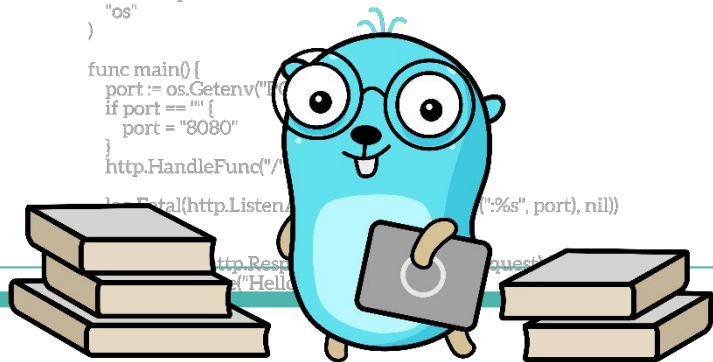

HTTPS deep dive

veljača 2025., Tehničko veleučilište u Zagrebu, Golang ZG meetup

```
// This server can run on App Engine.  
package main
```

```
import (  
    "fmt"  
    "log"  
    "net/http"  
    "os"  
)
```

```
func main() {  
    port := os.Getenv("PORT")  
    if port == "" {  
        port = "8080"  
    }  
    http.HandleFunc("/",
```



go learn()

0 meni



- Bruno Banelli
- 15+ godina iskustva
- Sistemski i mrežni development od L2 do L7
- Aktivni član raznih organizacija (Linux Foundation, OCP)
- Predajem na FER-u, Algebra i Sveučilištu u Varšavi

Go HTTP

- Još jedna od fantastičnih prednosti Go je njegov ugrađeni HTTP paket
- Paket sadrži klijentsku i serversku komponentu
- Po dizajnu, Go HTTP server:
 - se izvršava u gorutinama
 - nema blokirajući IO
 - trivijalan za korištenje, ne zahtijeva nikakav callback ili bilo što slično
- Izuzetno visoke out-of-the box performanse
- Intuitivan za korištenje

HTTP 101

- Što je “mreža”?
- Različiti profili struka dati će vam različite odgovore
- HTTP je layer 7 OSI aplikativni protokol
- Povijesno primitivan (req/resp)



HTTP 101

- I zahtijev (*request*) i odgovor (*response*) (najčešće) sadrže zaglavlje (*header*) i tijelo (*body*)
- Svaki request ima svoju metodu (ili glagol)
- GET/POST/PUT/DELETE/OPTION/PATCH/HEAD...
- Metode su preporučene RFC-om, ali možete izmisliti i svoju
- Metode su, kao i nazivi kemijskih elemenata, “case sensitive”
- Neke metode možemo smatrati “sigurnima” (**GET**, OPTION, HEAD...) jer su po konvenciji “read only”, odnosno samo dohvaćaju podatke
- Metode poput POST, **PUT, DELETE** ili PATCH obično kao posljedicu svoje akcije ostavljaju neki efekt na serveru
- **Po konvenciji**; ali možete učiniti i GET opasnim:
`mojkulwebservis.com/?action=kill&what=all` prevedete kao “rm -rf”

HTTP 101

- Response sadrži:
 - Status sa odgovarajućim statusnim kodom
 - Header
 - Body
- Statusi daju do znanja kakav tip odgovora smo dobili, gdje 200 označava uspjeh (OK), 404 je klijentska greška (Not found) dok je 500 serverska greška (Internal server error)
- Statusi (i njihovi kodovi) su definirani RFC 7231 dokumentom, no postoje i nestandardni statusi, pa samim time, možete nekada definirati i svoje

Unofficial codes [\[edit \]](#)

The following codes are not specified by any standard.

218 This is fine ([Apache Web Server](#))

Used as a catch-all error condition for allowing response bodies to flow through Apache when ProxyErrorOverride is enabled. When ProxyErrorOverride is enabled in Apache, response bodies that contain a status code of 4xx or 5xx are automatically discarded by Apache in favor of a generic response or a custom response specified by the ErrorDocument directive.^{[\[72\]](#)}

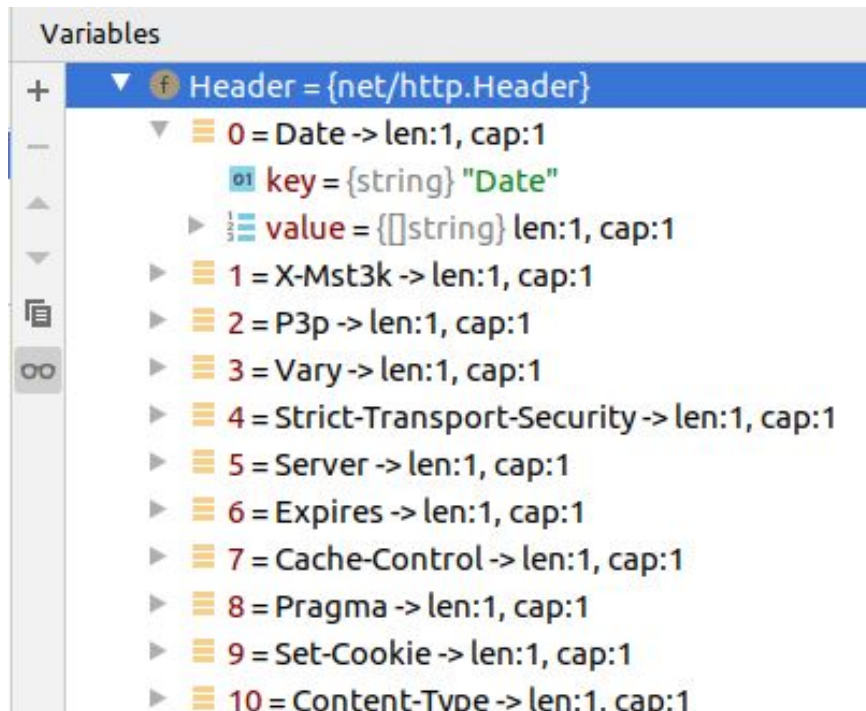


Go HTTP

- **Klijent**
- *Get*, *Post* i *PostForm* rade jednostavne HTTP(S) requestove
- Odgovornost klijenta je zatvoriti body nakon što završi manipulaciju istim
- <https://play.golang.org/p/HgfvnpBqc5P>
- Za veću kontrolu HTTP klijenta (headeri, redirect policy i slično), potrebno je kreirati i podesiti strukturu *Client*
- https://play.golang.org/p/X32a1c_w58F
- Dodatnu granulaciju klijenta ostvarujemo putem *Transport* i *Dialer* struktura
- Trivijalno je doći do headera odgovora klijentu (napomena – header je koncipiran kao *map[string][]string* pa su potrebne dvije petlje)

Go HTTP

- <https://play.golang.org/p/l14LDhDalFX>



The screenshot shows the 'Variables' window in the Go Playground. The root variable is 'Header' of type '{net/http.Header}'. It is expanded to show an array of 11 header entries, indexed 0 to 10. Each entry consists of a key and a value, both of type string. The keys are: 'Date', 'X-Mst3k', 'P3p', 'Vary', 'Strict-Transport-Security', 'Server', 'Expires', 'Cache-Control', 'Pragma', 'Set-Cookie', and 'Content-Type'. The values are all empty strings. The window also shows the length and capacity of each string (len:1, cap:1).

```
Variables
+ ▾ f Header = {net/http.Header}
  ▾ 0 = Date -> len:1, cap:1
    01 key = {string} "Date"
    1/2 value = {[]string} len:1, cap:1
  ▸ 1 = X-Mst3k -> len:1, cap:1
  ▸ 2 = P3p -> len:1, cap:1
  ▸ 3 = Vary -> len:1, cap:1
  ▸ 4 = Strict-Transport-Security -> len:1, cap:1
  ▸ 5 = Server -> len:1, cap:1
  ▸ 6 = Expires -> len:1, cap:1
  ▸ 7 = Cache-Control -> len:1, cap:1
  ▸ 8 = Pragma -> len:1, cap:1
  ▸ 9 = Set-Cookie -> len:1, cap:1
  ▸ 10 = Content-Type -> len:1, cap:1
```

Go HTTP

- Go omogućuje jednostavno korištenje HTTP glagola/metode (GET, POST, PUT, PATCH, DELETE...)
- Za to je potrebno eksplicitno definirati *Client* strukturu
- Poziv se definira *NewRequest* funkcijom
- Nakon eventualne dodatne konfiguracije, potrebno je pozvati *Do* na strukturu
- <https://play.golang.org/p/4Ct63BfvQlh>
- *NewRequest* prima tri parametra – naziv *metode*, *URL* i *body*, koji je tipa *io.Reader*!
- Tek koristeći *NewRequest* možemo definirati *headere*
- <https://play.golang.org/p/jvIFxjXUHNp>

Go HTTP

- **Server**
- Ključno - nonblocking IO server, nativno u gorutinama
- Učinkovit i lijep!

```
package main
```

```
import (  
    "net/http"  
)
```

```
func root(w http.ResponseWriter, r *http.Request) {  
    w.Write([]byte("Hello World!"))  
}
```

```
func main() {  
    http.HandleFunc("/", root)  
    http.ListenAndServe(":8080", nil)  
}
```

Go HTTP

- "net/http" ime je paketa u kojem se nalazi HTTP(S) server
- `Http.HandleFunc` kazuje *http* paketu da obradi sve zahtjeve prema rootu ("/") putem funkcije *root*
- Nakon toga pozivamo *http.ListenAndServe* specificirajući port 8080 (":8080"). Drugi parametar koji je `nil` nije nam trenutno interesantan.
- `Http.ListenAndServe` uvijek vraća error, pa ga se može zamotati u `log.Fatal()` funkciju. No, da pokažemo jednostavnost, ~~kako to često programeri vole raditi i u produkciji~~, grešku ignoriramo.
- Funkcija *root* je tipa *http.HandleFunc*. Uzima *http.ResponseWriter* i *http.Request* kao argumente.
- `Http.ResponseWriter` vrijednost objedinjuje odgovor HTTP servera, pišući u nju vraćamo podatke natrag HTTP klijentu

Go HTTPS

- Sigurnost i povjerenje
- Malo matematike, jer je važno
- Kriptografija (kryptós "skriveno, tajno") je praksa i proučavanje tehnika za sigurnu komunikaciju u prisutnosti neprijateljskog djelovanja
- Moderna kriptografija uvelike se temelji na matematičkoj teoriji i praksi računalne znanosti
- Rast kriptografske tehnologije pokrenuo je brojna pravna pitanja u informacijskom dobu

Malo o kriptografiji

- Kriptografija sa simetričnim ključem
- Kriptografija sa simetričnim ključem odnosi se na metode šifriranja u kojima i pošiljalac i primatelj dijele isti ključ
- Ovo je bila jedina javno poznata vrsta šifriranja do lipnja 1976.
- Stream šifre šifriraju znamenke (obično bajtove) ili slova (u zamjenskim šiframa) poruke jednu po jednu. Primjer je ChaCha20
- Blokove šifre uzimaju određeni broj bitova i šifriraju ih kao jednu jedinicu, popunjavajući otvoreni tekst tako da je višekratnik blokove veličina. Primjer je algoritam Advanced Encryption Standard (AES)
- Neki primjeri: Twofish, Serpent, AES (Rijndael), Salsa20, ChaCha20, Blowfish, RC4, DES, 3DES

- Kriptografija s javnim ključem ili asimetrična kriptografija je kriptografski sustav koji koristi parove ključeva
- Generiranje takvih parova ključeva ovisi o kriptografskim algoritmima koji se temelje na matematičkim problemima koji se nazivaju jednosmjernim funkcijama
- Učinkovita sigurnost zahtijeva privatnost privatnog ključa; javni ključ se može otvoreno distribuirati bez ugrožavanja sigurnosti
- Alice i Bob, dva najčešća razmjenjivača poruka na svijetu
- U nekom trenutku ćemo predstaviti Evu (kao "prisluškivača" - "eavesdropper")
- U simetričnoj kriptografiji, kako bi Alice i Bob ispravno komunicirali, prvo moraju razmijeniti iste ključeve
- Međutim, to može biti teško u slučaju da se Alice i Bob zapravo nikad ne sretnu osobno
- U nekim implementacijama, kao što je Diffie-Hellman razmjena ključeva, potrebno je dodatno opterećenje za razmjenu ključeva na prvom mjestu

- Dakle, da bi Alice mogla komunicirati s više ljudi, mora razmijeniti isti ključ sa svakim od njih
- Čak i ako vjerujemo pouzdanoj kuriri, ona mora upravljati svim tim ključevima i slati tisuće poruka samo da uspostavi
- Postoji li jednostavniji način?
- Sedamdesetih godina prošlog stoljeća britanski matematičar James Ellis radio je na ideji netajne enkripcije
- Zasnovao ju je na jednostavnom, ali pametnom konceptu
- Zaključavanje (šifriranje) i otključavanje (dešifriranje) su inverzne operacije

- Alice je mogla kupiti bravu, zadržati ključ i poslati otvorenu bravu Bobu
- Bob zatim zaključava svoju poruku i šalje je natrag Alice
- Ključevi se ne razmjenjuju!
- To znači da može naširoko objaviti distribuciju svojeg lakota i dopustiti bilo kome na svijetu da joj pošalje poruku
- I ne samo to, već sada treba pratiti samo jedan ključ
- Iako još uvijek nije matematički dokazano, Ellisu je intuitivno bilo jasno kako bi to trebalo funkcionirati

- Ključ (sic!) je podijeliti ključeve na ključ za šifriranje i ključ za dešifriranje
- Ključ za dešifriranje izvodi inverznu operaciju (ponišćavanje) koju primjenjuje ključ za šifriranje
- Kako bismo to pokušali intuitivno objasniti, možemo pokazati primjer s bojama
- Kako je Bob mogao poslati Alice određenu boju bez Eve, koja je uvijek sluša, presreće?
- Suprotnost neke boje naziva se komplementarna boja, koja, kada joj se doda (raspravljamo o suptraktivnoj shemi bojanja), proizvodi bijelu

- To zapravo "poništava" učinak prve boje
- U našem slučaju miješanje boja je jednosmjerna funkcija koja je laka i brza i znatno se sporije poništava
- Alice prvo generira svoj privatni ključ, odabirući nasumično odabranu boju, recimo **crvenu**
- Sada koristi "crnu kutiju", tajni stroj za boje koji generira komplementarnu boju - što rezultira **cijanom**, koji je njezin javni ključ i šalje ga Bobu
- Imajte na umu da Eve, kao kao i obično prisluškuje, također ima pristup javnom ključu

- Sada Bob želi poslati tajnu **žutu** boju Alice
- On miješa tu tajnu s Aliceinim javnim ključem i šalje dobivenu mješavinu Alice - **zeleno**
- Sada Alice dodaje svoj privatni ključ (boju) Bobovoj mješavini
- Ovo poništava učinak njezine javne boje ostavljajući Bobovu tajnu boju
- Sada je boja bila samo zabava i igra, ali bilo je potrebno matematičko rješenje da bi ovaj pristup doista funkcionirao u praksi
- Još jednom, najbolji Njezinog Veličanstva priskočili su u pomoć, kada je britanski matematičar Clifford Cox stvorio jednosmjernu funkciju "trapdoor"

PKC

- Ovo je funkcija koju je lako izračunati u jednom smjeru
- Ipak, teško je invertirati osim ako nemate posebne informacije koje se nazivaju “trapdoor”
- Odabrao je modularno potenciranje
- Operacija modularnog potenciranja izračunava ostatak kada se cijeli broj b (baza) podignut na potenciju e (eksponent), b^e , podijeli s pozitivnim cijelim brojem m (modul)
- U simbolima, s obzirom na bazu b , eksponent e i modulo m , modularno potenciranje c je: $c = b^e \bmod m$

U praksi

- Ovo se može koristiti za šifriranje poruke na sljedeći način:
 - Bob ima poruku koja se pretvara u broj M
 - Zatim množi svoj broj samim sobom e puta (gdje je e javni eksponent)
 - Sada ide zabavni dio - on dijeli rezultat s nasumičnim brojem N i ispisuje ostatak dijeljenja - c
 - $M^e \bmod N \equiv c$
 - Ovaj izraz je računalno jednostavan za izvođenje
 - Međutim, s obzirom da su samo dostupni c , e i N , bitno je teže odrediti koji je M korišten jer bismo ga morali grubo tjerati (šalim se, brute forceati - nisam nikada pronašao adekvatan prijevod)

U praksi

- $M^e \bmod N \equiv ?$ lagano
- $?^e \bmod N \equiv c$ teško
- Sada imamo našu matematičku bravu koju je teško invertirati, ali ju je lako izračunati
- Ali što je s ključem? Nismo to do sada nigdje spomenuli, barem ne eksplicitno
- Moramo podići c na neki drugi eksponent, recimo d , koji će poništiti početnu operaciju primijenjenu na M i vratiti izvornu poruku M
- Na kraju završavamo s ovim: $M^{ed} \bmod N \equiv M$
-

U praksi

- Sada Alice treba konstruirati e i d što otežava bilo kome drugom pronalaženje d
- Ovo zahtijeva još jednu jednosmjernu funkciju koja se koristi za generiranje d , a za to se možemo okrenuti sve do Euklida
- Prije više od 2000 godina Euklid je pokazao da svaki broj ima točno jednu faktORIZACIJU na proste brojeve, koji je u našem slučaju tajni ključ
- Da se podsjetimo, rastavljanje na proste faktore je način izražavanja broja kao umnoška njegovih prostih faktora

U praksi

- Kao jednostavan primjer, pogledajmo broj 30
- $30 = 5 \times 3 \times 2$
- U osnovi, faktORIZACIJA je težak problem u smislu vremenske složenosti
- Vremenska složenost za množenje obično je linearna, dok je za faktORIZACIJU prostih brojeva eksponencijalna
- Ovo je Cox koristio kao “trapdoor”!

U praksi

- Zamislite da Alice stvara nasumični prosti broj od 150 znamenki, nazovimo ga p_1
- Zatim stvara još jedan, otprilike iste veličine, p_2
- Zatim pomnoži ta dva broja kako bi dobila broj n , koji ima više od 300 znamenaka
- Ovo množenje je jednostavno jer traje samo djelić vremena
- Zatim uzima faktORIZACIJU $n = p_1 * p_2$ i skriva ga (u “trapdooru”)
- Sada, Cox je trebao pronaći funkciju koja ovisi o poznavanju faktORIZACIJE od n

U praksi

- Opet odlazimo u prošlost, oko 1700. godinu, do jednog od najvećih iz švicarske - Eulera
- Euler je napravio mnogo važnih stvari, a jedna od njih je istraživanje distribucije prostih brojeva
- Funkcija *Phi*, koju je definirao Euler, mjeri "lomljivost" broja
- $\varphi(n)$ ispisuje koliko je cijelih brojeva manje ili jednako n koji nemaju zajednički faktor (djeljitelj) s n
- Za $\varphi(8)$, gledamo sve vrijednosti od 1 do 8 i zatim brojimo koliko cijelih brojeva 8 ne dijeli faktor veći od 1 - ili, strože govoreći, nalazimo relativno proste brojeve

U praksi

- $\varphi(8) = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$
- To znači da svaki od ovih brojeva, u kombinaciji s 8 (n) ima najveći zajednički nazivnik 1
- Zanimljivo je da je izračunavanje φ teško osim u jednom slučaju
- Dok tražimo $\varphi(p)$, gdje je p prosti broj, a budući da prosti brojevi nemaju faktore veće od 1, φ prostog broja p je $\varphi(p) = p - 1$
- Dakle, u zaključku, φ bilo kojeg prostog broja je trivijalno izračunati
- φ funkcija je također multiplikativna, što znači $\varphi(A * B) = \varphi(A) * \varphi(B)$

U praksi

- Ako znamo da je neki broj n proizvod dva prosta broja p_1 i p_2 , $n = p_1 * p_2$, tada
 - $\varphi(n) = \varphi(p_1) * \varphi(p_2)$
 - $\varphi(n) = (p_1 - 1) * (p_2 - 1)$
- Sada moramo "spojiti" φ funkciju na modularno potenciranje
- Euler još jednom, sada sa svojim poznatim Eulerovim teoremom (također poznatim kao Fermat-Eulerov teorem ili Eulerov totijentni teorem)
- $m^{\varphi(n)} \equiv 1 \pmod n$
- Napraviti ćemo jednostavne izmjene ove funkcije s općim matematičkim znanjem

U praksi

- $1^k = 1, 1 * m = m$
- $m * m^{\varphi(n)*k} \equiv m \pmod n$
- $m^{(\varphi(n)*k)+1} \equiv m \pmod n$
- Sada imamo jednadžbu za pronalaženje $e*d$, ovisno o φ
- $d = ((\varphi(n)*k)+1)/e$
- Dakle, na ovaj način dobivamo da je d Alicein ključ!
- To je “trapdoor” koji će joj omogućiti da poništi učinke e
- Sada možemo finalizirati naš postupak

U praksi

- Bob ima poruku m , koju će, radi sažetosti, jednostavno pretvoriti u broj
- $m = \text{"hi"} \Rightarrow m = 89$; ovo će biti naš N
- Alice stvara javni/privatni par ključeva $p1 = 53$, $p2 = 59$ koji su oba prosti, a zatim ih množi
- $n = 3127$
- Izračunavanje $\varphi(n)$ je, kao što smo rekli, trivijalno $\varphi(n) = 52 * 58 = 3127$
- Sada odabire neki mali javni eksponent, recimo 3, uz uvjet da je to neparan broj koji ne dijeli faktore s $\varphi(n)$

U praksi

- Zatim pronalazi vrijednost svog privatnog eksponenta d
- $d = (2 * 2016 + 1)/3 = 2011$
- Sada je sve osim vrijednosti N i e skriveno u “trapdooru”, budući da N i e čine javni ključ - otvorenu bravu
- Šalje ovo Bobu da zaključa svoju poruku
- Ovo je urađeno na način da je $\text{mod } n \Rightarrow 893 \text{ mod } 3127 = 1394$, što je c , šifrirana poruka
- Poruka se šalje natrag Alice, a ona je dešifrira koristeći svoj privatni ključ d
- $1394^{2011} = 89 \text{ mod } 3127$

U praksi

- Jedini način za izračunavanje d je znati rastavljanje n na proste faktore, za što smo rekli da je težak problem
- Dakle, ako je n dovoljno velik, Alice je sigurna
- Kada je to otkriveno, odmah je klasificirano, ali su ga neovisno ponovno otkrili Rivest–Shamir–Adleman (RSA) 1977.

Kako do toga u Gou na najbolji način?

- Usklađenost sa standardom FIPS 140-3
- Počevši od Go 1.24, Go binary može izvorno raditi u načinu rada koji olakšava usklađenost sa FIPS 140-3
- NIST FIPS 140-3 je režim usklađenosti Vlade SAD-a za kriptografske aplikacije koji, između ostalog, zahtijeva upotrebu skupa odobrenih algoritama i upotrebu kriptografskih modula provjerenih od strane CMVP-a testiranih u ciljanim radnim okruženjima.
- Aplikacije koje nemaju potrebu za usklađenošću sa standardom FIPS 140-3 mogu ih sigurno ignorirati i ne bi trebale omogućiti način rada FIPS 140-3.
- Go kriptografski modul zbirka je standardne biblioteke Go paketa pod `crypto/internal/fips140/...` koji implementiraju algoritme odobrene od strane FIPS 140-3.
- Javni API paketi kao što su `crypto/ecdsa` i `crypto/rand` transparentno koriste Go Cryptographic Module za implementaciju FIPS 140-3 algoritama.

Vrlo komplicirano - a kako to radi Go?

```
package main

import (
    "crypto/tls"
    "log"
    "net/http"
)

func main() {
    mux := http.NewServeMux()
    mux.HandleFunc("/", func(w http.ResponseWriter, req *http.Request) {
        w.Header().Add("Strict-Transport-Security", "max-age=63072000; includeSubDomains")
        w.Write([]byte("This is an example server.\n"))
    })
    cfg := &tls.Config{
        MinVersion:      tls.VersionTLS13,
        CurvePreferences: []tls.CurveID{tls.CurveP521, tls.CurveP384, tls.CurveP256},
        PreferServerCipherSuites: true,
        CipherSuites: []uint16{
            tls.TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
            tls.TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
            tls.TLS_RSA_WITH_AES_256_GCM_SHA384,
            tls.TLS_RSA_WITH_AES_256_CBC_SHA,
        },
    }
    srv := &http.Server{
        Addr:      ":443",
        Handler:    mux,
        TLSConfig:  cfg,
        TLSNextProto: make(map[string]func(*http.Server, *tls.Conn, http.Handler), 0),
    }
    log.Fatal(srv.ListenAndServeTLS("tls.crt", "tls.key"))
}
```