Cryptography with Go

This lesson describes a technique for secure communication of data and how Go achieves this security.

WE'LL COVER THE FOLLOWING

Introduction

A technique to secure data from the third parties over communication is called **cryptography**. The data transferred over networks must be *encrypted* so that no hacker can read or change it. Also, checksums calculated over the data, before sending and after receiving, must be identical. Given the business of its mother company Google, it will come as no surprise to see that Go has more than **30** packages to offer in this field, all contained in its standard library:

- `hash` package: implements the *adler32*, *crc32*, *crc64* and *fnv* checksums
- `crypto` package: implements other hashing algorithms like *md5*, *sha1*, and so on and complete encryption implementations for *aes*, *rc4*, *rsa*, *x509*, and so on.

Example

In the following program, we compute and output some checksums with *sha1*:

```
package main
import (
  "fmt"
  "crypto/sha1"
  "io"

  "log"
)

func main() {
  hasher := sha1.New()
  io.WriteString(hasher, "test")
  b := []byte{}
  fmt.Printf("Result: %x\n", hasher.Sum(b))
  fmt.Printf("Result: %d\n", hasher.Sum(b))
  hasher.Reset()
  data := []byte("We shall overcome!")
  n, err := hasher.Write(data)
  if n!=len(data) || err!=nil {
    log.Printf("Hash write error: %v / %v", n, err)
  }
  checksum := hasher.Sum(b)
  fmt.Printf("Result: %x\n", checksum)
}
```

Cryptography with sha1

In the code above, we import the package `crypto/sha1` at **line 4**. At **line 10**, we make a new instance `hasher` of `sha1`, which we use at **line 11** to encrypt the string **test**.

With `sha1.New()`, a new `hash.Hash` object is made to compute the `SHA1` checksum. The type `Hash` is, in fact, an interface, itself implementing the `io.Writer` interface:

```
type Hash interface {
  // Write adds more data to the running hash.
  // It never returns an error.
  io.Writer
  // Sum returns the current hash, without changing the
  // underlying hash state.
  Sum() []byte

  // Reset resets the hash to one with zero bytes written.
  Reset()
  // Size returns the number of bytes Sum will return.
  Size() int
}
```

Through `io.WriteString` or `hasher.Write`, the checksum of the given string is

Through `io.WriteString` or `hasher.Write`, the checksum of the given string is computed. This is first done for **test** at **line 13** and **line 14**, the respective

outputs are:

```
Result: a94a8fe5ccb19ba61c4c0873d391e987982fbbd3
Result: [169 74 143 229 204 177 155 166 28 76 8 115 211 145 233 135 152 4
7 187 211]
```

The `hasher.Write` is used at **line 17** to encrypt **We shall overcome!** After error-handling (from **line 18** to **line 20**), we calculate the *checksum* and print it out:

```
Result: e2222bfc59850bbb00a722e764a555603bb59b2a
```

That is how Golang implements cryptography on data with its packages to make data secure. There is a quiz in the next lesson for you to solve.