



Dienst Uitvoering Onderwijs  
*Ministerie van Onderwijs, Cultuur en  
Wetenschap*

# Technische Realisatie

## Risicoanalyse Public Cloud

Versie 1.0

David van der Meer  
424665  
da.van.der.meer@st.hanze.nl

# Risicoanalyse Public Cloud



Auteur: David van der Meer  
Studentnummer: 424665  
E-mailadres: da.van.der.meer@st.hanze.nl  
Instituut: Hanze University of Applied Sciences  
Opleiding: HBO-ICT  
Major: Network & Security Engineering  
Bedrijfsbegeleider: Hiddo Hut  
Docentbegeleider: Roy van Leeuwen  
Plaats & Datum: Groningen, 17 juni 2024  
<https://www.hanze.nl>

## Inhoud

1.	Inleiding.....	4
2.	Ontwerp/Architectuur .....	5
2.1	Concept .....	5
2.2	Visueel ontwerp .....	5
2.3	S3 met Boto3 en CSP .....	6
2.4	Encryptie .....	6
2.5	Fragmentatie.....	7
3.	Resultaten Python .....	8
3.1	Werking van de tool .....	8
4.	Beschrijving code .....	11
5.	Encryptie tool.....	14
5.1	Cryptomator .....	14
5.2	Werking Cryptomator.....	15
6.	Effect op Risicoanalyse .....	17
7.	Vergelijking Cryptomator en het POC .....	18
7.1	Overeenkomsten.....	18
7.2	Verschillen.....	18
7.3	Voor- en nadelen .....	18
8.	Conclusie .....	19
9.	Bijlage .....	20
9.1	Bijlage 1: Python code van het Proof-of-Concept .....	20
9.2	Bijlage 2: Resultaten Risk Treatment.....	22

# 1. Inleiding

De risicoanalyse die gedaan wordt heeft als doel de risico's van Public Cloud inzichtelijk te maken voor DUO. De laatste stap in de risicoanalyse is de Risk Treatment. Bij deze fase wordt gekeken naar de maatregelen tegen de gevonden risico's. De Risk Treatment wordt bij deze risicoanalyse gedaan door middel van deze Technische Realisatie. Er wordt dus gekeken naar beveiligingsmaatregelen die de risico's uit de risicoanalyse kunnen verminderen en helemaal kunnen afschermen.

Het doel van deze technische realisatie is daarom het maken van een Proof-of-Concept (POC). Dit zal gedaan worden door het implementeren van één of meerdere beveiligingsmaatregelen uit de risicoanalyse. Van dit POC zal eerst een ontwerp gemaakt worden waarbij de keuzes toegelicht zullen worden. Hierna zal het ontwerp gerealiseerd worden. Ook wordt er gekeken naar een bestaande tool met dezelfde werking.

Het gaat hier om het maken van een Proof-of-Concept, het doel is dus het aantonen dat er beveiligingsmaatregelen zijn tegen Public Cloud. Daarnaast moet er gekeken worden wat voor effect deze maatregel(en) op de risicoanalyse heeft.

Op basis van deze informatie kunnen er enkele vragen opgesteld worden. Deze vragen zijn:

- 1. Hoe kunnen er beveiligingsmaatregelen geïmplementeerd worden in de Public Cloud.**
- 2. Wat voor invloed heeft het POC en de bestaande tool op het BIV-model?**
- 3. Welke risico's worden er gemitigeerd worden met het POC en de bestaande tool?**

Na afloop van deze realisatie zal er antwoord gegeven worden op deze vragen. Maar eerst zal het ontwerp besproken worden die gehanteerd wordt voor dit POC. Dit ontwerp met de gemaakte keuzes zullen hier toelicht worden. Daarnaast zal een bestaande tool besproken worden die getest is. Het gerealiseerde POC zal hierna besproken worden inclusief de werking hiervan. De bestaande tool komt hierna aanbod. Deze bestaande tool en de gemaakte tool zullen daarna vergeleken worden. Zo komen de overeenkomsten, voordelen en nadelen van beide tools aan het licht. Ten slotte zal tijdens de conclusie antwoord gegeven worden op de drie onderzoeksvragen bij deze technische realisatie.

## 2. Ontwerp/Architectuur

Voor deze realisatie moet er dus gekozen worden voor technische beveiligingsmaatregelen. Deze maatregelen hebben het doel om de risico's te verminderen of zelf helemaal af te schermen door middel van een technische oplossing. Tijdens deze realisatie worden dus één of meerdere van deze maatregelen toegepast op een Public Cloud omgeving.

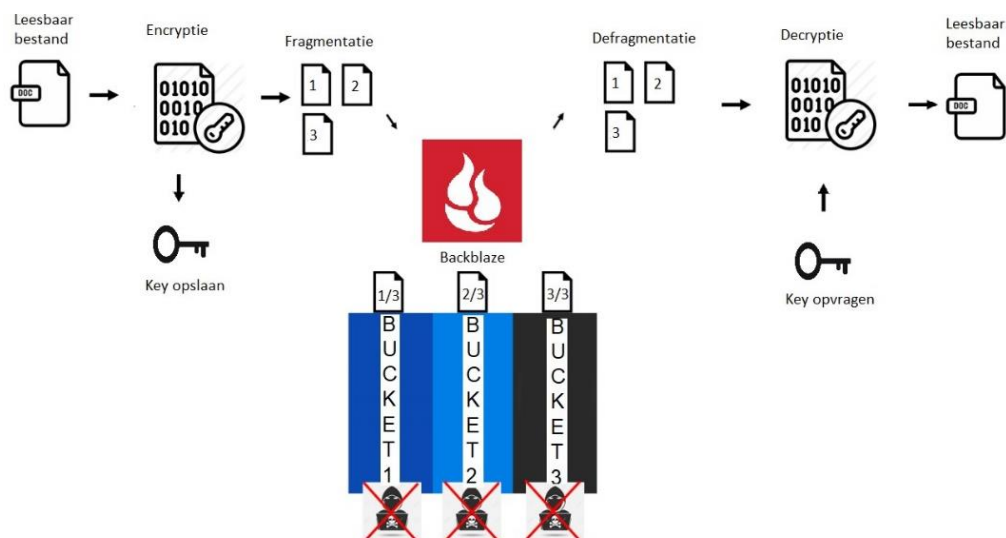
### 2.1 Concept

Er is in eerste instantie gekozen om een concept te gebruiken als uitgangspunt. Dit concept is afkomstig van een TNO onderzoek genaamd "Veilig werken in de cloud". Dit concept is vooral bedoel om de Vertrouwelijkheid en Integriteit van de data te beschermen. Dit kan behaald worden door alle data zelf te versleutelen, met eigen beheer van de encryptiesleutel. Dit zelf versleutelen van data wordt ook wel 'client-side' encryptie genoemd. Daarnaast kan de data opgedeeld worden in meerdere blokken (fragmentatie) en verdeeld worden over verschillende CSP's (Cloud Service Providers). Door dit concept toe te passen krijgt elke provider een stukje van een bestand, wat tevens versleuteld is. De CSP kan dan nooit het hele bestand zien, begrijpen, bewerken of verwijderen.

Dit concept wordt dus gebruikt als beginpunt van deze technische realisatie. Dit concept is conceptueel, er wordt niet gesproken over tools, standaarden of programmeertalen die gebruikt moeten worden. Deze zullen tijdens het ontwerp zelf gevonden moeten worden.

### 2.2 Visueel ontwerp

Het ontwerp dat uiteindelijk voor dit POC is gemaakt is hieronder hier onder te zien. Het ontwerp zal hier kort toegelicht worden, de individuele onderdelen zullen daarna verder toegelicht worden. Het ontwerp is bedoeld om in Python gerealiseerd te worden. Het begint allemaal bij het invoeren van een bestand, deze wordt eerst ge-encrypt met PyNaCl waarbij de sleutel lokaal wordt opgeslagen. Na de encryptie wordt het ge-encrypte bestand opgedeeld in 3 blokken. Deze blokken worden dan d.m.v. Boto3 naar een S3-bucket in Backblaze gestuurd. Wanneer een bestand wordt opgevraagd worden alle drie blokken gedownload vanuit Backblaze. De drie blokken worden weer gedefragmenteerd tot een ge-encrypt bestand. Het bestand wordt daarna gedecript met de bijbehorende sleutel tot leesbare tekst.



Figuur 1, ontwerp van het Proof-of-Concept van de technische realisatie

## 2.3 S3 met Boto3 en CSP

Voor het communiceren met de CSP is er gekozen voor het S3 protocol van Amazon met bijbehorende Boto3 SDK. Amazon S3 staat voor ‘Simple Storage Service’, S3 is voornamelijk bedoel voor het opslaan van data. Het verschilt hierin met bijvoorbeeld Amazon AWS, dit is voornamelijk bedoeld voor Cloud Computing.<sup>1</sup> S3 is een universeel protocol en kan dus ook door andere Cloud providers gebruikt worden.

Amazon S3 heeft een SDK (software development kit) in Python genaamd Boto3. Boto3 is bedoeld om S3-services te beheren via Python, dus zonder gebruik te maken van de web-interface. Doordat S3 een universeel protocol is, is Boto3 dit ook. Dit zijn de redenen dat er voor S3 is gekozen, het is universeel en is beschikbaar in Python.

Na het kiezen van het S3 protocol moest er een keuze gemaakt worden in CSP. De CSP moest aan twee eisen voldoen, ten eerste de werking met het S3 protocol. Ten tweede moest het gratis zijn. Aangezien Amazon AWS niet gratis is, valt deze af. De Cloud provider Backblaze voldoet daarentegen wel aan beide criteria, het is gratis (tot bepaalde hoogte) en kan gebruik maken van S3 en Boto3<sup>2</sup>. Backblaze is dus de CSP die gebruikt gaat worden.

## 2.4 Encryptie

Encryptie is een belangrijk beveiligingsmaatregel van het ontwerp. Om precies te zijn gaat het hier om client-side encryptie waarbij de sleutel in eigen beheer is. Er moet daarom een Python module komen die bestanden en/of tekst kan encrypten met een eigen sleutel die opgeslagen kan worden.

PyNaCl is de Python module die gekozen is. PyNaCl is een open-source module die data op verschillende manieren kan versleutelen. PyNaCl is een ‘wrapper’ van Libsodium, wat op zijn beurt een ‘port’ is van NaCl (Salt). PyNaCl biedt verschillende vormen van encryptie, waaronder digital signatures, secret-key encryptie en public-key encryptie.<sup>3</sup> Voor dit POC is er gekozen voor secret-key encryptie. Dit is encryptie waarbij er alleen gebruik wordt gemaakt van een sleutel, de key. Dit is een vorm van symmetrische cryptografie. Voor encryptie en de decryptie van de data wordt dezelfde key gebruikt.<sup>4</sup> PyNaCl maakt gebruik van het XSalsa20 stream cipher algoritme.<sup>5</sup>

Er is voor PyNaCl gekozen omdat het een relatief simpele maar toch effectieve vorm van encryptie is. PyNaCl maakt het gebruik van Libsodium en dus NaCl op een eenvoudige manier mogelijk.

---

<sup>1</sup> <https://aws.amazon.com/s3/>

<sup>2</sup> <https://www.backblaze.com/docs/cloud-storage-s3-compatible-sdks>

<sup>3</sup> <https://pynacl.readthedocs.io/en/latest/>

<sup>4</sup> [https://nl.wikipedia.org/wiki/Symmetrische\\_cryptografie](https://nl.wikipedia.org/wiki/Symmetrische_cryptografie)

<sup>5</sup> [https://libsodium.gitbook.io/doc/advanced/stream\\_ciphers/xsalsa20](https://libsodium.gitbook.io/doc/advanced/stream_ciphers/xsalsa20)

## 2.5 Fragmentatie

Het laatste onderdeel van het ontwerp is de fragmentatie. Datafragmentatie is het verdelen van data over meerdere locaties. Datafragmentatie kan ook het opsplitsen van data in verschillende blokken zijn. In de context van het POC is fragmentatie het opdelen van een bestand in twee of meer blokken. Eerst wordt de inhoud van het bestand ge-encrypt, daarna wordt het opgesplitst in blokken. Deze blokken komen dan weer in buckets van Backblaze. Voor dit POC is er gekozen voor een fragmentatie in 3 blokken. Het tekst bestand dat wordt ingevoerd wordt eerst ge-encrypt, daarna in drie gelijke blokken geknipt. Deze blokken komen daarna in een bucket van Backblaze. In het concept van TNO worden meerdere CSP's gebruikt, tijdens dit POC is dit niet zo. De drie S3-buckets staan hier symbool voor de CSP's die het S3 protocol ondersteunen. Tijdens de decryptie zullen de blokken weer samengevoegd moeten worden, de defragmentatie. Dit gedefragmenteerd bestand kan dan, mits de sleutel juist is opgeslagen, weer gedecrypt worden tot leesbare tekst. Voor de fragmentatie zal geen Python module gebruikt worden.

### 3. Resultaten Python

Na het ontwerp van hoofdstuk 2 in de praktijk gebracht te hebben is er een Python bestand ontstaan die bestanden kan encrypten, fragmenteren en uploaden naar de Cloud. Daarnaast kan het gefragmenteerde bestand weer gedownload, gedefragmenteerd en gedecrypt worden. De tool is bedoeld om tekst bestanden te verwerken, dus alleen .txt bestanden. Voor een POC is dit voldoende om het concept aan te tonen. Het is dus gelukt om het gehele ontwerp te realiseren. Het Python bestand is te zien in de bijlage. In dit hoofdstuk zal de werking van de gemaakte tool toegelicht worden. De code zal in het volgende hoofdstuk toegelicht worden. Naast deze tekstuele beschrijving is er ook een video beschikbaar die de werking van de tool toelicht.

#### 3.1 Werking van de tool

Het begint allemaal met het aanmaken van een tekst bestand die geupload moet worden naar de Cloud. Bij deze tekstuele demonstratie heet dit bestand 'data.txt', hierin staan fictieve persoonsgegevens. Na het aanmaken van het bestand kan de tool gestart worden, hier wordt eerst gevraagd of een bestand geupload of gedownload moet worden. Er wordt gekozen voor uploaden, zie screenshot hieronder.

```
PS C:\Users\David\Documents\text_files_DUO> & C:/Users/David/anaconda3/python.exe
Enter 1 to upload or 2 to decrypt: 1
Enter filename to upload: data.txt
encryption of file complete
fragmentation of file complete
upload of first_fragment_data.txt is a success
upload of second_fragment_data.txt is a success
upload of third_fragment_data.txt is a success
```

Figuur 2, Het uitvoeren van het Python bestand van het POC. Het bestand 'data.txt' wordt hier geupload.

In deze screenshot is ook te zien dat de encryptie van het bestand succesvol is, dit betekent ook dat er gebruik gemaakt is van een 'key'. Deze key wordt na gebruik opgeslagen in dezelfde directory als het tekstbestand. Deze key zal de naam 'key\_encrypted\_data.txt' krijgen, zie screenshot hieronder. Elk bestand dat geupload wordt krijgt dus ook zijn eigen key.

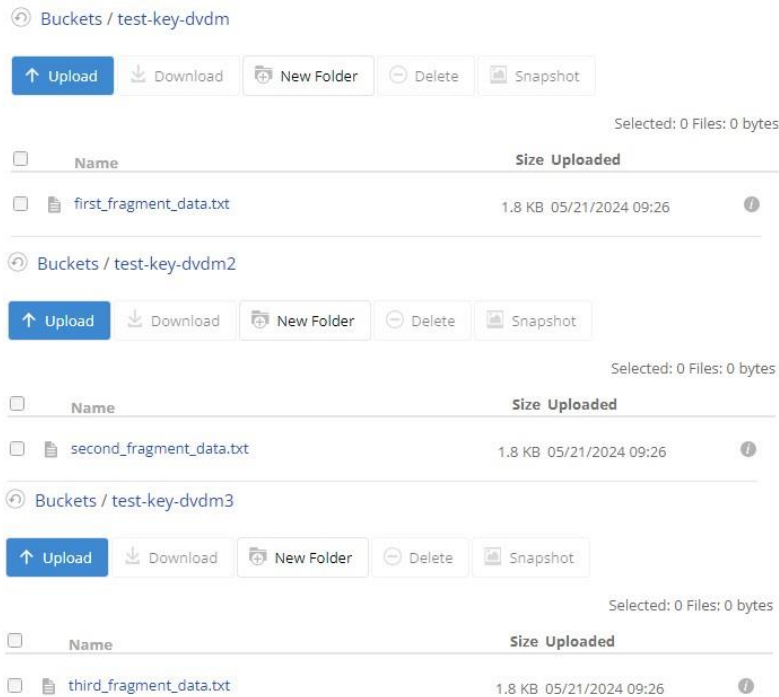
This PC > Documents > text\_files\_DUO

Name	Date	Type	Size	Tags
data.txt	5/6/2024 4:12 PM	Text Document	6 KB	
<input checked="" type="checkbox"/> key_encrypted_data.txt	5/21/2024 9:26 AM	Text Document	1 KB	
movie.txt	5/6/2024 3:09 PM	Text Document	61 KB	
willy.txt	5/7/2024 11:24 AM	Text Document	4 KB	

Figuur 3, De encryptiesleutel van het 'data.txt' bestand is aangemaakt en te zien

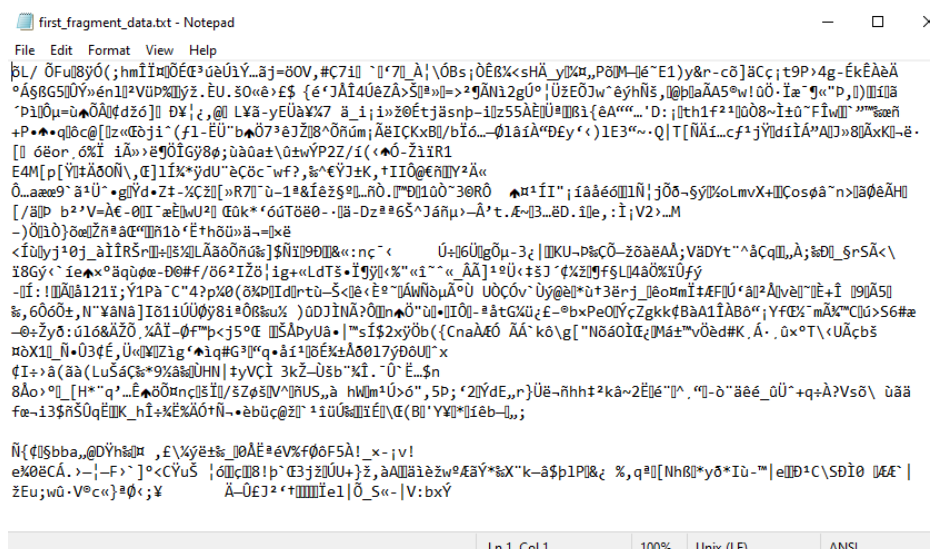
Daarna is te lezen dat uploaden van de drie blokken van het bestand gelukt is. Dit betekend dus dat het bestand ook al gefragmenteerd is, hiervoor worden geen lokale bestanden opgeslagen. Deze blokken worden direct geupload naar Backblaze. De drie buckets die gebruik worden zijn: 'test-key-dvdm', 'test-key-dvdm2' en 'test-key-dvdm3'. De bestanden van de blokken moeten nu dus in de juiste bucket in Backblaze te vinden zijn. Hieronder zijn de buckets in Backblaze te zien, hier staan de juiste bestanden in de juiste bucket.





Figuur 4, Alle drie fragmenten van 'data.txt' zijn te zien in drie verschillende buckets.

Hierboven is te zien dat alle drie de blok bestanden in de juiste bucket staan. Het format van deze blok bestanden is altijd 'first\_fragment\_data.txt'. Aan de tijd en datum van de drie bestanden en het key-bestand is te zien dat ze op hetzelfde moment zijn aangemaakt, namelijk 21 mei om 09:26. Het bestand staat nu dus ge-encrypt en gefragmenteerd in de Cloud. Wanneer dit één van de drie bestanden via de web-interface gedownload wordt, is dit de inhoud:



Figuur 5, afbeelding van het bestand dat Backblaze te zien krijgt na gebruik van het POC

Zoals hierboven te zien is het tekst bestand niet meer terug te lezen. Backblaze geeft ook aan dat het een binary/octet stream is, zie hieronder.

```

Name: first_fragment_data.txt
Bucket Name: test-key-dvdm
Bucket Type: Private
Friendly URL: https://f005.backblazeb2.com/file/test-key-dvdm/first_fragment_data.txt
S3 URL: https://test-key-dvdm.s3.us-east-005.backblazeb2.com/first_fragment_data.txt
Native URL: https://f005.backblazeb2.com/b2api/v1/b2_download_file_by_id?fileid=4_z2038b8bc2829209a8df50f18_f11034bdd0caa810d_d20240521_m072635_c005_v0501019_t0009_u01716276395000
Kind: binary/octet-stream
Size: 1.8 KB
Uploaded: 05/21/2024 09:26
Fguid: 4_z2038b8bc2829209a8df50f18_f11034bdd0caa810d_d20240521_m072635_c005_v0501019_t0009_u01716276395000
Sha1: 389a508a1593e505b1af841b1320304c6f06647d
File Info:
Encryption: none

```

Figuur 6, Nadere informatie van het 1e fragment in Backblaze, hier is te zien dat het bestand een binair/octet-stream is

Nu het uploaden klaar is kan het weer gedownload en gedecript worden. Dit wordt gedaan door het bestand opnieuw uit te voeren en voor optie 2, decryptie te kiezen. Het decryptie proces begint bij het downloaden van de drie blokken uit de drie buckets. Daarna worden de blokken samengevoegd, de defragmentatie. Tijdens de decryptie wordt gebruik gemaakt van de key van data.txt. Na de decryptie is de inhoud te lezen in de terminal. Bij de decryptie worden verder geen extra bestanden aangemaakt, het ge-encrypte bestand komt in de direct terminal terecht.

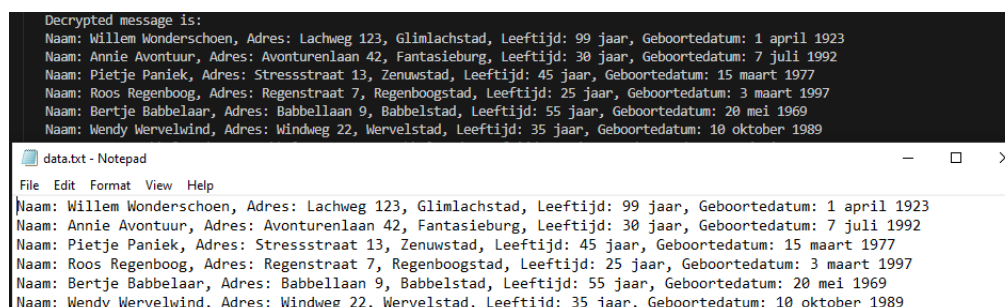
```

Enter filename to download: data.txt
first_fragment_data.txt has been downloaded from bucket test-key-dvdm
second_fragment_data.txt has been downloaded from bucket test-key-dvdm3
third_fragment_data.txt has been downloaded from bucket test-key-dvdm3
Decrypted message is:
Naam: Willem Wonderschoen, Adres: Lachweg 123, Glimlachstad, Leeftijd: 99 jaar, Geboortedatum: 1 april 1923
Naam: Annie Avontuur, Adres: Avonturenlaan 42, Fantasieburg, Leeftijd: 30 jaar, Geboortedatum: 7 juli 1992
Naam: Pietje Paniek, Adres: Stressestraat 13, Zenuwstad, Leeftijd: 45 jaar, Geboortedatum: 15 maart 1977
Naam: Roos Regenboog, Adres: Regenstraat 7, Regenboogstad, Leeftijd: 25 jaar, Geboortedatum: 3 maart 1997
Naam: Bertje Babbelaar, Adres: Babbellaan 9, Babbelstad, Leeftijd: 55 jaar, Geboortedatum: 20 mei 1969
Naam: Wendy Wervelwind, Adres: Windweg 22, Wervelstad, Leeftijd: 35 jaar, Geboortedatum: 10 oktober 1989
Naam: Kees Kwebbel, Adres: Kwebbelstraat 15, Kwebbelstad, Leeftijd: 40 jaar, Geboortedatum: 5 juni 1982
Naam: Lisa Lach, Adres: Lachlaan 8, Lachstad, Leeftijd: 28 jaar, Geboortedatum: 12 december 1996
Naam: Frank Flater, Adres: Flaterplein 3, Flaterstad, Leeftijd: 32 jaar, Geboortedatum: 25 september 1990
Naam: Emma Euforie, Adres: Euforielaan 11, Euforiastad, Leeftijd: 22 jaar, Geboortedatum: 18 augustus 2000
Naam: Tommie Toverbal, Adres: Toverlaan 6, Toverstad, Leeftijd: 38 jaar, Geboortedatum: 8 juli 1984
Naam: Nina Nieuwsgierig, Adres: Nieuwstraat 4, Nieuwsgierigstad, Leeftijd: 29 jaar, Geboortedatum: 16 april 1993

```

Figuur 7, de output van het decrypten van 'data.txt'

Het originele bestand, de data.txt kan wel vergeleken worden met het gedecripte output. Hieronder staan de eerste zes regels van het tekst bestand en de eerste zes regels van de output, zoals te lezen is de inhoud identiek. De inhoud van het bestand is dus niet veranderd.



```

Decrypted message is:
Naam: Willem Wonderschoen, Adres: Lachweg 123, Glimlachstad, Leeftijd: 99 jaar, Geboortedatum: 1 april 1923
Naam: Annie Avontuur, Adres: Avonturenlaan 42, Fantasieburg, Leeftijd: 30 jaar, Geboortedatum: 7 juli 1992
Naam: Pietje Paniek, Adres: Stressestraat 13, Zenuwstad, Leeftijd: 45 jaar, Geboortedatum: 15 maart 1977
Naam: Roos Regenboog, Adres: Regenstraat 7, Regenboogstad, Leeftijd: 25 jaar, Geboortedatum: 3 maart 1997
Naam: Bertje Babbelaar, Adres: Babbellaan 9, Babbelstad, Leeftijd: 55 jaar, Geboortedatum: 20 mei 1969
Naam: Wendy Wervelwind, Adres: Windweg 22, Wervelstad, Leeftijd: 35 jaar, Geboortedatum: 10 oktober 1989

```

```

data.txt - Notepad
File Edit Format View Help
Naam: Willem Wonderschoen, Adres: Lachweg 123, Glimlachstad, Leeftijd: 99 jaar, Geboortedatum: 1 april 1923
Naam: Annie Avontuur, Adres: Avonturenlaan 42, Fantasieburg, Leeftijd: 30 jaar, Geboortedatum: 7 juli 1992
Naam: Pietje Paniek, Adres: Stressestraat 13, Zenuwstad, Leeftijd: 45 jaar, Geboortedatum: 15 maart 1977
Naam: Roos Regenboog, Adres: Regenstraat 7, Regenboogstad, Leeftijd: 25 jaar, Geboortedatum: 3 maart 1997
Naam: Bertje Babbelaar, Adres: Babbellaan 9, Babbelstad, Leeftijd: 55 jaar, Geboortedatum: 20 mei 1969
Naam: Wendy Wervelwind, Adres: Windweg 22, Wervelstad, Leeftijd: 35 jaar, Geboortedatum: 10 oktober 1989

```

Figuur 8, Vergelijking van de output van de decryptie en het originele 'data.txt' bestand

Deze tekstuele demonstratie geeft dus aan het ontwerp daadwerkelijk in Python is gerealiseerd. Het POC kan bestanden encrypten, fragmenteren, verdelen over meerdere buckets in de Cloud, downloaden, defragmenteren en weer decrypten tot leesbare teksts.

## 4. Beschrijving code

In dit hoofdstuk wordt de Python code van de gemaakte tool toegelicht. Het toelichten zal gedaan worden per functie en/of cluster van code, in totaal zijn er 7 stukjes code dat toegelicht zal worden. Steeds zal onder het stukje code de toelichting komen. De volledige code staat in de bijlage en in het portfolio onder POC\_portfolio.py. Let op, in deze code staan geen credentials van Backblaze en ook geen namen van buckets in Backblaze.

```
s3_client = boto3.client('s3',
    aws_access_key_id="0123456789",
    aws_secret_access_key="abcdefghijklmnopqrstuvwxyz",
    endpoint_url="https://endpoint.com")
```

Hier wordt verbinding gemaakt met Backblaze via Boto3. De access en secret key zijn hier vervangen door niet werkende keys. De access en secret keys kunnen vergeleken worden met public en private keys.

```
def encrypt_and_fragment(input_file):

    bucket1 = 'name_of_the_first_bucket'
    bucket2 = 'name_of_the_second_bucket'
    bucket3 = 'name_of_the_third_bucket'
    key = nacl.utils.random(nacl.secret.SecretBox.KEY_SIZE)
    box = nacl.secret.SecretBox(key)

    key_file = open('key_encrypted_' + input_file, "wb")
    key_file.write(key)

    file = open(input_file, "rb")
    binary_data = file.read()

    encrypted_data = box.encrypt(binary_data)
    print("encryption of file complete")
    block_length = len(encrypted_data) // 3
    block1 = encrypted_data[:block_length]
    block2 = encrypted_data[block_length:2 * block_length]
    block3 = encrypted_data[2 * block_length:]
    print("fragmentation of file complete")

    upload_file(block1, bucket1, 'first_fragment_' + input_file)
    upload_file(block2, bucket2, 'second_fragment_' + input_file)
    upload_file(block3, bucket3, 'third_fragment_' + input_file)
```

Dit is de functie 'encrypt\_and\_fragment', hier wordt het bestand ge-encrypt en gefragmenteerd. Eerst wordt er key aangemaakt, deze wordt in de 'key' variabele gezet. Daarna wordt er een box aangemaakt met deze key. Nadat de box is aangemaakt wordt de key in een bestand gezet met de naam 'key\_encrypted\_file.txt'. Dit moet in binair, daarom de "wb" (write binary). De input\_file wordt daarna geopend in binair, de inhoud van dit bestand komt in de variabele 'binary\_data'. Nu de key, box en inhoud bekend is, wordt de inhoud van het bestand ge-encrypt met de 'box.encrypt' functie. De encryptie wordt dus gedaan met de gemaakte box, die gekoppeld is aan de key. Nu de encryptie klaar is, komt de fragmentatie. Eerst wordt de lengte het ge-encrypte bestand bepaald, daarna komen in blokken 1 t/m 3 steeds een derde deel van het bestand. Het bestand is nu gefragmenteerd in 3 blokken met dezelfde lengte. Deze blokken kunnen nu geupload worden naar Backblaze, dit wordt gedaan door de 'upload\_file' functie.

```
def upload_file(data, bucket_name, file_name):
    s3_client.put_object(Bucket=bucket_name, Key=file_name, Body=data)
    print("upload of " + file_name + " is a success")
```

De upload\_file functie uploadt steeds een bestand naar de bucket die is meegegeven. Dit wordt gedaan door de Boto3 functie put\_object. Put\_object heeft 3 variabelen nodig, de bucket, de

bestandsnaam en de inhoud van het bestand. De bestandsnaam is geen bestaand bestand, de bestandsnaam wordt de naam die in Backblaze komt te staan.

```
def download_file(bucket, object_name, download_file_name):
    if object_name is None:
        object_name = os.path.basename(download_file_name)
    s3_client.download_file(bucket, object_name, download_file_name)
    print(download_file_name + " has been downloaded from bucket " + bucket)
```

De ‘download\_file’ functie kan, zoals de naam al zegt, bestanden downloaden. Dit kan gedaan worden door het invoeren van de bucketnaam, objectnaam en de bestandsnaam. De ‘object\_name’ is altijd gelijk aan de bestandsnaam, daarom kan er ook ‘None’ meegegeven worden. De ‘object\_name’ wordt dan gelijk aan de ‘download\_file\_name’. Het downloaden zelf wordt gedaan door de Boto3 functie ‘download\_file’. Hier moet de bucketnaam en de bestandsnaam meegegeven worden.

```
def download_decrypt_combine(output_file):
    download_file('test-key-dvdm', None, 'first_fragment_' + output_file)
    download_file('test-key-dvdm2', None, 'second_fragment_' + output_file)
    download_file('test-key-dvdm3', None, 'third_fragment_' + output_file)
    combine_and_decrypt(output_file)
```

De functie ‘download\_decrypt\_combine’ is alleen bedoeld om andere functies aan te roepen. Eerst worden alle drie fragmentbestanden gedownload door de ‘download\_file’ functie. Daarna worden deze bestanden aan elkaar geplakt en gedecrypt door de ‘combine\_and\_decrypt’ functie.

```
def combine_and_decrypt(output_file):
    with open('first_fragment_' + output_file, "rb") as file1:
        block1 = file1.read()
    with open('second_fragment_' + output_file, "rb") as file2:
        block2 = file2.read()
    with open('third_fragment_' + output_file, "rb") as file3:
        block3 = file3.read()

    encrypted_data = block1 + block2 + block3

    key_file = open('key_encrypted_' + output_file, "rb")
    key = key_file.read()

    box = nacl.secret.SecretBox(key)
    plaintext = box.decrypt(encrypted_data)
    print("Decrypted message is:")
    print(plaintext.decode('utf-8'))

    os.remove('first_fragment_' + output_file)
    os.remove('second_fragment_' + output_file)
    os.remove('third_fragment_' + output_file)
```

Bij de laatste functie worden de drie fragment bestanden weer aan elkaar geplakt en wordt de inhoud hiervan gedecrypt. Eerst worden alle drie fragmenten geopend en in een ‘block’ variabele gezet. Daarna worden alle drie ‘block’ variabelen achter elkaar gezet en in ‘encrypted\_data’ variabele gezet. Deze variabele is nu het gedefragmenteerde, maar nog wel ge-encrypte bestand. Vervolgens wordt de key bestand (dat tijdens de eerste functie is gemaakt) weer geopend. De inhoud komt weer in de ‘key’ variabele. Ook wordt de box herhaald, met de key, dit is nu dezelfde box als bij de eerste functie. Nu kan de inhoud van encrypted\_data gedecrypt worden, dit wordt gedaan door box.decrypt. De inhoud hiervan komt eerst in de ‘plaintext’ variabele. Omdat PyNaCl alleen in binair werkt, moeten de gedecrypte inhoud nog in leesbare tekens komen. Dit wordt gedaan door de plaintext.decode(“utf-8”). Dit zet de binair inhoud om in leesbare tekst. Deze leesbare tekst wordt hierna weergegeven in de terminal. De drie fragmenten die gedownload zijn worden verwijderd.

```
choice = input('Enter 1 to upload or 2 to decrypt: ')
if choice == "1":
    input_file = input("Enter filename to upload: ")
    encrypt_and_fragment(input_file)
elif choice == "2":
    output_file = input("Enter filename to download: ")
    download_decrypt_combine(output_file)
else:
    print("error")
```

Dit is de laatste code van de tool. Bij dit stukje code wordt de vraag aan de gebruiker gesteld wat deze willen, uploaden of decrypten. Wanneer er gekozen is voor uploaden wordt er gevraagd naar het inputbestand. Dit bestand wordt daarna meegegeven aan de 'encrypt\_and\_fragment' functie. Bij de keuze voor decryptie wordt er ook gevraagd naar een bestand, dit keer het bestand dat gedownload en gedecrypt moet worden. Dit bestand wordt meegegeven aan de 'download\_decrypt\_combine' functie.

## 5. Encryptie tool

Om een compleet beeld te krijgen van de opties die DUO heeft met betrekking tot client-side encryptie is er gekozen om ook te kijken naar een bestaande tool. Er is gezocht naar een tool die lokaal data kan encrypten en kan doorsturen naar een Cloud. Het liefst kan deze tool ook data fragmenteren en verdelen over meerdere CSP's. Op deze manier is er zowel een eigen POC van een tool en een bestaande tool die data veilig in de Cloud kan zetten. In dit hoofdstuk zal toegelicht worden welke tool er gekozen is en waarom. Daarnaast zal de werking van de tool besproken worden

### 5.1 Cryptomator

Voor de encryptie tool is er gekozen voor Cryptomator. Cryptomator is een Duitse open-source encryptie tool die client-side encryptie kan toepassen op een Public Cloud. Met de tool blijft de encryptie sleutel zelf in beheer. Naast Cryptomator dient er een CSP gekozen te worden, de keuze is gevallen op Google Drive. Hiervoor is gekozen omdat dit de enige beschikbare CSP is die alleen gebruik maakt van poort 443 (HTTPS<sup>6</sup>). De andere CSP maken allemaal ook gebruik van poort 80 (HTTP), het gebruik hiervan is niet toegestaan binnen DUO. Daarom is de keuze op Google Drive gevallen.

Cryptomator maakt gebruik van een 'vault', deze vault is een virtuele drive die vervolgens in Google Drive gezet moet worden. Deze vault is via de Cryptomator applicatie te benaderen. Voor het maken van deze virtuele drive maakt Cryptomator gebruik van WinFSP (Windows 'File System Driver').<sup>7</sup>

Elke keer als een bestand in deze drive wordt gezet wordt deze automatisch ge-encrypt. Voor de gebruiker gaat de encryptie en decryptie automatisch en is dus niet te zien. Wanneer de bestanden in de Cloud interface bekeken worden, kan er gezien worden dat er daadwerkelijk encryptie plaatsvindt.<sup>8</sup> Het type encryptie dat gebruikt wordt is AES-256.<sup>9</sup>

---

<sup>6</sup> [Drive and Sites firewall and proxy settings - Google Workspace Admin Help](#)

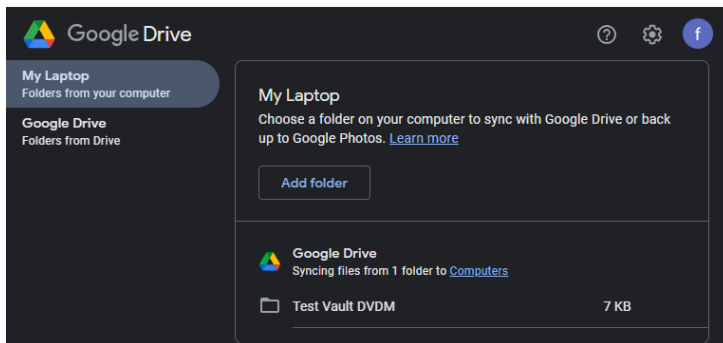
<sup>7</sup> <https://github.com/winfsp/winfsp>

<sup>8</sup> <https://cryptomator.org/>

<sup>9</sup> <https://en.wikipedia.org/wiki/Cryptomator>

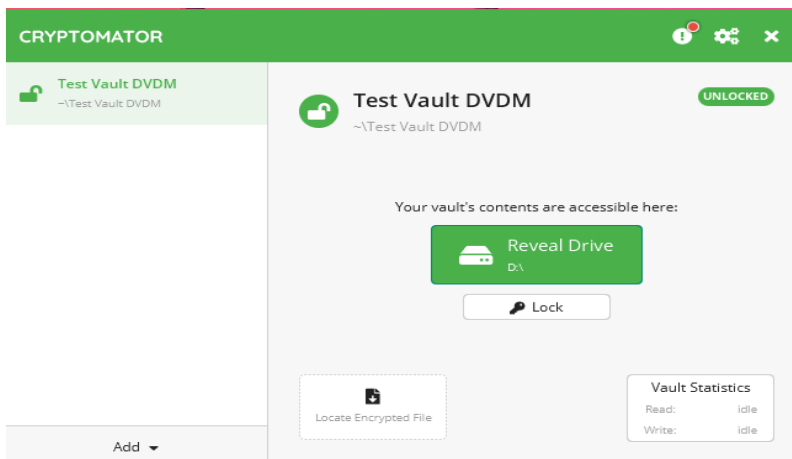
## 5.2 Werking Cryptomator

Cryptomator werkt dus met een vault, Cryptomator maakt hiervoor een virtuele drive aan. In deze virtuele drive komen de bestanden die ge-encrypt moeten worden. Cryptomator maakt ook nog een lokale folder aan, deze folder moet in Google Drive gezet worden. Dit kan gedaan worden door de desktop applicatie van Google Drive te installeren en de folder aan Google Drive te koppelen. Hieronder is te zien dat vault 'Test Vault DVDM' aan Google Drive is toegevoegd.



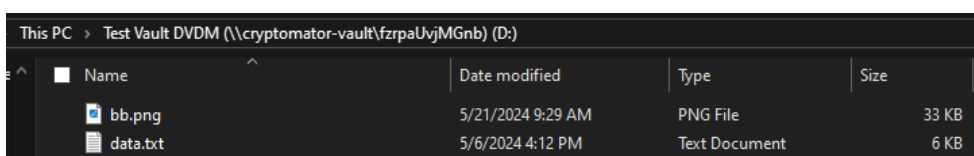
Figuur 9, hoe toevoegen van de vault folder aan Google Drive via de desktop applicatie

Bij de set-up van Cryptomator wordt er eerst een vault aangemaakt, hiervoor is een wachtwoord nodig. Dit wachtwoord is nodig om toegang te krijgen tot de virtuele drive waar de gedecrypte bestanden in komen. Hieronder is te zien hoe zo'n vault eruit ziet. De gebruiker kan de drive bekijken wanneer het wachtwoord is ingegeven. Ook kan de gebruiker de drive weer versleutelen, dan is de toegang tot de drive geweigerd. De drive verdwijnt dan uit ook de File Explorer van Windows, wanneer het wachtwoord ingegeven wordt is de drive weer toegankelijk.



Figuur 10, de vault in de Cryptomator applicatie, er kan gekozen worden om de bestanden te bekijken of om de vault te versleutelen

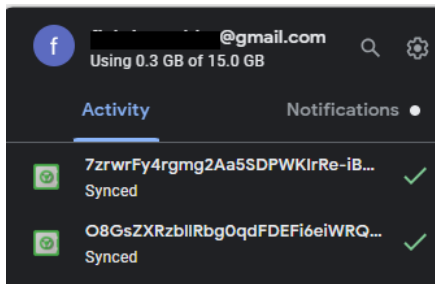
Nu de werking van Cryptomator duidelijk is moet er gekeken worden of de encryptie gelukt is. Om dit te onderzoeken zijn er twee bestanden in de virtuele drive gezet, dit zijn bb.png en data.txt. Hieronder is te zien dat de beide bestanden in de drive gezet zijn.



Figuur 11, de twee bestanden die in de vault gezet zijn



Nu is er bij de desktop app van Google Drive gekeken of er bestanden geupload zijn, dit was zo. Hieronder staan de twee bestanden die naar Google Drive gestuurd zijn. Hier is te zien dat de bestandsnamen ook worden ge-encrypt. Ze hebben ook de file extension .c9r gekregen.



Figuur 12, hoe de Google Drive applicatie de twee toegevoegde bestanden ziet

In de web-interface van Google Drive was het volgende te zien. Twee bestanden met een .c9r extensie. Dit is de file extensie van Cryptomator zelf.<sup>10</sup>

Naam	Reden voorgesteld	Eigenaar	Locatie
O8GsZXRzblIRBgOqdFDEfi6eiWRQnw==.c9r	Geopend door jou • 11:51	ik	WOWX24RU...
7zrwrFy4rgmg2Aa5SDPWKlrRe-iBMNXf.c9r	Je hebt dit document geupload • 11:45	ik	WOWX24RU...

Figuur 13, hoe de web-interface van Google Drive de geüploadde bestanden ziet

In Google Drive is dit bestand niet te openen, daarom is het bestand gedownload en geopend in Notepad. Hieronder staat het resultaat van de encryptie van data.txt. Hier is te zien dat het tekst bestand niet meer te lezen is. De encryptie is dus gelukt, Google Drive kan de inhoud van het bestand niet meer lezen. Decryptie van bestanden gaat weer via de vault in de Cryptomator applicatie. Wanneer het wachtwoord ingegeven wordt zal de virtuele drive tevoorschijn komen met de gedecrypte bestanden.



Figuur 14, de inhoud van het ge-encrypte bestand 'data.txt' na het downloaden vanuit de web-interface van Google Drive

<sup>10</sup> <https://fileinfo.com/extension/c9r>



## 6. Effect op Risicoanalyse

In de vorige hoofdstukken zijn zowel het gerealiseerde POC als de bestaande encryptie tool besproken. Er kan nu gekeken worden op wat voor manier beide tools invloed hebben op de gemaakte risicoanalyse. Daarnaast moet er gekeken worden hoe het invloed heeft op het BIV-model.

Door naar afbeelding 13 en 14 te kijken is te zien dat Cryptomator de inhoud van het bestand zo encrypt dat het niet meer te begrijpen is. Ook de bestandsnaam is niet meer te lezen. Voor het Python POC geldt ook dat de inhoud van de bestanden niet meer te begrijpen is. Dit is te zien in afbeelding 5.

Backblaze en Google Drive zien alleen maar bestanden die ge-encrypt zijn. Voor de CSP, een medewerkers van de CSP, een autoriteit of wie dan ook is het dus niet mogelijk om de inhoud van het bestand te begrijpen. Alleen de gebruiker kan de inhoud bekijken. Decrypten is ook niet mogelijk, de encryptiesleutel is bij beide oplossingen immers in eigen bezit. Er kan dus gezegd worden dat de Vertrouwelijkheid van de data na gebruik van het POC en Cryptomator gegarandeerd kan worden.

De Integriteit kan ook gegarandeerd worden, wanneer een bestand aangepast wordt zal de decryptie niet meer werken. Op deze manier weet de eigenaar van de data dat de data is aangepast. Beschikbaarheid kan niet gegarandeerd worden. Deze constatering heeft ook invloed op de risicoanalyse. Alle Vertrouwelijkheid en Integriteitsrisico's kunnen hierdoor verlaagd worden. Sommige risico's kunnen zelf compleet afgeschermd worden. De Beschikbaarheidsrisico's blijven dus nog over. In bijlage 2 staat de tabel met de resultaten van de Risk Treatment. Hierin zijn alle risico's te zien en op welk aspect van het BIV-model het risico impact heeft. Daarnaast is te zien of het POC en Cryptomator dit risico kan verlagen.

## 7. Vergelijking Cryptomator en het POC

Na het maken van het Proof-of-Concept en het testen van Cryptomator kunnen de twee vergeleken worden. Er zal gekeken worden naar de overeenkomsten, verschillen en de voor- en nadelen van beide tools. Wel moet er rekening gehouden worden dat het POC nog steeds conceptueel is. Het is dus geen ‘echt’ werkende tool. Cryptomator is wel een echt werkende tool die implementeerbaar is.

### 7.1 Overeenkomsten

1. Beide tool passen client-side encryptie toe
2. Beide tool hebben de encryptie sleutel in eigen beheer
3. Beide tools sturen ge-encrypte bestanden naar de Cloud
4. Beide tools kunnen de Vertrouwelijkheid van de data garanderen
5. Beide tools kunnen de Integriteit van de data garanderen
6. Beide tools kunnen de ge-encrypte bestanden weer decrypten

### 7.2 Verschillen

1. Cryptomator past geen fragmentatie toe, het POC wel
2. Cryptomator werkt automatisch, het POC heeft een gebruiker nodig om te werken
3. Cryptomator encrypt ook de bestandsnaam, het POC niet
4. Het POC werkt alleen met S3 providers, Cryptomator werkt met meerdere protocollen en CSP's
5. Het POC verwerkt alleen meer .txt bestanden, Cryptomator werkt met alle bestandstypen
6. Cryptomator werkt met een vault met een wachtwoord, het POC werkt alleen met een sleutel zonder wachtwoord
7. Cryptomator gebruikt AES-256, het POC gebruikt XSalsa20 stream cipher

### 7.3 Voor- en nadelen

1. Door fragmentatie wordt er een extra beveiligingsmaatregel toegepast bij het POC
2. Door gebruik van een vault is Cryptomator gebruiksvriendelijker, de encryptie en decryptie is bijna niet merkbaar
3. Het POC is nog een concept en kan aangepast worden aan de gebruiker. Cryptomator is minder aanpasbaar
4. Het POC zal door de fragmentatie lastiger zijn om te implementeren.
5. Cryptomator is al een bestaande tool en kan dus al gebruikt worden, het POC is niet niet.

## 8. Conclusie

Het doel van de Technische Realisatie was het maken van een Proof-of-Concept van een manier om data veilig in de Public Cloud op te slaan. De vragen die hierbij gesteld zijn waren:

1. **Hoe kunnen er beveiligingsmaatregelen geïmplementeerd worden in de Public Cloud.**
2. **Wat voor invloed heeft het POC en de bestaande tool op het BIV-model?**
3. **Welke risico's worden er gemitigeerd worden met het POC en de bestaande tool?**

### Conclusie 1

Na het realiseren van het POC en het testen van Cryptomator kan er geconcludeerd worden dat beveiligingsmaatregelen zowel zelf geïmplementeerd kunnen worden of door het kiezen van een bestaande tool. Er is gekeken naar een bestaande encryptie tool en het maken van een eigen Python tool. Qua werking zijn beide tools fundamenteel vergelijkbaar, beide implementeren ze client-side encryptie in de Cloud. De bestaande tool is daarentegen wel daadwerkelijk implementeerbaar. Het POC is conceptueel, wel toont het aan dat fragmentatie ook mogelijk is, boven op de encryptie.

### Conclusie 2

Na het realiseren van het POC en het testen van Cryptomator kan er gezegd worden dat de Integriteit en Vertrouwelijkheid gegarandeerd kunnen worden. Door de client-side encryptie kan een CSP de inhoud van de data nooit begrijpen. Daarnaast is het mogelijk om te achterhalen dat de data veranderd is wanneer de decryptie niet meer werkt. Het POC en Cryptomator kunnen niet de Beschikbaarheid van de data garanderen.

### Conclusie 3

Met het oog op de risicoanalyse kunnen er door middel van het POC en Cryptomator 24 risico's gemitigeerd worden. Dit zijn alle Vertrouwelijkheidsrisico's en alle Integriteitsrisico's. Dit geldt zowel voor Cryptomator als voor het gemaakte POC. Sommige risico's kunnen helemaal afgeschermd worden met de beide methodes. De 16 risico's die overblijven zijn allemaal Beschikbaarheidsrisico's

## 9. Bijlage

### 9.1 Bijlage 1: Python code van het Proof-of-Concept

```
import nacl.secret
import nacl.utils
import boto3
import os

s3_client = boto3.client('s3',
                          aws_access_key_id="0123456789",
                          aws_secret_access_key="abcdefghijklmnopqrstuvwxyz",
                          endpoint_url="https://endpoint.com")

private key
private key

#verbind met de Backblaze S3
#keyID van Backblaze, vervang door daadwerkelijke keyID
#private key van Backblaze, vervang door daadwerkelijke
#locatie van Backblaze, vervang door daadwerkelijke

def encrypt_and_fragment(input_file):
    bucket1 = 'name_of_the_first_bucket'
    bucket2 = 'name_of_the_second_bucket'
    bucket3 = 'name_of_the_third_bucket'

    key = nacl.utils.random(nacl.secret.SecretBox.KEY_SIZE)
    box = nacl.secret.SecretBox(key)

    key_file = open('key_encrypted_' + input_file, "wb")
    key_file.write(key)

    file = open(input_file, "rb")
    binary_data = file.read()

    encrypted_data = box.encrypt(binary_data)

    print("encryption of file complete")
    block_length = len(encrypted_data) // 3

    block1 = encrypted_data[:block_length]
    block2 = encrypted_data[block_length:2 * block_length]
    block3 = encrypted_data[2 * block_length:]
    print("fragmentation of file complete")

    upload_file(block1, bucket1, 'first_fragment_' + input_file)
    upload_file(block2, bucket2, 'second_fragment_' + input_file)
    upload_file(block3, bucket3, 'third_fragment_' + input_file)

def upload_file(data, bucket_name, file_name):
    s3_client.put_object(Bucket=bucket_name, Key=file_name, Body=data)
    print("upload of " + file_name + " is a success")

def download_file(bucket, object_name, download_file_name):
    if object_name is None:
    object_name = os.path.basename(download_file_name)

    s3_client.download_file(bucket, object_name, download_file_name)
    print(download_file_name + " has been downloaded from bucket " + bucket)

def download_decrypt_combine(output_file):
    bucket1 = 'name_of_the_first_bucket'
    bucket2 = 'name_of_the_second_bucket'
    bucket3 = 'name_of_the_third_bucket'

    download_file(bucket1, None, 'first_fragment_' + output_file)
    download_file(bucket2, None, 'second_fragment_' + output_file)
    download_file(bucket3, None, 'third_fragment_' + output_file)
    combine_and_decrypt(output_file)

def combine_and_decrypt(output_file):
    with open('first_fragment_' + output_file, "rb") as file1:
        block1 = file1.read()
    with open('second_fragment_' + output_file, "rb") as file2:
        block2 = file2.read()
    with open('third_fragment_' + output_file, "rb") as file3:
        block3 = file3.read()

    encrypted_data = block1 + block2 + block3

    key_file = open('key_encrypted_' + output_file, "rb")
    key = key_file.read()
```

```
box = nacl.secret.SecretBox(key)           #maak box variabele aan door de sleutel te gebruiken uit
het bestand                                #decrypt de inhoud van encrypted_data (samengevoegde
plaintext = box.decrypt(encrypted_data)
blokken) met de box
print("Decrypted message is:")
print(plaintext.decode('utf-8'))           #print de gedecrypte inhoud en zet in utf-8 (ascii)

os.remove('first_fragment_' + output_file) #verwijder het bestand voor blok 1
os.remove('second_fragment_' + output_file) #verwijder het bestand voor blok 2
os.remove('third_fragment_' + output_file)  #verwijder het bestand voor blok 3

choice = input('Enter 1 to upload or 2 to decrypt: ') #keuze maken wat gebruiker wil doen

if choice == "1":
    input_file = input("Enter filename to upload: ")
    encrypt_and_fragment(input_file)         #roep encrypt_fragment_upload aan en geen het ingegeven
    bestand mee
elif choice == "2":
    output_file = input("Enter filename to download: ")
    download_decrypt_combine(output_file)    #roep download_decrypt aan en geef ingegevn bestand
    mee
else:
    print("error")
```

## 9.2 Bijlage 2: Resultaten Risk Treatment

ID	Omschrijving	BIV	POC
1	Verlies van controle over de data	I & V	✓
2	Datalekken	V	✓
3	CSP toegang tot data	V	✓
4	Toegangsrechten bekend bij derde partij	I	✓
5	Eigen data niet zelf encrypten	V	✓
6	Locatie van data veranderd van rechtsgebied	I & V	✓
7	CLOUD Act, FISA + PATRIOT Act, Defense Production Act	V	✓
8	Kapen en/of veranderen van dataverkeer naar Cloud	I & V	✓
9	Ongeautoriseerde toegang tot fysieke locatie	V	✓
10	Data niet in EU of in Nederland opgeslagen	V	✓
11	Verantwoording ligt bij CSP	B	-
12	Hypervisor falen of aanvallen	B	-
13	(Indirecte) (D)DOS van CSP	B	-
14	CSP heeft zelf encryptiesleutel in beheer	V	✓
15	Foutieve of uitgelekte toegangsrechten/inloggegevens	I & V	✓
16	Data in plaintext opgeslagen	V	✓
17	CSP faalt, gaat failliet of wordt overgenomen	B	-
18	CSP kan niet belofde of geen extra capaciteit geven	B	-
19	Natuurrampen	B	-
20	Storing en/of sabotage	B	-
21	Land blokkeert CSP	B	-
22	Netwerk falen	B	-
23	Malware: XSS, wrapping attack, SQL injection en Metadata spoofing	I	✓
24	Subpoena	V	✓
25	CSP stopt met contract	B	-
26	Data wordt eigendom van CSP	I & V	✓
27	Incomplete verwijdering van data	I & V	✓
28	VM replicatie, rollback of hopping	I	✓
29	CSP admin/medewerkers toegang tot data + Insider threat	I & V	✓
30	Onveilige API's	I & V	✓
31	Economic Denial of Service (EDOS)	B	-
32	Pen-test door gebruikers	B	-
33	Supply chain fouten	B	-
34	Log data met informatie van CSP	V	✓
35	Foutieve VMI of VMI templates	V & I	✓
36	Back-up van data door derde partij	V	✓
37	Vendor lock-in	B	-
38	Soevereine CSP te duur	B	-
39	'Take it or leave it' mentaliteit	B	-
40	Data outsourcing door CSP	V	✓