Projekt SYCY

-

Zespół nr 9

Jakub Szweda

Mateusz Plichta

Maksymilian Młodnicki

Karol Żelazowski

Anna Paziewska

Politechnika Warszawska, Wydział Elektroniki i Technik Informacyjnych

14 lutego 2025

Historia zmian

Wersja	Data	Autor	Opis zmian
1.0	9.03.2023	JS, AP, MP, MM, KŻ	Pierwsza wersja raportu etapu 1
1.1	12.03.2023	JS, AP, MP, MM, KŻ	Wyjaśnienie Design Thinking 2.1
1.2	14.03.2023	JS, AP, MP, MM, KŻ	Zarządzanie projektem2.2
2.0	30.03.2023	JS, AP, MP, MM, KŻ	Rozpoczęcie pracy nad etapem drugim 3
2.1	4.04.2023	JS, AP, MP, MM, KŻ	Poprawki odnosnie drugiego etapu
3.0	22.04.2023	JS, AP, MP, MM, KŻ	Rozpoczęcie pracy nad etapem trzecim - burza mózgów
3.1	23.04.2023	JS, AP, MP, MM, KŻ	Dalsza praca nad etapem trzecim- schemat blokowy i implementacja w pythonie i c
4.0	9.05.2023	JS, AP, MP, MM, KŻ	Pierwsza wersja etapu IV
4.1	15.05.2023	JS, AP, MP, MM, KŻ	Dodanie zawartości na temat modułów w etapie IV
4.2	16.05.2023	JS, AP, MP, MM, KŻ	Weryfikacja i ocena rozwiązania w etapie IV

Opracowanie w ramach programu NERW, 2020. Paweł Tomaszewicz, Zakład Cyberbezpieczeństwa, IT, PW.

4.3	21.05.2023	JS, AP, MP, MM, KŻ	Małe poprawki w etapie IV
5.0	26.05.2023	JS, AP, MP, MM, KŻ	Zaczęcie prac nad etapem V
5.1	27.05.2023	JS, AP, MP, MM, KŻ	Dodane poprawki do etapu V

Spis treści

Historia zmian . . . 1

1. Wstęp . . . 3

2. Organizacja prac . . . 3

2.1. Design Thinking . . . 4

2.2. Zarządzanie projektem . . . 5

2.2.1. Treść zadania . . . 5

2.2.2. Metody . . . 6

2.2.3. Narzędzia . . . 8

3. Informacje podstawowe . . . 8

3.1. Szyfrowanie . . . 8

3.1.1. Rodzaje szyfrowania . . . 8

3.2. Możliwe rozwiązania . . . 10

3.2.1. RSA . . . 10

3.2.2. Algorytm Diffiego-Hellmana . . . 10

3.2.3. Szyfr Rabina . . . 10

3.2.4. Puzzle Merkle'a . . . 12

3.2.5. Curve25519 . . . 12

3.3. Narzędzia . . . 13

3.3.1. Quartus . . . 13

3.3.2. Wolfram Mathematica . . . 13

3.3.3. Python . . . 13

4. Koncepcja . . . 14

4.1. Mapa myśli . . . 14

4.2. Wybór algorytmu . . . 14

4.2.1. Curve25519 . . . 14

4.2.2. Puzzle Merkle'a . . . 14

4.2.3. Szyfr rabina . . . 14

4.2.4. RSA . . . 14

4.2.5. Diffie-Hellman . . . 15

4.3. Schemat blokowy . . . 16

4.4. Przykład ilustrujący działanie . . . 16

4.5. Model referencyjny . . . 17

4.6. Przykładowe dane do testowania . . . 18

5. Implementacja . . . 19

5.1. Moduł World . . . 19

5.1.1. Moduł C&C . . . 21

5.1.2. Moduł Powermod . . . 24

5.1.3. Moduł generowania pseudolosowych liczb 8-bitowych . . . 26

5.1.4. Moduł losowania liczb pierwszych . . . 27

5.1.5. Moduł odszyfrowywania i zaszyfrowywania wiadomości . . . 29

5.1.6. Moduł Dron . . . 30

5.1.7. Symulacja . . . 33

5.2. Szybkość działania . . . 34

5.3. Weryfikacja funkcjonalna . . . 34

5.4. Ocena rozwiązania . . . 34

6. Uruchomienie . . . 34

7. Rozbudowa systemu . . . 36

8. Podsumowanie 36
9. Bibliografia 36

1. Wstęp

Projekt polega na stworzeniu koncepcji mechanizmu, który umożliwi przekazywanie z centrum dowodzenia do dronów patrolujących przybrzeżnych obszarów klucz szyfrujący (lub informacji umożliwiającej wygenerowanie klucza po stronie drona) dla szyfru strumieniowego oraz zaprojektować system cyfrowy umożliwiający realizację przekazywania klucza lub informacji do jego generacji w sposób bezpieczny.

Raport ma na celu dokumentację projektu z przedmiotu Systemy Cyfrowe semestr letni 2023. Jest on wykonywano przyrostowo. Regularnie jest on rozbudowywany o kolejne etapy względem poniższego schematu.

I Etap wstępny
stworzenie zespołu i organizacja warsztatu pracy, wybór metody zarządzania projektem i opis jego założeń. Podział pracy i wybór środowisk do wymiany informacji.

II Etap zdobywania informacji
analiza literatury, zebranie wiedzy teoretycznej odnośnie kryptografii, poznanie różnych sposób na szyfrowanie

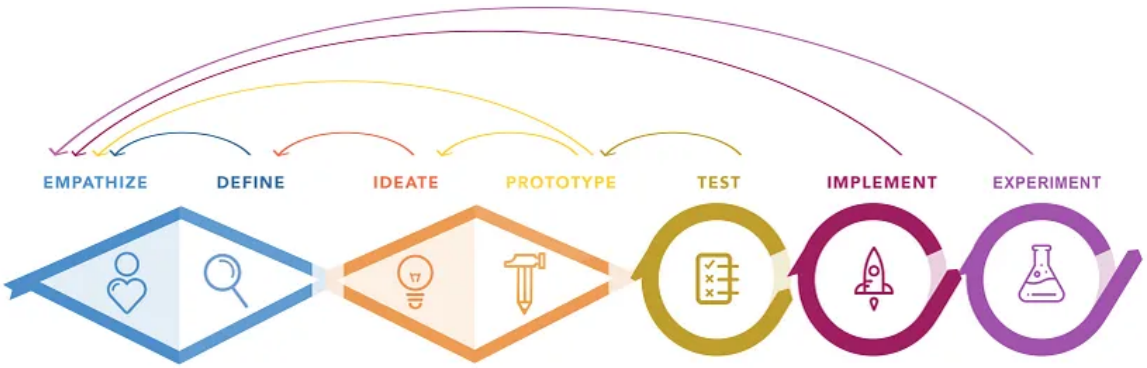
III Etap opracowania koncepcji
szukanie rozwiązań, tworzenie mapy myśli przy pomocy burzy mózgów, opracowanie koncepcji rozwiązania na podstawie zdobytej wiedzy, opracowanie prostego modelu referencyjnego (najprawdopodobniej przy użyciu Javy, której uczymy się w trakcie studiowania) w celu przeprowadzenia wstępnych testów

IV Etap implementacji – na tym etapie rozwijamy i rozbudowujemy koncepcje projektowe docelowego systemu, modelujemy elementy systemu w HDL, weryfikujemy funkcjonalnie, integrujemy i oceniamy prototypy,

V Etap uruchomienia – wdrożenie projektu, uruchomienie na docelowej platformie, przetestowanie według wcześniej opracowanych scenariuszy testowych.

2. Organizacja prac

Praca w naszym projekcie opiera się na dwóch konceptach. Łączymy założenia design thinking wraz z systemem zasad pracy i wartości zwanym agile. Obie metody nie wykluczają siebie nawzajem a ich wspólne wykorzystanie nazywane jest "Dual Track"



Rys. 1. Dualtrack `https://miro.medium.com/v2/resize:fit:828/format:webp/1*B9Kn2si8zRAwJoCHtKSxcw.jpeg`

Rozdział ten powinien opisywać zadania zrealizowane w ramach EtapuI. Można omówić:

— podejście Design Thinking (w wersji Double Diamond lub innej),
— wybór sposobu zarządzania projektem
— organizacja warsztatu pracy, dobór narzędzi (Overleaf, Microsoft Teams, GutHub, itp.)

2.1. Design Thinking

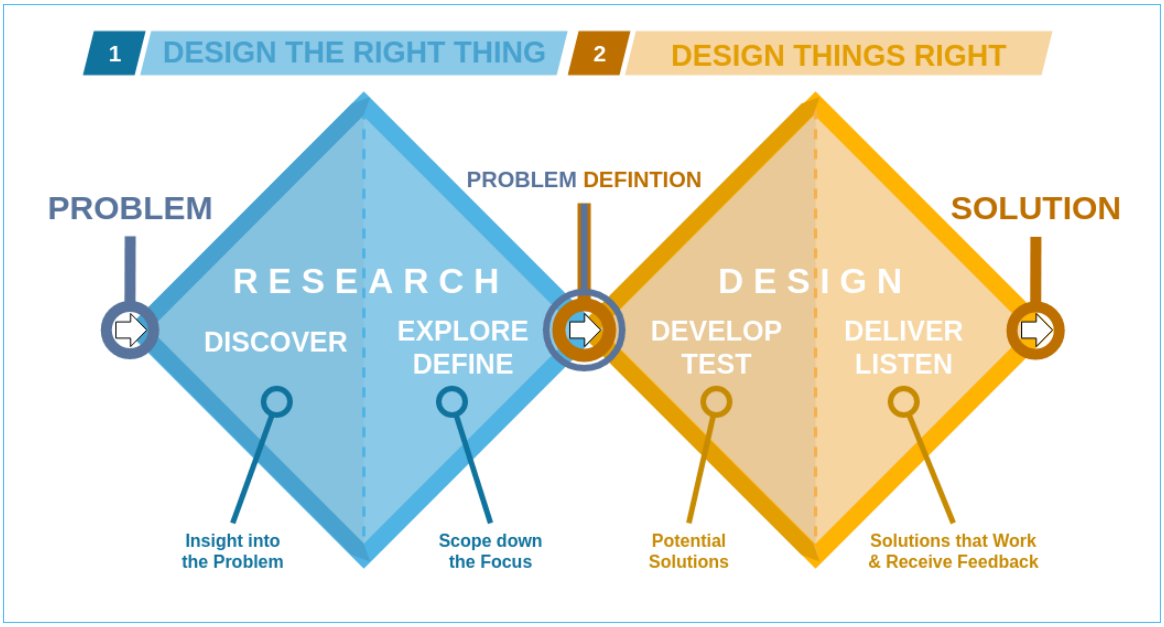
Postanowiliśmy wybrać Design Thinking w wersji Double Diamond.
Proces w metodyce Double Diamond zakłada cztery etapy:

Discover – pierwszy etap "odkrywania"
W tym etapie zbieramy informacje z wielu źródeł na różne sposoby. W tym etapie celem jest zrozumienie potrzeb użytkownika. Nie otrzymujemy tu gotowych rozwiązań a jedynie zbieramy wszystkie możliwości.

Define – drugi etap "definiowania" lub "precyzowania"
W kolejnym kroku na bazie danych zdobytych w etapie odkrywania analizujemy je dogłębnie i staramy się wyciągnąć konkretne wnioski i zostawiamy najistotniejsze informacje.

Develop – trzeci etap "generowania pomysłów"
Trzeci etap polega na generowaniu pomysłów. Na podtsawie wniosków z poprzednich etapów należy wygenerować jak najwięcej tych pomysłów na wykonanie danego zadania, chociażby za pomocą burzy mózgów. W naszym przypadku będzie to na przykład obmyślanie w jaki sposób ma być przekazywany klucz ze skrzynki na skrzynke, tak aby za każdym razem była możliwość generowania innego klucza.

Deliver – czwarty etap "dostarczania"
W ostatnim etapie z bazy pomysłów, które otrzymaliśmy wybieramy jeden i staramy sie go idealizować. W tym celu tworzymy prototypy a następnie je testujemy wprowadzając drobne poprawki.

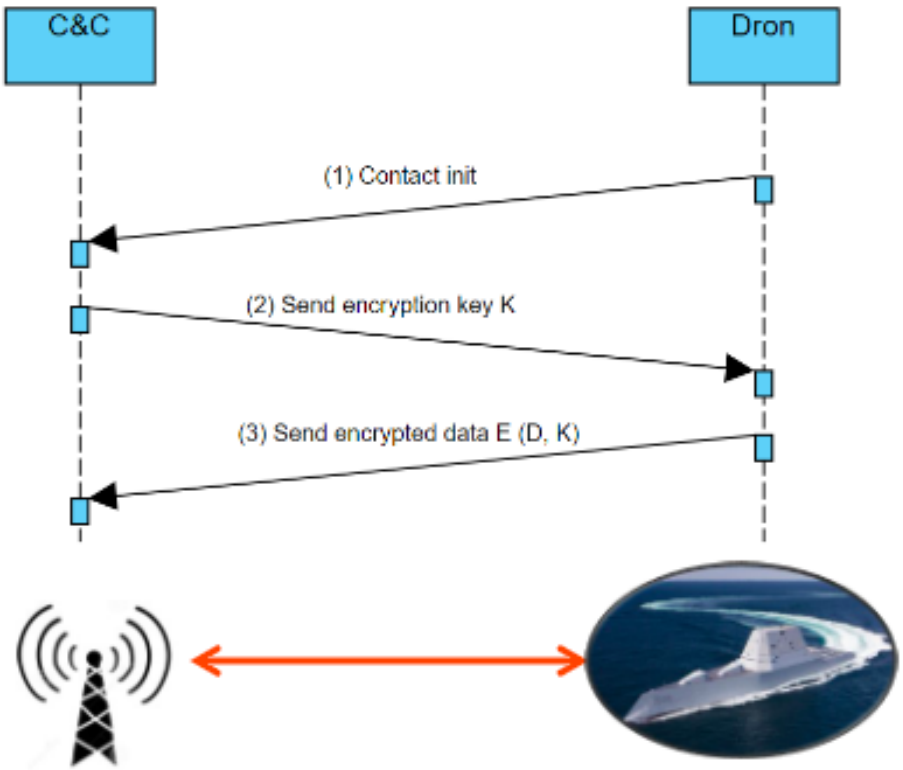


Rys. 2. Grafika przedstawiająca Design Thinking w wersji Double Diamond `https://upload.wikimedia.org/wikipedia/commons/b/bd/Double_diamond.png`

2.2. Zarządzanie projektem

2.2.1. Treść zadania

Należy zaproponować koncepcję mechanizmu umożliwiającego przekazywania z centrum dowodzenia(Command & Control) do drona(Dron) klucza szyfrującego (lub informacji umożliwiającej wygenerowanie klucza po stronie drona) dla szyfru strumieniowego. Należy wykorzystać lekki algorytm strumieniowy z kluczem o długości 8 bitów. Dla podniesienia poziomu bezpieczeństwa klucz musi być zmieniany przy każdej transmisji oraz jedynie konkretny dron może wykorzystać klucz K. Atakujący nawet po przechwyceniu innych dronów nie może być w stanie odtworzyć klucza w transmisji z innymi dronami.



Rys. 3. Grafika przedstawiająca działania wykonywane przez C&C i Dron

Naszym zadaniem jest stworzenie dwóch czarnych skrzynek (C&C, Dron) bez brania pod uwagę procesu trasmisji.

2.2.2. Metody

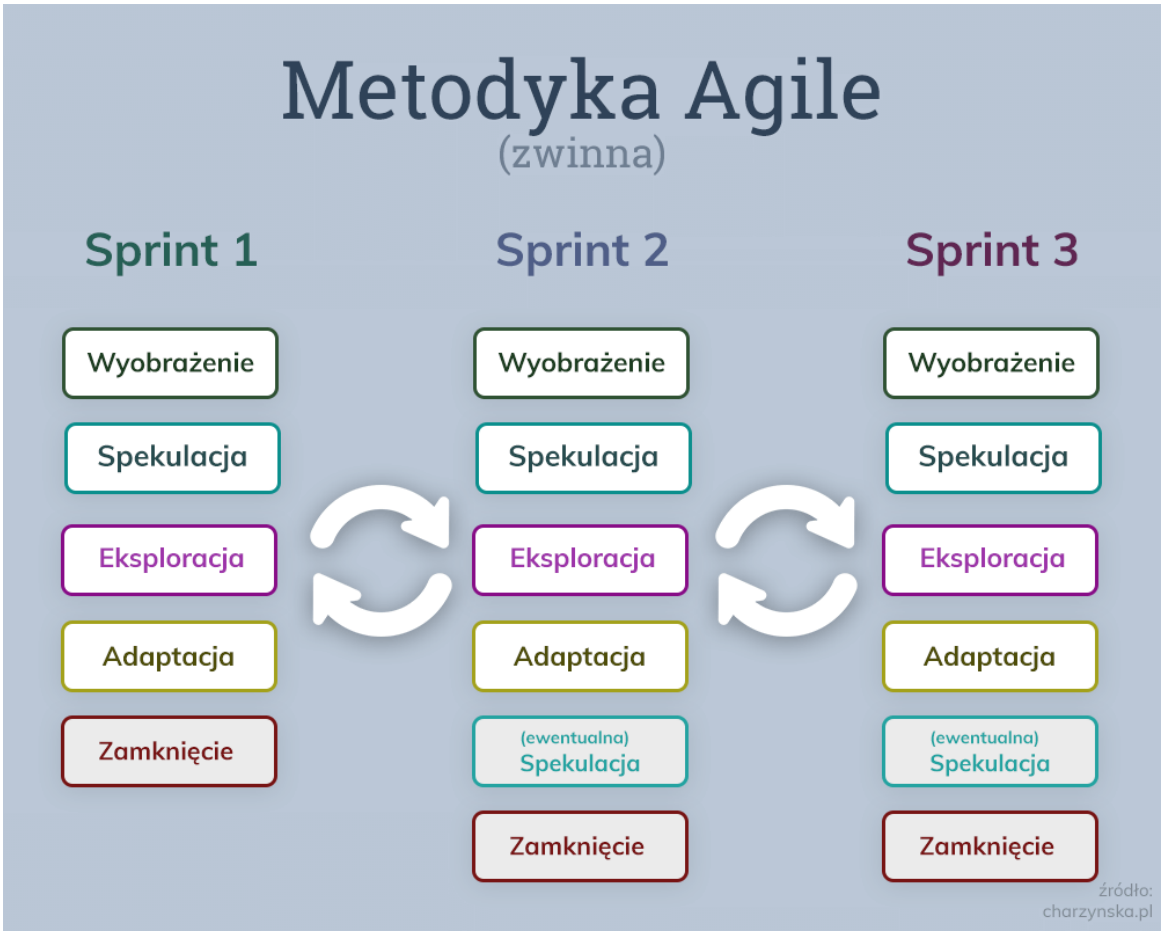
Wybraliśmy "Agile Project Management" jako sposób zarządzania etapami projektu. Tego rodzaju sposób zarządzania zakłada:

— Elastyczność

— Brak konieczności definiowania całego zakresu zadania na samym początku

— Wzmocnienie samodzielność zespołu

— Oparcie o "Sprinty"



Rys. 4. Wykorzystanie sprintów w metodyce Agile `https://charzynska.pl/agile-pm-poradnik-cz1/`

Elementy Agile:

Wyobrażenie: Co chcemy zbudować?
Kto powinien należeć do zespołu?
Jakie normy i wartości chcemy przyjąć?

Spekulacja:Plan dostarczenia elementów.
Estymacja kosztów każdego elementu.
Zagrożenia utrudniające realizację każdego elementu.

Eksploracja:Budowa produktu
Codzienne spotkania.
Usuwanie problemów utrudniających pracę.

Adaptacja: Finalna ocena funkcji produktu.
Burza mózgów-omawianie sposobów rozwiązania problemów.
Dodawanie lub usuwanie funkcjonalności.
Porównanie postępu prac do planu.

Zamknięcie:Przydzielenie członków zespołu do innych zadań.
Przedstawienie ogólnego podsumowania projektu.
Upewnienie się,że cele projektu zostały osiągnięte.

2.2.3. Narzędzia

W ramach zarządzania tym projektem założyliśmy korzystanie z poszczególnych platform:

Messenger: Ze względu na ogólną dostępność do szybkiej komunikacji tekstowej jak również planowania przyszłych spotkań zamierzamy korzystać z tej platformy.

Discord: Spotkania w ramach realizacji projektu planujemy częściowo przeprowadzić na miejscu, jednakże w większości bedą się one odbywać na tej platformie z uwagi na łatwość udostępniania ekranu.

GitLab: Aby ułatwić wspólną pracę nad kodem zamierzamy korzystać z platformy Gitlab umożliwiającą podział na "branche" jak również wersjonowanie plików. Wybraliśmy ją także ze względu na ogólne zaznajomienie z nią w trakcie trwania studiów.

Microsoft Teams: Do kontaktu z koordynatorem projektu jak również do wysyłania raportów z realizacji poszczególnych etapów.

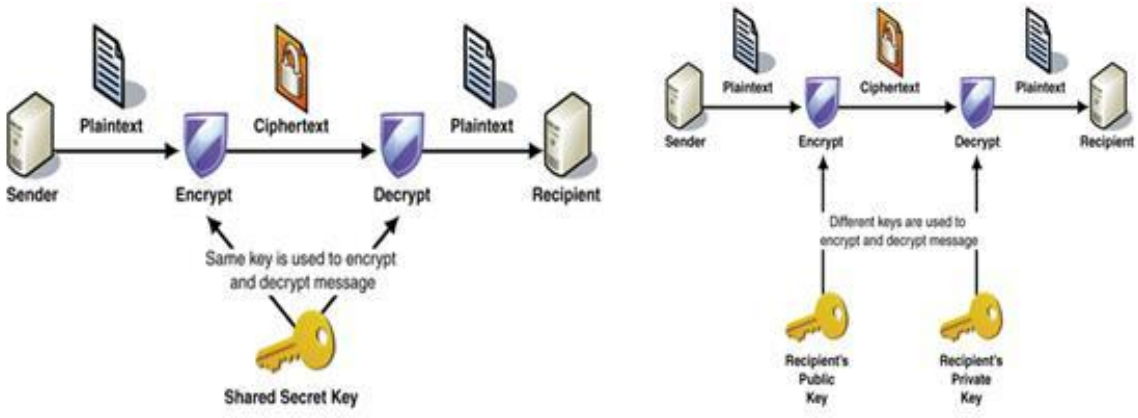
Overleaf: Strona internetowa umożliwiająca edycję plików z rozszerzeniem **.tex** pozwalająca na współprace użytkowników w czasie rzeczywistym jak również na podgląd skompilowanej wersji pliku

3. Informacje podstawowe

3.1. Szyfrowanie

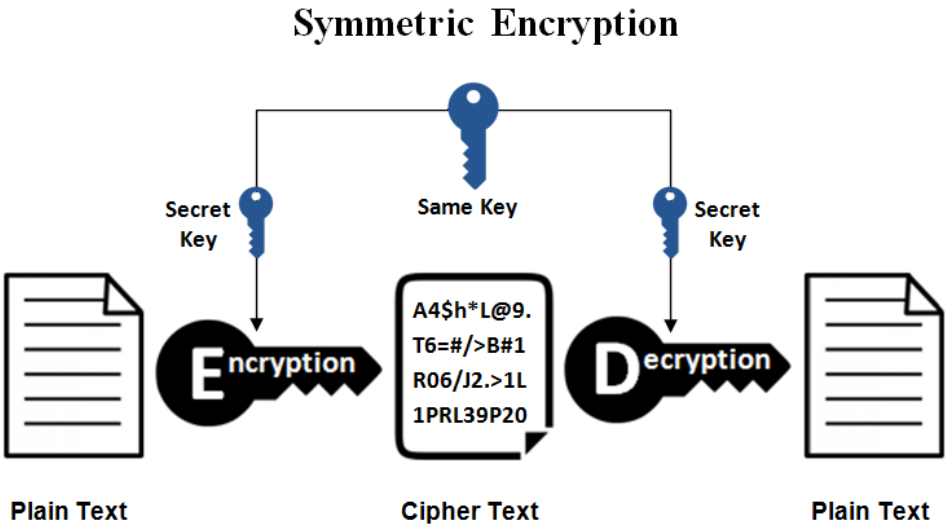
3.1.1. Rodzaje szyfrowania

Szyfrowanie dzieli się na dwie kategorie: szyfrowanie symetryczne i asymetryczne.



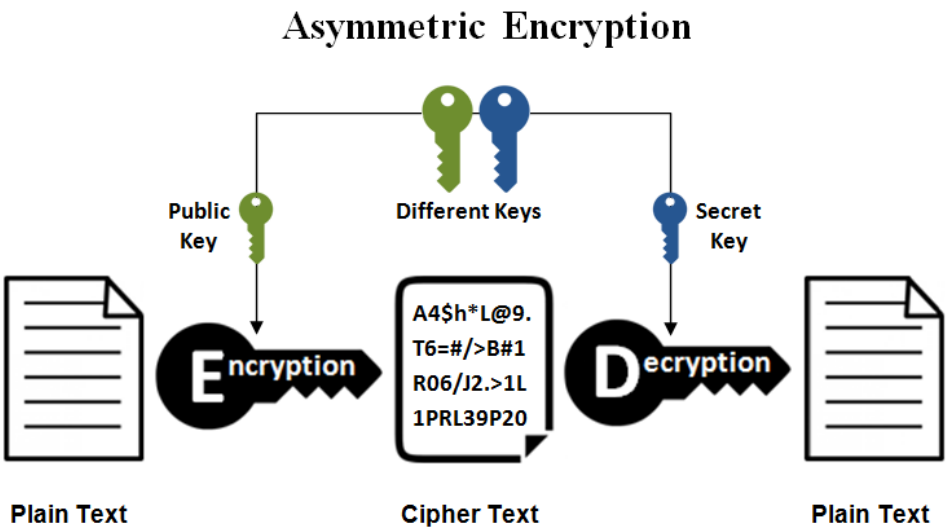
Rys. 5. Różnice pomiędzy symetrycznym, a asymetrycznym szyfrowaniem `https://www.websiterating.com/pl/vpn/glossary/what-is-asymmetric-symmetric-encryption/`

Szyfrowanie symetryczne: Szyfrowanie symetryczne wyróżnia się tym, że do zaszyfrowania i odszyfrowania wiadomości wykorzystuje się jeden klucz. Przykładem algorytmu symetrycznego jest DES, AES. Algorytmy symetryczne dzielą się też na kolejne dwie kategorie: algorytmy strumieniowe i algorytmy blokowe.



Rys. 6. Szyfrowanie symetryczne https://www.websiterating.com/pl/vpn/glossary/what-is-asymmetric-symmetric-encryption/

Szyfrowanie asymetryczne: Szyfrowanie asymetryczne wykorzystuje dwa klucze(w przeciwieństwie do symetrycznego, który wykorzystuje tylko jeden), jeden publiczny i jeden prywatny. Wszyscy mają dostęp do klucza publicznego i za jego pomocą mogą zaszyfrować wiadomość. Z kolei tylko osoba posiadająca klucz prywatny może rozszyfrować te wiadomości. Przykładem szyfru asymetrycznego jest RSA.



Rys. 7. Szyfrowanie asymetryczne https://www.websiterating.com/pl/vpn/glossary/what-is-asymmetric-symmetric-encryption/

3.2. Możliwe rozwiązania

3.2.1. RSA

Algorytm RSA opiera się na matematycznych właściwościach dużych liczb pierwszych i faktoryzacji. Dzięki nim jest możliwe generowanie kluczy kryptograficznych składających się z pary liczb: publicznej i prywatnej. Klucz publiczny może być udostępniony każdemu, kto chce przesłać wiadomość do właściciela klucza prywatnego. Klucz prywatny natomiast powinien być przechowywany w tajemnicy i używany tylko przez właściciela, który może z niego korzystać do odczytania wiadomości przesłanej za pomocą klucza publicznego.
Generowanie klucza prywatnego i publicznego:

Wybieramy dwie duże liczby pierwsze p i q
Obliczamy ich iloczyn n = p * q
Obliczamy funkcję Eulera ϕ(n) = (p - 1)(q - 1)
Wybieramy liczbę e względnie pierwszą z ϕ(n), gdzie 1 < e < ϕ(n)
Obliczamy odwrotność modulo $\phi(n)$ dla e, czyli liczbę d, taką że: (e*d) $\equiv_{\phi(n)}$ 1

Klucz publiczny to para (n, e), a klucz prywatny to para (n, d)
Wiadomość jest szyfrowana za pomocą klucza publicznego (n, e) zgodnie z wzorem: c $\equiv_n m^e$
Zaszyfrowana wiadomość c jest deszyfrowana za pomocą klucza prywatnego (n, d) zgodnie z wzorem: m $\equiv_n c^d$

Algorytm RSA jest bardzo bezpieczny, ponieważ faktoryzacja dużych liczb pierwszych jest bardzo trudna do wykonania. Jednakże, gdy klucz jest zbyt krótki, to atakujący mogą spróbować przełamać klucz w krótkim czasie,

3.2.2. Algorytm Diffiego-Hellmana

Algorytm Diffiego-Hellmana służy do uzgodnienia klucza szyfrującego. Aby go ustalić na początku obie strony muszą ustalić podstawę g oraz pewną liczbę pierwszą p. Następnie każda ze stron generuje swoją własną liczbę pierwszą nazwyną dalej a i b.

Potem przesyłają sobie odpowiednio:

$$A = g^a(modp)$$

$$B = g^b(modp)$$

Następnie wyliczamy s:

$$s_1 = B^a(modp)$$

$$s_2 = A^b(modp)$$

Z własności przemienności mnożenia wiadomo, że s1 = s2
Uzyskaliśmy zatem nasz cel obie osoby uzyskały ten sam klucz.

Siła tego algorytmu polega na trudności obliczenia logarytmu dyskretnego w ciałach skończonych. Najszybszy algorytm (sito ciała liczbowego) obliczania logarytmu dyskretnego w GF(p) ma złożoność czasową:

$$e^{c*log_2^{1/3}*(p)*log_2^{2/3}*(log_2(p))}$$

Algorytm ten jest jednak podatny na ataki man in the middle, przykładowo: Alicja chce wysłać wiadomość do Boba. Alicja nie wie jednak, że pod Boba podszywa się Ewa. W momencie otrzymania wiadomości Ewa podszywa się również pod alicję. Wysyła Bobowi swoje a_2 i alicji swoje b_2 dzięki czemu jest w stanie ustalić z nimi dwoma klucze i być pośrednikiem w ich konwersacji.

3.2.3. Szyfr Rabina

Algorytm Rabina jest szyfrem asymetrycznym, którego bezpieczeństwo oparte jest na trudności obliczenia pierwiastków kwadratowych modulo liczba złożona. Kluczem tajnym są dwie duże liczby pierwsze p i q, wybrane w taki sposób, że p $\equiv$ 3 (mod 4) oraz q $\equiv$ 3 (mod 4).

Generowanie klucza polega na wybraniu losowo dwóch dużych liczb pierwszych p i q. Następnie naszym kluczem jest wartość n = pq. Następnie, aby zaszyfrować tekst, należy wykonać operację:

$$c = mod^2(modn)$$

W celu zdeszyfrowania wiadomości należy obliczyć pierwiastki kwadratowe liczby c (mod p) i liczby c (mod q), czyli:

$$m_p = c^{(p+1)/4}(modp)$$

$$m_q = c^{(q+1)/4}(modq)$$

Następnie obliczamy yp i yq korzystając z rozszerzonego algorytmu Euklidesa.
Liczymy:

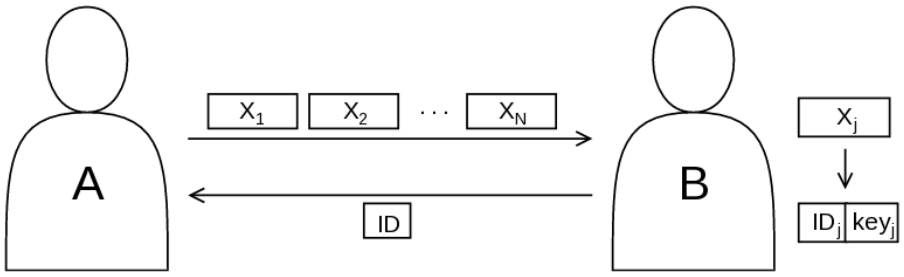
$$r = (y_p p m_q + y_q q m_p) modn$$

$$s = (y_p p m_q - y_q q m_p) modn$$

+-r i +-s są czterema pierwiastkami kwadratowymi z c (mod n) w zbiorze 0,1,...,n-1. Jednym z tych pierwiastków jest zaszyfrowana wiadomość m.

3.2.4. Puzzle Merkle'a

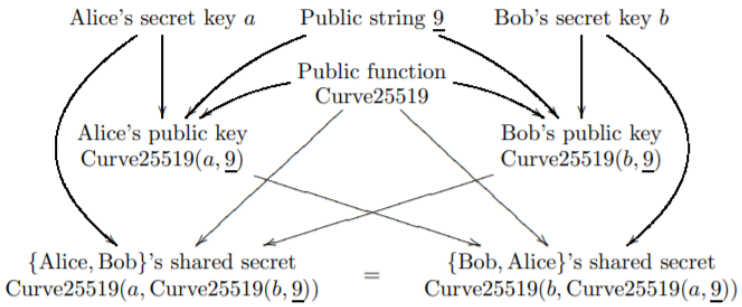
Algorytm Puzzli Merkle'a pozwala na ustalenie i współdzielenie sekretnego klucza pomiędzy dwiema jednostkami. Dany klucz może być następnie wykorzystany do szyfrowania waidomości algorytmami symetrycznymi. Sam algorytm polega na tym, że najpierw jedna ze stron (Alice) wysyła drugiej stronie(Bob) wiele wiadomości- z czego każda składa się z indetyfikatora i unikalnego dla tej wiadomości klucz. Te wiadomości są zaszyfrowane jakimś słabym szyfrem symetrycznym, każda innym. Następnie Bob odszyfrowuję jedną z wiadomości za pomocą ataku siłowego i wysyła identyfikator z powrotem do Alice. Teraz Alice wie, która z wielu wiadomości została odszyfrowana, a więc zna też do niej przypisany unikatowy klucz, Bob również zna ten klucz. Oboje dzielą jeden unikatowy, sekretny klucz.



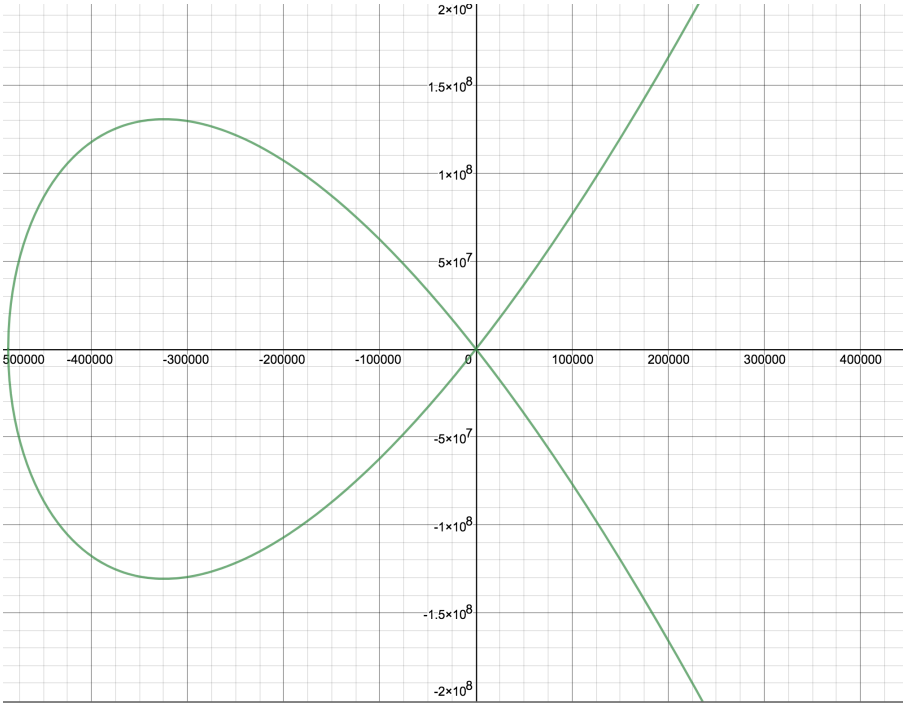
Rys. 8. Puzzle Merkle'a `https://manchestersiam.wordpress.com/2016/01/29/public-key-cryptography-merkles-puzzles/`

3.2.5. Curve25519

Algorytm asymetryczny, który opiera się na krzywej eliptycznej opartej równaniem $y^2 = x^3 + 486662x^2 + x$. Algorytm wykorzystuje tę krzywą do generowania kluczy publicznych i prywatnych, które składają się z 32-bajtowych wartości.
Curve25519 jest algorytmem klasyfikowanym jako bezpieczny według standardów NIST i został przebadany przez liczne organizacje i ekspertów w dziedzinie kryptografii. Jest on szybki, wydajny i działa na wielu platformach, w tym na systemach wbudowanych i urządzeniach mobilnych.
Jedną z cech algorytmu Curve25519, która czyni go lekkim, jest jego minimalna długość klucza. Wynosi ona zaledwie 256 bitów, co oznacza, że generowanie i przechowywanie kluczy jest bardziej efektywne w porównaniu do innych algorytmów asymetrycznych, takich jak RSA czy DSA.



Rys. 9. Algorytm Curve25519 `https://manchestersiam.wordpress.com/2016/01/29/public-key-cryptography-merkles-puzzles/`



Rys. 10. Algorytm Curve25519 `https://asecuritysite.com/ecdh`

3.3. Narzędzia

Do realizacji dalszych etapów projketu będą nam potrzebne kolejne narzędzia. W tej sekcji przedstawimy, według nas przydatne narzędiza, którymi będziemi się posługiwać oraz omówimy ich działanie.

3.3.1. Quartus

Jest to oprogramowanie, które pozwala nam na zaprogramowanie logiki naszego urządzenia. Umożliwia nam także analizę i syntezę designu HDL. Jest użyteczne także przy przydziału do układów logicznych, a także do łączenia ich. Pozwala na symulację i analizę czasową oraz na zarządzaniem użyciem mocy i na programowanie układu FPGA. W programie Quartus posługujemy się językiem HDL - Verilog.

3.3.2. Wolfram Mathematica

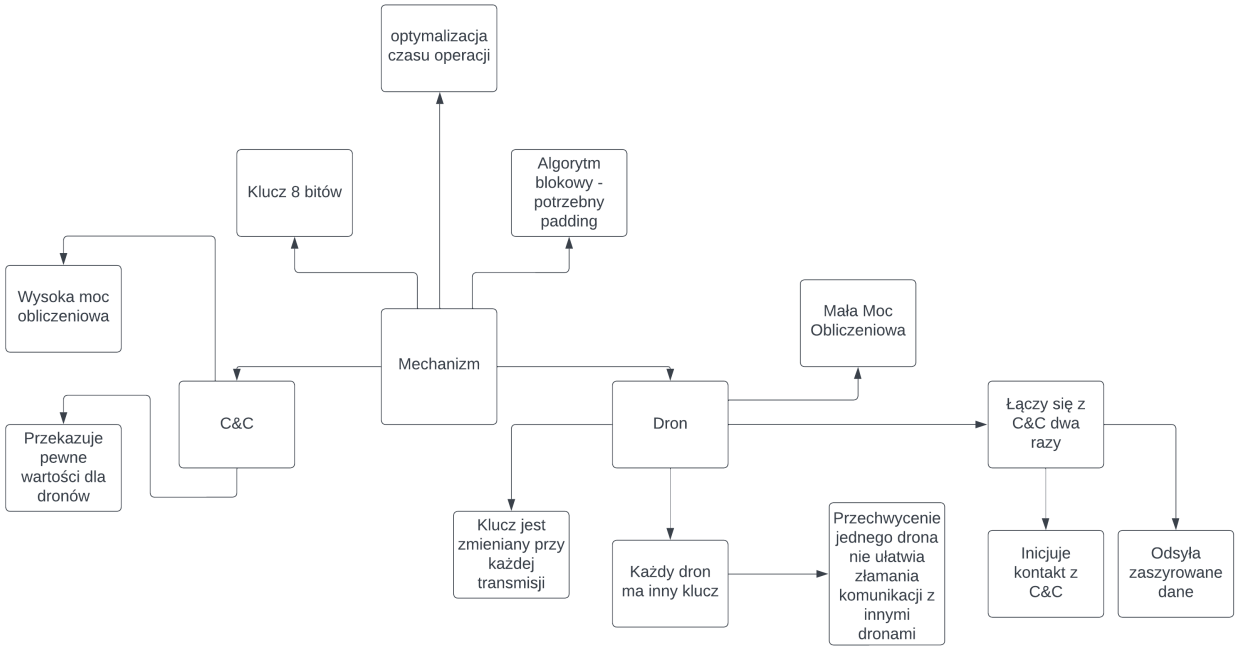
Oprogramowanie do obliczen symbolicznych i numerycznych. Będziemy używać tego programu do obliczeń i przystępniejszego prezentowania wykresów, danych.

3.3.3. Python

Użyjemy tego języka programowania w celu zaprojektowania modelu referencyjnego naszego projektu.

4. Koncepcja

4.1. Mapa myśli



Rys. 11. Mapa myśli

4.2. Wybór algorytmu

4.2.1. Curve25519

Algorytm używający krzywej eliptycznej do generowania kluczy nie został przez nas wybrany z uwagi na to, że według jego specyfikacji generuje on klucze wielkości 32-bajtów (256bitów) co jest zdecydowanie większe od 8bitów klucza podanych w zadaniu.

4.2.2. Puzzle Merkle'a

Algorytm ten wymaga utworzenia wielu szyfrogramów a następnie odszyfrowania jednego z nich atakiem siłowym. Problem w tym że chcemy żeby nasz klucz był przesyłany jak najmniejszą ilością informacji jak również uzyskiwane w miarę małym kosztem obliczeń. Atak siłowy na szyfrogramy wymagałby wielu obliczeń zatem niezdecydowaliśmy się na implementację tego algorytmu do ustalenia klucza.

4.2.3. Szyfr rabina

O ile nie wymaga od drona skomplikowanych obliczeń, tak po stronie C&C jest już trochę gorzej w dodatku wiemy tylko, że jedna z odszyfrowanych wiadomości jest wiadomościa faktyczną co w wypadku gdyby logi były przechowywane po stronie C&C utrudniałoby to ich odczytywanie i parsowanie.

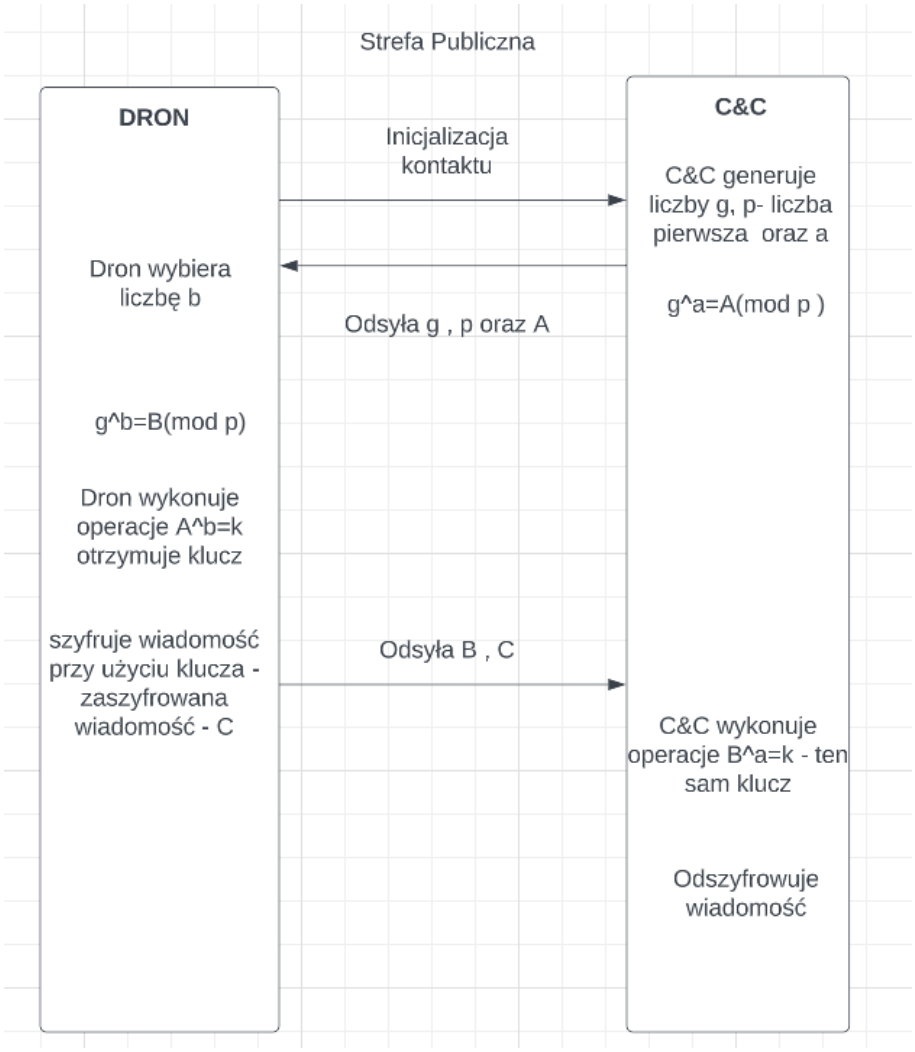
4.2.4. RSA

Podobnie jak w szyfrze rabina o ile po stronie drona nie musiałoby zostać wykonanych wiele obliczeń tak po stronie C&C musielibyśmy wygenerować dwie liczby pierwsze następnie policzyć tocjent i liczbę względnie pierwszą do tocjenta. W tym projekcie zdecydowaliśmy się na optymalizację pod kątem prędkości wykonywania operacji odczytu i zapisu danych dlatego zdecydowaliśmy się nie użyć tego algorytmu.

4.2.5. Diffie-Hellman

Ostatecznie zdecydowaliśmy się na użycie tego algorytmu aby wygenerować klucz szyfrujący dane. Przede wszystkim po obu stronach wykonujemy tylko operacje generowania liczb (jednej pierwszej w przypadku c&c) i operacji modulo. Jedną z zalet tego algorytmu nad RSA jest minimalnie trudniejsze odszyfrowanie wartości "a" od odszyfrowania wartości "d" Uznaliśmy, że będzie on najszybszy i postanowiliśmy zastosować go w naszym mechanizmie ustalenia klucza.

4.3. Schemat blokowy



Rys. 12. Schemat blokowy komunikacji pomiędzy dronem a C&C

4.4. Przykład ilustrujący działanie

— Dron wysyła sygnał z prośbą o przesłanie potrzebnych liczb do wymiany klucza przy użyciu Protokołu Diffiego Hellmana

— C&C generuje liczby : liczba pierwsza p: 127 - 0111 1111, liczba g: 109 - 0110 1101, liczba a : 5, następnie wykonuje operacje $g^a \equiv_p A$, dla podanych A wyniesie : 65

— C&C wysyła do drona wartości g , p oraz A czyli kolejno 109, 127 oraz 65

— Dron wybiera liczbę b - na przykład 7, po czym wykonuje operacje $g^b \equiv_p B$, dla tych naszych liczb B wyniesie : 105 - 0110 1001

— Po czym wykonuje operacje $A^b \equiv_p k$, dla tych naszych liczb klucz k wyniesie : 28 - 0001 1100

— Dron zamienia wiadomość na system binarny i dzieli ją na bloki 8 bitowe, w przypadku gdy blok nie jest podzielny przez osiem wykorzystujemy padding i uzupełniamy ostatni blok zerami. Wiadomość którą

przekazuje dron : ENEMY 21N 15E, na system binarny wyniesie: 01000101 01001110 01000101 01001101 01011001 00100000 00110010 00110001 01001110 00100000 00110001 00110101 01000101

— Dla przykładu użyjemy zaszyfrowania wiadomości operacją XOR, klucz k : 0001 1100, wiadomość zaszyfrowana: 01011001 01010010 01011001 01010001 01000101 00111100 00101110 00101101 01010010 00111100 00101101 00101001 01011001

— Teraz dron odsyła do C&C B czyli, 105 - 0110 1001 oraz zaszyfrowaną wiadomość C - 01011001 01010010 01011001 01010001 01000101 00111100 00101110 00101101 01010010 00111100 00101101 00101001 01011001

— C&C wykonuje operację $B^a \equiv_p k$, otrzyma klucz k - 28, jak widzidmy jest ten sam, czyli wszystko się zgadza,

— Teraz przy użyciu klucza odszyfrowuje wiadomość operacją XOR, wynik operacji:01000101 01001110 01000101 01001101 01011001 00100000 00110010 00110001 01001110 00100000 00110001 00110101 01000101 i po przekonwertowaniu na tekst otrzymuje informacje o wrogu: ENEMY 21N 15E

W tym rozdziale (i podrozdziałach) należy opisać zadania zrealizowane w ramach Etapu`III`. Może on zawierać mapę myśli ilustrującą burzę mózgów, która doprowadziła do opracowania koncepcji. Konieczne są natomiast:

— schemat blokowy koncepcji rozwiązania i jego omówienie,
— przykład ilustrujący działanie,
— model referencyjny w Python, MATLAB/GNU Octave, itp),
— danych do testowania i opis scenariuszy testowych.

4.5. Model referencyjny

Model referencyjny systemu został wykonany w języku programowania Python. Model składa się z dwóch plików:
- C&C
- Drone
Sposób implementacji jest następujący:

Krok 1: Dron wysyła do C&C prośbę o inicjalizację kontaktu i w tym samym czasie C&C generuje liczby g, p oraz A, które będą potrzebne do protokołu Diffiego-Hellmana, a następnie wysyła je do drona. Funkcja za to odpowiadająca do connect(), która przy odpowiednim inpucie ("initialize") generuje wyżej wymienione liczby. Funkcja primRoots() służy do wyszukiwania pierwiastków pierwotnych, potrzebnych do protokołu.

```
def primRoots(modulo):
    required_set = {num for num in range(1, modulo) if bltin_gcd(num, modulo) }
    return [g for g in range(1, modulo) if required_set == {pow(g, powers, modulo)
            for powers in range(1, modulo)}]

def connect(input):
    if input == "initialize":
        return create()


def create():
    global g, a, A
    idP = random.randint(0, len(primeNumbers) - 1)
    p = primeNumbers[idP]
    gAll = primRoots(p)
    g = gAll[random.randint(0, len(gAll)) - 1]
    a = random.randint(2,5)
    A = pow(g, a) % p
    return p, g, A
```

Rys. 13. connect(), create(), primRoots()

Krok 2: Dron oblicza b i B za pomocą funkcji createb() i createB() oraz uzyskuje klucz przy pomocy funkcji createk(). Po uzyskaniu tych wartości, szyfruje wiadomość (C) używając funkcji createC(), która polega na XOR'owaniu klucza z podaną wiadomością, którą dron chce wysłać do C&C.

```
def createb():
    global b
    b = p
    while(b == p):
        b = primeNumbers[random.randint(0, len(primeNumbers) - 1)]

def createB():
    global B
    global g
    B = pow(g, b) % p

def createk():
    global k
    k = pow(A, b)

def createC():
    global C
    C = XOR(k, data)
    return C
```

Rys. 14. createb(), createB(), createk(), createC()

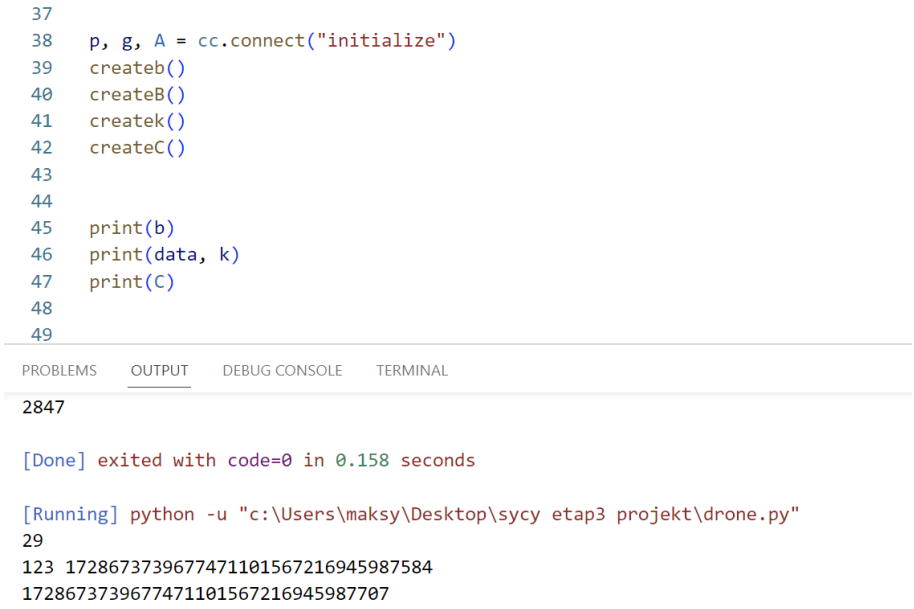
Krok 3: Dron odsyła informacje do C&C (które C&C odbiera za pomocą funkcji getBC() return B, C). Następnie C&C oblicza klucz przy pomocy createk() i odszyfrowuje wiadomość używając funkcji createData()

```
def createk():
    global k
    k = pow(B, a)

def createData():
    global data
    data = XOR(k, C)
    return data

def XOR(a,b):
    return a ^ b
```

Rys. 15. createk(), createData(), XOR()

4.6. Przykładowe dane do testowania

Poniższy rysunek przedstawia działanie modelu referencyjnego dla losowo wygenerowanych danych.



Rys. 16. Wyniki dla danych testowych

5. Implementacja

5.1. Moduł World

Moduł world łączy ze sobą wszystkie wcześniej wymienione moduły i pozwala na komunikację. między nimi.

```
module world (
        input           clk,
        input       ena,
        input       start,
        input       rst,
        input [63:0] message_from_drone,
        output [7:0] key,
        output [63:0] decrypted_message
);

//Zmienne do cc
//wire start_cc;
wire [63:0] encrypted_message;
wire [7:0] B_part_key_from_drone;
wire message_is_encrypted;

//Zmienne do drona
wire [7:0] A_part_key_from_drone;
wire [7:0] g_from_cc;
wire [7:0] p_from_cc;
wire values_are_ready;

cc base(
        .initiate (start),
        .rst    (rst),
        .clk    (clk),
        .ena    (ena),
        .B_part_key (B_part_key_from_drone),
        .rdy_drone (message_is_encrypted),
        .c (encrypted_message),
        .A_part_key_output (A_part_key_from_drone),
        .cc_rdy (values_are_ready),
        .g_output  (g_from_cc),
        .p_output  (p_from_cc),
        .decrypted (decrypted_message)

);

drone sea_dron(
        .initiate  (start),
```

```
        //.cc_contact (start_cc),
        .rst (rst),
        .clk (clk),
        .ena (ena),
        .message (message_from_drone),
        .rdy_cc  (values_are_ready),
        .A_part_key (A_part_key_from_drone),
        .g (g_from_cc),
        .p (p_from_cc),
        .B_part_key_output (B_part_key_from_drone),
        .messageEncrypted_output(encrypted_message),
        .confirmation_of_encryption (message_is_encrypted)

);

endmodule
```

5.1.1. Moduł C&C

C&C odbiera inicjalizacje kontaku od Drona, a następnie generuje trzy liczby-g,a oraz p. Liczby g oraz a są 8-bitowymi liczbami losowymi, które są generowane za pomocą modułu random_generator. Liczba p jest 8-bitową liczbą pierwszą generowaną za pomocą modułu primenumber.

Po wygenerowaniu owych trzech liczb, C&C przechodzi do wygenerowania A za pomocą wcześniej wygenerowanych g,a oraz p. Robi to za pomocą modułu powermod, który bierze liczbę g do potęgi a, modulo p.

Po wykonaniu tych czterech operacji, C&C odsyła A, g oraz p do Drona.

Po jakimś czasie, odbiera od drona liczbę B, oraz zaszyfrowaną wiadomość. Moduł C&C następnie generuje klucz również za pomocą modułu powermod, jako że nasz klucz to B do potęgi a, modulo p.

Na koniec, C&C odszyfrowuje zaszyfrowaną wiadomość , za pomocą funkcji XOR oraz modułu codeMessage, wiadomość używając 8-bitowego klucza.

Moduł odpowiada za funkcjonalność modelu Command and Control, uruchamiając określone, wyżej wymienione moduły.

```
module cc (
input initiate,  //SygnaĆ z inicjalizacj  kontatku
input rst,
input clk,
input ena,
input [7:0] B_part_key, // Komponent klucza ze strony drona
input rdy_drone,
input [63:0] c, //Zaszyfrowana wiadomo Ż
output [7:0] A_part_key_output, //Komponent klucza ze strony cc
output cc_rdy,
output [7:0] g_output,
output [7:0] p_output,
output [63:0] decrypted
);
        //Stany automatu
        localparam SIZE = 3;
        localparam [SIZE-1:0] idle = 3'h0,  // oczekiwanie
                                                        lfsr_generate1   = 3'h1, //Wygenerowania liczby a
                                                        lfsr_generate2   = 3'h2,//Wygenerowanie liczby g
                                                        prime_number   =   3'h3,//Wybranie liczby pierwszej
                                                        powmod_A   =  3'h4,//PotŹgowanie modularne A
                                                        wait_for_drone = 3'h5,//Czekanie a  dron odeŹle dane
                                                        get_key = 3'h6,//Wygenerowanie klucza
                                                        message_decription = 3'h7; //Odzyfrowanie wiadomo Ż i

        //Rejestr stan w
        reg[SIZE-1:0] state_reg, state_next;

        //Zmienne do potŹgowania
        wire [7:0] g, a, p, key, A_part_key;
        wire rdy_random_a, rdy_random_g, rdy_prime, rdy_pow_A, rdy_pow_key, rdy_decrypt;
        reg  start, start_random_g, start_prime, start_pow_A, start_pow_key,
    start_decryption, cc_ready_reg, drone_reg;
        reg [7:0] a_reg, g_reg, p_reg, A_part_key_reg, key_reg;


        //Used modules
random_generator g_1 (
                .rst      (rst),
                .clk      (clk),
                .start (start),
                .ena      (ena),
                .valuea(a),
                .rdy_random(rdy_random_a)
);
random_generator1 g_2 (
                .rst      (rst),
                .clk      (clk),
                .start (start_random_g),
                .ena      (ena),
                .valuea(g),
                .rdy_random(rdy_random_g)
);

primenumber p_1 (
```

```
                .clk        (clk),
                .index    (g_reg),
                .start (start_prime),
                .data_out(p),
                .rdy_prime (rdy_prime)
);

powermod A_1 (
                .rst      (rst),
                .clk      (clk),
                .ena      (ena),
                .start (start_pow_A),
                .a                (g_reg),
                .b                (a_reg),
                .m                (p_reg),
                .res    (A_part_key),
                .rdy    (rdy_pow_A)
);
powermod key_creation (
                .rst      (rst),
                .clk      (clk),
                .ena      (ena),
                .start (start_pow_key),
                .a                (B_part_key),
                .b                (a_reg),
                .m                (p_reg),
                .res    (key),
                .rdy    (rdy_pow_key)
);
codeMessage decryption (
                .clk    (clk),
                .ena    (ena),
                .letter(c),
                .key    (key_reg),
                .confirm (start_decryption),
                .codedMessage (decrypted),
                .done   (rdy_decrypt)
);

// State register
  always@(posedge clk, posedge rst) begin
    if (rst) begin
      state_reg <= idle;

    end
    else begin
      state_reg <= state_next;

    end
  end

  // Registers
  always@(posedge clk, posedge rst) begin
    if (rst) begin
//     a_reg <=8'b00000000;
//        g_reg <=8'b00000000;
//        p_reg <=8'b00000000;
//        A_part_key_reg <=8'b00000000;
//        key_reg <=8'b00000000;
    end
    else begin
//       a_reg   <= a_next;
//       g_reg   <= g_next;
//       p_reg   <= p_next;
//      A_part_key_reg <= A_part_key_next;
    end
  end

  // Next state logic
  always@(*)
    case(state_reg)
      idle   : if (initiate) state_next = lfsr_generate1;
               else          state_next = idle;
      lfsr_generate1   : if(rdy_random_a) state_next = lfsr_generate2;
                                                       else state_next = lfsr_generate1;
                lfsr_generate2   : if(rdy_random_g) state_next = prime_number;
                                                       else state_next = lfsr_generate2;
      prime_number   : if(rdy_prime) state_next = powmod_A;
                                                       else state_next = prime_number;
      powmod_A : if(rdy_pow_A) state_next = wait_for_drone;
                                                  else state_next = powmod_A;
                wait_for_drone: if(drone_reg) state_next = get_key;
                                                       else state_next = wait_for_drone;
                get_key : if(rdy_pow_key) state_next = message_decription;
```

```
                message_decription : state_next = message_decription;
      default: state_next = idle;
    endcase

  // Microoperation logic
  always@(*) begin
//     a_next      = a_reg;
//     g_next      = g_reg;
//     p_next      = p_reg;
//     A_part_key_next    = A_part_key_reg;


    case(state_reg)
      idle:
                        begin

                        end
                lfsr_generate1: begin
                                                        start = 1;
                                                        if(rdy_random_a) begin
                                                                a_reg <= a;
                                                                end
                                                end
                lfsr_generate2: begin
                                                                //start = 0;
                                                                start_random_g <= 1;
                                                                if(rdy_random_g)begin
                                                                        g_reg <= g;
                                                                        end
                                                                end
                prime_number : begin
                                                                //start_random_g = 0;
                                                                start_prime = 1;
                                                                if(rdy_prime) begin
                                                                        p_reg <= p;
                                                                        end
                                                        end
                powmod_A : begin
                                                                start_prime = 0;
                                                                start_pow_A = 1;
                                                                if(rdy_pow_A) begin
                                                                        A_part_key_reg <= A_part_key;
                                                                        cc_ready_reg <= 1;
                                                                        end
                                                end
                wait_for_drone: begin
                                                                        start_pow_A = 0;
                                                                        if(rdy_drone) begin
                                                                                drone_reg <= 1;
                                                                        end
                                                                end
                get_key : begin
                                                drone_reg = 0;
                                                start_pow_key =1;
                                                if(rdy_pow_key) begin
                                                                key_reg <= key;
                                                end
                                        end
                message_decription : begin
                                                                        start_pow_key = 0;
                                                                        start_decryption = 1;
                                                                        if(rdy_decrypt)begin
                                                                                start_pow_key = 0;
                                                                        end
                                                                end


      default: ;
    endcase
  end

  assign A_part_key_output = A_part_key_reg;
  assign g_output = g_reg;
  assign p_output = p_reg;
  assign cc_rdy = cc_ready_reg;


endmodule
```

5.1.2. Moduł Powermod

Moduł służy do potęgowania modulo dwóch liczb.

```
module powermod (
  input          clk,
  input          rst,
  input          start,
  input                          ena,
  input  [7:0] a,
  input  [7:0] b,
  input  [7:0] m,
  output [7:0] res,
  output reg                rdy
);

  localparam SIZE = 3;
  localparam [SIZE-1:0] idle   = 3'h0,
                        init   = 3'h1,
                        loop   = 3'h2,
                        store  = 3'h3;

  reg [SIZE-1:0] state_reg, state_next;
  reg            rdy_next;

  reg  [7:0] a_reg, a_next;
  reg  [7:0] b_reg, b_next;
  reg  [7:0] m_reg, m_next;
  reg  [7:0] res_reg, res_next;

  // State register
  always@(posedge clk, posedge rst) begin
    if (rst) begin
      state_reg <= idle;
      rdy       <= 1'b0;
    end
    else begin
      state_reg <= state_next;
      rdy       <= rdy_next;
    end
  end

  // Registers
  always@(posedge clk, posedge rst) begin
    if (rst) begin
      a_reg <= 8'h0;
      b_reg <= 8'h0;
      m_reg <= 8'h0;
      res_reg <= 8'h0;
    end
    else begin
      a_reg   <= a_next;
      b_reg   <= b_next;
      m_reg   <= m_next;
      res_reg <= res_next;
    end
  end

  // Next state logic
  always@(*)
    case(state_reg)
      idle  : if (start) state_next = init;
              else       state_next = idle;
      init  : state_next = loop;
      loop  : if (b_reg == 0) state_next = store;
              else            state_next = loop;
      store : state_next = idle;
      default: state_next = idle;
    endcase

  // Microoperation logic
  always@(*) begin
    a_next    = a_reg;
    b_next    = b_reg;
    m_next    = m_reg;
    res_next  = res_reg;
    rdy_next  = 1'b0;

    case(state_reg)
      init  : begin
                a_next   = a;
                b_next   = b;
                m_next   = m;
                res_next = 1;
```

```
            end
      loop  : begin
                                                  if (b_reg[0] == 1) res_next = (a_reg * res_reg) % m_reg;
                                        a_next = (a_reg * a_reg) % m_reg;
                                        b_next = b_reg >> 1;
            end

      store : begin
                rdy_next   = 1'b1;
            end
      default: ;
    endcase
  end

  assign res = res_reg;

endmodule
```

5.1.3. Moduł generowania pseudolosowych liczb 8-bitowych

Moduł służy do generowania losowej liczby 8-bitowej za pomocą rejestru przesuwającego LFSR. Rejestr przesuwający LFSR jest deterministycznym generatorem liczb pseudolosowych. Jako, że rejestr ma skończoną liczbę stanów, to ostatecznie jednak wejdzie w powtarzający się cykl. To, jakie liczby wygeneruje i w jakiej kolejności zależy również od wielomianu. W naszym wypadku jest to $x^8 + x^6 + x^5 + x^4 + 1$.
Moduł generuje 2 liczby: a i g.

```
module random_generator(
    input rst,
    input clk,
    input ena,
         input start,
    output [7:0] valuea,
         output rdy_random
);

reg    [7:0] lfsr_reg;
wire [7:0] lfsr_next;
wire    feedback;
reg done;

always@(posedge clk, posedge rst) begin
    if(rst)
        lfsr_reg<=8'b10000000;
    else if(ena && start)
        lfsr_reg <= lfsr_next;
end

assign feedback = lfsr_reg[7] ^ lfsr_reg[5] ^ lfsr_reg[4] ^ lfsr_reg[3];
assign lfsr_next = {lfsr_reg[6:0],feedback};

always @(posedge clk or posedge rst) begin
    if (rst)
        done <= 1'b0;
    else if (ena && lfsr_reg == 8'b1) // Sprawdzamy, czy lfsr_reg osi gn Ć warto Ż 0
        done <= 1'b1;
    else
        done <= 1'b0;
end

assign valuea = lfsr_reg;
assign rdy
```

5.1.4. Moduł losowania liczb pierwszych

Moduł służy do generowania losowej liczby pierwszej poprzez losowe wybranie liczby z tablicy 53 8-bitowych liczb pierwszych.

```
module primenumber(
  input clk,
  input [7:0] index,
  input start,
  output reg [7:0] data_out,
  output rdy_prime
);

reg [7:0] array [0:52];
reg [5:0] randomindex;
reg done;

// Initialize the array
initial begin
                    randomindex = index % 53;
                    array[0] = 8'b00000010;
        array[1] = 8'b00000011;
        array[2] = 8'b00000101;
        array[3] = 8'b00000111;
        array[4] = 8'b00001011;
        array[5] = 8'b00001101;
        array[6] = 8'b00010001;
        array[7] = 8'b00010011;
        array[8] = 8'b00010111;
        array[9] = 8'b00011101;
        array[10] = 8'b00011111;
        array[11] = 8'b00100101;
        array[12] = 8'b00101001;
        array[13] = 8'b00101011;
        array[14] = 8'b00101111;
        array[15] = 8'b00110101;
        array[16] = 8'b00111011;
        array[17] = 8'b00111101;
        array[18] = 8'b01000011;
        array[19] = 8'b01000111;
        array[20] = 8'b01001001;
        array[21] = 8'b01001111;
        array[22] = 8'b01010011;
        array[23] = 8'b01011001;
        array[24] = 8'b01100001;
        array[25] = 8'b01100101;
        array[26] = 8'b01100111;
        array[27] = 8'b01101011;
        array[28] = 8'b01101101;
        array[29] = 8'b01110001;
        array[30] = 8'b01111111;
        array[31] = 8'b10000011;
        array[32] = 8'b10001001;
        array[33] = 8'b10001011;
        array[34] = 8'b10010101;
        array[35] = 8'b10010111;
        array[36] = 8'b10011101;
        array[37] = 8'b10100011;
        array[38] = 8'b10100111;
        array[39] = 8'b10101101;
        array[40] = 8'b10110011;
        array[41] = 8'b10110101;
        array[42] = 8'b10111111;
        array[43] = 8'b11000001;
        array[44] = 8'b11000101;
        array[45] = 8'b11000111;
        array[46] = 8'b11010011;
        array[47] = 8'b11100011;
        array[48] = 8'b11101001;
        array[49] = 8'b11101001;
        array[50] = 8'b11101111;
        array[51] = 8'b11110001;
        array[52] = 8'b11111011;
end

// Assign data_out based on the input index
always @(start) begin
randomindex = index % 53;
    case (randomindex)
        6'd0: array[0] = 8'b00000010;
        6'd1: array[1] = 8'b00000011;
        6'd2: array[2] = 8'b00000101;
        6'd3: array[3] = 8'b00000111;
```

```
        6'd4: array[4] = 8'b00001011;
        6'd5: array[5] = 8'b00001101;
        6'd6: array[6] = 8'b00010001;
        6'd7: array[7] = 8'b00010011;
        6'd8: array[8] = 8'b00010111;
        6'd9: array[9] = 8'b00011101;
        6'd10: array[10] = 8'b00011111;
        6'd11: array[11] = 8'b00100101;
        6'd12: array[12] = 8'b00101001;
        6'd13: array[13] = 8'b00101011;
        6'd14: array[14] = 8'b00101111;
        6'd15: array[15] = 8'b00110101;
        6'd16: array[16] = 8'b00111011;
        6'd17: array[17] = 8'b00111101;
        6'd18: array[18] = 8'b01000011;
        6'd19: array[19] = 8'b01000111;
        6'd20: array[20] = 8'b01001001;
        6'd21: array[21] = 8'b01001111;
        6'd22: array[22] = 8'b01010011;
        6'd23: array[23] = 8'b01011001;
        6'd24: array[24] = 8'b01100001;
        6'd25: array[25] = 8'b01100101;
        6'd26: array[26] = 8'b01100111;
        6'd27: array[27] = 8'b01101011;
        6'd28: array[28] = 8'b01101101;
        6'd29: array[29] = 8'b01110001;
        6'd30: array[30] = 8'b01111111;
        6'd31: array[31] = 8'b10000011;
        6'd32: array[32] = 8'b10001001;
        6'd33: array[33] = 8'b10001011;
        6'd34: array[34] = 8'b10010101;
        6'd35: array[35] = 8'b10010111;
        6'd36: array[36] = 8'b10011101;
        6'd37: array[37] = 8'b10100011;
        6'd38: array[38] = 8'b10100111;
        6'd39: array[39] = 8'b10101101;
        6'd40: array[40] = 8'b10110011;
        6'd41: array[41] = 8'b10110101;
        6'd42: array[42] = 8'b10111111;
        6'd43: array[43] = 8'b11000001;
        6'd44: array[44] = 8'b11000101;
        6'd45: array[45] = 8'b11000111;
        6'd46: array[46] = 8'b11010011;
        6'd47: array[47] = 8'b11100011;
        6'd48: array[48] = 8'b11101001;
        6'd49: array[49] = 8'b11101001;
        6'd50: array[50] = 8'b11101111;
        6'd51: array[51] = 8'b11110001;
        6'd52: array[52] = 8'b11111011;
        default: array[0] = 8'b00000000; // default case, if randomindex is out of range
    endcase
        data_out = array[randomindex];
        if(data_out != 0) begin
                done = 1'b1;
        end

end

assign rdy_prime = done;


endmodule
```

5.1.5. Moduł odszyfrowywania i zaszyfrowywania wiadomości

Moduł służy do odszyfrowywania wiadomości za pomocą funkcji XOR.

```
module codeMessage (
    input clk,
    input ena,
         input [63:0] letter,
         input [7:0] key,
         input confirm,
         output reg [63:0] codedMessage,
         output done
);

reg [7:0] tempCode;
reg done_reg;
reg [63:0]long_key;
//reg [7:0] currentLetter;

initial begin
        codedMessage <= 64'b0; // reset codedMessage to 0
        tempCode <= 8'b0; // reset tempCode to 0

end


always @(posedge clk) begin
                long_key <= {key, key, key, key, key, key, key, key};
        if ( confirm == 1) begin
                                //currentLetter <= letter;
            codedMessage <= letter ^ long_key; // XOR the letter and key
            //codedMessage <= {codedMessage, tempCode}; // add the 8-bit code to codedMessage
                                done_reg <= 1;
        end
    end

 assign done = done_reg;
endmodule
```

5.1.6. Moduł Dron

Dron inicjalizuje kontakt z C&C.

Następnie, otrzymuje liczby g, p oraz A od C&C. Za pomocą modułu random_generator generuje losową, 8-bitową liczbę b.

Kolejnym krok, jaki wykonuje, to za pomocą modułu powermod tworzy liczbę B. Liczba to g do potęgi b, modulo p (Liczby g oraz p otrzymał wcześniej od C&C). Aby otrzymać 8-bitowy klucz, podnosi A do potęgi b, modulo p. (Liczbę A otrzymał wcześniej od modułu C&C).

Na końcu, za pomocą 8-bitowego klucza, funkcji XOR oraz modułu codeMessgae,zaszyfrowuje wiadomość, a następnie wysyła ją oraz B do modułu C&C.

Do realizacji użyto również wcześniej wymienionych modułów:
- powermod x2
- codeMessage
- random_generator

```
module drone (
input initiate, //SygnaĆ z inicjalizacj kontatku
//output cc_contact,
input rst,
input clk,
input ena,
input [63:0]message,
input rdy_cc,
input [7:0] A_part_key, // Komponent klucza ze strony drona
input [7:0] g,
input [7:0] p,
output [7:0] B_part_key_output, //Komponent klucza ze strony cc
output [63:0] messageEncrypted_output, // wiadomoŻ zaszyfrowana
output [7:0] confirmation_of_encryption
);
        //Stany automatu
        localparam SIZE = 3;
        localparam [SIZE-1:0] idle = 3'h0,  // oczekiwanie
                                                get_random_b   = 3'h1, //Wygenerowania liczby b
                                                wait_for_cc = 3'h2,
                                                get_B_get_key = 3'h3,//Czekanie a  cc odeŹle dane
                                                key_creation = 3'h4,
                                                encryption   = 3'h5,//PotŹgowanie modularne B
                                                send_to_cc = 3'h6;

        //Rejestr stan w
        reg[SIZE-1:0] state_reg, state_next;

        //Zmienne do potŹgowania
        wire [7:0] b, B_part_key, key;
        wire [63:0] messageEncrypted;
        wire rdy_random_b, rdy_pow_key, rdy_message_C, rdy_pow_B;
        reg rdy_cc_reg, start_key_creation, start_pow_B, start_encription, start_rand;
        reg [63:0] messageEncrypted_reg;
        reg [7:0] b_reg;
        reg [7:0] B_reg;
        reg [7:0] K_reg;


        //Used modules
random_generator2
        dronb (
                .rst     (rst),
                .clk     (clk),
                .start (start_rand),
                .ena     (ena),
                .valuea(b),
                .rdy_random(rdy_random_b)
);


powermod B_dron (
                .rst     (rst),
```

```
                .clk      (clk),
                .ena      (ena),
                .start (start_pow_B),
                .a                (g),
                .b                (b_reg),
                .m                (p),
                .res    (B_part_key),
                .rdy    (rdy_pow_B)
);


powermod key_creation_mod (
                .rst      (rst),
                .clk      (clk),
                .ena      (ena),
                .start (start_key_creation),
                .a                (A_part_key),
                .b                (b_reg),
                .m                (p),
                .res    (key),
                .rdy    (rdy_pow_key)
);

codeMessage  code_dron(
      .clk (clk),
      .ena (ena),
           .letter (message),
           .key (K_reg),
                .confirm (start_encription),
           .codedMessage (messageEncrypted),
                .done (confirmation_of_encryption)
);


// State register
  always@(posedge clk, posedge rst) begin
    if (rst) begin
      state_reg <= idle;

    end
    else begin
      state_reg <= state_next;

    end
  end

  // Registers
  always@(posedge clk, posedge rst) begin
    if (rst) begin
//                b_reg<=8'b00000000;
//                B_reg<=8'b00000000;
//                K_reg<=8'b00000000;
//                Message_reg<=8'b00000000;

    end
    else begin
//        b_reg<=b_next;
//        B_reg<= B_next;
//        K_reg <= K_next;
//        Message_reg<= Message_next;
    end
  end

  // Next state logic
  always@(*)
    case(state_reg)
      idle  : if (initiate) state_next = get_random_b;
              else          state_next = idle;
                get_random_b : if(rdy_random_b) state_next = wait_for_cc;
                               else         state_next = get_random_b;
           wait_for_cc   : if(rdy_cc_reg) state_next = get_B_get_key;
                                                                  else state_next = wait_for_cc;
      get_B_get_key: if(rdy_pow_B) state_next = key_creation;
                                                                  else state_next = get_B_get_key;
                key_creation : if(rdy_pow_key) state_next = encryption;
                                                                  else state_next = key_creation;
      encryption    : if(messageEncrypted) state_next = send_to_cc;
                                                                  else state_next = encryption;
                send_to_cc    : //if(message_sent)
                                                                  state_next = send_to_cc;
                                                                  //else state_next = send_to_cc;
      default       : state_next = idle;
    endcase
```

```
  // Microoperation logic
  always@(*) begin
//     a_next     = a_reg;
//     g_next     = g_reg;
//     p_next     = p_reg;
//     A_part_key_next   = A_part_key_reg;


    case(state_reg)
      idle:
                        begin

                        end
                get_random_b : begin
                                                        start_rand = 1;
                                                        if(rdy_random_b) begin
                                                                b_reg <= b;
                                                                end
                                                end
                wait_for_cc  : begin
                                                        start_rand = 0;
                                                        if(rdy_cc) begin
                                                                rdy_cc_reg =1;
                                                        end
                                                end
                get_B_get_key: begin
                                                        rdy_cc_reg = 0;
                                                        start_pow_B = 1;
                                                        if(rdy_pow_B) begin
                                                                B_reg <= B_part_key;
                                                        end
                                                end
                key_creation : begin
                                                                start_pow_B = 0;
                                                                start_key_creation = 1;
                                                                if(rdy_pow_key) begin
                                                                        K_reg <= key;
                                                                end
                                                end
                encryption   : begin
                                                        start_key_creation = 0;
                                                        start_encription = 1;
                                                        if(confirmation_of_encryption) begin
                                                                messageEncrypted_reg <= messageEncrypted;
                                                        end
                                                end
                send_to_cc   : begin

                                                end
      default       : ;
    endcase
  end

  assign B_part_key_output = B_reg;
  assign messageEncrypted_output = messageEncrypted_reg;
endmodule
```

5.1.7. Symulacja

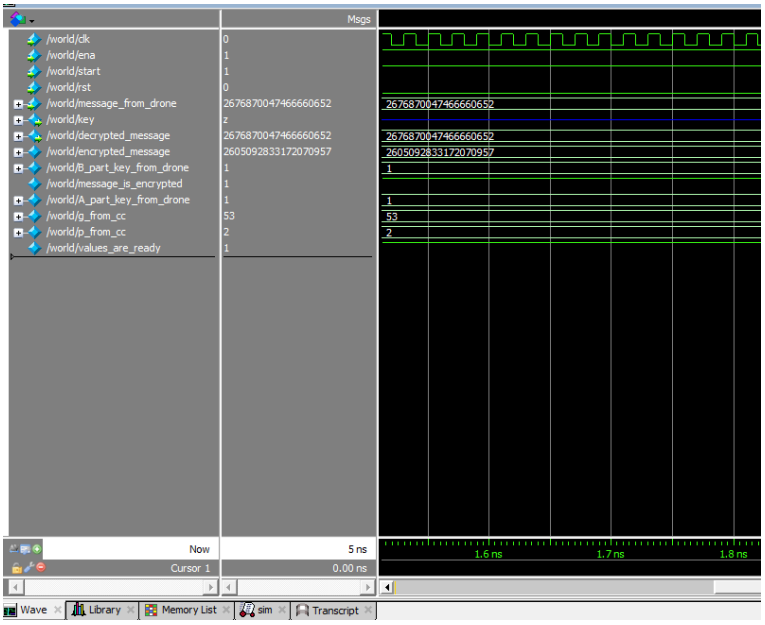
Symulacja przeprowadzana jest za pomocą ModelSim, który jest częścią pobranej przez nas wersji Quartus'a. Aby odpowiednio przeprowadzić symulacje za pomocą tego narzędzie, należy najpierw skompilować pliki w Quartusie, a następnie wybrać Tools - Run Simulation Tool - RTL Simulation. W nowo otwartym oknie należy wybrać moduł, który chcemy przetestować, a następnie utworzyć plik .do z odpowiednio przygotowanymi parametrami. Symulacje odpalamy wpisując do nazwapliku.do- istotne wykresy symulacji pojawią się w okienku Wave.

Plik .do z potrzebnymi parametrami do uruchomienia symulacji modułu world. Plik ten inicjuje takie parametry jak: clk, rst, ena i start oraz co ważniejsze message_from_drone. Parametr message_from_drone to jeszcze nie zaszyfrowana wiadomość w postaci liczby binarnej. Z założenia, pewna wiadomość jest szyfrowana przez Drona, za pomocą 8-bitowego klucza i funkcji XOR, a następnie wysyłana do C&C.

```
Ln#
1   restart -nowave -force
2   add wave -radix unsigned *
3   force clk 0 0, 1 10 -r 20
4   force rst 1 0, 0 1
5   force ena 1
6   force start 1
7   force message_from_drone 00100101001001100010011100100000001000010010001000100001100101100
8
9   run 5000
10
11
```

Rys. 17. Plik .do o odpowiednich parametrach potrzebnych do symulacji modułu world

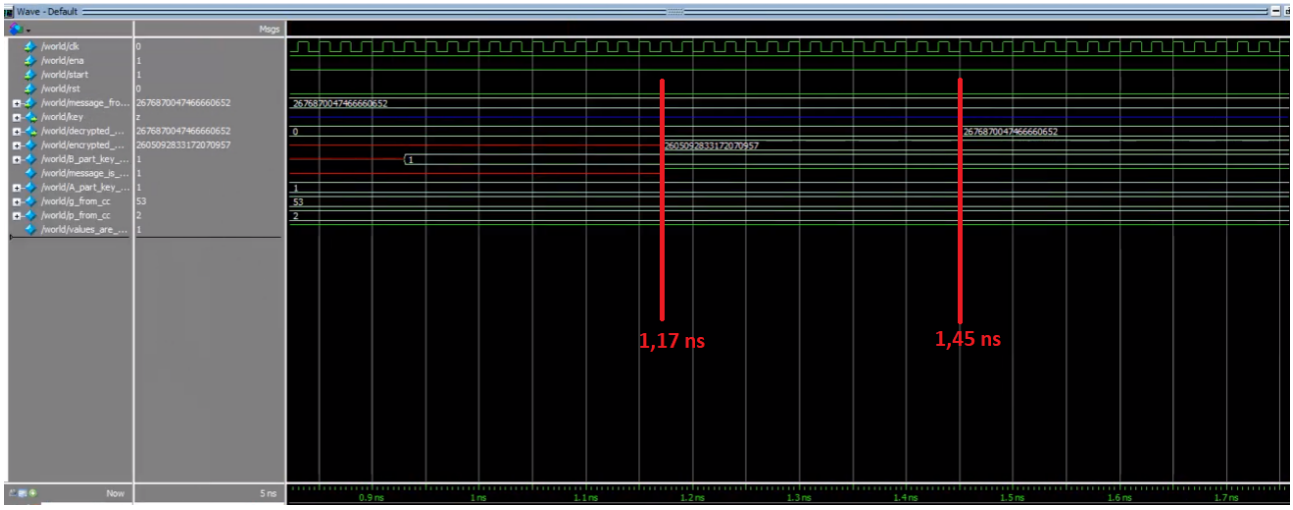
Tak wygląda symulacja modułu world. Jak widać, wiadomość została poprawnie zaszyfrowana i odszyfrowana. W symulacji można też zobaczyć takie parametry jak klucz, g, p.



Rys. 18. Symulacja modułu world

5.2. Szybkość działania

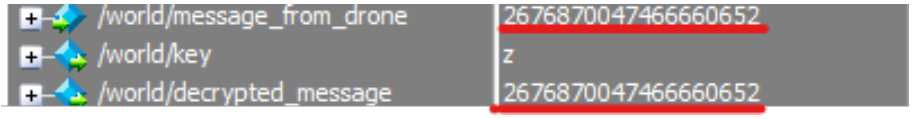
Działanie naszego programu można podzielić na 2 etapy:
- Przygotowanie wiadomości i zaszyfrowanie jej w dronie.
- Wysłanie jej do C&C i odszyfrowanie.
Pierwsza faza, czyli zaszyfrowanie wiadomości w dronie odbywa się w czasie 1,17 nanosekund.
Druga faza, czyli przekazanie zaszyfrowanej wiadomości do drona i odszyfrowanie jej zajmuje 0,28 nanosekund.
Podsumowując cały program realizuje swoje rozwiązanie w przeciągu 1,45 nanosekund.



Rys. 19. Czasy w symulacji

5.3. Weryfikacja funkcjonalna

Jak widać na załączonym screenshot'cie wiadomość przed zaszyfrowaniem oraz odszyfrowana wiadomość są takie same.



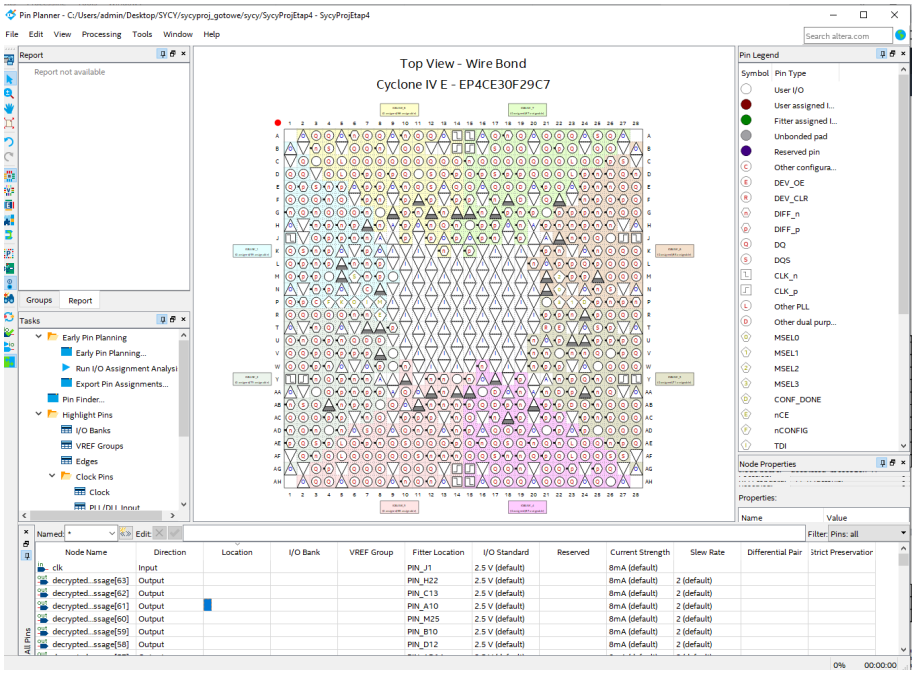
Rys. 20. Weryfikacja funkcjonalności

5.4. Ocena rozwiązania

Rozwiązanie w pełni realizuje zadanie projektowe, wszystkie kroki działają poprawnie, wiadomości są szyfrowane w bezpieczny sposób, a komunikacja przebiega prawidłowo. Dodatkowo atutem naszego rozwiązania jest jego szybkie działanie (1,45 nanosekund).

6. Uruchomienie

Uruchamiamy nasz system na płytce DE2-115. Początkowo należy przipsać wyjściom i wejściom odpowiednie piny.Znajduje się to odpowiednio w zakładce **Assignments− > Pin Planner**.



Rys. 21. Pin Planner

Aby odpowiednio dopasować piny należy skorzystać z manuala, który można znaleźć w internecie. Tam jest opisane, które piny są przypisane do przycisków znajdujących się na naszej docelowej płytce.

Inputom rst oraz start przydzieliliśmy odpowiednio pushbutton[0] oraz pushbutton[1]. Dodatkowo pierwszym 18 bitom odpowiedającym wiadomości od drona przypisaliśmy pierwsze 18 switchy. Na płytce nie będzie wykorzystana pełna możliwość przesyłania 64 bitowych wiadomości, ale jest to wystarczające aby przetestować czy nasz system poprawnie wykonuje zadanie.

message_f...drone[17]	Input	PIN_Y23	5	B5_N2	PIN_B25	2.5 V (default)	8mA (default)
message_f...drone[16]	Input	PIN_Y24	5	B5_N2	PIN_A19	2.5 V (default)	8mA (default)
message_f...drone[15]	Input	PIN_AA22	5	B5_N3	PIN_G28	2.5 V (default)	8mA (default)
message_f...drone[14]	Input	PIN_AA23	5	B5_N2	PIN_A22	2.5 V (default)	8mA (default)
message_f...drone[13]	Input	PIN_AA24	5	B5_N3	PIN_F22	2.5 V (default)	8mA (default)
message_f...drone[12]	Input	PIN_AB23	5	B5_N3	PIN_F19	2.5 V (default)	8mA (default)
message_f...drone[11]	Input	PIN_AB24	5	B5_N3	PIN_B11	2.5 V (default)	8mA (default)
message_f...drone[10]	Input	PIN_AC24	5	B5_N3	PIN_B8	2.5 V (default)	8mA (default)
message_f..._drone[9]	Input	PIN_AB25	5	B5_N1	PIN_J25	2.5 V (default)	8mA (default)
message_f..._drone[8]	Input	PIN_AC25	5	B5_N3	PIN_G18	2.5 V (default)	8mA (default)
message_f..._drone[7]	Input	PIN_AB26	5	B5_N2	PIN_F21	2.5 V (default)	8mA (default)
message_f..._drone[6]	Input	PIN_AD26	5	B5_N3	PIN_G22	2.5 V (default)	8mA (default)
message_f..._drone[5]	Input	PIN_AC26	5	B5_N2	PIN_B22	2.5 V (default)	8mA (default)
message_f..._drone[4]	Input	PIN_AB27	5	B5_N2	PIN_D20	2.5 V (default)	8mA (default)
message_f..._drone[3]	Input	PIN_AD27	5	B5_N2	PIN_D11	2.5 V (default)	8mA (default)
message_f..._drone[2]	Input	PIN_AC27	5	B5_N2	PIN_J17	2.5 V (default)	8mA (default)
message_f..._drone[1]	Input	PIN_AC28	5	B5_N2	PIN_C23	2.5 V (default)	8mA (default)
message_f..._drone[0]	Input	PIN_AB28	5	B5_N2	PIN_C19	2.5 V (default)	8mA (default)
rst	Input	PIN_M23	6	B6_N2	PIN_AH14	2.5 V (default)	8mA (default)
start	Input	PIN_M21	6	B6_N1	PIN_Y28	2.5 V (default)	8mA (default)

Rys. 22. Pin Planner-rst,start i message_from_drone

Table 2: Pin assignments for slide switches

Signal Name	FPGA Pin No.	Description	I/O Standard
SW[0]	PIN_AB28	Slide Switch[0]	Depending on JP7
SW[1]	PIN_AC28	Slide Switch[1]	Depending on JP7
SW[2]	PIN_AC27	Slide Switch[2]	Depending on JP7
SW[3]	PIN_AD27	Slide Switch[3]	Depending on JP7
SW[4]	PIN_AB27	Slide Switch[4]	Depending on JP7
SW[5]	PIN_AC26	Slide Switch[5]	Depending on JP7
SW[6]	PIN_AD26	Slide Switch[6]	Depending on JP7
SW[7]	PIN_AB26	Slide Switch[7]	Depending on JP7
SW[8]	PIN_AC25	Slide Switch[8]	Depending on JP7
SW[9]	PIN_AB25	Slide Switch[9]	Depending on JP7
SW[10]	PIN_AC24	Slide Switch[10]	Depending on JP7
SW[11]	PIN_AB24	Slide Switch[11]	Depending on JP7
SW[12]	PIN_AB23	Slide Switch[12]	Depending on JP7
SW[13]	PIN_AA24	Slide Switch[13]	Depending on JP7
SW[14]	PIN_AA23	Slide Switch[14]	Depending on JP7
SW[15]	PIN_AA22	Slide Switch[15]	Depending on JP7
SW[16]	PIN_Y24	Slide Switch[16]	Depending on JP7
SW[17]	PIN_Y23	Slide Switch[17]	Depending on JP7

Table 3: Pin assignments for pushbutton (debounced) switches

Signal Name	FPGA Pin No.	Description	I/O Standard
KEY[0]	PIN_M23	Push-button[0]	Depending on JP7
KEY[1]	PIN_M21	Push-button[1]	Depending on JP7
KEY[2]	PIN_N21	Push-button[2]	Depending on JP7
KEY[3]	PIN_R24	Push-button[3]	Depending on JP7

Rys. 23. Przypis pinów do przycisków

7. Rozbudowa systemu

Rozważając rozbudowe systemu wpadliśmy na kilka pomysłów:

— Po pierwsze w przypadku rozbudowy systemu o większą ilość dronów należałoby zmienić input initialize na większą liczbą bitów, tak aby system centrala rozpoznawała z którym dronem się komunikuje, tzn. że zmienna initialize poza sygnałem do początku pracy systemu byłaby jednocześnie identyfikatorem każdego drona.

— Aby dron miał inny generator losowości wystarczy zmienić seed w LFSR na inny wielomian w każdym z dronów. Dzięki temu w przypadku gdy jeden dron zostałby rozpoznany i stał się niebezpieczny to reszta systemu dalej działa nienagannie i jest bezpieczna.

— Jednym ze spososób na zwiększenie bezpieczeństwa systemu jest implementacja bezpiecznego szyfrowania. Funkcja XOR nie jest bezpieczna, jednak jeżeli chodzi o klucz 8 bitowy nie ma to większego znaczenia, gdyż każdy klucz może zostać złamany przy użyciu ataku brute force.

8. Podsumowanie

W tym rozdziale (i podrozdziałach) należy dokonać krytycznej oceny zaproponowanego rozwiązania i możliwości jego rozwoju.

9. Bibliografia

[1]Szyfrowanie symetryczne, a asymetryczne
[2] Algorytm symetryczny
[3]Kryptografia asymetryczna
[4]Protokół Diffiego-Hellmana
[5]Puzzle Merkle'a

[6]RSA
[7]RSA
[8]Curve25519
[9]Algorytm Rabina
[10]Ataki typu "man in the middle" w protkole Diffiego-Hellmana
[11]Trudność obliczenia logarytmu dyskretnego
[12]LFSR
[13]Przypisywanie pinów