Examen

—

Securitatea Sistemelor Informatice

Theodor-Pierre Moroianu – 334

January 26, 2022

Exercitiul 1

A

Rulam urmatorul script:

```
C = 0x253505ba
K = 0x717056ee
M = C ^ K
mesaj = M.to_bytes(4, 'big')

print(''.join([chr(i) for i in mesaj]))
# > TEST
```

Asadar, afirmatia este FALSA. Folosind OTP, textul criptat reprezinta in clar mesajul **"TEST"**.

B

Stim ca daca un sistem este CCA-sigur, atunci este automat CPA sigur.
Orice sistem determinist nu este CPA-sigur (putem sa cerem oracolului de decriptare sa il decripteze pe m_1 si m_2), deci afirmatia este ADEVARATA.

C

Un atac activ este un atac in care atacatorul intervine fizic (spre diferenta unui atac pasiv, in care atacatorul doar asculta ce se intampla). Asadar, un atac Man-in-the-Middle este activ, si afirmatia este ADEVARATA.

D

Afirmatia este FALSA. Prin permutare random, se refera la faptul ca functia este o auto-bijectie peste multimea $\{0,1\}^4$, nu ca permuta bitii unui numar (in acelasi mod in care o permutare a numerelor de la 0 la 99 nu reprezinta o permutare a celor doua cifre a fiecarui numar).

Un PRP presupune ca pentru fiecare intrare **iesirea pare aleatoare pentru oricine nu stie cheia de encriptare, si ia valori distincte (este bijectiva)**.

E

RSA este un algoritm foarte lent, folosit in principal pentru schimb de chei sau semnaturi digitale. Asadar, nu este recomandata transmiterea de fisiere doar cu RSA. Este recomandata schimbarea unei chei publice cu RSA si dupa-aceea folosirea unui algoritm simetric cum ar fi AES. Afirmatia este FALSA.
Este recomandat sa se foloseasca **RSA combinat cu o metroda de encodare simetrica, intr-o schema hibrida** pentru transmiterea fisierelor in mod criptat.

F

Salvarea unui hash pe langa fiecare fisier personal ofera integritatea datelor impotriva unor modificari datorite unor defectiuni hardware, dar nu ofera nicio securitate impotriva unui atacator activ, care poate altera un fisier si inlocui hash-ul acestuia cu noul hash. Asadar, afirmatia este FALSA.

Pentru a asigura integriatea unor fisiere personale, este suficient sa stocati pe calculatorul propriu fisierele si **pe un dispozitiv securizat – ideal mai multe dispozitive, neconectate la retea, hashurile fisierelor. Alternativ, se pot salva pe calculator hasurile fisierelor, encodate cu o cheie care este tinuta pe un dispozitiv securizat / scrisa pe o foaie.**

G

Hash-ul SHA256 al cuvantului "PAROLA" este *467b4a3eca61a4e6 2447400d93fc35d4295c08ffa2b04 ae942f4de03fa62f464*. Asadar, afirmatia este FALSA.

Folosind website-ul *https://hashes.com/en/decrypt/hash*, putem verifica ce cuvant ne-a dat hashul dat in enunt: "PAR123".

SHA256(PAROLA)=**0x467b4a3eca61a4e6 2447400d93fc35d42 95c08ffa2b04 ae942f4de 03fa62f464**.

H

Un atac impotriva algoritmului de schimb de chei Diffie-Hellman poate fi efectuat prin rezolvarea problemei logaritmului discret, dar rezolvarea acestei probleme este foarte grea, si nu constituie scopul principal al atacului (este doar una din posibilele metode de-a afla cheia generata prin Diffie-Hellman), asadar afirmatia este FALSA.

Scopul principal al unui adversar impotriva schimului de chei Diffie-Hellman este **sa afle care este cheia generata de cele doua entitati care urmeaza sa fie folosita in encriptarea simetrica a datelor**.

I

Afirmatia este ADEVARATA. Folosirea exponentului 65537 nu este o problema. Din punct de vedere teoretic, nu exista nicio restrictie asupra exponentului e, infara de cerinta ca acesta sa fie prim cu p si q, cei doi factori ai modulului, si suficient de mare ca m^e sa fie considerabil mai mare ca N. In practica, 65537 este cel mai folosit folosit exponent de RSA, atat din motive istorice cat si de performanta ($65537 = 2^{16} + 1$, deci exponentierea poate fi efectuata prin 17 inmultiri).

J

Afirmatia este ADEVARATA. TLS foloseste mai multe protocoale cryptografice, cum ar fi un schimb de chei printr-o comunicare pe baza de chei asimetrice, o encriptare a traficului cu o cheie simetrica si generarea unui digest, pentru a oferi autenticitate, confidentialitate si integritate.

Exercitiul 2

A

Un principiu care este satisfacut este principiul diversitatii.

Conform definitiei, principul diversitatii spune ca consta in "Use different types of cryptographic algorithms", pentru a evita un singur atac impotriva intregului sistem.
Putem observa ca aplicatia web foloseste:

- Stocarea parolelor sub forma de hash cu un salt, ceea ce ofera (cel putin in principiu) confidentialitatea parolelor.
- O conectiune TLS intre server si client, care ofera confidentialitatea si integritatea datelor trimise.
- Stocarea locala a fisierelor prin criptarea cu AES-ECB (care din pacate nu este perfect sigur, permite unui atacator sa observe corelatii intre blocurile criptate).
- Foloseste (sau cel putin incearca sa foloseasca) un sistem MAC pentru a oferi integritatea mesajelor in sistemul lor end-to-end.
- Sanitizeaza formul de login, pentru a evita atacuri de tipul SQL-injection.

Asadar, prin multitudinea abordarilor si algoritmilor criptografici folositi, aplicatia respecta principiul diversitatii.

B

Un principiu care nu este satisfacut de aplicatie este principiul lui Kerckhoff, care afirma ca singurele informatii ascunse trebuie sa fie cheile folosite.
Cum functia H de hash folosita pentru stocarea parolelor este o functie hash proprietara, aceasta incalca principiul lui Kerckhoff. Aceasta abordare nu este sigura, pentru ca nu poate garanta folosirea unei functii hash sigure (mult mai putine persoane o pot verifica), si leakuirea acesteia poate conduce la probleme grave de securitate si de confidentialitate a parolelor.
Alt principiu incalcat este Security By Design, avand in vedere ca website-ul nu ofera validare pentru toate informatiile primite (accepta preturi negative, introducerea unei date din trecut etc).

C

Atat confidentialitatea cat si integritatea la nivel de aplicatie sunt compromise.

Integritatea este compromisa prin sistemul de integritate end-to-end, care foloseste o functie CRC pentru a asigura integritatea datelor trimise.
CRC este folosit in retelistica pentru a asigura ne-alterarea din defecte hardware a pachetelor trimise, dar nu ofera nicio securitate impotriva unui atac activ. Spre diferenta folosirii unui algoritm MAC, CRC-ul nu necesita nicio cheie, si deci atacatorul poate sa schimbe mesajul trimis, si sa recalculeze CRC-ul corespunzator.

Confidentialitatea datelor este compromisa prin folosirea algoritmului de criptare AES-ECB, care nu este sigur din punct de vedere semantic. Acest lucru este explicat atat la laborator cat si pe pagina de documentatie al unei implementari ale algoritmului:
https://pycryptodome.readthedocs.io/en/latest/src/cipher/classic.html#ecb-mode.
Confidentialitatea este si compromisa prin sistemul naiv folosit pentru resetarea parolelor, care permite unui atacator sa impersoneze cu usurinta alt utilizator.

D

Un atac foarte simplu impotriva sistemului de recuperare a parolei, care permite unui atacator sa impersoneze orice utilizator, este urmatorul:

1. Atacatorul afla username-ul persoanei pe care doreste sa o impersoneze.
2. Atacatorul foloseste functionalitatea website-ului de recuperare a parolei, folosind usernamul gasit, si cere website-ului sa genereze si sa trimita pe email linkul de resetare a parolei.
3. Pentru a gasi linkul de resetare a parolei, atacatorul poate folosi PRNG-ul cunoscut, folosind ca seed username-ul si ziua curenta (ambele cunoscute). Poate asadar initializa acelasi PRNG ca cel folosit pentru generarea linkului de resetare, cu acelasi seed. Obtine deci acelasi link.
4. Dupa ce a generat link-ul cu ajutorul PRNG-ului compromis, atacatorul poate folosi linkul pentru a reseta parola.
5. Atacatorul se logheaza pe contul utilizatorului cu noua parola.

Exercitiul 3

A

Algoritmul RSA functioneaza pe baza urmatoarei ecuatii:

$$(a^d)^e \equiv (a^e)^d \equiv a \pmod{N}$$

Asadar, pentru a obtine pe m' inapoi din σ, este suficient sa folosim egalitatea $m' \equiv \sigma^e$.

Dupa ce a fost gasit m', dorim sa verificam daca poate fi scris conform regulii de mai sus:

$$m' = 0^8||0^71||FF^x||0^8||m$$

Asadar, algoritmul de verificare a unei semnaturi este $Verif(N, e, m, \sigma)$:

1. Calculam $m' \equiv \sigma^e \pmod{N}$.
2. Verificam daca scrierea in baza 2 al lui m' este de forma urmatoare:
   - Cei $|m|$ cei mai nesemnificativi biti sunt egali cu m.
   - Urmatorii 8 cei mai nesemnificativi biti sunt 0.
   - Urmatorii $8k + 1$ biti sunt 1, pentru un k intreg pozitiv.
   - Toti ceilalti biti sunt 0.
3. Daca pasul 2. a functionat, atunci acceptam semnatura, daca nu o refuzam.

Functia primeste ca input cheia publica, mesajul plain text, si semnatura acestuia.
Concert, primeste:

- Cheia publica (N si e).
- Mesajul pe care dorim sa il verificam (m).
- Mesajul semnat cu cheia privata (σ).

Outputul verificatorului este un boolean DA/NU.

B

Fie $m_2 = 2 * m$, si m_2' mesajul cu padding corespunzator lui m_2.
Uitandu-ne la reprezentarea in baza 2, m_2 arata identic ca m, cu exceptia unui bit de 0 adaugat la sfarsit. Asadar, si m_2' va fi identic cu m' cu exceptia unui bit de 0 in plus la sfarsit.

Asadar, avem:

$$m_2' = 2 * m'$$

Fie σ_2 semnatura lui m_2. Conform definitiei, σ_2 va fi:

$$\sigma_2 \equiv m_2'^d \equiv (2 * m')^d \equiv 2^d * m'^d \equiv 2^d * \sigma$$

Asadar, $sign(m', sk) = 2^d * \sigma$, deci perechea $(2m, 2^d\sigma)$ este o semnatura valida.

Desi poate parea un risc de securitate, generarea a astfel de semnaturi, plecand de la o alta semnatura valida, necesita cunoasterea cheii private sk, mai precis de d. Cum acesta nu este facut public, aceasta metoda nu poate fi folosita pentru a semna mesaje fara a cunoaste cheia privata.

C

Vulnerabilitatea acestei extensii este ca mesajele m, $m + N$, ..., $m + k * N$ se vor reduce toate la aceeasi semnatura, toate numerele fiind reduse prin **PAS 0.** la $m \ mod \ N$. Concret, adaugarea sau scaderea din m al lui N nu schimba validitatea semnaturii.

Un atac posibil este:

1. Atacatorul intercepteaza un mesaj (m, σ) cu o semnatura valida.
2. Atacatorul modifica m, scazand sau adaugand multiplii de N, obtinand $m_{bad} = m + k * N$.
3. Atacatorul transmite mai departe mesajul (m_{bad}, σ), care prezinta in continuare o semnatura valida, pierzand astfel integritatea mesajului.

Concert, orice mesaj m_{bad} cu $m_{bad} \equiv m \ (mod \ N)$ va avea aceeasi semnatura ca m. Putem asadar sa luam orice fisier / program / date, sa le apenduim $log_2(N)$ biti la sfarsit ca sa dea restul bun la impartirea la N, si acesta va pastra semnatura valida.

Putem astfel sa interceptam de exemplu un program normal, pe care il inlocuim pe un program malitios. Pentru a pastra valoarea modulo N, putem sa adaugam bitii necesari la sfarsitul programului malitios, care nu influenteaza rularea acestuia.

D

Pentru a evita atacul descris mai sus, propun doua modificari posibile:

Semnam hashul lui m

In loc sa il semnam direct pe m, care poate fi mai mare decat N, putem sa:

1. Calculam $h = SHA256(m)$. Evident, putem folosi orice alta functie de hash, si presupunem ca N are cel putin 512 de biti (cam toate modulurile de RSA au 2048 de biti).
2. Semnam valoarea h, care este in limitele necesare algoritmului descris in enunt.

Aceasta abordare evita complet nevoia de-a trata situatii cand numarul de semnat este prea mare, fara sa ofere unui atacator posibilitatea de-a altera mesajul (presupunand ca functia de hash folosita este rezistenta la a doua preimagine).

Pentru a verifica semnatura, este suficient ca functia de verificare sa aplice aceeasi functie de hash asupra lui m.

Semnam independent cifrele dintr-o baza mai mica

Ne alegem un numar MOD, de exemplu $\sqrt{N}$. Il scriem pe m in baza MOD:

$$m = \sum_{i=0} b_i * MOD^i$$

Observam ca scrierea este unica, si fiecare "cifra" b_i are cel mult $|N|/2$ cifre.

Putem acum sa ne construim vectorul σ_i, prin ecuatia:

$$\sigma_i = sign(b_i, sk)$$

Vectorul σ_i reprezinta asadar semnatura fiecarui bloc de $|N|/2$ biti, care poate fi transmis cu mesajul. Functia de verificare a semnaturi imparte in mod analog m in baza MOD, si verifica independent fiecare bucata.

Desi aceasta metoda este mai simpla, are doua probleme:

- Necesita transmiterea unui vector pentru semnatura, in loc de o singura valoare.
- Este vulnerabil la un atac in care atacatorul schimba ordinea blocurilor din m si din σ. Acest lucru poate fi mitigat prin folosirea altui mod de operatie al block ciphers, cum ar fi modul cu feedback (CFB), care impiedica inversiunea bucatilor din m.

Exercitiul 4

Observatia cheie pentru a rezolva exercitiul este urmatoarea:

$$\forall X, \quad X \text{ AND NOT}(X) = 0$$

Asadar, conform definitiei:

$$\text{Mac'}(k, m) = \text{Mac}(k, m \text{ AND NOT}(M)) = \text{Mac}(k, 0)$$

Altfel spus, Mac'(k, m) este o functie constanta, care depinde doar de cheia k.

Asadar, observam ca indiferent ce mesaj m este pasat, calcularea MAC-ului se traduce in calcularea Mac$(k, 0)$, si verificarea MAC-ului se traduce in calcularea Vrfy$(k, 0, t)$.

In mod evident, MAC-ul acesta nu este sigur de oarece putem modifica mesajul pasat oricum ne dorim, si verificarile de integritate vor valida mesajul.