Projekt SYCY

-

Zespół nr 8

Michał Kozłowski

Maciej Lipski

Jan Sakowski

Kacper Średnicki

Michał Topolski

1 czerwca 2024

Spis treści

1. Wstęp

Zleconym nam zadaniem jest stworzenie systemu bezpiecznej komunikacji dla floty autonomicznych dronów podwodnych patrolujących przybrzeżne obszary morskie. Komunikacja z centrum dowodzenia (w późniejszej części dokumentu określanym także jako C&C) odbywa się okresowo, co wynurzenie. Wynurzenia odbywają się z częstotliwością co kilka godzin. Gdy dron znajdują się pod wodą prowadzi rekonesans zbierając dane do przesłania i nie komunikuje się z centrum dowodzenia.
W celu poprawnej realizacji zadania należy:
— zapewnić poufność danych za pomocą szyfrowania lekkim algorytmem strumieniowym z kluczem o długości 8 bitów
— dla podniesienia poziomu bezpieczeństwa zmieniać klucz przy każdej transmisji
— zaproponować koncepcję mechanizmu umożliwiającego przekazywania z centrum dowodzenia do drona klucza szyfrującego (lub informacji umożliwiającej wygenerowanie klucza po stronie drona) dla szyfru strumieniowego
— zaprojektować system cyfrowy (po stronie C&C oraz drona) umożliwiający realizację przekazywania klucza (lub informacji do jego generacji) w sposób bezpieczny
— sprawić, iż jedynie konkretny dron może wykorzystać klucz K (klucze powinny być jednorazowe i unikatowe)

Dodatkowo w ramach treści zadania otrzymujemy informację, że do jego realizacji można wykorzystać informacje przekazywane przez drona tekstem jawnym przy nawiązywaniu kontaktu.

2. Organizacja prac

2.1. Informacje wstępne dotyczące realizacji projektu

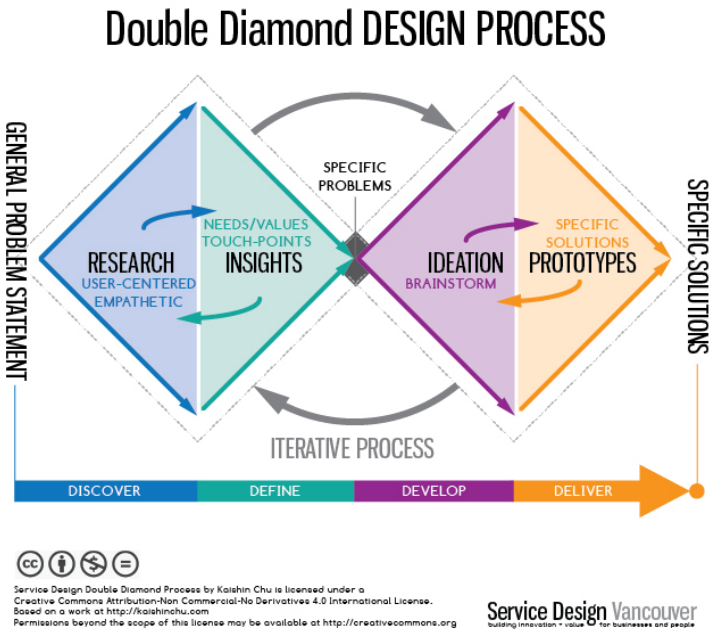
Raport tworzony jest w sposób przyrostowy. Kolejne jego wersje dokumentujące realizację poszczególnych etapów projektu powstają na podstawie wersji poprzednich (na zasadzie nadpisywania). Etapy realizacji projektu są następujące:

1. Etap wstępny, obejmujący: stworzenie zespołu, zapoznanie się z podejściem Design Thinking, wybranie sposobu zarządzania projektem, organizacja warsztatu pracy, dobór narzędzi;
2. Etap odkrywania, obejmujący: analizę literatury i istniejących metod, zebranie wiedzy teoretycznej związanej z tematem projektu;
3. Etap rozwijania, obejmujący: szukanie rozwiązań (burza mózgów, mapa myśli), opracowanie koncepcji rozwiązania na podstawie zdobytej wiedzy, opracowanie prostego modelu referencyjnego oraz danych do testowania;
4. Etap definiowania, obejmujący: rozwinięcie i rozbudowanie koncepcji projektowych docelowego systemu, modelowanie elementów systemu w HDL, weryfikację funkcjonalną, integrację i ocenę prototypów;
5. Etap dostarczania, obejmujący: wdrożenie projektu, uruchomienie go na docelowej platformie (lub przygotowanie rozbudowanego środowiska testowego), przetestowanie według wcześniej opracowanych scenariuszy testowych.

2.2. Zapoznanie się z podejściem Design Thinking

Problem projektowy będzie rozwiązywany z użyciem metody Design Thinking w wersji Double Diamond. Taki model ma zapewnić unikatowe i kreatywne rozwiązanie problemu. Wyszczególnia się dwie zasadnicze przestrzenie realizacji projektu - przestrzeń problemu i przestrzeń rozwiązania. Przestrzeń problemu dzieli się na dwie fazy - faza odkrywania i definiowania [16]. Przestrzeń rozwiązania dzieli się na fazy rozwijania i dostarczania.

W ramach fazy odkrywania zostanie przeprowadzony przegląd literatury i istniejących metod rozwiązania problemu. Zespół projektowy zdobędzie wiedzę teoretyczną potrzebną do realizacji dalszych faz projektu. Następnie przeprowadzone zostanie poszukiwanie rozwiązania i przetestowanie go w modelu referencyjnym. Koncepcje projektowe zostaną dalej rozwinięte i rozbudowane w celu dostarczenia gotowego rozwiązania w ostatniej fazie. Kolejne fazy metody Double Diamond pokrywają się ze wspomnianymi wcześniej etapami realizacji projektu.



Rysunek 1. Schemat podejścia Design Thinking w wersji Double Diamond [17]

2.3. Wybranie modelu realizacji projektu

Jako sposób zarządzania projektem został wybrany model kaskadowy (Waterfall). Jest to jedna ze starszych metod zarządzania, charakteryzująca się klasycznym podejściem do rozwoju projektu. Głównym założeniem tej metody jest sporządzenie planu działania obejmującego dokładne terminy ukończenia etapów oraz opis zasobów potrzebnych do "zamknięcia"projektu. Projekt jest realizowany "krok po kroku", przy czym niemożliwe jest "cofanie się"do poprzedniego etapu, zaś wszelkie odstępstwa od planu nie powinny być tolerowane.

Podczas podejmowania decyzji pod uwagę były brane również tzw. metody zwinne (np. Agile), jednakże metoda Waterfall zdaje się być lepiej dopasowana do potrzeb projektu.

2.4. Organizacja i zarządzanie projektem

Członkowie zespołu znają się nawzajem i pracowali wspólnie nad projektami w pierwszym semestrze studiów. Wspólnie podjęto decyzję o niewyłanianiu lidera zespołu. Każdy z jego członków posiada inny zestaw umiejętności i mocnych stron. Dzięki dobrej komunikacji i wykorzystaniu dostępnych narzędzi jesteśmy przekonani, że uda się skutecznie, grupowo zarządzać projektem.

2.5. Wybór narzędzi

Do komunikacji pomiędzy członkami zespołu, poza klasycznym kontaktem "twarzą w twarz"w wydziałowej bibliotece, wykorzystywane będzie kilka narzędzi. Dyskusja oraz szybkie ustalenia dotyczące poszczególnych elementów realizacji projektu będą odbywać się z wykorzystaniem komunikatora Messenger, ze względu na łatwy i codzienny dostęp członków zespołu do tej platformy. W przypadku konieczności połączenia się członków w czasie rzeczywistym w formie zdalnej, zostanie wykorzystany grupowy kanał na platformie Discord. Kluczowe dla realizacji projektu informacje, postępy w pracy oraz ewentualne wątpliwości będą umieszczane na utworzonym przez koordynatora projektu kanale zespołu na platformie MS Teams. Takie podejście pozwoli zachować porządek informacji i wpisów na tym kanale oraz umożliwi koordynatorowi weryfikację postępów. Do wymiany kodów (podczas tworzenia modelu referencyjnego i modelowania w języku HDL) zostanie wykorzystany system kontroli wersji GIT. Kody będą umieszczane w repozytorium na platformie GitLab.

Sprawozdanie z poszczególnych etapów realizacji projektu będzie tworzone z wykorzystaniem platformy Overleaf umożliwiającej współtworzenie raportu przez wszystkich członków zespołu.

Podsumowanie narzędzi, które zostaną wykorzystane podczas realizacji projektu:
1. MS Teams
2. Messenger
3. Discord
4. Overleaf
5. Gitlab

Podczas rozwijania projektu i realizowania kolejnych jego etapów lista ta zostanie rozszerzona o narzędzia, które pozwolą na realizację projektu w sferze typowo technicznej. W miarę potrzeby mogą zostać wykorzystane także inne narzędzia komunikacyjne czy pomocnicze, w takim przypadku w raporcie pojawi się taka informacja.

Na etapie rozwijania projektu zostały wykorzystane dodatkowe narzędzia:
1. IntelliJ IDEA - narzędzie użyto do przygotowania modeli referencyjnych w języku Java.
2. UMLStar - narzędzie użyto do przygotowania schematu uzgadniania klucza.

Na etapie definiowania zostały wykorzystane następujące dodatkowe narzędzia:
1. Quartus Prime Lite Edition - narzędzie do pisania i kompilowania kodu w języku opisu sprzętu Verilog.
2. ModelSim - narzędzie do symulowania kodu Verilog.
3. Creately - narzędzie wykorzystano do przygotowania diagramu algorytmu szybkiego potęgowania modularnego.

3. Etap odkrywania

Aby zapewnić bezpieczeństwo szyfrowania transmisji, obie jej strony - tj. centrum C&C oraz dron - muszą uzgodnić między sobą w bezpieczny sposób klucz dla szyfru strumieniowego. Wybrany do tego sposób musi gwarantować brak możliwości rozkodowania sygnału innych dronów w razie złamania zabezpieczeń jednego oraz należy zmieniać go po każdej transmisji. Na tym etapie zapoznano się zarówno z szyframi symetrycznymi jak i asymetrycznymi, oraz z protokołami wymiany klucza. Poniżej znajdują się opisy różnych technologii i rozwiązań, które uznaliśmy za ciekawe w kontekście projektu.

3.1. Algorytmy asymetryczne - ogólne informacje

Szyfry asymetryczne opierają się parze klucz prywatny - klucz publiczny. Każda strona ustanawia swoją własną, unikatową parę, której używa do komunikacji z drugą - w tym przypadku dwie strony komunikacji to dron i centrala.

W wymianie informacji na tej zasadzie publicznie transmitowany jest tylko klucz publiczny, podczas gdy prywatny, używany do zakodowania wiadomości pozostaje sekretem każdej ze stron. Algorytmy szyfrów asymetrycznych operują bazując na potęgowaniu modulo dużych liczb pierwszych. Swoje bezpieczeństwo zawdzięczają potężnym kosztom faktoryzacji dużych liczb pierwszych oraz *problemowi logarytmu dyskretnego*. Poziom skomplikowania tych operacji czyni *brute force* jedyną skuteczną metodą łamania ich. Jednak, jak wiadomo, jej koszt obliczeniowy czyni ją niepraktyczną.

3.2. Protokół Diffiego-Hellmana

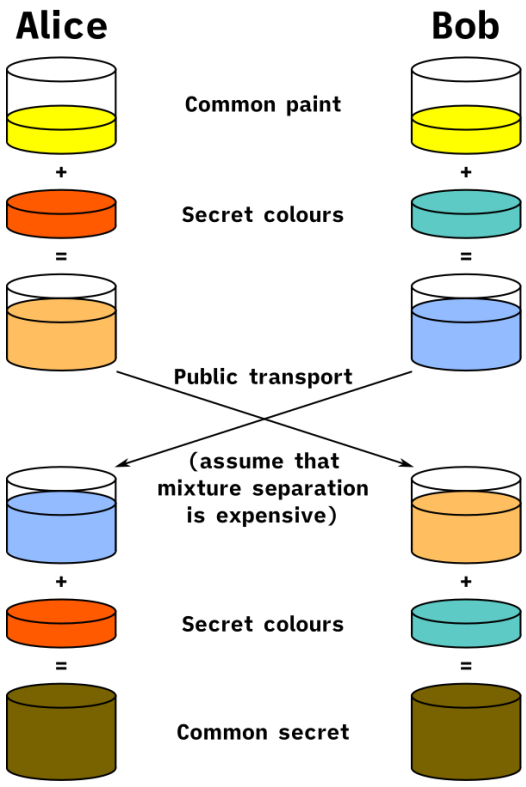
Projektowany przez zespół system opierać się będzie na kryptografii symetrycznej, potrzebuje zatem sposobu na bezpieczne ustalanie wspólnego klucza. Optymalnym rozwiązaniem tej kwestii jest zatem protokół Diffiego-Hellmana (skrótowo DH) . Protokół ten jest powszechnie używany, a jego skuteczność została potwierdzona w praktyce w wielu zastosowaniach. Jest on również mało kosztowny obliczeniowo, szczególnie gdy generowana przez niego wiadomość jest relatywnie niewielka. Dzięki temu idealnie pasuje do potrzeb projektowanego systemu.

Działanie protokołu DH polega na uzgodnieniu liczb pierwszych p oraz n, takich, że n jest pierwiastkiem pierwotnym grupy $\mathbf{D_H} = (\mathbb{Z}_p^*, \cdot_p)$. [13] Obie te liczby są upubliczniane - są kluczem publicznym. Każda ze stron generuje swój klucz prywatny (niech będzie to kolejno a oraz b), którym może być dowolna liczba całkowita. Następnie obie strony podnoszą n do potęgi modulo p swojego klucza i udostępniają publicznie.

Podsłuchujący zna w ten sposób wynik $n^a \bmod p$ i $n^b \bmod p$, oraz parę p, n, jednak znalezienie pary a, b jest bardzo trudne, gdyż moduł wymaga zbadania bardzo wielu możliwych par, co oznacza wielokrotne, kosztowne rozwiązywanie problemu *dyskretnego logarytmu*.

Obie strony biorą otrzymaną w ten sposób wartość i znów wykonują na niej operację potęgowania modulo z użyciem swojego klucza. W rezultacie obie strony kończą z tym samym wynikiem! Rezultatem jest wytworzenie klucza po obu stronach, bez ujawniania go, czyli dokładnie to, co jest od systemu wymagane.

$$(n^b \mod p)^a \mod p = (n^a \mod p)^b \mod p$$



Rysunek 2. Wizualna reprezentacja działania protokołu DH[18]

3.3. Rozwinięcie protokołu Diffiego-Hellmana z wykorzystaniem RSA

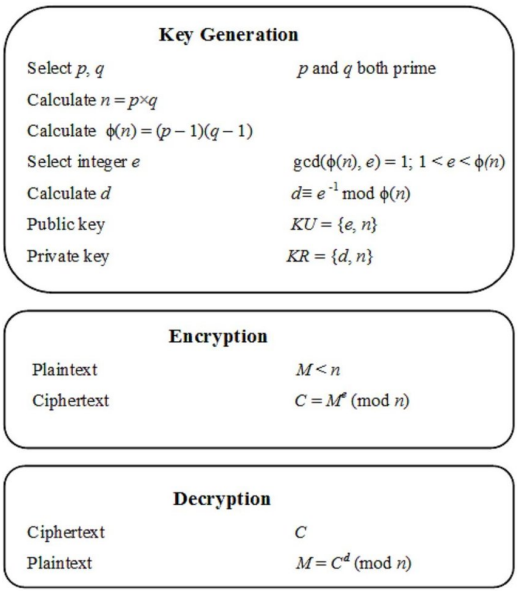
Problemem, z którym wiąże się wykorzystanie protokołu Diffiego-Hellmana (skrótowo DH) jest podatność na ataki typu Man-in-the-Middle[7]. Po przechwyceniu kluczy publicznych i przy równoczesnym posiadaniu znajomości liczb uzgodnionych przez strony, możliwe jest wygenerowanie własnego klucza do komunikacji ze stronami, bez możliwości ich zorientowania się o komunikacji z niewłaściwym podmiotem.

Rozwiązaniem tego problemu może być zastosowanie algorytmu RSA [7]. RSA to algorytm szyfrujący oparty na kryptografii asymetrycznej. Powstał w 1978 roku. Możliwe jest jego wykorzystywanie zarówno do kryptografii klucza publicznego, jak i do podpisów cyfrowych. Algorytm RSA jest używany m.in. przy zabezpieczeniu komunikacji pocztą email oraz podczas transakcji bankowych. Siła szyfrowania RSA opiera się na trudności potęgowania modularnego i braku wydajnych algorytmów przeprowadzania operacji odwrotnej[3].

3.3.1. Działanie algorytmu RSA

Zarówno klucz publiczny, jak i klucz prywatny składają się z dwóch zmiennych - klucz publiczny z pary (n,e), a prywatny z (n,d). Konieczne jest wybranie dwóch dużych liczb pierwszych. Następnie należy pomnożyć je przez siebie w celu uzyskania liczby n, nazywanej modułem klucza. Dalszym krokiem generowania kluczy RSA jest policzenie wartości funkcji Eulera dla n i wybranie liczbę e, która jest całkowita, mniejsza niż wyliczona wartość funkcji Eulera, oraz jest z nią względnie pierwsza. Liczba d uzyskiwana jest natomiast poprzez rozwiązanie równania $d * e \equiv 1 \mod \phi(n)$. Szyfrowanie odbywa się za pomocą klucza publicznego (n,e), a deszyfrowanie za pomocą klucza prywatnego (n,d). W razie potrzeby (przykładowo przy podpisie cyfrowym), można również

zaszyfrować wiadomość kluczem prywatnym. Wtedy deszyfrowanie odbywa się z wykorzystaniem klucza publicznego. Wiadomość należy podzielić na fragmenty, a następnie każdy fragment należy przełożyć na liczbę. W tym celu można wykorzystać przykładowo system kodowania znaków ASCII. W celu uzyskania szyfrogramu należy obliczyć dla każdego następującą wartość[3] $c \equiv m^e \mod n$, gdzie c to fragment szyfrogramu, m to fragment wiadomości do zaszyfrowania. Deszyfrowanie odbywa się według następującego wzoru $m \equiv c^d \mod n$.



Rysunek 3. Schemat szyfrowania z wykorzystaniem algorytmu RSA[15]

3.3.2. Propozycja zastosowanie RSA w protokole Diffiego-Hellmana

Schemat zastosowania RSA w protokole Diffiego-Hellmana został zaproponowany w pracy Chiradeep Gupta'y i N V Subba Reddy'ego[7]. W celu zastosowania algorytmu łączącego RSA i DH konieczne jest wybranie czterech początkowych liczb p, q, X, g (generator) spełniających następujące warunki:
— $p \neq q$
— $p > g \wedge q > g$
— $X > g$
Liczby p i q są prywatne, natomiast X, g są jawne. Następnie używa się schematu znanego z klasycznego RSA, w celu obliczenia kluczy prywatnych i publicznych dla obu stron. Liczbę, oznaczaną w zawartym w raporcie opisie RSA jako n, oblicza się według następującego wzoru: $n = p * q$. Następnie konieczne jest uzgodnienie przez obie strony wartości, nazwanej w opisie RSA e. Tak samo jak w klasycznym RSA, $e > \Phi(n) \wedge e, \Phi(n)$ są względnie pierwsze. Obie strony muszą wybrać dowolną liczbę mniejszą od X (ozn. a - klucz prywatny, druga strona oblicza według tego samego schematu $a \rightarrow b,\ \alpha \rightarrow \beta$) i obliczyć klucz publiczny zgodnie ze wzorami:

$$de \equiv 1 \mod \Phi(n)$$
$$klucz\ publiczny(\alpha) \equiv g^a \mod (X)$$

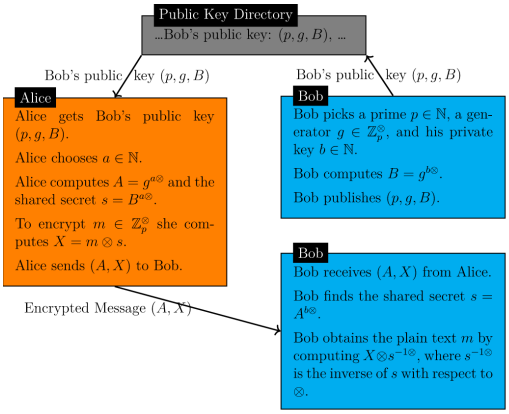
Następnie następuje wysłanie wiadomości do drugiej strony z zaszyfrowanym kluczem publicznym $C \equiv \alpha^e \mod (n)$. Każda ze stron oblicza klucz publiczny drugiej według wzoru $\beta \equiv C^d \mod (n)$. Na podstawie wzoru $k \equiv \beta^a \mod (X) = \alpha^b \mod (X) = g^a b \mod (x)$ oblicza się wspólny klucz dla szyfrowania symetrycznego. Ze względu na to, że klucze publiczne obu stron są współdzielone, możliwe jest zweryfikowanie nadawcy wiadomości, poprzez deszyfrowanie wysłanego klucza publicznego.

3.3.3. Wnioski dotyczące możliwości zastosowania RSA w projekcie

Opisany algorytm zapewnia bezpieczeństwo wobec ataków typu Man-in-the-Middle. Jego wadą jest duża złożoność obliczeniowa, większa niż RSA i DH osobno [7]. RSA jest także wolniejsze niż szyfrowanie symetryczne i wymaga dużych zasobów obliczeniowych [9], co może być przeszkodą do zastosowania tego algorytmu w projekcie. Mimo to warto uwzględnić fakt, że stosowanie RSA do transmisji kluczy dla kryptografii symetrycznej jest popularne[9].

3.4. Algorytm ElGamal

Algorytm ElGamal jest oparty na teorii logarytmu dyskretnego w ciele liczb całkowitych modulo duża liczba pierwsza. W celu rozpoczęcia komunikacji, jedna ze stron musi wygenerować klucz publiczny. W tym celu wybiera dużą liczbę pierwszą p oraz generator w grupie multiplikatywnej modulo p w zbiorze liczb całkowitych. Następnie wybiera jakąś liczbę b i oblicza $B = g^b \ mod \ p$ i udostępnia drugiej stronie $p, \ g \ i \ B$. Druga strona wybiera sobie a i oblicza wspólny klucz tajny $s = B^a \ mod \ p$ oraz $A = g^a \ mod \ p$. Następnie szyfruje wiadomość X za pomocą wspólnego klucza i przesyła $A \ oraz \ X$ do inicjatora komunikacji. Inicjator oblicza wspólny klucz tajny i z jego pomocą rozszyfrowuje wiadomość (mnożenie przez element odwrotny do $s \ mod \ p$)[11].



Rysunek 4. Schemat przesyłania wiadomości między stronami z użyciem kryptosystemu ElGamal[12]

Algorytm ElGamal jest uznawany za bezpieczny, gdyż problem dyskretnego logarytmu grupy modulo jest trudny do rozwiązywania. Dodatkowo korzystanie z tego kryptosystemu umożliwia zostawienie cyfrowego podpisu. Jednakże, w ramach bezpieczeństwa komunikacji, należy użyć bardzo dużej długości klucza, a sam algorytm jest wolny w porównaniu do innych używanych w tym samym celu[1]. Wydaje się to być problemem z racji ograniczeń systemowych dronów.

3.5. Algorytmy strumieniowe

3.5.1. Opis działania, bezpieczeństwo, zastosowanie, wady i zalety

Algorytmy strumieniowe należą do rodziny algorytmów symetrycznych. Podstawą ich działania jest klucz będący parametrem generatora strumienia bitów. Generowany klucz sesyjny ma długość równą długości tekstu jawnego, aby z wykorzystaniem jego a także elementu dodającego (np. XOR), możliwe było szyfrowanie danych wejściowych bit po bicie (każdy n-ty element jest szyfrowany n-tym elementem strumienia).[6]

Kluczową kwestią w kontekście każdego rozwiązania wykorzystującego takie algorytmy jest szeroko pojęte bezpieczeństwo. Aby takowe zapewnić, strumień klucza powinien być możliwie bliski losowemu. Wystąpienie 0 i 1 powinno być równie prawdopodobne, a także bity w strumieniu powinny być od siebie statystycznie niezależne.

Szyfry strumieniowe są powszechnie stosowane w dzisiejszych rozwiązaniach, także w systemach cyfrowych. Wyróżniają się takimi cechami jak szybkość czy efektywność, ponieważ generowanie klucza nie jest skomplikowane ani nie wymaga pokaźnych zasobów sprzętowych czy obliczeniowych. Z tych względów algorytmy strumieniowe znalazły zastosowanie w połączeniach bezprzewodowych czy rozwiązaniach o znacznym prawdopodobieństwie błędów transmisji (dzięki niewielkiej propagacji błędów).

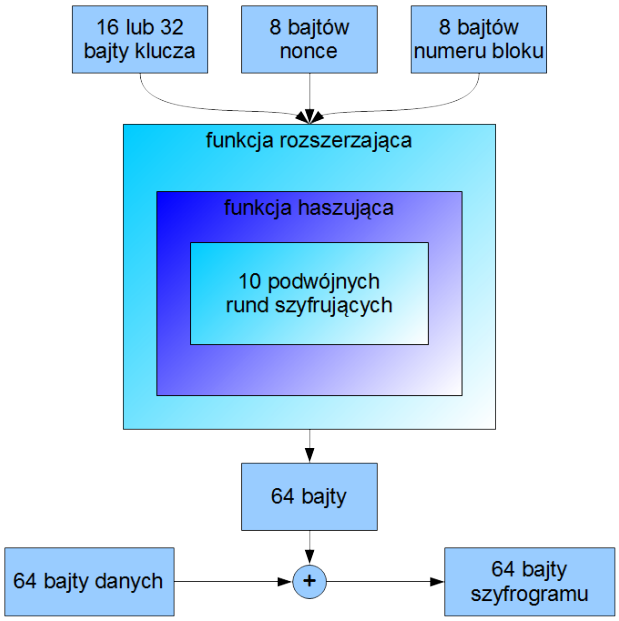
Istotną kwestią w kontekście algorytmów strumieniowych jest konieczność synchronizacji pomiędzy nadawcą i odbiorcą, aby możliwe było synchroniczne generowanie klucza sesyjnego u obu stron.

Szyfry strumieniowe mają też swoje wady. Są podatne na ataki zmiany bitów, powodujące zniekształcenie wiadomości. Innym ataku, na który podatność wykazują te algorytmy są ataki wstawiania/usuwania bitów, prowadzące do utraty synchronizacji.[14]

Ogólne informacje o rodzinie algorytmów strumieniowych przedstawione powyżej na pierwszy rzut oka zdają się pasować do naszych potrzeb projektowych (drony wynurzają się co kilka godzin umożliwiając synchronizację, klucz może być zmieniany przy każdej transmisji, itd.). Poniżej zostaną wyszczególnione konkretne algorytmy strumieniowe, które w kolejnym etapie będą rozważone w kontekście zastosowania w projekcie.

3.5.2. Salsa20

Jest to nowoczesny oraz wydajny szyfr strumieniowy. Powstał w 2005 roku w Chicago za sprawą prof. D. Bernstein'a. Jego mocną zaletą jest brak, jak do tej pory, znanych skutecznych sposobów ataku na ten algorytm. Dla 64-bajtowego bloku danych Salsa20 tworzy funkcję rozszerzającą (z klucza, nonce oraz numeru bloku), która jest przekazywana do funkcji haszującej. Ta funkcja haszująca wykorzystując klucz, nonce oraz stałe wektory miesza dane i zwraca inne 64 bajty danych do funkcji rozszerzającej. Te 64 bajty są XOR-owane z 64 bajtami tekstu jawnego, a wynikiem tego działania jest szyfrogram.[4]



Rysunek 5. Schemat szyfrowania Salsa20 [5]

Poza szybkością Salsa20 oferuje wysoki poziom bezpieczeństwa. Wbrew pozorom implementacja tego algorytmu nie jest niesamowicie skomplikowana na tle innych algorytmów. Oczywiście jak każdy algorytm posiada pewne wady takie jak konieczność doboru odpowiedniej długości klucza. Mimo to jest bardzo dobrym rozwiązaniem, wartym rozważenia w przypadku naszego projektu.

3.5.3. RC4

RC4 (znany również jako Rivest Cipher 4 lub ARC4) to symetryczny szyfr strumieniowy, który został opracowany przez Rona Rivesta w 1987 roku. Był on szeroko stosowany w różnych aplikacjach, ale obecnie ze względu na wiele odnalezionych podatności nie jest zalecany (wręcz przeciwnie - zaleca się zaprestanie jego wykorzystania).

RC4 jest szyfrem strumieniowym o zmiennej wielkości klucza, co oznacza, że szyfruje tekst jawny poprzez wygenerowanie strumienia klucza złożonego z pseudolosowych bitów, który następnie jest łączony z tekstem jawnym za pomocą bitowego XOR. Strumień klucza jest generowany na podstawie tajnego klucza i wektora inicjalizacji (IV) przy użyciu algorytmu szeregowania kluczy (KSA) i algorytmu generowania pseudolosowego (PRGA).

Jest rozwiązanie ciekawe pod względem historycznym, które można by było rozważyć gdyby pomimo podatności przodowało w prostocie i lekkości implementacji, niestety choć jest to szfr wydajny to istniej parę lepszych rozwiązań po tym względem.

3.5.4. OTP

Szyfr z kluczem jednorazowym, znany również jako szyfr Vernama, to technika szyfrowania, która wykorzystuje tajny klucz złożony z losowych znaków do zaszyfrowania wiadomości. Klucz i wiadomość muszą być tej samej długości, a każdy znak w kluczu jest dodawany (w oryginalnej historycznej implementacji mod 26) do odpowiedniego znaku w wiadomości, aby utworzyć zaszyfrowany tekst.

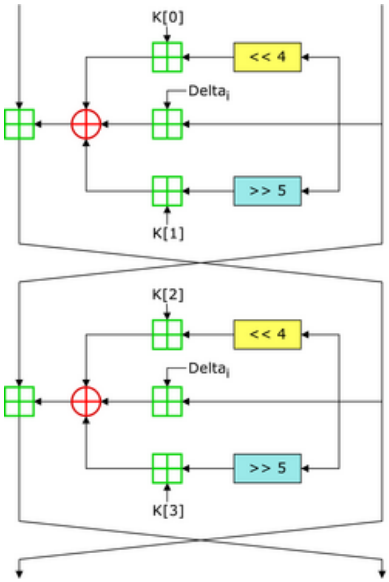
Klucz jest generowany losowo i nigdy nie jest ponownie używany. Bezpieczeństwo szyfru zależy od tego, czy klucz jest naprawdę losowy i utrzymywany w tajemnicy. Jeśli klucz jest używany więcej niż raz lub nie jest naprawdę losowy, szyfr może być łatwo złamany.

Jedną z zalet szyfrowania OTP jest to, że jest on matematycznie nie do złamania, jeśli jest używany prawidłowo. Nawet przy nieograniczonej mocy obliczeniowej, atakujący nie może ustalić klucza lub oryginalnej wiadomości. Jednak klucz musi być bezpiecznie dzielony między nadawcę i odbiorcę, co w praktyce może być trudne.

3.5.5. TEA

Szyfr TEA (Tiny Encryption Algorithm) jest symetrycznym algorytmem szyfrującym, który operuje na 64-bitowych blokach danych. Został zaprojektowany jako lekki, wydajny algorytm do stosowania w aplikacjach sprzętowych i programowych, gdzie zasoby są ograniczone.

TEA wykorzystuje 128-bitowy klucz do szyfrowania i deszyfrowania danych. Proces szyfrowania obejmuje kilka rund operacji, z których każda składa się z połączenia operacji przesunięcia bitów i operacji XOR na bloku danych i kluczu. Liczba rund zależy od wielkości klucza, przy czym dla klucza 128-bitowego stosuje się 32 rundy.



Rysunek 6. Schemat dwóch przebiegów TEA[8]

Szyfr TEA jest stosunkowo prosty, ale jednocześnie uważany jest za bardzo bezpieczny. Jego prostota sprawia, że można go łatwo zaimplementować, a jego bezpieczeństwo zostało dokładnie przetestowane i zweryfikowane na przestrzeni lat. Stwierdzono jednak, że ma on słabe punkty w stosunku do niektórych ataków, takich jak ataki z użyciem powiązanych kluczy, i nie powinien być uznawany za bezpieczny dla zastosowań wysokiego ryzyka.

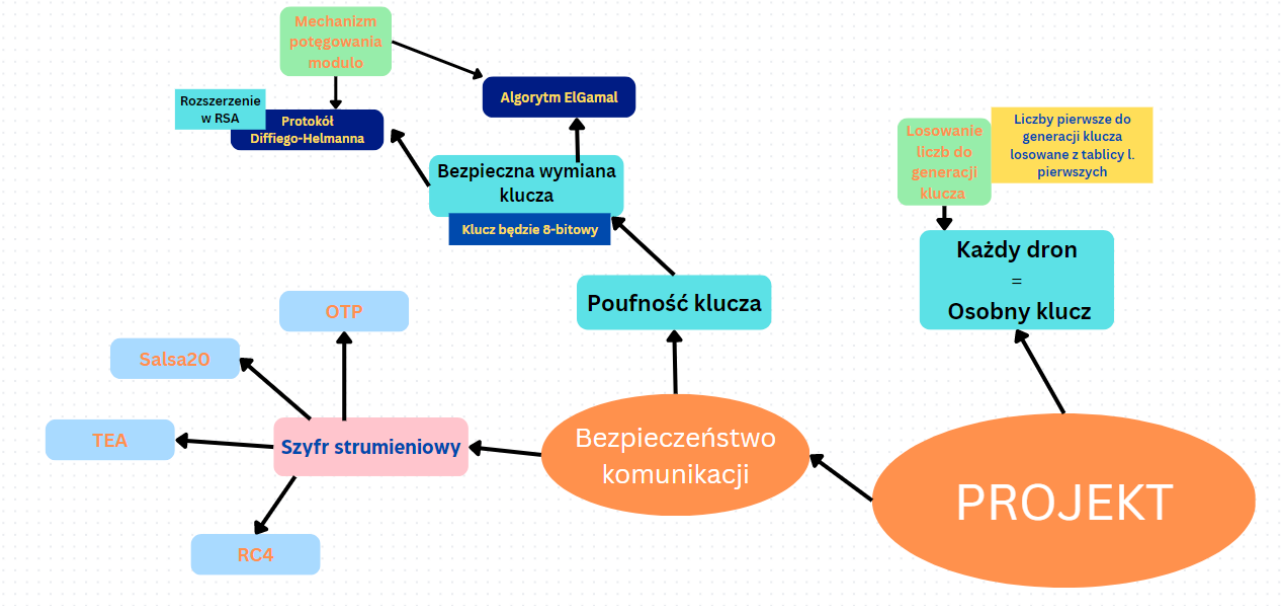
Jest on atrakcyjny do implementacji w układach takich jakie potencjalnie mogą się przydać w projekcie, łatwo go bowiem zaimplementować sprzętowo.

3.6. Podsumowanie etapu odkrywania

Na tym etapie dokonano przeglądu literatury i dostępnych rozwiązań. Zapoznano się z protokołem Diffiego-Hellmana i jego rozwinięciem z wykorzystaniem RSA, oraz algorytmem ElGamal. Na podstawie zdobytej wiedzy, na dalszym etapie rozwijania projektu zostanie podjęta decyzja odnośnie wyboru jednego z tych algorytmów uzgadniania klucza. Dokonano także przeglądu szyfrów strumieniowych, które mogą zostać wykorzystane do szyfrowania wiadomości, oraz wybranych lekkich szyfrów symetrycznych. W kolejnym etapie jeden z nich zostanie wytypowany jako docelowy.

4. Etap rozwijania

W etapie rozwijania pomysłu zespół skupił się na obraniu optymalnego rozwiązania, które zapewniałoby balans pomiędzy wydajnością systemu, jego skomplikowaniem i bezpieczeństwem. Główne problemy zadania, a zarazem możliwe elementy ostatecznego systemu zostały zidentyfikowane w procesie *burzy mózgów* i opisane poniższą mapą myśli.



Rysunek 7. Mapa myśli obrazująca moduły realizujące założenia projektowe

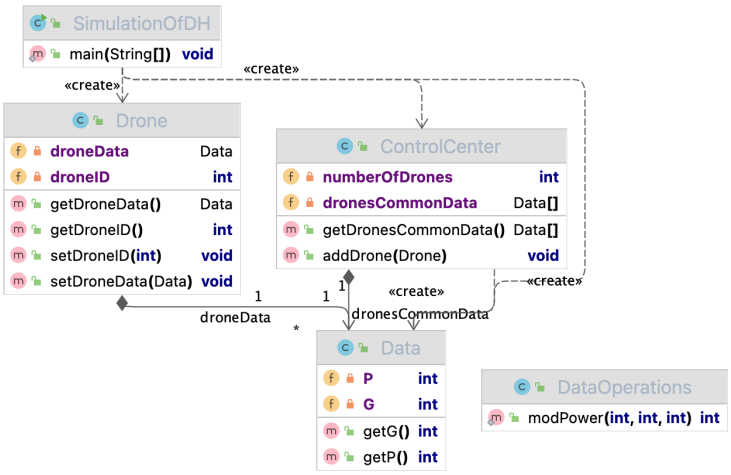
Część dotycząca szyfrów strumieniowych nie jest, naturalnie, obowiązkową częścią projektu, jednak bezpieczny szyfr strumieniowy byłby w realnym użyciu urządzenia konieczny. Z tego to powodu sekcja tych szyfrów dotycząca została umieszczona na mapie, pomimo braku ich implementacji w ostatecznej wersji projekty (zapewne).

Podane metody uzgadniania klucza zostały w tym etapie zrealizowane w języku Java. Wykonano kod symulujący uzgadnianie klucza zarówno protokołem Diffiego-Hellmana, jak i algorytmem ElGamal. Dołączone do opisów kodu diagramy UML mają na celu ułatwienie zrozumienia sposobu konstrukcji kodu.

4.1. Model protokołu Diffiego-Hellmana

4.1.1. Ogólny opis rozwiązania

Przygotowano, w języku Java, model uzgadniania klucza według protokołu Diffiego-Hellmana (dalej w skrócie DH) między control-center a dronem. Dron i control-center (dalej skrótowo CC) zostały przedstawione jako klasy. Klasa Drone posiada jako atrybuty obiekt klasy Data i swoje ID. Control-center przechowuje liczbę dronów, które kontroluje, oraz tablicę z ich danymi. ID drona to jego indeks w Data[] dronesCommonData. Klasa Data zawiera informacje o uzgodnionymi między danym dronem, a CC liczbie pierwszej (P w modelu protokołu DH) oraz generatorze grupy $G = \mathbb{Z}_P$. Liczby te są na stałe uzgodnione między każdym dronem a CC i są unikatowe dla danego drona.



Rysunek 8. Diagram UML przygotowanego modelu protokołu Diffiego-Hellmana

4.1.2. Potęgowanie modularne

Przygotowano algorytm pozwalający na szybkie potęgowanie modularne. Zaimplementowana wersja tej operacji ma złożoność obliczeniową O($log\ y$) gdzie y to wykładnik potęgi.[2] Nie wymaga też dodatkowej pamięci. Kod wydaje się także być łatwy do implementacji na dalszym etapie w języku opisu sprzętu Verilog, ponieważ opiera się na operacjach bitowych. Działanie tego algorytmu jest następujące:

1. Dbamy o to, aby podstawa potęgi (base) była mniejsza niż liczba modulo, do której ma być obliczona potęga (modulus), obliczając base *mod* modulus i przypisując wartość do base.
2. Sprawdzamy czy base jest równe 0,
   — jeśli tak - wynikiem jest 0,
   — jeśli nie - wykorzystujemy pętle do wykonania kolejnych operacji.
3. W pętli sprawdzamy czy ostatni bit wykładnika (exponent) jest równy 1, wykorzystując operator bitowy AND z argumentem 1. Jeżeli tak - mnożymy zmienną result przez podstawę potęgi, a następnie obliczamy resztę z dzielenia wyniku tego mnożenia przez modulus i przypisujemy ową wartość do result.
4. W tej samej iteracji pętli przesuwamy ostatni bit wykładnika o 1 bit w prawo, a także obliczamy resztę z dzielenia kwadratu podstawy potęgi przez modulus i przypisujemy nową wartość do zmiennej podstawy - base.
5. Pętla wykonuje się dopóki exponent jest większy od 0.
6. Metoda zwraca wynik result.

```
public static int modPower(int base, int exponent, int modulus) {
    int result = 1;
    base = base % modulus;
    if (base == 0)
        return 0;
    while (exponent > 0) {
        if ((exponent & 1) != 0)
            result = (result * base) % modulus;
        exponent = exponent >> 1;
        base = (base * base) % modulus;
    }
    return result;
```

Rysunek 9. Algorytm potęgowania modularnego zaimplementowany w języku Java

4.1.3. Symulacja protokołu wymiany klucza

Przygotowano symulację ustalania wspólnego klucza między dronem,a CC. Na początku (przed wypuszczeniem drona do patrolowania) wgrywane są mu liczby opisane w protokole DH jako P (liczba pierwsza) i G (generator grupy $\mathbb{Z}_P$), oraz unikalny identyfikator drona. W wypadku tego drona liczba P została ustawiona jako 8-bitowy wektor binarny 11011111 (dziesiętnie 223) a G jako 00000101 (dziesiętnie 5).

```
Drone drone1 = new Drone(new Data(0b11011111, 0b00000101));
controlCenter.addDrone(drone1);
```

Rysunek 10. Symulacja działań przed wypuszczeniem drona do patrolowania

Przed wynurzeniem dron generuje losowo swój klucz prywatny - b. Klucz prywatny jest jednorazowy. Następnie z jego użyciem generuje klucz, który wyśle do CC według wzoru $y = G^b \mod P$ [13]. Tworzy wiadomość o długości 16-bitów. Pierwsze osiem bitów zajmują dane identyfikacyjne drona, a pozostałe 8 obliczony klucz (oznaczony wcześniej jako y). Dron wysyła tą wiadomość tekstem jawnym - jest to wiadomość inicjalizująca ustalanie klucza.

```
//Setting drone and control center private keys.
int privateKeyDrone = (int) (Math.random() * 255);
Data drone1Data = drone1.getDroneData();
int keyToSendByDrone = DataOperations.modPower(drone1Data.getG(),
↪  privateKeyDrone, drone1Data.getP());
//Message 1 - drone sends its ID and generated key.
String keyInMes1 = (String.format("%8s",
↪  Integer.toBinaryString(keyToSendByDrone))).replaceAll(" ", "0");
String drone1ID = (String.format("%8s",
↪  Integer.toBinaryString(drone1.getDroneID()))).replaceAll(" ", "0");
String message1 = drone1ID + keyInMes1;
```

Rysunek 11. Symulacja wysłania przez drona wiadomości inicjalizującej

Po otrzymaniu wiadomości inicjalizującej, CC na jej podstawie identyfikuje drona. CC generuje losowo swój klucz prywatny - a i podobnie jak w przypadku drona, oblicza klucz który do niego wyśle - x. W tym samym kroku, CC oblicz klucz do deszyfrowania wiadomości od drona zgodnie ze wzorem $k = y^b \mod P$ [13]. CC wysyła dronowi 8-bitową wiadomość z wygenerowanym kluczem x.

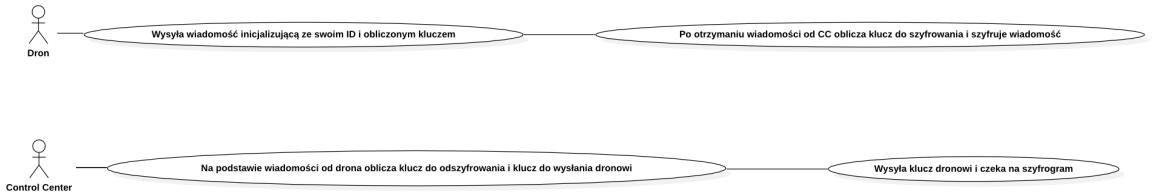
```
//Control center generates its key to send it to the drone and computes
↪  the final key.
int receivedDroneID = Integer.parseInt(message1.substring(0, 8), 2);
int receivedKeyByCC = Integer.parseInt(message1.substring(8, 16), 2);
Data currentDrone =
↪  controlCenter.getDronesCommonData()[receivedDroneID];
int privateKeyCC = (int) (Math.random() * 255);
int keyToBeSendByCC = DataOperations.modPower(drone1Data.getG(),
↪  privateKeyCC, currentDrone.getP());
int finalKeyCC = DataOperations.modPower(receivedKeyByCC, privateKeyCC,
↪  currentDrone.getP());
//Message 2 - control center sends to drone its key;
String message2 = (String.format("%8s",
↪  Integer.toBinaryString(keyToBeSendByCC))).replaceAll(" ", "0");
```

Rysunek 12. Działania podejmowane przez CC po otrzymaniu wiadomości od drona

Po otrzymaniu wiadomości od CC dron musi obliczyć klucz do szyfrowania wiadomości. Robi to ze wzoru $k = x^b \mod P$. Następnie dron może wykorzystać ten 8-bitowy klucz do szyfrowania symetrycznego wiadomości do CC.

```
//Drone calculates its finalKey
int receivedKeyByDrone = Integer.parseInt(message2, 2);
int finalKeyDrone = DataOperations.modPower(receivedKeyByDrone,
↪  privateKeyDrone, drone1Data.getP());
```

Rysunek 13. Obliczanie klucza do szyfrowania przez drona



Rysunek 14. Schemat ustalania klucza między dronem a CC według protokołu DH

```
Message 1 from drone to CC: 0000000001000110
Final key computed by CC: 10111100
Message 2 - from CC to the Drone: 11010100
Final key computed by the Drone: 10111100
Final keys of both sides are the same: true
```

Rysunek 15. Wyniki symulacji ustalania klucza między dronem a CC według protokołu DH

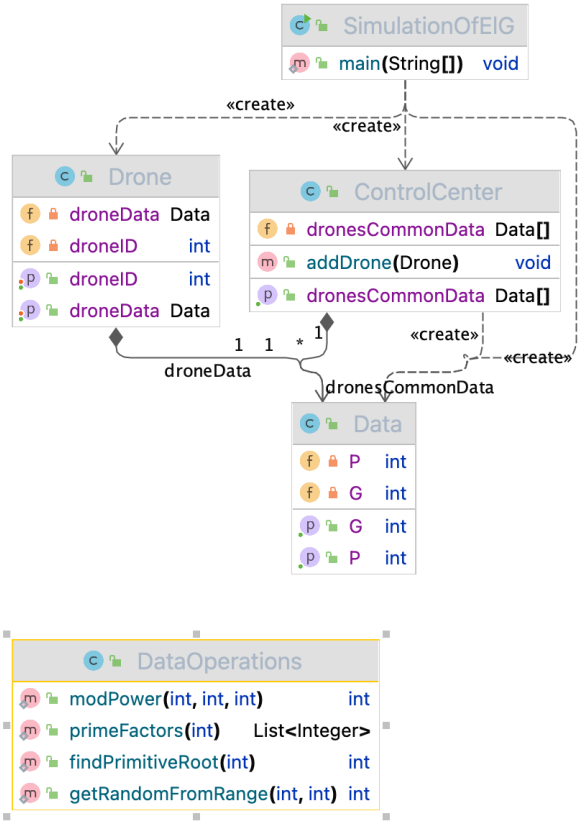
4.1.4. Wnioski dotyczące modelu i jego dalszego wykorzystania

Model wydaje się być obiecujący do implementacji na układzie FPGA. Zawarto w nim wydajną wersję potęgowania modularnego, która może zostać łatwo zaimplementowana na FPGA. Konieczne będzie także zaimplementowanie generatora liczb pseudolosowych, w celu wyboru jednorazowego klucza prywatnego. W wypadku przechwycenia drona, CC skreślałoby go z listy kontrolowanych dronów, stąd atak typu Man-in-the-Middle nie stanowi zagrożenia. W związku z tym, model z wykorzystaniem protokołu Diffiego-Hellmana spełnia założenia projektowe i wydaje się być dobrym wyborem do dalszej implementacji.

4.2. Model protokołu ElGamala

4.2.1. Ogólny opis rozwiązania

Przygotowano również model uzgadniania klucza według protokołu ElGamal między control-center a dronem w jezyku Java. Dron i control-center ponownie zostały przedstawione jako klasy. Atrybuty i metody klas są niemal identyczne, co w przypadku implementacji protokołu Diffiego-Hellmana. Zasadniczą różnicą między dwiema implementacjami jest ilość metod w klasie DataOperations; w przypadku implementacji ElGamal potrzebne są metody do znajdowania pierwiastka pierwotnego, losowej liczby w danym zakresie oraz czynników pierwszych.



Rysunek 16. Diagram UML przygotowanego modelu protokołu ElGamal

4.2.2. Wyznaczanie pierwiastka pierwotnego modulo

Pierwiastkiem pierwotnym modulo p, gdzie p jest liczbą pierwszą, jest liczba całkowita g, dla której każda liczba między 1 a p-1 może być zapisana jako potęga g modulo p. Innymi słowy, jeśli weźmiemy dowolną liczbę całkowitą a między 1 a p-1, istnieje liczba całkowita k spełniająca warunek:

$$g^k = a (mod\, p)$$

Ponadto, potęgi $g^1, g^2, ..., g^{(p-1)} modulo\, p$ są wszystkie różne, co oznacza, że g jest generatorem grupy multiplikatywnej liczb całkowitych $modulo\, p$.

Pierwiastki pierwotne są ważne w teorii liczb i kryptografii, ponieważ mają wiele zastosowań w faktoryzacji liczb pierwszych, problemie logarytmu dyskretnego i innych pokrewnych tematach.

1. Oblicz wartość funkcji Eulera dla liczby p.
2. Znajdź wszystkie czynniki pierwsze liczby ϕ.
3. Dla każdej liczby całkowitej z przedziału $[2, p-1]$:
   a) Ustaw zmienną, która będzie przechowywać informację, czy dana liczba jest pierwiastkiem pierwotnym, na wartość `true`.
   b) Dla każdego czynnika pierwszego ϕ, oblicz $exp = \phi / factor$ oraz wynik `modPower(g, exp, p)`.
   c) Jeśli wynik `modPower(g, exp, p)` jest równy 1, ustaw zmienną z punktu (a) na wartość `false` i przerwij pętlę.
   d) Jeśli zmienna z punktu (a) jest nadal równa `true` po przejściu przez wszystkie czynniki, zwróć wartość g.
4. Jeśli nie znaleziono pierwiastka pierwotnego, rzuć wyjątek `IllegalArgumentException` z odpowiednim komunikatem.

```java
public static int findPrimitiveRoot(int p) {
        // Find the totient of p
        int phi = p - 1;
        // Find the prime factors of phi
        List<Integer> factors = primeFactors(phi);
        // Check every integer from 2 to p-1
        for (int g = 2; g <= p-1; g++) {
            boolean isPrimitiveRoot = true;
            // Check if g is a generator of the multiplicative group
            for (int factor : factors) {
                int exp = phi / factor;
                int result = modPower(g, exp, p);
                if (result == 1) {
                    isPrimitiveRoot = false;
                    break;
                }
            }
            // If g is a primitive root, return it
            if (isPrimitiveRoot) {
                return g;
            }
        }
        throw new IllegalArgumentException("No primitive root found for p");
    }
    public static List<Integer> primeFactors(int n) {
        List<Integer> factors = new ArrayList<>();
        for (int i = 2; i <= n/i; i++) {
            while (n % i == 0) {
                factors.add(i);
                n /= i;
            }
        }
        if (n > 1) {
            factors.add(n);
        }
        return factors;
    }
```

Rysunek 17. Algorytm wyznaczania pierwiastka pierwotnego modulo p (gdzie p jest liczbą pierwszą)

4.2.3. Symulacja protokułu wymiany klucza

Na samym początku dokonuje się konfiguracji drona. Wykorzystując struktury danych stworzone na potrzeby wcześniejszej symulacji do stworzenia klucza publicznego wykorzystamy przypisaną na stałe dronowi wartość p. Pozostałe wartości generujemy na jej podstawie. Dla uproszczenia symulacji kluczem, który uzgadniamy jest ID drona.

```java
ControlCenter controlCenter = new ControlCenter(3);
Drone drone1 = new Drone(new Data(0b11011111, 0b00000101));
controlCenter.addDrone(drone1);
```

Rysunek 18. Symulacja konfiguracji drona przed wypuszczeniem do patrolowania

```java
//Setting drone and control center private keys.
int p = drone1.getDroneData().getP(); // prime number p (encryption key
↪  part)
int e1 = DataOperations.findPrimitiveRoot(p); // encryption key part e1
int d = DataOperations.getRandomFromRange(1,p-2); // decryption key d
int e2 = DataOperations.modPower(e1, d, p); // encryption key part e2
int[] publicKey = {e1, e2, p}; // public key
int privateKey = d; // private key
//Printing Keys
System.out.println("Public key components: " +
    Integer.toBinaryString(e1) + " " + Integer.toBinaryString(e2) + " "
↪  + Integer.toBinaryString(p) + " ");
System.out.println("Private key: " + Integer.toBinaryString(d));
```

Rysunek 19. Matematyczne generowanie klucza publicznego i prywatnego

```
//Message 1 - drone sends its ID and generated key.
int r = (int) (Math.random() * (p - 2)) + 1; // random number r
int c1 = DataOperations.modPower(e1, r, p); // first part of ciphertext
int plaintext = drone1.getDroneID(); // plaintext message
int c2 = (DataOperations.modPower(e2, r, p) * plaintext) % p; // second
↪  part of ciphertext
String message1 = String.format("%8s",
    Integer.toBinaryString(c1)).replaceAll(" ", "0") +
↪  String.format("%8s", Integer.toBinaryString(c2)).replaceAll(" ",
↪↪  "0");
System.out.println("Message 1 - from drone to CC: " + message1);
```

Rysunek 20. Wysyłanie wiadomości zaszyfrowanej kluczem publicznym przez drona

```
//Control center decrypts the message and sends a response.
        int receivedC1 = Integer.parseInt(message1.substring(0, 8), 2);
        int receivedC2 = Integer.parseInt(message1.substring(8, 16), 2);
        int response = DataOperations.modPower(receivedC1, p - 1 - privateKey,
        ↪  p) * receivedC2 % p; // response message
        int r2 = (int) (Math.random() * (p - 2)) + 1;
        int c1Response = DataOperations.modPower(e1, r2, p); // first part of
        ↪  response ciphertext
        int c2Response = (DataOperations.modPower(publicKey[1], r2,
            publicKey[2]) * response) % publicKey[2]; // second part of
        ↪↪  response ciphertext
        String message2 = String.format("%8s",
            Integer.toBinaryString(c1Response)).replaceAll(" ", "0") +
            String.format("%8s",
        ↪↪  Integer.toBinaryString(c2Response)).replaceAll(" ", "0");
        System.out.println("Message 2 - from CC to the Drone: " + message2);
```

Rysunek 21. Odszyfrowywanie i odpowiedź C&C

```
Public key components: 11 11011000 11011111
Private key: 1100011
Message 1 - from drone to CC: 1000110100000000
Message 2 - from CC to the Drone: 1010011100000000
```

Rysunek 22. Efekt symulacji

4.3. Wnioski dotyczące modelu i jego dalszego wykorzystania

Model nadaje się do implementacji na układzie FPGA. Należy jednak zauważyć, że algorytm ElGamal wymagał stworzenia większej ilości funkcji w tym nietrywialnej funkcji odnajdywania pierwiastka pierwotnego. Konieczne będzie także zaimplementowanie generatora liczb pseudolosowych i to w wielokrotnie. Analogicznie jak we wcześniej analizowanym protokole zakładamy, że w wypadku przechwycenia drona, CC skreślałoby go z listy kontrolowanych dronów, stąd atak typu Man-in-the-Middle nie stanowi zagrożenia. Model z wykorzystaniem algorytmu ElGamal spełnia założenia projektowe i zapewnia wysoki poziom bezpieczeństwa, zdaje się jednak być jednak rozwiązaniem pochłaniającym więcej zasobów, zarówno jeżeli chodzi o architekturę układu, jak i czas potrzebny na uzgodnienie klucza. Wstrzymujemy się jednak z ostateczną decyzją, gdyż złożoność implementacji w Javie, niekoniecznie musi przełożyć dokładnie na jej wygląd w HDL Verilog.

4.4. Podsumowanie etapu rozwijania

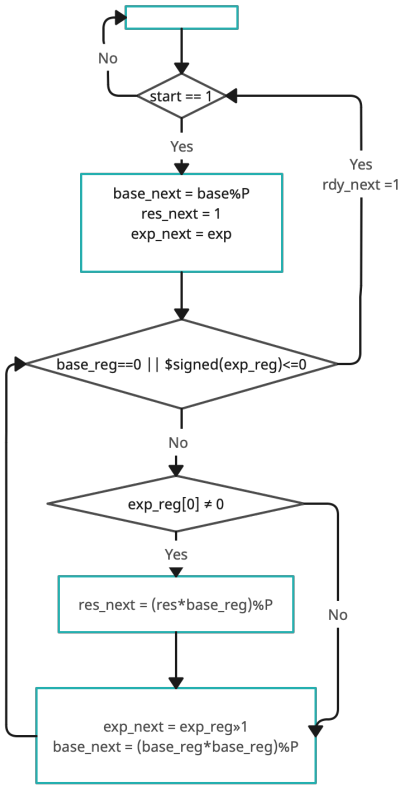
Posiadając gotowe modele działania obu algorytmów, będzie można znacznie łatwiej sporządzić w języku Verilog ostateczne moduły je wykonujące. Celem zespołu jest maksymalna optymalizacja ostatecznego systemu, toteż wykonane zostaną implementacje obu algorytmów i na podstawie ich złożoności wytypowany zostanie ten bardziej pożądany. Z racji, iż klucz będzie zmieniany przy każdorazowej próbie komunikacji, bardziej skomplikowane algorytmy omówione we wcześniejszym etapie (np. RSA) oraz modyfikacje dodające np. możliwość podpisu cyfrowego nie są brane pod uwagę. Korzystanie z łatwiejszych kryptosystemów zgodnie z warunkami zadania zapewnia wystarczające bezpieczeństwo, przy okazji nie wymagając zbyt dużych zasobów. Cały przebieg komunikacji (pomijając małą długość klucza będącą jednym z założeń projektu) zdaje się nie stwarzać zagrożeń dla poufności przesyłanych informacji.

5. Etap definiowania

Pomimo wcześniejszego założenia o implementacji zarówno algorytmu ElGamal jak i protokołu Diffiego-Hellmana, na etapie definiowania, po naradzie zespołu, podjęto decyzję o wyborze protokołu Diffiego-Hellmana jako bazę do docelowego rozwiązania. Odrzucono algorytm ElGamal ze względu na jego złożoność, wymaganą ilość zasobów i większą trudność w implementacji w sprzęcie niż protokołu DH. Przygotowano w języku Verilog moduł potęgowania modularnego inspirując się, przygotowanym na etapie rozwijania, modelem w języku Java. Jako źródło liczb losowych (pseudolosowych) do generowania kluczy prywatnych użyto generatora opartego na rejestrze przesuwającym z liniowym sprzężeniem zwrotnym (LFSR). Przygotowano także moduł drona i moduł control-center (CC). Całość przygotowanej implementacji została załączona w plikach kanału zespołu w Microsoft Teams (8).

5.1. Algorytm szybkiego potęgowania modularnego w języku Verilog

W celu implementacji algorytmu szybkiego potęgowania modularnego w Verilog wykorzystano algorytmiczny układ sekwencyjny ze zintegrowaną ścieżką danych (ASMD). Przygotowano diagram algorytmu (23), który stanowił podstawę do implementacji modułu w języku opisu sprzętu Verilog. Wzorowano się na implementacji algorytmu szybkiego potęgowania modularnego przygotowanej na poprzednim etapie w języku Java.



Rysunek 23. Diagram algorytmu szybkiego potęgowania modularnego

Moduł przygotowany w Verilogu ma dwa parametry - **N** oznaczający maksymalną liczbę bitów zmiennych oraz **P** oznaczający liczbę przez którą resztę z dzielenia wyznacza. Przyjmuje na wejściu sygnały **rst, ena, start**, oraz sygnał zegarowy **clk**. Przyjmuje też dwa wektory o długości N bitów - **base** (podstawa) i **exp** (wykładnik). Ma dwa wyjścia - wyjścia rejestrowe **res** (wynik działania) oraz **rdy** (informacja o wykonaniu operacji).

Przygotowane ASMD ma cztery stany: *idle, init, check, operations*. Posiada też rejestry **state_reg, state_next, base_reg, base_next, exp_reg, exp_next, res_next, res_reg, rdy_next**. Służą one do przechowywania wewnętrznych stanów modułu i odpowiedników zmiennych wewnętrznych z algorytmu. Rejestr przechowujący wykładnik ma rozmiar 9 bitów, ponieważ w trakcie działania modułu istotna jest jego wartość ze znakiem.

```
module modularPowering
#(parameter N=8,
parameter P=89)
(input rst, clk, ena, start,
input [N-1:0] base,
input [N-1:0] exp,
output reg [N-1:0] res,
output reg rdy);
localparam size = 2;
localparam [size-1:0] idle = 3'h0, init = 3'h1, check = 3'h2, operations = 3'h3;
reg [size-1:0] state_reg, state_next;
reg [N-1:0] base_reg,base_next;
reg [N:0] exp_reg, exp_next;
reg [N-1:0] res_next;
reg rdy_next;
```

Rysunek 24. Wejścia, wyjścia i wewnętrzne rejestry modułu szybkiego potęgowania modularnego

Zaimplementowano także instrukcję *always* odpowiedzialną za ładowanie danych do rejestrów oraz ich resetowanie. Logikę stanu następnego zaimplementowano w następujący sposób:

— *Idle* - jeśli otrzymano na wejściu sygnał **start** należy przejść do stanu *init*, w przeciwnym wypadku zostać w *idle*,
— *Init* - po wykonaniu operacji opisanych w logice mikrooperacji należy przejść do stanu *check*.
— *Check* - należy sprawdzić, czy *(base_reg == 0 || $signed(exp_reg)<=0)*. Jeśli tak przypisać do **rdy_next** wartość 1'b1 i wrócić do stanu *idle*. W przeciwnym wypadku przejść do stanu *operations*.
— *Operations* - po wykonaniu operacji należy przejść do stanu *Check*.

Logika mikrooperacji opisana jest poprzez instrukcję *case(state_reg)*, która pozwala na uzależnienie wykonywanych operacji od aktualnego stanu FSM. W stanie *init* następuje załadowanie, poddanych operacjom opisanym na diagramie algorytmu, wartości wektorów wejściowych do odpowiednich rejestrów. W stanie *operations* dane z rejestrów poddawane są odpowiednim operacjom i ładowane do rejestrów w celu wykorzystania ich w następnych stanach.

```
always@(*) begin
    exp_next = exp_reg;
    base_next = base_reg;
    res_next = res;

    case(state_reg)
        init:       begin
                            base_next = base % P;
                            res_next = 1;
                    exp_next = {1'b0,exp};
                    end
        operations: begin
                            if (exp_reg[0] != 0) res_next = (res*base_reg) % P;
                            exp_next = exp_reg >> 1;
                            base_next = (base_reg*base_reg) % P;
                    end
    endcase
end
```

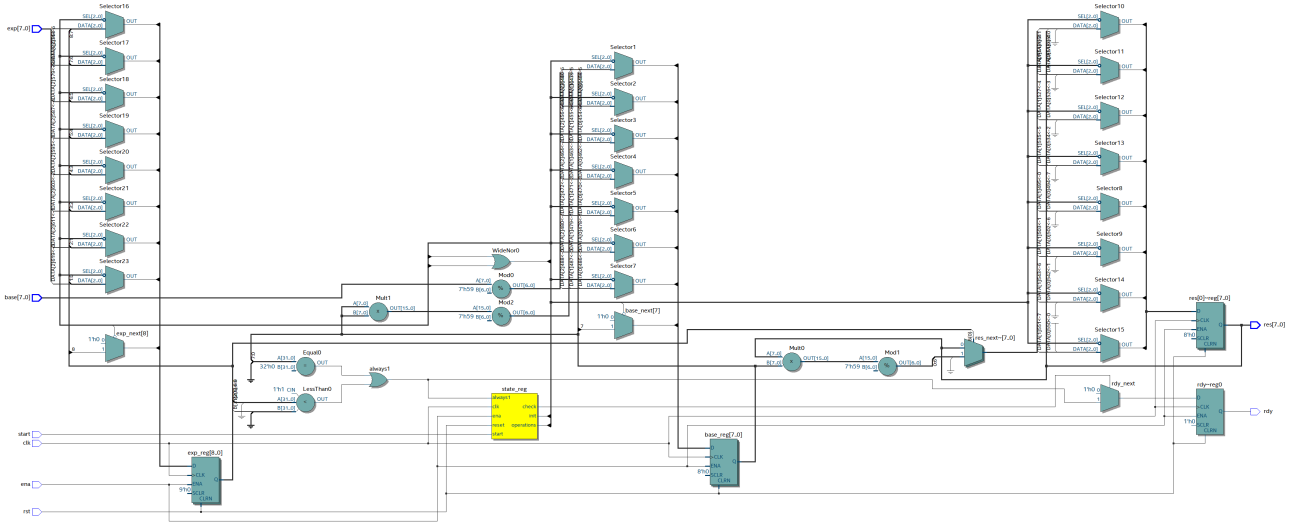
Rysunek 25. Implementacja logiki mikrooperacji algorytmu szybkiego potęgowania modularnego w języku Verilog

5.1.1. Kompilacja modułu szybkiego potęgowania modularnego i jego statystyki

Po skompilowaniu przygotowanego kodu modułu pod platformę Cyclone IV E EP4CE115F29C7 widoczne jest, że moduł używa 436 elementów logicznych oraz 29 pinów (26). Narzędzie CAD wstawiło do niego m.in. jednostki mnożące i jednostki do operacji modulo zawarte na układzie FPGA (27). Moduł może być taktowany zegarem o maksymalnej częstotliwości 35,69 MHz.

Flow Summary	
Flow Status	Successful - Tue May 23 13:17:08 2023
Quartus Prime Version	19.1.0 Build 670 09/22/2019 SJ Lite Edition
Revision Name	dh
Top-level Entity Name	modularPowering
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	436 / 114,480 (< 1 %)
Total combinational functions	430 / 114,480 (< 1 %)
Dedicated logic registers	27 / 114,480 (< 1 %)
Total registers	27
Total pins	29 / 529 (5 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	2 / 532 (< 1 %)
Total PLLs	0 / 4 (0 %)

Rysunek 26. Wynik kompilacji modułu szybkiego potęgowania modularnego - statystki



Rysunek 27. Implementacja przez narzędzie CAD przygotowanego kodu modułu szybkiego potęgowania modularnego

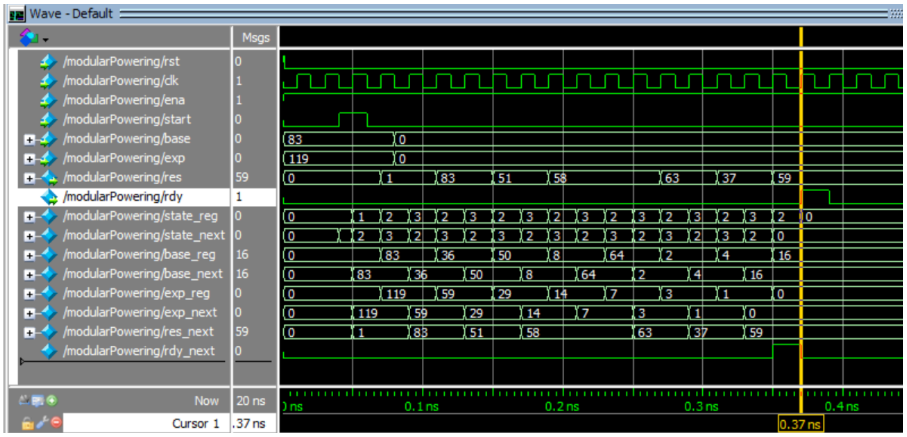
5.1.2. Testy modułu szybkiego potęgowania modularnego

Do przetestowania przygotowanego modułu wykorzystano symulator ModelSim. Przeprowadzono wiele testów, jednak w raporcie opisano test dla parametru P = 89 (liczba przez którą reszta z dzielenia jest poszukiwana). Przygotowano skrypt narzędzia ModelSim do symulacji (28). W raporcie opisano obliczanie przez moduł wartości $83^{119} \mod 89$.

```
restart -nowave -force
add wave -radix unsigned *
force clk 0 0, 1 10 -r 20
force rst 1 0, 0 1
force ena 1 0
force start 0 0, 1 40, 0 60
force base 2#01010011 0,10#0 80
force exp  2#01110111 0, 10#0 80
run 20000
```

Rysunek 28. Jeden ze skryptów narzędzia ModelSim przygotowanych do testowania modułu szybkiego potęgowania modularnego

Zgodnie z kalkulatorem matematycznym zapewnianym przez wyszukiwarkę Microsoft Bing wartość 83^{119} mod 89 wynosi 59. Jak wynika z symulacji, taka wartość uzyskiwana jest na wyjściu res, gdy wartość wyjścia **rdy** wynosi 1'b1 (29). Świadczy to o poprawności działania modułu dla tego przypadku. Czas potrzebny do osiągnięcia przez moduł gotowego wyniki wynosi 0,37 ns.



Rysunek 29. Wynik symulacji obliczania przez moduł wyniku działania 83^{119} mod 89

5.2. Generator liczb pseudolosowych

Do generowania liczb będących kluczami prywatnymi CC i drona wykorzystano generator oparty na rejestrze przesuwającym z liniowym sprzężeniem zwrotnym (LFSR). Generator oparto na wielomianie $x^{13}+x^4+x^3+x+1$ co zapewnia okres sekwencji równy 8191 cykli zegara. Na wyjściu **stream** pojawia się 8 bitów będących źródłem klucz prywatnych. Moduł taktować zegarem 802.57 MHz, jednak układ Cyclone IV E EP4CE115F29C7 ma zegar ograniczony do 250 MHz.

```
module random_generator(
input  rst, clk, ena,
output [7:0] stream
);
reg  [12:0] lfsr_reg;
wire [12:0] lfsr_next;
wire feedback;
always@(posedge clk or posedge rst) begin
    if (rst)
        lfsr_reg <= 13'b1001001000101;
    else if (ena)
        lfsr_reg <= lfsr_next;
end
assign feedback = lfsr_reg[12] ^ lfsr_reg[4] ^ lfsr_reg[3] ^ lfsr_reg[1];
assign lfsr_next = {lfsr_reg[11:0], feedback};
assign stream = lfsr_reg[9:2];

endmodule
```

Rysunek 30. Implementacja modułu generatora liczb pseudolosowych w języku Verilog

5.3. Moduł drona

Moduł drona zrealizowano jako FSM. Jego inicjalizacja następuje w momencie wynurzenia. Przy każdym nawiązaniu komunikacji z CC, pobierany jest nowy klucz prywatny z generatora liczb pseudolosowych LFSR. Operacje potęgowania modularnego wykonuje moduł szybkiego potęgowania modularnego.

Moduł jest parametryzowany liczbą bitów klucza (**N**) oraz liczbę pierwszą (**P**) będącą liczbą, przez którą reszta z dzielenia używana jest w operacjach potęgowania modularnego. Jego wejścia to **init, received** służące kolejno jako inicjalizator komunikacji i informacja o otrzymaniu wiadomości od CC, oraz **mess_input** będący

wejściem na którym pojawia się wiadomość od CC z jego kluczem publicznym. Moduł drona posiada też wejścia **rst, clk, ena** będące kolejno wejściem resetu, zegarowym i aktywującym drona.

Moduł drona posiada następujące wewnętrzne rejestry:

— **id** - unikalny identyfikator drona, pozwalający centrum command&control na rozpoznanie nadającej jednostki (dzięki czemu wie on jakiej liczby pierwszej i generatora jej grupy użyć).

— **privateKey_reg** - przechowuje wartość wygenerowaną przez generator liczb pseudolosowych, będącą kluczem prywatnym w danej sesji. Akutualizuje ją tylko gdy dron otrzyma sygnał *init* (koresponduje to ze stanem *idle*).

— **modPower_base** - to on przesyła wartość podstawy potęgi modułowi szybkiego potęgowania modularnego. Jest to wartość generatora grupy lub klucza publicznego CC (**mess_input**) zależnie od stanu sesji komunikacji z CC.

— **start** - jest to wewnętrzna zmienna, sterująca uruchamianiem modułu szybkiego potęgowania modularnego. Jej wartość zmienia się z 0 na 1 gdy następny stan w rejestrze stanów jest różny od następnego - czyli gdy dron przetwarza dane.

— **G** - przechowuje generator grupy multiplikatywnej $\mathbb{Z}_P$.

Moduł posiada też obiekty wire **privateKey_stream** będący strumieniem liczb ośmiobitowych z LFSR oraz **rdy_modPower** będący wyjściem, na którym pojawia się informacja o wykonaniu działania potęgowania modularnego.

Dron jest automatem, który posiada trzy stany. Steruje nimi standardowy rejestr stanów (**state_reg, state_next**). Moduł drona posiada następujące stany:

— *idle* - w tym stanie dron czeka na sygnał **init**, rozpoczynający jego działanie i wymuszający pobranie do rejestru **privateKey_reg** aktualnej wartości podawanej przez generator na wejściu *privateKey_stream*, która jest kluczem prywatnym danej sesji i jest podawana do modułu potęgowania modularnego. Gdy na wejściu pojawi się sygnał **init**, automat przechodzi do stanu *send_mess*. W przeciwnym wypadku zostaje w stanie *idle*.

— *send_mess* - w tym stanie dron oblicza klucz publiczny, który wyśle do CC. Jeżeli na wejściu **received** podawana jest wartość 1 (CC wysłało swój klucz publiczny), przesyła odebraną wiadomość do modułu potęgowania jako podstawę potęgi i przechodzi do następnego stanu. *compute_key*. W obecnym stanie zostaje dopóki nie otrzyma sygnału **received**.

— *compute_key* - w tym stanie dron czeka aż moduł szybkiego potęgowania modularnego zakończy swoją pracę, co oznacza obliczenie klucza, i przechodzi do stanu *idle*.

```
always@(*) begin
    case(state_reg)
        idle: if (init)
                begin
                                state_next = send_mess;
                                privateKey_reg = privateKey_stream;
                                modPower_base = G;
                end
                else state_next = idle;
        send_mess: if(received)
                begin
                    modPower_base = mess_input;
                    state_next = compute_key;
                end
                else state_next = send_mess;
        compute_key: if (!key_rdy) state_next = compute_key;
                     else state_next = idle;
        default: state_next = idle;
    endcase
end
```

Rysunek 31. Logika stanu następnego oraz operacje wykonywane przez moduł drona opisane w języku Verilog

Moduł drona posiada następujące wyjścia:
— wektor **mess_out** będący wiadomością wysyłaną przez drona do CC. Jest 16-bitowy. Jego 8 najstarszych bitów to identyfikator drona a pozostałe 8 to obliczony klucz prywatny.
— **mess_rdy** informujące o przygotowaniu przez drona wiadomości inicjalizującej dla CC.
— **key** obliczony ośmiobitowy klucz.
— **key_rdy** informujące o obliczeniu klucza.

```
assign mess_out = {id, res_modPower};
assign mess_rdy = (rdy_modPower & state_reg == send_mess) ? 1'b1:1'b0;
assign key = (rdy_modPower & state_reg == compute_key) ? res_modPower:8'b0;
assign key_rdy = (rdy_modPower & state_reg == compute_key) ? 1'b1:1'b0;
```

Rysunek 32. Logika przypisania wartości na wyjściach modułu drona opisane w języku Verilog

5.3.1. Kompilacja modułu drona i jego statystyki

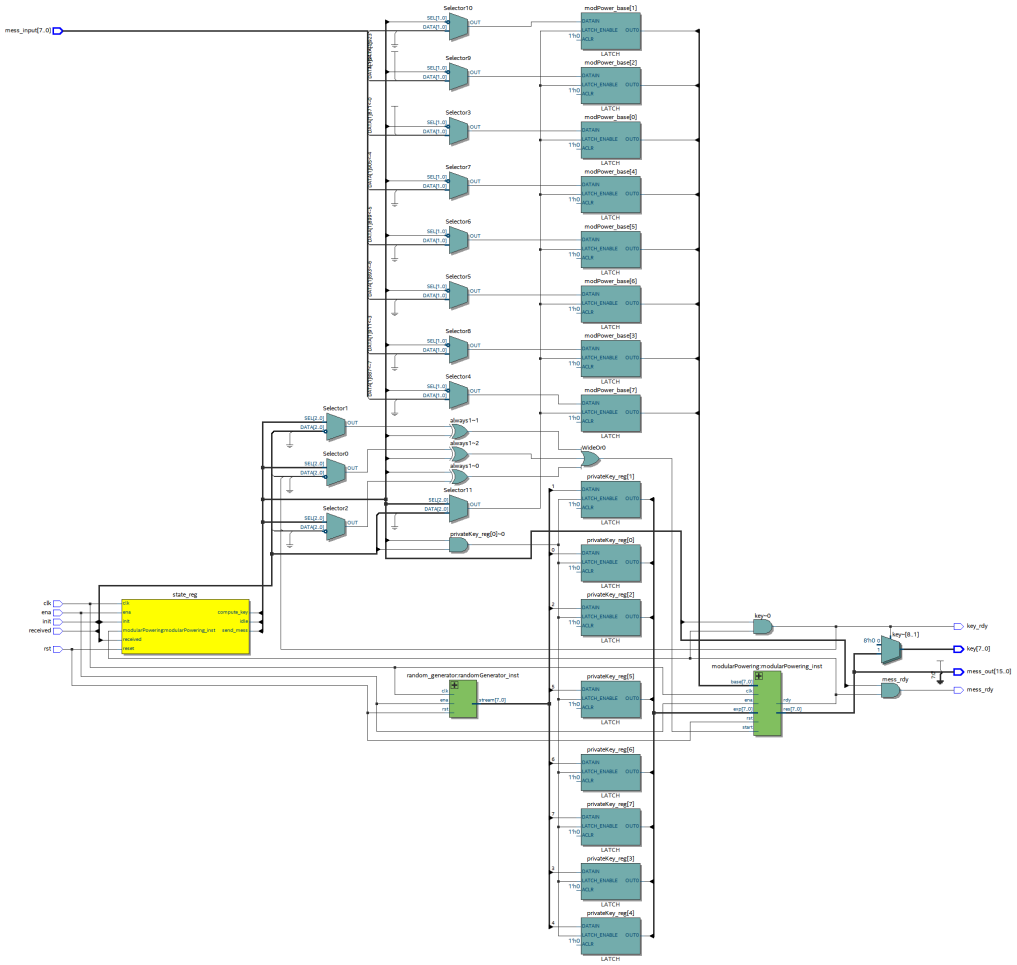
Po skompilowaniu przygotowanego kodu modułu dron pod platformę Cyclone IV E EP4CE115F29C7 widoczne jest, że moduł używa 566 elementów logicznych (< 1% elementów logicznych FPGA) oraz 39 pinów (33). Można go taktować zegarem o maksymalnej częstotliwości 31,39 MHz (34). Widok implementacji przygotowanej przez narzędzie CAD (35) jest zgodny z oczekiwaniami zespołu.

```
+-------------------------------------------------------------------------------+
; Flow Summary                                                                  ;
+-----------------------------------+-------------------------------------------+
; Flow Status                       ; Successful - Tue May 23 13:22:52 2023       ;
; Quartus Prime Version             ; 19.1.0 Build 670 09/22/2019 SJ Lite Edition ;
; Revision Name                     ; dh                                        ;
; Top-level Entity Name             ; drone                                     ;
; Family                            ; Cyclone IV E                              ;
; Device                            ; EP4CE115F29C7                             ;
; Timing Models                     ; Final                                     ;
; Total logic elements              ; 566 / 114,480 ( < 1 % )                   ;
;     Total combinational functions ; 560 / 114,480 ( < 1 % )                   ;
;     Dedicated logic registers     ; 45 / 114,480 ( < 1 % )                    ;
; Total registers                   ; 45                                        ;
; Total pins                        ; 39 / 529 ( 7 % )                          ;
; Total virtual pins                ; 0                                         ;
; Total memory bits                 ; 0 / 3,981,312 ( 0 % )                     ;
; Embedded Multiplier 9-bit elements ; 2 / 532 ( < 1 % )                        ;
; Total PLLs                        ; 0 / 4 ( 0 % )                             ;
+-----------------------------------+-------------------------------------------+
```

Rysunek 33. Wynik kompilacji modułu drona - statystki

```
+-------------------------------------------------+
; Slow 1200mV 85C Model Fmax Summary              ;
+-----------+-----------------+------------+------+
; Fmax      ; Restricted Fmax ; Clock Name ; Note ;
+-----------+-----------------+------------+------+
; 31.39 MHz ; 31.39 MHz       ; clk        ;      ;
+-----------+-----------------+------------+------+
```

Rysunek 34. Wynik kompilacji modułu drona - taktowanie zegara



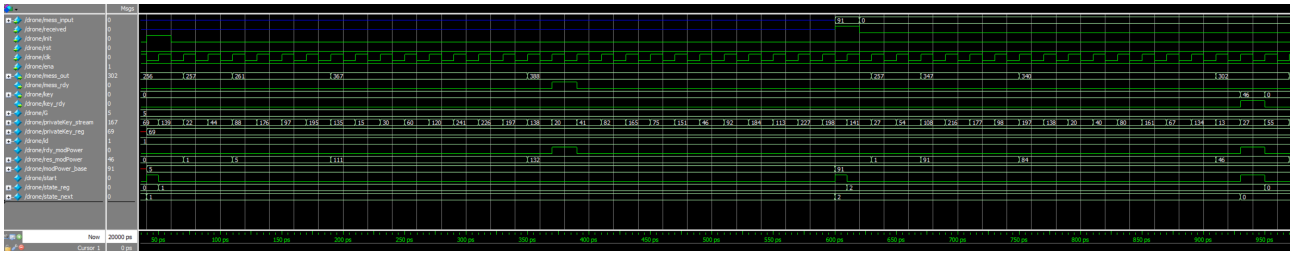
Rysunek 35. Implementacja przez narzędzie CAD przygotowanego kodu modułu drona

5.3.2. Testy modułu drona

Do testowania przygotowanego modułu drona wykorzystano narzędzie ModelSim. Przygotowano wektory testowe (36), w taki sposób aby symulowały one sesję komunikacji z CC. Dla testowanego drona wartości zostały zakodowanego następujące wartości: P = 137 (liczba pierwsza), G = 5 (generator grupy $\mathbb{Z}_P$).

```
restart -nowave -force
add wave -radix unsigned *
force clk 0 0, 1 10 -r 20
force rst 1 0, 0 1
force ena 1 0
force init 0 20, 1 40, 0 60

force received 1 600 , 0 620
force mess_input 10#91 600, 0 620
run 5000
```

Rysunek 36. Skrypt narzędzia ModelSim do testowania modułu drona



Rysunek 37. Wynik symulacji modułu drona w narzędziu ModelSim

Jak wynika z symulacji (37) wiadomość dla CC jest gotowa do wysłania po 330 ms od wynurzenia. Taki sam okres czasu zajmuje dronowi obliczenie klucza do szyfrowania, po otrzymaniu klucza publicznego CC. Na podstawie obliczeń wykonanych z wykorzystaniem kalkulatora, stwierdzono, że moduł drona działa poprawnie. Dodatkowo, przy danych symulujących rzeczywistą wymianę, w dronie i w CC zostanie uzyskany taki sam klucz.

5.4. Moduł Control-Center

Moduł ten posiada wiele tych samych komponentów i cech struktury co moduł drona: również oparty jest na automacie skończonym, tak samo posiada generator liczb pseudolosowych oparty na tym samym module LFSR oraz moduł szybkiego potęgowania modularnego. Również parametryzowany jest liczbą bitów klucza (**N**) i liczbą pierwszą (**P**). W celu dodania do CC kolejnych dronów należy zakodować dla nich dodatkowe parametry. Poza wejściami **rst, clk, ena** posiada wejścia **drone_rdy, mess_input**. Pierwsze z nich sygnalizuje, że można odebrać wiadomość od drona (że jest to już ostateczna, stabilna wersja wiadomości), na drugim podawana jest sama treść wiadomości od drona. Moduł CC posiada następujące wewnętrzne rejestry, analogiczne do rejestrów drona:

— **id** - identyfikator drona, z którym następuje komunikacja.
— **privateKey_reg** - przechowuje wartość wygenerowaną przez generator liczb pseudolosowych, będącą kluczem prywatnym w danej sesji. Dzieje się to tylko, gdy CC otrzyma od drona sygnał *drone_rdy* (koresponduje to ze stanem *listening*).
— **modPower_base** - przesyła wartość podstawy potęgi modułowi szybkiego potęgowania modularnego. Jest to wartość generatora grupy lub klucza prywatnego CC (**mess_input**) zależnie od stanu sesji.
— **start** - jest to wewnętrzna zmienna, sterująca uruchamianiem modułu szybkiego potęgowania modularnego. Jej wartość zmienia się z 0 na 1 gdy następny stan w rejestrze stanów jest różny od następnego - czyli gdy CC przetwarza dane.
— **G** - przechowuje generator grupy multiplikatywnej liczby pierwszej $\mathbb{Z}\_P$.

Posiada jednak również unikalne rejestry, stworzone na potrzeby jego poprawnego działania. Jako, że CC, otrzymawszy od drona 1. część klucza, musi najpierw odesłać mu swoją 1. część, a potem obliczyć ostateczny klucz na podstawie danych od drona, dane te muszą być przechowane w module CC. Stąd też, rejestr **received_first_stage** służy przechowaniu wiadomości od drona, by móc na niej operować po zakończeniu transmisji do drona.

Z różnic między funkcjonowaniem obu modłów wynika też różne nazewnictwo ich trzech stanów, jednakże ich funkcje pozostały niemal takie same:

— *listening* - w tym stanie CC czeka na sygnał **drone_rdy**, rozpoczynający jego działanie oraz oznaczający konieczność pobierania do rejestru aktualnej wartości podawanej przez generator na wejściu **privateKey_stream**, która jest kluczem prywatnym danej sesji i jest podawana do modułu potęgowania modularnego. W tym stanie również zczytana do rejestru zostaje wiadomość od drona. W przypadku braku sygnału **drone_rdy**, CC pozostaje w stanie *listening*.
— *compute_second_stage* - w tym stanie CC oblicza klucz publiczny, który wyśle do drona. Kiedy **cc_rdy** przyjmie wartość 1 (czyli kiedy wyjście modułu szybkiego potęgowania modularnego się ustabilizje) przechodzi do stanu *compute_key*. Do tego momentu pozostaje w stanie *compute_second_stage*.
— *compute_key* - w tym stanie CC oblicza swoją drugą część klucza. Robi to poprzez oddzielenie najmłodszych 8 bitów zapisanej wiadomości od drona i przekazanie ich jako podstawa potęgi do modułu szybkiego potęgowania modularnego. Kiedy **key_rdy** zmienia wartość na 1 (czyli moduł wykonał działania - obliczył klucz), przechodzi do stanu *listening*.

```
always@(*) begin
    case(state_reg)
        listening        :
        if(drone_rdy) begin
            privateKey_reg = privateKey_stream;
                received_first_stage = mess_input;
                state_next = compute_second_stage;
        end
        else state_next = listening;
        compute_second_stage        :
        begin
                modPower_base = G;
                if(cc_rdy)state_next = compute_key;
                else state_next = compute_second_stage;
        end
        compute_key:
        begin
                modPower_base = received_first_stage[7:0];
                if(key_rdy) state_next = listening;
                else state_next = compute_key;
        end
        default: state_next = listening;
    endcase
    if(state_reg!=state_next) start<=1'b1;
    else start<=1'b0;
end
```

Rysunek 38. Logika stanu następnego oraz operacje wykonywane przez moduł CC opisane w języku Verilog

Moduł CC posiada następujące wyjścia:

— **mess_out** - 8 bitowa wiadomość do drona, zawierająca pierwszą część obliczeń CC. Nie posiada identyfikatora, gdyż nie jest on dronowi potrzebny.

— **mess_rdy** - flaga pokazująca dronowi, że może odczytać wiadomość od CC. Przyjmuje wartość 1, tylko gdy moduł potęgowania zakończy swoje obliczenia w stanie *compute_second_stage*.

— **key** - gotowy klucz ustalony protokołem Diffiego-Hellmana po stronie CC. Zapisywany jest on tylko, gdy moduł potęgowania zakończy swoje obliczenia w stanie *compute_key*.

— **key_rdy** - flaga pokazująca, że klucz jest gotowy do użycia.

```
assign mess_out = (rdy_modPower & state_reg == compute_second_stage) ? res_modPower:8'b0;
assign mess_rdy = (rdy_modPower & state_reg == compute_second_stage) ? 1'b1:1'b0;

assign key = (rdy_modPower & state_reg == compute_key) ? res_modPower:8'b0;
assign key_rdy = (rdy_modPower & state_reg == compute_key) ? 1'b1:1'b0;
```

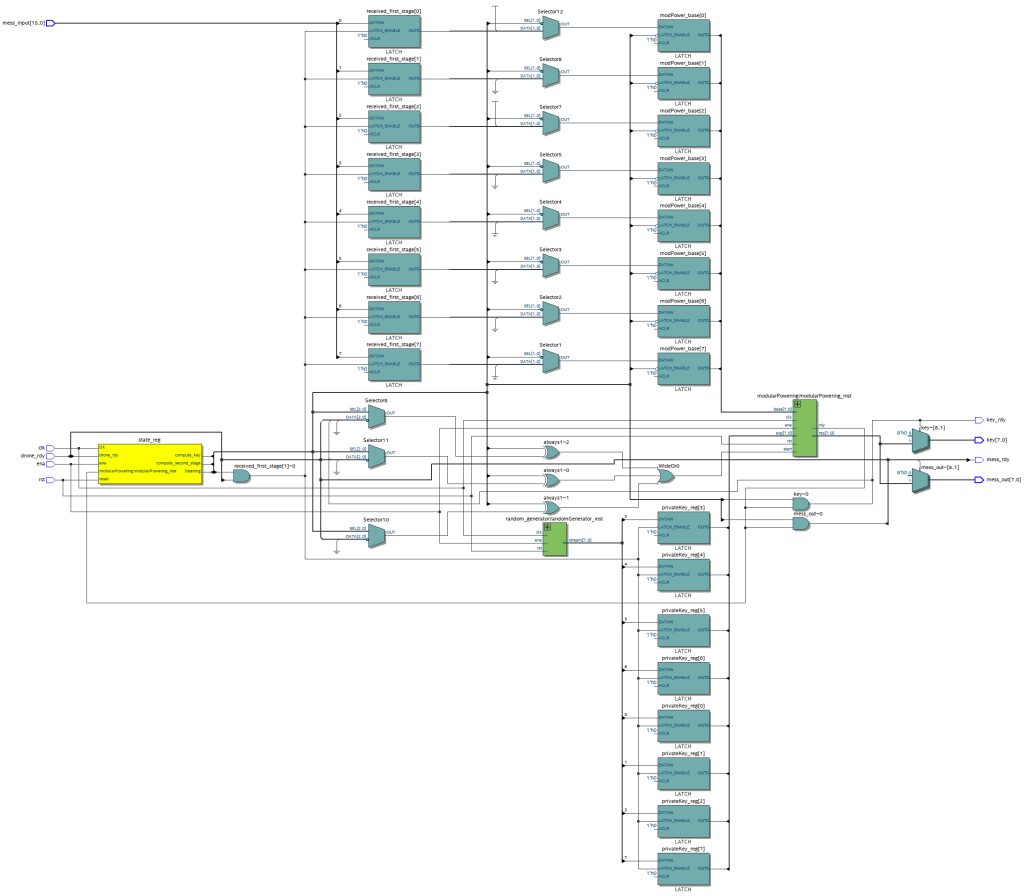
Rysunek 39. Logika przypisująca wartości kluczowi i wiadomości, bazując na stanach komponentów modułu CC

5.4.1. Kompilacja modułu CC i jego statystyki

Po skompilowaniu przygotowanego kodu modułu CC pod platformę Cyclone IV E EP4CE115F29C7 widoczne jest, że moduł używa 581 elementy logiczne (mniej niż 1% elementów logicznych FPGA) oraz 38 pinów 40. Można go taktować zegarem o maksymalnej częstotliwości 31,07 MHz 42. Widok implementacji przygotowanejprzez narzędzie CAD jest zgodny z oczekiwaniami zespołu (41).

Flow Summary	
Flow Status	Successful - Tue May 23 13:43:10 2023
Quartus Prime Version	19.1.0 Build 670 09/22/2019 SJ Lite Edition
Revision Name	dh
Top-level Entity Name	cc
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	581 / 114,480 (< 1 %)
Total combinational functions	575 / 114,480 (< 1 %)
Dedicated logic registers	45 / 114,480 (< 1 %)
Total registers	45
Total pins	38 / 529 (7 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	2 / 532 (< 1 %)
Total PLLs	0 / 4 (0 %)

Rysunek 40. Wynik kompilacji modułu CC - statystki



Rysunek 41. Implementacja przez narzędzie CAD przygotowanego kodu modułu CC

```
+-----------------------------------------------------------+
; Slow 1200mV 85C Model Fmax Summary                        ;
+------------+-----------------+----------------------+------+
; Fmax       ; Restricted Fmax ; Clock Name           ; Note ;
+------------+-----------------+----------------------+------+
; 31.07 MHz  ; 31.07 MHz       ; clk                  ;      ;
; 200.16 MHz ; 200.16 MHz      ; state_reg.listening  ;      ;
+------------+-----------------+----------------------+------+
```

Rysunek 42. Wynik kompilacji modułu CC - taktowanie zegara

5.4.2. Testy modułu CC

Do przetestowania modułu CC również wykorzystany został ModelSim. Pokazany na dole test (44) odbył się dla parametrów P = 137, N = 8. Moduł przyjął wiadomość o wartości 388 (taką samą jak wiadomość wysłana przez moduł drona w jego teście) i zwrócił wiadomość 91 (użyta w teście drona).

$$388_{10} = 0000000110000100_2$$

po odtrąceniu najstarszych 8 bitów identyfikatora, klucz publiczny drona to:

$$10000100_2 = 132_{10}$$

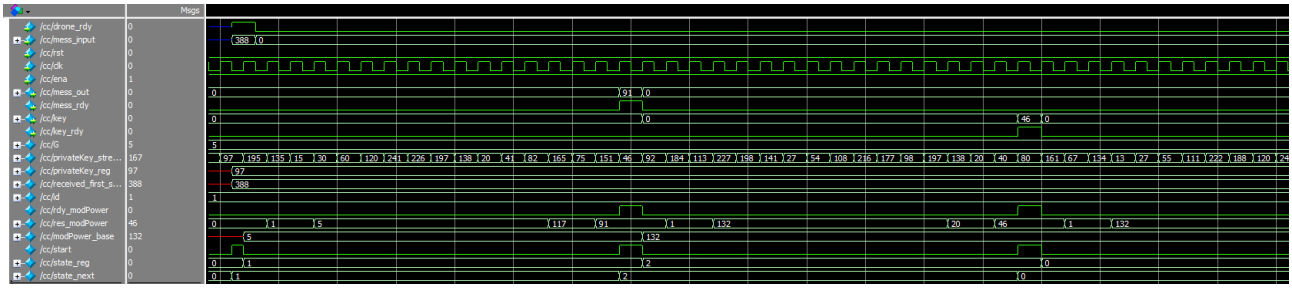
privateKey_reg = 97 w momencie nadania sygnału **drone_rdy**, zatem przyjmuje taką wartość na całą sesję.

$$132^{97} \ mod \ 137 = 46$$

I to też jest nasz wynik - **key** (taki sam jak u drona). Moduł działa poprawnie.

```
restart -nowave -force
add wave -radix unsigned *
force clk 0 0, 1 10 -r 20
force rst 1 0, 0 1
force ena 1 0

force drone_rdy 1 160, 0 180
force mess_input 10#388 160, 0 180

run 20000
```

Rysunek 43. Skrypt narzędzia ModelSim do testowania modułu CC



Rysunek 44. Wynik symulacji modułu CC w narzędziu ModelSim

5.5. Podsumowanie etapu definiowania

W ramach etapu definiowania udało się zrealizować początkowe założenia i wykonać znaczny postęp w rozwoju projektu. W tymże etapie dokonano ostatecznego wyboru protokołu uzgadniania klucza i pomyślnie zaimplementowano potrzebne moduły w języku opisu sprzętu Verilog. Dzięki szczegółowej i starannie przygotowanej implementacji protokołu Diffiego-Hellmana w języku Java w ramach etapu rozwijania, implementacja w docelowym języku okazała się łatwiejsza i bardziej przejrzysta.

Po przeprowadzeniu testów przygotowanej implementacji okazało się, że cała operacja wraz z wymianą danych (na tym etapie) zajmuje ok. 1 sekundy. Stwierdzono, że wszystkie przygotowane moduły działają poprawnie, zgodnie z założeniami wybranego protokołu. W kolejnym etapie, poza realizacją założonych w jego

ramach celów i dokładniejszym przetestowaniem ostatecznej wersji, dołożone zostaną starania aby możliwie jeszcze bardziej zoptymalizować działanie końcowej implementacji. Widoczne są perspektywy na optymalizację modułu potęgowania modularnego oraz rozbudowę modułu CC o kolejne drony.

6. Etap dostarczania

Na etapie dostarczania połączono przygotowane na etapie definiowania moduły w jeden moduł talk_simulation (6.2). Jest to „moduł rozmowy". Prezentuje on sytuację, w której po otrzymaniu sygnału **init** od drona, następuje uzgodnienie klucza. Na etapie definiowania dokonano także zmian w module drona i CC, umożliwiających obsługę wielu dronów. Przygotowano rozbudowane wektory testowe, umożliwiające weryfikację poprawności ustalenia wspólnego klucza.

6.1. Zmiany w modułach drona i CC na etapie definiowania

Zmiany w modułach drona i CC były konieczne w celu umożliwienia obsługi przez CC wielu dronów. Dotyczyły one przede wszystkim dodawania kolejnych zmiennych i rejestrów do modułów. Moduł drona oraz CC ponownie przetestowano w celu potwierdzenia poprawności zmian.

6.1.1. Zmiany w module drona

W module drona konieczne było dodanie dwóch nowych parametrów - **Prime, Generator** - liczba pierwsza i jej generator będące danymi uzgodnionymi wcześniej z CC, a także **ID** w celu łatwiejszego dodawania kolejnych dronów. Wartości tych parametrów przepisywane są do odpowiednich rejestrów, zaimplementowanych w poprzednim etapie. Ze względu na to, że są one stałe, wartości te są zatrzaskiwane.

Dodano także **output reg wait_for_cc** będące wyjściem, które informuje moduł CC, że dron czeka na odpowiedź. Było to konieczne, aby umożliwić CC wybór właściwego drona do wysłania wiadomości. Jego wartość ustawiana jest na 1 przy przejściu ze stanu *idle* do *send_mess* oraz ustawiana na 0 - po otrzymaniu wiadomości (sygnał **received**).

Po opisanych zmianach, taktowanie zegara i ilość zużywanych zasobów w module drona praktycznie nie uległy zmianie. Moduł drona zużywa obecnie 570 elementów logicznych (w porównaniu do 566 elementów w module na etapie definiowania (33) oraz jest taktowany zegarem o częstotliwości 30.96 MHz (31.39 MHz na etapie definiowania (34)).

6.1.2. Zmiany w module CC

W module CC także konieczna była implementacja dodatkowych parametrów. Dla obsługiwanego drona (obecny moduł obsługuje dwa drony) dodano parametry **droneX_prime, droneX_generator**, gdzie X zastępowany jest przez ID drona. Dodano także lokalne parametry w celu łatwiejszego dodawania większej ilości dronów (**localparam [drones_bits-1:0] drone 1 = 1, drone2 = 2**).

Zewnętrzne wyjścia i wejścia modułu CC pozostały bez zmian. Rejestr **G** zmienił swoją funkcję - aktualnie przechowuje on wartość uzgodnionego z aktualnym dronem generatora grupy multiplikatywnej. Dodano także rejestr **reg [N-1:0] id** przechowujący identyfikator aktualnie obsługiwanego drona.

Ze względu na to, że moduł potęgowania modularnego jest parametryzowany liczbą pierwszą, przez którą resztę z dzielenia oblicza, CC posiada osobny moduł potęgowania modularnego dla każdego drona. Zwiększa to jednak szybkość działania modułu. Konieczne było utworzenie dodatkowych wejść do modułów potęgowania modularnego w celu obsługi wielu dronów oraz implementacja logiki obsługi wielu dronów (45).

```
reg [N-1:0] id;
wire rdy_modPower1,rdy_modPower2;
wire [N-1:0] res_modPower1, res_modPower2;
reg [N-1:0] modPower_base1,modPower_base2;
reg start;
reg start1,start2;

always@(*) begin
case(id)
drone1: begin
G = drone1_generator;
modPower_base1 = modPower_base;
start1 = start;
end
drone2: begin
G = drone2_generator;
modPower_base2 = modPower_base;
start2 = start;
end
endcase
end

assign res_modPower =  (id==drone1) ? res_modPower1:res_modPower2;
assign rdy_modPower =  (id==drone1) ? rdy_modPower1:rdy_modPower2;
```

Rysunek 45. Dodatkowe rejestry i zmienne typu wire związane z obsługą wielu dronów, logika używania wielu modułów potęgowania modularnego.

Po opisanych zmianach moduł CC zużywa prawie dwukrotnie więcej zasobów, ze względu na użycie dwóch jednostek potęgowania modularnego (pochłaniających najwięcej zasobów). Taktowanie zegara spadło jednak tylko nieznacznie. Moduł CC zużywa obecnie 1130 elementów logicznych (w porównaniu do 581 elementów w module na etapie definiowania (40) oraz jest taktowany zegarem o częstotliwości 30.23 MHz (31.07 MHz na etapie definiowania (42)).

6.2. Moduł talk_simulation

Moduł talk_simulation to moduł łączący dwa drony i CC w „rozmawiające ze sobą moduły". Umożliwia on obserwację realnego przepływu wiadomości i testowanie prawidłowości oraz jednakowości uzgodnionego klucza po obu stronach.

Poza standardowymi wejściami dla modułów synchronicznych - **rst, clk, ena**, moduł talk_simulation posiada wejścia **init1, init2** będące sygnałami o konieczności uzgodnienia klucza odpowiednio od drona 1 i drona 2. Wyjścia modułu talk_simulation to **key_drone1, key_drone2, key_CC** będące uzgodnionymi kluczami oraz **key_rdy_drone1, key_rdy_drone2,key_rdy_CC** będące sygnałami informującymi o uzgodnieniu klucza. Moduł talk_simulation posiada także zmienne typu wire wykorzystywane jako wejścia i wyjścia modułów dronów i CC oraz ich logikę (46).

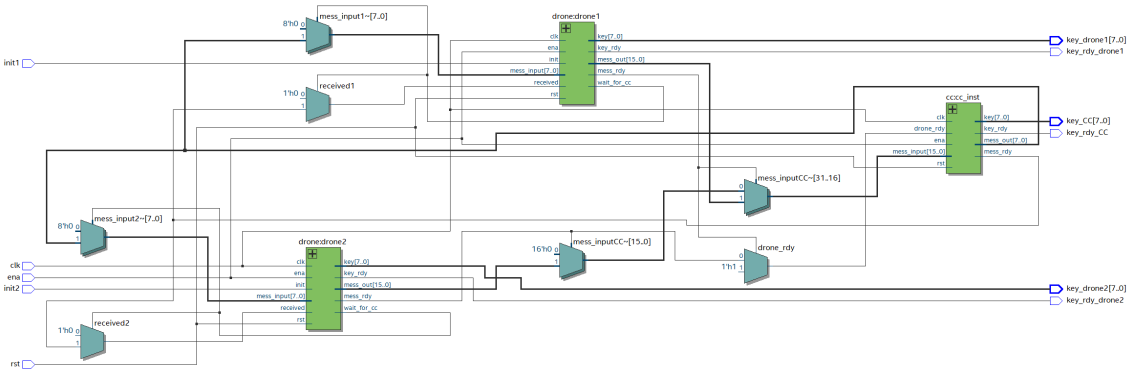
```
wire [N-1:0] mess_input1, mess_input2,mess_outCC;
wire received1,received2;
wire [2*N-1:0] mess_out1, mess_out2, mess_inputCC;
wire mess_rdy1, mess_rdy2;
wire drone_rdy,cc_rdy;
wire wait_for_cc_drone1, wait_for_cc_drone2;

assign mess_inputCC = (mess_rdy1 == 1'b1) ? mess_out1 : (mess_rdy2 == 1'b1) ? mess_out2 : 0;
assign drone_rdy = (mess_rdy1 == 1'b1) ? mess_rdy1 : (mess_rdy2 == 1'b1) ? mess_rdy2 : 0;
assign mess_input1 = (wait_for_cc_drone1 == 1'b1) ? mess_outCC:0;
assign mess_input2 = (wait_for_cc_drone2 == 1'b1) ? mess_outCC:0;
assign received1 = (wait_for_cc_drone1 == 1'b1) ? cc_rdy:0;
assign received2 = (wait_for_cc_drone2 == 1'b1) ? cc_rdy:0;
```

Rysunek 46. Zmienne typu wire modułu talk_simulation oraz ich logika.

W module talk_simulation drone1 został sparametryzowany liczbą pierwszą 137 i generatorem jej grupy 5, natomiast drone2 - liczbą pierwszą 229 i generatorem jej grupy 2.

6.2.1. Kompilacja modułu talk_simulation i jego statystyki

Moduł talk_simulation zużywa 2245 elementów logicznych (2% zasobów). Jest to równe w przybliżeniu sumie zasobów dwóch dronów i CC. Można go taktować zegarem o maksymalnej częstotliwości 29.86 MHz.



Rysunek 47. Implementacja przez narzędzie CAD przygotowanego kodu modułu talk_simulation

Flow Summary	
Flow Status	Successful - Mon Jun 05 19:49:38 2023
Quartus Prime Version	19.1.0 Build 670 09/22/2019 SJ Lite Edition
Revision Name	dh
Top-level Entity Name	talk_simulation
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	2,245 / 114,480 (2 %)
Total combinational functions	2,226 / 114,480 (2 %)
Dedicated logic registers	138 / 114,480 (< 1 %)
Total registers	138
Total pins	32 / 529 (6 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	8 / 532 (2 %)
Total PLLs	0 / 4 (0 %)

Rysunek 48. Wynik kompilacji modułu talk_simulation - statystki

```
+-----------------------------------------------------------------+
; Slow 1200mV 85C Model Fmax Summary                               ;
+------------+-----------------+-------------------------------+------+
; Fmax       ; Restricted Fmax ; Clock Name                     ; Note ;
+------------+-----------------+-------------------------------+------+
; 29.86 MHz  ; 29.86 MHz       ; clk                           ;      ;
; 68.52 MHz  ; 68.52 MHz       ; cc:cc_inst|id[0]              ;      ;
; 221.24 MHz ; 221.24 MHz      ; init2                         ;      ;
; 243.43 MHz ; 243.43 MHz      ; cc:cc_inst|state_reg.listening ;      ;
; 246.73 MHz ; 246.73 MHz      ; init1                         ;      ;
+------------+-----------------+-------------------------------+------+
```

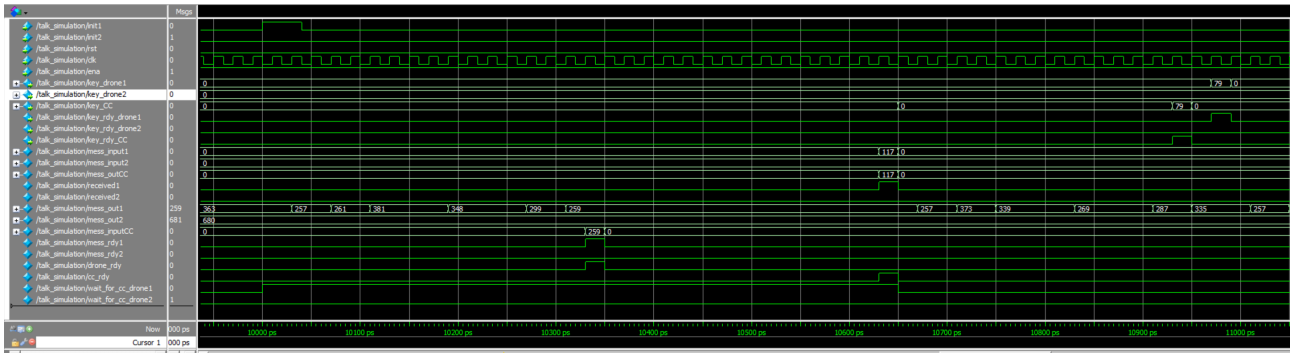
Rysunek 49. Wynik kompilacji modułu talk_simulation - taktowanie zegara

6.3. Testy projektu

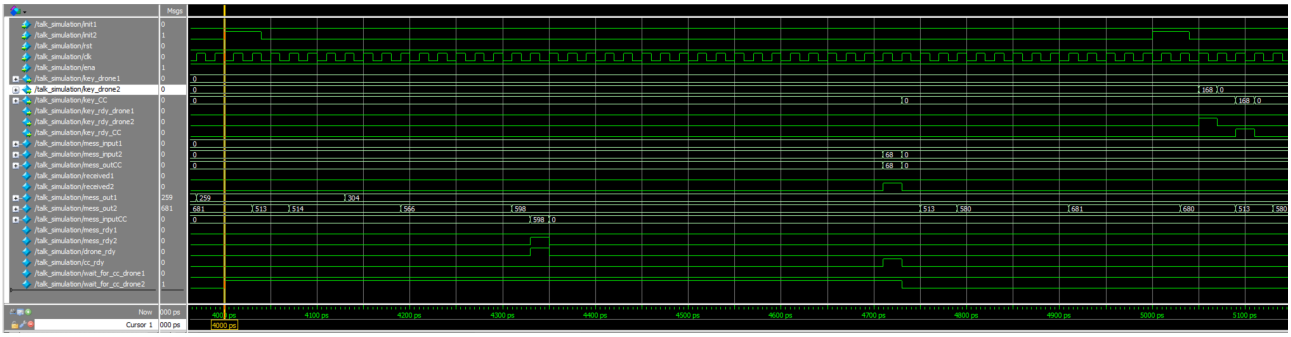
Projekt przetestowano z wykorzystaniem narzędzia ModelSim. Przygotowano wektory testowe (50). Założeniem testu było, iż po wymuszeniu przez wektor testowy sygnału init1 lub init2 uzgodni się wspólny klucz dla CC i odpowiednio drona 1 lub drona 2. Wszystkie wykonane testy przebiegły pomyślnie. Na podstawie testów oceniono, że uzgodnienie klucza po obu stronach zajmuje ok. 1 ns.

```
restart -nowave -force
add wave -radix unsigned *
force clk 0 0, 1 10 -r 20
force rst 1 0, 0 1
force ena 1 0

force init1 1 40, 0 80
force init2 1 1000, 0 1040
force init1 1 2800, 0 2840
force init2 1 4000, 0 4040
force init2 1 5000, 0 5040
force init1 1 6000, 0 6040
force init1 1 8000, 0 8040
force init1 1 10000, 0 10040
force init2 1 12000, 0 12040
force init2 1 14000, 0 14040
force init1 1 16000, 0 16040
run 20000
```

Rysunek 50. Przykładowy skrypt narzędzia ModelSim do testowania projektu.
(plik simulation.do)



Rysunek 51. Widok programu ModelSim z przykładowego uzgadniania klucza z dronem 1.



Rysunek 52. Widok programu ModelSim z przykładowego uzgadniania klucza z dronem 2.

6.4. Podsumowanie etapu dostarczania

W ramach etapu dostarczania udało się sfinalizować projekt. Dokonane modyfikacje (umożliwiające dodawanie do przygotowanego rozwiązania kolejnych dronów), a także implementacja modułu symulacyjnego, nadały projektowi potencjalnie praktycznego zastosowania. Rozbudowane względem etapu definiowania testy, które przeprowadzono dla dwóch dronów, dowiodły ostatecznej poprawności działania wszystkich modułów. Ich przetestowanie dla mnogiej liczby dronów (tj. więcej niż jednego) było kluczowe dla postawienia takiego wniosku. Za satysfakcjonujące uznano również parametry osiągane przez poszczególne moduły. Ostatecznie stwierdzono, że przygotowana implementacja działa prawidłowo i zgodnie z oczekiwaniami, a prace nad projektem można uznać za zakończone.

7. Bibliografia

[1] GeeksForGeeks user - SamanvayaPanda. *ElGamal Encryption Algorithm.* URL: https://www.geeksforgeeks.org/elgamal-encryption-algorithm/ (term. wiz. 04. 04. 2023).

[2] Shivam Agrawal. *Modular Exponentiation (Power in Modular Arithmetic).* URL: https://www.geeksforgeeks.org/modular-exponentiation-power-in-modular-arithmetic/ (term. wiz. 23. 04. 2023).

[3] Crypto-IT. *RSA szyfrowanie lub uwierzytelnianie wiadomości.* URL: http://www.crypto-it.net/pl/asymetryczne/rsa.html (term. wiz. 02. 04. 2023).

[4] Crypto-IT. *Salsa20 szyfr strumieniowy z kluczem symetrycznym.* URL: http://www.crypto-it.net/pl/symetryczne/salsa20.html (term. wiz. 02. 04. 2023).

[5] Crypto-IT. *Schemat blokowy algorytmu Salsa20.* URL: http://www.crypto-it.net/Images/salsa20/algorithm_scheme_pl.png (term. wiz. 02. 04. 2023).

[6] Crypto-IT. *Strumieniowe Szyfry Symetryczne.* URL: http://www.crypto-it.net/pl/symetryczne/szyfry-strumieniowe.html (term. wiz. 02. 04. 2023).

[7] Chiradeep Gupta i N V Subba Reddy. „Enhancement of Security of Diffie-Hellman Key Exchange Protocol using RSA Cryptography". W: *Journal of Physics: Conference Series* (2022). URL: https://doi.org/10.1088/1742-6596/2161/1/012014.

[8] via Wikimedia Commons Matt Crypto at English Wikipedia Public domain. *TEA InfoBox Diagram.* URL: https://commons.wikimedia.org/wiki/File:TEA_InfoBox_Diagram.png (term. wiz. 04. 04. 2023).

[9] Evgeny Milanov. „The RSA algorithm". W: *RSA laboratories* (2009), s. 1–11.

[10] Souvik Nandi. *Implementation of Diffie-Hellman Algorithm.* URL: https://www.geeksforgeeks.org/implementation-diffie-hellman-algorithm/ (term. wiz. 23. 04. 2023).

[11] Sebastian Pauli. *ElGamal Encryption System.* URL: https://mathstats.uncg.edu/sites/pauli/112/HTML/secelgamal.html (term. wiz. 04. 04. 2023).

[12] Sebastian Pauli. *ElGamal Encryption System.* URL: https://mathstats.uncg.edu/sites/pauli/112/HTML/generated/latex-image/imelgmal.svg (term. wiz. 04. 04. 2023).

[13] Eric Rescorla. *Diffie-Hellman Key Agreement Method.* IETF. 1999. URL: https://datatracker.ietf.org/doc/html/rfc2631#section-2.1.

[14] Marcin Rogawski PROKOM Software S.A. *Szyfry strumieniowe w strukturach FPGA.* URL: http://mason.gmu.edu/~mrogawsk/arch/enigma2005.pdf.

[15] Shihab Shawkat. „Enhancing Steganography Techniques in Digital Images". Prac. dokt. List. 2016. DOI: 10.13140/RG.2.2.16678.57925.

[16] Sean Sullivan. *Double Diamond Design Process.* URL: https://productstride.substack.com/p/double-diamond-design-process/ (term. wiz. 12. 03. 2023).

[17] Service Design Vancouver. *Creative Briefs and Design Processes.* URL: http://servicedesignvancouver.ca/briefs-and-design-processes/ (term. wiz. 14. 03. 2023).

[18] A.J. Vinick. *DH key exchange.*

8. Załączniki

— **ReferenceModel.zip** - projekt w języku Java zawierający referencyjny model uzgadniania klucza z wykorzystaniem protokołu Diffiego-Hellmanana oraz algorytmu El-Gamala.

— **dh.rar** - implementacja modułów drona, CC, szybkiego potęgowania modularnego, generatora liczb losowych opartego na LFSR w języku Verilog oraz wektory testowe.

Wszystkie załączniki umieszczone są w plikach kanału zespołu w aplikacji MS Teams