Projeto de Segurança Informática e Nas Organizações

...

Universidade de Aveiro

Isac Cruz (90513)
Luís Oliveira (98543)
Rui Rosmaninho (113553)

Capítulo 1

Cifra

1.1 Introdução

Para facilitar e isolar todo o processo de cifragem foi criado um módulo paralelo à comunicação.

As mensagens são criadas e enviadas pelo cliente/servidor de forma transparente, e o módulo de cifra garante um aumento de segurança na comunicação. O módulo de cifra adiciona marcas de verificação (como marcas de verificação temporais), cifra a mensagem e envia-a para o receptor. Na recepção, a mensagem é decifrada e retornada num formato legível.

Este módulo fornece ainda um conjunto de ferramentas adicionais para operações intermédias (como cifra simétrica e assimétrica), criação e verificação de assinaturas digitais, criação e verificação de MAC's, entre outras.

As funções principais deste módulo são 'encrypt_request' e 'encrypt_request' (Seção 1.2) que cifram toda a comunicação pré-envio e pós-recepção. Será precisamente nestas funções que as marcas de verificação serão adicionadas (no envio). Cada campo do pedido é cifrado com as funções 'encrypt_easy' (Seção 1.5) e, caso não seja um ficheiro de uma organização, será codificado para ser enviado no formato json (Seção 1.6).

No caso de envio de um ficheiro de uma organização, a mensagem será assinada com um MAC para garantir a integridade. O diagrama de fluxo é apresentado na Figura 1.1

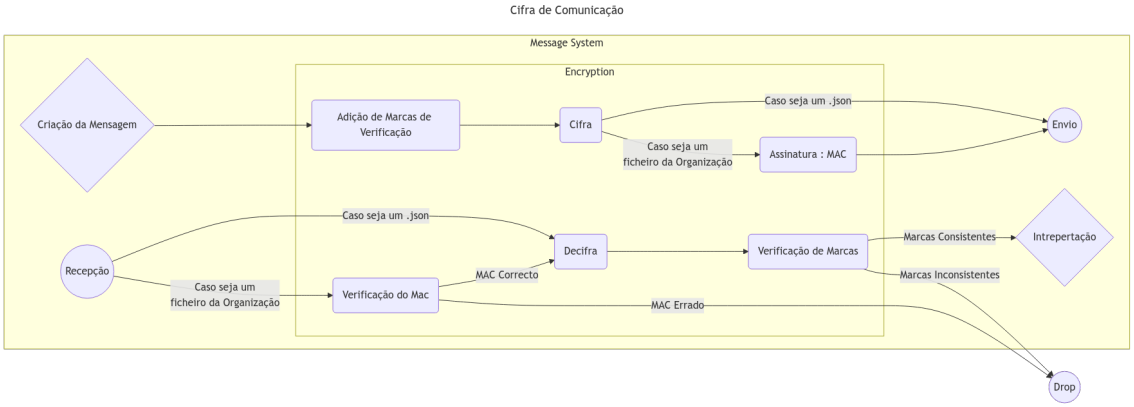

Figura 1.1: Fluxo de Cifra

1.2 Cifra e decifra de pedidos

Os campos do pedido são todos cifrados com uma cifra híbrida utilizando encrypt_easy (Seção 1.5) com o algoritmo 'AES', sendo codificados os dados cujo envio precisa ser efetuado no formato json (Seção 1.6). Na decifra, é criado um objeto 'Req' cujos atributos contêm as informações relevantes do pedido: form, files, json. Sempre que possível, os pedidos são cifrados utilizando a chave pública do servidor.

```
1     class Req:
2     def __init__(self, data, files, json_file):
```

```
        self.form = data
        self.files = files
        self.json = json_file

    def __str__(self): return f"Data: {self.form}\nFiles: {self.files}\nJson: {self.json}"

    @property
    def form(self): return self._data

    @form.setter
    def form(self, value): self._data = value

    @property
    def files(self): return self._files

    @files.setter
    def files(self, value): self._files = value

    @property
    def json(self): return self._json_file

    @json.setter
    def json(self, value): self._json_file = value

def encrypt_request(rep_public_key, action, endpoint, data={}, files={}, json_file={}):
    response = None
    file_lisr = []

    if rep_public_key != None:
        data["previous_interaction"] = datetime.now().isoformat()
        data = {"data" : decode_bytes2string(encrypt_easy(rep_public_key,
        ↪  json.dumps(data).encode(), "AES"))} if not data == {} else {}
        for key in files:
            secret = encrypt_easy(rep_public_key, files[key].read(), "AES")
            inner_file = io.BufferedReader(io.BytesIO(secret))
            file_lisr.append(inner_file)
            files[key] = inner_file
        json_file = {"json" : decode_bytes2string(encrypt_easy(rep_public_key,
        ↪  json.dumps(json_file).encode(), "AES"))} if not json_file == {} else {}
    else:
        data = {"data" : json.dumps(data)} if not data == {} else {}
        json_file = {"json" : json.dumps(json_file)} if not json_file == {} else {}

    if action == 'POST': response = requests.post(endpoint, data=data, files=files)
    elif action == 'GET': response = requests.get(endpoint, data=data, files=files)

    for file in file_lisr: file.close()
    return response


def decrypt_request(rep_private_key, request) -> Req:
    data = request.form["data"]
    files = {}
    json_file = {}

    for file_name in request.files:
        secret = decrypt_easy(rep_private_key, request.files[file_name].read(), "AES")
        request.files[file_name].close()
        files[file_name] = secret

    data = json.loads(decrypt_easy(rep_private_key, encode_string2bytes(data), "AES")) if
    ↪  data != {} else {}
    json_file = json.loads(decrypt_easy(rep_private_key, encode_string2bytes(json_file),
    ↪  "AES")) if json_file != {} else {}
    return Req(data, files, json_file)
```

1.3 Cifra e decifra de respostas

As respostas são cifradas utilizando a chave pública do utilizador nos pedidos não anónimos. As respostas são cifradas pela função 'easy' (Seção 1.5) e codificadas pelas funções 'encode' (Seção 1.6) para envio no formato 'json'. Na chegada ao cliente, são aplicados os processos inversos para a reconstrução da resposta original. Esta parte de código não chegou a ser incluída na versão final por equívoco, mas toda a estrutura está desenhada para funcionar com esta 'feature'.

```
1  def encrypt_response(key, data:{}):
2      if key == None: return jsonify(data)
3      jsonify({
4          "data" : decode_bytes2string(encrypt_easy(key, json.dumps(data).encode(), "AES"))
5      })
6
7  def decrypt_response(key, data:{}):
8      if key == None: return data
9      if "data" in data:
10         return json.loads(decrypt_easy(key, encode_string2bytes(data["data"]), "AES"))
11
```

1.4 Marcas de Verificação

As marcas de verificação são campos adicionados ao pedido que contêm informações relevantes para a validação e verificação do pedido. Neste projeto, apenas foram utilizados marcadores temporais. No entanto, outros tipos de marcações podem ser adicionadas. Todas as marcas são adicionadas antes da cifra dos dados.

Estas têm como intuito impedir pedidos duplicados, registando e comparando no sistema a última data de um pedido.

1.5 Cifra Genérica

A cifra genérica trata-se de uma cifra híbrida que, utilizando um algoritmo definido no argumento de entrada, cria um ficheiro cifrado com uma chave aleatória, a qual é cifrada com a chave publica do servidor (Figura 1.4).

Na saída da função, é criado um ficheiro onde os primeiros 20 bits contêm o tamanho N da chave simétrica cifrada. Os N bits seguintes contêm a chave cifrada e os restantes bits pertencentes ao ficheiro cifrado (Figura 1.3). Os ficheiros criados podem ser armazenados, enviados ou incluídos num ficheiro json (Seção 1.6).

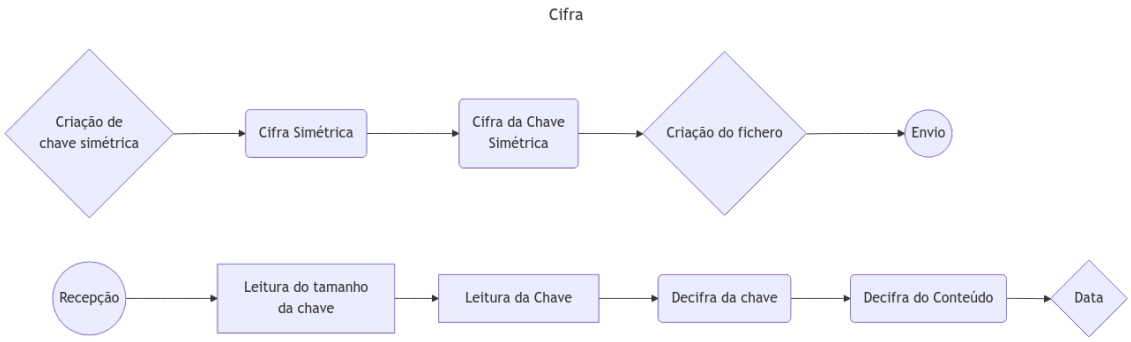

Figura 1.2: Cifra Genérica

Tamanho da Chave	Chave Cifrada	Informação Cifrada

Figura 1.3: Ficheiro Criado

```python
def encrypt_easy(public_key, data, alg_name):
    encrypted_message, simetric_key = simetric_encryption(data, alg_name)
    encrypted_key,size = encrypt_key(public_key, simetric_key)
    return size + encrypted_key + encrypted_message

def decrypt_easy(private_key, data, alg_name):
    key_lenght = int.from_bytes(data[0:KEY_SIZE],byteorder="big")
    data = data[KEY_SIZE:]
    encrypted_key = data[0:key_lenght]
    data = data[key_lenght:]
    simetric_key = decrypt_key(private_key, encrypted_key)
    message = simetric_decryption(simetric_key, data, alg_name)
    return message
```

1.6 Cifra e envio no formato Json

Para o envio de informação no formato json, é necessário a criação de informação descodificável (possível de ser convertida para caracteres). Os bits criados pelas funções de cifra nem sempre são descodificáveis. Por esta razão, foram criadas funções para o efeito. No sistema criado é possível cifrar informação sem perder a possibilidade do envio no formato mais comum.

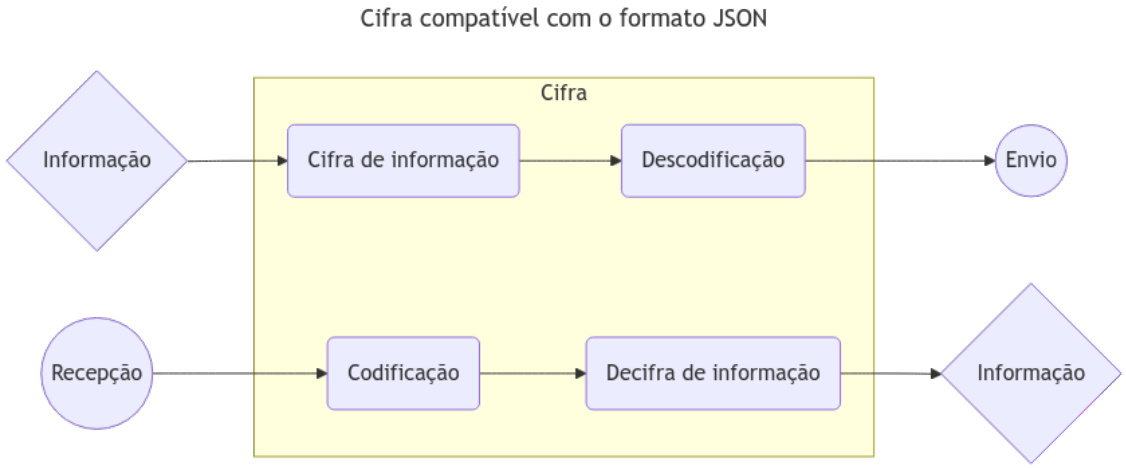

Figura 1.4: Cifra Genérica

1.6.1 Codificação

Codifica o ficheiro em base64 e, de seguida, em utf-8 para possibilitar o envio no formato de string.

```python
    def encode_string2bytes(data):
        return base64.b64decode(codecs.encode(data, 'utf-8'))
```

Descodifica o ficheiro de utf-8 para base64 e de base64 para binario para possibilitar a leito no formato string do ficheiro.

```python
    def decode_bytes2string(data:bytes):
        return codecs.decode(base64.b64encode(data), 'utf-8')
```

1.7 Envio de ficheiro

No Envio de ficheiros do utilizador, é criado um MAC que é enviado após o ficheiro que confirma a autenticidade do mesmo. Este MAC é criado pós cifra do documento. Apesar de ser utilizado apenas para o envio do ficheiro, o código pode ser adaptado para incluir verificação de MAC em todas as mensagens.

```
    def mic_calc(data:bytes) -> bytes:
    digest = hashes.Hash(hashes.SHA256(), backend=default_backend())
    digest.update(data)
    return digest.finalize()

def mac_sign(public_key, data:bytes) -> bytes:
    digest = hashes.Hash(hashes.SHA256(), backend=default_backend())
    digest.update(data)
    mic = digest.finalize()
    mac,size = encrypt_key(public_key, mic)
    return size + mac + data

def mac_check(private_key, data:bytes):
    mac_lenght = int.from_bytes(data[0:KEY_SIZE],byteorder="big")
    data = data[KEY_SIZE:]
    mac = data[0:mac_lenght]
    data = data[mac_lenght:]
    mic = decrypt_key(private_key, mac)
    digest = hashes.Hash(hashes.SHA256(), backend=default_backend())
    digest.update(data)
    mic = digest.finalize()
    return data, mic == mic

def mac_sign_easy(private, data:bytes) -> bytes:
    sign = private.sign(data, assim_padding.PSS(mgf=assim_padding.MGF1(hashes.SHA256()),
    ↪  salt_length=assim_padding.PSS.MAX_LENGTH), hashes.SHA256())
    return sign

def mac_check_easy(public, data:bytes, sign:bytes) -> bool:
    try:
        public.verify(sign, data,
        ↪  assim_padding.PSS(mgf=assim_padding.MGF1(hashes.SHA256()),
        ↪  salt_length=assim_padding.PSS.MAX_LENGTH), hashes.SHA256())
        return True
    except cryptography.exceptions.InvalidSignature:
        return False
```

Capítulo 2

Repository

Este capítulo apresenta as funcionalidades relacionadas ao repositório, detalhando as implementações que garantem a integridade e a segurança do sistema. Todas as funções descritas foram desenvolvidas com foco na validação de permissões, integridade dos dados e autenticação de sessões.

As verificações incluem:

- **Autenticação de Sessões:** Cada operação exige a validação da sessão, incluindo o uso de chaves criptográficas e a validação de interações anteriores.
- **Autorização Baseada em Permissões:** São verificadas as permissões específicas de cada 'role' associada aos utilizadores, garantindo que ações sejam realizadas apenas por sujeitos autorizados.
- **Consistência de Dados:** Os dados armazenados são verificados para evitar duplicações ou inconsistências, como adicionar utilizadores ou documentos já existentes.
- **Controle de Acesso:** A configuração detalhada da *Access Control List* (ACL) assegura que apenas 'roles' adequadas possam aceder ou modificar documentos e permissões.

Com esse modelo de validação, o sistema promove um ambiente seguro e confiável para a manutenção de utilizadores, 'roles', documentos e permissões.

2.1 Auxiliar Functions

As funções auxiliares foram desenvolvidas para simplificar e minimizar a repetição de código, aperfeiçoando os sistemas de segurança essenciais do sistema.

2.1.1 Load Rep Key

Carrega ou cria as chaves pública e privada do repositório, guardando-as de forma segura caso ainda não existam.

```
1  def key_loader():
2      global rep_public_key, rep_private_key
3      if not os.path.isfile("private/rep_public_key.pem") and not
       ↪  os.path.isfile("private/rep_private_key.pem"):
4          rep_private_key, rep_public_key = repUtils.generate_credentials()
5          with open("private/rep_private_key.pem", "wb") as file:
           ↪  file.write(repUtils.serialize_private_key(rep_private_key))
6          with open("private/rep_public_key.pem", "wb") as file:
           ↪  file.write(repUtils.serialize_public_key(rep_public_key))
7          return
8
9      with open("private/rep_private_key.pem", "rb") as file: rep_private_key =
       ↪  load_pem_private_key(file.read(), password=None, backend=default_backend())
10     with open("private/rep_public_key.pem", "rb") as file: rep_public_key =
       ↪  load_pem_public_key(file.read(), backend=default_backend())
```

2.1.2 Save State

Cifra e guarda o estado atual do repositório num ficheiro seguro.

```
def save_state():
    global repository
    os.makedirs("private", exist_ok=True)
    with open("private/repository.cifr", 'wb') as file:
    ↪  file.write(repUtils.encrypt_easy(rep_public_key, json.dumps(repository).encode(),
    ↪  "AES"))
```

2.1.3 Load State

Decifra e carrega o estado salvo do repositório a partir de um ficheiro seguro, caso exista.

```
def load_state():
    global repository
    with open("private/repository.cifr", 'rb') as file: repository =
    ↪  json.loads(repUtils.decrypt_easy(rep_private_key, file.read(), "AES"))
```

2.1.4 Save Key

Cifra e guarda a chave privada de uma organização de forma segura.

```
def save_key(organization_name, private_key):
    global org_private_keys
    org_private_keys[organization_name] = private_key
    os.makedirs("private", exist_ok=True)
    with open("private/private.cifr", 'wb') as file:
    ↪  file.write(repUtils.encrypt_easy(rep_public_key,
    ↪  json.dumps(org_private_keys).encode(), "AES"))
```

2.1.5 Load Org Private Keys

Decifra e carrega as chaves privadas de todas as organizações a partir de um ficheiro seguro.

```
def load_org_private_keys():
    global org_private_keys
    with open("private/private.cifr", 'rb') as file: org_private_keys =
    ↪  json.loads(repUtils.decrypt_easy(rep_private_key, file.read(), "AES"))
```

2.1.6 Get Valid Session Information

Valida as informações de uma sessão, garantindo que esteja ativa e associada a uma organização e utilizador válidos. Pode ainda remover a sessão, caso esta se encontre inativa por mais de trinta minutos.

```
def get_valid_session_information(session_file, previous_interaction):
    organization, session_id = next(iter(json.loads(session_file).items()))
    if organization not in repository: return {"error": "Invalid organization"}, None,
    ↪  None
    if datetime.fromisoformat(previous_interaction) ==
    ↪  datetime.fromisoformat(repository[organization]['sessions'][session_id]['last_active']):
    ↪  return {"abort": "Duplicate request detected"}, None, None
    else: repository[organization]['sessions'][session_id]['last_active'] =
    ↪  previous_interaction
    if session_id not in repository[organization]['sessions']: return {"error": "Invalid
    ↪  session ID"}, None, None
    username = repository[organization]['sessions'][session_id]['username']
```

```
    if not repository[organization]['subjects'][username]['status']: return {"error":
    ↪  "User suspended"}, None, None
    if datetime.now() >
    ↪  datetime.fromisoformat(repository[organization]['sessions'][session_id]['last_active'])
    ↪  + SESSION_TIMEOUT:
        del repository[organization]['sessions'][session_id]
        return {"error": "Session timeout"}, None, None
    return organization, session_id, username
```

2.1.7 Validate Subject Permission

Verifica se um utilizador possui as permissões necessárias para realizar uma operação específica.

```
def validate_subject_permission(organization, session_id, username, permission):
    for role in repository[organization]['sessions'][session_id]['roles']:
        role = repository[organization]['roles'][role]
        if role['status'] and username in role['subjects'] and permission in
        ↪  (role['permissions'] + role['acl']): return True
    return False
```

2.1.8 Validate Document Permission

Confirma se um sujeito possui as permissões necessárias para aceder a um documento específico.

```
def validate_document_permission(organization, session_id, document_name, permission):
    with open(f"metadatas/{document_name}", 'r') as metadata_file:
        metadata = json.load(metadata_file)
        for role in metadata['acl']:
            if role in repository[organization]['sessions'][session_id]['roles'] and
            ↪  permission in metadata['acl'][role]: return True
    return False
```

2.1.9 Compute Hash

Calcula o hash SHA-256 do conteúdo fornecido.

```
def compute_hash(content):
    return hashlib.sha256().update(content).hexdigest()
```

2.1.10 Add Org Private Keys

Adiciona e guarda de forma segura a chave privada de uma organização.

```
def add_org_private_key(org_name, private_key):
    global org_private_keys
    org_private_keys[org_name] = repUtils.serialize_private_key(private_key).decode()
    os.makedirs("private", exist_ok=True)
    with open("private/private.cifr", 'wb') as file:
        data = repUtils.encrypt_easy(rep_public_key,
        ↪  json.dumps(org_private_keys).encode(), "AES")
        file.write(data)
```

2.1.11 Load Org Private Keys

Carrega todas as chaves privadas das organizações de forma segura, garantindo a uma decifragem correta e concisa.

```python
def load_org_private_keys():
    global org_private_keys
    if os.path.isfile("private/private.cifr"):
        with open("private/private.cifr", 'rb') as file:
            data = file.read()
            data = repUtils.decrypt_easy(rep_private_key, data, "AES")
            org_private_keys = json.loads(data)
```

2.2 Anonymous API Commands

As funções anónimas podem ser executadas sem a criação prévia de uma sessão e não estão sujeitas a verificações. Apesar de serem anónimas, foram desenvolvidas garantindo a confidencialidade das informações trocadas do cliente para o repositório.

2.2.1 Get Rep Key

Obtém e devolve a chave pública do repositório, a fim de serem estabelecidas comunicações seguras.

```python
@app.route('/repository/get_rep_key', methods=['GET'])
def get_rep_key():
    return jsonify({"public_key": repUtils.serialize_public_key(rep_public_key).decode()})
```

2.2.2 Rep Create Org

Cria uma nova organização no repositório, atribuindo chaves criptográficas à organização e assegurando as permissões essenciais da 'role' "Managers".

```python
@app.route('/repository/create_org', methods=['POST'])
def rep_create_org():
    req = repUtils.decrypt_request(rep_private_key, request)
    public_key = json.loads(req.files["public_key_file"])['pubkey']
    organization = req.form['organization']
    identifier = req.form["identifier"]
    name = req.form["name"]
    email = req.form["email"]

    if organization in repository: return jsonify({"error": "Organization already
    ↪  exists"}), 400
    org_private_key, org_public_key = repUtils.generate_credentials()
    repository[organization] = {
        "documents": [],
        "subjects": {
            identifier: {
                "name": name,
                "email": email,
                "pubkey": public_key,
                "status": True
            }
        },
        "roles": {
            "Managers": {
                "subjects": [identifier],
                "acl": ["DOC_NEW", "SUBJECT_NEW", "SUBJECT_DOWN", "SUBJECT_UP",
                ↪  "ROLE_ACL"],
                "permissions": ["ROLE_NEW", "ROLE_DOWN", "ROLE_UP", "ROLE_MOD"],
                "status": True
            }
        },
        "sessions": {},
        "public_key": repUtils.serialize_public_key(org_public_key).decode()
    }
```

```
33
34      save_state()
35      save_key(organization, repUtils.serialize_private_key(org_private_key).decode())
36      return jsonify({"organization_public_key":
        ↪  repUtils.serialize_public_key(org_public_key).decode()})
```

2.2.3 Rep List Orgs

Devolve a lista de organizações registadas no repositório.

```
1   @app.route('/repository/list_orgs', methods=['GET'])
2   def rep_list_orgs():
3       return jsonify({"organizations": list(repository.keys())})
```

2.2.4 Rep Create Session

Cria uma nova sessão para um utilizador numa organização, criando um identificador único para a mesma. Utiliza os parâmetros 'start_time' e 'last_active' para acompanhar a criação e a última interação da sessão com o repositório.

Estes parâmetros permitem a validação de 'timeouts', encerrando a sessão em caso de inatividade e garantindo a segurança do sistema.

```
1   @app.route('/repository/organization/sessions/create_session', methods=['POST'])
2   def rep_create_session():
3       request_data = repUtils.decrypt_request(rep_private_key, request)
4       public_key = json.loads(request_data.files["credentials_file"])['pubkey']
5       organization = request_data.form['organization']
6       identifier = request_data.form['identifier']
7       password = request_data.form['password']
8       session_id = hashlib.sha256(public_key.replace("\n",
        ↪  "").strip().encode('utf-8')).hexdigest()
9
10      if organization not in repository: return jsonify({"error": "Invalid organization"}),
        ↪  400
11      if identifier not in repository[organization]['subjects']: return jsonify({"error":
        ↪  "Invalid username"}), 400
12      if not repository[organization]['subjects'][identifier]['status']: return
        ↪  jsonify({"error": "User suspended"}), 400
13      if session_id in repository[organization]['sessions']: return jsonify({"error":
        ↪  "Session already active"}), 400
14
15      repository[organization]["sessions"][session_id] = {
16          "username": identifier,
17          "keys": [public_key],
18          "roles" : [],
19          "start_time": datetime.now().isoformat(),
20          "last_active": datetime.now().isoformat()
21      }
22      save_state()
23      return jsonify({organization: session_id})
```

2.2.5 Rep Get File

Permite o download de um ficheiro do repositório a partir da sua 'file_handle', garantindo a integridade com uma assinatura digital.

```
1   @app.route('/repository/files/get_file', methods=['GET'])
2   def rep_get_file():
3       file_handle = request.form['file_handle']
4       if not os.path.exists(f"files/{file_handle}"): return jsonify({"error": "Invalid
        ↪  document"}), 400
```

```
    with open(f"files/{file_handle}", 'rb') as file:
        content = file.read()
        mac = repUtils.mac_sign_easy(rep_private_key, content)
        buff = io.BufferedReader(io.BytesIO(len(mac).to_bytes(20, byteorder='big') + mac +
        ↪  content))
        return send_file(buff, as_attachment=True, download_name="save")
```

2.3 Authenticated API Commands

As funções autenticadas utilizam sessões criadas previamente com IDs únicos e são validadas rigorosamente. As validações incluem verificação de 'timestamps' para negar acesso a utilizadores inativos e prevenção de pedidos duplicados, garantindo também a integridade das operações.

Em caso de falha em alguma fase das validações, é retornado um erro apropriado.

2.3.1 Rep Assume Role

Permite um utilizador assumir uma 'role' específica dentro de uma organização, durante a sua sessão.

O sistema previne a assunção de 'roles' já assumidas, ou que o utilizador não tem acesso, ou que não existem.

```
@app.route('/repository/organization/roles/assume_role', methods=['POST'])
def rep_assume_role():
    request_data = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username =
    ↪  get_valid_session_information(request_data.files["session_file"],
    ↪  request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is
    ↪  an error message
    role = request_data.form["role"]

    if role not in repository[organization]["roles"]: return jsonify({"error": "Invalid
    ↪  role"}), 400
    if role in repository[organization]['sessions'][session_id]['roles']: return
    ↪  jsonify({"error": "Role already assigned"}), 400
    if username not in repository[organization]['roles'][role]['subjects']: return
    ↪  jsonify({"error": "Invalid permission"}), 400

    repository[organization]['sessions'][session_id]['roles'].append(role)
    save_state()
    return jsonify({"success": True})
```

2.3.2 Rep Drop Role

Remove uma 'role' específica da sessão de um utilizador.

O sistema previne a remoção de 'roles' não assumidas durante a sessão, ou que não existam.

```
@app.route('/repository/organization/roles/drop_role', methods=['POST'])
def rep_drop_role():
    request_data = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username =
    ↪  get_valid_session_information(request_data.files["session_file"],
    ↪  request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is
    ↪  an error message
    role = request_data.form["role"]

    if role not in repository[organization]['sessions'][session_id]['roles']: return
    ↪  jsonify({"error": "Invalid role"}), 400

    repository[organization]['sessions'][session_id]['roles'].remove(role)
```

```
    save_state()
    return jsonify({"success": True})
```

2.3.3 Rep List Roles

Lista todas as 'roles' associadas à sessão de um utilizador.

```
@app.route('/repository/organization/list_roles', methods=['GET'])
def rep_list_roles():
    request_data : Req = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username =
    ↪  get_valid_session_information(request_data.files["session_file"],
    ↪  request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is
    ↪  an error message

    return jsonify({"roles": repository[organization]['sessions'][session_id]['roles']})
```

2.3.4 Rep List Subjects

Lista os sujeitos (utilizadores) de uma organização e o seu estado ativo/inativo, podendo ser feita uma filtragem a partir de um 'username' específico.

```
@app.route('/repository/organization/list_subjects', methods=['GET'])
def rep_list_subjects():
    request_data : Req = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username =
    ↪  get_valid_session_information(request_data.files["session_file"],
    ↪  request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is
    ↪  an error message
    identifier = request_data.form.get("identifier")

    if identifier and identifier in repository[organization]["subjects"]: return
    ↪  jsonify({identifier: repository[organization]["subjects"][identifier]["status"]})
    return jsonify({subject: repository[organization]["subjects"][subject]["status"] for
    ↪  subject in repository[organization]["subjects"]})
```

2.3.5 Rep List Role Subjects

Lista os sujeitos associados a uma 'Role' específica numa organização.

```
@app.route('/repository/organization/roles/list_role_subjects', methods=['GET'])
def rep_list_role_subjects():
    request_data : Req = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username =
    ↪  get_valid_session_information(request_data.files["session_file"],
    ↪  request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is
    ↪  an error message
    role = request_data.form["role"]

    if role not in repository[organization]["roles"]: return jsonify({"error": "Invalid
    ↪  role"}), 400

    return jsonify({"roles": repository[organization]['roles'][role]['subjects']})
```

2.3.6 Rep List Subject Roles

Lista todas as 'roles' associadas a um sujeito específico numa organização.

```python
@app.route('/repository/organization/subjects/list_subject_roles', methods=['GET'])
def rep_list_subject_roles():
    request_data : Req = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username = get_valid_session_information(request_data.files["session_file"], request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is an error message
    identifier = request_data.form["identifier"]

    if identifier not in repository[organization]["subjects"]: return jsonify({"error": "Invalid subject"}), 400

    return jsonify({"roles": [role for role in repository[organization]['roles'] if identifier in repository[organization]['roles'][role]['subjects']]})
```

2.3.7 Rep List Role Permissions

Lista todas as permissões associadas a uma 'role' específica numa organização.

```python
@app.route('/repository/organization/roles/list_role_permissions', methods=['GET'])
def rep_list_role_permissions():
    request_data : Req = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username = get_valid_session_information(request_data.files["session_file"], request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is an error message
    role = request_data.form["role"]

    if role not in repository[organization]['roles']: return jsonify({"error": "Invalid role"}), 400

    return jsonify({"permissions": repository[organization]['roles'][role]['permissions'] + repository[organization]['roles'][role]['acl']})

```

2.3.8 Rep List Permission Roles

Lista todas as 'roles' que contêm uma permissão específica numa organização.

```python
@app.route('/repository/organization/roles/list_permission_roles', methods=['GET'])
def rep_list_permission_roles():
    request_data : Req = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username = get_valid_session_information(request_data.files["session_file"], request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is an error message
    permission = request_data.form["permission"]

    if permission not in ["DOC_NEW", "SUBJECT_NEW", "SUBJECT_DOWN", "SUBJECT_UP", "ROLE_ACL", "ROLE_NEW", "ROLE_DOWN", "ROLE_UP", "ROLE_MOD"]: return jsonify({"error": "Invalid permission"}), 400

    return jsonify({"roles": [role for role in repository[organization]['roles'] if permission in (repository[organization]['roles'][role]['permissions'] + repository[organization]['roles'][role]['acl'])]})
```

2.3.9 Rep List Docs

Lista os documentos pertencentes a uma organização, com filtros opcionais por criador e data.

```python
@app.route('/repository/organization/documents/list_docs', methods=['GET'])
def rep_list_docs():
    request_data : Req = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username = get_valid_session_information(request_data.files["session_file"], request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is an error message
    identifier = request_data.form.get("identifier", None)
    date = request_data.form.get("date", None)

    if identifier and not date: return jsonify({ "documents": [document for document in os.listdir('metadatas') if document in repository[organization]['documents'] and json.load(open(f"metadatas/{document}"))['creator'] == identifier]})
    if not identifier and date: return jsonify({ "documents": [document for document in os.listdir('metadatas') if document in repository[organization]['documents'] and json.load(open(f"metadatas/{document}"))['create_data'] == date]})
    if identifier and date: return jsonify({ "documents": [document for document in os.listdir('metadatas') if document in repository[organization]['documents'] and json.load(open(f"metadatas/{document}"))['creator'] == identifier and json.load(open(f"metadatas/{document}"))['create_data'] == date]})
    return jsonify({ "documents": [document for document in os.listdir('metadatas') if document in repository[organization]['documents']]})
```

2.4 Authorized API Commands

Os comandos autorizados abrangem várias operações fundamentais que servem de manutenção ao repositório seguro, utilizando permissões e controlo de acessos para além das sessões previamente usadas.

2.4.1 Rep Add Subject

Adiciona um novo sujeito ao repositório, prevenindo a adição de sujeitos já adicionados.

```python
@app.route('/repository/organization/subjects/add_subject', methods=['POST'])
def rep_add_subject():
    request_data : Req = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username = get_valid_session_information(request_data.files["session_file"], request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is an error message
    identifier = request_data.form["identifier"]
    name = request_data.form["name"]
    email = request_data.form["email"]
    public_key = json.loads(request_data.files["credentials_file"])['pubkey']

    if not validate_subject_permission(organization, session_id, username, 'SUBJECT_NEW'): return jsonify({"error": "Permission denied"}), 400
    if identifier in repository[organization]['subjects']: return jsonify({"error": "User already exists"}), 400

    repository[organization]['subjects'][identifier] = {
        "name": name,
        "email": email,
        "pubkey": public_key,
        "roles": [],
        "status": True
    }
    save_state()
    return jsonify({"success": True})
```

2.4.2 Rep Suspend Subject

Suspende um sujeito específico, impedindo a suspensão no caso do sujeito já se encontrar suspenso.

```
@app.route('/repository/organization/subjects/suspend_subject', methods=['POST'])
def rep_suspend_subject():
    request_data : Req = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username =
    ↪  get_valid_session_information(request_data.files["session_file"],
    ↪  request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is
    ↪  an error message
    identifier = request_data.form['identifier']

    if not validate_subject_permission(organization, session_id, username,
    ↪  'SUBJECT_DOWN'): return jsonify({"error": "Permission denied"}), 400
    if identifier not in repository[organization]['subjects']: return jsonify({"error":
    ↪  "Invalid username"}), 400
    if not repository[organization]['subjects'][identifier]['status']: return
    ↪  jsonify({"error": "Subject already suspended"}), 400

    repository[organization]['subjects'][identifier]['status'] = False
    save_state()
    return jsonify({"success": True})
```

2.4.3 Rep Activate Subject

Reativa um sujeito, garantindo que este se encontrava previamente suspenso.

```
@app.route('/repository/organization/subjects/activate_subject', methods=['POST'])
def rep_activate_subject():
    request_data : Req = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username =
    ↪  get_valid_session_information(request_data.files["session_file"],
    ↪  request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is
    ↪  an error message
    identifier = request_data.form['identifier']

    if not validate_subject_permission(organization, session_id, username, 'SUBJECT_UP'):
    ↪  return jsonify({"error": "Permission denied"}), 400
    if identifier not in repository[organization]['subjects']: return jsonify({"error":
    ↪  "Invalid username"}), 400
    if repository[organization]['subjects'][identifier]['status']: return
    ↪  jsonify({"error": "Subject already active"}), 400

    repository[organization]['subjects'][identifier]['status'] = True
    save_state()
    return jsonify({"success": True})
```

2.4.4 Rep Add Role

Criar uma nova 'role' numa organização, sem permissões associadas.

```
@app.route('/repository/organization/roles/add_role', methods=['POST'])
def rep_add_role():
    request_data : Req = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username =
    ↪  get_valid_session_information(request_data.files["session_file"],
    ↪  request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is
    ↪  an error message
    role = request_data.form["role"]
```

```
7
8       if not validate_subject_permission(organization, session_id, username, 'ROLE_NEW'):
        ↪  return jsonify({"error": "Permission denied"}), 400
9       if role in repository[organization]["roles"]: return jsonify({"error": "Role already
        ↪  exists"}), 400
10
11      repository[organization]['roles'][role] = {
12          "subjects": [],
13          "acl": [],
14          "permissions": [],
15          "status": True
16      }
17      save_state()
18      return jsonify({"success": True})
```

2.4.5 Rep Suspend Role

Suspende uma 'role' existente numa organização, verificando o 'status' da mesma previamente. **Correção:** Previne a suspensão da 'role' "Managers".

```
1   @app.route('/repository/organization/roles/suspend_role', methods=['POST'])
2   def rep_suspend_role():
3       request_data : Req = repUtils.decrypt_request(rep_private_key, request)
4       organization, session_id, username =
        ↪  get_valid_session_information(request_data.files["session_file"],
        ↪  request_data.form["previous_interaction"])
5       if not username: return jsonify(organization), 400 # In this case, 'organization' is
        ↪  an error message
6       role = request_data.form["role"]
7
8       if role == "Managers": return jsonify({"abort": "Role cannot be suspended"}), 400
9       if not validate_subject_permission(organization, session_id, username, 'ROLE_DOWN'):
        ↪  return jsonify({"error": "Permission denied"}), 400
10      if role not in repository[organization]["roles"]: return jsonify({"error": "Invalid
        ↪  role"}), 400
11      if not repository[organization]["roles"][role]['status']: return jsonify({"error":
        ↪  "Role already suspended"}), 400
12
13      repository[organization]['roles'][role]['status'] = False
14      save_state()
15      return jsonify({"success": True})
```

2.4.6 Rep Reactivate Role

Reativa uma role existente numa organização, verificando o 'status' da mesma previamente.

```
1   @app.route('/repository/organization/roles/reactivate_role', methods=['POST'])
2   def rep_reactivate_role():
3       request_data : Req = repUtils.decrypt_request(rep_private_key, request)
4       organization, session_id, username =
        ↪  get_valid_session_information(request_data.files["session_file"],
        ↪  request_data.form["previous_interaction"])
5       if not username: return jsonify(organization), 400 # In this case, 'organization' is
        ↪  an error message
6       role = request_data.form["role"]
7
8       if not validate_subject_permission(organization, session_id, username, 'ROLE_UP'):
        ↪  return jsonify({"error": "Permission denied"}), 400
9       if role not in repository[organization]["roles"]: return jsonify({"error": "Invalid
        ↪  role"}), 400
10      if repository[organization]["roles"][role]['status']: return jsonify({"error": "Role
        ↪  already active"}), 400
11
```

```
    repository[organization]['roles'][role]['status'] = True
    save_state()
    return jsonify({"success": True})
```

2.4.7 Rep Add Permission

Adiciona sujeitos ou permissões a uma 'role' específica numa organização, assegurando previamente a existência de ambos.

```
@app.route('/repository/organization/roles/add_permission', methods=['POST'])
def rep_add_permission():
    request_data : Req = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username =
    ↪  get_valid_session_information(request_data.files["session_file"],
    ↪  request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is
    ↪  an error message
    role = request_data.form["role"]
    identifier = request_data.form["identifier"]

    if not validate_subject_permission(organization, session_id, username, 'ROLE_MOD'):
    ↪  return jsonify({"error": "Permission denied"}), 400
    if role not in repository[organization]["roles"]: return jsonify({"error": "Invalid
    ↪  role"}), 400

    if identifier.isalnum():
        if identifier not in repository[organization]['subjects']: return
        ↪  jsonify({"error": "Invalid username"}), 400
        if identifier in repository[organization]["roles"][role]['subjects']: return
        ↪  jsonify({"error": "Role already assigned"}), 400
        repository[organization]["roles"][role]['subjects'].append(identifier)
        save_state()
        return jsonify({"success": True})

    if identifier in repository[organization]["roles"][role]['permissions']: return
    ↪  jsonify({"error": "Permission already exists"}), 400
    if identifier in ["ROLE_NEW", "ROLE_DOWN", "ROLE_UP", "ROLE_MOD"]:
    ↪  repository[organization]["roles"][role]['permissions'].append(identifier)
    elif identifier in ["DOC_NEW", "SUBJECT_NEW", "SUBJECT_DOWN", "SUBJECT_UP",
    ↪  "ROLE_ACL"]: repository[organization]["roles"][role]['acl'].append(identifier)
    else: return jsonify({"error": "Invalid permission"}), 400
    save_state()
    return jsonify({"success": True})
```

2.4.8 Rep Remove Permission

Remove sujeitos ou permissões de uma 'role' específica numa organização, assegurando a existência dos mesmos previamente.

Para além das funcionalidades básicas, o sistema retorna um aviso, com a possibilidade de forçar o comando, caso o utilizador tente retirar uma permissão crítica associada a si mesmo (p. ex: retirar "ROLE_MOD"da 'role' associada ao mesmo utilizador).

```
@app.route('/repository/organization/roles/remove_permission', methods=['POST'])
def rep_remove_permission():
    request_data = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username =
    ↪  get_valid_session_information(request_data.files["session_file"],
    ↪  request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is
    ↪  an error message
    role = request_data.form["role"]
    identifier = request_data.form["identifier"]
    force_flag = int(request_data.form["force_flag"])
```

```
9
10      if role not in repository[organization]["roles"]: return jsonify({"error": "Invalid
        ↪  role"}), 400
11      if not validate_subject_permission(organization, session_id, username, 'ROLE_MOD'):
        ↪  return jsonify({"error": "Permission denied"}), 400
12
13      if identifier.isalnum():
14          if role == "Managers" and len(repository[organization]["roles"][role]['subjects'])
            ↪  == 2: return jsonify({"error": "Role must have at least 2 members"}), 400
15          if identifier not in repository[organization]["roles"][role]['subjects']: return
            ↪  jsonify({"error": "Invalid username"}), 400
16          repository[organization]["roles"][role]['subjects'].remove(identifier)
17          if not force_flag and not validate_subject_permission(organization, session_id,
            ↪  username, 'ROLE_MOD'):
18              repository[organization]["roles"][role]['subjects'].append(identifier)
19              return jsonify({"abort": "Use f/force to remove user admin privileges"}), 400
20          save_state()
21          return jsonify({"success": True})
22
23      if identifier == "ROLE_MOD" and role == "Managers": return jsonify({"error":
        ↪  "Permission cannot be removed from role"}), 400
24      if identifier in repository[organization]["roles"][role]['permissions']:
        ↪  repository[organization]["roles"][role]['permissions'].remove(identifier)
25      elif identifier in repository[organization]["roles"][role]['acl']:
        ↪  repository[organization]["roles"][role]['acl'].remove(identifier)
26      else: return jsonify({"error": "Invalid permission"}), 400
27      if not force_flag and not validate_subject_permission(organization, session_id,
        ↪  username, 'ROLE_MOD'):
28              repository[organization]["roles"][role]['permissions'].append(identifier)
29              return jsonify({"abort": "Use f/force to remove user admin privileges"}), 400
30      save_state()
31      return jsonify({"success": True})
```

2.4.9 Rep Add Doc

Adiciona um novo documento ao repositório associado a uma organização, garantindo a sua integridade, armazenando de forma segura os parâmetros críticos da 'metadata' referentes ao documento.

```
1   @app.route('/repository/organization/documents/add_doc', methods=['POST'])
2   def rep_add_doc():
3       request_data : Req = repUtils.decrypt_request(rep_private_key, request)
4       organization, session_id, username =
        ↪  get_valid_session_information(request_data.files["session_file"],
        ↪  request_data.form["previous_interaction"])
5       if not username: return jsonify(organization), 400 # In this case, 'organization' is
        ↪  an error message
6       document_name = request_data.form["document_name"]
7       file_content = request_data.files['file']
8       metadata = request_data.form['metadata']
9
10      if not validate_subject_permission(organization, session_id, username, 'DOC_NEW'):
        ↪  return jsonify({"error": "Permission denied"}), 400
11      if document_name in repository[organization]['documents'] or
        ↪  os.path.exists(f"files/{metadata['file_handle']}"): return jsonify({"error": "File
        ↪  already exists"}), 400
12      os.makedirs(f"files/", exist_ok=True)
13      os.makedirs(f"metadatas/", exist_ok=True)
14      with open(f"files/{metadata['file_handle']}", 'wb') as file: file.write(file_content)
15      with open(f"metadatas/{document_name}", 'w') as metadata_file:
16          json.dump({
17              "document_handle": None,
18              "create_date": datetime.now().isoformat(),
19              "creator": username,
20              "file_handle": metadata['file_handle'],
21              "acl": {"Managers": ["DOC_ACL", "DOC_READ", "DOC_DELETE"]},
```

```
            "deleter": None,
            "key":
            ↪  repUtils.decode_bytes2string(repUtils.encrypt_easy(load_pem_public_key(repository[organizatio
            ↪  backend=default_backend()), repUtils.encode_string2bytes(metadata['key']),
            ↪  "AES")),
            "algorithm": metadata["algorithm"],
        }, metadata_file)

    repository[organization]['documents'].append(document_name)
    save_state()
    return jsonify({"success": True})
```

2.4.10 Rep Get Doc Metadata

Obtém a 'metadata' de um documento específico associado a uma organização, decifrando os parâmetros necessários, mas assegurando a transmissão segura dos dados para o cliente.

```
@app.route('/repository/organization/documents/get_doc_metadata', methods=['GET'])
def rep_get_doc_metadata():
    request_data : Req = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username =
    ↪  get_valid_session_information(request_data.files["session_file"],
    ↪  request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is
    ↪  an error message
    document_name = request_data.form["document_name"]

    if not validate_document_permission(organization, session_id, document_name,
    ↪  'DOC_READ'): return jsonify({"error": "Permission denied"}), 400
    if document_name not in repository[organization]['documents']: return
    ↪  jsonify({"error": "Invalid document"}), 400

    with open(f"metadatas/{document_name}", 'r') as metadata_file:
        metadata_file = metadata_file.read()

        content = repUtils.encrypt_easy(repository[organization][username]["pubkey"],
        ↪  metadata_file.encode(), "AES")
        mac = repUtils.mac_sign_easy(rep_private_key, content)
        s = len(mac).to_bytes(20, byteorder='big')

        buff = io.BufferedReader(io.BytesIO(s + mac + content))
        return send_file(buff, as_attachment=True, download_name="save")
```

2.4.11 Rep Get Doc File

Obtém um documento específico associado a uma organização, assim como a sua 'metadata', assegurando a transmissão segura dos mesmos.

```
@app.route('/repository/organization/documents/get_doc_file', methods=['GET'])
def rep_get_doc_file():
    request_data : Req = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username =
    ↪  get_valid_session_information(request_data.files["session_file"],
    ↪  request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is
    ↪  an error message
    document_name = request_data.form["document_name"]
    info = request_data.form["info"]

    if not validate_document_permission(organization, session_id, document_name,
    ↪  'DOC_READ'): return jsonify({"error": "Permission denied"}), 400
    if document_name not in repository[organization]['documents']: return
    ↪  jsonify({"error": "Invalid document"}), 400
```

```
    if info == "meta":
        with open(f"metadatas/{document_name}", 'r') as metadata_file:
            metadata_file = json.loads(metadata_file)
            return jsonify({"res": metadata_file})

    if info == "file":
        with open(f"files/{metadata_file['file_handle']}", 'r') as file:
            content = file.read()
            mac = repUtils.mac_sign_easy(rep_private_key, content)
            s = len(mac).to_bytes(20, byteorder='big')

            buff = io.BufferedReader(io.BytesIO(s + mac + content))
            return send_file(buff, as_attachment=True, download_name="save")
```

2.4.12 Rep Delete Doc

Remove um documento através da destruição da associação do ficheiro com a sua 'metadata', e não propriamente o conteúdo do documento. Além disso, remove ainda o documento da lista de documentos da organização ao qual estava associado.

```
@app.route('/repository/organization/documents/delete_doc', methods=['POST'])
def rep_delete_doc():
    request_data : Req = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username =
    ↪  get_valid_session_information(request_data.files["session_file"],
    ↪  request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is
    ↪  an error message
    document_name = request_data.form["document_name"]

    if not validate_document_permission(organization, session_id, document_name,
    ↪  'DOC_DELETE'): return jsonify({"error": "Permission denied"}), 400
    if document_name not in repository[organization]['documents']: return
    ↪  jsonify({"error": "Invalid document"}), 400

    if os.path.isdir('metadatas') : os.makedirs("metadatas", exist_ok=True)

    with open(f"metadatas/{document_name}", 'r') as metadata_file:
        metadata = json.load(metadata_file)
        file_handle = metadata["file_handle"]
        metadata["file_handle"] = None
    with open(f"metadatas/{document_name}", 'w') as metadata_file:
        json.dump(metadata, metadata_file)
    repository[organization]['documents'].remove(document_name)
    save_state()
    return jsonify({"file_handle": file_handle})
```

2.4.13 Rep Acl Doc

Altera as permissões de controlo de acesso (ACL) de um documento específico, protegendo a permissão "DOC_ACL"da 'role' "Managers".

```
@app.route('/repository/organization/documents/acl_doc', methods=['POST'])
def rep_acl_doc():
    request_data : Req = repUtils.decrypt_request(rep_private_key, request)
    organization, session_id, username =
    ↪  get_valid_session_information(request_data.files["session_file"],
    ↪  request_data.form["previous_interaction"])
    if not username: return jsonify(organization), 400 # In this case, 'organization' is
    ↪  an error message
    document_name = request_data.form["document_name"]
    operation = int(request_data.form["operation"])
```

```
    role = request_data.form["role"]
    permission = request_data.form["permission"]

    if not validate_document_permission(organization, session_id, document_name, 'DOC_ACL'): return jsonify({"error": "Permission denied"}), 400
    if document_name not in repository[organization]['documents']: return jsonify({"error": "Invalid document"}), 400

    with open(f"metadatas/{document_name}", 'r') as metadata_file:
        metadata = json.load(metadata_file)
        if role not in metadata['acl']: return jsonify({"error": "Invalid role"}), 400
        if operation:
            if permission in metadata['acl'][role]: return jsonify({"error": "Permission already exists"}), 400
            metadata['acl'][role].append(permission)
        else:
            if permission == "DOC_ACL" and role == "Managers": return jsonify({"error": "Permission cannot be removed from role"}), 400
            if permission not in metadata['acl'][role]: return jsonify({"error": "Invalid permission"}), 400
            metadata['acl'][role].remove(permission)

    if os.path.exists("metadatas/"): os.makedirs("metadatas/", exist_ok=True)
    with open(f"metadatas/{document_name}", 'w') as metadata_file:
        json.dump(metadata, metadata_file)
    save_state()
    return jsonify({"success": True})
```

Capítulo 3

Aplication Security Verification Standats (ASVS)

De acordo com o que foi proposto, tendo em conta o ASVS 4.0.3, foram escolhidos padrões de verificação dos conteúdos do capítulo 6 do ASVS, que descrevem padrões referentes a cifragem armazenada.

1. 6.1.1 : Os ficheiros pessoais estão cifrados, excepto a 'metadata'. No entanto, o sistema está preparado para cifrar essa informação (com alterações mínimas). (não verificado)
2. 6.1.2 e 6.1.3 : todos os ficheiros não referentes a 'metadata' são cifrados.(verificado)
3. 6.2.1 : Apesar dos erros serem manuseados na sua maioria, não é possível garantir proteção contra Padding Oracle attacks. (não verificado)
4. 6.2.2 : Cifras programadas por defeito são utilizadas. (não verificado)
5. 6.2.3 : (não verificado)
6. 6.2.4 : É possível trocar algoritmos e tamanhos de chave com certa facilidade no código, mas não é uma função disponível para o utilizador. (discutível)
7. 6.2.5 : Apesar de algoritmos menos seguros serem possíveis de serem utilizados, por defeito, o programa utiliza algoritmos mais seguros. (verificado)
8. 6.2.6 : Os números aleatórios criados são adaptados para cada algoritmo. Porém, apesar da chance de repetição ser baixa, não é garantidamente nula. (não verificado)
9. 6.2.7 : Autenticação com MAC é utilizada apenas para transmissão de ficheiros. MIC's são utilizados para os ficheiros guardados. (verificado)
10. 6.2.8 : (não verificado)
11. 6.3.1 : É utilizado OS.random() do python que garante imprevisibilidade na criação de números aleatórios. (verificado)
12. 6.3.2 : (não verificado)
13. 6.3.3 : (não verificado)
14. 6.4.1 : (não verificado)
15. 6.4.2 : (não verificado)

Dos 16 padrões abordados 6 são cumpridos pelo sistema criado.