Čavrljanje uz šifriranje od kraja do kraja

Kriptografija i kriptoanaliza 2025./2026.
Laboratorijska vježba

1 Uvod

Cilj laboratorijske vježbe je implementacija sigurnog klijenta za čavrljanje (*chat*) temeljenog na enkripciji s kraja na kraj (*End-to-end encryption* [E2EE]) koristeći inačicu **Double Ratchet** algoritma. Double Ratchet je protokol koji se koristi u popularnoj *chat* aplikaciji Signal, a njegove inačice koriste se u mnogim drugim aplikacijama, uključujući WhatsApp i Messenger. Protokol zadovoljava mnoga snažna sigurnosna svojstva, uključujući unaprijednu sigurnost (*perfect forward secrecy*) i oporavak od kompromitiranja (*break-in recovery* ili *post-compromise security*).

Double Ratchet protokol koriste dva sudionika kako bi razmjenjivali enkriptirane poruke koristeći dijeljeni ključ. Svaka poslana ili primljena poruka enkriptirana je jedinstvenim simetričnim ključem koji je izveden iz dijeljenog ključa koristeći funkciju za derivaciju ključa. Ovaj dio protokola naziva se **symmetric-key ratchet** i osigurava svojstvo unaprijedne sigurnosti (*perfect forward secrecy*)—kompromitirani izvedeni ključ u budućnosti ne može se iskoristiti za dekripciju poruka iz prošlosti.

Tijekom razmjene poruka sudionici također razmjenjuju Diffie-Hellman javne ključeve pomoću kojih dolaze do novih dijeljenih ključeva. Ovaj dio protokola naziva se **Diffie-Hellman ratchet** i osigurava svojstvo oporavka od kompromitiranja: uz određene pretpostavke, protokol može uspostaviti sigurnu komunikaciju između sudionika u budućnosti čak i ako je trenutni dijeljeni ključ sudionika kompromitiran.

Prije početka *Double Ratchet* protokola, sudionici se moraju putem servera međusobno autentificirati i uspostaviti dijeljeni ključ. Za ovu svrhu koriste **Extended Triple Diffie-Hellman Key Agreement** (X3DH) protokol. X3DH omogućuje jednom sudioniku (Ani) da autentificira drugog sudionika (Krešu), izvede dijeljeni ključ i pošalje poruku Kreši čak i ako je on trenutno *offline*. Ana ovo postiže putem servera koji sadrži javne ključeve Kreše i koji će proslijediti poruku Kreši kada on bude *online*.

Dodatno, pretpostavit ćemo da se naš klijent za čavrljanje koristi u zemlji s totalitarnim režimom gdje **Veliki Brat** pomno prati sve građane! U ovoj Orwellovskoj državi, zahtijeva se da svaka poslana poruka u svom zaglavlju

uključi kriptirani sjednički ključ pomoću javnog ključa koji je izdala Vlada, koristeći kriptosustav javnog ključa **ElGamal**. Na taj način **Ministarstvo Istine** može lako doći do sadržaja svake poslane poruke. Stoga, pazite da ne dijelite rješenje laboratorijske vježbe jer ćete biti uhvaćeni na djelu!

Protokol možete implementirati u proizvoljnom programskom jeziku, na proizvoljnom operacijskom sustavu. Grafičko korisničko sučelje (GUI) nije potrebno implementirati i neće se dodatno bodovati.

Prilikom implementacije koristit ćete razne kriptografske primitive s kojima ćete se upoznati na predavanjima. Neke od njih su asimetrična enkripcija, digitalni potpis, autentificirana enkripcija i funkcija za derivaciju ključa. Iz razloga što se u praksi nikako ne preporučuje implementacija vlastitih kriptografskih primitiva, koristit ćete već dostupne kriptografske primitive u sklopu programske biblioteke.

U suštini, ideja ove vježbe je da **na pravilan način** naučite koristiti već dostupne kriptografske primitive. Ovo uključuje i **pridržavanje dobrih programskih praksi**. Rješenja koja koriste loše programske prakse bodovat će se negativno, čak i ako su funkcionalno ispravna i zadovoljavaju sva tražena sigurnosna svojstva!

2 Algoritam Double Ratchet

U sklopu laboratorijske vježbe implementirat ćete klijent za čavrljanje koji se temelji na algoritmu **Double Ratchet**. Opis algoritma dostupan je na adresi https://signal.org/docs/specifications/doubleratchet. Preporučujemo da pročitate sva poglavlja, s posebnim naglaskom na poglavlja 1, 2 i 3. Vaša implementacija mora pravilno koristiti *Double Ratchet* kako je opisan u poglavlju 3, uz sljedeće izmjene i napomene.

- Za *Diffie-Hellman ratchet* i *symmetric-key ratchet* možete koristiti HKDF — funkciju za derivaciju ključa zasnovanu na autentifikacijskom kodu HMAC. Pravilno korištenje opisano je u poglavlju 5.2. Ako HKDF nije dostupan u kriptografskoj biblioteci programskog jezika koji ste odabrali, potrebno je koristiti neku drugu **sigurnu** funkciju za derivaciju ključa. Obratite pozornost da navedenu funkciju koristite na ispravan način.
- Za šifriranje sadržaja poruka koristite simetričnu šifru AES-GCM te ključeve za slanje (*sending*) i primanje (*receiving*), kako je opisano u poglavlju 2.4. Javne ključeve za *Diffie-Hellman ratchet* potrebno je autenticirati i objaviti u zaglavlju.
- Klijent mora biti sposoban dešifrirati poruke koje su stigle neispravnim redoslijedom (poglavlje 2.6). Dakle, potrebno je putem zaglavlja proslijediti redni broj trenutne poruke (`N`) i redni broj posljednje poruke iz prethodnog lanca slanja (`PN`). Obje vrijednosti potrebno je autentificirati.
- Zanemarite niz bajtova `AD` prilikom poziva funkcija `RatchetEncrypt` i `RatchetDecrypt`.

- Umjesto korištenja protokola za uspostavu ključeva *Extended Triple Diffie-Hellman*, javne ključeve ćemo dijeliti putem jednostavnih certifikata. Na početku, svaki klijent generira certifikacijski objekt koji sadrži njegov inicijalni javni Diffie-Hellman ključ. Pretpostavljamo da postoji povjerljiva središnja strana (npr. središnji server naše *chat* aplikacije) koja može na siguran način primati certifikacijske objekte koje generiraju klijenti. Nakon što potvrdi identitet klijenta, središnja strana potpisuje certifikacijski objekt i distribuira tako dobiveni certifikat drugim klijentima, koji sada mogu verificirati njegovu valjanost. Klijenti zatim kombiniraju dobiveni javni ključ sa svojim privatnim ključem kako bi izračunali zajedničku tajnu—inicijalni *root* ključ.

- Svaka poslana poruka treba se moći dekriptirati uz pomoć privatnog kljuca od Velikog brata; pripadni javni ključ dobiti ćete prilikom inicijalizacije klijenta. U tu svrhu koristite protokol enkripcije *ElGamal* tako da, umjesto množenja, *sending key* (tj. *message key*) kriptirate pomoću AES-GCM uz pomoć javnog ključa od Velikog Brata. Prema tome, neka zaglavlje dodatno sadržava polja `gov_pub` (*ephemeral key*) i `gov_ct` (*ciphertext*) koja predstavljaju izlaz (k_E, y) kriptosustava javnog kljuca Elgamal te `gov_iv` kao pripadni inicijalizacijski vektor. U ovu svrhu proučite *Elgamal Encryption Protocol* iz udžbenika *Understanding Cryptography* autora Christopha Paara i Jana Pelzla (glavna literatura). Takoder, pogledajte dokumentaciju funkcije `gov_decrypt`.

- Nasumično generirajte novi inicializacijski vektor (IV) prilikom svake enkripcije s AES-GCM i, kao što smo rekli, proslijedite ga putem zaglavlja poruke. U slučaju da funkcija/metoda za AES-GCM iz biblioteke koju koristite ne prima IV kao parametar, ispravljača morate uvjeriti (putem dokumentacije ili izvornog koda) da se IV unutar metode uistinu nasumično generira.

- Prostorna složenost algoritma (klijenta) s obzirom na korištenje ključeva treba biti $\mathcal{O}(1)$ neovisno o broju poslanih poruka. Dakle, dozvoljeno je spremati neki konstantan broj (npr. `MAX_MSG_SKIP`) ključeva u svrhu dekriptiranja određenog broja "starih" poruka.

3 Početno programsko okruženje

Unutar `kik-lab.zip` datoteke nalaze se dva direktorija: `python_template` i `java_template`. Direktoriji sadrže početni kôd `messenger` i *unit* testove `test-messenger` pisane u Python i Java programskim jezicima. U sklopu laboratorijske vježbe možete koristiti jedan od navedenih predložaka. Početni kôd sadrži specificirane metode koje trebate implementirati (označene s `TODO`). Implementacija metoda i sve pomoćne funkcije trebaju se nalaziti **samo** u datoteci `messenger`. Molimo **ne pisati** kôd u drugim datotekama.

U svrhu korištenja kriptografskih primitiva morate koristiti pouzdanu kriptografsku biblioteku prema Vašem izboru. Za programski jezik Python predlažemo biblioteku *cryptography* (https://cryptography.io/en/latest/) koju možete instalirati pomoću naredbe `pip install cryptography`. Za programski jezik Java preporučamo biblioteku *Google Tink* (https://developers.google.com/tink) koja ima jednostavnije sučelje u usporedi sa standardnim *Java Cryptography Extension* okvirom te, dodatno, podržava HKDF. Navedenu biblioteku možete instalirati kao *Maven project dependency* koristeći konfiguraciju iz dostupnog predloška.

Aplikaciju ne morate nužno implementirati u programskom jeziku Python ili Java. U slučaju da se odlučite za neki drugi programski jezik zahtjevamo da sami napišete *unit* testove analogne postojećima. Grafičko korisničko sučelje (GUI) nije nužno implementirati i neće se dodatno bodovati.

4 Programsko sučelje

U početnom kôdu dozvoljeno je mijenjati sve osim sljedećeg sučelja koje trebate implementirati. Ovaj opis se odnosi na Python, no slično vrijedi i za Javu.

- `Messenger.generate_certificate()`

  Metoda generira i vraća certifikacijski objekt.

  Metoda generira inicijalni par *Diffie-Hellman* ključeva. Serijalizirani javni ključ, zajedno s imenom klijenta, pohranjuje se u certifikacijski objekt kojeg metoda vraća. Certifikacijski objekt može biti proizvoljnog tipa (npr. dict ili tuple). Ključ je potrebno serijalizirati; format (PEM ili DER) je proizvoljan.

  Certifikacijski objekt koji metoda vrati bit će potpisan od strane CA te će tako dobiveni certifikat biti proslijeđen drugim klijentima.

- `Messenger.receive_certificate(cert_data, cert_sig)`

  Metoda verificira certifikat od CA i sprema informacije o klijentu.

  Metoda prima certifikat — certifikacijski objekt koji sadrži inicijalni Diffie-Hellman javni ključ i ime klijenta s kojim želi komunicirati te njegov potpis. Certifikat se verificira pomoću javnog ključa CA (Certificate Authority), a ako verifikacija uspije, informacije o klijentu (ime i javni ključ) se pohranjuju. Javni ključ CA je spremljen tijekom inicijalizacije objekta.

  U slučaju da verifikacija ne prođe uspješno, potrebno je baciti iznimku.

- `Messenger.send_message(name, message)`

  Metoda šalje kriptiranu poruku `message` i odgovarajuće zaglavlje korisniku `name`.

  Zaglavlje poruke treba sadržavati podatke potrebne 1) klijentu da derivira nove ključeve i dekriptira poruku; 2) Velikom Bratu da dekriptira *sending* ključ i dođe do sadržaja poruke.

Pretpostavite da već posjedujete certifikacijski objekt klijenta (dobiven pomoću metode `receive_certificate`) i da klijent posjeduje vaš. Ako prethodno niste komunicirali, uspostavite sesiju generiranjem ključeva potrebnih za *Double Ratchet* prema specifikaciji. Inicijalni korijenski ključ (*root key* za *Diffie-Hellman ratchet*) izračunajte pomoću ključa dobivenog u certifikatu i vašeg inicijalnog privatnog ključa.

Svaka poruka se sastoji od sadržaja i zaglavlja. Svaki put kada šaljete poruku napravite korak u lancu *symmetric-key ratchet* i lancu *Diffie-Hellman ratchet* ako je potrebno prema specifikaciji (ovo drugo možete napraviti i prilikom primanja poruke); *Diffie-Helman ratchet* javni ključ oglasite putem zaglavlja. S novim ključem za poruke *message key* kriptirajte i autentificirajte sadržaj poruke koristeći simetrični kriptosustav AES-GCM; inicijalizacijski vektor proslijedite putem zaglavlja. Dodatno, autentificirajte odgovarajuća polja iz zaglavlja, prema specifikaciji.

Sve poruke se trebaju moći dekriptirati uz pomoć privatnog kljuca od Velikog brata; pripadni javni ključ dobiti ćete prilikom inicijalizacije klijenta. U tu svrhu koristite protokol enkripcije *ElGamal* tako da, umjesto množenja, *sending key* (tj. *message key*) kriptirate pomoću AES-GCM uz pomoć javnog ključa od Velikog Brata. Prema tome, neka zaglavlje dodatno sadržava polja `gov_pub` (*ephemeral key*) i `gov_ct` (*ciphertext*) koja predstavljaju izlaz (k_E, y) kriptosustava javnog kljuca *Elgamal* te `gov_iv` kao pripadni inicijalizacijski vektor.

U ovu svrhu proučite *Elgamal Encryption Protocol* u udžbeniku *Understanding Cryptography* (glavna literatura). Takoder, pročitajte dokumentaciju funkcije `gov_decrypt`.

Za zaglavlje možete koristiti već dostupnu strukturu `Header` koja sadrži sva potrebna polja.

Metoda treba vratiti *tuple*: `(header, ciphertext)`.

- `Messenger.receive_message(name, (header, ciphertext))`

  Metoda prima kriptiranu poruku od korisnika s imenom `name`. Pretpostavite da već posjedujete certifikacijski objekt od korisnika (dobiven pomoću `receive_certificate`) i da je korisnik izračunao inicijalni korijenski ključ uz pomoć javnog *Diffie-Hellman* ključa iz vašeg certifikata. Ako već prije niste komunicirali, uspostavite sesiju tako da generirate nužne *Double Ratchet* ključeve prema specifikaciji.

  Svaki put kada primite poruku napravite *ratchet* korak u *receiving* lanacu (i *root* lanacu ako je potrebno prema specifikaciji) koristeći informacije dostupne u zaglavlju i dekriptirajte poruku uz pomoć novog *receiving* ključa. Ako detektirate da je integritet poruke narušen, zaustavite izvršavanje programa i generirajte iznimku.

  Metoda treba vratiti dekriptiranu poruku.

- `gov_decrypt(gov_priv, (header, ciphertext))`

Dekriptiranje poruke unutar kriptosustava javnog ključa *ElGamal*, gdje se umjesto kriptiranja jasnog teksta množenjem u $\mathbb{Z}_p^*$, jasni tekst kriptira koristeći simetričnu šifru AES-GCM.

Procitati poglavlje *The Elgamal Encryption Scheme* u udzbeniku *Understanding Cryptography* (glavna literatura) te obratiti pozornost na *Elgamal Encryption Protocol*.

Funkcija treba:

1. Izračunati *masking key* `k_M` koristeci privatni kljuc `gov_priv` i javni kljuc `gov_pub` koji se nalazi u zaglavlju `header`.
2. Iz `k_M` derivirati kljuc `k` za AES-GCM koristeci odgovarajucu funkciju za derivaciju kljuca.
3. Koristeci `k` i AES-GCM dekriptirati `gov_ct` iz zaglavlja da se dobije *sending key* `mk`
4. Koristeci `mk` i AES-GCM dekriptirati sifrat `ciphertext` orginalne poruke.
5. Vratiti tako dobiveni jasni tekst.

5 Predaja i obrana

Rješenje laboratorijske vježbe `messenger`, zajedno s pripadajućim testovima `test-messenger`, zapakirajte u `.zip` datoteku nazvanu prema obrascu *kik-2025-2026-prezime-ime-jmbag.zip*. Laboratorijsku vježbu potrebno je predati putem sustava **Moodle** najkasnije do **22. siječnja 2026. u 23:59**.

Laboratorijska vježba nosi ukupno **10 bodova**. Obrana laboratorijske vježbe obuhvaća pokretanje postojećih testova, potencijalno dodavanje novih, ispitivanje razumijevanja programskog kôda te provjeru gradiva vezanog uz laboratorijsku vježbu. Obrana će se održavati **uživo**, prema terminima vidljivim u vašim kalendarima. Studenti su obavezni pristupiti obrani u predviđenom terminu; promjena termina moguća je isključivo u **iznimnim slučajevima**.

Pitanja za laboratorijsku vježbu moguća su putem službene elektroničke pošte predmeta (kik@fer.hr) ili konzultacija uz prethodni dogovor putem elektroničke pošte.

Važno: Dozvoljeno je i poželjno diskutiranje mogućih pristupa rješavanju vježbe između studenata. Međutim, samu laboratorijsku vježbu studenti moraju raditi samostalno. Nastavno osoblje će provesti provjere sličnosti predanih rješenja, a ponašanje koje nije u skladu s Kodeksom ponašanja studenata FER-a ćemo prijaviti Povjerenstvu za stegovnu odgovornost studenatate odrediti dodatne sankcije u sklopu predmeta. U slučaju problema ili nedoumica prilikom izrade vježbe molimo da pravovremeno kontaktirate nastavno osoblje.

6 Korisne povezice

- Specifikacija algoritma *Double-Ratchet*:
  https://signal.org/docs/specifications/doubleratchet.
- Bibilioteka *pyca/cryptography* za Python:
  https://cryptography.io/en/latest/
- Bibilioteka *Google Tink* za Javu:
  https://developers.google.com/tink
- Glavna literatura: *Understanding Cryptography* autora Christopha Paara i Jana Pelzla.