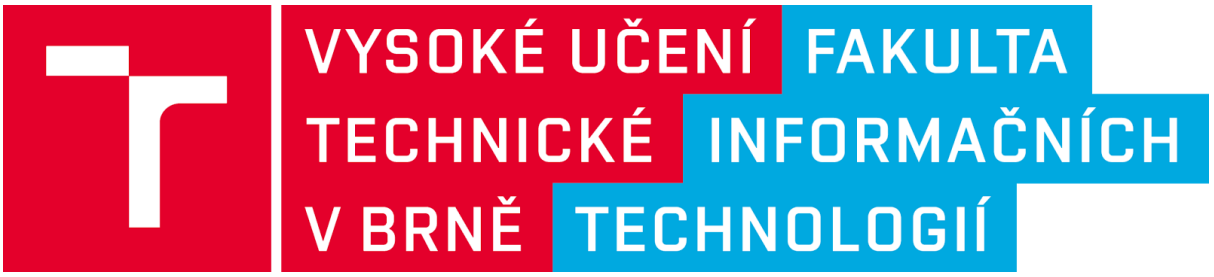


Kryptografia
Projekt č.1

2016/2017

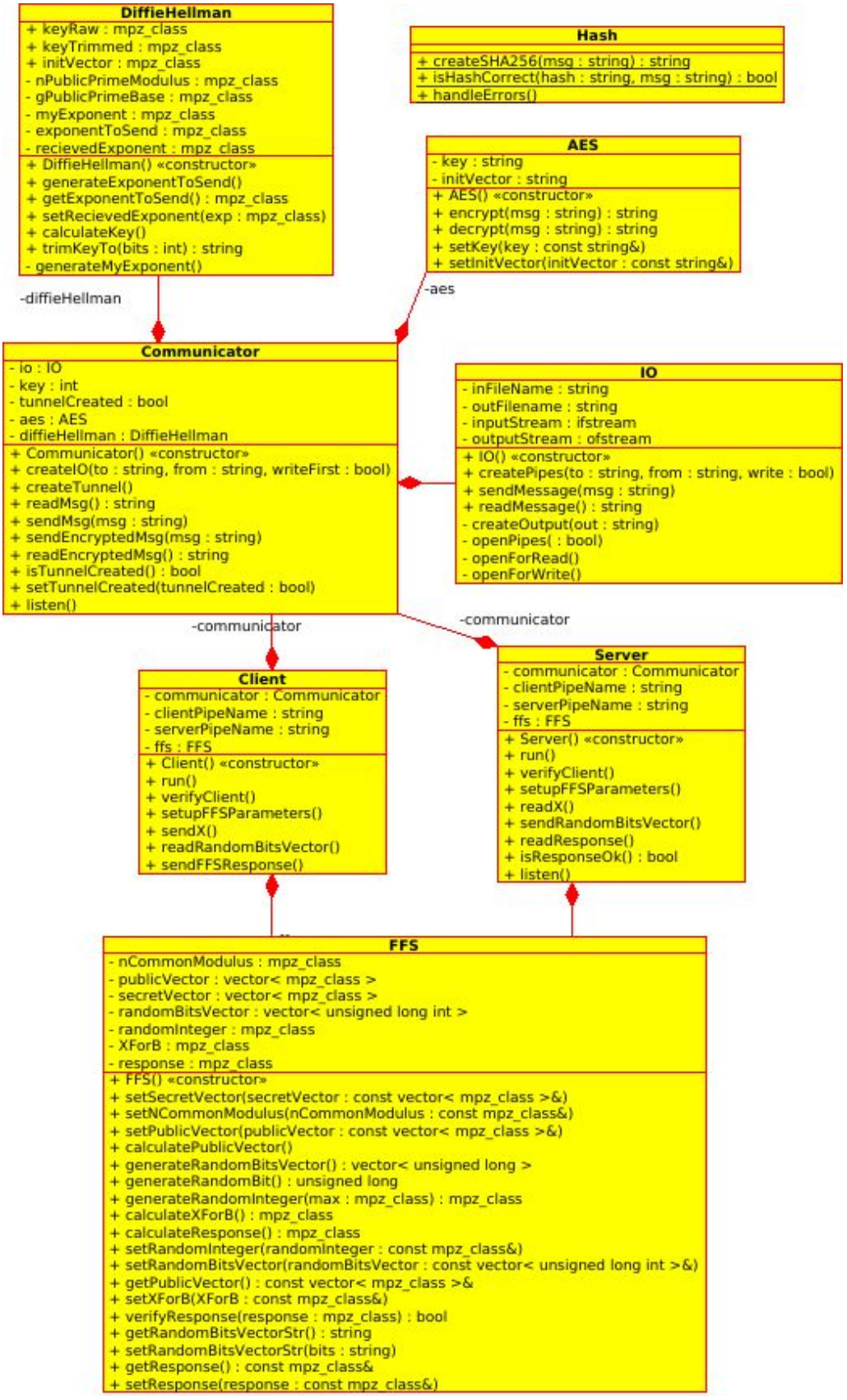
Marek Marušic

xmarus05

Úvod

Cieľom projektu bolo vytvoriť bezpečný tunel medzi klientom a serverom. Bezpečnú komunikáciu umožňujú použité protokoly a algoritmy. Pre nadviazanie komunikácie je použitý Diffie Hellman algoritmus, pomocou ktorého sa klient a server dohodnú na klúči. Ten je použitý pre symetrické šifrovanie zvyšnej komunikácie. Symetrické šifrovanie zabezpečuje algoritmus AES s 256 bitovým kľúčom v móde CBC (AES256-CBC). Server obsahuje verejný vektor klienta a po dohodnutí kľúča Diffie Hellman algoritmom sa klient autentizuje pomocou svojho privátneho vektoru Feige-Fiat-Shamirovým algoritmom. Po autentizácii klient čaká na štandardnom vstupe správy, ktoré zasieľa na server. Server na správy odpovedá SHA256 hashom pre overenie, že správa bola prijatá správne a nebola cestou modifikovaná.

Objektový diagram



Popis vlastného riešenia

Pre klienta a server boli vytvorené triedy *Communicator* a *IO*, ktoré majú na starosti komunikáciu medzi klentom a serverom. Pre komunikáciu boli použité odporúčané pomenované rúry. *IO* trieda má na starosti vytvorenie a o tvorenie pomenovaných rúr. *Communicator* má za úlohu posielanie správ medzi klientom a serverom. Taktiež iniciuje dohodnutie Diffie-Hellman kľúča. Po dohodnutí kľúča, je možné pomocou tejto triedy posielať šifrované správy.

Komunikačný protokol

Vytvorenie tunelu spočíva na dohodnutí 256bitového kľúča a 128bitového inicializačného vektoru. Tieto parametre sú získané redukciou z kľúča, na ktorom sa dohodne klient a server pomocou Diffie-Hellman algoritmu. Ďalej sa použijú spomínané parametre pre symetrické šifrovanie pomocou AES256-CBC. Po ustanovení kľúčov je klient overený pomocou Feige-Fiat-Shamirovho schéma. Po overení klienta môže klient zasielať správy serveru, ktorý na ne odpovie SHA256 hashom pre overenie správneho prijatia správ.

Metóda Diffie Hellman a redukcia kľúča

Výpočty pre Diffie Hellman algoritmus sú implementované v triede *DiffieHellman*. Ako n som zvolil 4096 bitové prvočíslo[1] a ako g som zvolil 1536 bitové prvočíslo[2]. Pre reprezentáciu čísel bola použitá GMP knižnica. Pomocou funkcie *generateMyExponent()*, ktorá používa funkciu *RAND_bytes()* z Openssl[3] knižnice, klient aj server vygenerujú 256bitový náhodný exponent. Každý z nich vypočíta $X = g^x mod\ n$ pomocou funkcie generateExponentToSend() a pošle X protistrane. Získané X od protistrany je potom použité na výpočet kľúča vo funkcii *calculateKey()* pomocou rovnice $K = g^X mod\ n$. Redukcia kľúča na dľžku 256bitov je riešená funkciou *trimKeyTo(bits)* pomocou rovnice $K' = K\ mod\ (2^{bits} - 1)$. Rovnako je vypočítaný aj inicializačný vektor.

Feige-Fiat-Shamirové schéma

Metódy pre Autentizáciu klienta sú impelemntované v triede *FFS*. Pomocou *calculatePublicVector()* bol vypočítaný verejný vektor, ktorý je súčasťou certifikátu. Tento vektor bol staticky uložený do implementovanej časti servera. Overenie bolo vykonané 4x

[1] http://www.ietf.org/rfc/rfc3526.txt

[2] http://www.ietf.org/rfc/rfc3526.txt

[3] https://www.openssl.org

ako bolo spomínané v zadaní projektu. Pomocou metódy *calculateXForB()* bolo vypočítané x, ktoré bolo následne zaslané serveru. Server potom pomocou *generateRandomBitsVector()*, vygeneruje vektor s náhodne zvolenými bitovými hodnotami. Tento vektor si uloží a zašle ho klientovi. Klient pomocou *calculateResponse()* vypočíta odpoveď a zašle ju serveru. Server potom použije *verifyResponse()* pre overenie správnosti odpovedi. Spracovanie veľkých čísel bolo vykonané pomocou knižnice GMP a jej metód ako napr. *mpz_pow_ui()* pre umocnenie čísla, *mpz_mul()* pre násobenie čísel, *mpz_powm()* pre umocnenie a následné modulo čísel, *mpz_mod()* modulo čísel a *mpz_neg()* pre negáciu čísel.

Symetrické šifrovanie AES

Symetrické šifrovanie je vykonávané triedou *AES* pomocou AES256-CBC. Metóda *encrypt(msg)* zašifruje správu pomocou ustanovených parametrov počas Diffie Hellman výmeny a redukcie kľúčov. Rovnako metóda *decrypt(msg)* použije spomínané parametre pre dešifráciu zašifrovanej správy. Metódy používajú secure_string datový typ pre uloženie dát. Zašifrované dáta sú konvertované z reťazca bytov na reťazec celých čísel oddelených medzerou pre ich jednoduchší prenos pomocou datového typu string (nevzniká problém s reprezentáciou špeciálnych symbolov ako napr \n, \0 atď.). Pred dešifráciou su spomínané čísla znova prekonvertované naspäť do reťazca bytov. Spomínané metódy sú inšpirované príkladom uvedeným pri popise EVP symetrického šifrovania[4].

Hashovanie

Pre hashovanie je vytvorená trieda Hash so statickými metódami *createSHA256()*, ktorá vytvorí hash zo zadaného reťazca.
Táto metóda bola vytvorená podla vzoru v EVP knižnici Otpenssl[5]. Metóda *isHashCorrect()* overí, či prijatý hash je zhodný so správnym hashhom správy čím zaistíme integritu správ.

Analýza zabezpečenia

Pre dobré zabezpečenie je nutné použiť spomínané algoritmy pretože každý z nich ma za úlohu iným spôsobom poskytovať bezpečnosť. Diffie-Hellman algoritmus má na starosti bezpečné ustanovenie kľúčov medzi klientom a serverom, avšak nekontroluje či je server alebo klient ten za koho sa vydáva resp. jeho autenticitu. Preto je nutné použiť niektorý z

[4] https://wiki.openssl.org/index.php/EVP_Symmetric_Encryption_and_Decryption

[5] https://wiki.openssl.org/index.php/EVP_Message_Digests

autentizačných algoritmov, ktorý dokáže overiť či je klient naozaj ten za koho sa vydáva. Avšak, keby nebol použitý Diffie-Hellman algoritmus a AES symetrické šifrovanie, potom by klient a server komunikovali pomocou otvoreného textu, ku ktorému by mal potencionálny útočník prístup a teda by overenie klienta pomocou Feige-Fiat-Shamirovho schema nemalo veľký význam. Pretože útočník by dokázal odpočúvať a tak isto aj upravovať otvorený text resp. správy vymieňané medzi klientom a serverom. Hashovacia funkcia je potrebná pre overenie integrity správ a teda či nebola správa po ceste od klienta k serveru zmenená. Takejto zmene nedokážeme v tomto protokole predísť ani šifrovaním pomocou AES. Pretože, ak sa útočník dostane k šifrovanému textu môže ho zmeniť (napr. zmenou niektorého bitu šifrovaného textu) čím zmení celý význam správy po dešifrovaní.

Spustenie projektu

Pre kompiláciu projektu treba použiť príkaz

make.

Spustenie klienta je možné pomocou príkazu

./KRY_projekt1 -c

a pre spustenie serveru je potrebné použiť

./KRY_projekt1 -s.

Po spustení a nadviazania komunikácie je nutné zadať na štandardný vstup správy, ktoré majú byť zaslané serveru. Po zaslaní správy “exit,, sa klient aj server správne ukončia.

Záver

V tomto projekte sme demonštrovali a implementovali rôzne bezpečnostné algoritmy, ktoré ako jeden celok poskytujú bezpečný prenos správ (Diffie Hellman a AES), autentizáciu klienta (Feige-Fiat-Shamirovo schéma) a integritu správ (SHA256). Projekt bol naimplementovaný v jazyku C++. Pre implementáciu bol použitý tzv. clean code, ktorý by mal čitatelovi umožniť jednoduchšiu orientáciu a čitateľnosť kódu.