Jerzy Jaskuc

Student Number: 17341645

Design and Implementation

Introduction

Our task was to - design and implement a suitable key management system for your application that allows any member of the group to share social media messages securely, and allows you to add or remove people from a group. You are free to implement your application for a desktop or mobile platform and make use of any open source cryptographic libraries.

Implementation and Design

The whole project consists of two parts

- Backend server that handles the key management and encryption/decryption of messages. Written in **GoLang.**
- Front end extension, that takes care of user management and takes care of input/output. Written in **HTML/JS**.

Back End

On startup, server connects with MongoDB database, which stores all the information about users of the extension.

```
func main(){
  // get mongoDB username and password
  m_username, err := ioutil.ReadFile("src/backend/username.txt") // file wi
  if err != nil {
      log.Fatal(err)
  }
  m_password, err := ioutil.ReadFile("src/backend/password.txt") // file wi
  if err != nil {
      log.Fatal(err)
  }
  URI := "mongodb+srv://" + string(m_username) + ":" + string(m_password) +

  // Set MongoDB client options
  clientOptions := options.Client().ApplyURI(URI)
  mongo_client, err := mongo.Connect(context.Background(), clientOptions)
  if err != nil {
      log.Fatal(err)
  }
  // Check the connection
  err = mongo_client.Ping(context.Background(), nil)
  if err != nil {
      log.Fatal(err)
  }
  fmt.Println("Connected to MongoDB!")
```

After that, server starts to listen to incoming requests from the extension.

Once the request is received, it is checked which type of request it is.

Request can be:

- Encrypt
- Decrypt
- Add
- Remove
- Login
- Register

Depending on the type we proceed accordingly.

When we register the user we just save the credentials of the user to the database and generate a key pair for him and also convert it into string, to be able to save it in DB. RSA key is 2048 bits length.

```
privateKey, err := rsa.GenerateKey(rand.Reader, 2048)
var reworkedPrimes []string
for _, prime := range privateKey.Primes {
    reworkedPrimes = append(reworkedPrimes, prime.String())
}
reworkedPrivK := &ReworkedPrivateKey{
    D: privateKey.D.String(),
    Primes: reworkedPrimes,
    Precomputed: &ReworkedPrecomputed{
        Dp: privateKey.Precomputed.Dp.String(),
        Dq: privateKey.Precomputed.Dq.String(),
        Qinv: privateKey.Precomputed.Qinv.String(),
        CRTValues: []string{},
    },
}
if err != nil {
    log.Printf("Private key generation failed: %v", err)
}
publicKey := privateKey.Public().(*rsa.PublicKey)
reworkedPK := &ReworkedPublicKey{
    E: publicKey.E,
    N: publicKey.N.String(),
}
_, err = collection.InsertOne(context.Background(), bson.M{"nam
```

Similarly, when we receive login request we try to check if user exists and if credentials are correct.

When Adding/Removing from group we just update the user group list.

When we encrypt, we use recipients public key to encrypt the message. Recipient is provided in an incoming request. When we have multiple recipients, we generate messages for each recipient, as each of them will use different private key to decrypt a message on their end. In order to make encrypted message parsable by browser, we additionally encode it into hex.

```go
//Encrypts the message for a single user
func encryptSingle(username, plaintext string) (string, error) {
    var result UserEntry
    filter := bson.M{"name": username}
    err := collection.FindOne(context.Background(), filter).Decode(&result)
    if err != nil {
        log.Printf("User does not exist or there was another problem: %v", err)
        return "", err
    }

    publicKey := result.Public_key
    trueN, _ := big.NewInt(0).SetString(publicKey.N, 0)
    truePublicKey := &rsa.PublicKey{
        E: publicKey.E,
        N: trueN,
    }
    ciphertextBytes, err := rsa.EncryptPKCS1v15(rand.Reader, truePublicKey, []byte(plaintext))
    if err != nil {
        return "", err
    }
    //convert to hext for more readable encoding
    ciphertext := hex.EncodeToString(ciphertextBytes)
    return ciphertext, nil
}
```

When decrypting, we convert back from hex and then use Private key of the user, from which the request is coming, to decrypt the message. If the user is correct, message will be decrypted. Otherwise it will result in an error.

```go
  ciphertextBytes, err := hex.DecodeString(req.Message)
  if err != nil {
      log.Printf("Problem converting from hex ciphertext: %v", err)
      return Response{}, err
  }
  plaintextBytes, err := truePrivateKey.Decrypt(rand.Reader, ciphertextBytes, nil)
  if err != nil {
      log.Printf("Problem decrypting the message: %v", err)
      return Response{}, err
  }
  plaintext := string(plaintextBytes)
  response := Response{
      Message: plaintext,
  }
  return response, nil
```

Front End Extension

For the extension, we use manifest to initialize out extension and use pop-up pages for specific tasks.

That means Login, Register, Add, Remove, Encrypt and Decrypt have their own HTML files and their own scripts.

To take an example of Encrypt popup, we set up listeners on buttons and then proceed accordingly, depending on what user wants to do:

```js
// Listeners for the buttons
document.getElementById('addToGroup').addEventListener('click', openGroupAdder);
document.getElementById('removeFromGroup').addEventListener('click', openGroupRemover);
document.getElementById('encryptionMode').addEventListener('click', openEncryptor);
document.getElementById('decryptionMode').addEventListener('click', openDecryptor);
document.getElementById('send').addEventListener('click', parseMessage);
```

When user wants to change functionality we just move him to another page:

```
function openGroupRemover() {
    console.log("Entered Remove From Group");
    chrome.browserAction.setPopup({popup: 'remove_popup.html'});
    window.location.href = "remove_popup.html";
}
```

Otherwise when user clicks on functional button, we issue the request to the server using XMLHttpRequest() and then wait for a callback from it

```
var request = new XMLHttpRequest();
var url = "http://localhost:420";
request.open("POST", url, true);
request.setRequestHeader("Content-Type", "application/json");
request.onreadystatechange = function () { // when we receive the message, this function is a listener
    if (request.readyState === 4 && request.status === 200) { // receive json from the server
        var json = JSON.parse(request.responseText);
        document.getElementById("output").innerHTML = "Encrypted message is: " + json.Message;
    } else {
        document.getElementById("output").innerHTML = "Encryption failed";
    }
};
var data = JSON.stringify({"type": "encrypt", "user": username, "password": password, "message" : text, "recipients" : recipients});
request.send(data); // send the json to the server
```

We also store essential data of the session inside of the extension and get it when needed

```
chrome.storage.local.get(['password'], function(result) {
    password = result.password
    chrome.storage.local.get(['username'], function(result) {
        username = result.username
        chrome.storage.local.get(['group'], function(result) {
```

Demo

I'm also attaching demo of how it works to the submission.

Appendix

Server

```
package main

import (
	"net/http"
	"go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"
    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/bson/primitive"
    "fmt"
    "io/ioutil"
    "context"
    "log"
    "encoding/json"
    "crypto/rsa"
    "crypto/rand"
    "math/big"
    "encoding/hex"
)

type ReworkedPublicKey struct {
	N string
	E int
}

type ReworkedPrivateKey struct {
	D                   string
	Primes      []string
	Precomputed *ReworkedPrecomputed
```

```
}

type ReworkedPrecomputed struct {
	Dp, Dq		string
	Qinv		string
	CRTValues	[]string
}

type UserEntry struct {
	ID			*primitive.ObjectID`bson:"_id,omitempty"`
	Name		string			`bson:"name"`
	Password	string			`bson:"password"`
	Private_key ReworkedPrivateKey	`bson:"private_key"`
	Public_key	ReworkedPublicKey	`bson:"public_key"`
	Group		[]string			`bson:"group",omitempty`
}

type Request struct {
	Type		string		`json:type`
	User		string		`json:user`
	Password	string		`json:password`
	Message		string		`json:message,omitempty`
	Recipients	[]string	`json:recipients,omitempty`
}

type Response struct {
	Message		string		`json:message`
}

type GroupListResponse struct {
	GroupMembers []string	`json:members,omitempty`
}

var collection *(mongo.Collection)

func main(){
	// get mongoDB username and password
	m_username, err := ioutil.ReadFile("src/backend/username.txt") // file with
just mongoDB username in it
	if err != nil {
	log.Fatal(err)
    }
	m_password, err := ioutil.ReadFile("src/backend/password.txt") // file with
just mongoDB password in it
	if err != nil {
	log.Fatal(err)
    }
	URI := "mongodb+srv://" + string(m_username) + ":" + string(m_password) +
"@telecomms-mkx7q.mongodb.net/test?retryWrites=true&w=majority"

	// Set MongoDB client options
	clientOptions := options.Client().ApplyURI(URI)
	mongo_client, err := mongo.Connect(context.Background(), clientOptions)
	if err != nil {
	    log.Fatal(err)
	}
	// Check the connection
	err = mongo_client.Ping(context.Background(), nil)
	if err != nil {
	    log.Fatal(err)
	}
	fmt.Println("Connected to MongoDB!")
```

```
	collection = mongo_client.Database("Telecomms").Collection("Userbase")

	log.Fatal(http.ListenAndServe(":420", http.HandlerFunc(requestHandler)))
}

//Handles the requests from the extension
func requestHandler(w http.ResponseWriter, req *http.Request) {
	log.Printf("Request received!")
	var decodedRequest Request
	err := json.NewDecoder(req.Body).Decode(&decodedRequest)
	if err != nil {
		log.Printf("JSON decoding failed! %v", err)
		http.Error(w, err.Error(), http.StatusBadRequest)
		return
	}

	if decodedRequest.Type == "encrypt" {
		response, err := encryptTheMessage(decodedRequest)
		if err != nil {
			log.Printf("Encrypting failed: %v", err)
			http.Error(w, err.Error(), http.StatusInternalServerError)
			return
		}
		responseJSON, err := json.Marshal(response)
		if err != nil {
			log.Printf("Marshalling failed: %v", err)
			http.Error(w, err.Error(), http.StatusInternalServerError)
			return
		}
		w.Header().Set("Content-Type", "application/json")
		w.WriteHeader(http.StatusOK)
		w.Write(responseJSON)
	} else if decodedRequest.Type == "decrypt" {
		response, err := decryptTheMessage(decodedRequest)
		if err != nil {
			log.Printf("Decrypting failed: %v", err)
			http.Error(w, err.Error(), http.StatusInternalServerError)
			return
		}
		responseJSON, err := json.Marshal(response)
		if err != nil {
			log.Printf("Marshalling failed: %v", err)
			http.Error(w, err.Error(), http.StatusInternalServerError)
			return
		}
		w.Header().Set("Content-Type", "application/json")
		w.WriteHeader(http.StatusOK)
		w.Write(responseJSON)
	} else if decodedRequest.Type == "add" {
		response, err := addToGroup(decodedRequest)
		if err != nil {
			log.Printf("Adding failed: %v", err)
			http.Error(w, err.Error(), http.StatusInternalServerError)
			return
		}
		responseJSON, err := json.Marshal(response)
		if err != nil {
			log.Printf("Marshalling failed: %v", err)
			http.Error(w, err.Error(), http.StatusInternalServerError)
			return
		}
```

```
                w.Header().Set("Content-Type", "application/json")
                w.WriteHeader(http.StatusOK)
                w.Write(responseJSON)
        } else if decodedRequest.Type == "remove" {
                response, err := removeFromGroup(decodedRequest)
                if err != nil {
                        log.Printf("Removing failed: %v", err)
                        http.Error(w, err.Error(), http.StatusInternalServerError)
                        return
                }
                responseJSON, err := json.Marshal(response)
                if err != nil {
                        log.Printf("Marshalling failed: %v", err)
                        http.Error(w, err.Error(), http.StatusInternalServerError)
                        return
                }
                w.Header().Set("Content-Type", "application/json")
                w.WriteHeader(http.StatusOK)
                w.Write(responseJSON)
        } else if decodedRequest.Type == "login" {
                response, err := logIn(decodedRequest)
                if err != nil {
                        log.Printf("Login failed: %v", err)
                        http.Error(w, err.Error(), http.StatusInternalServerError)
                        return
                }
                responseJSON, err := json.Marshal(response)
                if err != nil {
                        log.Printf("Marshalling failed: %v", err)
                        http.Error(w, err.Error(), http.StatusInternalServerError)
                        return
                }
                w.Header().Set("Content-Type", "application/json")
                w.WriteHeader(http.StatusOK)
                w.Write(responseJSON)
        } else if decodedRequest.Type == "register" {
                success, err := registerUser(decodedRequest)
                if err != nil {
                        log.Printf("Registration failed: %v", err)
                        http.Error(w, err.Error(), http.StatusInternalServerError)
                        return
                }
                if success {
                        w.WriteHeader(http.StatusOK)
                } else {
                        w.WriteHeader(http.StatusInternalServerError)
                }
        }
}

//Encrypts the message and sends back ciphertext/s
func encryptTheMessage(req Request) (Response, error) {
        username := req.User
        password := req.Password
        fmt.Println(req.Message)
        var result UserEntry
        filter := bson.M{"name": username, "password": password}
        err := collection.FindOne(context.Background(), filter).Decode(&result)
        if err != nil {
                log.Printf("User does not exist or there was another problem: %v", err)
                return Response{}, err
        }
```

```
	recipients := req.Recipients
	var ciphertext string
	for _, recipient := range recipients {
		encryptedMessage, err := encryptSingle(recipient, req.Message)
		if err != nil {
			log.Printf("Problem encrypting the message: %v", err)
			return Response{}, err
		}
		ciphertext = ciphertext+ " "+recipient+":"+encryptedMessage
	}
	response := Response{
		Message: ciphertext,
	}
	return response, nil
}

//Encrypts the message for a single user
func encryptSingle(username, plaintext string) (string, error) {
	var result UserEntry
	filter := bson.M{"name": username}
	err := collection.FindOne(context.Background(), filter).Decode(&result)
	if err != nil {
		log.Printf("User does not exist or there was another problem: %v", err)
		return "", err
	}

	publicKey := result.Public_key
	trueN, _ := big.NewInt(0).SetString(publicKey.N, 0)
	truePublicKey := &rsa.PublicKey{
		E: publicKey.E,
		N: trueN,
	}
	ciphertextBytes, err := rsa.EncryptPKCS1v15(rand.Reader, truePublicKey,
[]byte(plaintext))
	if err != nil {
		return "", err
	}
	//convert to hext for more readable encoding
	ciphertext := hex.EncodeToString(ciphertextBytes)
	return ciphertext, nil
}

//Decrypts the message and sends back the plaintext
func decryptTheMessage(req Request) (Response, error) {
	username := req.User
	password := req.Password

	var result UserEntry
	filter := bson.M{"name": username, "password": password}
	err := collection.FindOne(context.Background(), filter).Decode(&result)
	if err != nil {
		log.Printf("User does not exist or there was another problem: %v", err)
		return Response{}, err
	}
	publicKey := result.Public_key
	trueN, _ := big.NewInt(0).SetString(publicKey.N, 0)
	truePublicKey := &rsa.PublicKey{
		E: publicKey.E,
		N: trueN,
	}
	privateKey := result.Private_key
	var truePrimes []*big.Int
```

```
	for _, prime := range privateKey.Primes {
		truePrime, _ := big.NewInt(0).SetString(prime, 0)
		truePrimes = append(truePrimes, truePrime)
	}
	trueD, _ := big.NewInt(0).SetString(privateKey.D, 0)
	trueDp, _ := big.NewInt(0).SetString(privateKey.Precomputed.Dp, 0)
	trueDq, _ := big.NewInt(0).SetString(privateKey.Precomputed.Dq, 0)
	trueQinv, _ := big.NewInt(0).SetString(privateKey.Precomputed.Qinv, 0)
	truePrivateKey := &rsa.PrivateKey{
		PublicKey:  *truePublicKey,
		D: trueD,
		Primes: truePrimes,
		Precomputed : rsa.PrecomputedValues {
			Dp: trueDp,
			Dq: trueDq,
			Qinv: trueQinv,
			CRTValues: []rsa.CRTValue{},
		},
	}
	ciphertextBytes, err := hex.DecodeString(req.Message)
	if err != nil {
		log.Printf("Problem converting from hex ciphertext: %v", err)
		return Response{}, err
	}
	plaintextBytes, err := truePrivateKey.Decrypt(rand.Reader, ciphertextBytes,
nil)
	if err != nil {
		log.Printf("Problem decrypting the message: %v", err)
		return Response{}, err
	}
	plaintext := string(plaintextBytes)
	response := Response{
		Message: plaintext,
	}
	return response, nil
}

//Adds a user to the group and updates BD before returning new list to extension
func addToGroup(req Request) (GroupListResponse, error) {
	username := req.User
	password := req.Password

	var result UserEntry
	filter := bson.M{"name": username, "password": password}
	err := collection.FindOne(context.Background(), filter).Decode(&result)
	if err != nil {
		log.Printf("User does not exist or there was another problem: %v", err)
		return GroupListResponse{}, err
	}

	recipients := append(result.Group, req.Message)

	replacement := bson.M{
		"name" : username,
		"password" : password,
		"private_key" : result.Private_key,
		"public_key" : result.Public_key,
		"group" : recipients}
	var replacedDoc bson.M
	err = collection.FindOneAndReplace(context.Background(), filter,
replacement).Decode(&replacedDoc)
	if err != nil {
```

```
            log.Printf("Problem replacing the document: %v", err)
            return GroupListResponse{}, err
        }
        response := GroupListResponse{
                GroupMembers : recipients,
        }
        return response, nil
}

//Removes a user from the group and updates BD before returning new list to extension
func removeFromGroup(req Request) (GroupListResponse, error) {
        username := req.User
        password := req.Password

        var result UserEntry
        filter := bson.M{"name": username, "password": password}
        err := collection.FindOne(context.Background(), filter).Decode(&result)
        if err != nil {
                log.Printf("User does not exist or there was another problem: %v", err)
                return GroupListResponse{}, err
        }

        toRemove := req.Message
        var newRecipients []string
        for _, recipient := range result.Group {
                if recipient == toRemove {
                        continue
                } else {
                        newRecipients = append(newRecipients, recipient)
                }
        }

        replacement := bson.M{
                "name" : username,
                "password" : password,
                "private_key" : result.Private_key,
                "public_key" : result.Public_key,
                "group" : newRecipients}
        var replacedDoc bson.M
        err = collection.FindOneAndReplace(context.Background(), filter,
replacement).Decode(&replacedDoc)
        if err != nil {
            log.Printf("Problem replacing the document: %v", err)
            return GroupListResponse{}, err
        }
        response := GroupListResponse{
                GroupMembers : newRecipients,
        }
        return response, nil
}

//Check if user exists and has exact password, and then return his group list
func logIn(req Request) (GroupListResponse, error) {
        username := req.User
        password := req.Password

        var result UserEntry
        filter := bson.M{"name": username, "password": password}
        err := collection.FindOne(context.Background(), filter).Decode(&result)
        if err != nil {
                log.Printf("User does not exist or there was another problem: %v", err)
                return GroupListResponse{}, err
```

```
	}
	response := GroupListResponse{
		GroupMembers : result.Group,
	}
	return response, nil
}

//Register the user with given username and password to DB
func registerUser(req Request) (bool, error) {
	username := req.User
	password := req.Password

	var result UserEntry
	filter := bson.D{{"name",username}}
	err := collection.FindOne(context.Background(), filter).Decode(&result)
	if err!= nil {
		if err == mongo.ErrNoDocuments {
			privateKey, err := rsa.GenerateKey(rand.Reader, 2048)
			var reworkedPrimes []string
			for _, prime := range privateKey.Primes {
				reworkedPrimes = append(reworkedPrimes, prime.String())
			}
			reworkedPrivK := &ReworkedPrivateKey{
				D: privateKey.D.String(),
				Primes: reworkedPrimes,
				Precomputed: &ReworkedPrecomputed{
					Dp: privateKey.Precomputed.Dp.String(),
					Dq: privateKey.Precomputed.Dq.String(),
					Qinv: privateKey.Precomputed.Qinv.String(),
					CRTValues: []string{},
				},
			}
			if err != nil {
				log.Printf("Private key generation failed: %v", err)
			}
			publicKey := privateKey.Public().(*rsa.PublicKey)
			reworkedPK := &ReworkedPublicKey{
				E: publicKey.E,
				N: publicKey.N.String(),
			}
			_, err = collection.InsertOne(context.Background(),
bson.M{"name": username, "password": password, "private_key": reworkedPrivK,
"public_key": reworkedPK})
			if err != nil {
				log.Printf("Problem registering the user: %v", err)
				return false, err
			}
			return true, nil
		} else {
			log.Printf("Problem registering the user: %v", err)
			return false, err
		}
	}
	log.Printf("User already exists")
	return false, nil
}
```

Front end

Manifest.json

```
{
   "name": "Encryption/Decryption app",
   "version": "1.0",
   "description": "Social media message encryption app",
          "permissions": ["activeTab","declarativeContent","storage","http://localhost:420/"],
          "background": {
    "scripts": ["background.js"],
    "persistent": false
   },
          "browser_action" : {
            "default_popup": "popup_log_in.html",
            "default_icon": {
      "16": "images/get_started16.png",
      "32": "images/get_started32.png",
      "48": "images/get_started48.png",
      "128": "images/get_started128.png"
    }
          },
          "icons": {
    "16": "images/get_started16.png",
    "32": "images/get_started32.png",
    "48": "images/get_started48.png",
    "128": "images/get_started128.png"
   },
   "manifest_version": 2
 }
```

HTML's

```
<!DOCTYPE html>
<html>

<head>
   <title>Registration Menu</title>
```

```
  <style="text/css">
      .buttons button { outline: none; float: left; background-color: #4CAF50; color: white; padding: 5px 10px; margin: 5px; } .buttons button:hover { background-color: #3e8e41; } .buttons:after { content: ""; clear: both; display: table; } .form_part { margin: 5px; } .registration_window { width:230px; height:145px; } .error_output { margin 5px; }
    </style>
</head>

<body>
  <div class="registration_window">
    <div class="form_part">
      <form>
        <label for="username">Username:</label>
        <br>
        <input type="text" id="username" name="user">
        <br>
        <label for="password">Password:</label>
        <br>
        <input type="password" id="password" name="pass">
      </form>
    </div>
    <div class="buttons">
      <button id="register">Register now!</button>
      <button id="login">Login Instead</button>
    </div>
    <p id="error_output"></p>
  </div>
  <script src="register_popup.js"></script>
</body>

</html>

<!DOCTYPE html>
<html>
```

```
<head>
  <title>Log In Menu</title>
  <style="text/css">
    .buttons button { outline: none; float: left; background-color: #4CAF50; color: white; padding: 5px 10px; margin: 5px; } .buttons button:hover { background-color: #3e8e41; } .buttons:after { content: ""; clear: both; display: table; } .form_part { margin: 5px; } .login_window { width:200px; height:145px; } .error_output { margin 5px; }
    </style>
</head>

<body>
  <div class="login_window">
    <div class="form_part">
      <form>
        <label for="username">Username:</label>
        <br>
        <input type="text" id="username" name="user">
        <br>
        <label for="password">Password:</label>
        <br>
        <input type="password" id="password" name="pass">
      </form>
    </div>
    <div class="buttons">
      <button id="login">Login</button>
      <button id="register">Register Instead</button>
    </div>
    <p id="error_output"></p>
  </div>
  <script src="popup_log_in.js"></script>
</body>

</html>

<!DOCTYPE html>
```

```
<html>


<head>
   <title>Remove from Group Menu</title>
   <style="text/css">
      .upper_menu button { outline: none; float: left; background-color: #4CAF50; color: white; padding: 10px 20px; margin: 5px; } .upper_menu button:hover { background-color: #3e8e41; } .upper_menu:after { content: ""; clear: both; display: table; } .bottom_part { margin: 5px; float: left; } .main_window { width:600px; height:120px; } .text_field { margin: 5px; }
      </style>
</head>


<body>
   <div class="main_window">
      <div class="upper_menu">
         <button id="addToGroup">Add To Group</button>
         <button id="removeFromGroup">Remove From Group</button>
         <button id="encryptionMode">Encrypt Mode</button>
         <button id="decryptionMode">Decrypt Mode</button>
      </div>
      <div class="text_field">
         <form>
            <label for="removeUser">User to remove:</label>
            <input type="text" id="removeUser" name="removegrp">
         </form>
      </div>
      <div class="bottom_part">
         <button id="remove">Remove</button>
         <p id="output"></p>
      </div>
   </div>
   <script src="remove_popup.js"></script>
</body>


</html>
```

```
<!DOCTYPE html>
<html>

<head>
  <title>Add to Group Menu</title>
  <style="text/css">
    .upper_menu button { outline: none; float: left; background-color: #4CAF50; color: white; padding: 10px 20px; margin: 5px; } .upper_menu button:hover { background-color: #3e8e41; } .upper_menu:after { content: ""; clear: both; display: table; } .bottom_part { margin: 5px; float: left; } .main_window { width:600px; height:120px; } .text_field { margin: 5px; }
    </style>
</head>

<body>
  <div class="main_window">
    <div class="upper_menu">
      <button id="addToGroup">Add To Group</button>
      <button id="removeFromGroup">Remove From Group</button>
      <button id="encryptionMode">Encrypt Mode</button>
      <button id="decryptionMode">Decrypt Mode</button>
    </div>
    <div class="text_field">
      <form>
        <label for="addUser">User to add:</label>
        <input type="text" id="addUser" name="addgrp">
      </form>
    </div>
    <div class="bottom_part">
      <button id="add">Add</button>
      <p id="output"></p>
    </div>
  </div>
  <script src="add_popup.js"></script>
</body>
```

```
</html>


<!DOCTYPE html>
<html>


<head>
  <title>Decryption Menu</title>
  <style="text/css">
    .main_menu button { outline: none; float: left; background-color: #4CAF50; color: white; padding: 10px 20px; margin: 5px; } .main_menu button:hover { background-color: #3e8e41; } .main_menu:after { content: ""; clear: both; display: table; } .main_window { width:600px; height:200px; } .textbox { margin: 5px; } .output { margin: 5px; } .out { word-wrap: break-word; }
    </style>
</head>


<body>
  <div class="main_window">
    <div class="main_menu">
      <button id="addToGroup">Add To Group</button>
      <button id="removeFromGroup">Remove From Group</button>
      <button id="encryptionMode">Encrypt Mode</button>
      <button id="decryptionMode">Decrypt Mode</button>
    </div>
    <div class="textbox">
      <textarea name="enc_message" id="decryptMsg" cols="60" rows="3">Please enter your message to decrypt</textarea>
      </br>
      <button id="decrypt">Decrypt!</button>
    </div>
    <div class="out">
      <p id="output"></p>
    </div>
  </div>
  <script src="decrypt_popup.js"></script>
```

```
</body>


</html>


<!DOCTYPE html>
<html>


<head>
  <title>Encryption Menu</title>
  <style="text/css">
    .main_menu button { outline: none; float: left; background-color: #4CAF50; color: white; padding: 10px 20px; margin: 5px; } .main_menu button:hover { background-color: #3e8e41; } .main_menu:after { content: ""; clear: both; display: table; } .main_window { width:600px; height:200px; } .textbox { margin: 5px; } .output { margin: 5px; } .out { word-wrap: break-word; }
    </style>
</head>


<body>
  <div class="main_window">
    <div class="main_menu">
      <button id="addToGroup">Add To Group</button>
      <button id="removeFromGroup">Remove From Group</button>
      <button id="encryptionMode">Encrypt Mode</button>
      <button id="decryptionMode">Decrypt Mode</button>
    </div>
    <div class="textbox">
      <textarea name="enc_message" id="encryp" cols="60" rows="3">Please enter your message to encrypt</textarea>
      </br>
      To:
      <select id="recipient" name="to">
      </select>
      <button id="send">Encrypt!</button>
    </div>
    <div class="out">
```

```
        <p id="output"></p>
      </div>
    </div>
    <script src="popup.js"></script>
</body>

</html>
```

JavaScripts

```
function openRegistration() {
        chrome.browserAction.setPopup({popup: 'register_popup.html'});
        window.location.href = "register_popup.html";
}
function login() {
        var username = document.getElementById('username').value;
        var password = document.getElementById('password').value

        //send the credentials and receive the response and save the group and username + password
        var request = new XMLHttpRequest();
        var url = "http://localhost:420";
        request.open("POST", url, true);
        request.setRequestHeader("Content-Type", "application/json");
        request.onreadystatechange = function () { // when we receive the message, this function is a
listener
                if (request.readyState === 4 && request.status === 200) { // proceed accordingly when
received
                        var json = JSON.parse(request.responseText);
                        chrome.storage.local.set({"username": username}, function() {
                                console.log('Username saved!');
                        });
                        chrome.storage.local.set({"password": password}, function() {
                                console.log('Password saved!');
                        });
                        chrome.storage.local.set({"group": json.GroupMembers}, function() {
                                console.log('Group saved!');
```

```
                        });
                        chrome.browserAction.setPopup({popup: 'popup.html'});
                        window.location.href = "popup.html";
                } else {
                        document.getElementById("error_output").innerHTML = "Login Failed!"
                }
        };
        var data = JSON.stringify({"type": "login", "user": username, "password": password});
        request.send(data); // send the json to the server

}

// Listeners for the buttons
document.getElementById('login').addEventListener('click', login);
document.getElementById('register').addEventListener('click', openRegistration);
function openLogin() {
        chrome.browserAction.setPopup({popup: 'popup_log_in.html'});
        window.location.href = "popup_log_in.html";
}

function register() {
        var username = document.getElementById('username').value;
        var password = document.getElementById('password').value

        //send the credentials and receive the response and save the group and username + password
        var request = new XMLHttpRequest();
        var url = "http://localhost:420";
        request.open("POST", url, true);
        request.setRequestHeader("Content-Type", "application/json");
        request.onreadystatechange = function () { // when we receive the message, this function is a
listener
                if (request.readyState === 4 && request.status === 200) { // proceed accordingly when
received
                        chrome.storage.local.set({"username": username}, function() {
                                console.log('Username saved!');
```

```
                        });
                        chrome.storage.local.set({"password": password}, function() {
                                console.log('Password saved!');
                        });
                        chrome.storage.local.set({"group": []}, function() {
                                console.log('Group saved!');
                        });
                        chrome.browserAction.setPopup({popup: 'popup.html'});
                        window.location.href = "popup.html";
                } else {
                        document.getElementById("error_output").innerHTML = "Username already
exists!"
                }
        };
        var data = JSON.stringify({"type": "register", "user": username, "password": password});
        request.send(data); // send the json to the server

}

// Listeners for the buttons
document.getElementById('login').addEventListener('click', openLogin);
document.getElementById('register').addEventListener('click', register);



function openGroupAdder() {
        console.log("Entered Add To Group");
        chrome.browserAction.setPopup({popup: 'add_popup.html'});
        window.location.href = "add_popup.html";
}

function openGroupRemover() {
        console.log("Entered Remove From Group");
        chrome.browserAction.setPopup({popup: 'remove_popup.html'});
        window.location.href = "remove_popup.html";
```

```
}

function openEncryptor() {
	console.log("Entered Encryption");
	chrome.browserAction.setPopup({popup: 'popup.html'});
	window.location.href = "popup.html";
}

function openDecryptor() {
	console.log("Entered Decryption");
	chrome.browserAction.setPopup({popup: 'decrypt_popup.html'});
	window.location.href = "decrypt_popup.html";
}

function parseMessage() {
	var text = document.getElementById('encryp').value;
	var currentOption = document.getElementById('recipient').value
	var groupMembers;
	var username;
	var password;


	chrome.storage.local.get(['password'], function(result) {
    password = result.password
		chrome.storage.local.get(['username'], function(result) {
			username = result.username
			chrome.storage.local.get(['group'], function(result) {
				groupMembers = result.group;
				var recipients
				if (currentOption == 'all') {
					recipients = groupMembers;
				} else {
					recipients = [currentOption];
				}
```

```
                var request = new XMLHttpRequest();
                var url = "http://localhost:420";
                request.open("POST", url, true);
                request.setRequestHeader("Content-Type", "application/json");
                request.onreadystatechange = function () { // when we receive the message, this function is a listener
                    if (request.readyState === 4 && request.status === 200) { // receive json from the server
                        var json = JSON.parse(request.responseText);
                        document.getElementById("output").innerHTML = "Encrypted message is: " + json.Message;
                    } else {
                        document.getElementById("output").innerHTML = "Encryption failed";
                    }
                };
                var data = JSON.stringify({"type": "encrypt", "user": username, "password": password, "message" : text, "recipients" : recipients});
                request.send(data); // send the json to the server
            });
        });
    });
}

// Listeners for the buttons
document.getElementById('addToGroup').addEventListener('click', openGroupAdder);
document.getElementById('removeFromGroup').addEventListener('click', openGroupRemover);
document.getElementById('encryptionMode').addEventListener('click', openEncryptor);
document.getElementById('decryptionMode').addEventListener('click', openDecryptor);
document.getElementById('send').addEventListener('click', parseMessage);

let selectList = document.getElementById('recipient');
selectList.length = 0;
let defaultOpt = document.createElement('option');
defaultOpt.text = 'All in Group';
defaultOpt.value = 'all';
```

```
selectList.add(defaultOpt);
selectList.selectedIndex = 0;

var group;
chrome.storage.local.get(['group'], function(result) {
	group = result.group;
	let option;
	for (let i=0;i<group.length;i++) {
		option = document.createElement('option');
		option.text = group[i];
		option.value = group[i];
		selectList.add(option);
	}
});


function openGroupAdder() {
	console.log("Entered Add To Group");
	chrome.browserAction.setPopup({popup: 'add_popup.html'});
	window.location.href = "add_popup.html";
}

function openGroupRemover() {
	console.log("Entered Remove From Group");
	chrome.browserAction.setPopup({popup: 'remove_popup.html'});
	window.location.href = "remove_popup.html";
}

function openEncryptor() {
	console.log("Entered Encryption");
	chrome.browserAction.setPopup({popup: 'popup.html'});
	window.location.href = "popup.html";
}
```

```
function openDecryptor() {
	console.log("Entered Decryption");
	chrome.browserAction.setPopup({popup: 'decrypt_popup.html'});
	window.location.href = "decrypt_popup.html";
}

function decrypt() {
	var ciphertext = document.getElementById('decryptMsg').value;
	var username;
	var password;
	chrome.storage.local.get(['password'], function(result) {
      password = result.password
		chrome.storage.local.get(['username'], function(result) {
			username = result.username
			var request = new XMLHttpRequest();
			var url = "http://localhost:420";
			request.open("POST", url, true);
			request.setRequestHeader("Content-Type", "application/json");
			request.onreadystatechange = function () { // when we receive the message, this function is a listener
				if (request.readyState === 4 && request.status === 200) { // receive json from the server
					var json = JSON.parse(request.responseText);
					document.getElementById("output").innerHTML = "Decrypted message is: " + json.Message;
				} else {
					document.getElementById("output").innerHTML = "Decryption failed";
				}
			};
			var data = JSON.stringify({"type": "decrypt", "user": username, "password": password, "message" : ciphertext});
			request.send(data); // send the json to the server
		});
   });
}
```

```
// Listeners for the buttons
document.getElementById('addToGroup').addEventListener('click', openGroupAdder);
document.getElementById('removeFromGroup').addEventListener('click', openGroupRemover);
document.getElementById('encryptionMode').addEventListener('click', openEncryptor);
document.getElementById('decryptionMode').addEventListener('click', openDecryptor);
document.getElementById('decrypt').addEventListener('click', decrypt);


function openGroupAdder() {
	console.log("Entered Add To Group");
	chrome.browserAction.setPopup({popup: 'add_popup.html'});
	window.location.href = "add_popup.html";
}

function openGroupRemover() {
	console.log("Entered Remove From Group");
	chrome.browserAction.setPopup({popup: 'remove_popup.html'});
	window.location.href = "remove_popup.html";
}

function openEncryptor() {
	console.log("Entered Encryption");
	chrome.browserAction.setPopup({popup: 'popup.html'});
	window.location.href = "popup.html";
}

function openDecryptor() {
	console.log("Entered Decryption");
	chrome.browserAction.setPopup({popup: 'decrypt_popup.html'});
	window.location.href = "decrypt_popup.html";
}

function remove() {
```

```
        var userToRemove = document.getElementById('removeUser').value;
        var username;
        var password;
        chrome.storage.local.get(['password'], function(result) {
    password = result.password
                chrome.storage.local.get(['username'], function(result) {
                        username = result.username
                        // send the username and current credentials recieve new username list and save it
                        var request = new XMLHttpRequest();
                        var url = "http://localhost:420";
                        request.open("POST", url, true);
                        request.setRequestHeader("Content-Type", "application/json");
                        request.onreadystatechange = function () { // when we receive the message, this function is a listener
                                if (request.readyState === 4 && request.status === 200) { // proceed accordingly when received
                                        var json = JSON.parse(request.responseText);
                                        chrome.storage.local.set({"group": json.GroupMembers}, function() {
                                                console.log('Group saved!');
                                        });
                                        document.getElementById("output").innerHTML = "User successfully removed from the group!"
                                } else {
                                        document.getElementById("output").innerHTML = "There was a problem removing user from the group. Try again!"
                                }
                        };
                        var data = JSON.stringify({"type": "remove", "user": username, "password": password, "message" : userToRemove});
                        request.send(data); // send the json to the server
                });
   });
}
```

```
// Listeners for the buttons
document.getElementById('addToGroup').addEventListener('click', openGroupAdder);
document.getElementById('removeFromGroup').addEventListener('click', openGroupRemover);
document.getElementById('encryptionMode').addEventListener('click', openEncryptor);
document.getElementById('decryptionMode').addEventListener('click', openDecryptor);
document.getElementById('remove').addEventListener('click', remove);


function openGroupAdder() {
	console.log("Entered Add To Group");
	chrome.browserAction.setPopup({popup: 'add_popup.html'});
	window.location.href = "add_popup.html";
}

function openGroupRemover() {
	console.log("Entered Remove From Group");
	chrome.browserAction.setPopup({popup: 'remove_popup.html'});
	window.location.href = "remove_popup.html";
}

function openEncryptor() {
	console.log("Entered Encryption");
	chrome.browserAction.setPopup({popup: 'popup.html'});
	window.location.href = "popup.html";
}

function openDecryptor() {
	console.log("Entered Decryption");
	chrome.browserAction.setPopup({popup: 'decrypt_popup.html'});
	window.location.href = "decrypt_popup.html";
}

function add() {
	var userToAdd = document.getElementById('addUser').value;
```

```
        var username;
        var password;
        chrome.storage.local.get(['password'], function(result) {
    password = result.password;
                chrome.storage.local.get(['username'], function(result) {
                        username = result.username;
                        // send the username and current credentials and recieve new username list and save it
                        var request = new XMLHttpRequest();
                        var url = "http://localhost:420";
                        request.open("POST", url, true);
                        request.setRequestHeader("Content-Type", "application/json");
                        request.onreadystatechange = function () { // when we receive the message, this function is a listener
                                if (request.readyState === 4 && request.status === 200) { // proceed accordingly when received
                                        var json = JSON.parse(request.responseText);
                                        chrome.storage.local.set({"group": json.GroupMembers}, function() {
                                                console.log('Group saved!');
                                        });
                                        document.getElementById("output").innerHTML = "User successfully added to the group!"
                                } else {
                                        document.getElementById("output").innerHTML = "There was a problem adding user to the group. Try again!"
                                }
                        };
                        var data = JSON.stringify({"type": "add", "user": username, "password": password, "message" : userToAdd});
                        request.send(data); // send the json to the server
                });
   });


}
```

```
// Listeners for the buttons
document.getElementById('addToGroup').addEventListener('click', openGroupAdder);
document.getElementById('removeFromGroup').addEventListener('click', openGroupRemover);
document.getElementById('encryptionMode').addEventListener('click', openEncryptor);
document.getElementById('decryptionMode').addEventListener('click', openDecryptor);
document.getElementById('add').addEventListener('click', add);
```