Go TLS Full Project

Bu proje: **self-signed CA ile imzalanmış sunucu sertifikası üretme**, **opsiyonel mutual TLS** ve hem sunucu hem istemci örnekleri içerir. Aşağıdaki dosyalar canvas içinde yer alıyor — kopyala/çalıştır veya terminaldeki talimatları izle.

Dosya: `README.md`

````
# go-tls-full-project

Bu proje, Go ile TLS kullanarak güvenli bir sunucu ve istemci örneği içerir. Ayrıca `scripts/generate_certs.sh` ile bir CA oluşturur, bu CA ile sunucu ve (opsiyonel) istemci sertifikaları imzalar.

### İçindekiler
- `scripts/generate_certs.sh` - CA, server ve client sertifikalarını oluşturur.
- `server/main.go` - TLS sunucusu. İki mod:
  - Normal TLS (server cert sadece)
  - Mutual TLS (istemciden sertifika doğrulama) — `MUTUAL=true` ortam değişkeni ile aktifleşir
- `client/main.go` - TLS istemcisi. İki mod:
  - Server doğrulama only (CA trust)
  - Mutual TLS (istemci sertifikası sunar) — `--mutual` flag ile

### Gereksinimler
- Go 1.18+
- OpenSSL (sertifika oluşturmak için)

### Hızlı kullanım
1. Sertifikaları üret:

```bash
chmod +x scripts/generate_certs.sh
./scripts/generate_certs.sh
````

1. Sunucuyu çalıştır (normal TLS):

```
cd server
go run main.go
# veya
MUTUAL=false go run main.go
```

1. İstemci (server doğrulama):
```

```
cd client
go run main.go --ca ../scripts/ca/ca.pem
```

1. Mutual TLS testi (istemci sertifikası ile):

Önce sunucuyu mutual modda başlat:

```
MUTUAL=true go run main.go
```

Sonra istemciyi mutual flag ile çalıştır:

```
go run main.go --ca ../scripts/ca/ca.pem --cert ../scripts/ca/
client_cert.pem --key ../scripts/ca/client_key.pem --mutual
```

```
Not: Sunucu 8443 portunda dinler. Eğer port kullanımında sorun varsa
değiştirebilirsin.
```

```
---

## Dosya: `scripts/generate_certs.sh`

```bash
#!/usr/bin/env bash
set -euo pipefail

OUT=./scripts/ca
mkdir -p "$OUT"
cd "$OUT"

echo "Creating CA..."
# CA private key
openssl genrsa -out ca.key 4096
# CA self-signed cert
openssl req -x509 -new -nodes -key ca.key -sha256 -days 3650
  -subj "/C=TR/ST=Istanbul/L=Istanbul/O=ExampleCA/OU=Dev/CN=Example Root
CA"
  -out ca.pem

echo "Creating server key & CSR..."
openssl genrsa -out server_key.pem 2048
openssl req -new -key server_key.pem -subj "/C=TR/ST=Istanbul/L=Istanbul/
O=ExampleServer/OU=Dev/CN=localhost" -out server.csr

# Create server cert signed by CA (include SAN localhost)
cat > server_ext.cnf <<EOF
basicConstraints=CA:FALSE
```
```

```
subjectAltName=DNS:localhost,IP:127.0.0.1
keyUsage = digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
EOF

openssl x509 -req -in server.csr -CA ca.pem -CAkey ca.key -CAcreateserial -
out server_cert.pem -days 365 -sha256 -extfile server_ext.cnf

# Optional: create client cert (for mutual TLS tests)
echo "Creating client key & CSR..."
openssl genrsa -out client_key.pem 2048
openssl req -new -key client_key.pem -subj "/C=TR/ST=Istanbul/L=Istanbul/
O=ExampleClient/OU=Dev/CN=client.local" -out client.csr

cat > client_ext.cnf <<EOF
basicConstraints=CA:FALSE
keyUsage = digitalSignature, keyEncipherment
extendedKeyUsage = clientAuth
EOF

openssl x509 -req -in client.csr -CA ca.pem -CAkey ca.key -CAcreateserial -
out client_cert.pem -days 365 -sha256 -extfile client_ext.cnf

# cleanup
rm -f server.csr client.csr server_ext.cnf client_ext.cnf ca.srl

echo "Certs generated in: $OUT"
ls -la "$OUT"
```

Dosya: `server/main.go`

```
package main

import (
    "crypto/tls"
    "crypto/x509"
    "flag"
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
    "os"
)

func main() {
    addr := flag.String("addr", ":8443", "listen address")
    certFile := flag.String("cert", "../scripts/ca/server_cert.pem", "server
cert PEM")
```

```
    keyFile := flag.String("key", "../scripts/ca/server_key.pem",
"server key PEM")
    caFile := flag.String("ca", "../scripts/ca/ca.pem", "CA cert to verify
clients (for mutual TLS)")
    flag.Parse()

    mutual := false
    if v := os.Getenv("MUTUAL"); v == "true" || v == "1" {
        mutual = true
    }

    mux := http.NewServeMux()
    mux.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        if r.TLS != nil && len(r.TLS.PeerCertificates) > 0 {
            fmt.Fprintf(w, "Hello, mutual TLS client CN=%s\n",
r.TLS.PeerCertificates[0].Subject.CommonName)
            return
        }
        w.Write([]byte("Hello, TLS world!\n"))
    })

    cert, err := tls.LoadX509KeyPair(*certFile, *keyFile)
    if err != nil {
        log.Fatalf("failed to load server key pair: %v", err)
    }

    cfg := &tls.Config{
        Certificates: []tls.Certificate{cert},
        MinVersion:   tls.VersionTLS12,
    }

    if mutual {
        // load CA pool for client cert verification
        caPEM, err := ioutil.ReadFile(*caFile)
        if err != nil {
            log.Fatalf("failed to read CA file: %v", err)
        }
        pool := x509.NewCertPool()
        if !pool.AppendCertsFromPEM(caPEM) {
            log.Fatalf("failed to append CA cert")
        }
        cfg.ClientCAs = pool
        cfg.ClientAuth = tls.RequireAndVerifyClientCert
        log.Println("Mutual TLS: enabled (require client cert)")
    } else {
        log.Println("Mutual TLS: disabled")
    }

    server := &http.Server{
        Addr:      *addr,
        Handler:   mux,
```

```
        TLSConfig: cfg,
    }

    log.Printf("Listening on %s (TLS)\n", *addr)
    log.Fatal(server.ListenAndServeTLS("", "")) // certs come from TLSConfig
}
```

Dosya: `client/main.go`

```
package main

import (
    "crypto/tls"
    "crypto/x509"
    "flag"
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
    "os"
)

func main() {
    caFile := flag.String("ca", "../scripts/ca/ca.pem", "CA cert to trust")
    certFile := flag.String("cert", "", "client cert PEM (for mutual TLS)")
    keyFile := flag.String("key", "", "client key PEM (for mutual TLS)")
    url := flag.String("url", "https://localhost:8443/", "server URL")
    mutual := flag.Bool("mutual", false, "use client cert (mutual TLS)")
    flag.Parse()

    // load CA
    caPEM, err := ioutil.ReadFile(*caFile)
    if err != nil {
        log.Fatalf("failed to read CA file: %v", err)
    }
    pool := x509.NewCertPool()
    if !pool.AppendCertsFromPEM(caPEM) {
        log.Fatalf("failed to append CA cert")
    }

    cfg := &tls.Config{
        RootCAs:    pool,
        MinVersion: tls.VersionTLS12,
        // ServerName: "localhost", // set if CN/SAN differs
    }

    if *mutual {
        if *certFile == "" || *keyFile == "" {
```

```
            log.Fatalln("mutual mode requires --cert and --key")
        }
        cert, err := tls.LoadX509KeyPair(*certFile, *keyFile)
        if err != nil {
            log.Fatalf("failed to load client key pair: %v", err)
        }
        cfg.Certificates = []tls.Certificate{cert}
    }

    tr := &http.Transport{TLSClientConfig: cfg}
    client := &http.Client{Transport: tr}

    resp, err := client.Get(*url)
    if err != nil {
        log.Fatalf("GET error: %v", err)
    }
    defer resp.Body.Close()
    b, _ := ioutil.ReadAll(resp.Body)
    fmt.Printf("status: %s\n", resp.Status)
    fmt.Printf("body:\n%s\n", string(b))

    os.Exit(0)
}
```

Dosya: `Makefile` (opsiyonel)

```
.PHONY: certs server client client-mutual

certs:
    chmod +x scripts/generate_certs.sh
    ./scripts/generate_certs.sh

server:
    cd server && go run main.go

server-mutual:
    cd server && MUTUAL=true go run main.go

client:
    cd client && go run main.go --ca ../scripts/ca/ca.pem

client-mutual:
    cd client && go run main.go --ca ../scripts/ca/ca.pem --cert ../scripts/
ca/client_cert.pem --key ../scripts/ca/client_key.pem --mutual
```

Son notlar

- Bu proje test/dev amaçlıdır. Üretimde sertifika yönetimi için Let's Encrypt, cert-manager veya HashiCorp Vault gibi çözümler tercih edin.
- `scripts/generate_certs.sh` OpenSSL kullanır; Windows kullanıcıları WSL veya benzeri bir ortamda çalıştırabilir.

Başka bir isteğin olursa (örneğin: Dockerfile ekleme, systemd servis, veya Let's Encrypt entegrasyonu) söyle, ona göre güncelleyeyim.