# go

## for the paranoid network programmer

@feyeleanor

twitter://@feyeleanor


go for the would-be network programmer

http://slides.games-with-brains.net/

# high voltage

networking
concurrency
cryptography

# http

```
package main
import (
  . "fmt"
  "net/http"
)

const MESSAGE = "hello world"
const ADDRESS = ":1024"

func main() {
  http.HandleFunc("/hello", Hello)
  if e := http.ListenAndServe(ADDRESS, nil); e != nil {
    Println(e)
  }
}

func Hello(w http.ResponseWriter, r *http.Request) {
  w.Header().Set("Content-Type", "text/plain")
  Fprintf(w, MESSAGE)
}
```

```
package main
import (
  . "fmt"
  "net/http"
)

const MESSAGE = "hello world"
const ADDRESS = ":1024"

func main() {
  http.HandleFunc("/hello", Hello)
  if e := http.ListenAndServe(ADDRESS, nil); e != nil {
    Println(e)
  }
}

func Hello(w http.ResponseWriter, r *http.Request) {
  w.Header().Set("Content-Type", "text/plain")
  Fprintf(w, MESSAGE)
}
```

```
package main
import (
  . "fmt"
  "net/http"
)

const MESSAGE = "hello world"
const ADDRESS = ":1024"

func main() {
  http.HandleFunc("/hello", Hello)
  if e := http.ListenAndServe(ADDRESS, nil); e != nil {
    Println(e)
  }
}

func Hello(w http.ResponseWriter, r *http.Request) {
  w.Header().Set("Content-Type", "text/plain")
  Fprintf(w, MESSAGE)
}
```

```
package main
import (
  . "fmt"
  . "net/http"
)

const MESSAGE = "hello world"
const ADDRESS = ":1024"

func main() {
  HandleFunc("/hello", Hello)
  if e := ListenAndServe(ADDRESS, nil); e != nil {
    Println(e)
  }
}

func Hello(w ResponseWriter, r *Request) {
  w.Header().Set("Content-Type", "text/plain")
  Fprintf(w, MESSAGE)
}
```

```
package main
import (
  . "fmt"
  . ”net/http"
)

const MESSAGE = "hello world"
const ADDRESS = ":1024"

func main() {
  HandleFunc("/hello", Hello)
  if e := ListenAndServe(ADDRESS, nil); e != nil {
    Println(e)
  }
}

func Hello(w ResponseWriter, r *Request) {
  w.Header().Set("Content-Type", "text/plain")
  Fprintf(w, MESSAGE)
}
```

```
package main
import (
  . "fmt"
  . ”net/http"
)

const MESSAGE = "hello world"
const ADDRESS = ":1024"

func main() {
  HandleFunc("/hello", Hello)
  if e := ListenAndServe(ADDRESS, nil); e != nil {
    Println(e)
  }
}

func Hello(w ResponseWriter, r *Request) {
  w.Header().Set("Content-Type", "text/plain")
  Fprintf(w, MESSAGE)
}
```

```
package main
import (
  . "fmt"
  . ”net/http"
)

const MESSAGE = "hello world"
const ADDRESS = ":1024"

func main() {
  HandleFunc("/hello", Hello)
  if e := ListenAndServe(ADDRESS, nil); e != nil {
    Println(e)
  }
}

func Hello(w ResponseWriter, r *Request) {
  w.Header().Set("Content-Type", "text/plain")
  Fprintf(w, MESSAGE)
}
```

```
package main
import (
  . "fmt"
  . ”net/http"
)

const MESSAGE = "hello world"
const ADDRESS = ":1024"

func main() {
  HandleFunc("/hello", Hello)
  if e := ListenAndServe(ADDRESS, nil); e != nil {
    Println(e)
  }
}

func Hello(w ResponseWriter, r *Request) {
  w.Header().Set("Content-Type", "text/plain")
  Fprintf(w, MESSAGE)
}
```

```
package main
import (
  . "fmt"
  . "net/http"
)

const MESSAGE = "hello world"
const ADDRESS = ":1024"

func main() {
  HandleFunc("/hello", Hello)
  if e := ListenAndServe(ADDRESS, nil); e != nil {
    Println(e)
  }
}

func Hello(w ResponseWriter, r *Request) {
  w.Header().Set("Content-Type", "text/plain")
  Fprintf(w, MESSAGE)
}
```

```
package main
import (
  . "fmt"
  . "net/http"
)

const MESSAGE = "hello world"
const ADDRESS = ":1024"

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, MESSAGE)
  })
  ListenAndServe(ADDRESS, nil)
}
```

```
package main
import (
  . "fmt"
  . "net/http"
)

const MESSAGE = "hello world"
const ADDRESS = ":1024"

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, MESSAGE)
  })
  ListenAndServe(ADDRESS, nil)
}
```

# https

```
package main
import (
  . "fmt"
  . "net/http"
)

const ADDRESS = ":1025"

func main() {
  message := "hello world"
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, message)
  })
  ListenAndServeTLS(ADDRESS, "cert.pem", "key.pem", nil)
}
```

```
package main
import (
  . "fmt"
  . "net/http"
)

const ADDRESS = ":1025"

func main() {
  message := "hello world"
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, message)
  })
  ListenAndServeTLS(ADDRESS, "cert.pem", "key.pem", nil)
}
```

```
package main
import (
  . "fmt"
  . "net/http"
)

const ADDRESS = ":1025"

func main() {
  message := "hello world"
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, message)
  })
  ListenAndServeTLS(ADDRESS, "cert.pem", "key.pem", nil)
}
```

```
package main
import (
  . "fmt"
  . "net/http"
)

const ADDRESS = ":1025"

func main() {
  message := "hello world"
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, message)
  })
  ListenAndServeTLS(ADDRESS, "cert.pem", "key.pem", nil)
}
```

# multiple http servers

```
package main
import . "fmt"
import . "net/http"

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, "hello world")
  })

  done := make(chan bool)
  go func() {
    ListenAndServe(":1024", nil)
    done <- true
  }()

  ListenAndServeTLS(":1025", "cert.pem", "key.pem", nil)
  <- done
}
```

```
package main
import . "fmt"
import . "net/http"

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, "hello world")
  })

  done := make(chan bool)
  go func() {
    ListenAndServe(":1024", nil)
    done <- true
  }()

  ListenAndServeTLS(":1025", "cert.pem", "key.pem", nil)
  <- done
}
```

```
package main
import . "fmt"
import . "net/http"

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, "hello world")
  })

  done := make(chan bool)
  go func() {
    ListenAndServe(":1024", nil)
    done <- true
  }()

  ListenAndServeTLS(":1025", "cert.pem", "key.pem", nil)
  <- done
}
```

```
package main
import . "fmt"
import . "net/http"

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, "hello world")
  })

  done := make(chan bool)
  go func() {
    ListenAndServe(":1024", nil)
    done <- true
  }()

  ListenAndServeTLS(":1025", "cert.pem", "key.pem", nil)
  <- done
}
```

```
package main
import . "fmt"
import . "net/http"

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, "hello world")
  })

  done := make(chan bool)
  go func() {
    ListenAndServe(":1024", nil)
    done <- true
  }()

  ListenAndServeTLS(":1025", "cert.pem", "key.pem", nil)
  <- done
}
```

```
package main
import . "fmt"
import . "net/http"

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, "hello world")
  })

  done := make(chan bool)
  go func() {
    ListenAndServe(":1024", nil)
    done <- true
  }()

  ListenAndServeTLS(":1025", "cert.pem", "key.pem", nil)
  <- done
}
```

```
package main
import . "fmt"
import . "net/http"

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, "hello world")
  })

  done := make(chan bool)
  go func() {
    ListenAndServe(":1024", nil)
    done <- true
  }()

  ListenAndServeTLS(":1025", "cert.pem", "key.pem", nil)
  <- done
}
```

```
package main
import . "fmt"
import . "net/http"

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, "hello world")
  })

  done := make(chan bool)
  go func() {
    ListenAndServe(":1024", nil)
    done <- true
  }()

  ListenAndServeTLS(":1025", "cert.pem", "key.pem", nil)
  <- done
}
```

```
package main
import . "fmt"
import . "net/http"

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, "hello world")
  })

  done := make(chan bool)
  go func() {
    ListenAndServe(":1024", nil)
    done <- true
  }()

  ListenAndServeTLS(":1025", "cert.pem", "key.pem", nil)
  <- done
}
```

```
package main
import . "fmt"
import . "net/http"

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, "hello world")
  })

  Spawn(
    func() {
      ListenAndServe(":1024", nil)
    },
    func() {
      ListenAndServeTLS(":1025", "cert.pem", "key.pem", nil)
    },
  )
}
```

```
package main
import . "fmt"
import . "net/http"

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, "hello world")
  })

  Spawn(func() {
    ListenAndServe(":1024", nil)
  })
  Spawn(func() {
    ListenAndServeTLS(":1025", "cert.pem", "key.pem", nil)
  })
}
```

```
package main

func Spawn(f ...func()) {
  done := make(chan bool)
  for _, s := range f {
    go func(server func()) {
      server()
      done <- true
    }(s)
  }

  for l := len(f); l > 0; l-- {
    <- done
  }
}
```

```
package main

func Spawn(f ...func()) {
  done := make(chan bool)
  for _, s := range f {
    go func(server func()) {
      server()
      done <- true
    }(s)
  }

  for l := len(f); l > 0; l-- {
    <- done
  }
}
```

```
package main

func Spawn(f ...func()) {
  done := make(chan bool)
  for _, s := range f {
    go func(server func()) {
      server()
      done <- true
    }(s)
  }

  for l := len(f); l > 0; l-- {
    <- done
  }
}
```

```
package main

func Spawn(f ...func()) {
  done := make(chan bool)
  for _, s := range f {
    go func(server func()) {
      server()
      done <- true
    }(s)
  }

  for l := len(f); l > 0; l-- {
    <- done
  }
}
```

```
package main

func Spawn(f ...func()) {
  done := make(chan bool)
  for _, s := range f {
    go func(server func()) {
      server()
      done <- true
    }(s)
  }

  for l := len(f); l > 0; l-- {
      <- done
    }
  }
```

```
package main

func Spawn(f ...func()) {
  done := make(chan bool)
  for _, s := range f {
    go func(server func()) {
      server()
      done <- true
    }(s)
  }

  for l := len(f); l > 0; l-- {
    <- done
  }
}
```

```
package main

func Spawn(f ...func()) {
  done := make(chan bool)
  for _, s := range f {
    go func(server func()) {
      server()
      done <- true
    }(s)
  }

  for l := len(f); l > 0; l-- {
    <- done
  }
}
```

# waitgroups

```
package main
import . "fmt"
import . "net/http"
import "sync"

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {})

  var servers sync.WaitGroup
  servers.Add(1)
  go func() {
    defer servers.Done()
    ListenAndServe(":1024", nil)
  }()

  servers.Add(1)
  go func() {
    defer servers.Done()
    ListenAndServeTLS(":1025", "cert.pem", "key.pem", nil)
  }()
  servers.Wait()
}
```

```
package main
import . "fmt"
import . "net/http"
import "sync"

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {})

  var servers sync.WaitGroup
  servers.Add(1)
  go func() {
    defer servers.Done()
    ListenAndServe(":1024", nil)
  }()

  servers.Add(1)
  go func() {
    defer servers.Done()
    ListenAndServeTLS(":1025", "cert.pem", "key.pem", nil)
  }()
  servers.Wait()
}
```

```
package main
import . "fmt"
import . "net/http"
import "sync"

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {})

  var servers sync.WaitGroup
  servers.Add(1)
  go func() {
    defer servers.Done()
    ListenAndServe(":1024", nil)
  }()

  servers.Add(1)
  go func() {
    defer servers.Done()
    ListenAndServeTLS(":1025", "cert.pem", "key.pem", nil)
  }()
  servers.Wait()
}
```

```
package main
import . "fmt"
import . "net/http"
import "sync"

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {})

  var servers sync.WaitGroup
  servers.Add(1)
  go func() {
    defer servers.Done()
    ListenAndServe(":1024", nil)
  }()

  servers.Add(1)
  go func() {
    defer servers.Done()
    ListenAndServeTLS(":1025", "cert.pem", "key.pem", nil)
  }()
  servers.Wait()
}
```

```
package main
import . "fmt"
import . "net/http"
import "sync"

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {})

  var servers sync.WaitGroup
  servers.Add(1)
  go func() {
    defer servers.Done()
    ListenAndServe(":1024", nil)
  }()

  servers.Add(1)
  go func() {
    defer servers.Done()
    ListenAndServeTLS(":1025", "cert.pem", "key.pem", nil)
  }()
  servers.Wait()
}
```

```
package main
import . "fmt"
import . "net/http"
import "sync"

var servers sync.WaitGroup

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, "hello world")
  })

  Spawn(func() {
    ListenAndServe(":1024", nil)
  })
  Spawn(func() {
    ListenAndServeTLS(":1025", "cert.pem", "key.pem", nil)
  })
  servers.Wait()
}
```

```
package main
import . "fmt"
import . "net/http"
import "sync"

var servers sync.WaitGroup

func main() {
  HandleFunc("/hello", func(w ResponseWriter, r *Request) {
    w.Header().Set("Content-Type", "text/plain")
    Fprintf(w, "hello world")
  })

  Spawn(func() {
    ListenAndServe(":1024", nil)
  })
  Spawn(func() {
    ListenAndServeTLS(":1025", "cert.pem", "key.pem", nil)
  })
  servers.Wait()
}
```

```
package main

func Spawn(f ...func()) {
  for _, s := range f {
    servers.Add(1)
    go func(server func()) {
      defer servers.Done()
      server()
    }(s)
  }
}
```

```
package main

func Spawn(f ...func()) {
  for _, s := range f {
    servers.Add(1)
    go func(server func()) {
      defer servers.Done()
      server()
    }(s)
  }
}
```

# tcp server

```
package main
import . "fmt"
import "net"

func main() {
  if listener, e := net.Listen("tcp", ":1024"); e == nil {
    for {
      if connection, e := listener.Accept(); e == nil {
        go func(c net.Conn) {
          defer c.Close()
          Fprintln(c, "hello world")
        }(connection)
      }
    }
  }
}
```

```
package main
import . "fmt"
import "net"

func main() {
  if listener, e := net.Listen("tcp", ":1024"); e == nil {
    for {
      if connection, e := listener.Accept(); e == nil {
        go func(c net.Conn) {
          defer c.Close()
          Fprintln(c, "hello world")
        }(connection)
      }
    }
  }
}
```

```
package main
import . "fmt"
import "net"

func main() {
  if listener, e := net.Listen("tcp", ":1024"); e == nil {
    for {
      if connection, e := listener.Accept(); e == nil {
        go func(c net.Conn) {
          defer c.Close()
          Fprintln(c, "hello world")
        }(connection)
      }
    }
  }
}
```

```
package main
import . "fmt"
import "net"

func main() {
  if listener, e := net.Listen("tcp", ":1024"); e == nil {
    for {
      if connection, e := listener.Accept(); e == nil {
        go func(c net.Conn) {
          defer c.Close()
          Fprintln(c, "hello world")
        }(connection)
      }
    }
  }
}
```

```
package main
import . "fmt"
import "net"

func main() {
  if listener, e := net.Listen("tcp", ":1024"); e == nil {
    for {
      if connection, e := listener.Accept(); e == nil {
        go func(c net.Conn) {
          defer c.Close()
          Fprintln(c, "hello world")
        }(connection)
      }
    }
  }
}
```

```
package main
import . "fmt"
import "net"

func main() {
  Listen("tcp", ":1024", func(c net.Conn) {
    defer c.Close()
    Fprintln(c, "hello world")
  })
}

func Listen(p, a string, f func(net.Conn)) (e error) {
  var listener net.Listener
  if listener, e = net.Listen(p, a); e == nil {
    for {
      if connection, e := listener.Accept(); e == nil {
        go f(connection)
      }
    }
  }
  return
}
```

```
package main
import . "fmt"
import "net"

func main() {
  Listen("tcp", ":1024", func(c net.Conn) {
    defer c.Close()
    Fprintln(c, "hello world")
  })
}

func Listen(p, a string, f func(net.Conn)) (e error) {
  var listener net.Listener
  if listener, e = net.Listen(p, a); e == nil {
    for {
      if connection, e := listener.Accept(); e == nil {
        go f(connection)
      }
    }
  }
  return
}
```

```
package main
import . "fmt"
import "net"

func main() {
  Listen("tcp", ":1024", func(c net.Conn) {
    defer c.Close()
    Fprintln(c, "hello world")
  })
}

func Listen(p, a string, f func(net.Conn)) (e error) {
  var listener net.Listener
  if listener, e = net.Listen(p, a); e == nil {
    for {
      if connection, e := listener.Accept(); e == nil {
        go f(connection)
      }
    }
  }
  return
}
```

```
package main
import . "fmt"
import "net"

func main() {
  Listen("tcp", ":1024", func(c net.Conn) {
    defer c.Close()
    Fprintln(c, "hello world")
  })
}

func Listen(p, a string, f func(net.Conn)) (e error) {
  var listener net.Listener
  if listener, e = net.Listen(p, a); e == nil {
    for {
      if connection, e := listener.Accept(); e == nil {
        go f(connection)
      }
    }
  }
  return
}
```

# tcp client

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  if c, e := net.Dial("tcp", ":1024"); e == nil {
    defer c.Close()
    if m, e := bufio.NewReader(c).ReadString('\n'); e == nil {
      Printf(m)
    }
  }
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  if c, e := net.Dial("tcp", ":1024"); e == nil {
    defer c.Close()
    if m, e := bufio.NewReader(c).ReadString('\n'); e == nil {
      Printf(m)
    }
  }
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  if c, e := net.Dial("tcp", ":1024"); e == nil {
    defer c.Close()
    if m, e := bufio.NewReader(c).ReadString('\n'); e == nil {
      Printf(m)
    }
  }
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  if c, e := net.Dial("tcp", ":1024"); e == nil {
    defer c.Close()
    if m, e := bufio.NewReader(c).ReadString('\n'); e == nil {
      Printf(m)
    }
  }
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  if c, e := net.Dial("tcp", ":1024"); e == nil {
    defer c.Close()
    if m, e := bufio.NewReader(c).ReadString('\n'); e == nil {
      Printf(m)
    }
  }
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  if c, e := net.Dial("tcp", ":1024"); e == nil {
    defer c.Close()
    if m, e := bufio.NewReader(c).ReadString('\n'); e == nil {
      Printf(m)
    }
  }
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  Dial("tcp", ":1024", func(c net.Conn) {
    if m, e := bufio.NewReader(c).ReadString('\n'); e == nil {
      Printf(m)
    }
  })
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  Dial("tcp", ":1024", func(c net.Conn) {
    if m, e := bufio.NewReader(c).ReadString('\n'); e == nil {
      Printf(m)
    }
  })
}

func Dial(p, a string, f func(net.Conn)) (e error) {
  var c net.Conn
  if c, e = net.Dial(p, a); e == nil {
    defer c.Close()
    f(c)
  }
  return
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  Dial("tcp", ":1024", func(c net.Conn) {
    if m, e := bufio.NewReader(c).ReadString('\n'); e == nil {
      Printf(m)
    }
  })
}
```

```
func Dial(p, a string, f func(net.Conn)) (e error) {
  var c net.Conn
  if c, e = net.Dial(p, a); e == nil {
    defer c.Close()
    f(c)
  }
  return
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  Dial("tcp", ":1024", func(c net.Conn) {
    if m, e := bufio.NewReader(c).ReadString('\n'); e == nil {
      Printf(m)
    }
  })
}
```

```
func Dial(p, a string, f func(net.Conn)) (e error) {
  var c net.Conn
  if c, e = net.Dial(p, a); e == nil {
    defer c.Close()
    f(c)
  }
  return
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  Dial("tcp", ":1024", func(c net.Conn) {
    if m, e := bufio.NewReader(c).ReadString('\n'); e == nil {
      Printf(m)
    }
  })
}

func Dial(p, a string, f func(net.Conn)) (e error) {
  var c net.Conn
  if c, e = net.Dial(p, a); e == nil {
    defer c.Close()
    f(c)
  }
  return
}
```

# tcp/tls server

```
package main
import "crypto/tls"
import . "fmt"

func main() {
  Listen(":1025", ConfigTLS("scert", "skey"), func(c *tls.Conn) {
    Fprintln(c, "hello world")
  })
}
```

```
package main
import "crypto/tls"
import . "fmt"

func main() {
  Listen(":1025", ConfigTLS("scert", "skey"), func(c *tls.Conn) {
    Fprintln(c, "hello world")
  })
}
```

```
package main
import "crypto/tls"
import . "fmt"

func main() {
  Listen(":1025", ConfigTLS("scert", "skey"), func(c *tls.Conn) {
    Fprintln(c, "hello world")
  })
}
```

```
package main
import "crypto/rand"
import "crypto/tls"

func ConfigTLS(c, k string) (r *tls.Config) {
  if cert, e := tls.LoadX509KeyPair(c, k); e == nil {
    r = &tls.Config{
      Certificates: []tls.Certificate{ cert },
      Rand: rand.Reader,
    }
  }
  return
}
```

```
package main
import "crypto/rand"
import "crypto/tls"

func ConfigTLS(c, k string) (r *tls.Config) {
  if cert, e := tls.LoadX509KeyPair(c, k); e == nil {
    r = &tls.Config{
      Certificates: []tls.Certificate{ cert },
      Rand: rand.Reader,
    }
  }
  return
}
```

```
package main
import "crypto/rand"
import "crypto/tls"

func ConfigTLS(c, k string) (r *tls.Config) {
  if cert, e := tls.LoadX509KeyPair(c, k); e == nil {
    r = &tls.Config{
      Certificates: []tls.Certificate{ cert },
      Rand: rand.Reader,
    }
  }
  return
}
```

```
package main
import "crypto/rand"
import "crypto/tls"

func ConfigTLS(c, k string) (r *tls.Config) {
  if cert, e := tls.LoadX509KeyPair(c, k); e == nil {
    r = &tls.Config{
      Certificates: []tls.Certificate{ cert },
      Rand: rand.Reader,
    }
  }
  return
}
```

```
package main
import "crypto/rand"
import "crypto/tls"

func ConfigTLS(c, k string) (r *tls.Config) {
  if cert, e := tls.LoadX509KeyPair(c, k); e == nil {
    r = &tls.Config{
      Certificates: []tls.Certificate{ cert },
      Rand: rand.Reader,
    }
  }
  return
}
```

```
package main
import "crypto/tls"

func Listen(a string, conf *tls.Config, f func(*tls.Conn)) {
  if listener, e := tls.Listen("tcp", a, conf); e == nil {
    for {
      if connection, e := listener.Accept(); e == nil {
        go func(c *tls.Conn) {
          defer c.Close()
          f(c)
        }(connection.(*tls.Conn))
      }
    }
  }
}
```

```
package main
import "crypto/tls"

func Listen(a string, conf *tls.Config, f func(*tls.Conn)) {
  if listener, e := tls.Listen("tcp", a, conf); e == nil {
    for {
      if connection, e := listener.Accept(); e == nil {
        go func(c *tls.Conn) {
          defer c.Close()
          f(c)
        }(connection.(*tls.Conn))
      }
    }
  }
}
```

```
package main
import "crypto/tls"

func Listen(a string, conf *tls.Config, f func(*tls.Conn)) {
  if listener, e := tls.Listen("tcp", a, conf); e == nil {
    for {
      if connection, e := listener.Accept(); e == nil {
        go func(c *tls.Conn) {
          defer c.Close()
          f(c)
        }(connection.(*tls.Conn))
      }
    }
  }
}
```

```
package main
import "crypto/tls"

func Listen(a string, conf *tls.Config, f func(*tls.Conn)) {
  if listener, e := tls.Listen("tcp", a, conf); e == nil {
    for {
      if connection, e := listener.Accept(); e == nil {
        go func(c *tls.Conn) {
          defer c.Close()
          f(c)
        }(connection.(*tls.Conn))
      }
    }
  }
}
```

```
package main
import “crypto/tls"
import "net"

func Listen(a string, conf *tls.Config, f func(net.Conn)) {
  if listener, e := tls.Listen("tcp", a, conf); e == nil {
    for {
      if connection, e := listener.Accept(); e == nil {
        go func(c net.Conn) {
          defer c.Close()
          f(c)
        }(connection)
      }
    }
  }
}
```

```
package main
import "crypto/tls"
import "net"

func Listen(a string, conf *tls.Config, f func(net.Conn)) {
  if listener, e := tls.Listen("tcp", a, conf); e == nil {
    for {
      if connection, e := listener.Accept(); e == nil {
        go func(c net.Conn) {
          defer c.Close()
          f(c)
        }(connection)
      }
    }
  }
}
```

# tcp/tls client

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  Dial(":1025", ConfigTLS("ccert", "ckey"), func(c net.Conn) {
    if m, e := bufio.NewReader(c).ReadString('\n'); e == nil {
      Printf(m)
    }
  })
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  Dial(":1025", ConfigTLS("ccert", "ckey"), func(c net.Conn) {
    if m, e := bufio.NewReader(c).ReadString('\n'); e == nil {
      Printf(m)
    }
  })
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  Dial(":1025", ConfigTLS("ccert", "ckey"), func(c net.Conn) {
    if m, e := bufio.NewReader(c).ReadString('\n'); e == nil {
      Printf(m)
    }
  })
}
```

```
package main
import "crypto/tls"

func ConfigTLS(c, k string) (r *tls.Config) {
  if cert, e := tls.LoadX509KeyPair(c, k); e == nil {
    r = &tls.Config{
      Certificates: []tls.Certificate{ cert },
      InsecureSkipVerify: true,
    }
  }
  return
}
```

```
package main
import "crypto/tls"

func ConfigTLS(c, k string) (r *tls.Config) {
  if cert, e := tls.LoadX509KeyPair(c, k); e == nil {
    r = &tls.Config{
      Certificates: []tls.Certificate{ cert },
      InsecureSkipVerify: true,
    }
  }
  return
}
```

```
package main
import "crypto/tls"

func ConfigTLS(c, k string) (r *tls.Config) {
  if cert, e := tls.LoadX509KeyPair(c, k); e == nil {
    r = &tls.Config{
      Certificates: []tls.Certificate{ cert },
      InsecureSkipVerify: true,
    }
  }
  return
}
```

```
package main
import “crypto/tls"
import "net"

func Dial(a string, conf *tls.Config, f func(net.Conn)) {
  if c, e := tls.Dial("tcp", a, conf); e == nil {
    defer c.Close()
    f(c)
  }
}
```

```
package main
import “crypto/tls"
import "net"

func Dial(a string, conf *tls.Config, f func(net.Conn)) {
  if c, e := tls.Dial("tcp", a, conf); e == nil {
    defer c.Close()
    f(c)
  }
}
```

# udp serve

```
package main
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World\n")
  Listen(":1024", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    c.WriteToUDP(HELLO_WORLD, a)
  })
}

func Listen(a string, f func(*net.UDPConn, *net.UDPAddr, []byte)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.ListenUDP("udp", address); e == nil {
      for b := make([]byte, 1024); ; b = make([]byte, 1024) {
        if n, client, e := conn.ReadFromUDP(b); e == nil {
          go f(conn, client, b[:n])
        }
      }
    }
  }
  return
}
```

```
package main
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World\n”)
  Listen(":1024", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    c.WriteToUDP(HELLO_WORLD, a)
  })
}

func Listen(a string, f func(*net.UDPConn, *net.UDPAddr, []byte)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.ListenUDP("udp", address); e == nil {
      for b := make([]byte, 1024); ; b = make([]byte, 1024) {
        if n, client, e := conn.ReadFromUDP(b); e == nil {
          go f(conn, client, b[:n])
        }
      }
    }
  }
  return
}
```

```
package main
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World\n")
  Listen(":1024", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    c.WriteToUDP(HELLO_WORLD, a)
  })
}

func Listen(a string, f func(*net.UDPConn, *net.UDPAddr, []byte)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.ListenUDP("udp", address); e == nil {
      for b := make([]byte, 1024); ; b = make([]byte, 1024) {
        if n, client, e := conn.ReadFromUDP(b); e == nil {
          go f(conn, client, b[:n])
        }
      }
    }
  }
  return
}
```

```
package main
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World\n")
  Listen(":1024", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    c.WriteToUDP(HELLO_WORLD, a)
  })
}

func Listen(a string, f func(*net.UDPConn, *net.UDPAddr, []byte)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.ListenUDP("udp", address); e == nil {
      for b := make([]byte, 1024); ; b = make([]byte, 1024) {
        if n, client, e := conn.ReadFromUDP(b); e == nil {
          go f(conn, client, b[:n])
        }
      }
    }
  }
  return
}
```

```
package main
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World\n")
  Listen(":1024", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    c.WriteToUDP(HELLO_WORLD, a)
  })
}

func Listen(a string, f func(*net.UDPConn, *net.UDPAddr, []byte)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.ListenUDP("udp", address); e == nil {
      for b := make([]byte, 1024); ; b = make([]byte, 1024) {
        if n, client, e := conn.ReadFromUDP(b); e == nil {
          go f(conn, client, b[:n])
        }
      }
    }
  }
  return
}
```

```
package main
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World\n")
  Listen(":1024", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    c.WriteToUDP(HELLO_WORLD, a)
  })
}

func Listen(a string, f func(*net.UDPConn, *net.UDPAddr, []byte)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.ListenUDP("udp", address); e == nil {
      for b := make([]byte, 1024); ; b = make([]byte, 1024) {
        if n, client, e := conn.ReadFromUDP(b); e == nil {
          go f(conn, client, b[:n])
        }
      }
    }
  }
  return
}
```

```
package main
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World\n")
  Listen(":1024", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    c.WriteToUDP(HELLO_WORLD, a)
  })
}

func Listen(a string, f func(*net.UDPConn, *net.UDPAddr, []byte)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.ListenUDP("udp", address); e == nil {
      for b := make([]byte, 1024); ; b = make([]byte, 1024) {
        if n, client, e := conn.ReadFromUDP(b); e == nil {
          go f(conn, client, b[:n])
        }
      }
    }
  }
  return
}
```

```
package main
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World\n")
  Listen(":1024", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    c.WriteToUDP(HELLO_WORLD, a)
  })
}

func Listen(a string, f func(*net.UDPConn, *net.UDPAddr, []byte)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.ListenUDP("udp", address); e == nil {
      for b := make([]byte, 1024); ; b = make([]byte, 1024) {
        if n, client, e := conn.ReadFromUDP(b); e == nil {
          go f(conn, client, b[:n])
        }
      }
    }
  }
  return
}
```

```
package main
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World\n")
  Listen(":1024", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    c.WriteToUDP(HELLO_WORLD, a)
  })
}

func Listen(a string, f func(*net.UDPConn, *net.UDPAddr, []byte)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.ListenUDP("udp", address); e == nil {
      for b := make([]byte, 1024); ; b = make([]byte, 1024) {
        if n, client, e := conn.ReadFromUDP(b); e == nil {
          go f(conn, client, b[:n])
        }
      }
    }
  }
  return
}
```

```
package main
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World\n")
  Listen(":1024", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    c.WriteToUDP(HELLO_WORLD, a)
  })
}

func Listen(a string, f func(*net.UDPConn, *net.UDPAddr, []byte)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.ListenUDP("udp", address); e == nil {
      for b := make([]byte, 1024); ; b = make([]byte, 1024) {
        if n, client, e := conn.ReadFromUDP(b); e == nil {
          go f(conn, client, b[:n])
        }
      }
    }
  }
  return
}
```

```
package main
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World\n")
  Listen(":1024", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    c.WriteToUDP(HELLO_WORLD, a)
  })
}

func Listen(a string, f func(*net.UDPConn, *net.UDPAddr, []byte)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.ListenUDP("udp", address); e == nil {
      for b := make([]byte, 1024); ; b = make([]byte, 1024) {
        if n, client, e := conn.ReadFromUDP(b); e == nil {
          go f(conn, client, b[:n])
        }
      }
    }
  }
  return
}
```

```
package main
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World\n")
  Listen(":1024", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    c.WriteToUDP(HELLO_WORLD, a)
  })
}

func Listen(a string, f func(*net.UDPConn, *net.UDPAddr, []byte)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.ListenUDP("udp", address); e == nil {
      for b := make([]byte, 1024); ; b = make([]byte, 1024) {
        if n, client, e := conn.ReadFromUDP(b); e == nil {
          go f(conn, client, b[:n])
        }
      }
    }
  }
  return
}
```

```
package main
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World\n")
  Listen(":1024", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    c.WriteToUDP(HELLO_WORLD, a)
  })
}

func Listen(a string, f func(*net.UDPConn, *net.UDPAddr, []byte)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.ListenUDP("udp", address); e == nil {
      for b := make([]byte, 1024); ; b = make([]byte, 1024) {
        if n, client, e := conn.ReadFromUDP(b); e == nil {
          go f(conn, client, b[:n])
        }
      }
    }
  }
  return
}
```

```
package main
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World\n")
  Listen(":1024", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    c.WriteToUDP(HELLO_WORLD, a)
  })
}

func Listen(a string, f func(*net.UDPConn, *net.UDPAddr, []byte)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.ListenUDP("udp", address); e == nil {
      for b := make([]byte, 1024); ; b = make([]byte, 1024) {
        if n, client, e := conn.ReadFromUDP(b); e == nil {
          go f(conn, client, b[:n])
        }
      }
    }
  }
  return
}
```

# udp request

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  Dial(":1024", func(conn net.Conn) {
    if _, e := conn.Write([]byte("\n")); e == nil {
      if m, e := bufio.NewReader(conn).ReadString('\n'); e == nil {
        Printf("%v", m)
      }
    }
  })
}

func Dial(a string, f func(net.Conn)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.DialUDP("udp", nil, address); e == nil {
      defer conn.Close()
      f(conn)
    }
  }
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  Dial(":1024", func(conn net.Conn) {
    if _, e := conn.Write([]byte("\n")); e == nil {
      if m, e := bufio.NewReader(conn).ReadString('\n'); e == nil {
        Printf("%v", m)
      }
    }
  })
}

func Dial(a string, f func(net.Conn)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.DialUDP("udp", nil, address); e == nil {
      defer conn.Close()
      f(conn)
    }
  }
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  Dial(":1024", func(conn net.Conn) {
    if _, e := conn.Write([]byte("\n")); e == nil {
      if m, e := bufio.NewReader(conn).ReadString('\n'); e == nil {
        Printf("%v", m)
      }
    }
  })
}

func Dial(a string, f func(net.Conn)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.DialUDP("udp", nil, address); e == nil {
      defer conn.Close()
      f(conn)
    }
  }
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  Dial(":1024", func(conn net.Conn) {
    if _, e := conn.Write([]byte("\n")); e == nil {
      if m, e := bufio.NewReader(conn).ReadString('\n'); e == nil {
        Printf("%v", m)
      }
    }
  })
}

func Dial(a string, f func(net.Conn)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.DialUDP("udp", nil, address); e == nil {
      defer conn.Close()
      f(conn)
    }
  }
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  Dial(":1024", func(conn net.Conn) {
    if _, e := conn.Write([]byte("\n")); e == nil {
      if m, e := bufio.NewReader(conn).ReadString('\n'); e == nil {
        Printf("%v", m)
      }
    }
  })
}

func Dial(a string, f func(net.Conn)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.DialUDP("udp", nil, address); e == nil {
      defer conn.Close()
      f(conn)
    }
  }
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  Dial(":1024", func(conn net.Conn) {
    if _, e := conn.Write([]byte("\n")); e == nil {
      if m, e := bufio.NewReader(conn).ReadString('\n'); e == nil {
        Printf("%v", m)
      }
    }
  })
}

func Dial(a string, f func(net.Conn)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.DialUDP("udp", nil, address); e == nil {
      defer conn.Close()
      f(conn)
    }
  }
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  Dial(":1024", func(conn net.Conn) {
    if _, e := conn.Write([]byte("\n")); e == nil {
      if m, e := bufio.NewReader(conn).ReadString('\n'); e == nil {
        Printf("%v", m)
      }
    }
  })
}

func Dial(a string, f func(net.Conn)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.DialUDP("udp", nil, address); e == nil {
      defer conn.Close()
      f(conn)
    }
  }
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  Dial(":1024", func(conn net.Conn) {
    if _, e := conn.Write([]byte("\n")); e == nil {
      if m, e := bufio.NewReader(conn).ReadString('\n'); e == nil {
        Printf("%v", m)
      }
    }
  })
}

func Dial(a string, f func(net.Conn)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.DialUDP("udp", nil, address); e == nil {
      defer conn.Close()
      f(conn)
    }
  }
}
```

```
package main
import "bufio"
import . "fmt"
import "net"

func main() {
  Dial(":1024", func(conn net.Conn) {
    if _, e := conn.Write([]byte("\n")); e == nil {
      if m, e := bufio.NewReader(conn).ReadString('\n'); e == nil {
        Printf("%v", m)
      }
    }
  })
}

func Dial(a string, f func(net.Conn)) {
  if address, e := net.ResolveUDPAddr("udp", a); e == nil {
    if conn, e := net.DialUDP("udp", nil, address); e == nil {
      defer conn.Close()
      f(conn)
    }
  }
}
```

# aes encrypt

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

const AES_KEY = "0123456789012345"

func main() {
  Listen(":1025", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    if m, e := Encrypt("Hello World", AES_KEY); e == nil {
      c.WriteToUDP(m, a)
    }
  })
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

const AES_KEY = "0123456789012345"

func main() {
  Listen(":1025", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    if m, e := Encrypt("Hello World", AES_KEY); e == nil {
      c.WriteToUDP(m, a)
    }
    return
  })
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

const AES_KEY = "0123456789012345"

func main() {
  Listen(":1025", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    if m, e := Encrypt("Hello World", AES_KEY); e == nil {
      c.WriteToUDP(m, a)
    }
    return
  })
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

const AES_KEY = "0123456789012345"

func main() {
  Listen(":1025", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    if m, e := Encrypt("Hello World", AES_KEY); e == nil {
      c.WriteToUDP(m, a)
    }
    return
  })
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

const AES_KEY = "0123456789012345"

func main() {
  Listen(":1025", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    if m, e := Encrypt("Hello World", AES_KEY); e == nil {
      c.WriteToUDP(m, a)
    }
    return
  })
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func Encrypt(m, k string) (o []byte, e error) {
  if o, e = Quantise([]byte(m)); e == nil {
    var b cipher.Block
    if b, e = aes.NewCipher([]byte(k)); e == nil {
      var iv []byte
      if iv, e = IV(); e == nil {
        c := cipher.NewCBCEncrypter(b, iv)
        c.CryptBlocks(o, o)
        o = append(iv, o...)
      }
    }
  }
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func Encrypt(m, k string) (o []byte, e error) {
  if o, e = Quantise([]byte(m)); e == nil {
    var b cipher.Block
    if b, e = aes.NewCipher([]byte(k)); e == nil {
      var iv []byte
      if iv, e = IV(); e == nil {
        c := cipher.NewCBCEncrypter(b, iv)
        c.CryptBlocks(o, o)
        o = append(iv, o...)
      }
    }
  }
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func Encrypt(m, k string) (o []byte, e error) {
  if o, e = Quantise(m); e == nil {
    var b cipher.Block
    if b, e = aes.NewCipher([]byte(k)); e == nil {
      var iv []byte
      if iv, e = IV(); e == nil {
        c := cipher.NewCBCEncrypter(b, iv)
        c.CryptBlocks(o, o)
        o = append(iv, o...)
      }
    }
  }
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func Quantise(m string) (b []byte, e error) {
  b = append(b, m...)
  if p := len(b) % aes.BlockSize; p != 0 {
    p = aes.BlockSize - p
    //  this is insecure and inflexible as we're padding with NUL!
    b = append(b, make([]byte, p)...)
  }
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func Quantise(m string) (b []byte, e error) {
  b = append(b, m...)
  if p := len(b) % aes.BlockSize; p != 0 {
    p = aes.BlockSize - p
    //  this is insecure and inflexible as we're padding with NUL!
    b = append(b, make([]byte, p)...)
  }
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func Quantise(m string) (b []byte, e error) {
  b = append(b, m...)
  if p := len(b) % aes.BlockSize; p != 0 {
    p = aes.BlockSize - p
    //  this is insecure and inflexible as we're padding with NUL!
    b = append(b, make([]byte, p)...)
  }
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func Quantise(m string) (b []byte, e error) {
  b = append(b, m...)
  if p := len(b) % aes.BlockSize; p != 0 {
    p = aes.BlockSize - p
    //  this is insecure and inflexible as we're padding with NUL!
    b = append(b, make([]byte, p)...)
  }
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func Quantise(m string) (b []byte, e error) {
  b = append(b, m...)
  if p := len(b) % aes.BlockSize; p != 0 {
    p = aes.BlockSize - p
    //  this is insecure and inflexible as we're padding with NUL!
    b = append(b, make([]byte, p)...)
  }
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func Quantise(m string) (b []byte, e error) {
  b = append(b, m...)
  if p := len(b) % aes.BlockSize; p != 0 {
    p = aes.BlockSize - p
    //  this is insecure and inflexible as we're padding with NUL!
    b = append(b, make([]byte, p)...)
  }
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func Quantise(m string) (b []byte, e error) {
  b = append(b, m...)
  if p := len(b) % aes.BlockSize; p != 0 {
    p = aes.BlockSize - p
    //  this is insecure and inflexible as we're padding with NUL!
    b = append(b, make([]byte, p)...)
  }
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func Quantise(m string) (b []byte, e error) {
  b = append(b, m...)
  if p := len(b) % aes.BlockSize; p != 0 {
    p = aes.BlockSize - p
    //  this is insecure and inflexible as we're padding with NUL!
    b = append(b, make([]byte, p)...)
  }
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func Encrypt(m, k string) (o []byte, e error) {
  if o, e = Quantise(m); e == nil {
    var b cipher.Block
    if b, e = aes.NewCipher([]byte(k)); e == nil {
      var iv []byte
      if iv, e = IV(); e == nil {
        c := cipher.NewCBCEncrypter(b, iv)
        c.CryptBlocks(o, o)
        o = append(iv, o...)
      }
    }
  }
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func Encrypt(m, k string) (o []byte, e error) {
  if o, e = Quantise(m); e == nil {
    var b cipher.Block
    if b, e = aes.NewCipher([]byte(k)); e == nil {
      var iv []byte
      if iv, e = IV(); e == nil {
        c := cipher.NewCBCEncrypter(b, iv)
        c.CryptBlocks(o, o)
        o = append(iv, o...)
      }
    }
  }
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func Encrypt(m, k string) (o []byte, e error) {
  if o, e = Quantise(m); e == nil {
    var b cipher.Block
    if b, e = aes.NewCipher([]byte(k)); e == nil {
      var iv []byte
      if iv, e = IV(); e == nil {
        c := cipher.NewCBCEncrypter(b, iv)
        c.CryptBlocks(o, o)
        o = append(iv, o...)
      }
    }
  }
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func Encrypt(m, k string) (o []byte, e error) {
  if o, e = Quantise(m); e == nil {
    var b cipher.Block
    if b, e = aes.NewCipher([]byte(k)); e == nil {
      var iv []byte
      if iv, e = IV(); e == nil {
        c := cipher.NewCBCEncrypter(b, iv)
        c.CryptBlocks(o, o)
        o = append(iv, o...)
      }
    }
  }
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func IV() (b []byte, e error) {
  b = make([]byte, aes.BlockSize)
  _, e = rand.Read(b)
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func IV() (b []byte, e error) {
  b = make([]byte, aes.BlockSize)
  _, e = rand.Read(b)
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func Encrypt(m, k string) (o []byte, e error) {
  if o, e = Quantise(m); e == nil {
    var b cipher.Block
    if b, e = aes.NewCipher([]byte(k)); e == nil {
      var iv []byte
      if iv, e = IV(); e == nil {
        c := cipher.NewCBCEncrypter(b, iv)
        c.CryptBlocks(o, o)
        o = append(iv, o...)
      }
    }
  }
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func Encrypt(m, k string) (o []byte, e error) {
  if o, e = Quantise(m); e == nil {
    var b cipher.Block
    if b, e = aes.NewCipher([]byte(k)); e == nil {
      var iv []byte
      if iv, e = IV(); e == nil {
        c := cipher.NewCBCEncrypter(b, iv)
        c.CryptBlocks(o, o)
        o = append(iv, o...)
      }
    }
  }
  return
}
```

```
package main
import "crypto/aes"
import "crypto/cipher"
import "crypto/rand"
import "net"

func Encrypt(m, k string) (o []byte, e error) {
  if o, e = Quantise(m); e == nil {
    var b cipher.Block
    if b, e = aes.NewCipher([]byte(k)); e == nil {
      var iv []byte
      if iv, e = IV(); e == nil {
        c := cipher.NewCBCEncrypter(b, iv)
        c.CryptBlocks(o, o)
        o = append(iv, o...)
      }
    }
  }
  return
}
```

# aes decrypt

```
package main
import "crypto/cipher"
import "crypto/aes"
import . "fmt"
import "net"

const AES_KEY = "0123456789012345"

func main() {
  Dial(":1025", func(c *net.UDPConn) {
    c.Write(make([]byte, 1))
    if m, e := ReadStream(c); e == nil {
      if m, e := Decrypt(m, AES_KEY); e == nil {
        Println(string(m))
      }
    }
  })
}
```

```
package main
import "crypto/cipher"
import "crypto/aes"
import . "fmt"
import "net"

const AES_KEY = "0123456789012345"

func main() {
  Dial(":1025", func(c *net.UDPConn) {
    c.Write(make([]byte, 1))
    if m, e := ReadStream(c); e == nil {
      if m, e := Decrypt(m, AES_KEY); e == nil {
        Println(string(m))
      }
    }
  })
}
```

```
package main
import "crypto/cipher"
import "crypto/aes"
import . "fmt"
import "net"

const AES_KEY = "0123456789012345"

func main() {
  Dial(":1025", func(c *net.UDPConn) {
    c.Write(make([]byte, 1))
    if m, e := ReadStream(c); e == nil {
      if m, e := Decrypt(m, AES_KEY); e == nil {
        Println(string(m))
      }
    }
  })
}
```

```
package main
import "crypto/cipher"
import "crypto/aes"
import . "fmt"
import "net"

func Decrypt(m []byte, k string) (r string, e error) {
  var b cipher.Block
  if b, e = aes.NewCipher([]byte(k)); e == nil {
    var iv []byte
    iv, m = Unpack(m)
    c := cipher.NewCBCDecrypter(b, iv)
    c.CryptBlocks(m, m)
    r = Dequantise(m)
  }
  return
}
```

```
package main
import "crypto/cipher"
import "crypto/aes"
import . "fmt"
import "net"

func Decrypt(m []byte, k string) (r string, e error) {
  var b cipher.Block
  if b, e = aes.NewCipher([]byte(k)); e == nil {
    var iv []byte
    iv, m = Unpack(m)
    c := cipher.NewCBCDecrypter(b, iv)
    c.CryptBlocks(m, m)
    r = Dequantise(m)
  }
  return
}
```

```
package main
import "crypto/cipher"
import "crypto/aes"
import . "fmt"
import "net"

func Decrypt(m []byte, k string) (r string, e error) {
  var b cipher.Block
  if b, e = aes.NewCipher([]byte(k)); e == nil {
    var iv []byte
    iv, m = Unpack(m)
    c := cipher.NewCBCDecrypter(b, iv)
    c.CryptBlocks(m, m)
    r = Dequantise(m)
  }
  return
}
```

```
package main
import "crypto/cipher"
import "crypto/aes"
import . "fmt"
import "net"

func Unpack(m []byte) (iv, r []byte) {
  return m[:aes.BlockSize], m[aes.BlockSize:]
}
```

```
package main
import "crypto/cipher"
import "crypto/aes"
import . "fmt"
import "net"

func Unpack(m []byte) (iv, r []byte) {
  return m[:aes.BlockSize], m[aes.BlockSize:]
}
```

```
package main
import "crypto/cipher"
import "crypto/aes"
import . "fmt"
import "net"

func Unpack(m []byte) (iv, r []byte) {
  return m[:aes.BlockSize], m[aes.BlockSize:]
}
```

```
package main
import "crypto/cipher"
import "crypto/aes"
import . "fmt"
import "net"

func Decrypt(m []byte, k string) (r string, e error) {
  var b cipher.Block
  if b, e = aes.NewCipher([]byte(k)); e == nil {
    var iv []byte
    iv, m = Unpack(m)
    c := cipher.NewCBCDecrypter(b, iv)
    c.CryptBlocks(m, m)
    r = Dequantise(m)
  }
  return
}
```

```
package main
import "crypto/cipher"
import "crypto/aes"
import . "fmt"
import "net"

func Decrypt(m []byte, k string) (r string, e error) {
  var b cipher.Block
  if b, e = aes.NewCipher([]byte(k)); e == nil {
    var iv []byte
    iv, m = Unpack(m)
    c := cipher.NewCBCDecrypter(b, iv)
    c.CryptBlocks(m, m)
    r = Dequantise(m)
  }
  return
}
```

```
package main
import "crypto/cipher"
import "crypto/aes"
import . "fmt"
import "net"

func Decrypt(m []byte, k string) (r string, e error) {
  var b cipher.Block
  if b, e = aes.NewCipher([]byte(k)); e == nil {
    var iv []byte
    iv, m = Unpack(m)
    c := cipher.NewCBCDecrypter(b, iv)
    c.CryptBlocks(m, m)
    r = Dequantise(m)
  }
  return
}
```

```
package main
import "crypto/cipher"
import "crypto/aes"
import . "fmt"
import "net"

func Dequantise(m []byte) string {
  var i int
  for i = len(m) - 1; i > 0 && m[i] == 0; i-- {}
  return string(m[:i + 1])
}
```

```
package main
import "crypto/cipher"
import "crypto/aes"
import . "fmt"
import "net"

func Dequantise(m []byte) string {
  var i int
  for i = len(m) - 1; i > 0 && m[i] == 0; i-- {}
  return string(m[:i + 1])
}
```

```
package main
import "crypto/cipher"
import "crypto/aes"
import . "fmt"
import "net"

func Dequantise(m []byte) string {
  var i int
  for i = len(m) - 1; i > 0 && m[i] == 0; i-- {}
  return string(m[:i + 1])
}
```

# rsa encrypt

```
package main
import . "bytes"
import "crypto/rsa"
import "encoding/gob"
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World")
  RSA_LABEL := []byte("served")
  Listen(":1025", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    var key rsa.PublicKey
    if e := gob.NewDecoder(NewBuffer(b)).Decode(&key); e == nil {
      if m, e := Encrypt(&key, HELLO_WORLD, RSA_LABEL); e == nil {
        c.WriteToUDP(m, a)
      }
    }
    return
  })
}
```

```
package main
import . "bytes"
import "crypto/rsa"
import "encoding/gob"
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World")
  RSA_LABEL := []byte("served")
  Listen(":1025", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    var key rsa.PublicKey
    if e := gob.NewDecoder(NewBuffer(b)).Decode(&key); e == nil {
      if m, e := Encrypt(&key, HELLO_WORLD, RSA_LABEL); e == nil {
        c.WriteToUDP(m, a)
      }
    }
    return
  })
}
```

```
package main
import . "bytes"
import "crypto/rsa"
import "encoding/gob"
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World")
  RSA_LABEL := []byte("served")
  Listen(":1025", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    var key rsa.PublicKey
    if e := gob.NewDecoder(NewBuffer(b)).Decode(&key); e == nil {
      if m, e := Encrypt(&key, HELLO_WORLD, RSA_LABEL); e == nil {
        c.WriteToUDP(m, a)
      }
    }
    return
  })
}
```

```
package main
import . "bytes"
import "crypto/rsa"
import "encoding/gob"
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World")
  RSA_LABEL := []byte("served")
  Listen(":1025", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    var key rsa.PublicKey
    if e := gob.NewDecoder(NewBuffer(b)).Decode(&key); e == nil {
      if m, e := Encrypt(&key, HELLO_WORLD, RSA_LABEL); e == nil {
        c.WriteToUDP(m, a)
      }
    }
    return
  })
}
```

```
package main
import . "bytes"
import "crypto/rsa"
import "encoding/gob"
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World")
  RSA_LABEL := []byte("served")
  Listen(":1025", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    var key rsa.PublicKey
    if e := gob.NewDecoder(NewBuffer(b)).Decode(&key); e == nil {
      if m, e := Encrypt(&key, HELLO_WORLD, RSA_LABEL); e == nil {
        c.WriteToUDP(m, a)
      }
    }
    return
  })
}
```

```
package main
import . "bytes"
import "crypto/rsa"
import "encoding/gob"
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World")
  RSA_LABEL := []byte("served")
  Listen(":1025", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    var key rsa.PublicKey
    if e := gob.NewDecoder(NewBuffer(b)).Decode(&key); e == nil {
      if m, e := Encrypt(&key, HELLO_WORLD, RSA_LABEL); e == nil {
        c.WriteToUDP(m, a)
      }
    }
    return
  })
}
```

```
package main
import . "bytes"
import "crypto/rsa"
import "encoding/gob"
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World")
  RSA_LABEL := []byte("served")
  Listen(":1025", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    var key rsa.PublicKey
    if e := gob.NewDecoder(NewBuffer(b)).Decode(&key); e == nil {
      if m, e := Encrypt(&key, HELLO_WORLD, RSA_LABEL); e == nil {
        c.WriteToUDP(m, a)
      }
    }
    return
  })
}
```

```
package main
import . "bytes"
import "crypto/rsa"
import "encoding/gob"
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World")
  RSA_LABEL := []byte("served")
  Listen(":1025", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    var key rsa.PublicKey
    if e := gob.NewDecoder(NewBuffer(b)).Decode(&key); e == nil {
      if m, e := Encrypt(&key, HELLO_WORLD, RSA_LABEL); e == nil {
        c.WriteToUDP(m, a)
      }
    }
    return
  })
}
```

```
package main
import . "bytes"
import "crypto/rsa"
import "encoding/gob"
import "net"

func main() {
  HELLO_WORLD := []byte("Hello World")
  RSA_LABEL := []byte("served")
  Listen(":1025", func(c *net.UDPConn, a *net.UDPAddr, b []byte) {
    var key rsa.PublicKey
    if e := gob.NewDecoder(NewBuffer(b)).Decode(&key); e == nil {
      if m, e := Encrypt(&key, HELLO_WORLD, RSA_LABEL); e == nil {
        c.WriteToUDP(m, a)
      }
    }
    return
  })
}
```

```
package main
import "crypto/rand"
import "crypto/rsa"
import "crypto/sha1"

func Encrypt(key *rsa.PublicKey, m, l []byte) ([]byte, error) {
  return rsa.EncryptOAEP(sha1.New(), rand.Reader, key, m, l)
}
```

```
package main
import "crypto/rand"
import "crypto/rsa"
import "crypto/sha1"

func Encrypt(key *rsa.PublicKey, m, l []byte) ([]byte, error) {
  return rsa.EncryptOAEP(sha1.New(), rand.Reader, key, m, l)
}
```

```
package main
import "crypto/rand"
import "crypto/rsa"
import "crypto/sha1"

func Encrypt(key *rsa.PublicKey, m, l []byte) ([]byte, error) {
  return rsa.EncryptOAEP(sha1.New(), rand.Reader, key, m, l)
}
```

```
package main
import "crypto/rand"
import "crypto/rsa"
import "crypto/sha1"

func Encrypt(key *rsa.PublicKey, m, l []byte) ([]byte, error) {
  return rsa.EncryptOAEP(sha1.New(), rand.Reader, key, m, l)
}
```

```
package main
import "crypto/rand"
import "crypto/rsa"
import "crypto/sha1"

func Encrypt(key *rsa.PublicKey, m, l []byte) ([]byte, error) {
  return rsa.EncryptOAEP(sha1.New(), rand.Reader, key, m, l)
}
```

# rsa decrypt

```
package main
import "crypto/rsa"
import . "fmt"
import "net"

func main() {
  Dial(":1025", "ckey", func(c *net.UDPConn, k *rsa.PrivateKey) {
    if m, e := ReadStream(c); e == nil {
      if m, e := Decrypt(k, m, []byte("served")); e == nil {
        Println(string(m))
      }
    }
  })
}
```

```
package main
import "crypto/rsa"
import . "fmt"
import "net"

func main() {
  Dial(":1025", "ckey", func(c *net.UDPConn, k *rsa.PrivateKey) {
    if m, e := ReadStream(c); e == nil {
      if m, e := Decrypt(k, m, []byte("served")); e == nil {
        Println(string(m))
      }
    }
  })
}
```

```
package main
import "crypto/rsa"
import . "fmt"
import "net"

func main() {
  Dial(":1025", "ckey", func(c *net.UDPConn, k *rsa.PrivateKey) {
    if m, e := ReadStream(c); e == nil {
      if m, e := Decrypt(k, m, []byte("served")); e == nil {
        Println(string(m))
      }
    }
  })
}
```

```
package main
import "crypto/rand"
import "crypto/rsa"
import "crypto/sha1"

func Decrypt(key *rsa.PrivateKey, m, l []byte) ([]byte, error) {
  return rsa.DecryptOAEP(sha1.New(), rand.Reader, key, m, l)
}
```

```
package main
import "crypto/rsa"
import . "fmt"
import "net"

func main() {
  Dial(":1025", "ckey", func(c *net.UDPConn, k *rsa.PrivateKey) {
    if m, e := ReadStream(c); e == nil {
      if m, e := Decrypt(k, m, []byte("served")); e == nil {
        Println(string(m))
      }
    }
  })
}
```

```
package main
import "crypto/rsa"
import . "fmt"
import "net"

func main() {
  Dial(":1025", "ckey", func(c *net.UDPConn, k *rsa.PrivateKey) {
    if m, e := ReadStream(c); e == nil {
      if m, e := Decrypt(k, m, []byte("served")); e == nil {
        Println(string(m))
      }
    }
  })
}
```

```
package main
import "crypto/rsa"
import . "fmt"
import "net"

func Dial(a, file string, f func(*net.UDPConn, *rsa.PrivateKey)) {
  if k, e := LoadPrivateKey(file); e == nil {
    if address, e := net.ResolveUDPAddr("udp", a); e == nil {
      if conn, e := net.DialUDP("udp", nil, address); e == nil {
        defer conn.Close()
        SendKey(conn, k.PublicKey, func() {
          f(conn, k)
        })
      }
    }
  }
}
```

```
package main
import "crypto/rsa"
import . "fmt"
import "net"

func Dial(a, file string, f func(*net.UDPConn, *rsa.PrivateKey)) {
  if k, e := LoadPrivateKey(file); e == nil {
    if address, e := net.ResolveUDPAddr("udp", a); e == nil {
      if conn, e := net.DialUDP("udp", nil, address); e == nil {
        defer conn.Close()
        SendKey(conn, k.PublicKey, func() {
          f(conn, k)
        })
      }
    }
  }
}
```

```
package main
import "crypto/rsa"
import . "fmt"
import "net"

func Dial(a, file string, f func(*net.UDPConn, *rsa.PrivateKey)) {
  if k, e := LoadPrivateKey(file); e == nil {
    if address, e := net.ResolveUDPAddr("udp", a); e == nil {
      if conn, e := net.DialUDP("udp", nil, address); e == nil {
        defer conn.Close()
        SendKey(conn, k.PublicKey, func() {
          f(conn, k)
        })
      }
    }
  }
}
```

```
package main
import "crypto/rsa"
import "crypto/x509"
import "encoding/pem"
import "io/ioutil"

func LoadPrivateKey(file string) (r *rsa.PrivateKey, e error) {
  if file, e := ioutil.ReadFile(file); e == nil {
    if block, _ := pem.Decode(file); block != nil {
      if block.Type == "RSA PRIVATE KEY" {
        r, e = x509.ParsePKCS1PrivateKey(block.Bytes)
      }
    }
    return
  }
```

```
package main
import "crypto/rsa"
import "crypto/x509"
import "encoding/pem"
import "io/ioutil"

func LoadPrivateKey(file string) (r *rsa.PrivateKey, e error) {
  if file, e := ioutil.ReadFile(file); e == nil {
    if block, _ := pem.Decode(file); block != nil {
      if block.Type == "RSA PRIVATE KEY" {
        r, e = x509.ParsePKCS1PrivateKey(block.Bytes)
      }
    }
    return
  }
```

```
package main
import "crypto/rsa"
import "crypto/x509"
import "encoding/pem"
import "io/ioutil"

func LoadPrivateKey(file string) (r *rsa.PrivateKey, e error) {
  if file, e := ioutil.ReadFile(file); e == nil {
    if block, _ := pem.Decode(file); block != nil {
      if block.Type == "RSA PRIVATE KEY" {
        r, e = x509.ParsePKCS1PrivateKey(block.Bytes)
      }
    }
    return
  }
```

```
package main
import "crypto/rsa"
import "crypto/x509"
import "encoding/pem"
import "io/ioutil"

func LoadPrivateKey(file string) (r *rsa.PrivateKey, e error) {
  if file, e := ioutil.ReadFile(file); e == nil {
    if block, _ := pem.Decode(file); block != nil {
      if block.Type == "RSA PRIVATE KEY" {
        r, e = x509.ParsePKCS1PrivateKey(block.Bytes)
      }
    }
    return
  }
```

```
package main
import "crypto/rsa"
import "crypto/x509"
import "encoding/pem"
import "io/ioutil"

func LoadPrivateKey(file string) (r *rsa.PrivateKey, e error) {
  if file, e := ioutil.ReadFile(file); e == nil {
    if block, _ := pem.Decode(file); block != nil {
      if block.Type == "RSA PRIVATE KEY" {
        r, e = x509.ParsePKCS1PrivateKey(block.Bytes)
      }
    }
    return
  }
```

```
package main
import "crypto/rsa"
import "crypto/x509"
import "encoding/pem"
import "io/ioutil"

func LoadPrivateKey(file string) (r *rsa.PrivateKey, e error) {
  if file, e := ioutil.ReadFile(file); e == nil {
    if block, _ := pem.Decode(file); block != nil {
      if block.Type == "RSA PRIVATE KEY" {
        r, e = x509.ParsePKCS1PrivateKey(block.Bytes)
      }
    }
    return
  }
```

```
package main
import "crypto/rsa"
import . "fmt"
import "net"

func Dial(a, file string, f func(*net.UDPConn, *rsa.PrivateKey)) {
  if k, e := LoadPrivateKey(file); e == nil {
    if address, e := net.ResolveUDPAddr("udp", a); e == nil {
      if conn, e := net.DialUDP("udp", nil, address); e == nil {
        defer conn.Close()
        SendKey(conn, k.PublicKey, func() {
          f(conn, k)
        })
      }
    }
  }
}
```

```
package main
import "crypto/rsa"
import . "fmt"
import "net"

func Dial(a, file string, f func(*net.UDPConn, *rsa.PrivateKey)) {
  if k, e := LoadPrivateKey(file); e == nil {
    if address, e := net.ResolveUDPAddr("udp", a); e == nil {
      if conn, e := net.DialUDP("udp", nil, address); e == nil {
        defer conn.Close()
        SendKey(conn, k.PublicKey, func() {
          f(conn, k)
        })
      }
    }
  }
}
```

```
package main
import "crypto/rsa"
import . "fmt"
import "net"

func Dial(a, file string, f func(*net.UDPConn, *rsa.PrivateKey)) {
  if k, e := LoadPrivateKey(file); e == nil {
    if address, e := net.ResolveUDPAddr("udp", a); e == nil {
      if conn, e := net.DialUDP("udp", nil, address); e == nil {
        defer conn.Close()
        SendKey(conn, k.PublicKey, func() {
          f(conn, k)
        })
      }
    }
  }
}
```

```
package main
import "bytes"
import “crypto/rsa"
import "encoding/gob"
import "net"

func SendKey(c *net.UDPConn, k rsa.PublicKey, f func()) {
  var b bytes.Buffer
  if e := gob.NewEncoder(&b).Encode(k); e == nil {
    if _, e = c.Write(b.Bytes()); e == nil {
      f()
    }
  }
}
```

```
package main
import "bytes"
import “crypto/rsa"
import "encoding/gob"
import "net"

func SendKey(c *net.UDPConn, k rsa.PublicKey, f func()) {
  var b bytes.Buffer
  if e := gob.NewEncoder(&b).Encode(k); e == nil {
    if _, e = c.Write(b.Bytes()); e == nil {
      f()
    }
  }
}
```

```
package main
import "bytes"
import “crypto/rsa"
import "encoding/gob"
import "net"

func SendKey(c *net.UDPConn, k rsa.PublicKey, f func()) {
  var b bytes.Buffer
  if e := gob.NewEncoder(&b).Encode(k); e == nil {
    if _, e = c.Write(b.Bytes()); e == nil {
      f()
    }
  }
}
```

```
package main
import "crypto/rsa"
import . "fmt"
import "net"

func Dial(a, file string, f func(*net.UDPConn, *rsa.PrivateKey)) {
  if k, e := LoadPrivateKey(file); e == nil {
    if address, e := net.ResolveUDPAddr("udp", a); e == nil {
      if conn, e := net.DialUDP("udp", nil, address); e == nil {
        defer conn.Close()
        SendKey(conn, k.PublicKey, func() {
          f(conn, k)
        })
      }
    }
  }
}
```

# http://golang.org/

# twitter://#golang