Encryption Decryption in Go

Miti Bhat

- Being able to encrypt and decrypt data within an application is very useful for a lot of circumstances. Let’s not confuse encryption and decryption with hashing like that found in a bcrypt library, where a hash is only meant to transform data in one direction.

Hashing Passwords to Compatible Cipher Keys

- When encrypting and decrypting data, it is important that you are using a 32 character, or 32 byte key. Being realistic, you're probably going to want to use a passphrase and that passphrase will never be 32 characters in length.
- To get around this, you can actually hash your passphrase using a hashing algorithm that produces 32 byte hashes. I found a list of hashing algorithms on Wikipedia that provide output lengths. We're going to be using a simple MD5 hash. It is insecure, but it doesn't really matter since we won't be storing the output.
- The function will take a passphrase or any string, hash it, then return the hash as a hexadecimal value.

```
func createHash(key string) string {
	hasher := md5.New()
	hasher.Write([]byte(key))
	return hex.EncodeToString(hasher.Sum(nil))
}
```

Encrypting Data with an AES Cipher

- Now that we have a key of an appropriate size, we can start the encryption process. We can be encrypting text, or any binary data, it doesn't really matter.

```
func encrypt(data []byte, passphrase string) []byte {
	block, _ := aes.NewCipher([]byte(createHash(passphrase)))
	gcm, err := cipher.NewGCM(block)
	if err != nil {
		panic(err.Error())
	}
	nonce := make([]byte, gcm.NonceSize())
	if _, err = io.ReadFull(rand.Reader, nonce); err != nil {
		panic(err.Error())
	}
	ciphertext := gcm.Seal(nonce, nonce, data, nil)
	return ciphertext
}
```

- First we create a new block cipher based on the hashed passphrase. Once we have our block cipher, we want to wrap it in Galois Counter Mode (GCM) with a standard nonce length.
- Before we can create the ciphertext, we need to create a nonce.
- The first parameter in the Seal command is our prefix value. The encrypted data will be appended to it. With the ciphertext, we can return it back to a calling function.

Decrypting Data that uses an AES Cipher

- we can decrypt that same data. The process for decryption is nearly the same as the encryption process. In the above code we create a new block cipher using a hashed passphrase. We wrap the block cipher in Galois Counter Mode and get the nonce size.

```
func decrypt(data []byte, passphrase string) []byte {
	key := []byte(createHash(passphrase))
	block, err := aes.NewCipher(key)
	if err != nil {
		panic(err.Error())
	}
	gcm, err := cipher.NewGCM(block)
	if err != nil {
		panic(err.Error())
	}
	nonceSize := gcm.NonceSize()
	nonce, ciphertext := data[:nonceSize], data[nonceSize:]
	plaintext, err := gcm.Open(nil, nonce, ciphertext, nil)
	if err != nil {
		panic(err.Error())
	}
	return plaintext
}
```

Encrypting and Decrypting Files

```
func encryptFile(filename string, data []byte, passphrase string) {
	f, _ := os.Create(filename)
	defer f.Close()
	f.Write(encrypt(data, passphrase))
}
```

```
func decryptFile(filename string, passphrase string) []byte {
	data, _ := ioutil.ReadFile(filename)
	return decrypt(data, passphrase)
}
```

Thank You