Code-uitleg:

Fase 2:

We gebruiken int om van het getal tekst te maken dus 3 wordt dan “3” en dan gebruiken we chr om van het getal een karakter te maken. Daarna plakt die ze allemaal aan elkaar.

Fase 2 expert:

Dit is vrijwel hetzelfde als fase 2, maar we sturen daarna ook nog OK terug naar de verzender als bevestiging. Wij hebben geen komma lijst gebruikt, want dan zal het bericht te lang worden om via de radio te versturen.

Fase 3:

Hier hebben we een key afgesproken en door op knop B te drukken verhoogt die hem totdat die bij 26 is, dan gaat die weer naar 0. Ook als je een bericht hebt ontvangen verhoogt die de key zodat het Caesar Cipher wordt.

Fase 4:

Bij fase 4 moesten we een 2 weg chat maken. Het eerste wat we hebben gedaan is de eigen bit een letter geven, bijvoorbeeld “A” en de bit waarmee we willen chatten een letter te geven, bijvoorbeeld “B”. Als we dan op de linker knop van de microbit klikken, kan je invoeren wat er verzonden moest worden. Daarna verstuurd die iedere letter in het bericht met ceasar cipher en ascii, maar in tegenstelling tot fase 3 wordt er voor het bericht ook de letter van de andere bit neer gezet. Als laatste stuurt die dan de eigen bit letter met daarna “END” om aan te geven dat dat het hele bericht was.
Bij het ontvangen schrijft die elke letter op en haalt die de ascii weg totdat die “END” binnen krijgt, dan stuurt die nog even de eigen bit letter en “OK” om aan te geven dat het bericht is ontvangen.
Als verzender wacht je na het verzenden op de andere bit letter en “OK” en dan weet die ook dat de tekst ontvangen is.

Fase 5:

Fase 5 is bijna dezelfde code als fase 4, het enige echte verschil is dat de bit waarnaar je een bericht wilt versturen nu niet hard coded erin staat. Maar aangezien niet iedereen dezelfde key voor de cipher kan hebben wordt er een lijst bijgehouden waar staat welke bit welke key heeft zodat ze niet in de war raken.

Fase 6:

Fase 6 lijkt weer een hoop op fase 5, wij hadden ook al een half ACK systeem ingebouwd, het enige wat moest worden toegevoegd was retries. Dit was vrij simpel, we hebben een een variabelen gemaakt die het aantal retries telt, als de verzender geen OK terug krijgt blijft die hem na een tijdsinterval verzenden totdat die hem 5 keer heeft verzonden, dan stopt die en zegt die dat het is mislukt. Daarnaast zijn we ook gaan bijhouden hoeveel bericht correct zijn

verzonden. Dit wordt tegelijk bij ACK gedaan, als die de 5 retries heeft gedaan telt die de gefaalde variabelen op en als die OK terug krijgt telt die de geslaagde variabelen op.

Fase 7:

Hier is niet heel veel aangepast, het was ook voornamelijk een fase voor de onderschepper. Het enige wat we hebben gedaan is de key verhogen per letter in plaats van per bericht. Hierdoor lijkt de letter E bijvoorbeeld niet de hele tijd op elkaar, en is het moeilijker voor de onderschepper om de key te raden.

Fase 8:

Bij fase 8 moesten we het netwerk verder beschermen, in eerste instantie moest dat met 2 extra functies, maar dat werd te ingewikkeld dus hebben we alleen gekozen voor vaste berichtlengte. Om dat te doen hebben we een variabelen gemaakt met elk mogelijke karakter. Bij het verzenden wordt gekeken hoe lang het bericht is dat je wilt versturen als het langer dan 100 tekens is werd die afgeknipt, als die geen 100 tekens is wordt die aangevuld met de random karakters van eerst totdat die bij 100 tekens is, en om aan te geven waar de random karakters beginnen hebben we een of ander ascii tekentje die niemand gebruikt. Daarna verzendt die hem op dezelfde manier als eerst.
De ontvanger doet dan precies hetzelfde als eerst, maar als die dat tekentje tegenkomt stopt die ermee, want dan weet die dat de rest onzin is.

Fase 9:

Bij fase 9 moesten we nog iets anders kunnen verzenden dan tekst, wij hebben gekozen voor afbeeldingen aangezien dat al een soort van ingebouwd zat in de microbits. Je krijgt bij het verzenden de optie om te kiezen voor tekst verzenden of een foto, als je voor foto kiest krijg je elk mogelijke afbeelding die op het schermpje kan worden gezet. Dan toets je het nummer van die foto in en gaat die een bericht maken, als eerst zet die “ifedvkm:” (ImageFotoEmojiDingenVoorKutMicrobit) neer en daarna geeft die de naam van de foto die in de microbit staan. Als de andere microbit dan het bericht ontving en het begon met “ifedvkm:”, dan het enige wat die nog moet doen is de naam van het fototje pakken en op het schermpje laten zien door “display.show(‘foto naam’)”. Voor de rest doet die alles precies hetzelfde.
Is dit de meest efficiënte manier om dit te doen? Waarschijnlijk niet, een hele lijst met elif’s lijkt mij niet super efficiënt, maar het was wel makkelijk.

AI-verantwoording:

De grote vraag is natuurlijk, hebben we AI gebruikt? Het antwoord daarop: ja, natuurlijk, anders was deze opdracht niet op tijd af. Hebben we AI alles laten doen, nee. AI heeft voornamelijk korte stukjes uitgelegd, denk aan vragen zoals: “wat houdt ACK in” en “Hoe split je een stuk tekst na een specifiek teken in python”. Af en toe hebben we gevraagd of AI stukjes code kon schrijven, deze hebben wij zoveel mogelijk aangepast zodat het beter past bij onze manier van coderen (vaak wat minder efficiënt, maar wel werkend). Als iets erg ingewikkeld was hebben we AI hele blokken code laten maken, maar hebben we het wel zelf teruggelezen en ons best gedaan om te begrijpen, wat wel goed gelukt is want we hebben

er op door kunnen werken en code uitleg kunnen geven. We hebben geen enkele keer volledige fases door AI laten maken.

Zwakte analyse (bij fase 9):

In onze code is de grootste zwakte dat als je de key onderschept gelijk het hele bericht ontcijfert kan worden. Ook omdat de rolling key steeds met 1 verhoogd wordt zorgt het ervoor dat de key in het volgende bericht best wel makkelijk opnieuw te ontcijferen is. We hebben ook geen dummy berichten wat ervoor zorgt dat de key goed te ontcijferen is al snap je het systeem. Ook dat we maar met 25 verschillende keys werken zorgt ervoor dat het best wel makkelijk gebruteforced kan worden omdat er maar 25 mogelijke keys zijn. Ook is er een zwakte in hoe de ACK werkt. De acknowledgement is het systeem waardoor de zender weet dat de ontvanger het bericht ontvangen heeft. Dit doet onze code met een simpel OK bericht. De onderschepper kan de ACK makkelijk namaken en ervoor zorgen dat het lijkt alsof de ontvanger het bericht ontvangen heeft terwijl die hem niet ontvangen heeft. Als de onderschepper dit doet dan zorgt het er ook voor dat de verzender stopt met verzenden wat ervoor kan zorgen dat ontvanger niks ontvangt.

Reflectie:

Als we deze opdracht in één woord moeten omschrijven is het K-U-T. Wij merkte zelf dat de microbits best wel irritant werkten, dan hebben we het voornamelijk over de snelheid van de dingen, de limieten en de verbinding met de laptops. Als we dat even erbuiten houden merken we dat de opdracht best ingewikkeld was, dit kwam waarschijnlijk ook wel gedeeltelijk door onszelf. We merken dat we soms wel bepaalde dingen al doen voordat het eigenlijk nodig was, denk aan fase 5 naar 6, toen hadden we al een soort van eigen ACK systeem gebouwd.
Fase 8 was verreweg het ingewikkeldst, uiteindelijk is er ook besloten om deze fase te verkleinen. Het nepverkeer raakte compleet in de war met al het normale verkeer waardoor niks meer werkte.
Wat heb ik hier nou van geleerd? Deze opdracht heeft mij wat meer over python geleerd, voor mijn idee ook op een betere manier dan de andere python opdrachten die ik eerder dit jaar had gedaan. En het heeft me wat geleerd over encryptie, niet de meest bijzondere dingen, ik herkende wel aardig wat, maar ik had het nog nooit zelf toegepast dus daar heb ik nu wel van geleerd.
Zou ik de opdrachten nu anders hebben gedaan? Nee en ja, in principe niet, want het werkt gewoon en alles is af, maar als ik mijzelf een kritiek puntje mag geven is het dat het waarschijnlijk beter was geweest om je echt precies aan de opdracht te houden en niet extra dingen al te proberen, want dan moet het later weer anders ofzo en is het zonde van de tijd.