrującego z hasła podanego przez użytkownika. Algorytm generujący klucz posiada określony koszt obliczeniowy, po to aby zmniejszyć podatność na ataki typu brute-force lub ataki słownikowe. Koszt obliczeniowy ustalany jest zazwyczaj poprzez określenie ilości iteracji niezbędnych do wygenerowania klucza. Czas generowania klucza, liczony np. w sekundach nie ma znaczenia z punktu widzenia użytkowania w przypadku typowej konieczności użycia. Ma jednak kolosalne znaczenie kiedy podczas próby ataku zachodzi konieczność generowania kluczy z wykorzystaniem np. słowników. Wydłużenie czasu generowania skutecznie spowalnia taki rodzaj ataku czyniąc go nieopłacalnym z uwagi na wykorzystanie zasobów.

W przypadku biblioteki openss; możliwość użycia algorytmów **pbkdf** i **pbkdf2** określa obecność parametru **-pbkdf2**. Domyślna ilość iteracji wynosi 10000. Można ją zmienić wykorzystując parametr **-iter** i podając liczbę iteracji. Należy pamiętać, że wykorzystanie funkcji **pbkdf2** wymaga podania hasła po parametrze **-k**.

Przykład:

```
openssl enc -aes-256-cbc -pbkdf2 -iter 100000 \
-k a9-nP-2b-8H-qgCy -in file.txt > file.enc
```

Szyfrowanie pliku **file.txt** algorytmem **aes-256** w trybie cbc z kluczem generowanym za pomocą funkcji **PBKDF2** z hasła **a9-nP-2b-8H-qgCy** iterując proces generowania klucza 100 000 razy.

Atak typu brute-force

W przypadku kryptografii symetrycznej atak typu “brute-force” polega na sprawdzeniu po kolei wszystkich możliwych kluczy szyfrujących w celu odszyfrowania szyfrogramu, przy założeniu, że wiemy jakiego algorytmu szyfrującego użyto. Zbiór wszystkich możliwe kluczy tworzy tak zwaną przestrzeń kluczy. Jej rozmiar zależy od długości klucza. W przypadku 128 bitowego klucza binarnego przestrzeń kluczy zawiera ich $2^{128}=3.4 \cdot 10^{38}$. Jeśli dysponowalibyśmy komputerem, który jest w stanie w ciągu sekundy dokonać 10^3 operacji deszyfrowania, to przegląd całej przestrzeni kluczy trwałby ponad 10^{28} lat, co stanowi niewyobrażalny czas oczekiwania. Należy jednak pamiętać, że ze 100% pewnością jeden z kluczy jest dokładnie tym, którego użyto do zaszyfrowania danych.

Dodatek

Algorytmy szyfrujące wykorzystywane w OpenSSL (dla komendy enc):

```
base64                  Base 64

 bf-cbc                 Blowfish in CBC mode
 bf                     Alias for bf-cbc
 blowfish               Alias for bf-cbc
 bf-cfb                 Blowfish in CFB mode
 bf-ecb                 Blowfish in ECB mode
 bf-ofb                 Blowfish in OFB mode

 cast-cbc               CAST in CBC mode
 cast                   Alias for cast-cbc
 cast5-cbc              CAST5 in CBC mode
 cast5-cfb              CAST5 in CFB mode
 cast5-ecb              CAST5 in ECB mode
 cast5-ofb              CAST5 in OFB mode

 chacha20               ChaCha20 algorithm

 des-cbc                DES in CBC mode
 des                    Alias for des-cbc
 des-cfb                DES in CFB mode
 des-ofb                DES in OFB mode
 des-ecb                DES in ECB mode

 des-ede-cbc            Two key triple DES EDE in CBC mode
 des-ede                Two key triple DES EDE in ECB mode
 des-ede-cfb            Two key triple DES EDE in CFB mode
 des-ede-ofb            Two key triple DES EDE in OFB mode

 des-ede3-cbc           Three key triple DES EDE in CBC mode
 des-ede3               Three key triple DES EDE in ECB mode
 des3                   Alias for des-ede3-cbc
 des-ede3-cfb           Three key triple DES EDE CFB mode
 des-ede3-ofb           Three key triple DES EDE in OFB mode

 desx                   DESX algorithm.

 gost89                 GOST 28147-89 in CFB mode (provided by ccgost engine)
 gost89-cnt            `GOST 28147-89 in CNT mode (provided by ccgost engine)

 idea-cbc               IDEA algorithm in CBC mode
 idea                   same as idea-cbc
 idea-cfb               IDEA in CFB mode
 idea-ecb               IDEA in ECB mode
 idea-ofb               IDEA in OFB mode
```


email: contact@cyberskiller.com
cyberskiller.com

```
rc2-cbc             128 bit RC2 in CBC mode
rc2                 Alias for rc2-cbc
rc2-cfb             128 bit RC2 in CFB mode
rc2-ecb             128 bit RC2 in ECB mode
rc2-ofb             128 bit RC2 in OFB mode
rc2-64-cbc          64 bit RC2 in CBC mode
rc2-40-cbc          40 bit RC2 in CBC mode

rc4                 128 bit RC4
rc4-64              64 bit RC4
rc4-40              40 bit RC4

rc5-cbc             RC5 cipher in CBC mode
rc5                 Alias for rc5-cbc
rc5-cfb             RC5 cipher in CFB mode
rc5-ecb             RC5 cipher in ECB mode
rc5-ofb             RC5 cipher in OFB mode

seed-cbc            SEED cipher in CBC mode
seed                Alias for seed-cbc
seed-cfb            SEED cipher in CFB mode
seed-ecb            SEED cipher in ECB mode
seed-ofb            SEED cipher in OFB mode

sm4-cbc             SM4 cipher in CBC mode
sm4                 Alias for sm4-cbc
sm4-cfb             SM4 cipher in CFB mode
sm4-ctr             SM4 cipher in CTR mode
sm4-ecb             SM4 cipher in ECB mode
sm4-ofb             SM4 cipher in OFB mode

aes-[128|192|256]-cbc  128/192/256 bit AES in CBC mode
aes[128|192|256]       Alias for aes-[128|192|256]-cbc
aes-[128|192|256]-cfb  128/192/256 bit AES in 128 bit CFB mode
aes-[128|192|256]-cfb1 128/192/256 bit AES in 1 bit CFB mode
aes-[128|192|256]-cfb8 128/192/256 bit AES in 8 bit CFB mode
aes-[128|192|256]-ctr  128/192/256 bit AES in CTR mode
aes-[128|192|256]-ecb  128/192/256 bit AES in ECB mode
aes-[128|192|256]-ofb  128/192/256 bit AES in OFB mode

aria-[128|192|256]-cbc  128/192/256 bit ARIA in CBC mode
aria[128|192|256]       Alias for aria-[128|192|256]-cbc
aria-[128|192|256]-cfb  128/192/256 bit ARIA in 128 bit CFB mode
aria-[128|192|256]-cfb1 128/192/256 bit ARIA in 1 bit CFB mode
aria-[128|192|256]-cfb8 128/192/256 bit ARIA in 8 bit CFB mode
aria-[128|192|256]-ctr  128/192/256 bit ARIA in CTR mode
aria-[128|192|256]-ecb  128/192/256 bit ARIA in ECB mode
aria-[128|192|256]-ofb  128/192/256 bit ARIA in OFB mode
```

```
camellia-[128|192|256]-cbc  128/192/256 bit Camellia in CBC mode
camellia[128|192|256]       Alias for camellia-[128|192|256]-cbc
camellia-[128|192|256]-cfb  128/192/256 bit Camellia in 128 bit CFB mode
camellia-[128|192|256]-cfb1 128/192/256 bit Camellia in 1 bit CFB mode
camellia-[128|192|256]-cfb8 128/192/256 bit Camellia in 8 bit CFB mode
camellia-[128|192|256]-ctr  128/192/256 bit Camellia in CTR mode
camellia-[128|192|256]-ecb  128/192/256 bit Camellia in ECB mode
camellia-[128|192|256]-ofb  128/192/256 bit Camellia in OFB mode
```