Labo Chapter 2: CIA (Encryption and Integrity)

Team nummer:	**Teamleden:**
1TINH2	1. Aleyna Arslan 2. Stef Swinnen 3. Rasmus Leseberg 4. Tomas Soors

Inhoudsopgave

Inhoudsopgave

Tijdsbesteding

	Aleyna	Rasmus	Stef	Tomas	TOTAAL
Deelopdracht 1	/	/	2.5 uur	/	
Deelopdracht 2	/	3 uur	/	/	
Deelopdracht 3	5 uur	/	/	/	
Deelopdracht 4	3 uur	/	/	/	
Deelopdracht 5	/	/	4.5 uur	/	
Deelopdracht 6	/	/	2.5 uur	/	
Deelopdracht 7	/	4 uur	/	/	
Deelopdracht 8	/	5 uur	/	/	
Deelopdracht 9	/	/	/	3 uur	
Deelopdracht 10	/	/	/	2 uur	
Deelopdracht 11	3,5 uur	/	/	/	
Deelopdracht 12	/	/	/	5 uur	

Deelopdracht 13	/	/	/	1 uur	
Deelopdracht 14	/	4 uur	/	/	
Deelopdracht 15	/	15 uur	/	/	
Deelopdracht 16	/	/	0.5 uur	/	
Deelopdracht 17	/	/	/	3 uur	
Deelopdracht 18	40 minuten	/	/	/	
Deelopdracht 19	30 minuten	/	/	/	
Layout/afwerking + bronvermelding + eindcontrole	3 uur	1 uur	/	/	
Gezamenlijke taken *					
Totaal	15,7 uur	32 uur	10 uur	14 uur	71.7 uur

1. Terminologie en Vakjargon

Opdracht gemaakt door: Stef

Encryption: encryptie is iets dat gebruikt wordt voor het beveiligen van data. Er wordt gebruik gemaakt van wiskundige technieken om een wachtwoord of "sleutel" te beveiligen of ontcijferen. Het encryptieproces verandert de leesbare informatie naar tekst die onleesbaar is voor mensen.

Decryption: dit is het proces waarbij de geëncrypteerde gegevens terug leesbaar gemaakt worden. Het decoderen zorgt ervoor dat de tekst leesbaar is voor mensen maar ook voor het systeem.

Cryptography: een studie van communicatiemiddelen waarbij enkel de verzender en de ontvanger de inhoud van de berichten kunnen ontcijferen. Het woord zou afgeleid zijn van het Griekse kryptos, wat verborgen betekent.

Algorithm: de essentie van een algoritme is een eindige reeks van instructies voor het bekomen van een einddoel.

Key (Cryptography): de sleutel van een encryptie is een set van gegevens die gebruikt wordt om een boodschap te versleutelen of encrypteren en te ontcijferen.

Steganography: het verbergen van bepaalde informatie in een soort van geheim bericht, dit kan met verschillende technieken gedaan worden zoals onzichtbare inkt of bepaalde letters op een andere manier schrijven om erna een tekst mee te vormen.

Security Through Obscurity: STO is een proces waarbij beveiliging in een systeem wordt geïmplementeerd met interne ontwerparchitectuur van het systeem of netwerk veilig te houden. Met STO wordt geprobeerd de systemen te beveiligen door juist de gebreken ervan opzettelijk te verbergen of onthullen.

Kerckhoff's Principle: dit is een principe waarbij al de details van een systeem die publiek gekend zijn alsnog veilig moeten zijn. Alleen de sleutel is niet openbaar bij dit principe.

PKI (Public Key Infrastructure): het uitgeven en beheren van digitale certificaten en het beheren van de versleuteling van public keys.

Brute Force: door gebruik te maken van brute rekenkracht om problemen op te lossen in een systeem, zonder dat men gebruik maakt van algoritmen.

End-to-End Encryption: het versleutelen van berichten zodat alleen de zender en de ontvanger de inhoud ervan kunnen lezen. Dit maakt het moeilijker voor andere partijen om deze berichten te kraken.

The Enigma Machine: ook wel de Schlüsselmaschine E genoemd is een machine gemaakt in WOII om berichten gecodeerd door te geven aan bondgenoten. Na een bepaalde tijd had de vijand door hoe de code werkte en werd het even gebruikt om valse berichten door te geven om de vijand op het verkeerde spoor te brengen.

Entropy (als concept gebruik bij cryptography): dit is de basis waarop alle cryptografische functies werken. Entropie is een maat voor de willekeurigheid van een functie gegenereerd door gegevens te onderzoeken. Gegevens met een volledige entropie kunnen dus gezien worden als volledig willekeurig gevonden patronen.

Confusion (als concept gebruik bij cryptography): confusion wil zeggen dat elke bit van de gecodeerde tekst afhangt van een bepaald deel van de sleutel. Hierdoor worden verbanden onduidelijk. Dit verhoogt de ambiguïteit van de cijfertekst.

Diffusion (als concept gebruik bij cryptography): dit wil zeggen dat als we een karakter van de gewone tekst veranderen, verschillende karakters van de cijfertekst ook moeten veranderen. Andersom werkt dit hetzelfde, al wordt er cijfertekst aangepast, past de klare tekst ook aan.

Session Key: een session key wordt gebruikt voor het symmetrisch versleutelen van slechts 1 communicatiesessie. Het is dus een tijdelijke sleutel die 1 keer wordt gebruikt bij het versleutelen en ontsleutelen van een boodschap.

Ephemeral Key: ook wel een kortstondige sleutel is een sleutel die wordt gegenereerd voor elke uitvoering van een sleutel vormingsproces en die voldoet aan andere vereisten van het sleuteltype.

Block Cipher: dit is een algoritme van symmetrische cryptografie. Het doel hiervan is om een blok leesbare tekst te veranderen naar cijfertekst. In dit proces zal het algoritme gebruik maken van een geheime sleutel voor zowel het versleutelen als het ontsleutelen.

Cipher: een cipher is een algoritme dat gebruikt wordt bij het uitvoeren van het versleutelen en ontsleutelen van een boodschap. Het is een reeks van uitgewerkte, gedefinieerde stappen die kunnen worden opgevolgd.

Ciphertext: geheimschrift dat wordt bekomen door een cipher toe te passen op leesbare tekst.

Keyspace: een sleutelruimte is de verzameling van alle geldige, mogelijke, verschillende sleutels van een bepaald cryptosysteem. Over het algemeen is er één sleutelruimte per applicatie.

Hash: in het Nederlands 'mengelmoes'. Een reeks getallen en/of letters van gelijk welke lengte wordt omgevormd naar een string met een vaste lengte. Een hash is een soort digitale vingerafdruk, het identificeert iedere tekstreeks of bestand via een reeks getallen.

Monoalphabetic Ciphers: elke letter van een tekst wordt toegewezen aan een andere letter uit het alfabet op basis van één enkel cijfer. Het Caesar-shift-cijfer is een voorbeeld waarbij elke letter wordt verschoven op basis van een numerieke sleutel. Bij het atbash-cijfer, wordt elke letter toegewezen aan de letter die symmetrisch is ten opzichte van het midden van het alfabet.

Plaintext: tekst die uitsluitend bestaat uit letters, cijfers, spaties en leestekens. Platte tekst is niet opgemaakt zoals bijvoorbeeld cursief of vetgedrukte tekst.

Polyalphabetic Ciphers: bij een polyafabetisch cijfer wordt gebruik gemaakt van meerdere vervangingsalfabetten. Het Vigenère-cijfer is het bekendste voorbeeld.

Stream Cipher: stroomvercijfering werkt byte voor byte om platte tekst te coderen. Stukjes leesbare tekst komen in het stroomcijfer en het cijfer manipuleert elk bit met een wiskundige formule. De resulterende tekst is volledig vervormd en de ontvanger kan deze niet lezen zonder de juiste sleutel.

Symmetric Key: hier wordt gebruik gemaakt van 1 sleutel, deze zorgt voor het versleutelen en ontsleutelen.

Asymmetric Key: hier wordt gebruik gemaakt van 2 aparte sleutels, een private en een public key. De eerste sleutel wordt gebruikt om de normale tekst te versleutelen en de andere sleutel wordt gebruikt om de versleutelde tekst te ontsleutelen.

Cryptanalysis: de studie van cijfertekst, vercijferen en cryptosystemen met als doel te begrijpen hoe de systemen werken en ze te verbeteren of verslaan.

Frequency Analysis: dit houdt bij hoe vaak bepaalde letters of tekens voorkomen in een bepaald stuk tekst.

Superseded Cryptographic Keys: het is een parameter men gebruikt samen met een cryptografisch algoritme. Dit bepaalt dat een entiteit met de kennis van de sleutel de werking ervan kan reproduceren. Een entiteit zonder de kennis van de sleutel zou dit niet kunnen.

Obfuscation (in cryptography): dit wordt gedefinieerd als de transformatie van een tekst leesbaar voor mensen naar een tekenreeks die niet leesbaar meer is.

RTO (availability): recovery time objective is de maximale draaglijke duur dat een toepassing of dienst na een ramp buiten gebruik kan zijn zonder significante schade aan het bedrijf te veroorzaken.

RPO (availability): de Recovery Point Objective beschrijft het tijdsinterval dat tijdens een verstoring kan verstrijken voordat de hoeveelheid gegevens die tijdens die periode kwijt gaat, de maximaal toelaatbare drempel of tolerantie van het bedrijfscontinuïteitsplan overschrijdt.

2. Identificeer het algoritme

Opdracht gemaakt door: Rasmus

ROTATION

Cipher: Guvf grkg vf rapelcgrq, ohg lbh qrpvcurerq vg.
Plaintext: This text is encrypted, but you deciphered it.
Key: ROT13
CodeTool: ROT Cipher - Rotation - Online Rot Decoder, Solver, Translator (dcode.fr)

BASE64

Cipher: VGhpcyBhbGdvcml0aG0gaXMgb2Z0ZW4gdXNlZCBidXQgaXMgbm
90IGEgZ29vZCBlbmNyeXB0aW9uLg==
Plaintext: This algorithm is often used but is not a good encryption.
CodeTool: Base64 Decode and Encode - Online

HEX

Cipher: 54 68 69 73 20 74 65 78 74 20 69 73 20 6e 6f 74 20 65 6e 63 72 79 70 74 65 64
Plaintext: This text is not encrypted
CodeTool: Hex to Text - Online Hex Decoder (convertstring.com)

OTP

Cipher: UEPHFBUCGJSGXXL
Plaintext: CAN NOT BE CRACKED
Key: Security Essentials
CodeTool: One-Time Pad (OTP) Decoder and Encoder | Boxentriq

VIGENERE

Cipher: Iep zbj jpps czg qsxp pczcnmexly xp AMI!
Plaintext: The key used for this encryption is PXL!
Key: PXL
CodeTool: Vigenere Cipher - Online Decoder, Encoder, Solver, Translator (dcode.fr)

SUBSTITUTION

Cipher: lpn you deliaher this tebt?
Plaintext: Can you decipher this text?
Key: A ⇔ P / B ⇔ X / C ⇔ L

XOR

Cipher: ece8ece0ece8ece
Plaintext: 123456789abcdef
Key: fedcba987654321
CodeTool: XOR Online Decrypt & Encrypt with our decrypter (md5decrypt.net)

Werking en security **ROT:** ROT (Rotation) is een eenvoudige encryptiemethode, afkomstig uit de tijden van Caesar, dat wordt gebruikt om tekst te encrypteren door elke letter te vervangen door de letter 'n' plaatsen verder in het alfabet. Absolut niet secure, een password geencrypteerd met ROT13 (of ROT*N*) zou tegenwoordig in een splitseconde gecracked kunnen worden.

Werking en security **BASE64:** Base64-codering converteert elke drie bytes aan gegevens (24 bits) tot vier base64-tekens. Base64-codering verdeelt de binaire gegevens tot zes-bit chunks en wijst ze toe aan een base64-teken volgens de BASE64 ASCII tabel.

Werking en security **OTP:** Een One-Time-Password (OTP) is een automatisch gegenereerde numerieke of alfanumerieke tekenreeks die een gebruiker authenticeert voor een enkele transactie of inlogsessie. Een OTP is veiliger dan een 'statisch' (niet 'one-time') wachtwoord.

Werking en security **HEX:** HEX wordt gebruikt in de Base16 transfer encoding, waarin elke byte van de leesbare tekst wordt opgesplitst in twee 4-bit waarden en wordt weergegeven door twee hexadecimale cijfers. Helemaal niet secure, want het is makkelijk dat te encoderen, tenzij het als een extra laag encryptie functioneert.

Werking en security **SUBSTITUTION:** Subsitutie wisselt bepaalde letters uit met andere letters. Afhankelijk van de lengte en complexiteit van de uitwisseling kan dit of heel makkelijk, of heel moeilijk zijn om zelfstandig te cracken. Voor een computer zou dit echter wel heel simpel zijn, en daardoor is het niet secure.

Werking en security **VINEGERE:** Vinegere is substitutie, met een extra laag aan complexiteit. Het Vigenère-cijfer is een methode voor het versleutelen van alfabetische tekst met behulp van een reeks verweven Caesar-cijfers, gebaseerd op de letters van een trefwoord. Het maakt gebruik van een vorm van polyalfabetische substitutie. Het is met eenvoudige tools mogelijk om dit soort cijfers te kraken.

Werking en security **XOR:** Het XOR-coderingsalgoritme is een voorbeeld van symmetrische encryptie waarbij dezelfde sleutel wordt gebruikt om een bericht zowel te encrypteren als te decrypteren. Als hetzelfde XOR-masker opnieuw toegepast wordt (met dezelfde key) op de geencrypteerde tekst, wordt de oorspronkelijke plain tekst weergegeven. Het XOR-coderingsalgoritme kan worden toegepast op alle digitale/binaire informatie, inclusief op tekst gebaseerde informatie die is gecodeerd met behulp van 8-bits ASCII-code. In dit geval kan de coderingssleutel worden uitgedrukt als een reeks tekens.

Op zichzelf kan de XOR-codering zeer robuust zijn als:

- De XOR-codering gebaseerd op een lange sleutel die zich niet herhaalt.
- Voor elke nieuwe versleuteling wordt willekeurig een nieuwe sleutel gegenereerd.
- De sleutel wordt door zowel de afzender als de ontvanger geheimgehouden.

Conclusie

Deze vormen van encryptie zijn voornamelijk simpel en ouderwets, ROT13 is de oudste encryptiemethode van hun allemaal. Deze vormen van encryptie zijn in hun werking interessant om te kennen als basis voor latere, meer complexe vormen van encryptie. Met uitzondering van OTP en XOR zijn de meeste van deze encryptiemethoden makkelijk om te kraken, in sommige gevallen zelfs zonder de hulp van een computer. XOR en OTP hebben meer te bieden op het vlak van veiligheid, en gebruiken versleutelingsmethoden die alleen met een passende key te kraken zijn.

3. Encrypties & Conversies

Opdracht gemaakt door: Aleyna

1. BASE64:

Output

```
4CcT25lIHNpbmdsZSB2dWxuZXJhYmlsaXR5IGlzIGFsbCBhbiBhdHRhY2tlciBuZWVkc+KAnSAKLVRlY
W0gSC0yLQ==
```

Vragen

1. Type van het algoritme

a. Base64 is een binair naar ASCII-coderingsschema. Het is ontworpen als een manier om binaire gegevens over te dragen over kanalen die beperkte ondersteuning hebben voor verschillende inhoudstypen. Base64 gebruikt 64 dezelfde tekens die in de meeste tekensets aanwezig zijn.
b. Base64 is een encoding algoritme.
c. Je kunt Base64 de hele dag heen en weer encoden en decoden. Er is geen geheim, geen bescherming geen encryptie.
d. Security through obscurity.

2. Hoe kan je de cipher herkennen?

a. De lengte van een Base64 encoded string is altijd het veelvoud van 4. Enkel deze karakters worden gebruikt bij het encoden van Base64: “A” tot “Z”, “a” tot “z”, “0” tot “9”, “+” en “/”. Op het einde van de string kan twee keer worden opgevuld met “=” (onthoud dat dit teken enkel en alleen toegestaan is aan het einde).
b. Er zijn andere alternatieven voor Base64-encoding, zoals Uuencoding, het enige verschil is dat Uuencoding een andere karakterset gebruikt.
c. Je kunt de quote in Base64 niet terug herkennen omdat de binaire data worden vervangen door ASCII-tekens.
d. Base64 heeft geen vaste outputsize.

3. De principiële werking

a. Base64-encoding neemt de oorspronkelijke binaire data en verdeelt deze in tokens van drie bytes, een byte bestaat uit acht bits, uiteindelijk neemt Base64 vierentwintig bits in beslag. Deze drie bytes worden omgezet in vier afdrukbare karakters uit het ASCII-tabel.
b. Base64 wordt voornamelijk gebruikt bij het opslaan van complexe data in XML. Het wordt ook gebruikt om gegevens makkelijker te verzenden via e-mail en HTML-formuliergegevens.

4. Conclusie

We kunnen hieruit concluderen dat BASE64 een encoding algoritme is, die de oorspronkelijke data verdeelt in tokens van 3 bytes, op het einde van de encoding zal je altijd “==” tegenkomen. Het algoritme zorgt ervoor dat gegevens tijdens transport intact blijven zonder wijziging. Het biedt geen beveiliging omdat het makkelijk heen en weer te coderen is.

2. ROT47:

Output

“~?6 D:?8=6 GF=?6C23:=:EJ :D 2== 2? 2EE24<6C ?665D”
\%62> w\a

Vragen

1. Type van het algoritme

a. ROT47 is een Caesar-cipher met 47 tekens, het is een verbetering op ROT13. ROT47 kan bijna alle zichtbare ASCII-tekens encrypteren terwijl ROT13 enkel letters kan encrypteren. Om dit te kunnen, gebruikt ROT47 een alfabet van 94 tekens dat een subset is van de ASCII-tekens.
b. ROT47 is net zoals ROT13 een symmetrische encryptie.
c. Ondanks dat het elk karakter 47 posities verschuift, is ROT47 net zoals ROT13 geen veilig algoritme.
d. STO is van toepassing.

2. Hoe kan je de cipher herkennen?

a. ROT47 bevat gewone letters als 6 of ‘t’, dan zijn de geëncrypteerde waarde E en e.
b. ROT47 en ROT13 zijn gelijkaardige algoritmen.
c. De quote valt lichtjes te herkennen. Niet om te zeggen dat je ziet wat er staat, maar de spaties, aantal karakters zijn terug te zien in de encryptie.
d. ROT47 heeft geen vaste outputsize.

3. De principiële werking

a. In plaats van A-Z te werken, gebruikt ROT47 een grotere reeks tekens van het ASCII-tabel. Dit gebeurt door ASCII-tekens tussen 33 en 126 te nemen en deze 47 keer te verplaatsen.
b. ROT47 wordt gebruikt om berichten te coderen op discussieplatformen en sociale netwerken.

4. Conclusie

ROT47 is een symmetrische encryptie algoritme wat een verbetering is op ROT13, het algoritme is een Caesar-cipher dat gebruik maakt van 47 tekens, vandaar de “47” in de naam. Het bestaat uit het vervangen van een teken door een ander teken uit het alfabet die 47 posities verder staat. Dit algoritme wordt niet veilig beschouwd aangezien het karakters maar 47 posities verschuift, dus het is makkelijk te kraken. Dit algoritme is een goede toepassing voor het encrypteren van berichten op discussieplatformen of socialenetwerken.

3. Hex:

Output

```
e2 80 9c 4f 6e 65 20 73 69 6e 67 6c 65 20 76 75 6c 6e 65 72 61 62 69 6c 69 74 79
20 69 73 20 61 6c 6c 20 61 6e 20 61 74 74 61 63 6b 65 72 20 6e 65 65 64 73 e2 80
                  9d 20 0a 2d 54 65 61 6d 20 48 2d 32 2d
```

Vragen

1. Type van het algoritme

a. Hex is een transfer encoding waarbij elke byte wordt omgezet naar de 2-cijferige Base16 encoding van die byte. Deze bytes worden vervolgens gecodeerd in ASCII.
b. Hex is een encoding algoritme.
c. Net zoals Base64, is Hex geen veilige encoding algoritme.
d. Security through obscurity.

2. Hoe kan je de cipher herkennen?

a. Het algoritme is te herkennen aan de twee hexadecimale cijfers die per paar worden weergegeven.
b. Hex encoding lijkt voornamelijk op Base64 omdat beide ASCII-tekens gebruiken om karakters te vervangen.
c. De quote is absoluut niet te herkennen.
d. Hex encoding heeft geen vaste outputsize.

3. De principiële werking

a. HEX wordt gebruikt in de Base16 transfer encoding, waarin elke byte van de leesbare tekst wordt opgesplitst in twee 4-bit waarden en wordt weergegeven door twee hexadecimale cijfers.
b. Hex wordt gebruikt bij computer engineering om data makkelijk om te zetten van en naar binair.

4. Conclusie

Hex is een encoding algoritme dat gebruikt wordt voor het omzetten van gegevens naar binaire waarden en omgekeerd. Dit algoritme is te herkennen in Base64 omdat beide algoritmen gebruik maken van het ASCII-tabel. Het werkt door elke byte van een leesbare tekst op te splitsen in twee 4-bit waarden en deze worden vervolgens weergegeven in twee hexadecimale cijfers.

4. ROT13:

Output

“Bar fvatyr ihyarenovyvgl vf nyy na nggnpxre arrqf”
-Grnz U-2-

Vragen

1. Type van het algoritme

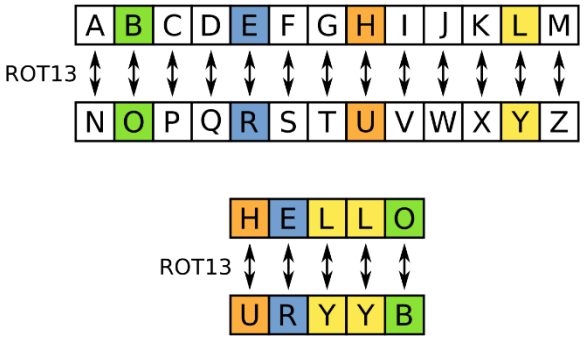
a. ROT13 is een substitutie cipher met een specifieke key waar de letters van het alfabet 13 plaatsen verschoven worden.
b. Het ROT13-algoritme is een symmetrische encryptie methode.
c. ROT13 is absoluut geen veilige cipher en kan makkelijk gekraakt worden. ROT13 is tegelijkertijd ook een Caesar-cipher, met een sleutel van 13. Dus het kraken van de Caesar-cipher werkt in dit geval ook.
d. Security through obscurity.

2. Hoe kan je de cipher herkennen?

a. Het ROT13-algoritme lijkt behoorlijk veel op ROT5. In sommige gevallen kunnen ROT5 en ROT13 in hetzelfde bericht worden gebruikt, dit voorval noemen we ROT18.
b. Je kunt de quote niet volledig herkennen, een aantal karakters zoals “2” van “Groep-02” komen terug, ook de aanhalingstekens waar de quote tussen staat zie je terug in de encryptie.
c. ROT13 heeft geen vaste outputsize.

3. De principiële werking

a. ROT13 vervangt elke letter door het 13 plaatsen verder in het alfabet te verschuiven. Zo wordt bijvoorbeeld “A” vervangen door “N”.
b. ROT13 wordt gebruikt in online forums om spoilers, punchlines, puzzeloplossingen te verbergen.



Figuur 1: ROT13

4. Conclusie

ROT13 is een symmetrische encryptie met een specifieke key waarbij de letters van de string 13 plaatsen in het alfabet worden verschoven. Het heeft hetzelfde principe als ROT47, het enige verschil is dat dit algoritme maar 13 plaatsen verschuift terwijl het andere dit met 47 plaatsen doet. Het is een makkelijke cipher om te kraken en wordt daarom niet secure beschouwd voor encryptie omdat hij maar liefst één specifieke key vereist voor te decrypteren. Dit algoritme wordt gebruikt in forums om spoilers en puzzeloplossingen te verbergen.

5. AES:

Output

e731ff9a8716299e013f67808699c4f5ca5e9623d8aba19482f608b7ceef2ac631ae599e125b070b3
0bc835d805c84395684198379093c9fc9308fb0a0b8bd4b

Vragen

1. Type van het algoritme

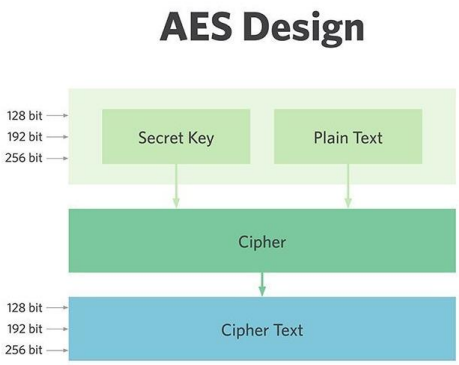
a. AES (Advanced Encryption Standard) is de eerste en enige encryptiemethode die is goedgekeurd door de National Security Agency (NSA) voor de bescherming van topgeheime (confidentiële) informatie. Het AES-algoritme gebruikt 3 sleutellengtes: 128, 192 en 256-bit.
b. AES is een symmetrische encryptie-algoritme.
c. We kunnen wel zeggen dat AES een secure algoritme is. Het is daarom een van de meest gebruikte algoritmen door overheidinstanties etc.
d. Het Kerckhoffs principe is van toepassing op het AES-algoritme.

2. Hoe kan je de cipher herkennen?

a. Je kunt het algoritme herkennen aan de cipher-tekst die oftewel 128, 192 of 256-bit lang zijn.
b. Een alternatief voor AES zou RSA kunnen zijn, beide algoritmen worden gebruikt in datacommunicatie voor het encrypteren van data.
c. Je kunt de quote niet herkennen wanneer deze geëncrypteerd is met het AES-algoritme.
d. Het AES-algoritme heeft geen vaste outputsize.

3. De principiële werking

a. Het AES-algoritme werkt als volgt: de bytes vervangen, rijen opschuiven, kolommen mixen en als laatste de key van de ronde toevoegen.
b. Dit algoritme wordt gebruikt bij overheidsinstanties, financiële instellingen en VPN-systemen. Het is daarom ook een van de populairse encryptie-methode die wordt gebruikt vanwege zijn betrouwbaarheid.



Figuur 2: AES

4. Conclusie

AES is een symmetrische encryptie-methode met een sleutellengte van 128, 192 en 256-bits, dat gebruikt wordt voor het beschermen van confidentiële informatie. Het algoritme is door de NSA beschouwd als een van de meest betrouwbare encryptie-methoden. De werking is simpelweg door de bytes te vervangen, bytes op te schuiven, kolommen te mixen en vervolgens de key van de ronde toe te voegen. Tegenwoordig wordt het gebruikt door overheidinstanties en financiële instellingen. Daarom is het ook een van de populairse technieken om te gebruiken.

6. Bcrypt:

Output

$2a$04$0wUq6YkEaxePbS9lkMN1OOlRStyvryQMEbSOytPbylYgUCzIZPnJi

Vragen

1. Type van het algoritme

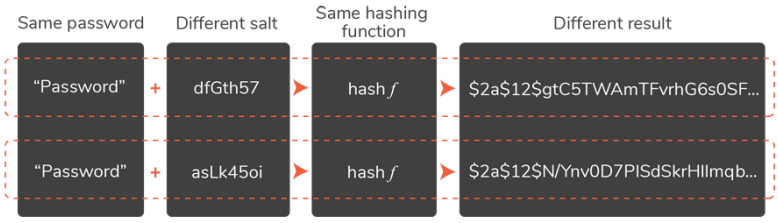
a. Bcrypt is een hashing algoritme ontworpen door Niels Provos en David Mazières dat gebruikt wordt om wachtwoorden te encoden. Het algoritme is gebaseerd op Blowfish, daarom wordt het ook Bcrypt genoemd.
b. Bcrypt is een hashingmethode.
c. Tot op de dag van vandaag kunnen we nog altijd zeggen dat Bcrypt een secure algoritme. Het is ontworpen om zich aan te passen, zodat het moeilijker wordt voor hackers het algoritme te kraken.
d. Security by obscurity.

2. Hoe kan je de cipher herkennen?

a. De eerste 4-digits van de encryptie duiden aan over welk algoritme het gaat, gevolgd door 3 tekens die verwijzen naar de opties van het algoritme. Als derde zie je welke salt op de encryptie is toegepast en tot slot je geëncrypteerde string.
b. Een goed alternatief voor Bcrypt is Pufferfish2, Pufferfish2 is gebaseerd op het idee van Bcrypt.
c. Je kunt de quote niet herkennen wanneer er een Bcrypt hash op is toegepast.
d. Bcrypt heeft een vaste outputsize van 60.

3. De principiële werking

a. Bcrypt verwijst naar Blowfish, dit algoritme gebruikt een salt van 128 bits en versleutelt een 192-bits magische waarde door gebruik te maken van de sleutelconfiguratie in eksblowfish.
b. Bcrypt wordt voornamelijk gebruikt voor het veilig encrypteren van wachtwoorden. Het wordt gebruikt wanneer een gebruiker een wachtwoord ingeeft dat vervolgens opgeslagen moet worden in een database op een manier dat het origineel wachtwoord niet herkend kan worden.



Figuur 3: Bcrypt

4. Conclusie

Bcrypt is een algoritme dat gebruikt wordt voor het hashen van wachtwoorden wanneer deze wordt ingegeven door de gebruiker. Het algoritme heeft een vaste outputsize van 60 en gebruikt een salt van 128 bits. Salting is gewoon "willekeurig gegevens die worden gebruikt als extra invoer voor een eenrichtings-hashfunctie." Het is een vrij secure algoritme omdat het gemaakt is om zich aan te passen.

7. Enigma:

Output

EQHND TZGHA GREMG ZLEAC YGVPY YAJDR UYUPA KAMXT IMAAD N

Vragen

1. Type van het algoritme

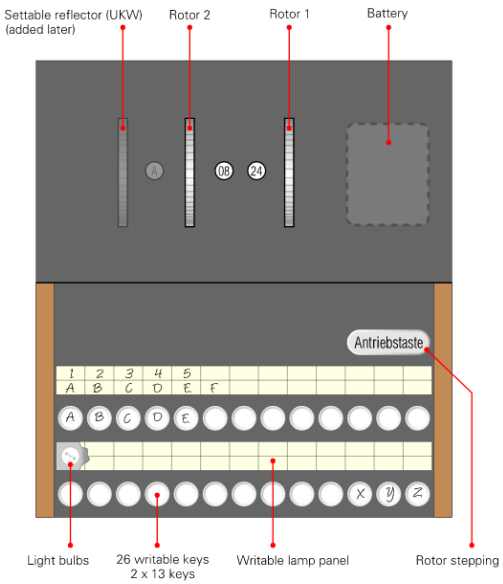
a. Het Enigma algoritme is ontworpen door in de twintigste eeuw tijdens de tweede wereldoorlog om commerciële, diplomatische en militaire communicatie te beschermen.
b. Enigma is een symmetrische encryptie-methode.
c. Tijdens wereldoorlog II werd het Enigma-algoritme heel secure beschouwd dat het werd gebruikt om top secret militaire informatie te encrypteren. Op de dag van vandaag is het heel erg makkelijk om de encryptie te kraken met een moderne computer.
d. Security through obscurity.

2. Hoe kan je de cipher herkennen?

a. De cipher heeft altijd een vaste outputsize van 56, hieraan kan je het algoritme vaak herkennen.
b. TypeX was een algoritme gebaseerd op Enigma, beide algoritmen sturen het signaal twee keer door de rotors.
c. Je kunt de quote niet terug herkennen.
d. Enigma heeft een vaste outputsize van 56.

3. De principiële werking

a. Het algoritme stuurt elektrische signalen van een typewriter door een reeks roterende wielen, vervolgens werd de uitvoer vervormd.
b. Enigma werd gebruikt tijdens de tweede wereldoorlog op top secret militaire communicatie te encrypteren.



Figuur 4: Enigma

4. Conclusie

Enigma maakt gebruik van symmetrische encryptie en werd tijdens de tweede wereldoorlog gebruikters door de Duitse NAZI voor het encrypteren van confidentiële militaire communcatie. Het algoritme werd gekraakt door Alan Turing en wordt sindsdien niet meer secure beschouwd. Het werkt door elektrische signalen van een typemachine door een reeks roterende wielen te sturen wat als volgt de uitvoer vervormd.

8. XOR:

Output

```
..vM.gÊq.l.n.".w.l.p.`.n.v.".qÊc.nÊc.".v.c.i.pÊl.g.q..w"à/¾g.oÊ0Ç
```

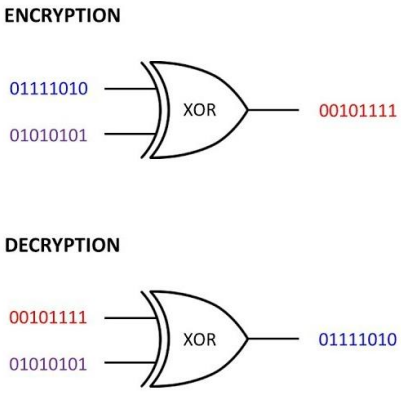
Vragen

1. Type van het algoritme

a. XOR is een effectieve en eenvoudig te implementeren algoritme. Het XOR-algoritme is een voorbeeld van symmetrische encryptie waarbij dezelfde sleutel gebruikt wordt om zowel een string te encrypteren als te decrypteren.
b. XOR is een symmetrische encryptie.
c. XOR is geen veilig algoritme omdat het bij lange runs van dezelfde tekens het heel gemakkelijk maakt om het wachtwoord te zien.
d. Kerckhoff's principe.

2. Hoe kan je de cipher herkennen?

a. Je kunt dit algoritme herkennen aan
b. Het eXclusive OR-algoritme heeft geen alternatieven.
c. De quote is niet te herkennen in de encryptie.
d. XOR maakt geen gebruik van een vaste outputsize.



Figuur 5: XOR

3. De principiële werking

a. XOR gaat te werk door twee input bits te vergelijken en genereert vervolgens één uitvoerbit. Wanneer de bits hetzelfde zijn, is het resultaat altijd 0.
b. Dit algoritme wordt veel gebruikt in cryptografie voor het genereren van pariteitsbits voor foutcontrole en fouttolerantie.

4. Conclusie

Het eXclusive OR-algoritme is een symmetrische encryptie-methode waar we dezelfde sleutel gebruiken voor het encrypteren en decrypteren van gegevens. XOR vergelijkt twee input bits en genereerd vervolgens een uitvoer bit. Qua security kunnen we dit algoritme niet als secure beschouwen omdat het bij lange runs het mogelijk maakt om de geëncrypteerde string te zien. Genereren van pariteit bits voor foutcontrole en fouttolerantie is een van de fields waar het algoritme gebruikt kan worden.

9. DES:

Output

```
7431e5e5789ae646d790391ff69aea6df4583b4a057ba4617bd3a23ecb176239f137ddd0ebc981450

35f385016bda5d954a921d738d0fae9a8809f986e635998
```

Vragen

1. Type van het algoritme

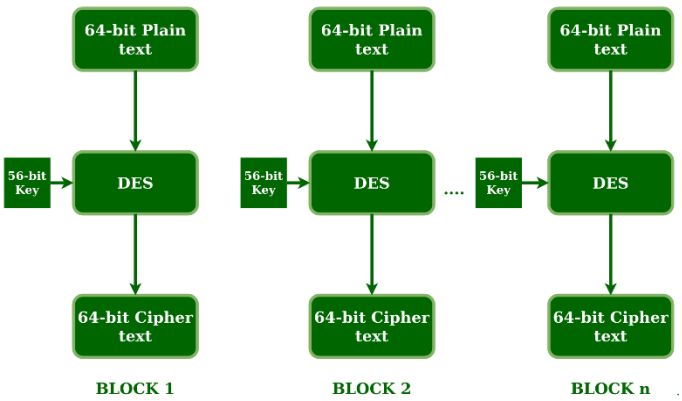
a. DES is een encryptie-algoritme bedacht door IBM-onderzoeker Horst Feistel. Het DES-algoritme gebruikt een Feistel-structuur van 16 rondes met voor elke ronde een andere sleutel.
b. DES is een symmetrische encryptie-algoritme.
c. Tegenwoordig wordt DES niet meer secure beschouwd, het algoritme kreeg een verbetering genaamd 3DES, maar ook die is zo secure niet.
d. Kerckhoff's principe.

2. Hoe kan je de cipher herkennen?

a. Deze cipher is te herkennen aan zijn 64-bit lange encrypties, deze komen +- overeen met 16 hex getallen.
b. DES lijkt behoorlijk veel op 3DES. Dit komt doordat 3DES een vervolg is op DES
c. Wanneer iets is geëncrypteerd met DES, is de geëncrypteerde string niet te herkennen.
d. Het Data Encryption Standard heeft geen vaste output size.

3. De principiële werking

a. DES werkt door groepen van 64-bits te encrypteren, dit komt overeen met 16 hexadecimale getallen. Om de encryptie uit te voeren, gebruikt DES keys die ook 16 hexadecimale getallen lang zijn.
b. DES werd gebruikt voor het encrypteren en decrypteren van non-classified computergegevens die werden gegenereerd door de Amerikaanse overheid.



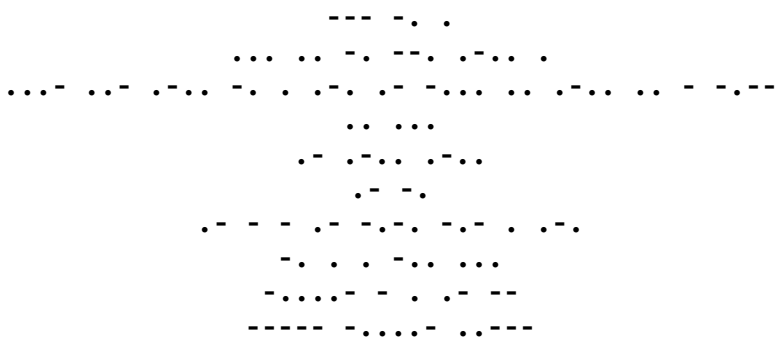
Figuur 6: DES

4. Conclusie

Het Data Encryption Standard is een symmetrische encryptie dat bedacht werd door IBM-onderzoeker Horst Feistel. Het werkt door groepen van 64-bits te encrypteren wat vervolgens overeenkomt met 16 hexadecimale getallen. Om deze encryptie uit te voeren gebruikt DES daarom keys die 16 hexadecimale getallen lang zijn. DES wordt tegenwoordig niet meer secure beschouwd en heeft daarom een verbetering gekregen genaamd 3DES (Triple DES). Dit algoritme werd gebruikt voor het encrypteren en decrypteren van non-classified computergegevens.

10. MORSE:

Output



Vragen

1. Type van het algoritme

a. Morsecode is een methode die gebruikt wordt in telecommunicatie om tekst te coderen als gestandaardiseerde reeksen van twee verschillende signaalduur, genaamd puntjes, streepjes of 'dits' en 'dahs'.
b. Morsecode is een substitutie-cipher, het wordt ook wel "**fixed-substitution**" genoemd omdat er geen key is.
c. Morsecode is geen beveiligd algoritme, mensen die geskilled en geëduceerd zijn in Morse zouden deze kunnen kraken.
d. Het Kerckhoffs principe is van toepassing.

2. Hoe kan je de cipher herkennen?

a. Je zult meteen opmerken dat Morsecode niet op andere ciphers lijkt, je kunt het herkennen aan de puntjes en streepjes.
b. Qua werking lijkt Tap Tap code wel op Morsecode, ze gebruiken beide tapping methoden.
c. Je kunt de quote absoluut niet herkennen.
d. Morsecode heeft geen vaste outputsize.

3. De principiële werking

a. Morsecode wordt gebruikt door op de benodigde combinatie van puntjes en streepjes te tikken en te pauzeren voor de juiste tijdsduur van de tussenruimte. Er zijn langere openingen tussen woorden dan letters in een woord.
b. Tegenwoordig wordt Morsecode gebruikt om noodsignalen te versturen.

4. Conclusie

Morsecode is een substitutie-methode om tekst te encoden als twee reeksen van twee verschillende signaalduur, zoals "streepjes" of "puntjes". Door te tikken en te pauzeren op de benodigde combinatie van puntjes en streepjes wordt de juiste tijdsduur van de tussenruimte bepaald. Het algoritme is op dag van vandaag nog altijd secure, gezien het een geskilled persoon vereist om de code te verstaan, daarom wordt het ook gebruikt om noodsignalen te versturen.

11. MD5:

Output

```
580e6ea451763559db4cb5e870189126
```

Vragen

1. Type van het algoritme

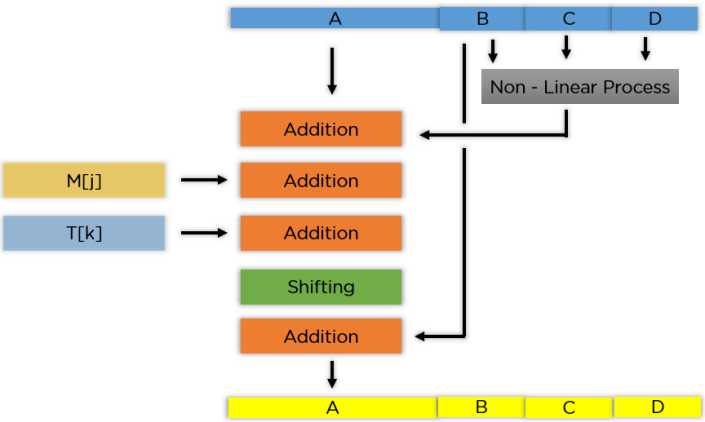
a. MD5 is een **Message-Digest Algorithm**, waarvan het doel is om te controleren of een bestand ongewijzigd is gebleven.
b. MD5 is een hashing-algoritme.
c. MD5 is in het algemeen een goed algoritme, maar het is onveilig voor het opslaan van wachtwoorden, omdat het te snel is.
d. Security through obscurity.

2. Hoe kan je de cipher herkennen?

a. Een MD5 hash is te herkennen aan zijn 32 karakters en hexadecimale getallen. Het algoritme is 128-bits.
b. Een alternatief voor MD5 is SHA-256.
c. Wanneer een string is gehashed met MD5, is die niet te herkennen.
d. MD5 heeft een vaste outputsize van 32.

3. De principiële werking

a. MD5 verwerkt de gegevens in een 512-bit strings, opgesplitst in 16 woorden die elk uit 32 bits bestaan.
b. MD5 wordt tegenwoordig gebruikt om te controleren of bestanden ongewijzigd blijven. Vroeger werd het gebruikt om wachtwoorden geëncrypteerd in databases op te slaan, omdat dit algoritme niet meer veilig is wordt dit niet meer gedaan.



Figuur 7: MD5

4. Conclusie

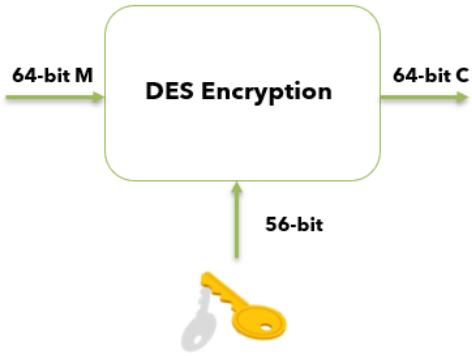
MD5 of Message Digest Algorithm is een hashing-functie wat als doel heeft om de integriteit van bestanden te controleren. Het algoritme heeft een vaste outputsize van 32 en verwerkt de gegevens in een 512-bit string, opgesplitst in 16 woorden die elk uit 32 bits bestaan. Het algoritme wordt tegenwoordig niet meer secure beschouwd en wordt dus enkel gebruikt om te controleren of een bestand ongewijzigd is gebleven, daarom is er een beter alternatief genaamd SHA-256.

4. From DES to AES

Opdracht gemaakt door: Aleyna

1. DES – Data Encryption Standard

a) DES is een encryptie-algoritme bedacht door IBM-onderzoeker Horst Feistel. Het DES-algoritme gebruikt een Feistel-structuur van 16 rondes met voor elke ronde een andere sleutel.



Figuur 8: DES Encryption

b) We kunnen dit algoritme niet langer veilig noemen. Hoewel het algoritme geen grote zwakheden heeft, is het fundamenteel onvoldoende omdat de 56-bits sleutel te kort is. → Vulnerable voor **Brute Force Attacks**

c) 3DEA ook wel **T**riple **D**ata **E**ncryption **A**lgorithm genoemd, is een symmetric key-block cipher algoritme, die het DES-algoritme driemaal toepast door te encrypteren met de eerste sleutel, decrypteren met de tweede en ten slotte te encrypteren met de derde sleutel. Tegenwoordig wordt 3DES niet veel gebruikt, dit komt omdat het algoritme gevoelig is tegen **Brute Force Attacks**.

2. AES – Advanced Encryption Standard

a) **AES** is de eerste en enige encryptiemethode die is goedgekeurd door de National Security Agency (NSA) voor de bescherming van topgeheime (confidentiële) informatie. Het AES-algoritme gebruikt 3 sleutellengtes: 128, 192 en 256-bit.

b) Nadelen van **AES** zijn:

- Elke blok is op dezelfde manier geëncrypteerd
- Moeilijk te implementeren met software
- Gebruikt een veel te makkelijke algebraïsche structuur

3. DES & AES

a) Verschillen tussen **AES** en **DES**:

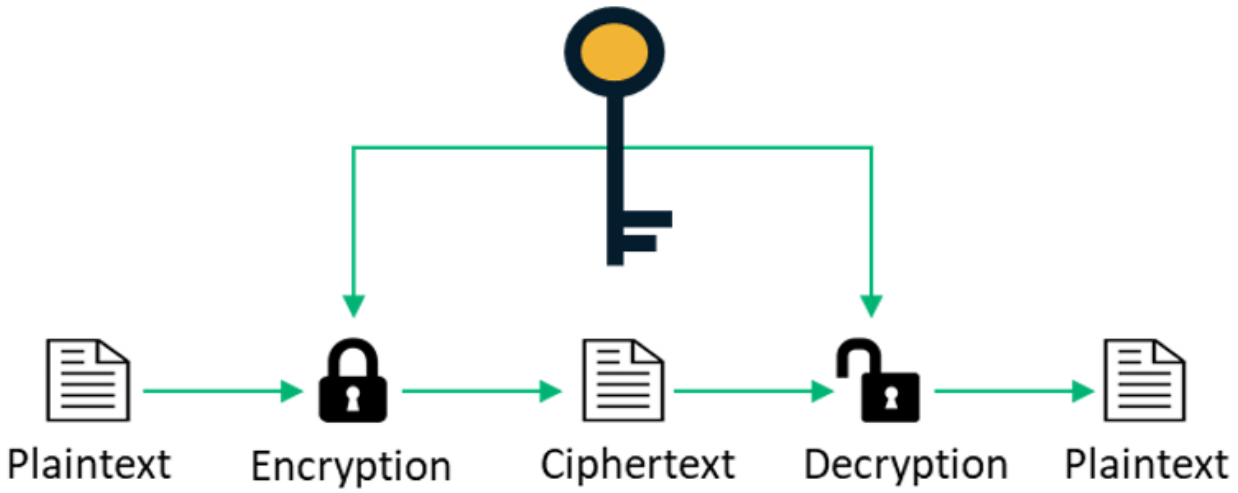
- **AES** is byte-georiënteerd terwijl **DES** bit-georiënteerd is.
- Sleutellengte van **AES** kan 128, 192 of 256-bit zijn terwijl **DES** een sleutellengte van 56-bit heeft.
- Het aantal ronden bij **AES** hangt af van de sleutellengte terwijl bij **DES** 16 ronden van identieke operaties omvat.
- **AES** is veiliger terwijl **DES** gemakkelijk gebroken kan worden omdat er bekende kwetsbaarheden inzitten.

b) Waar wordt **AES**, **DES** en **3DES** gebruikt?

- De dag van vandaag wordt **AES** gebruikt voor:
    - Wireless security (WPL2)
    - SSL/TLS protocollen
    - Mobiele telefoon encrypties
    - VPN
- **DES** werd gebruikt voor het encrypteren en decrypteren van niet-geclassificeerde computergegevens die gegenereerd werden door de regering van de VS en andere organisaties.
- **3DES** wordt tegenwoordig nog toegepast in de financiële sector voor betalingsmogelijkheden, om data in transit en data in storage te encrypteren.

4. Conclusie

We kunnen hieruit concluderen dat DES en 3DES niet zo betrouwbaar meer zijn en tegen 2024 zullen pensioneren. De reden waarom AES, het DES-algoritme vervangen heeft is voornamelijk omdat het algoritme zowel in software als hardware efficiënt geïmplementeerd moet kunnen worden.



Figuur 9: Symmetric Block Cipher

5. RSA - DH

Opdracht gemaakt door: Stef

Vragen

a.

p = 131
q = 199

n = p * q = 131 * 199 = 26069
φ(n) = (p – 1) * (q – 1) = 130 * 198 = 25740
e = 7
(d * e) mod φ(n) = 1 à (47803 * 7) mod 25740 = 1

RSA behoort tot asymmetrische encryptie waarbij er een private key nodig is om de ciphertext te ontcijferen. Om de communicatie te doen slagen beschikken beide partijen over elkaars public key en hebben ze zelf elk een private key.

We beginnen met het bepalen van twee priemgetallen, p en q. Wij kozen voor 131 en 199 in dit voorbeeld. De public key bestaat uit twee waarden, n en e. n is een modulus en wordt berekend door de twee priemgetallen met elkaar te vermenigvuldigen: 131 * 199 = 26069. E is een willekeurig gekozen getal dat aan drie voorwaarden voldoet: het is een priemgetal, het is kleiner dan ϕ(n) en het is geen deler van ϕ(n). In dit voorbeeld nemen we e = 7. Zo wordt de private key met de waarden van n en e (7, 26069).

De public key bereken we met het de formule van euler. Deze key bestaat uit de waarden d en n. Dit wordt dan (47803, 26069).

b. Het verschil tussen realistische waarden en de waarden die wij gebruikt hebben is vrij groot. Een realistische waarde heeft een 7-8 bit getal, onze waarden zijn veel kleiner.

c. Hetzelfde geldt voor de realistische keys. Deze zijn veel groter en dus moeilijker te kraken, onze waarden zijn klein en dus makkelijk om te kraken.

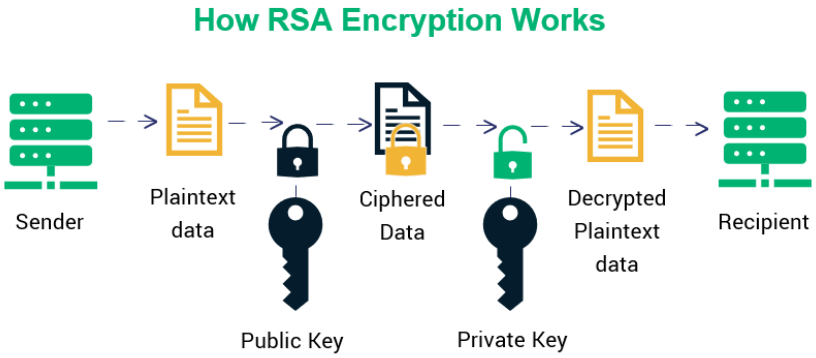
d. **p & q:** dit moeten priemgetallen zijn, liefst zo groot mogelijk om het moeilijk te maken voor krakers.
e: het is een priemgetal, het is kleiner dan ϕ(n) en het is geen deler van ϕ(n)
d: (d * e) mod ϕ(n) = 1

Conclusie

RSA is een veilig encryptiemodel zolang de gekozen priemgetallen voor p en q groot genoeg zijn, het zou lang duren om dit te kraken en dat is voordelig voor de gebruikers ervan.

RSA Praktisch

RSA is secure doordat het asymmetrische encryptie is. Dit wil zeggen dat we met 2 verschillende keys werken, een private en een public key. De public key valt nog te achterhalen, de private key is veel moeilijker zolang de gekozen priemgetallen groot genoeg zijn.

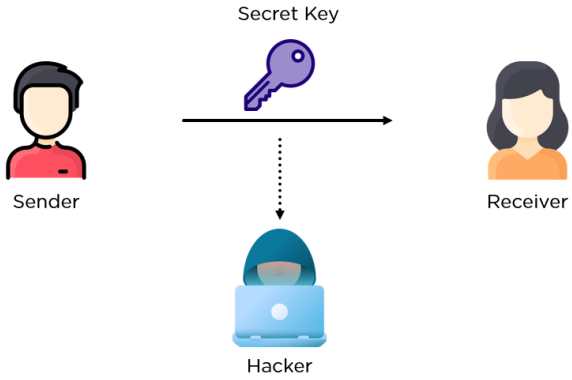


De sleutelgrens ligt op een minimaal van 3072 bits. Kwantumcomputers zijn enorm snelle machines die veel data kunnen analyseren. Ze maken gebruik van kwantummechanica, dat wil zeggen dat de machines gebruik maken van de allerkleinste deeltjes in het universum. Deze deeltjes zouden volgens kwantummechanica op verschillende plaatsen of toestanden kunnen zijn op hetzelfde moment. Er ontstaat een golfpatroon waar de kans groter is op deeltjes bij een hoge amplitude en een kleinere kans op deeltjes bij een kleinere amplitude. Hoe meer deeltjes er zijn op een bepaalde plaats, hoe meer combinaties er mogelijk zijn.

Ze kunnen we enorm veel werk tegelijk verrichten. Dit maakt het makkelijker voor een kwantumcomputer om om keys te ontcijferen. Het gaat daarom ook sneller dan met een normale computer.

Diffie Hellman

Bij DH wordt er langs twee kanten van een transactie een private en public key gegenereerd, hiervan is de public key hetzelfde. Dit is een vorm van asymmetrische encryptie. Met RSA kan men zowel symetrisch als asymmetrisch encrypteren. Hiervoor is er een uitwisseling nodig van de public keys, anders zou het niet gaan. De truc hiervan is dat de wiskundige functie die hiervoor gebruikt is makkelijk in een richting te berekenen is, maar heel moeilijk om te keren ookal zijn bepaalde aspecten hiervan bekend.

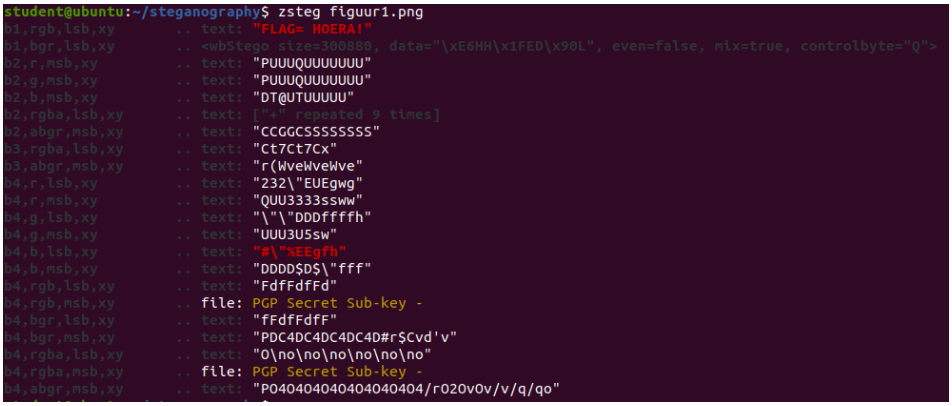


6. Steganography

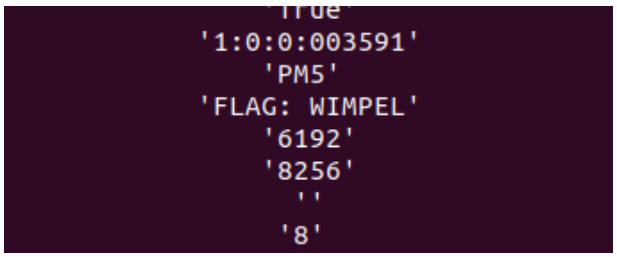
Opdracht gemaakt door: Stef

Figuur1.jpeg → flag + verklaring

Voor de eerste image moest men eerst **rubygems** installeren met het commando ***sudo apt install rubygems.*** Hierna kan je het commando **zsteg** installeren met ***sudo gem install zsteg***. De flag is te vinden met het commando ***zsteg figuur1.png***: **'FLAG= HOERA!'**.



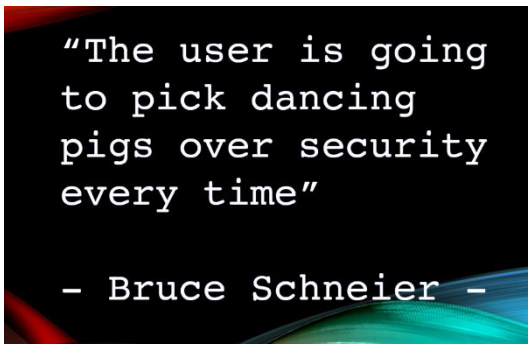
Figuur2.jpeg → flag + verklaring



De tweede flag is te vinden met het commando ***stegoveritas -xmp Figuur2.png.*** Hiervoor moet je eerst python3 installeren via ***sudo apt install python3-pip*** en ***sudo pip3 install stegoveritas***. Dan is de flag te vinden in een grote tabel: **'FLAG: WIMPEL'.**

BSOD.dub → flag + verklaring

Het derde document was een bestandsmap met zes foto's. Dit was enkel te zien in een mijn virtuele machine van Linux, op mijn eigen Windows. Elk van deze foto's had in een patroon een aantal letters staan. Deze foto's heb ik over elkaar gezet in een dia van PowerPoint en zo de achtergrondkleur op doorzichtig ingesteld. Na dit te doen voor de zes afbeeldingen kreeg ik de quote te zien van Bruce Schneier.



7. THM – Steganography

Opdracht gemaakt door: Rasmus ⇨ (Screenshots te vinden in bijlage opdracht 7)

In deze THM Room werden verschillende tools behandeld die informatie kunnen extraheren of embedden in geluid en jpg/png files. De eerste sectie ging over het kommando *steghide*. Steghide is een gebruikelijk kommando wat de mogelijk biedt, verborgen strings in image/audio files te plaatsen. Ook kan het hier gebruikt worden om informatie te extraheren. Via de **`man`** page van **`steghide`** is het mogelijk de juiste flags te vinden, en met **`steghide extract -sf jpeg1.jpeg`** was het mogelijk de hidden message te vinden. Zie bijlage voor opdracht 7, foto1.

Het tweede gedeelte ging over *zsteg*, daarvoor moet men eerst rubygems installeren m.h.v **`sudo apt install rubygems`**, daarna **`sudo gem install zsteg`**, en met **`zsteg png1.png`**, was het mogelijk de verborgen text in png1 te vinden, zie bijlage opdracht 7, foto2.

Het derde gedeelte ging over het kommando *exiftool*, een handige tool om verborgen metadata te tonen. Met behulp van **`exiftool jpeg3.jpeg`** was het mogelijk de document naam te tonen, zie bijlage opdracht 7, foto3.

Om gebruik te maken van *stegoveritas*, moet men eerst python3 installeeren voor de terminal, met behulp van **`sudo apt install python3-pip`**, en daarna **`sudo pip3 install stegoveritas`**. Via de manpage van stegoveritas, die online te vinden is, waren de flags vindbaar, en met behulp van **`stegoveritas jpeg2.jpeg`**, kon men de verborgen message vinden, zie bijlage opdracht 7, foto4.

Met Sonic Visualizer kan men een spectogram layer toevoegen over een audio (wav) file, die ook tussen de bestanden zit, en met behulp daarvan kon men de verborgen text in de wav2file vinden, wat "**Google**" was, zie foto5.

Voor de exam questions waren er 3 "keys" die men moest vinden. Nadat men de AttackBox machine van THM opent, en de IP adress in de browser intypt, opent een browser scherm met een afbeelding. Als men die afbeelding downloadt, en **`exiftool`** op dat bestand (exam1.jpeg) uitvoert, verschijnt een passphrase tussen de metadata. Met behulp van **`steghide extract -sf exam1.jpeg`** kon men dan een text vinden, namelijk "**superkeykey**". (key1, foto 6, 7)

Voor de tweede key moest men een audiofile alayseren m.b.v. Sonic Visualizer, waarna een link te voorscheen komt. Op die link is een afbeelding van een potlood te vinden. Na een download voert men **`zsteg <potlood>.png`** uit, en vind de tweede key: "**fatality**". (key2, foto 8, 9, 10).

Als men de tweede key heeft ingetypt op de portal, komt een link met een QR-code tevoorschijn. Als men die scant is die leeg. Maar als men **`stegoveritas`** gebruikt om de QR-code te analyseren, dan komt een hele folder aan QR-codes tevoorschijn. Als men een van de hogere resolutie/unscrambled QR-codes inscant, dan komt de derde key tevoorschijn, namelijk "**killshot**". (key3)

Conclusie: deze THM leert de gebruiker belangrijke steganografische tools aan om basic decodings te doen van afbeeldingen en audiofiles. Met behulp van **`steghide`**, **`zsteg`**, **`exiftool`**, **`stegoveritas`** en Sonic Visualizer leert men hoe men naar verborgen messages kan zoeken, en hoe men onder anderen verborgen metadata extraheert. Deze tools vormen een goede basis voor beginners om afbeeldingen van verschillende formaten en audiofiles steganografisch te benaderen.

8. Kraken van de encryptie

Opdracht gemaakt door: Rasmus → (Python script te vinden in bijage opdracht 8) ←

In deze opdracht was de bedoeling om een script te schrijven wat een frequentieanalyse zou kunnen uitvoeren op een bepaalde cipher. Ik zal in deze beschrijving verwijzen naar het script wat zich aan het einde van de Labo bevindt.

Disclaimer: Een script te schrijven wat na de frequentieanalyse ook nog de cipher in kwestie zou kunnen kraken aanhand van de frequentieanalyse is boven het niveau van onze huidige programeer skills. De bron die gebruikt werd voor het script om de frequentieanalyse te kunnen doen was: letterfrequentie in het Nederlands | Genootschap Onze Taal | Onze Taal

Volgens de bron:

"De onderstaande lijst is gebaseerd op onderzoek uit 1985 van de Stichting voor Publieksvoorlichting over Wetenschap en Techniek *(PWT). Voor haar onderzoek naar letterfrequenties baseerde de stichting zich op een corpus van anderhalf miljoen woorden redactionele krantentekst (afkomstig uit* De Haarlemse Courant*)."*

Dit betekent dat een frequentieanalyse werd gemaakt aanhand van 500 000 woorden, maar het betekent niet dat elke paragraaf tekst aan deze standaard zal voldoen. Dit is de hoofdredenen waarom het uiteindelijke kraken van de cipher manuel moest gebeuren, nadat het script de frequentieanalyse had doorgevoerd.

Het Python script: flag → **{Appelmoes}**

3 belangrijke functies:

1. **def getLetterCount(message):** deze functie krijgt de cipher mee als parameter. Aanhand daarvan maakt die een een array aan, en houd een bibliotheek bij aan voorkomende letters binnen de array.

Dit ziet er zo uit:

```
def getLetterCount(message):
    letterCount = {'A': 0, 'B': 0, 'C': 0, 'D': 0, 'E': 0, 'F': 0,
                   'G': 0, 'H': 0, 'I': 0, 'J': 0, 'K': 0, 'L': 0, 'M': 0,
                   'N': 0,'O': 0, 'P': 0, 'Q': 0, 'R': 0, 'S': 0, 'T': 0,
                   'U': 0, 'V': 0,'W': 0, 'X': 0, 'Y': 0, 'Z': 0}
    for letter in message.upper():
        if letter in LETTERS:
            letterCount[letter] += 1

    return letterCount
```

2. def getFrequencyOrder(message): deze functie krijgt weer de cipher mee als parameter. Binnen deze functie worden all letters van de letter bibliotheek boven toegevoegd aan een nieuwe array. Dan wordt die array opnieuw gesoorteerd aanhand van een key, namelijk de 'ENATIR' key, oftewel de variabele die bovenaan het script gedeclareerd is. De ENATIR key zijn alle letters die het meeste voorkomen in de Nederlandse taal, op volgorde genoteerd. De functie getFrequencyOrder retourneert dus uiteindelijk een string van 26 alfabetische chars, op volgorde van het meest voorkomend gesoorteerd.

3. def nlFreqMatchScore(message): retourneert een score die kijkt hoeveel van de eerste 6 meest voorkomende letters van de string uit de functie getFrequencyOrder ook voorkomen in de ENATIR key. Daaruit kan men concluderen met welke letters een substitutie gemaakt kan worden.

De uiteindelijke output van het script:

```
De meest voorkomende letters in NL op volgorde zijn:            ENATIRODSLGVHKMUBPWJZCFXYQ

De meest voorkomende letters in de cipher in volgorde zijn:     SIGHANQUCTYJFZMWORVXDEBPKL

De 6 meest voorkomende letters in volgorde, in NL zijn:         ENATIR

De 6 meest voorkomende letters in de cipher zijn:               SIGHAN

Het aantal meest voorkomende letters van de cipher
die ook in de eerste 6 letters van ENATIR present zijn:         3

De frequentie analyse:
                        {'A': 32, 'B': 2, 'C': 22, 'D': 3, 'E': 3, 'F': 15, 'G': 36, 'H': 32,
                         'I': 44, 'J': 16, 'K': 0, 'L': 0, 'M': 11, 'N': 31, 'O': 8, 'P': 1,
                         'Q': 28, 'R': 8, 'S': 110, 'T': 19, 'U': 25, 'V': 6, 'W': 8, 'X': 5,
                         'Y': 18, 'Z': 11}

Message:
                dit is een voorbeeld van een tekst die onleesbaar is gemaakt met behulp
                van substitutie.de letters in deze tekst zijn vervangen door andere letters.
                door gebruik te maken van eenfreduentietabel kan je deze tekst terug leesbaar
                maken. zo kan je de meest voorkomende letters in deze tekst vervangen door de
                meest voorkomende letter volgens de freduentietabel. de meest voorkomende
                letter in onze taal is de ‘e’.  je zal zien dat je snel een leesbare tekst
                uitkomt. een bijkomende vraag voor deze opdraeht is, hoe kan je deze vorm
                van enerpptie sterker maken? de flag die bij deze opdraeht hoort is {Appelmoes}.
```

Uiteindelijk heb ik de message manueel ge-descrambled, aanhand van de frequentie analyse. Ik begon natuurlijk met de letter ‘s’ te vervangen met de letter ‘e’, omdat ‘s’ bij verre het meest voorkwam in de cipher. Door middel van een soort ‘kruis-systeem’, was het mogelijk stuk voor stuk de letters te vervangen, zonder dat men al vervangen letters opnieuw zou vervangen:

```
new_string = new_string.replace('s', 'e')
new_string = new_string.replace('u', 's')
new_string = new_string.replace('o', 'u')
new_string = new_string.replace('h', 'o')
new_string = new_string.replace('x', 'h')
```

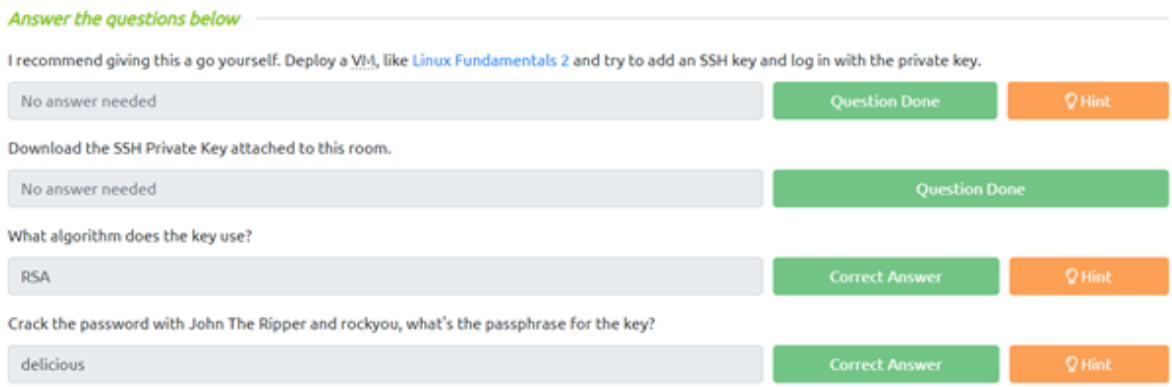
Zoals eerder beschreven kan dit script perfect een frequentieanalyse uitvoeren, maar geen cipher kraken. Om dat te doen moet men manueel letters vervangen, zoals hier links wordt aangeduid. Als u boven naar de output kijkt dat zijn de 3 meest voorkomende letters van de cipher ‘S’, ‘I’, en ‘G’.

Maar uiteindelijk is ‘S’ het einige letter wat vervangen kan worden door ‘E’, ‘I’ komt niet overeen met een vervanging van ‘N’, en ‘G’ niet overeen met een vervanging van ‘A’ (uit ENATIR). Uiteindelijk moet men manueel aan de hand van patroonherkenning de cipher deencrypteren.

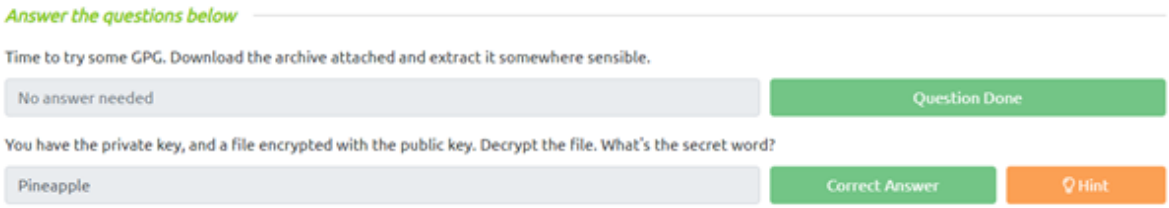
Conclusie: het is perfect mogelijk om aanhand van een python script een frequentie analyse uit te voeren over een cipher, alleen is het kraken van de cipher in kwestie aanhand van de frequentie analyse niet altijd mogelijk of even gemakkelijk. Door de frequentie analyse uit te voeren zal wel duidelijk worden welke letters het meeste voorkomen, maar de uiteindelijke vervanging van de meest voorkomende substituties zal aanhand van patroonherkenning moeten gedaan worden. Ook zijn de ‘standaard’ meest voorkomende letters uit het gestandardiseerde onderzoek van 1985 niet noodzakelijk de meest voorkomende letters in de cipher, wat in dit geval maar een paragraaf is.

9. THM - Crypto 101

Opdracht gemaakt door: Tomas



Bij deze opgave kregen we een .id_rsa file. En deze moesten we cracken om te weten wat de passphrase was voor de key. Hoe gaan we te werk? Eerst downloaden we de .id_rsa key, dit is een ssh private key, en zoals jullie kunnen zien gaat dit nogal moeilijk met John The Ripper. Dus we gaan deze eerst omvormen naar een hash met SSH2john. En dan gebruiken we john the ripper en een wordlist om de hash te cracken. En dan krijgen we de passphrase 'delicious'.



Bij deze opgave kregen we een key en een message als downloadable content, en de key konden we gebruiken om de secret wordt te vinden door deze the decrypten. Hiervoor gebruikte we de gpg tool. Eerst gingen we de key importen naar onze sleutelbos, zodat we deze kunnen gebruiken om files te decrypten, en dan gaan we de message decrypten, deze wordt dan een aparte file op onze opslagplaats, en deze kunnen we dan openen met 'cat' en daar staat de secret wordt aangegeven.

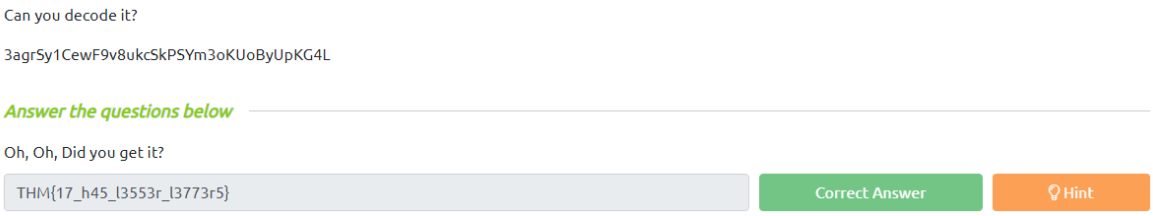
Conclusie

Deze room was een zeer korte uitleg over wat encryptie eigenlijk inhoudt, en waar deze gebruikt wordt. Deze room zat vol theorie, theorie die eerder al besproken is op deze paper. Deze room heeft ons aangeleerd over hoe we bijvoorbeeld een ssh private key kunnen omzetten en john the ripper hierop uitvoeren, of hoe we een geencrypted file kunnen decrypten met een key via Linux.

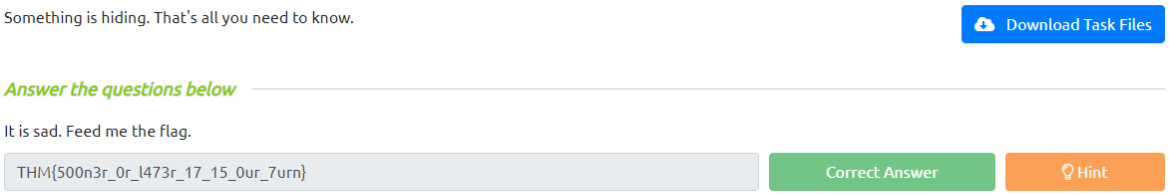
10. Crypto Challenges (extra)

Opdracht gemaakt door: Tomas

Voor deze opgave heb ik een simpele **CTF (Capture The Flag)** gedaan op THM. De allereerste van de series. Ik heb ze niet allemaal opgelost maar wel een paar die te maken hadden met steghide, of gewoon in de html zoeken. Of we moesten iets decoden zoals bij deze opgave:



Hier zagen we een cipher. Deze moesten we decrypten aan de hand van een online tool, deze moesten we decoden met Base58.



Of bij deze kregen we de kans om steghide eens uit te proberen.

11. VPN

Opdracht gemaakt door: Aleyna

PART 1: Research

Vraag 1

Via ProtonVPN heb ik verbinding gemaakt met een Amerikaans VPN-netwerk die als volgt **45.87.214.109** als publiek IP-adres teruggaf.

Mijn eigen IP-adres, zonder VPN geeft **84.192.90.XXX** als publiek IP-adres terug. (Door veiligheidsredenen zijn de laatste drie digits van mijn IP-adres gecensureerd).

Vraag 2

Het resultaat van de traceroute met VPN:

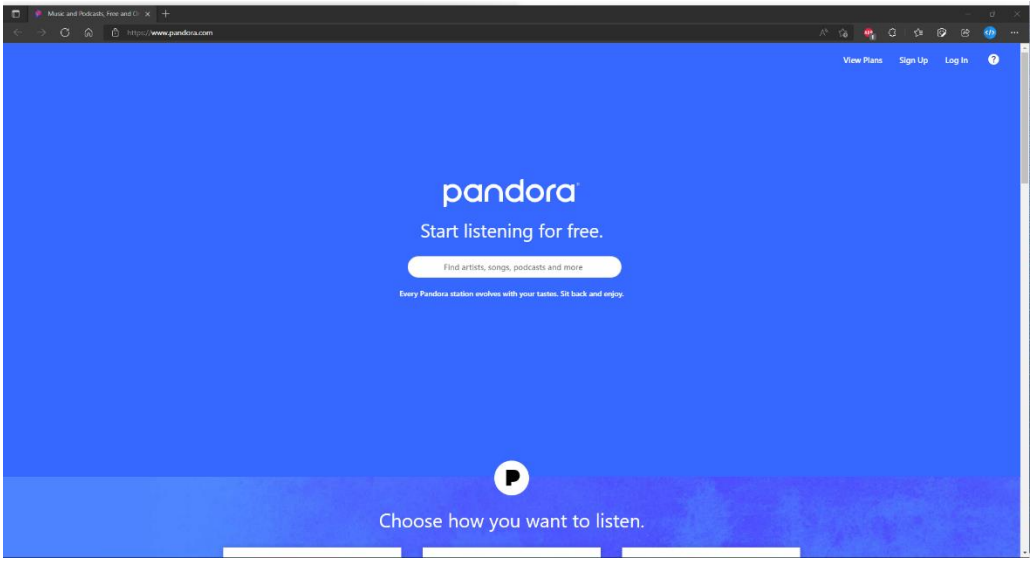
```
C:\Windows\system32>tracert www.pxl.be

Tracing route to www.pxl.be [193.190.154.243]
over a maximum of 30 hops:

  1   143 ms   140 ms   143 ms  10.30.0.1
  2   163 ms   195 ms   178 ms  vlan144.as07.mia1.us.m247.com [45.87.214.105]
  3   160 ms     *      200 ms  vlan2907.bb1.mia1.us.m247.com [212.103.51.84]
  4   194 ms   144 ms   186 ms  217.138.223.212
  5     *      216 ms   192 ms  te0-3-1-14.201.nr51.b002802-5.mia01.atlas.cogentco.com [38.140.53.65]
  6   172 ms   162 ms   166 ms  be3763.rcr21.b002802-2.mia01.atlas.cogentco.com [154.24.30.129]
  7   190 ms     *      179 ms  be3411.ccr22.mia01.atlas.cogentco.com [154.54.26.41]
  8   216 ms   211 ms   186 ms  be3483.ccr42.atl01.atlas.cogentco.com [154.54.28.49]
  9   171 ms   178 ms   198 ms  be2113.ccr42.dca01.atlas.cogentco.com [154.54.24.221]
 10   242 ms   250 ms   243 ms  be3111.ccr42.par01.atlas.cogentco.com [154.54.89.226]
 11   275 ms   294 ms     *     be3675.rcr21.bru01.atlas.cogentco.com [154.54.57.166]
 12   271 ms   291 ms   308 ms  be2525.nr61.b015929-1.bru01.atlas.cogentco.com [154.25.12.246]
 13   262 ms   285 ms   313 ms  149.11.150.2
 14     *        *       *     Request timed out.
 15   310 ms   345 ms   297 ms  pxl.r1.hasdie.belnet.net [193.191.8.29]
 16   280 ms   240 ms   253 ms  193.190.154.174
 17   274 ms   290 ms   300 ms  193.190.154.243

Trace complete.
```

Het resultaat van de traceroute zonder VPN:

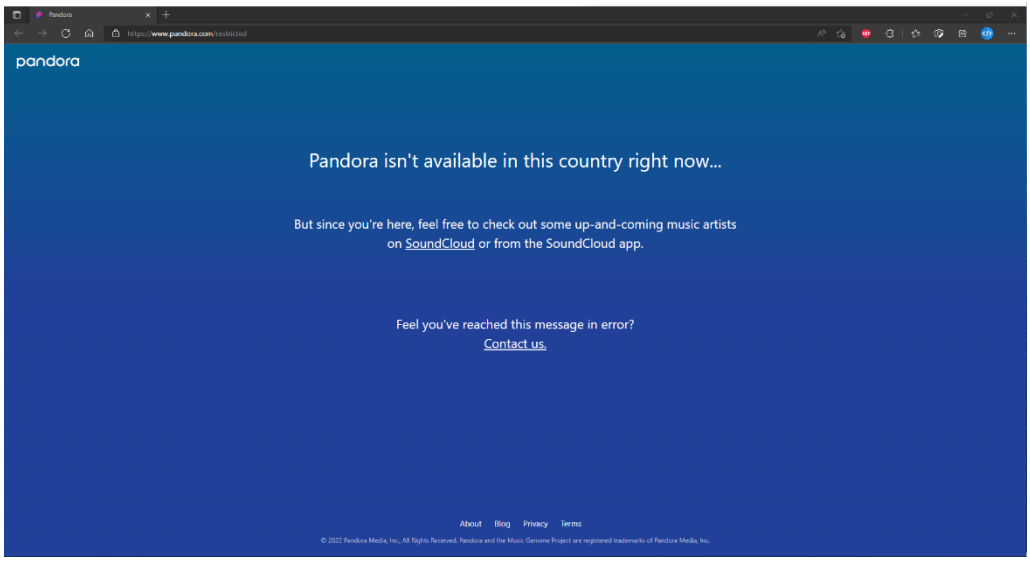
```
C:\Windows\system32>tracert pxl.be

Tracing route to pxl.be [193.190.154.242]
over a maximum of 30 hops:

  1     1 ms     1 ms     1 ms  192.168.0.1
  2    14 ms    13 ms    10 ms  d54C04001.access.telenet.be [84.192.64.1]
  3    12 ms    13 ms    15 ms  dD5E0CCE1.access.telenet.be [213.224.204.225]
  4    14 ms    14 ms    13 ms  dD5E0FA79.access.telenet.be [213.224.250.121]
  5    16 ms    14 ms    15 ms  bnix.brueve.belnet.net [194.53.172.65]
  6     *        *        *     Request timed out.
  7    22 ms    22 ms    14 ms  pxl.r1.hasdie.belnet.net [193.191.8.29]
  8    28 ms    26 ms    29 ms  193.190.154.174
  9     *       21 ms    21 ms  193.190.154.242

Trace complete.
```

Vraag 3

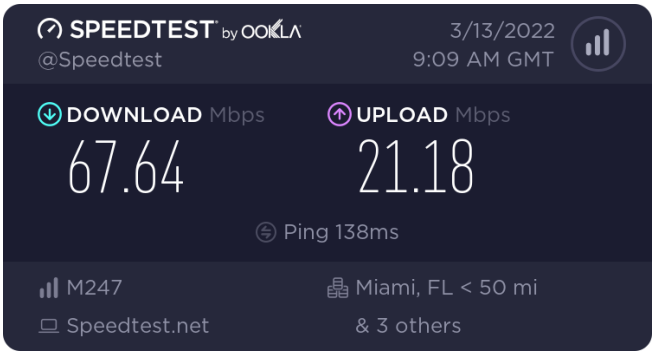
Via **`start iexplore`** **`www.pandora.com`** in CMD heb ik met de websiteverbinding proberen te maken, dit gaf als volgt resultaat:



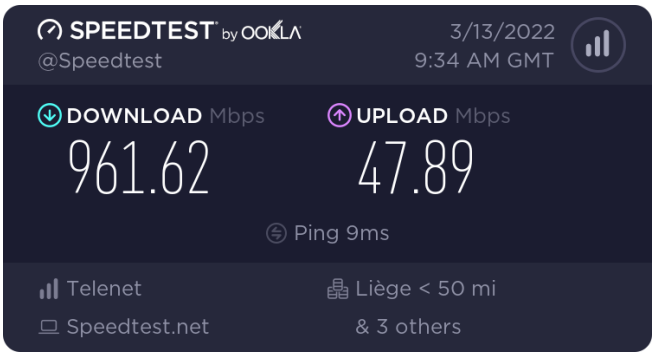
Vervolgens zul je merken dat deze verbinding zonder VPN niet gemaakt zal worden omdat Pandora sinds 2007 enkel toegang verleent aan Amerikaanse gebruikers vanwege copyright-houders. Gebaseerd op het IP-adres van de gebruikers uit andere landen, wordt toegang tot de website geblokkeerd.



Vraag 4



Figuur 10: Met VPN



Figuur 11: Zonder VPN

Je zult qua snelheid een groot verschil merken tussen de twee screenshots. De snelheidstest met VPN heeft een tragere snelheid vergeleken met de snelheidstest zonder VPN. Dit heeft meerdere redenen:

- Encryptie proces
- Afstand tussen gebruiker en server
- VPN-protocol dat gebruikt wordt
- Te veel gebruikers maken gebruik van dezelfde server waardoor de drops tussen de gebruikers verdeeld moeten worden

PART 2: Vragen m.b.t VPN

Vraag 1

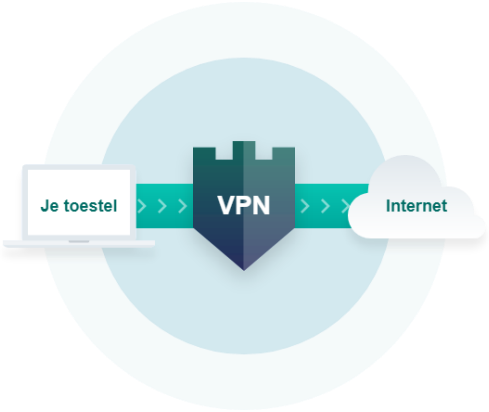
De term **VPN** staat voor **V**irtual **P**rivate **N**etwork.

Vraag 2

Een VPN helpt je om je IP-adres te verbergen door het netwerk om te leiden via een externe server die beheerd wordt door een VPN-host. Dat wil zeggen dat als je het internet verkent met een VPN, de bron van je gegevens worden vervangen door de gegevens van de VPN-server. Met andere woorden, derde partijen zoals uw internetprovider hebben geen zicht op je surfgedrag, omdat dit vertaald wordt naar “brabbeltaal”. We kunnen de VPN onder Integrity van de CIA TRIAD plaatsen, het belangrijkste punt van een VPN is de encryptie van packet contents tussen de endpoints die de VPN definiëren. Ook speelt VPN een belangrijke rol in de 3de dimensie van de McCumber framwork, genaamd “Cybersecurity Countermeasures”, VPN’s kunnen een goede oplossing bieden om jezelf met technologie te beschermen tegen attacks van buitenaf, om je identiteit online te verbergen en veilig te surfen op het internet.

Vraag 3

Er zijn een tal van redenen waarom je gebruik zou moeten maken van een VPN. In dit voorbeeld zullen 5 redenen aan bod komen waarom jij een VPN zou moeten gebruiken.



- Beveiliging op publieke WiFi-netwerken
- Om gegevens te beschermen tegen ISP
- Om gegevens te beschermen tegen de overheid
- Toegang tot diensten waar en wanneer je maar wil
- Beveiliging wanneer je op afstand werkt

Zoals vermeld bij vraag 2, helpt een VPN je privacy te beschermen zodat een derde partij of iemand die toegang heeft tot je WiFi-netwerk je online-activiteiten niet kan meevolgen. Wanneer je een website bezoekt, maakt je device verbinding met de server waarop de website wordt gehost, dit zorgt ervoor dat de website die je bezoekt een hoeveelheid van je data kan zien. Via een VPN wordt er verbinding gemaakt met een privéserver waardoor jouw eigen gegevens worden geëncrypteerd en het voor derde moeilijker wordt jouw surfgedrag te controleren.

a. **Privé:** in je privé-omgeving kan je omwille van verschillende redenen gebruik kunnen maken van een VPN. Dit verschilt van de behoeftes van de persoon die er gebruik van zou willen maken. Tegenwoordig maken veel mensen gebruik van een VPN voor streamingsdiensten zoals Netflix om meer content te unlocken die in andere landen wel beschikbaar zijn. Dit noemt men "**geo-blocking**".
b. **Schoolomgeving:** wanneer je verbonden bent met het schoolnetwerk, is de kans groot dat beheerders van het netwerk overzicht hebben op je internetgedrag. Dit kan je vermeiden met een VPN. Je IP-adres toont websites op basis van je locatie, wanneer je research doet voor een bepaald project, en de informatie die je nodig hebt op een website is waar jouw locatie geen toegang tot heeft, kan je dit simpelweg oplossen door gebruik te maken van een VPN.
c. **Werkomgeving:** bedrijven beschikken vaak over een VPN om externe toegang te verlenen tot interne applicaties of gegevens van het bedrijf. Het doel bij het gebruiken van een VPN in dit geval is om te vermijden dat deze gegevens bloot op het internet worden weergegeven.

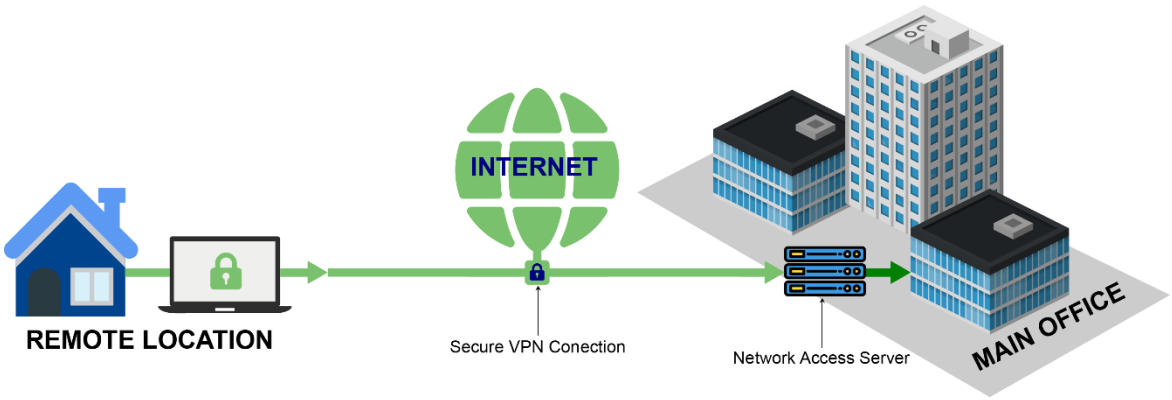
Vraag 4

Op basis van het OSI-model kunnen we VPN-tunnels plaatsen in laag 2 (datalinklaag) en laag 3 (netwerklaag). Om gegevens op laag 2 te verzenden, gebruiken VPN's "frames" en op laag 3 gebruiken VPN's "pakketten" om gegevens te verzenden. Het IPsec VPN-protocol werkt op de derde laag (netwerklaag) van het OSI-model. Het draait direct bovenop IP en is verantwoordelijk voor het routeren van pakketten. Aan de andere kant werkt SSL of TLS op laag 7 (applicatielaag) van het OSI-model, dat HTTP-verkeer versleutelt in plaats van rechtstreeks IP-pakketten te versleutelen.

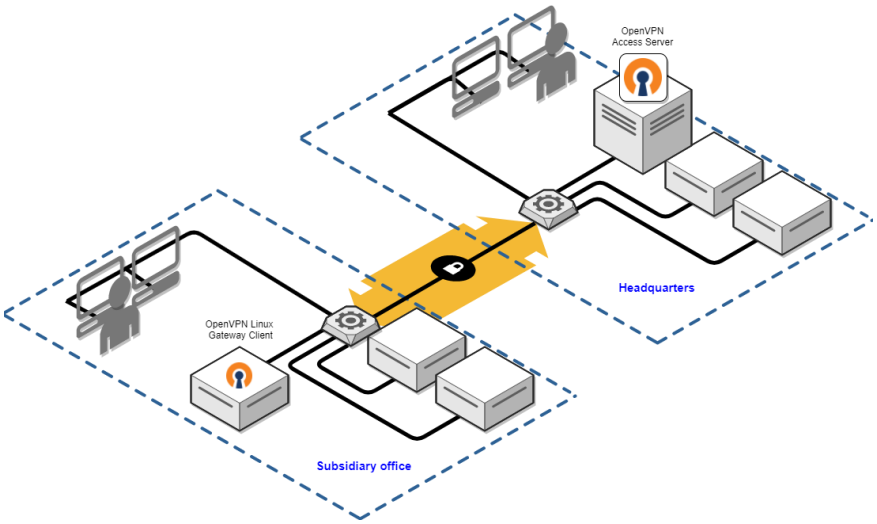
Vraag 5

Een **Remote Access VPN** creëert verbinding tussen individuele gebruikers en een extern netwerk, meestal het interne netwerk van het bedrijf zelf. Dit soort VPN's gebruiken twee componenten:

1. **Network Access Server (NAS):** Een dedicated server of softwaretoepassing op een gedeelde server die is aangesloten op het interne netwerk van het bedrijf.
2. **VPN-client:** software geïnstalleerd op de computer of mobiele telefoon van een gebruiker.



Wanneer de gebruiker toegang wil tot het bedrijfsnetwerk, activeert hij of zij zijn VPN-client, die een geëncrypteerde "tunnel" naar de NAS tot stand brengt. Deze geëncrypteerde tunnel geeft de gebruiker toegang tot het interne netwerk zonder dat zijn verkeer wordt blootgesteld.



Een **Site-to-Site VPN** creëert een virtueel netwerk dat wordt gedeeld over verschillende kantoorlocaties, die elk hun individuele gebruikers hebben. De VPN-client wordt gehost op het lokale netwerk van elk kantoor, in plaats van op de toestellen van de gebruikers. Als ze het kantoor verlaten, verliezen ze ook toegang tot de VPN-client.

PART 3: Conclusie

Een VPN zorgt voor beveiligde verbinding met het internet. Het zorgt ervoor dat al het verkeer via een geëncrypteerde tunnel wordt geleid, dit zorgt ervoor dat je eigen IP-adres verborgen wordt, dat wil zeggen dat niemand zicht heeft op je locatie via je IP-adres. VPN's helpen ook om attacks van buitenaf te blokkeren, dit komt omdat er maar één iemand toegang heeft tot je gegevens en dat ben jij.

12. Hashes!

Opdracht gemaakt door: Tomas

Algoritme	Year	Secure	Output size	Rounds	Toepassingen
Whirlpool	2000	Yes	128	4	Security
SHA-2	2001	Yes	128	80	Beter tegen collisions
SHA-3	2015	Yes	128	12	Beter tegen collisions
MD4	1990	No	32	3	Integrity
MD5	1991	No	32	64	Integrity
HMAC	1996	Yes	32	/	authentication
Bcrypt	1999	Yes	60	Variable	Password security
Scrypt	2009	Yes	128	Variable	Cryptocurrency proof-of-work

Whirlpool: is een redelijk lange hash van 128, met meestal 4 rounds. Deze wordt meestal gebruikt voor de security van een file, en is ontworpen in 2000, deze is ook relatief secure nog deze dagen

SHA-2: is een lange hash van 128, maar met wel 80 rounds, wat het wel secure maakt, deze is ook beter bestendig tegen collisions

SHA-3: is een even lange hash als deze hierboven, maar wel maar met 12 rounds. Deze is ook best gebruikt om collisions tegen te gaan.

MD4: is heel kort, maar 32 karakters. Met maar 3 rounds. Dat maakt deze hash totaal onbruikbaar in de security kant van hashing, dus deze wordt meestal gebruikt voor de integrity van files te garanderen.

MD5: is ook heel kort, even lang als MD5. Maar met meer rounds, 64. Maar deze hash is ook volledig onbruikbaar in de security van vandaag. Dus deze wordt, net zoals MD4, meestal gebruikt voor de integrity van files.

HMAC: is nog steeds een korte hash. Met een variabel aantal rounds. Deze is wel secure, en wordt gebruikt voor de authenticatie op een bedrijf.

Bcrypt: is 60 karakters lang, en heeft een variabel aantal rounds, dit hangt af van de kosten dat hieraan besteed worden. Deze is zeer secure en wordt gebruikt voor password security bijvoorbeeld.

Scrypt: is dubbel zo lang als Bcrypt, wel 128 karakters lang, en heeft ook een variabel aantal rounds, afhankelijk van de kosten. Deze is ook zeer secure, meer secure dan Bcrypt, en wordt vaak gebruikt voor cryptocurrencies, om hun proof-of-work aan te tonen.

Salting: bij salting gaan we een unieke string van 32 karakters of langer toevoegen aan de passwoorden bijvoorbeeld, en ze daarna te gaan hashen, dit zorgt ervoor dat hackers de passwoorden bijna onmogelijk kan gaan reverse engineeren.

1. De hash wordt gewoon veranderd van waarde, de lengte of het aantal rounds veranderd niet, alleen de waarde van de Hash veranderd.
2. Dit maakt het bijna onmogelijk om de Hash te reverse engineeren. Behalve als we bijvoorbeeld toegang hebben tot de salt die gebruikt is
3. Een pepper is veel meer geheim, aangezien de pepper zelf geheim wordt gehouden en ergens anders wordt opgeslaan of totaal niet wordt opgeslaan. In vergelijking met de salt die meestal zelfs bij de Hash wordt opgeslaan.

Bestand

Whirlpool:

19fa61d75522a4669b44e39c1d2e1726c530232130d407f89afee0964997f7a73e83be698b288febcf8
8e3e03c4f0757ea8964e59b63d93708b138cc42a66eb3

SHA-2:

cf83e1357eefb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36ce9ce47d0d13c5d85f2b0ff
8318d2877eec2f63b931bd47417a81a538327af927da3e

SHA-3:

a69f73cca23a9ac5c8b567dc185a756e97c982164fe25859e0d1dcc1475c80a615b2123af1f5f94c11e
3e9402c3ac558f500199d95b6d3e301758586281dcd26

MD4: 31d6cfe0d16ae931b73c59d7e0c089c0

MD5: d41d8cd98f00b204e9800998ecf8427e

HMAC: 6f6e031223b36cd2a997787a03d16bf5

Bcrypt:

$2a$10$VHcAbR4L.iDM.2aUqbVKeetCcLuXhvPY1Hy.fOXCoInBXxDovaXaC

Scrypt:

d72c87d0f077c7766f2985dfab30e8955c373a13a1e93d315203939f542ff86e73ee37c31f4c4b571f47
19fa8e3589f12db8dcb57ea9f56764bb7d58f64cf705

Wachtwoord: 25*treurwilg1992

Whirlpool:

7c459af5d0dc1638c8a3a003f7902a7eee5383fa77125c5c25b70c415ca6ef533573e10f92a127670cec1544e1ee4416c60d425a1a59a711f2f7676694e56dcb

SHA-2:

eca382abc2e2d2d3500063a27d3f9adf493e1db7b868a3b038ea10b77610be41872616196f906fb24cd1466316cdd3fdbcadd4ef99cb98d354e5dd583316b9d0

SHA-3:

be8ccf08235406ea35b02112d00af4599394515f93cca150dce775ad1b0892210ac8487709367f77a42151a8debdfbc2fb51e612e1aa32696a856477d1bfed01

MD4:

3bbf964d5a01426238b57f4caa64edc2

MD5:

bc52e86b176fa3c8215a150c4bdd0699

HMAC:

04660b20a9dbc114df337bb9190d7417

Bcrypt:

$2a$10$PdPg/A6pMtq6nSYZHfWy5uXLxNY8MTw6g/tln3MmAVI47BSKcbfEi

Scrypt:

9984eb388bb82e31d6e0bc10d3664990ddf00f0be9ea027f001fb0605b06b17803560d387d5ae1e7f42fa84f5c05dab37b87808934378d8425d767ba9d5f243c

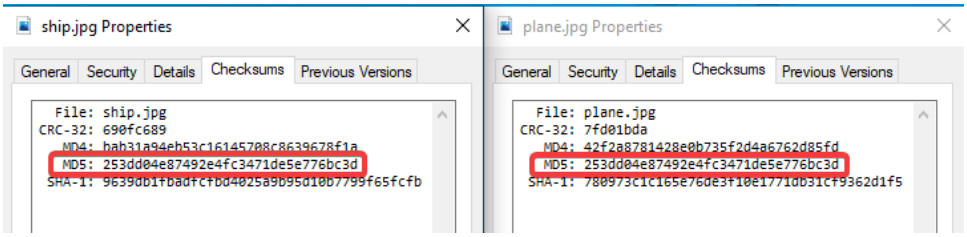
Conclusie

Elk algoritme wordt wel gebruikt voor andere doeleinden, sommige zijn minder secure dan de andere, maar deze kunnen nog steeds gebruikt worden in desdagelijkse toepassingen, bijvoorbeeld MD5 heeft nog steeds een functie als integriteit check, en Scrypt is dan weer veiliger op vlak van cryptocurrencies. Elke algoritme is uniek. En heeft wel een toepassing.

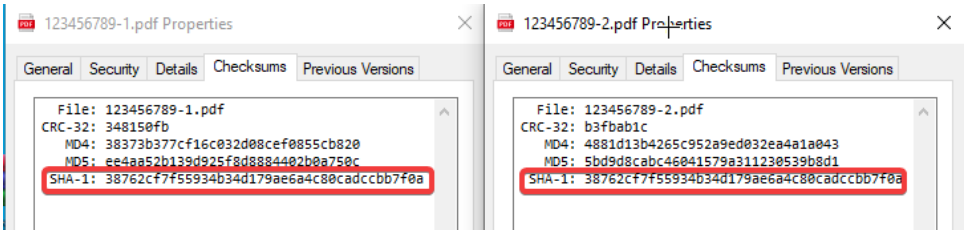
13. Collisions and Hashes

Opdracht gemaakt door: Tomas

1. Bij beide afbeeldingen is de MD5 checksum hetzelfde, dus deze hebben een collision.



2. Bij beide pdf-files is de SHA-1 checksum hash hetzelfde, dus deze hebben ook een collision.



3. Een hash collision of ook wel clash genoemd, is wanneer 2 verschillende stukken data, na een afleiding door een hash functie zelf, toch met dezelfde bits als uitkomst krijgen.

4. We kunnen deze vermijden door bijvoorbeeld onze hash functies random te maken, de chaining techniek, waarbij de hash table eigenlijk een array is van verschillende lijsten. En dan nog uniform hashing

5. We kunnen de functie manipuleren door het middelste stuk van een file te veranderen. Met bedenkingen van de limieten van de hash algoritmes.

14. THM - ‘Crack the hash”

Opdracht gemaakt door: Rasmus

In deze twee opgaves moest men hashes cracken, met behulp van verschillende tools. Bij opgave 1, Level1, waren hashes 1, 2, 3, en 5 crack-able met behulp van een online cracking tool, namelijk:

[CrackStation - Online Password Hash Cracking - MD5, SHA1, Linux, Rainbow Tables, etc.]

Ook de eerste twee hashes van Level2 waren crack-able met behulp van het bovengenoemde tool.

In de afbeeldingen is het zichtbaar wat de resultaten waren:

Hash	Type	Result
48bb6e862e54f2a795ffc4e541caed4d	md5	easy
CBFDAC6008F9CAB4083784CBD1874F76618D2A97	sha1	password123
1C8BFE8F801D79745C4631D09FFF36C82AA37FC4CCE4FC946683D7B336B63032	sha256	letmein
279412f945939ba78ce0758d3fd83daa	md4	Eternity22
F09EDCB1FCEFC6DFB23DC3505A882655FF77375ED8AA2D1C13F640FCCC2D0C85	sha256	paule
1DFECA0C002AE40B8619ECF94819CC1B	NTLM	n63umy8lkf4i

Answer the questions below

48bb6e862e54f2a795ffc4e541caed4d

easy — Correct Answer — Hint

CBFDAC6008F9CAB4083784CBD1874F76618D2A97

password123 — Correct Answer — Hint

1C8BFE8F801D79745C4631D09FFF36C82AA37FC4CCE4FC946683D7B336B63032

letmein — Correct Answer — Hint

$2y$12$Dwt1BZj6pcyc3Dy1FWZ5ieeUznr71EeNkJkUlypTsgbX1H68wsRom

bleh — Correct Answer — Hint

279412f945939ba78ce0758d3fd83daa

Eternity22 — Correct Answer — Hint

Bij de vierde hash moest men gebruik maken van wat meer gevorderde tools. Nadat men hashcat heeft ginstalleerd op de Linux variant die gebruikt wordt, en gecheckt heeft welke ‘hashname’ en ‘hashmode’ gebruikt werd, met behulp van: [example_hashes [hashcat wiki]], dan was het duidelijk, dat de hashname van deze bepaalde hash ‘*bcrypt $2*$, Blowfish (Unix)*’ was, en de hashmode 3200. Met die informatie kan men hashcat runnen, nadat de *rockyou.txt* file, die bij Level2 vermeld staat en downloadable is, toegankelijk is in de Linux variant. De hash in kwestie wordt geplaatst in een txt file, en daarna: **`hashcat -m 3200 hash.txt rockyou.txt --force`**. CD = student/Downloads, flag -m = op hashtype (3200 in dit geval), --force om de ‘Intel OpenCL’ error te bypassen. Na 6 uur werd het password gecracked:

```
$2y$12$Dwt1BZj6pcyc3Dy1FWZ5ieeUznr71EeNkJkUlypTsgbX1H68wsRom:bleh

Session..........: hashcat
Status...........: Cracked
Hash.Type........: bcrypt $2*$, Blowfish (Unix)
Hash.Target......: $2y$12$Dwt1BZj6pcyc3Dy1FWZ5ieeUznr71EeNkJkUlypTsgbX...8wsRom
Time.Started.....: Thu Mar  3 22:07:05 2022 (6 hours, 10 mins)
Time.Estimated...: Fri Mar  4 04:17:59 2022 (0 secs)
Guess.Base.......: File (rockyou.txt)
```

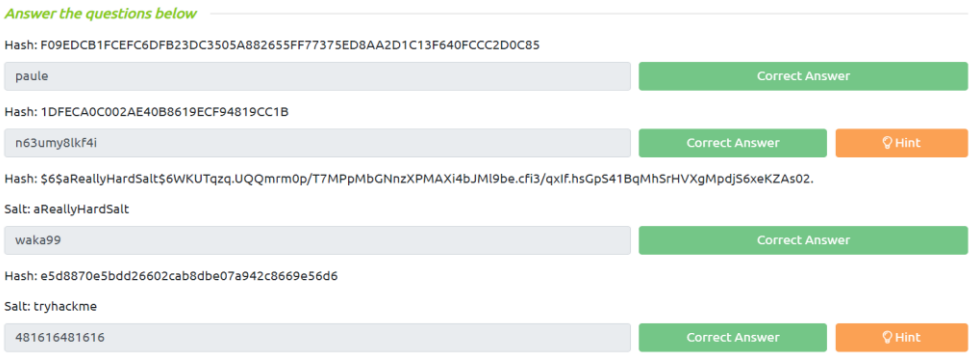
Voor de derde hash van Level2 was het opnieuw checken voor de hashmode en hashtype in de hashcat wikilist. Omdat de hash begint met *6,* was het mogelijk om de hashtype en mode als '*sha512crypt 6, SHA512 (Unix)*'te identificeren, met hashmode *1800*. Nadat de juiste hash (6...) in de hash.txt geplaatst wordt, en de oude verwijderd wordt, is het mogelijk *hashcat -m 1800 hash.txt rockyou.txt –force* te gebruiken, en na 1 uur was die ook gecracked:

```
$6$aReallyHardSalt$6WKUTqzq.UQQmrm0p/T7MPpMbGNnzXPMAXi4bJMl9be.cfi3/qxIf.hsGpS41BqMhSrHVXgMpdjS6xeKZAs02.:waka99

Session..........: hashcat
Status...........: Cracked
Hash.Type........: sha512crypt $6$, SHA512 (Unix)
Hash.Target......: $6$aReallyHardSalt$6WKUTqzq.UQQmrm0p/T7MPpMbGNnzXPM...ZAs02.
Time.Started.....: Fri Mar  4 08:35:45 2022 (1 hour, 2 mins)
Time.Estimated...: Fri Mar  4 09:38:33 2022 (0 secs)
Guess.Base.......: File (rockyou.txt)
```

Voor de laatste hash van level twee moest men eerst de hashtype identificeren, omdat dat niet met behulp van de hashcat wikilist mogelijk was, heb ik hiervoor Hash Analyzer - TunnelsUP gebruikt. Hieruit bleek dat de hashtype SHA1 of SHA128 was. In de hint van de vraag wordt de informatie HMAC-SHA1 gegeven, wat ons een tip geeft: HMAC is gebouwd over hash-functies en kan worden beschouwd als een "gesleutelde hash" - een hash-functie met een sleutel. Een sleutel is geen salt, en dus is de salt die gegeven wordt, eigenlijk een sleutel. Met die informatie kan men de hash en de salt in de hash.txt file plaatsen als hash:salt, en doet dat *hashcat -m 160 hash.txt rockyou.txt --force*. De hash was snel gecracked:

Hash:	e5d8870e5bdd26602cab8dbe07a942c8669e56d6
Salt:	Not Found
Hash type:	SHA1 (or SHA 128)
Bit length:	160
Character length:	40
Character type:	hexidecimal

```
e5d8870e5bdd26602cab8dbe07a942c8669e56d6:tryhackme:481616481616

Session..........: hashcat
Status...........: Cracked
Hash.Type........: HMAC-SHA1 (key = $salt)
Hash.Target......: e5d8870e5bdd26602cab8dbe07a942c8669e56d6:tryhackme
Time.Started.....: Fri Mar  4 09:42:33 2022 (7 secs)
Time.Estimated...: Fri Mar  4 09:42:40 2022 (0 secs)
Guess.Base.......: File (rockyou.txt)
```



Conclusie: in 'Crack The Hash' THM kwam veel research aan bod, maar door het het uiteindelijk voltooien van de room leert de gebruiker veel over de tool *hashcat*. De meeste van de hashes waren 'crack-able' met behulp van een online cracking tool, maar voor de hashes die niet crack-able waren door de online tool, bleek hashcat een krachtig alternatief te zijn. Met behulp van de hashcat wikilist en een online hash identifier, was het mogelijk om de resterdende hashes te identificeren, en daarna met hashcat te cracken. Afhankelijk van de hashmode en hashtype heeft dit lang geduurt. Als men weet, dat een bepaalde hash bijv. HMAC-SHA1 is, zoals in dit geval, dan kan men het gegeven 'salt' eerder als een key beschouwen, die het crack-proces veel sneller maakt. Het is dus echter van belang om de hashtypes en modes goed te kennen.

15. Hashing Challenge

Opdracht gemaakt door: Rasmus

15.1 Easy to Crack

Werkmethode: Door gebruik te maken van **`hashcat -m 0 hash.txt rockyou.txt -force`** was het heel makkelijk om de eerste vier hashes te kraken. De laatste hash is van type SHA-256, en heeft dus hashmode 1400. Met het commando **`hashcat -m 1400 hash.txt rockyou.txt`** was deze ook heel snel gekraakt:

Dit is het overzicht aan hashes:

HASH	PASS
5f4dcc3b5aa765d61d8327deb882cf99	password
81dc9bdb52d04dc20036dbd8313ed055	1234
d8578edf8458ce06fbc5bb76a58c5ca4	qwerty
098f6bcd4621d373cade4e832627b4f6	test
03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4	1234

Bijhorende vragen:

1. De eerste vier hashes zijn van type MD5/MD4 (hashmode 0), en de laatste hash is van type shah-256, en heeft hashmode 1400.
2. Ik heb de hashes geanalyseerd aan de hand van Hash Analyzer - TunnelsUP en heb daarna de hashcat wikilist van example hashes [hashcat wiki] gebruikt om de hashmode te vinden.
3. Deze hashes zijn makkelijk om te kraken omdat de passwords in kwestie de meest voorkomende passwords zijn, en omdat MD5 vergelijkingen makkelijk kunnen worden gedaan met rockyou entries.
4. a. Ouput is niet hetzelfde, de hash kon niet met dezelfde methode worden gekraakt.
   b. Nee, omdat de MD5 checksum de encryptie/decryptie kompleet veranderd.

15.2 Rainbow Table

```powershell
#Work method:
#Contents of hashes.txt = the hashes we are trying to crack
#Contents of md5woordenlijst = wordlist with hashes
#Contents of New-Item matches.txt = all matches found
#Iterate through hashes.txt and add all hashes to an array
#Iterate through woordenlijst.txt and add these to second array
#Nested for loop to check for matches in both arrays
#Transfer matches to matches.txt


$matches = New-Item -ItemType File -Name "matches.txt" `
-Path C:\Users\Rasmus\Documents\Jaar_1\Security_Essentials

[string[]]$wordlistArray = Get-Content `
-Path C:\Users\Rasmus\Documents\Jaar_1\Security_Essentials\md5woordenlijst.txt

[string[]]$hashesArray = Get-Content `
-Path C:\Users\Rasmus\Documents\Jaar_1\Security_Essentials\hashes.txt

foreach ($string in $hashesArray) {
    foreach ($hash in $wordlistArray) {
        if ($hash.Contains($string))
        {
            Add-Content `
            -Path C:\Users\Rasmus\Documents\Jaar_1\Security_Essentials\matches.txt $hash
        }
    }
}
```

De output van het script in een nieuwe file genaamd matches.txt:

```
d8ac13f95359d2a45256d312676193b3  "paperclip"
54b1c483a93051f1719ad0572b3d414e  "1 aprilgrap"
65f0480470eec8d12d8629b5dcad2f72  "een"
1ea63d4a03928f95d7028a4dd4fa9426  "sterk"
701f33b8d1366cde9cb3822256a62c01  "wachtwoord"
a2a551a6458a8de22446cc76d639a9e9  "is"
b95351311be5bb96359e6568afda299c  "belangrijk"
```

15.3 Herken de Hash

Door middel van [Hash Analyzer - TunnelsUP]

HASH	TYPE
e91e6348157868de9dd8b25c81aebfb9	MD4/MD5
5d2d3ceb7abe552344276d47d36a8175b7aeb250a9bf0bf00e850cd23ecf2e43	SHA2-256
56db393b9c6aae92406a9a89ee6a79fd4e	MD5
$2a$10$If3huPOIxrpjWySZ9eoToeCUr8yLQHuGMVhjH2fZ8aO1LgsrnUt6i	Bcrypt

15.4 Hashcat

Step 1:

Voor al deze hashes geldt dezelfde hashmode: 0, en alle hashes zijn van het type MD5/MD4. De command om elk van deze passwords te cracken is dus: **`hashcat -m 0 hash.txt rockyou.txt --force`**, en de crack-tijd bedraagt ongeveer 1 seconde.

Dit zijn de passes:

HASH	PASS
cfc723a39d137f8428d8eeec9cebd384	fifi123
204d7916206f2797d71ab26f523d931f	magicplanet
e75bbc3294ceffd90fd6febd13769b61	magic2004
6707efa899fa779af0f165cfe23b8569	23jupiter

Step 2:
Deze hash is van het type SHA2-256, maar de pass bestaat niet binnen de rockyou.txt file. Wel binnen de wordlistNL.txt. Het gebruikte commando: **`hashcat -m 0 hash.txt wordlistNL.txt --force`**

HASH	TYPE	PASS	CRACK-TIME
1e5fcfc4e60fcad03f92260202359caf0a7b8d8d0955b7b0020f158150b5305f	SHA2-256	vannillestokje	< 1 seconde

Step 3: Rule based hashes die voorkomen in NLwordlist

Voor deze stap heb ik gebruik gemaakt van een ruleset van GitHub: https://github.com/clem9669/hashcat-rule. De rules die een match waren tijdens de uitvoering van hashcat heb ik in een nieuwe file gestopt genaamd matched.rule, met het volgende commando:

```
hashcat -m 0 hash.txt wordlistNL.txt -r rules --debug-mode=1 --debug-file=matched.rule --force
```

Dit was de uitkomst:

HASH	PASS	MATCHED RULES
7f86d1368c2aa6e9bfe21654cf75351e	Appelboom1	$1
40267b17d04ff2344344635be1ecceee	appelboom1!	$1 $!
aabf01f3220280000e830d7f14c84cee	@ppelboom1	$1, 7 -0
30f16456b85fab3a016e07dc2f1a2fa2	@ppelb00m	so0 sa@
4951f0eed8d251980ef8497ed9ce2bf7	@ppelboom123	-0$1 $2 $3

15.5 Advanced Cracking

Ik zal een korte table tonen van de hashes inclusief de wachtwoorden die ik kon kraken. Daarna zal ik de werkwijze uitleggen:

HASH	PASS
93ac53bbc9470bd1f934b0c208f69090	3mei1961
e81c602454bbe574f7802456e2297495	Ernest
3f2460dea519c0ff07835805fd084ac9	ErnestPXL
4e6df07e9c56569e614163994c82ce27	PXL

Level 2: met password policy

HASH	PASS	MATCHED RULES
a616c2b8838d7e09d74704df47029356	@rtemis123	-0$1 $2 $3
9c63e2e15ed5ff7e19d0039bdc77f1df		
5e88214fba455805992b5c3fdff4d39f		
3a524555cba2dcf356580fbe975c4e4d		

Werkwijze:

Ik heb 2 verschillende tools gebruikt om een poging te doen aan deze hashes. Het eerste was het maken van een basale woordenlijst met de informatie van Labo 1 Opdracht 19 OSINT Lodrimed. Deze woordenlijst (ook te zien in de bijlage aan het einde) bevatte 84 woorden aan informatie over Ernest, Lodrimed, en alle andere bijhorende informatie. Aan de hand van een tool genaamd 'mentalist' was het mogelijk om die lijst uit te breiden. Mentalist is een hulpmiddel voor het genereren van custom woordenlijsten. Het maakt gebruik van gemeenschappelijke menselijke paradigma's voor het samenstellen van wachtwoorden en kan de volledige woordenlijst weergeven, evenals rules die compatibel zijn met Hashcat en John the Ripper. Mentalist voegt extra combinaties aan wordlistentries toe aan de reeds bestaande wordlist met substitutie, appending, prepending, lowercase combinaties, uppercase combinaties, en nog andere mogelijkheden. Op die manier kan men heel snel een woordenlijst van 84 woorden tot een woordenlijst van 35 984 246 woorden uitbreiden. Uiteindelijk, als men een rule based attack doet, kan men aanhand van een 'small-rule' file (de meest voorkomende rules), de lijst nog eens uitbreiden tot 11 995 822 300 entries (11 miljard!).

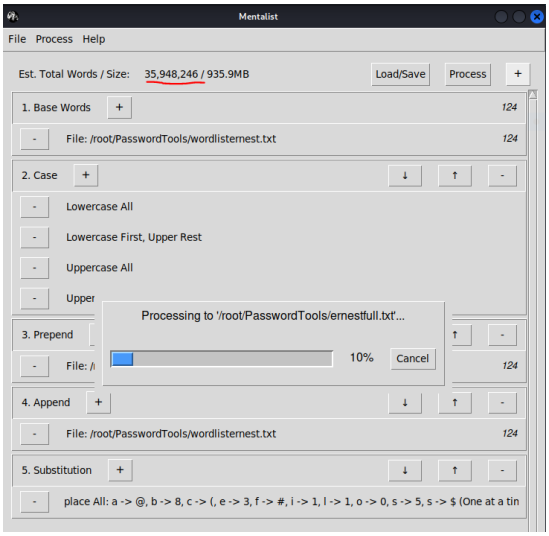
Vanuit verschillende originele entries zoals 'Ernest', 'Vanderbouwen', of 'Lodrimed', zou uiteindelijk ook een gecombineerde entry voor '~3rn3$[L0dr!m3dERNESTVANDERBAUWEN@L0drimed1' getest worden. Aan de hand van deze mega-uitgebreide lijst heb ik meerdere keren, met verschillende rulesets, en verschillende start-wordlist entries geprobeerd om de hashes te kraken. Helaas kon ik drie van de passwords niet decrypteren.

De link naar de tool (mentalist) die ik hiervoor heb gebruikt is:

https://github.com/sc0tfree/mentalist, en aan de hand van een goede youtube tutorial kon ik gebruik maken van deze tool: Create Custom Wordlists with the Mentalist for Brute-Forcing [Tutorial] - YouTube .

Helaas mist mijn initiële woordenlijst nog steeds de cruciale entries die drie van deze passwords benodigd, en daardoor helpt een woordenlijst van 11 miljard entries helaas ook niet. Uiteindelijk heb ik na meerdere pogingen geprobeerd om met John The Ripper de passwords te brute-forcen, ook zonder succes.

```
Session..........: hashcat
Status...........: Exhausted
Hash.Mode........: 0 (MD5)
Hash.Target......: 93ac53bbc9470bd1f934b0c208f69090
Time.Started.....: Fri Mar 18 07:50:05 2022, (16 mins, 36 secs
Time.Estimated...: Fri Mar 18 08:06:41 2022, (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (ernestfull.txt)
Guess.Mod........: Rules (smallrules.rule)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........: 12488.0 kH/s (7.82ms) @ Accel:256 Loops:256
Recovered........: 0/1 (0.00%) Digests
Progress.........: 11995822300/11995822300 (100.00%)
Rejected.........: 0/11995822300 (0.00%)
Restore.Point....: 31157980/31157980 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:256-385 Iteration:0-256
Candidate.Engine.: Device Generator
Candidates.#1....: ~Lodrim3d →
Hardware.Mon.#1..: Util: 98%
```



Conclusie:

1. Is een hash nog steeds een 'one-way-functie'?

Omdat het bij deze hashes om MD5/MD4 hashes gaat, en hashcat alleen een vergelijking maakt met de md5 hashversies van alle entries in de diverse woordenlijsten (rockyou, woordenlijstNL, ernestextended.txt), zijn de hashes nog steeds een one-way functie. Hashcat vergelijkt alleen hashes met elkaar, en doet verder geen decryptie.

2. Bestaan er hashes die niet te kraken zijn?

Technisch gezien zou je iedere hash kunnen kraken, als je quantum-computers ter beschikking zou hebben of duizenden jaren aan tijd. Als we realistisch erover zouden nadenken, met de beperkingen van de huidige 'persoonlijke' technologie, dan bestaan er veel hashes die niet te kraken zijn. In het geval van oefening 15.5 Advanced Cracking, waren drie van de passwords niet crackable, omdat de juiste woorden, of combinatie van woorden niet aanwezig waren in de woordenlijst. Afhankelijk van de lengte van de woordenlijst, en grootte van de rule-set, worden de kansen groter om een hash te kraken, maar er bestaan zeker nog hashes die niet te kraken zijn.

3. Hoe ga je hiermee om m.b.t. paswoorden opslaan?

Met betrekking tot het opslaan van wachtwoorden is MD5 echter wel een afrader. Encrypties die moeilijker zijn te kraken zoals SHA-512 of Triple-DES zou je eerder wachtwoorden moeten opslaan in een database.

4. Waaraan moet een wachtwoord voldoen, reken houden met deze oefening?

Aanhand van deze oefening werd het ook duidelijk, dat het aanmaken van wachtwoorden die opeenvolgende chars hebben, zoals '123', of symbolen als '!@#' altijd een slecht idee is, omdat ze vaak voorkomen in woordenlijsten. Verder kan de password beter gescrabbeld zijn, om de kans te minimeren dat het wachtwoord voorkomt in de woordenlijst.

5. Past de eerder omschreven password policy nog steeds?

Aanhand van de eerdere uitleg voldoet de standard password policy natuurlijk nog steeds, alleen zouden herkenbare woorden vermeden moeten worden, omdat dat de kans vergroot dat het gekozen woord ook zou voorkomen in de diverse woordenlijsten.

6. Wat kan je doen om rainbow table attacks tegen te gaan?

Rainbow-Table Attacks kunnen eenvoudig worden voorkomen door salting te gebruiken, wat willekeurige gegevens zijn die samen met de plain tekst aan de hash-functie worden doorgegeven. Dit zorgt ervoor dat elk wachtwoord een uniek gegenereerde hash heeft en dus wordt een rainbow-table attack, wat werkt volgens het principe dat meer dan één tekst dezelfde hash-waarde kan hebben, voorkomen.

16. Beveiligde Files

Opdracht gemaakt door: Stef

Inhoud van het bestand

De inhoud van het document is: **PXL{Cr4ck_Th3_P4ssw0rd}**

Werkwijze

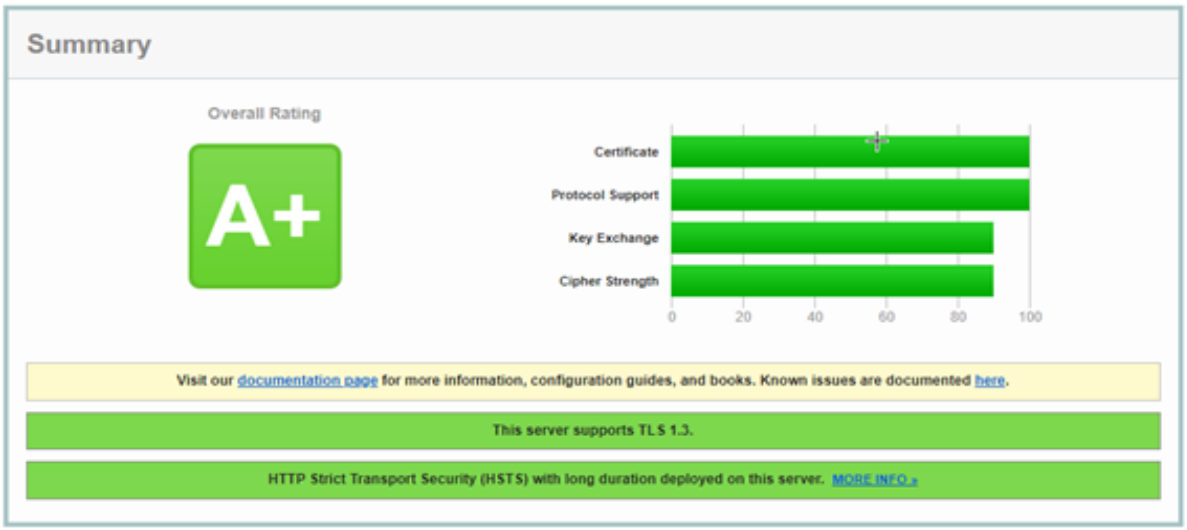
Ik heb online een tool opgezocht die via een dictionary attack het paswoord kan vinden. De werking van deze tool is heel simpel, het document dat je wilt gebruiken in de tool sleep je uit je bestandsmap in een kader die aangeeft waar het document geplaatst moet worden. Hierna neemt de tool het werk over en wordt het paswoord achterhaald. Dit duurde helemaal niet lang omdat het wachtwoord niet complex was. Hierna kon ik het gevonden paswoord ingeven bij het openen van het Worddocument en zo kon ik de inhoud ervan zien. De link naar deze tool staat in de bijlagen.

Conclusie

Het is belangrijk om je goed te houden aan de wachtwoordpolicy, hoofdlettergebruik, speciale tekens en cijfers. Dit kan verholpen worden door een tool die voor jou wachtwoorden genereert, dit maakt het veel moeilijker voor hackers om ze te kraken. Als deze wachtwoorden dan te moeilijk zijn om te onthouden kan je ze opslaan in een passwordmanager.

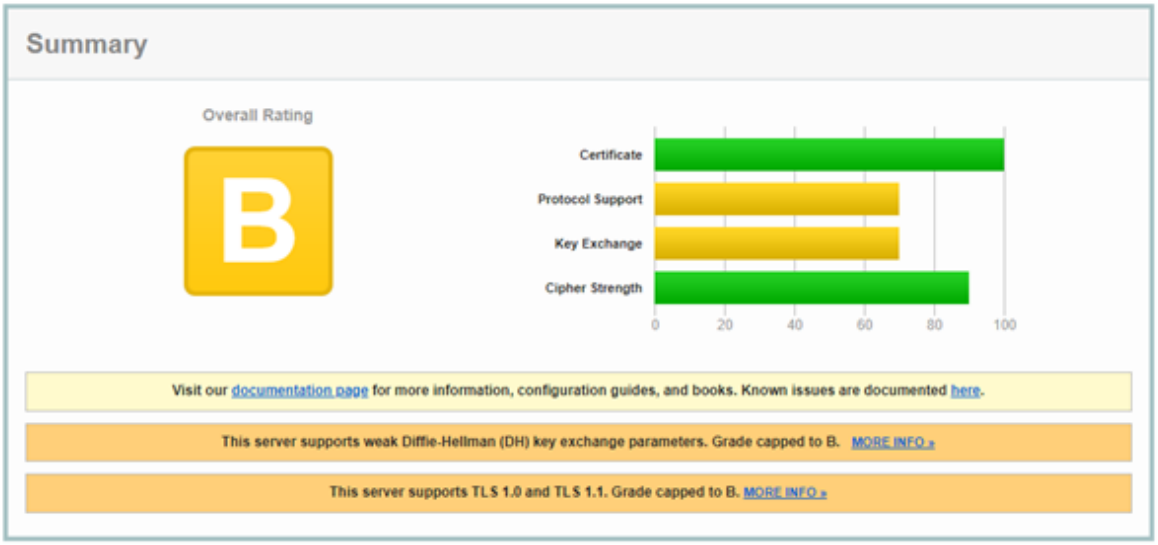
17. SSL Certificates

Opdracht gemaakt door: Tomas

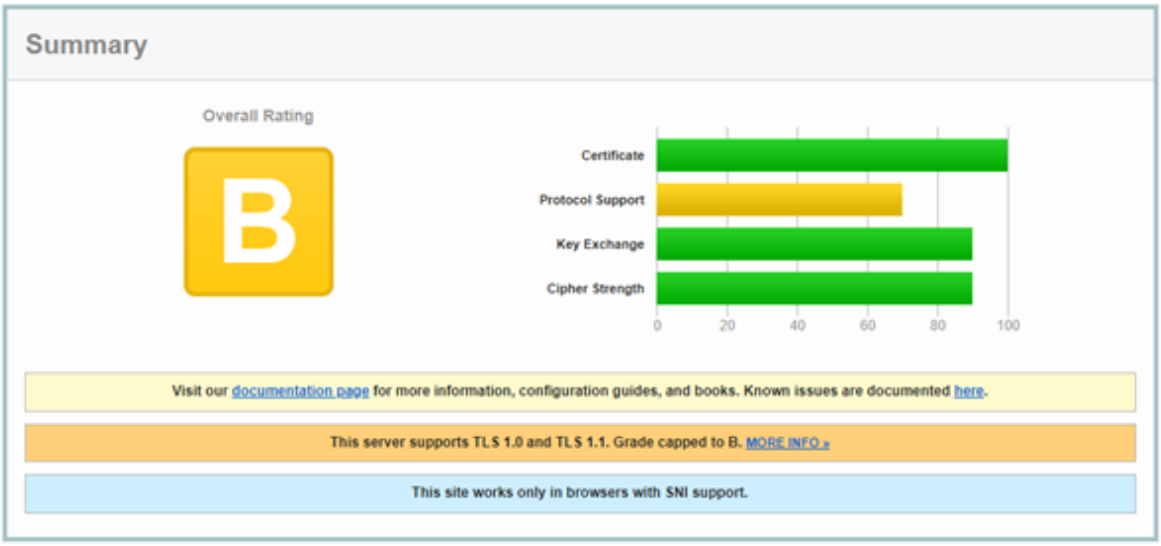


Bij www.ikea.com zijn er zo goed als geen problemen, het enige wat ze zouden kunnen aanpassen is dat ze niet meer available zijn voor oudere versies van safari, want deze hun "handshake" geeft een fatale error.

Een handshake is een actie waar bij de ssl client en server de secret keys gaan genereren om met elkaar te kunnen communiceren.

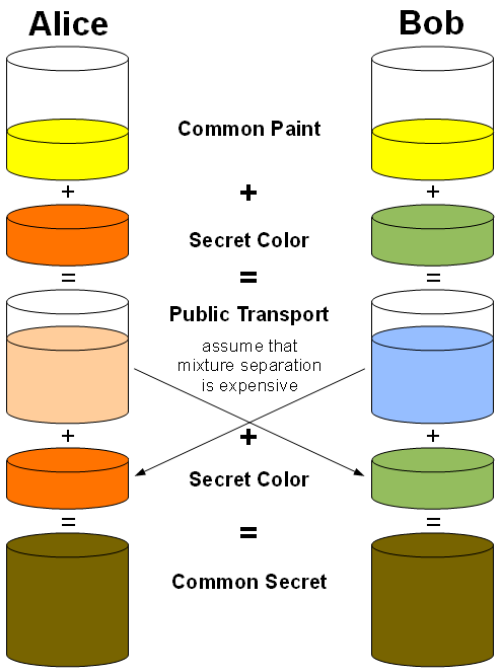


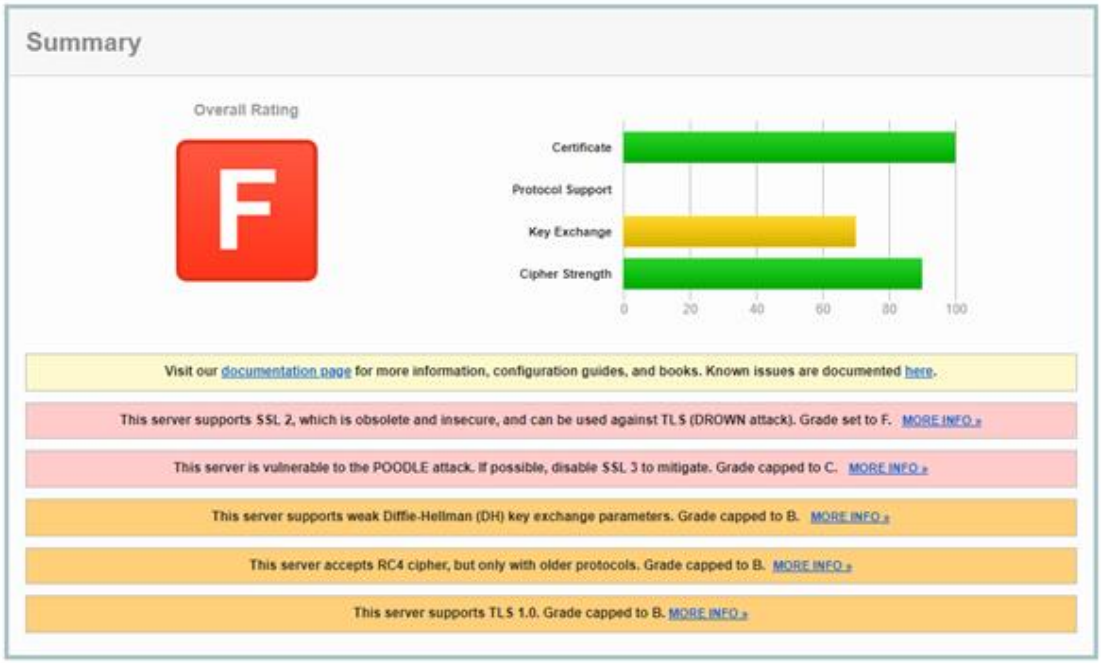
www.pxl.be doet het dan iets minder, zoals beschreven in de beschrijving. De server heeft support voor TLS 1.0 en TLS 1.1, deze zouden ze moeten upgraden naar TLS 1.3 en dan ook nog de zwakke Diffie-Hellman key exchange parameters. Diffie-Hellam zoals eerder gezien, werkt met exponenten en getallen om bepaalde decryptie keys te genereren, dit kunnen we oplossen door de software the reconfiguren zodat deze op z'n minst 1048-bit DH-parameters gebruikt.



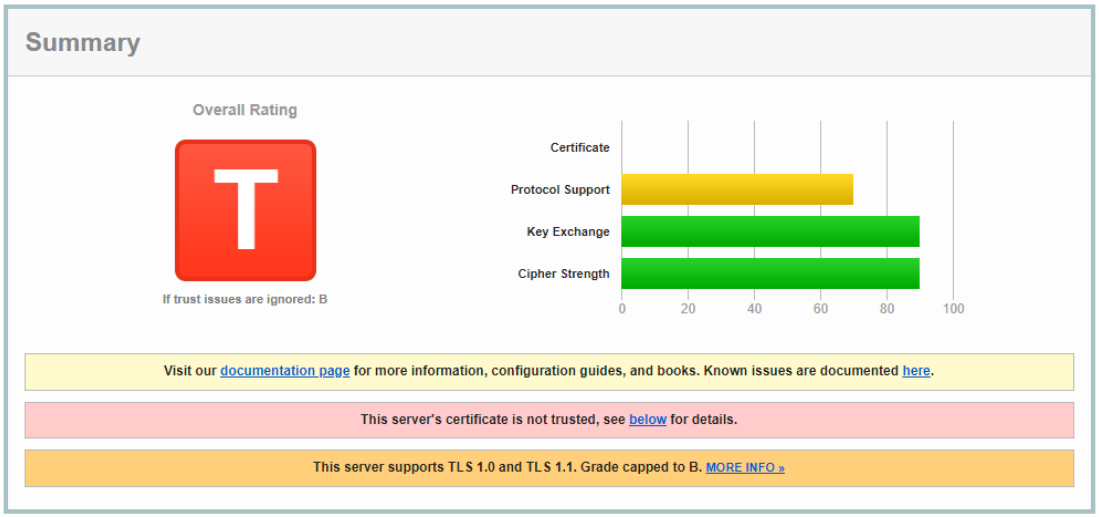
www.howest.be heeft hetzelfde probleem als PXL.be, ze hebben alleen support voor TLS 1.0 en TLS 1.1, dit kan opgelost worden door de support te verhogen naar TLS 1.3 of TLS 1.2. En nog een probleem met deze website, is dat deze alleen werkt op browsers die SNI supporten.

SNI of Server Name Indication, is een uitbreiding van TLS, hierbij gaat de client aanduiden welke hostnaam waarmee hij probeert te verbinden. Hierdoor kan de server het juiste virtuele domein selecteren en het juiste certificaat aan de browser presenteren.





Bij www.njtf.cn is er overduidelijk heel veel ruimte voor verbetering. Bijvoorbeeld, de server heeft alleen support voor SSL 2, wat verouderd en onveilig is. En kan misbruikt worden voor een Drown attack. De handshakes zijn zeer vatbaar voor een 'man-in-the-middle' attack, en deze gebruikt dezelfde key voor authenticatie en encryptie. Dit zouden we kunnen oplossen door bijvoorbeeld de TLS 1.2 aan te zetten. Dan is deze server nog zeer vatbaar voor een POODLE-attack, wat een beveiligingsprobleem is dat misbruik maakt van een fall back op SSL 3.0. Om dit tegen te gaan zouden we SSL 3.0 kunnen uitzetten op de server. Daarom, net zoals www.pxl.be is de Diffie Hellman support zeer zwak. De server heeft alleen support voor TLS 1.0, dus zouden we deze moeten upgraden naar TLS 1.2 of TLS 1.3



Bij www.xjufe.edu.cn hebben ze, zoals hierboven te zien, geen veilige of betrouwbare certificate, dit kan komen omdat ze deze zelf hebben gemaakt of door een onbetrouwbare certificaat supplier. Dus ze zouden hier een echt certificaat kunnen laten maken, of zelf een betrouwbar certificaat maken. En om dan nog een betere server te maken, kunnen we de TLS upgraden van 1.1 of 1.0 naar 1.2 of 1.3.

18. SSL and GitHub

Opdracht gemaakt door: Aleyna

Stap 1:

Allereerst maak je een private repository genaamd “SecEss_2” aan. Vervolgens ga je in Git Bash het onderstaand commando ingeven gevolgd met het e-mailadres waar je GitHub account aan gekoppeld is. Je zult vervolgens 3 vragen krijgen:

```
> Enter a file in which to save the key (/c/Users/you/.ssh/id_algorithm): [Press enter]
> Enter passphrase (empty for no passphrase): [Enter]
> Enter same passphrase again: [Enter]
```

In de opgave wordt gevraagd geen passphrase te gebruiken, dus bij deze 3 vragen klik je gewoon op de entertoets, dit resulteert in het genereren van een SSH key gebruik makend van het e-mailadres als label.

```
A-ley@DESKTOP-1QJ2SJM MINGW64 ~
$ ssh-keygen -t rsa -b 4096 -C "aleyna.arslan@student.pxl.be"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/A-ley/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/A-ley/.ssh/id_rsa
Your public key has been saved in /c/Users/A-ley/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:bc/LfGvxq5V5Ii7ZY1tUeKqxwgpufPasYpJcHVlZS5c aleyna.arslan@student.pxl.be
The key's randomart image is:
+---[RSA 4096]----+
|           oo .. |
|          o. oE. |
|         o  . . o|
|        o.     + |
|       .S.o . o  |
|      . .o o =. o|
|   . +.   oo* o*.|
|    +.=.ooo=+++.o|
|     +.+oooo**oo.|
+----[SHA256]-----+
```

Stap 2:

```
A-ley@DESKTOP-1QJ2SJM MINGW64 ~
$ eval "$(ssh-agent -s)"
Agent pid 1681
```

In deze stap gaan we SSH-agent via de bash terminal starten door volgend commando in te geven. Dit is om achteraf de SSH-key aan de SSH-agent toe te voegen. Dit heeft in ons geval niet veel nut omdat we geen passphrase gebruikt hebben. Moest dat wel het geval zijn, dan dient deze optie voor het herinneren van de passphrase en hoef je die niet telkens opnieuw in te geven.

Stap 3:

Nu kunnen we de SSH-key aan onze SSH-agent toevoegen door het onderstaand commando:

```
A-ley@DESKTOP-1QJ2SJM MINGW64 ~
$ ssh-add /c/Users/A-ley/.ssh/id_rsa
Identity added: /c/Users/A-ley/.ssh/id_rsa (aleyna.arslan@student.pxl.be)
```

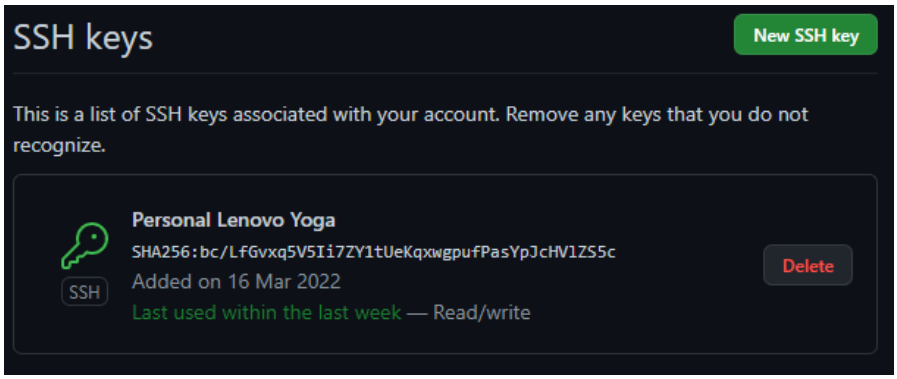
Stap 4:

```
A-ley@DESKTOP-1QJ2SJM MINGW64 ~
$ clip <  /c/Users/A-ley/.ssh/id_rsa.pub
```

Via het clip commando kopiëren we de SSH key om hem vervolgens toe te voegen in GitHub.

Stap 5:

Vervolgens ga je in Settings onder de sectie SSH and GPG keys een nieuwe SSH-key toevoegen, de key die we zojuist hebben gekopieërd plakken we erin en geven we het een naam. In mijn geval noemt het "Personal Lenovo Yoga" omdat het op mijn persoonlijke laptop is toegevoegd.



Stap 6:

```
A-ley@DESKTOP-1QJ2SJM MINGW64 ~
$ git clone git@github.com:AleynaArslanPXL/SecEss_2.git
Cloning into 'SecEss_2'...
The authenticity of host 'github.com (140.82.121.3)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
warning: You appear to have cloned an empty repository.
```

In de laatste stap gaan we in Git Bash via het **git clone** commando de gewenste repository clonen. In de screenshot zie je dat het clonen van de repository via SSH is gelukt.

Conclusie

Wanneer je met een GitHub-repository werkt, moet je jezelf vaak identificeren met gebruikersnaam en wachtwoord. De SSH-key is een manier van identificatie waarbij de gebruiker niet elke keer een gebruikersnaam en wachtwoord moet ingeven. Deze keys komen in paren van 2: **public key** en **private key**. De public key wordt gedeeld met services zoals GitHub en de private key wordt enkel op het systeem van de gebruiker opgeslagen, wanneer deze keys overeenkomen, krijgt de gebruiker toegang. De cryptografie achter SSH-sleutels zorgt ervoor dat niemand uw privésleutel kan reverse-engineeren van de public key.

19. SSL and SSH

Opdracht gemaakt door: Aleyna

Ik heb twee virtuele machines aangemaakt met als naam VM1Groep2 en VM2Groep2, vervolgens heb ik zoals gevraagd de gebruikers toegevoegd.

Op VM2Groep2 heb ik de SSH-server geïnstalleerd met het commando **`sudo apt install openssh-server`**

Dit commando zorgt ervoor dat je vanuit VM1 verbinding kunt maken met VM2 door de gebruikersnaam en het wachtwoord in te geven.

Om verbinding te maken met de tweede VM geef je het commando **`ssh vm2groep2@192.168.58.135`** in, het IP-adres dat is opgegeven is het IP-adres van VM2, dit is belangrijk om mee te geven, mits dat niet het geval is, zal de verbinding niet werken. Zoals je kunt zien zal de verbinding lukken, en kan je het wachtwoord ingeven om de verbinding te vervolledigen.

```
vm1groep2@vm1groep2:~$ ssh vm2groep2@192.168.58.135
vm2groep2@192.168.58.135's password:
Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.13.0-35-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

56 updates can be applied immediately.
41 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Wed Mar 16 14:02:11 2022 from 192.168.58.134
$ exit
Connection to 192.168.58.135 closed.
```

```
vm1groep2@vm1groep2:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vm1groep2/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/vm1groep2/.ssh/id_rsa
Your public key has been saved in /home/vm1groep2/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:Kh6Y5b5a/muxIcw6yHjhZNyqFNvuEW7CGm32e5sAb4c vm1groep2@vm1groep2
The key's randomart image is:
+---[RSA 3072]----+
|                 |
|                 |
|                 |
|                 |
| o.=.   S        |
|..XB*.o.         |
|+X*OE.o+         |
|==XB.=+.         |
|ooo=B*=o         |
+----[SHA256]-----+
```

Nu moeten we verbinding zien te maken zonder het wachtwoord in te geven. Dit doen we door een SSH-key te genereren, dit zal zoals bij oefening 18, ervoor zorgen dat de verbinding zonder wachtwoord gemaakt zal worden. Dit doen we door op VM1 het commando **`ssh-keygen`** mee te geven.

```
vm1groep2@vm1groep2:~$ ssh-copy-id vm2groep2@192.168.58.135
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/vm1groep2/.
ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompt
ed now it is to install the new keys
vm2groep2@192.168.58.135's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'vm2groep2@192.168.58.135'"
and check to make sure that only the key(s) you wanted were added.
```

Nadat we de key hebben gegenereerd moeten we deze kopiëren naar de 2é VM. Dat kan met het commando: **`ssh-copy-id vm2groep2 @192.168.58.135`**, dit commando installeert het SSH-key op de tweede VM als een authorized key.

Laten we nu het **`ssh vm2groep2@192.168.58.135`** commando opnieuw uit voeren op VM1 om te zien of de verbinding zonder het wachtwoord zal lukken.

```
$ ssh vm2groep2@192.168.58.135
Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.13.0-35-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

56 updates can be applied immediately.
41 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Wed Mar 16 14:02:31 2022 from 192.168.58.134
```

Conclusie

OpenSSH is de belangrijkste connectiviteitstool voor inloggen op afstand met het SSH-protocol. SSH is een belangrijk hulpmiddel dat wordt gebruikt om andere computers op afstand te bedienen, het is veilig omdat beide computers berichten kunnen versleutelen en ontsleutelen met identieke symmetrische keys. Het nut van deze SSH-keys zijn om toegang te verlenen zonder dat voor elke login een wachtwoord vereist is. Dit vergemakkelijkt geautomatiseerde, wachtwoordloze logins en single sign-on met behulp van het SSH-protocol.

Bijlage Opdracht 7 – THM Steganography

Foto1

```
student@ubuntu:~/Downloads/spect$ steghide extract -sf jpeg1.jpeg
Enter passphrase:
wrote extracted data to "a.txt".
student@ubuntu:~/Downloads/spect$ ls
a.txt  jpeg1.jpeg  jpeg2.jpeg  jpeg3.jpeg  png1.png  wav1.wav  wav2.wav
student@ubuntu:~/Downloads/spect$ cat a.txt
pinguftw
student@ubuntu:~/Downloads/spect$
```

Foto2

```
student@ubuntu:~/Downloads/spect$ zsteg png1.png
imagedata           .. file: DOS 2.0 backup id file, sequence 48
b1,r,lsb,xy         .. file: dBase III DBT, version number 0, next free block index 3234843654
b1,bgr,lsb,xy       .. text: "nootnoot$"
b2,r,lsb,xy         .. file: MacBinary, Mon Feb  6 07:28:16 2040 INVALID date, modified Mon Feb  6 07:28:16 2040 "PPPUP"
b2,b,msb,xy         .. text: "]UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU"
b4,r,lsb,xy         .. text: "DETUDUDUDUUUDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD"
b4,r,msb,xy         .. text: ["\"" repeated 243 times]
b4,g,lsb,xy         .. text: "\"\"#33223#2#2#3333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333"
b4,b,lsb,xy         .. text: "\"23\"333\"333#\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\""
b4,b,msb,xy         .. text: ["D" repeated 243 times]
b4,rgb,lsb,xy       .. text: "\"B$\"B5\"R43S%2C43S5#C4#S%2B43S5#R53S%#B$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C$2C"
b4,bgr,lsb,xy       .. text: "$\"B$2B%2S4#R53C43S%3C$#R52C43S%2S5#S%\"B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#B4#"
student@ubuntu:~/Downloads/spect$
```

Foto3

```
student@ubuntu:~/Downloads/spect$ exiftool jpeg3.jpeg
ExifTool Version Number         : 11.88
File Name                       : jpeg3.jpeg
Directory                       : .
File Size                       : 8.3 kB
File Modification Date/Time     : 2020:01:06 22:09:44+01:00
File Access Date/Time           : 2020:01:06 22:09:46+01:00
File Inode Change Date/Time     : 2022:03:17 19:20:20+01:00
File Permissions                : rw-r--r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Exif Byte Order                 : Big-endian (Motorola, MM)
Document Name                   : Hello :)
X Resolution                    : 1
Y Resolution                    : 1
Resolution Unit                 : None
Y Cb Cr Positioning             : Centered
Image Width                     : 213
Image Height                    : 160
Encoding Process                : Baseline DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:2:0 (2 2)
Image Size                      : 213x160
Megapixels                      : 0.034
student@ubuntu:~/Downloads/spect$
```

Foto4

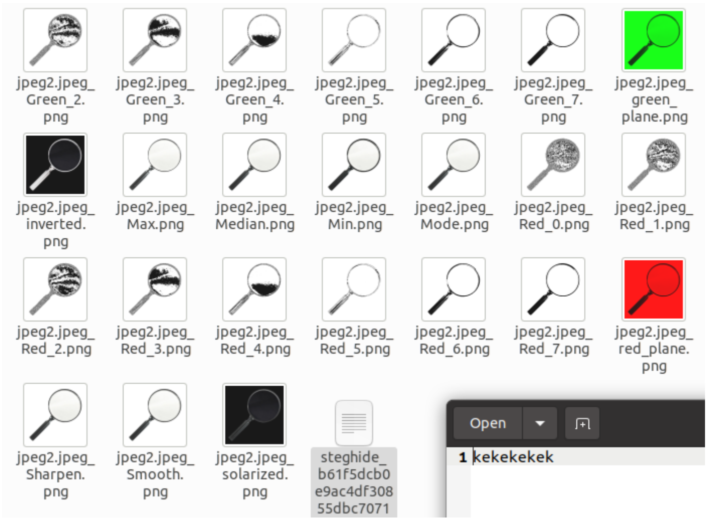


Foto5

Foto 6

```
File Size                       : 8.6 kB
File Modification Date/Time     : 2022:03:03 16:45:58+00:00
File Access Date/Time           : 2022:03:03 16:46:27+00:00
File Inode Change Date/Time     : 2022:03:03 16:45:58+00:00
File Permissions                : rw-r--r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Exif Byte Order                 : Big-endian (Motorola, MM)
Document Name                   : password=admin
X Resolution                    : 1
```

Foto7

```
root@ip-10-10-133-39:~/Downloads# steghide extract -sf exam1.jpeg
Enter passphrase:
the file "a.txt" does already exist. overwrite ? (y/n) y
wrote extracted data to "a.txt".
root@ip-10-10-133-39:~/Downloads#
```

Foto8

Foto9

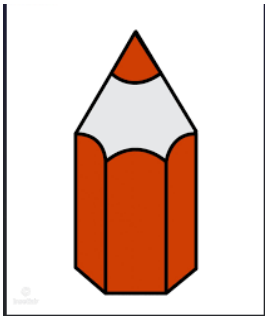


Foto10

```
student@ubuntu:~/Downloads$ zsteg pen.png
imagedata           .. text: ")))xxxLMO"
b1,bgr,lsb,xy       .. text: "\rKey: fatality"
b2,rgb,lsb,xy       .. file: SoftQuad DESC or font file binary
b2,rgb,msb,xy       .. file: VISX image file
b2,bgr,lsb,xy       .. file: SoftQuad DESC or font file binary
b2,bgr,msb,xy       .. file: VISX image file
```

Foto11

Foto12

```
student@ubuntu:~/Downloads$ cd results
student@ubuntu:~/Downloads/results$ ls
exif                    qr.PNG_Blue_4.png                qr.PNG_Find_Edges.png     qr.PNG_Mode.png
keepers                 qr.PNG_Blue_5.png                qr.PNG_GaussianBlur.png   qr.PNG_Red_0.png
qr.PNG_Alpha_0.png      qr.PNG_Blue_6.png                qr.PNG_grayscale.png      qr.PNG_Red_1.png
qr.PNG_Alpha_1.png      qr.PNG_Blue_7.png                qr.PNG_Green_0.png        qr.PNG_Red_2.png
qr.PNG_Alpha_2.png      qr.PNG_blue_plane.png            qr.PNG_Green_1.png        qr.PNG_Red_3.png
qr.PNG_Alpha_3.png      qr.PNG_Edge-enhance_More.png     qr.PNG_Green_2.png        qr.PNG_Red_4.png
qr.PNG_Alpha_4.png      qr.PNG_Edge-enhance.png          qr.PNG_Green_3.png        qr.PNG_Red_5.png
qr.PNG_Alpha_5.png      qr.PNG_enhance_sharpness_-100.png qr.PNG_Green_4.png       qr.PNG_Red_6.png
qr.PNG_Alpha_6.png      qr.PNG_enhance_sharpness_100.png qr.PNG_Green_5.png        qr.PNG_Red_7.png
qr.PNG_Alpha_7.png      qr.PNG_enhance_sharpness_-25.png qr.PNG_Green_6.png        qr.PNG_red_plane.png
qr.PNG_alpha_plane.png  qr.PNG_enhance_sharpness_25.png  qr.PNG_Green_7.png        qr.PNG_Sharpen.png
qr.PNG_Blue_0.png       qr.PNG_enhance_sharpness_-50.png qr.PNG_green_plane.png    qr.PNG_Smooth.png
qr.PNG_Blue_1.png       qr.PNG_enhance_sharpness_50.png  qr.PNG_Max.png
qr.PNG_Blue_2.png       qr.PNG_enhance_sharpness_-75.png qr.PNG_Median.png
qr.PNG_Blue_3.png       qr.PNG_enhance_sharpness_75.png  qr.PNG_Min.png
student@ubuntu:~/Downloads/results$
```

Foto13

Decode Succeeded

Raw text	http://key=killshot
Raw bytes	41 36 87 47 47 03 a2 f2 f6 b6 57 93 d6 b6 96 c6 c7 36 86 f7 40 ec 11 ec 11 ec 11 ec
Barcode format	QR_CODE
Parsed Result Type	URI
Parsed Result	http://key=killshot

Bijlage Opdracht 8 - Python script

```
# De meest frequent voorkomende letters in NL
ENATIR = 'ENATIRODSLGVHKMUBPWJZCFXYQ'
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def getLetterCount(message):
    # Returns a dictionary with keys of single letters and values of the
    # count of how many times they appear in the message parameter:
    letterCount = {'A': 0, 'B': 0, 'C': 0, 'D': 0, 'E': 0, 'F': 0,
                   'G': 0, 'H': 0, 'I': 0, 'J': 0, 'K': 0, 'L': 0, 'M': 0,
                   'N': 0,'O': 0, 'P': 0, 'Q': 0, 'R': 0, 'S': 0, 'T': 0,
                   'U': 0, 'V': 0,'W': 0, 'X': 0, 'Y': 0, 'Z': 0}

    for letter in message.upper():
        if letter in LETTERS:
            letterCount[letter] += 1
    return letterCount


def getItemAtIndexZero(items):
    return items[0]


def nlFreqMatchScore(message):

    freqOrder = getFrequencyOrder(message)
    matchScore = 0

    for commonLetter in ENATIR[:6]:
        if commonLetter in freqOrder[:6]:
            matchScore += 1

    return matchScore


def getFrequencyOrder(message):
    # Returns a string of the alphabet letters arranged in order of most
    # frequently occurring in the message parameter.
    letterToFreq = getLetterCount(message)

    freqToLetter = {}
    for letter in LETTERS:
        if letterToFreq[letter] not in freqToLetter:
            freqToLetter[letterToFreq[letter]] = [letter]
        else:
            freqToLetter[letterToFreq[letter]].append(letter)

    for freq in freqToLetter:
        freqToLetter[freq].sort(key=ENATIR.find, reverse=True)
        freqToLetter[freq] = ''.join(freqToLetter[freq])

    freqPairs = list(freqToLetter.items())
    freqPairs.sort(key=getItemAtIndexZero, reverse=True)

    freqOrder = []
    for freqPair in freqPairs:
        freqOrder.append(freqPair[1])

    return ''.join(freqOrder)
```

```
message = "Qci cu ssg jhhnzssyq jag ssg istui qcs hgyssuzaan cu wsfaati fsi
           zsxoyv jag uozuicioics. Qs ysiisnu cg qsms istui mcrg jsnjagwsg
           qhhn agqsns ysiisnu. Qhhn wsznoct is fatsg jag ssg
           dnsbosgicsiazsy tag rs qsms istui isnow yssuzaan fatsg. Mh tag
           rs qs fssui jhhnthfsgqs ysiisnu cg qsms istui jsnjagwsg qhhn qs
           fssui jhhnthfsgqs ysiisn jhywsgu qs dnsbosgicsiazsy. Qs fssui
           jhhnthfsgqs ysiisn cg hgms iaay cu qs 's'.  Rs may mcsg qai rs
           ugsy ssg yssuzans istui ocithfi. Ssg zcrthfsgqs jnaaw jhhn qsms
           hvqnaexi cu, xhs tag rs qsms jhnf jag sgenpvics uisntsn fatsg?
           Qs dyaw qcs zcr qsms hvqnaexi xhhni cu {Avvsyfhsu}."

print("De meest voorkomende letters in NL op volgorde zijn: ", ENATIR)
print("\nDe meest voorkomende letters in de cipher in volgorde zijn: ",
       getFrequencyOrder(message))
print("\nDe 6 meest voorkomende letters in volgorde, in NL zijn: ",
       ENATIR[:6])
print("\nDe 6 meest voorkomende letters in de cipher zijn: ",
       getFrequencyOrder(message)[:6])
print("\nHet aantal meest voorkomende letters van de cipher "
      "\ndie ook in de eerste 6 letters van ENATIR present zijn: ",
       nlFreqMatchScore(message))
print("\nDe frequentie analyse:\n", getLetterCount(message))

new_string = message.lower()
new_string = new_string.replace('s', 'e')
new_string = new_string.replace('u', 's')
new_string = new_string.replace('o', 'u')
new_string = new_string.replace('h', 'o')
new_string = new_string.replace('x', 'h')
new_string = new_string.replace('v', 'p')
new_string = new_string.replace('j', 'v')
new_string = new_string.replace('r', 'j')
new_string = new_string.replace('n', 'r')
new_string = new_string.replace('g', 'n')
new_string = new_string.replace('w', 'g')
new_string = new_string.replace('y', 'l')
new_string = new_string.replace('b', 'q')
new_string = new_string.replace('z', 'b')
new_string = new_string.replace('m', 'z')
new_string = new_string.replace('f', 'm')
new_string = new_string.replace('d', 'f')
new_string = new_string.replace('q', 'd')
new_string = new_string.replace('t', 'k')
new_string = new_string.replace('i', 't')
new_string = new_string.replace('c', 'i')

print("\nMessage: \n", new_string)

# Letter swap correlations

# ABCDEFGHIJKLMNOPQRSTUVWXYZ
#  QIF MNOTV  ZRU DJEKSPGHLB
```

Bijlage Opdracht 15 – PS-script

```
#Work method:
#Contents of hashes.txt = the hashes we are trying to crack
#Contents of md5woordenlijst = wordlist with hashes
#Contents of New-Item matches.txt = all matches found
#Iterate through hashes.txt and add all hashes to an array
#Iterate through woordenlijst.txt and add these to second array
#Nested for loop to check for matches in both arrays
#Transfer matches to matches.txt


$matches = New-Item -ItemType File -Name "matches.txt" `
-Path C:\Users\Rasmus\Documents\Jaar_1\Security_Essentials

[string[]]$wordlistArray = Get-Content `
-Path C:\Users\Rasmus\Documents\Jaar_1\Security_Essentials\md5woordenlijst.txt

[string[]]$hashesArray = Get-Content `
-Path C:\Users\Rasmus\Documents\Jaar_1\Security_Essentials\hashes.txt

foreach ($string in $hashesArray) {
       foreach ($hash in $wordlistArray) {
         if ($hash.Contains($string))
         {
             Add-Content `
             -Path C:\Users\Rasmus\Documents\Jaar_1\Security_Essentials\matches.txt
$hash
         }
     }
}
```

Bijlage Opdracht 15 - Woordenlijst Ernest

Initiele woordenlijst Ernest

Ar{emis, Ernest, Vanderbauwen, Password, Youareontherightpath, ernest! , HogeschoolPXL, Hengelen, Dierentraining, InviteUrlsAreNotSecure, WaitASecond, AlmostThere, Covid19, Vaccine, Covid19Vaccine, 15012021, YouDitIt, Sonny, PXL{InviteUrlsAreNotSecure}, PXl{SussyBaka}, PXL{Wait_A_Second}, PXL{Almost_There}, PXL{YOU_DID_IT}, GoodPassWord, PXL{You_Are_On_The_Right_Path}, @Lodrimed1, PXL{Ernest!}, Hondenshows, .CHECK HERE, medewerker, Siebe, Siebe#9999, medewerker1#1894, VanderbauwenPXL, ArtemisPXL, SonnyPXL, LodrimedPXL, VaccinePXL, ErnestVanderbauwen, password, ErnestVanderbauwen123, Artemis123, Sonny123, Password123, PXL123, 1234, qwerty, ErnestLodrimed, ErnestPXL, SonnyPXL, ErnestVaccine, ErnestCovid, ErnestCovd19, Artemis, artemis, pxl, ernestvanderbauwen, Vanderbouwen, erne${, 3rn3$[, 3rn3sT, OSINT, LoDriMed1, @LoDriMed1, CKY{Lbh_Qvq_vg!}, {You_DID_IT}, {Almost_There}, {Wait_A_Second}, {Sussy_Baka}, {InviteUrlsAreNotSecure}{We_Should_Not_Be_Here}, {Ernest!}, 3rn3$t, Erne${, V@nd3rb@uw3n, 3rn3$tV@nd3rb@uw3n, 3rn3$[L0dr!m3d, L0drim3d, PXL, Lodrimed Vaccine, SussyBaka, 3mei1961, 351961

Opdracht 16 – Gebruikte tool

https://www.lostmypass.com/file-types/ms-word/

Bibliografie

Adding a new SSH key to your GitHub account. (z.d.). GitHub Docs. Geraadpleegd op 19 maart 2022, van https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account

Advantages of AES | disadvantages of AES. (2017). RF Wireless World. Geraadpleegd op 19maart 2022, van https://www.rfwireless-world.com/Terminology/Advantages-and-disadvantages-of AES.html#:%7E:text=Drawbacks%20or%20disadvantages%20of%20AES&text=%E2%9E%A8It%20uses%20too%20simple,performance%20and%20security%20into%20considerations

AES Encryption in Java. (2021, 30 september). Metamug. Geraadpleegd op 19 maart 2022, van https://metamug.com/article/security/aes-encryption-in-java.html

C. (z.d.-a). *GitHub - clem9669/hashcat-rule: Rule for hashcat or john. Aiming to crack how people generate their password*. GitHub. Geraadpleegd op 19 maart 2022, van https://github.com/clem9669/hashcat-rule

Castellanos, R. (2022, 7 januari). *Secure your sensitive data with AES in Flutter - Ricardo Castellanos*. Medium. Geraadpleegd op 19 maart 2022, van https://ricardo-castellanos-herreros.medium.com/secure-your-sensitive-data-with-aes-in-flutter-63b747e1fd7e

Convert PHP script in Powershell script. (2017, 12 juli). Stack Overflow. Geraadpleegd op 19 maart 2022, van https://stackoverflow.com/questions/45056904/convert-php-script-in-powershell-script

CrackStation - Online Password Hash Cracking - MD5, SHA1, Linux, Rainbow Tables, etc. (z.d.). Crackstation. Geraadpleegd op 19 maart 2022, van https://crackstation.net/

Create Custom Wordlists with the Mentalist for Brute-Forcing [Tutorial]. (2018, 12 april). YouTube. Geraadpleegd op 19 maart 2022, van https://www.youtube.com/watch?v=01-Dcz1hFw8

Cryptography with Python - Quick Guide. (2018). TutorialsPoint. Geraadpleegd op 19 maart 2022, van https://www.tutorialspoint.com/cryptography_with_python/cryptography_with_python_quick_guide.htm

Disable unsecure SSL versions. (z.d.). Xolphin. Geraadpleegd op 19 maart 2022, van https://www.xolphin.com/support/FAQ/Disable_unsecure_SSL_versions#:%7E:text=The%20main%20reasons%20for%20the,for%20both%20authentication%20and%20encryption.

Ellingwood, J., & Boucheron, B. (2021, 16 juni). *How To Configure SSH Key-Based Authentication on a Linux Server*. DigitalOcean Community. Geraadpleegd op 19 maart 2022, van https://www.digitalocean.com/community/tutorials/how-to-configure-ssh-key-based-authentication-on-a-linux-server

example_hashes [hashcat wiki]. (z.d.). Hashcat. Geraadpleegd op 19 maart 2022, van https://hashcat.net/wiki/doku.php?id=example_hashes

GeeksforGeeks. (2021, 8 juni). *Difference between AES and DES ciphers*. Geraadpleegd op 19 maart 2022, van https://www.geeksforgeeks.org/difference-between-aes-and-des-ciphers/

How to install Mentalist in kali linux to generate wordlist. (2021, 26 juni). YouTube. Geraadpleegd op 19 maart 2022, van https://www.youtube.com/watch?v=NF6RReRdjIg
Hurer-Mackay, W. (2016, 12 september).

How To Perform A Rule-Based Attack Using Hashcat. 4ARMED Cloud Security Professional Services. Geraadpleegd op 19 maart 2022, van https://www.4armed.com/blog/hashcat-rule-based-attack/#:%7E:text=In%20short%2C%20a%20rule%2Dbased,generate%20new%20passwords%20to%20use.

Jack, P. B. (z.d.). *Hash Analyzer - TunnelsUP*. Tunnelsup. Geraadpleegd op 19 maart 2022, van https://www.tunnelsup.com/hash-analyzer/

Loshin, P. (2021, 20 augustus). *Data Encryption Standard (DES)*. SearchSecurity. Geraadpleegd op 19 maart 2022, van https://www.techtarget.com/searchsecurity/definition/Data-Encryption-Standard

M. (z.d.-b). *GitHub - Mebus/cupp: Common User Passwords Profiler (CUPP)*. GitHub. Geraadpleegd op 19 maart 2022, van https://github.com/Mebus/cupp

mask_attack [hashcat wiki]. (z.d.). Hashcat. Geraadpleegd op 19 maart 2022, van https://hashcat.net/wiki/doku.php?id=mask_attack

MD5 Hash Generator. (z.d.). MD5 Hash Generator. Geraadpleegd op 19 maart 2022, van https://www.md5hashgenerator.com/

N. (z.d.-c). *PS_MultiCrack/PS_MultiCrack.ps1 at d495323918e4fde5927991c9076d490e43a35a36 · NetSPI/PS_MultiCrack*. GitHub. Geraadpleegd op 19 maart 2022, van https://github.com/NetSPI/PS_MultiCrack/blob/d495323918e4fde5927991c9076d490e43a35a36/PS_MultiCrack.ps1

Rajora, H. (2022, 19 maart). *How to set up SSH and Clone Repository using SSH in Git?* TOOLSQA. Geraadpleegd op 19 maart 2022, van https://www.toolsqa.com/git/clone-repository-using-ssh/

rule_based_attack [hashcat wiki]. (z.d.). Hashcat. Geraadpleegd op 19 maart 2022, van https://hashcat.net/wiki/doku.php?id=rule_based_attack#:%7E:text=The%20rule%2Dengine%20in%20Hashcat,letter%2Dnames%20to%20avoid%20conflicts.

S. (z.d.-d). *GitHub - sc0tfree/mentalist: Mentalist is a graphical tool for custom wordlist generation. It utilizes common human paradigms for constructing passwords and can output the full wordlist as well as rules compatible with Hashcat and John the Ripper.* GitHub. Geraadpleegd op 19 maart 2022, van https://github.com/sc0tfree/mentalist

Secret Key Exchange (Diffie-Hellman) - Computerphile. (2017, 15 december). YouTube. Geraadpleegd op 19 maart 2022, van https://www.youtube.com/watch?v=NmM9HA2MQGI&ab_channel=Computerphile

Secure your data with AES-256 encryption. (2019, 26 juni). Atpinc. Geraadpleegd op 19 maart 2022, van https://www.atpinc.com/blog/what-is-aes-256-encryption

Server Name Indication (SNI). (z.d.). SSL Certificaten. Geraadpleegd op 19 maart 2022, van https://www.sslcertificaten.nl/support/Terminologie/Server_Name_Indication_(SNI)

Spacey, J. (z.d.). *Cryptography: Salt vs Pepper*. Simplicable. Geraadpleegd op 19 maart 2022, van https://simplicable.com/new/salt-vs-pepper

Using ssh-copy-id to install SSH keys on servers as authorized keys for passwordless authentication. Options and troubleshooting. (z.d.). SSH. Geraadpleegd op 19 maart 2022, van https://www.ssh.com/academy/ssh/copy-id