GitHub Gist

denji / **golang-tls.md**
forked from spikebike/client.go
Last active 2 days ago

Simple Golang HTTPS/TLS Examples

golang-tls.md

Generate private key (.key)

```
# Key considerations for algorithm "RSA" ≥ 2048-bit
openssl genrsa -out server.key 2048

# Key considerations for algorithm "ECDSA" ≥ secp384r1
# List ECDSA the supported curves (openssl ecparam -list_curves)
openssl ecparam -genkey -name secp384r1 -out server.key
```

Generation of self-signed(x509) public key (PEM-encodings `.pem` | `.crt`) based on the private (`.key`)

```
openssl req -new -x509 -sha256 -key server.key -out server.pem -days 3650
```

Simple Golang HTTPS/TLS Server

```
package main

import (
    "io"
    "net/http"
    "log"
)

func HelloServer(w http.ResponseWriter, req *http.Request) {
    io.WriteString(w, "hello, world!\n")
}

func main() {
    http.HandleFunc("/hello", HelloServer)
    err := http.ListenAndServeTLS(":443", "server.crt", "server.key", nil)
    if err != nil {
        log.Fatal("ListenAndServe: ", err)
    }
}
```

Hint: visit, please do not forget to use https begins,otherwise chrome will download a file as follows:

```
dotcoo-air:tls dotcoo$ cat ~/Downloads/hello | xxd
0000000: 1503 0100 0202 0a                        .......
```

TLS (transport layer security) — Server

```
package main

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

func main() {
    log.SetFlags(log.Lshortfile)
```

```
    cer, err := tls.LoadX509KeyPair("server.crt", "server.key")
    if err != nil {
        log.Println(err)
        return
    }

    config := &tls.Config{Certificates: []tls.Certificate{cer}}
    ln, err := tls.Listen("tcp", ":443", config)
    if err != nil {
        log.Println(err)
        return
    }
    defer ln.Close()

    for {
        conn, err := ln.Accept()
        if err != nil {
            log.Println(err)
            continue
        }
        go handleConnection(conn)
    }
}

func handleConnection(conn net.Conn) {
    defer conn.Close()
    r := bufio.NewReader(conn)
    for {
        msg, err := r.ReadString('\n')
        if err != nil {
            log.Println(err)
            return
        }

        println(msg)

        n, err := conn.Write([]byte("world\n"))
        if err != nil {
            log.Println(n, err)
            return
        }
    }
}
```

TLS (transport layer security) — `Client`

```
package main

import (
    "log"
    "crypto/tls"
)

func main() {
    log.SetFlags(log.Lshortfile)

    conf := &tls.Config{
        InsecureSkipVerify: true,
    }

    conn, err := tls.Dial("tcp", "127.0.0.1:8000", conf)
    if err != nil {
        log.Println(err)
        return
    }
    defer conn.Close()

    n, err := conn.Write([]byte("hello\n"))
    if err != nil {
        log.Println(n, err)
        return
    }

    buf := make([]byte, 100)
    n, err = conn.Read(buf)
```

```
        if err != nil {
            log.Println(n, err)
            return
        }

        println(string(buf[:n]))
    }
```

Generation of self-sign a certificate with a private (`.key`) and public key (PEM-encodings `.pem` | `.crt`) in one command:

```
# RSA recommendation key ≥ 2048-bit
openssl req -x509 -nodes -newkey ec:secp384r1 -keyout server.ecdsa.key -out server.ecdsa.crt -days 3650
# openssl req -x509 -nodes -newkey ec:<(openssl ecparam -name secp384r1) -keyout server.ecdsa.key -out serv
# -pkeyopt ec_paramgen_curve:… / ec:<(openssl ecparam -name …) / -newkey ec:…
ln -sf server.ecdsa.key server.key
ln -sf server.ecdsa.crt server.crt

# ECDSA recommendation key ≥ secp384r1
# List ECDSA the supported curves (openssl ecparam -list_curves)
openssl req -x509 -nodes -newkey rsa:2048 -keyout server.rsa.key -out server.rsa.crt -days 3650
ln -sf server.rsa.key server.key
ln -sf server.rsa.crt server.crt
```

`.crt` (synonymous most common among *nix systems)
`.der` — The DER extension is used for binary DER encoded certificates.
`.pem` = The PEM extension is used for different types of X.509v3 files which contain ASCII (Base64) armored data prefixed with a «—— BEGIN …» line.

Generating the Certficate Signing Request

```
openssl req -new -sha256 -key server.key -out server.csr
openssl x509 -req -sha256 -in server.csr -signkey server.key -out server.crt -days 3650
```

ECDSA & RSA — FAQ

- Validate the elliptic curve parameters `-check`
- List "ECDSA" the supported curves `openssl ecparam -list_curves`
- Encoding to explicit "ECDSA" `-param_enc explicit`
- Conversion form to compressed "ECDSA" `-conv_form compressed`
- "EC" parameters and a private key `-genkey`

Reference Link

- http://superuser.com/a/226229/205366
- https://gist.github.com/spikebike/2232102
- http://echo.labstack.com/guide/
- https://blog.bracelab.com/achieving-perfect-ssl-labs-score-with-go
- https://kjur.github.io/jsrsasign/sample-ecdsa.html
- https://www.guyrutenberg.com/2013/12/28/creating-self-signed-ecdsa-ssl-certificate-using-openssl/
- https://www.openssl.org/docs/manmaster/apps/ecparam.html
- https://www.openssl.org/docs/manmaster/apps/ec.html
- https://www.openssl.org/docs/manmaster/apps/req.html
- https://digitalelf.net/2016/02/creating-ssl-certificates-in-3-easy-steps/
- http://www.kaihag.com/https-and-go/
- https://blog.cloudflare.com/the-complete-guide-to-golang-net-http-timeouts/

c3mb0 commented on Feb 12 • edited

Instead of skipping insecure certificates which could expose your service to MITM attacks, you can create a client that accepts your self-signed certificate:

```
func main() {

    rootPEM := `-----BEGIN CERTIFICATE-----
MIIEBDCCAuygAwIBAgIDAjppMA0GCSqGSIb3DQEBBQUAMEIxCzAJBgNVBAYTAlVT
MRYwFAYDVQQKEw1HZW9UcnVzdCBJbmMuMRswGQYDVQQDExJHZW9UcnVzdCBHbG9i
YWwgQ0EwHhcNMTMwNDA1MTUxNTU1WhcNMTUwNDA0MTUxNTU1WjBJMQswCQYDVQQG
EwJVUzETMBEGA1UEChMKR29vZ2xlIEluYzElMCMGA1UEAxMcR29vZ2xlIEludGVy
bmV0IEF1dGhvcml0eSBHMjCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
AJwqBHdc2FCROgajguDYUEi8iT/xGXAaiEZ+4I/F8YnOIe5a/mENtzJEiaB0C1NP
VaTOgmKV7utZX8bhBYASxF6UP7xbSDj0U/ck5vuR6RXEz/RTDfRK/J9U3n2+oGtv
h8DQUB8oMANA2ghzUWx//zo8pzcGjr1LEQTrfSTe5vn8MXH7lNVg8y5Kr0LSy+rE
ahqyzFPdFUuLH8gZYR/Nnag+YyuENWllhMgZxUYi+FOVvuOAShDGKuy6lyARxzmZ
EASg8GF6lSWMTlJ14rbtCMoU/M4iarNOz0YDl5cDfsCx3nuvRTPPuj5xt970JSXC
DTWJnZ37DhF5iR43xa+OcmkCAwEAAaOB+zCB+DAfBgNVHSMEGDAWgBTAephojYn7
qwVkDBF9qn1luMrMTjAdBgNVHQ4EFgQUSt0GFhu89mi1dvWBtrtiGrpagS8wEgYD
VR0TAQH/BAgwBgEB/wIBADAOBgNVHQ8BAf8EBAMCAQYwOgYDVR0fBDMwMTAvoC2g
K4YpaHR0cDovL2NybC5nZW90cnVzdC5jb20vY3Jscy9ndGdsb2JhbC5jcmwwPQYI
KwYBBQUHAQEEMTAvMC0GCCsGAQUFBzABhiFodHRwOi8vZ3RnbG9iYWwtb2NzcC5n
ZW90cnVzdC5jb20wFwYDVR0gBBAwDjAMBgorBgEEAdZ5AgUBMA0GCSqGSIb3DQEB
BQUAA4IBAQA21waAESetKhSbOHezI6B1WLuxfoNCunLaHtiONgaX4PCVOzf9G0JY
/iLIa704XtE7JW4S615ndkZAkNoUyHgN7ZVm2o6Gb4ChulYylYbc3GrKBIxbf/a/
zG+FA1jDaFETzf3I93k9mTXwVqO94FntT0QJo544evZG0R0SnU++0ED8Vf4GXjza
HFa9llF7b1cq26KqltyMdMKVvvBulRP/F/A8rLIQjcxz++iPAsbw+zOzlTvjwsto
WHPbqCRiOwY1nQ2pM714A5AuTHhdUDqB1O6gyHA43LL5Z/qHQF1hwFGPa4NrzQU6
yuGnBXj8ytqU0CwIPX4WecigUCAkVDNx
-----END CERTIFICATE-----`

    roots := x509.NewCertPool()
    ok := roots.AppendCertsFromPEM([]byte(rootPEM))
    if !ok {
        panic("failed to parse root certificate")
    }
    tlsConf := &tls.Config{RootCAs: roots}
    tr := &http.Transport{TLSClientConfig: tlsConf}
    client := &http.Client{Transport: tr}

    conn, err := client.Dial("tcp", "127.0.0.1:8000")
    if err != nil {
        log.Println(err)
        return
    }
    defer conn.Close()

    ...
}
```

Kudos for examples!

andradei commented on Mar 4

@c3mb0 If I'm not mistaken, `http.Client` doesn't have define a `Dial()` method.

crondotnet commented on May 26

@andradei https://golang.org/pkg/net/http/#Client the struct Transport does have the Dial method

denji commented on May 28 • edited Owner

https://blog.bracelab.com/achieving-perfect-ssl-labs-score-with-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.VersionTLS12,
        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"))
}
```