Laboratorul 12 - Protocoale de securitate. OpenSSL CLI tools

Responsabili: Cătălin Leordeanu, Mihai Dumitru, Andrei Stanca

Obiective

Scopul acestui laborator este sa va familiarizeze cu notiunile de securitate si uneltele folosite in dezvoltarea aplicatiilor.

Criptare

Criptarea este un mecanism folosit pentru a asculta informatia, care poate oferi numeroase asigurari pentru emitator si receptor, printre care:

- **Confidentialitate** - nimeni altcineva nu poate citi datele
- **Integritate** - nimeni altcineva nu poate modifica datele fara sa fie detectat
- **Autenticitate** - asigurarea ca interlocutorul este cine sustine ca este

Acordul in criptografia moderna este ca algoritmii de encriptie/decriptie trebuie sa fie cunoscuti si aplicarea lor sa depinda de un fragment de date, numit **cheie** - singura parte care trebuie tinuta secreta. Acesta e cunoscut ca principiul lui Kerckhoffs.

Prin aplicarea algoritmului de encriptie asupra unui mesaj si a unei chei se obtine un cifru. Analog, prin aplicarea algoritmului de decriptie asupra unui cifru si al unei chei se recupereaza un mesaj.

- E(M, K) = C
- D(C, K) = M

Criptare Simetrica

Criptarea simetrica implica folosirea unei singure chei atat pentru criptarea cat si pentru decriptarea datelor. Astfel, pentru orice mesaj M si pentru orice cheie K, trebuie sa se respecte urmatoarea egalitate:

D(E(M, K), K) = M

Presupunem ca Alice si Bob au aceeasi cheie secreta K, iar Alice vrea sa-i trimita mesajul M lui Bob:

1. Alice foloseste algoritmul de encriptie pentru a obtine un cifru C = E(M, K)
2. Alice trimite C catre Bob. Oricine intercepteaza acest mesaj, nu poate obtine mesajul original M
3. Bob primeste C si aplica algoritmul de decriptie pentru a recupera mesajul original M = D(C, K)

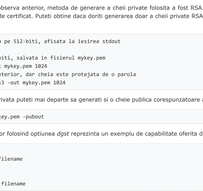

Figure 1: Mecanismul de criptare simetrica

Exemple de algoritmi de criptare simetrica:

AES, DES, 3DES, IDEA, Blowfish

In cazul criptarii simetrice dificultatea provine din necesitatea existentei unui mecanism de secretizare pentru distribuirea cheii secrete. In plus, nu este facuta ca orice pereche de entitati care doresc sa comunice criptat sa impartaseasca un secret (e.g.google ar trebui sa aiba cate o cheie pentru fiecare client). Exista protocoale de schimbare de chei (key exchange), astfel incat doua entitati sa poata alege impreuna o cheie privata peste un canal de comunicare nesigur, astfel incat nimeni altcineva care poate vedea informatia transmisa sa nu poata determina cheia (si, deci, nici mesajele ulterioare, criptate cu cheia aleasa). Cel mai cunoscut astfel de protocol este Diffie!Hellman. Acesta este totusi nesigur in cazul atacurilor de tipul Man-in-the-Middle cand atacatorul poate intercepta mesaje si produce proprille mesaje.

Criptare Asimetrica

Criptarea asimetrica presupune folosirea unei perechi de chei: una pentru encriptie, cealalta pentru decriptie. Ambii participanti la trafic au cate o pereche de chei. **Cheia de encriptie** este o cheie publica, absolut oricine o poate cunoaste si o poate folosi pentru a encripta un mesaj. **Cheie de decriptie** este o cheie privata (secreta), cunoscuta doar de proprietarul ei, acesta putand sa o foloseasca pentru a decripta mesaje encriptate cu cheia sa publica.

Astfel, pentru orice pereche de chei (*P, S*) si orice mesaj M, trebuie sa se respecte urmatoarea egalitate:

- D(E(M, P), S) = M

In unele sisteme de criptare asimetrica, este posibila si encriptarea cu cheie secreta, decriptarea cu cheie publica:

- D(E(M, S), P) = M

Presupunem ca Alice vrea sa-i trimita lui Bob mesajul M:

1. Alice obtine cheia publica PB a lui Bob (e.g. o cere explicit printr-un mesaj)
2. Alice calculeaza un cifru C = E(M, PB)
3. Alice trimite C catre Bob.
4. Bob primeste C si aplica algoritmul de decriptie pe cifrul primit si pe cheia sa secreta pentru a recupera mesajul original M = D(C, SB)

Deoarece Bob e singura persoana care cunoaste SB, este singurul care poate decripta C. Nici macar Alice nu mai poate recupera mesajul original din C. Similar, daca Bob doreste sa trimita un raspuns, el trebuie sa obtina cheia publica a lui Alice.

Un algoritm de criptare asimetrica larg utilizat este RSA.

Pentru sistemele de criptare asimetrica, exista problema: cum putem fi siguri ca cheia publica primita chiar apartine cui credem ca apartine? (un atacator ar putea folosi un atac Man-in-the-Middle pentru a ne livra propria sa cheie).

Rezumate de mesaje

Rezumatul (hash) unui mesaj este un sir de biti de lungime fixa, generat cu ajutorul unei functii de dispersie neinversabile aplicata mesajului. Functia de dispersie H trebuie sa aiba urmatoarele proprietati:

1. Dandu-se un mesaj M, este usor de calculat H(M)
2. Dandu-se un mesajul H(M), este greu de calculat M
3. Dandu-se un mesaj M, este greu de gasit M0 astfel incat H(M) = H(M0)
4. O schimbare mica in mesaj (chiar si de 1 bit) produce un rezumat foarte diferit

Rezumatele pot fi utilizate pentru a verifica rapid transmisia corecta a unui mesaj (rezumatul este transmis impreuna cu mesajul si destinatarul verifica daca rezumatul primit coincide cu rezumatul recalculat de catre el) si pentru realizarea semnaturilor digitale.

Exemple de algoritmi pentru calculul de rezumate: MD5, SHA-1, SHA-2, SHA-3

Semnaturi digitale

Semnaturile digitale asigura autenticitatea mesajelor, verificarea semnaturilor oferind garantiile:

- **Autentificare** - mesajul provine de la sursa pretinsa si nu a fost falsificat de altcineva
- **Integritate** - mesajul nu a fost alterat de altcineva, ci este asa cum a fost scris de sursa
- **Non-repudiere** - entitatea care a semnat mesajul nu poate nega ulterior semnarea acestuia

Pentru un sistem foarte simplu de semnatura digitala, sa consideram exemplul in care Alice doreste sa-i trimita lui Bob un mesaj M, folosindu-se de un sistem de criptare asimetrica si o functie de hashing.

1. Alice calculeaza rezumatul mesajului R = H(M)
2. Alice cripteaza acest rezumat cu cheia secreta obtinand un cifru C = E(R, SA)
3. Alice trimite perechea (M, C) catre Bob
4. Bob obtine cheia publica PA a lui Alice
5. Bob primeste perechea (M, C)
6. Bob calculeaza propriul rezumat R = H(M)
7. Bob decripteaza C folosind cheia publica RO = D(C, PA)
8. Daca R = R0, Bob are garantia ca mesajul a fost semnat de Alice si nimeni nu l-a modificat

Gestiunea cheilor publice

Pentru ca doua entitati sa poata comunica sigur utilizand sistemul de criptare asimetrica, fiecare trebuie sa cunoasca cheia publica a celeilalte si sa nu existe riscul ca un intrus sa substituie o cheia publica cu propria sa cheia publica. Una dintre solutiile utilizate la ora actuala pentru aceasta problema este certificarea cheilor de catre organizatii speciale numite autoritati de certificare (Certification Authority - CA).

O entitate care doreste un certificat trebuie sa se adreseze unei CA, autentificandu-se si furnizand cheia sa publica; autoritatea de certificare poate decide sa acorde persoanei certificatul, care va contine identitatea si cheia publica a solicitantului. Certificatul este semnat digital de catre autoritatea de certificare. Formatul utilizat de obicei pentru certificate este X.509.

Autoritatile de certificare sunt organizate ierarhic, existand o serie de CA-uri "radacina" care sunt bine cunoscute, alta serie de CA-uri certificate de CA-urile radacina s.a.m.d. In momentul in care este verificat certificatul unei entitati se verifica si autoritatea de certificare CA1 care i-a emis, si care are si ea un certificat de la o alta autoritate CA2; apoi se verifica CA2 si asa mai departe pana se ajunge la o CA in care se poate avea incredere sau la o CA radacina (astfel se formeaza un "lant de incredere" sau o "cale de certificare"). CA-urile radacina au certificate auto-semnate.

Informatii despre baza larilului de incredere sunt incluse in aplicatii (mail client, web browser etc.), sau in sistemul de operare, care serveste aplicatiile interesate.

OpenSSL

OpenSSL este o implementare open-source a protocoalelor SSL si TLS. Pe langa API-uri pentru diverse limbaje de programare, aceasta ofera si un utilitar CLI, openssl. Sintaxa pentru utilizarea in linia de comanda este urmatoarea:

```
openssl comanda [optiuni] [argumente]
```

Printre comenzi se numara:

- **ca** - utilizata pentru managementul unei autoritati de certificare (se pot genera certificate, care sunt stocate apoi intr-i baza de date)
- **dgst** - pentru calculul de rezumate de mesaje
- **genrsa** - pentru generarea de chei RSA
- **req** - pentru crearea si procesarea de cereri de certificate; se poate utiliza si pentru generarea de certificate auto-semnate
- **verify** - pentru verificarea de certificate X.509

Pentru inceput, emiteti urmatoarea comanda pentru a verifica versiunea de OpenSSL pe care o folositi:

```
openssl version
```

Generarea cererii de certificat se realizeaza astfel. Exemplul urmator produce un fisier **mycert.pem** ce contine atat cheia privata, cat si cea publica. Certificatul va fi valid timp de 365 de zile iar cheia este neencriptata (optiunea -nodes).

```
openssl req \
-new -newkey rsa:1024 -nodes \
-keyout mykey.pem -out myreq.pem
```

Dupa apelul comenzii veti fi pusi sa raspundeti unei serii de intrebari legate de: Tara, Stat, Oras, etc.

Rezultatul va consta in crearea a doua fisiere: **mykey.pem** va contine cheia privata, iar **myreq.pem** va contine o cerere de certificat. Cererea de certificat este trimisa (pe canale sigure) unei autoritati de semnare (de exemplu VeriSign). Puteti verifica continutul informatiilor continute in cererea de certificat folosind:

```
# verificarea semnaturii
openssl req -in myreq.pem -noout -verify -key mykey.pem
# verificarea informatiilor
openssl req -in myreq.pem -noout -text
```

Dupa cum ati putut observa anterior, metoda de generare a cheii private folosita a fost RSA. Metoda folosita insa realizaza si o cerere de certificat. Puteti obtine daca doriti generarea doar a cheii private RSA emitand o comanda precum:

```
# o cheie implicitia pe 512-biti, afisata la iesirea stdout
openssl genrsa
# o cheie pe 1024-biti, salvata in fisierul mykey.pem
openssl genrsa -out mykey.pem 1024
# cu in exemplul anterior, dar cheia este protejata de o parola
openssl genrsa -des3 -out mykey.pem 1024
```

Pornind de la cheia privata puteti mai departe sa generati si o cheie publica corespunzatoare astfel:

```
openssl rsa -in mykey.pem -pubout
```

Generarea digest-urilor folosind optiunea *dgst* reprezinta un exemplu de capabilitate oferita de OpenSSL.

```
# MD5 digest
openssl dgst -md5 filename

# SHA1 digest
openssl dgst -sha1 filename
```

Digesturile MD5 sunt similare celor create cu comanda *md5sum*, desi formatele de iesire difera.

```
$ openssl dgst -md5 foo-2.23.tar.gz
MD5(foo-2.23.tar.gz)= 81eda7985e95d28acd6d286aa0e13e07

$ md5sum foo-2.23.tar.gz
81eda7985e95d28acd6d286aa0e13e07  foo-2.23.tar.gz
```

Digesturile pot fi chiar semnate pentru a va asigura ca ele nu pot fi modificate fara permisiunea explicita a proprietarului.

```
# digestul semnat va fi foo-1.23.tar.gz.sha1
openssl dgst -sha1 -sign mykey.pem -out foo-1.23.tar.gz.sha1 foo-1.23.tar.gz
```

Ulterior, digestul poate fi verificat. Pentru aceasta aveti nevoie de fisierul din care digestul a fost generat, de digest, si de cheia publica a semnatarului.

```
# pentru verificarea arhivei foo-1.23.tar.gz folosind foo-1.23.tar.gz.sha1 si cheia publica
# pubkey.pem
openssl dgst -sha1 -verify pubkey.pem -signature foo-1.23.tar.gz.sha1 foo-1.23.tar.gz
```

Un alt exemplu de folosire a OpenSSL este si cel legat de criptarea/decriptarea unor documente. Astfel, pentru criptare puteti emite o comanda precum:

```
# cripteaza file.txt la file.enc folosind 256-bit AES in modul CBC
openssl enc -aes-256-cbc -salt -in file.txt -out file.enc

# acelasi lucru, dar iesirea este de data aceasta codata base64. de exemplu pentru email
openssl enc -aes-256-cbc -a -salt -in file.txt -out file.enc
```

In cadrul exemplului a fost folosit unul dintre algoritmii de criptografie suportati de OpenSSL. In practica insa puteti alege oricare dintre cei suportati. Pentru a vedea care sunt algoritmii inclusi in distributia OpenSSL pe care o folositi puteti emite:

```
openssl list-cipher-commands
```

Decriptarea fisierului rezultat anterior poate fi realizata astfel:

```
# decriptarea fisierului binar file.enc
openssl enc -d -aes-256-cbc -in file.enc

# decriptarea versiunii base64
openssl enc -d -aes-256-cbc -a -in file.enc
```

Cerinte laborator

Pentru analiza traficului, va recomandam sa folositi wireshark.

1. Obtineti certificatul de la google.com si identificati urmatoarele campuri:

   - tipul cifrului folosit
   - entitatea care a eliberat certificatul
   - intervalul de timp in care certificatul e valid
   - cheia publica

2. Porniti o unealta de monitorizare a traficului si logati-va pe o pagina prin http, apoi prin https (nu este nevoie de un login legitim, puteti introduce orice in campurile de credentiale). Analizati informatia disponibila in captura de trafic.

Exemple de pagini http cu formular de login:

   - http://login.onlineplanservice.com/Login.aspx?ReturnUrl=%2f
   - http://icm.org/
   - http://login.hotpotatoes.net/

3. Pentru login prin https, analizati portiunea de TLS handshake si identificati urmatoarele campuri:

   - versiunea de TLS folosita
   - suitele de cifruri (cipher suits) suportate de client
   - suita aleasa de server

4. Porniti o unealta de monitorizare a traficului si obtineti acces la un shell logandu-va prin telnet, apoi prin SSH; dati niste comenzi de shell. Analizati informatia disponibila in captura de trafic.