|  |  |
| --- | --- |
| **Nr.**  **crt.** | **Subiect** |
| **1** | Realizați o aplicație care va implementa următoarele operații:   1. Să se genereze o cheie de sesiune Ks pornind de la următoarele operații:    1. Se introduc un *username* și o *parolă* care se vor concatena, rezultând astfel un *input\_id*;    2. Se generează o valoare random pe *64 de octeți* cu un *seed* corespunzător, rezultând astfel *token\_seed*;    3. Se calculează *Ks = SHA512(input\_id) XOR token\_seed*. 2. Pornind de la valoarea lui Ks, să se cripteze un fișier introdus de utilizator, folosind algoritmul modificat AES-256-CTR, având ca input următorii parametri:   *tempValue = SHA384(Ks)*  *K\_aes = First\_32\_bytes [ tempValue ]*  *Counter = Last\_8\_bytes [ tempValue ] || 0x0000000000000000*  Algoritmul AES-256-CTR este modificat prin introducerea următoarei limitări:   * 1. De la blocul de date cu *indexul = 0* până la blocul de date cu *indexul = Length(plaintext) / 32*, valoarea Counter va crește cu 1 unitate după fiecare bloc de date criptat;   2. De la blocul de date cu *indexul = Length(plaintext) / 32* până la blocul de date cu *indexul = Length(plaintext)*, valoarea Counter va crește cu 2 unități.   La fiecare criptare a unui bloc, se va afișa pe ecran valoarea Counter-ului în hexazecimal.  Rezultatul criptării va fi salvat într-un fișier având formatul:  *token\_seed || AES-256-CTR (data)*   1. Se generează o pereche de chei RSA cu valoarea exponentului public stabilită astfel:    1. se va parcurge cheia Ks la nivel de octet, până când se identifică un octet reprezentând un număr prim;    2. octetul identificat va deveni valoarea exponentului public;    3. în cazul în care nu este identificat un octet care să satisfacă cerința menționată anterior, atunci se va folosi 43 ca valoare implicită. 2. Folosind această pereche de chei RSA, să se semneze, folosind RSASSA-PKCS1-v1\_5, un fișier introdus de utilizator. Semnătura va fi salvată într-un fișier având următorul format:   *token\_seed || Fișier\_intrare || OCTET\_STRING(semnătură)* |
| **2** | Se va implementa apoi un program care protejeaza un fisier de date folosind o schema de securizare cu control multiplu, asa cum este descris mai jos:   1. Pentru fiecare fisier care trebui criptat, programul genereaza aleator 3 chei secrete initiale k1, k2 si k3, fiecare pe 80 biti. 2. Este generata apoi o cheie seed *seed\_key = k1 xor k2 xor k3* 3. Folosind cheia seed, se genreaza generata alte trei chei simetrice K1, K2 si K3 pe 64 biti folosind o schema de drivare bazata pe urmatorul pseudocod:   *x = seed\_key*  *for i = 1...1576*  *x = SHA512 (x)*  *K1 = first64MSBbits (x)*  *K2 = next64MSBbits (x)*  *K3 = next64MSBbits (x)*   1. Datele fisierului care trebuie protejat este criptat cu algoritmul 3DES (varianta cu 3 chei) folosind cheile K1, K2 si K3, enc(filedata) = 3DES(K1, K2, K3, filedata) 2. Fiecare din cele 3 chei K1, K2 si K3 se cripteaza folosind RSA astfel:   *enc(K1) = K1 ^ 17 mod n1*  *enc(K2) = K2 ^ 17 mod n2*  *enc(K3) = K3 ^ 17 mod n3*  unde n1, n2, n3 sunt valorile de modul din cadrul cheilor RSA generate la punctul B pentru cei 3 utilizatori.   1. In final, componentele enc(filedata) si enc(K1), enc(K2) si enc (K3) obtinute la punctele d si e se asambleaza folosind urmatoarea structura:   *len(enc(K1)) – pe 2 bytes – lungimea informatiei enc(K1)*  *enc(K1) – pe len(enc(K1)) bytes*  *len(enc(K2)) – pe 2 bytes – lungimea informatiei enc(K2)*  *enc(K2) – len(enc(K2)) bytes*  *len(enc(K3)) – pe 2 bytes – lungimea informatiei enc(K3)*  *enc(K3) – pe len(enc(K3)) bytes*  *len(enc(filedata) – pe 2 bytes*  *enc(filedata) – pe len(enc(filedata) bytes.* |
| **3** | Sǎ se realizeze o aplicatie prin care se implementeaza un modul de “login”. Aplicatia va avea un o functie void set\_password (char \*user, char \*passphrase) care stabileste parola pentru utilizatorul user si functioneaza in felul urmator:  1. Alege la intamplare un salt de lungime 8 octeti  2. Calculeaza Hash = SHA256[passphrase || salt]  3. Calculeaza IV = First128bits{Hash}, Key = Next128bits{Hash}  4. Genereaza tokenul de logare ca fiind salt || AES-128-CBCIV, Key [user]. Acesta va fi scris in fisierul login.user  Sa se realizeze apoi o functie de bool login (char \*user, char \*passphrase) care intoarce true sau false, functie de rezultatul operatiei de login. Aceste functii vor fi apoi folosite intr-un program care afiseaza daca un utilizator s-a « logat » sau nu cu succes. |
| **4** | Sǎ se realizeze o aplicatie prin care se implementeaza un modul de autentificare a continutului unui fisier. Modulul contine 2 functii: void addAuthInfo (char \*filename, char \*passphrase, char n) care adauga la fisier informatia de autentificare si bool verifyAuthInfo (char \*filename, char \*passphrase) prin care se verifica un fisier daca este sau nu “autentic”. Functia de de adaugare  1. Alege la intamplare un salt de lungime 10 octeti.  2. Calculeaza iterativ un sir de U1,i = SHA256 [ passphrase || salt || i ], U2,i = SHA256 [passphrase || U1,i], ...Un = SHA256 [passphrase || Un-1,i]. Pe baza acestor U –uri, calculeaza Fi = U1,i XOR U2,i XOR ...Un,i  3. Calculeaza o cheie K = FirstBits [T] si un IV = NextBits [T], pentru algoritmul AES-192-CBC, unde T = F1 || F2 || F3 || ..., (numarul de F –uri necesar, se va calcula).  4. Calculeaza informatia de autentificare AI = AES-192-CBCIV, K [ SHA512[continutul fisierului] ]  5. Salveaza noul fisier ca fiind format din salt (10 octeti), n (1 octet), AI si vechiul continut. |
| **5** | A) Sǎ se realizeze o aplicatie de criptare a continutului content a unui fisier de intrare input.dat, folosind algoritmul urmator:  1. Incarca o cheie publica RSA de 2048 de biti (existenta deja). Denumim aceasta cheie Kpub.  2. Genereaza aleator o cheie simetrica KSym de 256 de biti.  3. Cripteaza fisierul folosind algoritmul AES-256-CTR si cheia generata la punctul 2. Saltul initial necesar pentru criptarea simetrica va fi generat aleator. Va rezulta un fisier output.enc care contine RSA\_Encryption [Kpub, Salt || KSym] || AES\_Encryption [KSym, content]  B) Sa se realizeze apoi decriptarea fisierului. |