Università degli Studi dell'Insubria

Dipartimento di Scienze Teoriche Applicate

Corso di Studio Triennale in Informatica

Definizione ed implementazione di protocolli di sicurezza all'interno di un simulatore di reti 5G

Relatore: Dr. Alessandra Rizzardi

Correlatore: Prof.ssa Sabrina Sicari

Tesi di Laurea di:

Ferri Matteo

Matr. N. 744234

Anno Accademico 2022/2023

Indice

Capitolo 1

Introduzione **3**

Capitolo 2

Stato dell'arte **5**

2.1 Il 5G 5

2.2 Caratteristiche principali del 5G 6

2.3 La crittografia 8

2.4 Chiavi simmetriche e asimmetriche 10

2.5 Protocolli di scambio della chiave 13

2.5.1 Diffie-Hellman (DH) 13

Capitolo 3

5G-air-simulator e la sicurezza **15**

3.1 5G-air-simulator 15

3.2 Componenti principali di 5G-air-simulator 16

3.3 Vulnerabilità di 5G-air-simulator 18

3.4 Sistema crittografico in 5G-air-simulator 20

3.5 Algoritmo di scambio delle chiavi in 5G-air-simulator 21

Capitolo 4

Gli algoritmi di crittografia utilizzati nel contesto del 5G **23**

4.1 Sicurezza della rete 5G 23

4.2 Standard di sicurezza 5G definiti dal 3GPP 24

4.3 Sistema crittografico nelle reti 5G 25

4.4 Complessità computazionale degli algoritmi di cifratura simmetrica 26
4.5 AES 26
4.5.1 Componenti di AES 27
4.5.2 Esecuzione di AES 29
4.5.3 AES e attacchi crittografici 31
4.5.4 Complessità computazionale di AES 32
4.6 ZUC 33
4.6.1 Componenti di ZUC 33
4.6.2 L'esecuzione di ZUC 36
4.6.3 ZUC e attacchi crittografici 37
4.6.4 Complessità computazionale di ZUC 38

Capitolo 5

Implementazione e risultati ottenuti 39

5.1 Implementazione di AES 39
5.1.1 Cifratura a blocchi 40
5.1.2 Il costruttore 44
5.1.3 "AES_SRC.cpp" e "AES_DST.cpp" 44
5.2 Implementazione di ZUC 45
5.2.1 "ZUC_encrypt.cpp" e "ZUC_decrypt.cpp" 48
5.3 Implementazione di Diffie-Hellman 49
5.3.1 "DH_SRC.cpp" e "DH_DST.cpp" 51
5.4 Tempi di Simulazione e Analisi delle Prestazioni 54

Capitolo 6

Conclusioni 58

Bibliografia 60

Capitolo 1

Introduzione

Oggi, le telecomunicazioni costituiscono una componente sempre più importante dell'infrastruttura della nostra società, ed è innegabile che stiano attraversando una evidente rivoluzione. Questo processo di evoluzione ha avuto inizio con la prima generazione di tecnologia cellulare wireless, nota come 1G, e ha proseguito senza sosta fino a giungere alla quinta generazione, il 5G.

Tuttavia, in contrasto con quanto avvenuto con i precedenti sistemi di comunicazione, il 5G non si limita a rinnovare l'infrastruttura di rete esistente e l'ecosistema delle telecomunicazioni. Al contrario, ha un impatto ancora più profondo in settori come la logistica, l'automotive, il turismo e il settore medico [1].

L'introduzione e la diffusione del 5G hanno portato notevoli vantaggi in termini di velocità, latenza ridotta e capacità di connessione. Tuttavia, questa crescente interconnessione e la proliferazione di dispositivi connessi alla rete hanno creato nuove opportunità per gli attacchi informatici [2].

Per far fronte a queste crescenti minacce, è fondamentale che le organizzazioni rafforzino le loro strategie di sicurezza informatica, adottando misure come la crittografia avanzata, la segmentazione di rete, l'analisi comportamentale e la formazione continua del personale per riconoscere e rispondere alle minacce informatiche.

L'importanza del 5G e dei sistemi di crittografia nell'attuale panorama delle telecomunicazioni e della sicurezza informatica è fondamentale, poiché entrambi svolgono un ruolo cruciale nello sviluppo e nella protezione delle reti di comunicazione moderne.

Questo lavoro di tesi intende esplorare in dettaglio le potenzialità del 5G e concentrarsi sulla definizione ed implementazione dei protocolli di sicurezza all'interno di un simulatore di reti 5G. Dettagliatamente, sono stati sviluppati dei sistemi di crittografia avanzata come AES (Standard di

Crittografia Avanzata) e ZUC, insieme al protocollo di scambio affidabile delle chiavi noto come Diffie-Hellman.

Inoltre, vengono presentati test che simulano situazioni di comunicazione sicura e vengono analizzati gli effetti della sicurezza sulla performance complessiva della rete. Per acquisire i dati, è stato utilizzato *5G-air-simulator*, ossia uno strumento open source e basato sugli eventi che modella gli elementi chiave dell'interfaccia aerea 5G da una prospettiva a livello di sistema

In particolare, nel secondo capitolo verranno approfonditi i concetti di base del 5G, inclusi aspetti come le nuove tecnologie, le frequenze, le velocità di trasmissione e le applicazioni. Inoltre, vengono introdotti i concetti fondamentali della crittografia, spiegando il ruolo dell'encryption e dei protocolli di scambio chiave come Diffie-Hellman.

Nel terzo capitolo verranno invece descritti 5G-air-simulator, i suoi componenti principali e i problemi da affrontare con la crittografia, le vulnerabilità e le strategie adottate per affrontare i problemi di sicurezza.

Il capitolo quattro, che conclude il quadro teorico dell'elaborato saranno elencati e descritti gli algoritmi di crittografia utilizzati nel contesto 5G, incluso AES e ZUC. Per ognuno di essi verranno spiegate le loro caratteristiche principali, vantaggi e svantaggi.

Proseguendo con il capitolo cinque, verrà presentato ed analizzato il lavoro di tesi. Al suo interno sono presentati le implementazioni dei vari sistemi crittografici sviluppati e le scelte progettistiche effettuate.

Nel capitolo sei, si giunge alla fase conclusiva, in cui saranno presentate le riflessioni e considerazioni finali emerse dallo studio condotto in questo progetto.

Capitolo 2

Stato dell'arte

In questo capitolo vengono illustrati gli aspetti chiave del 5G, approfondendo le sue caratteristiche principali, le nuove applicazioni e i casi d'uso nel quale viene utilizzato. Allo stesso tempo, vengono introdotti i concetti fondamentali della crittografia, spiegando il ruolo cruciale dell'encryption e dei protocolli di scambio delle chiavi, come il celebre Diffie-Hellman.

2.1 Il 5G

Il 5G è la quinta generazione di tecnologia mobile ed è stata progettata dal consorzio 3GPP (3rd Generation Partnership Project) per portare importanti miglioramenti rispetto alle generazioni precedenti (2G, 3G e 4G). Questa nuova tecnologia offre una serie di vantaggi chiave, tra cui velocità di trasmissione più elevate, latenza ridotta e una capacità di connessione molto maggiore.

È corretto notare che il 5G è stato distribuito a partire dal 2019, ma la sua adozione ed implementazione sono state graduali e variano notevolmente in tutto il mondo. L'implementazione del 5G richiede investimenti significativi nelle infrastrutture di rete, inclusa la sostituzione delle stazioni di base e l'installazione di attrezzature adatte alle nuove frequenze, come le onde millimetriche (mmWave).

In effetti, la disponibilità del 5G varia notevolmente tra le diverse regioni e paesi. Mentre alcune aree geografiche, come gli Stati Uniti e alcune parti dell'Unione Europea, hanno effettivamente implementato reti 5G a livello commerciale in alcune delle loro città, molte altre regioni del mondo stanno ancora lavorando all'espansione e all'implementazione di questa nuova tecnologia.

Il fatto che, nel quarto trimestre del 2021, il 5G fosse stato implementato commercialmente in 25 dei 27 stati membri dell'Unione Europea è un segno della rapida evoluzione di questa tecnologia. Tuttavia, è importante notare che l'adozione del 5G può variare anche all'interno dei paesi e tra le diverse zone urbane e rurali. La situazione dell'implementazione del 5G continua a cambiare e si prevede che si espanderà ulteriormente nei prossimi anni man mano che le infrastrutture vengono migliorate e ottimizzate [1].

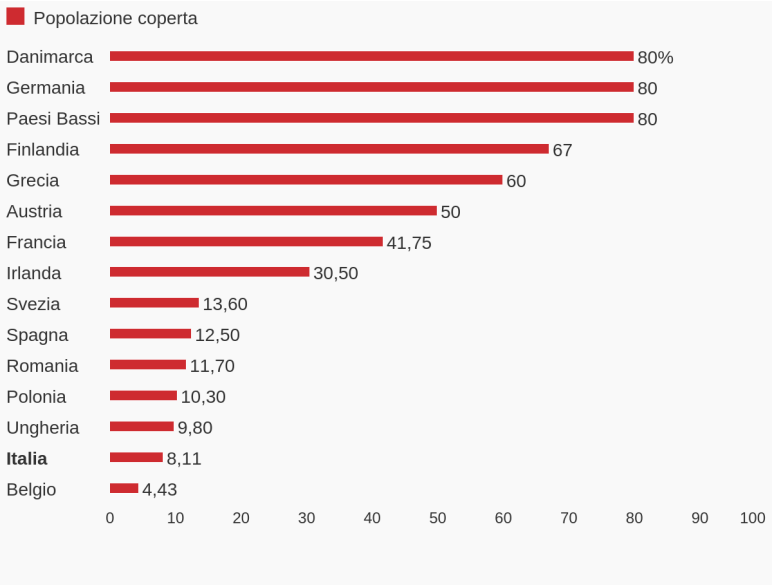

Figura 1. Copertura della rete 5G in Europa nel dicembre del 2021

2.2 Caratteristiche principali del 5G

Il 5G si basa su nuove tecnologie e standard di comunicazione, tra cui l'uso di onde millimetriche (mmWave), tecnologie MIMO (Multiple-Input, Multiple-Output) avanzate e virtualizzazione delle reti. Queste innovazioni consentono di ottenere velocità e prestazioni superiori rispetto alle generazioni precedenti [3].

Il 5G opera, inoltre, su una gamma più ampia di frequenze rispetto al 4G. Queste frequenze possono essere suddivise in tre bande principali:

- Banda bassa: Frequenze inferiori a 1 GHz, simili a quelle usate dal 4G. Queste bande offrono una copertura ampia ma velocità simili o leggermente migliori rispetto al 4G.

- Banda media: Frequenze tra 1 GHz e 6 GHz, comprese le frequenze già utilizzate per le reti cellulari. Questa banda offre un buon equilibrio tra copertura e velocità.
- Banda alta (mmWave): Frequenze superiori a 24 GHz. Questa banda offre velocità molto elevate ma ha una copertura molto limitata e può essere ostacolata da ostacoli fisici come edifici.

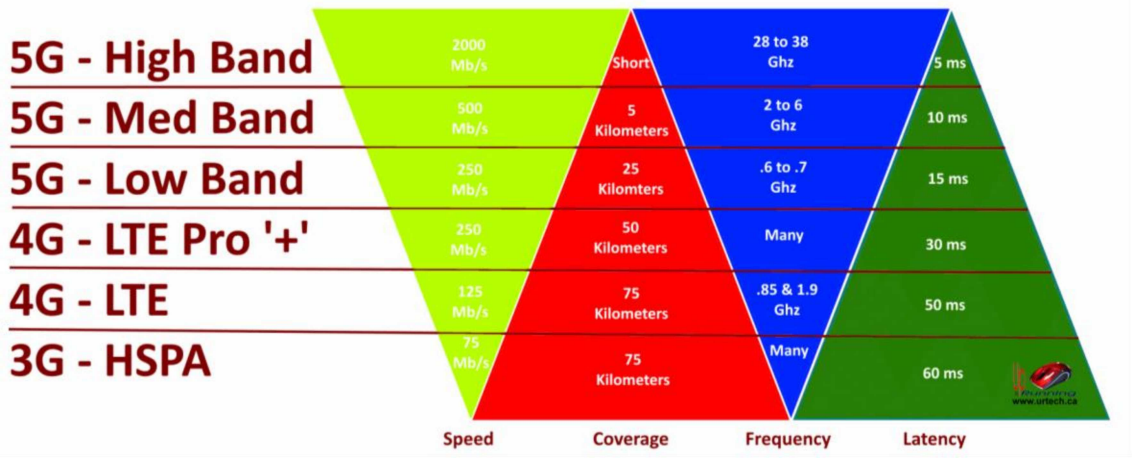

Figura 2. Confronto tra la velocità, copertura, frequenza e latenza delle varie tecnologie

Il 5G offre velocità di trasmissione molto più elevate rispetto al 4G. Le velocità possono variare notevolmente in base alla banda di frequenza e alla densità delle celle, ma in generale, si possono raggiungere velocità teoriche di gigabit al secondo (Gbps). Questo significa che il download e l'upload di file avvengono molto più velocemente, il che è cruciale per applicazioni ad alta intensità di dati come lo streaming video in 4K e 8K, la realtà virtuale (VR) e la realtà aumentata (AR).

Questa tecnologia mira a ridurre notevolmente la latenza, cioè il ritardo nella trasmissione dei dati tra il dispositivo e la rete. La latenza del 5G è prevista intorno a 1 millisecondo (ms), rispetto ai circa 30 ms del 4G. Questa bassa latenza è fondamentale per applicazioni sensibili alla latenza, come i veicoli autonomi e il controllo remoto di dispositivi [3][4].

Il 5G abilita una vasta gamma di nuove applicazioni e casi d'uso [5], tra cui:

- Internet delle cose (IoT): Il 5G supporta un numero molto maggiore di dispositivi IoT connessi, consentendo l'automazione domestica intelligente, le città intelligenti e l'industria 4.0.
- Telemedicina: La bassa latenza e l'alta affidabilità del 5G consentono consulenze mediche remote in tempo reale e chirurgia assistita da robot.
- Gaming in cloud: Il 5G rende possibile lo streaming di videogiochi ad alta risoluzione in tempo reale su dispositivi mobili.

- Veicoli autonomi: Il 5G è fondamentale per la comunicazione tra veicoli e infrastrutture stradali, migliorando la sicurezza e la gestione del traffico.

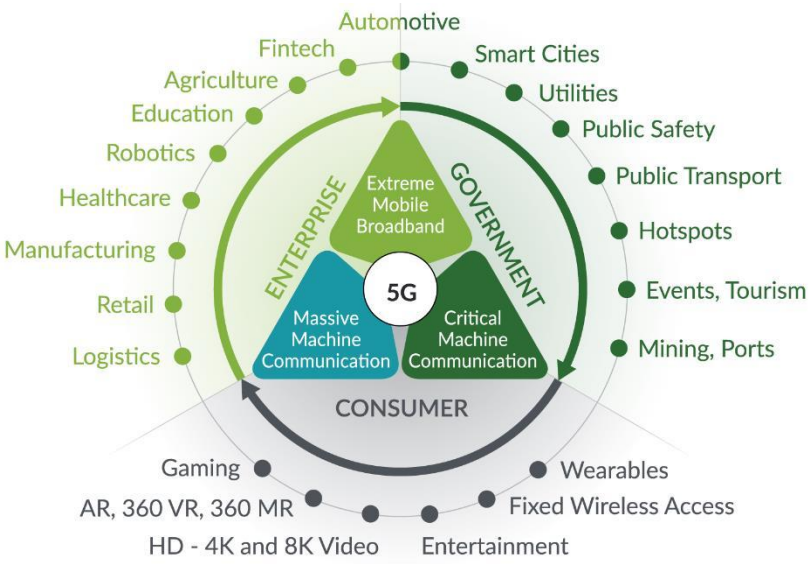

Figura 3. L'era del 5G consente nuove applicazioni

In generale, il 5G sta rivoluzionando il modo in cui le persone si connettono e utilizzano la tecnologia, aprendo la strada a nuove opportunità di innovazione in diversi settori. Tuttavia, è importante notare che la disponibilità del 5G può variare notevolmente in base alla regione e alla densità della copertura della rete.

2.3 La crittografia

Con la proliferazione di dispositivi collegati in rete, si è verificato un incremento delle incursioni informatiche e delle minacce online. Di conseguenza, negli ultimi anni, si è attribuita una crescente rilevanza alla protezione delle informazioni e alla salvaguardia dei dati trasmessi attraverso le reti digitali.

In questi ultimi anni, si sono ampiamente diffusi *i sistemi distribuiti*, l'esempio possono essere le cryptovalute che usano sistemi peer-to-peer e quindi richiedono una sicurezza interna, ovvero a livello di rete distribuita.

Gli attacchi informatici possono essere di due tipologie:

- Attacchi passivi: dove né il mittente né il destinatario si rendono conto di un terzo che li osserva la transizione nella rete o legge il messaggio.
- Attacchi attivi: implicano la modifica del flusso dei dati, creandone un altro falso.

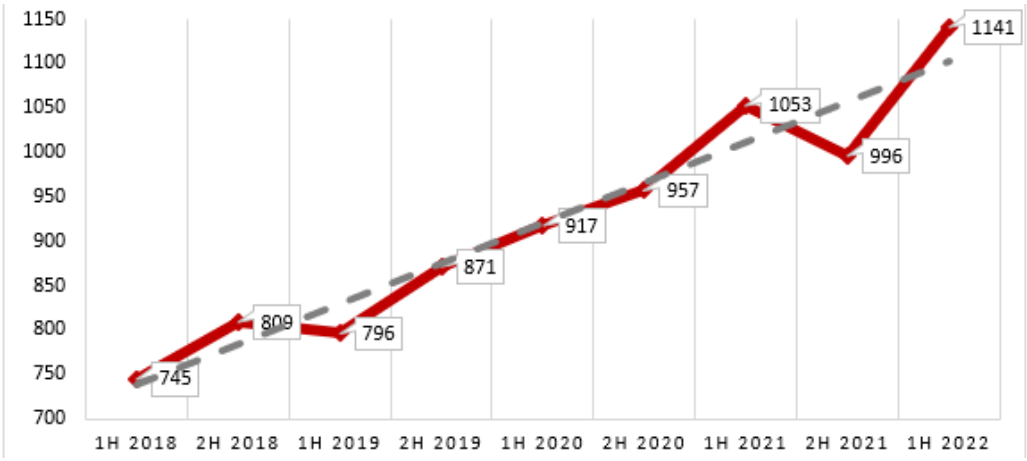

Figura 4. Attacchi per semestre (2018-2022)

I sistemi informatici, quindi, hanno bisogno di essere sicuri tramite strumenti come l'autenticazione, la riservatezza dei dati e il controllo all'accesso.

La crittografia è la scienza che si occupa di proteggere le informazioni rendendole incomprensibili o difficili da interpretare per coloro che non hanno le chiavi necessarie per decifrarle. Questo processo è fondamentale per garantire la privacy e la sicurezza delle comunicazioni, dei dati sensibili e delle transazioni su Internet [6].

In particolare, la **cifratura** (**Encryption**) è il processo di conversione di dati leggibili (testo normale o dati non criptati) in una forma incomprensibile chiamata testo cifrato, utilizzando un algoritmo matematico e una chiave segreta. La chiave è un parametro cruciale nella cifratura, poiché determina come verranno criptati e successivamente decifrati i dati. I dati cifrati sono sicuri a meno che qualcuno non abbia la chiave di decrittazione corretta.

Viceversa, la **decifratura** (**Decryption**) è il processo opposto alla cifratura. Coinvolge l'uso della chiave corretta per convertire il testo cifrato nuovamente in testo normale o dati leggibili. Solo chi possiede la chiave di decrittazione corretta può effettuare con successo questo processo.

2.4 Chiavi simmetriche e asimmetriche

Esistono due tipi principali di algoritmi di cifratura: chiavi simmetriche e chiavi asimmetriche [7].

Le **chiavi simmetriche** (o chiavi private), nell'ambito della crittografia, sono un tipo di chiave utilizzato per cifrare e decifrare i dati durante la comunicazione sicura. Queste chiavi seguono un approccio "segreto condiviso", il che significa che mittente e destinatario condividono la stessa chiave segreta. Il mittente utilizza questa chiave per cifrare i dati prima della trasmissione, mentre il destinatario la utilizza per decifrare i dati ricevuti. Questo processo di cifratura e decifratura richiede che entrambe le parti conoscano e mantengano segreta la chiave.

Figura 5. Funzionamento delle chiavi simmetriche

Le chiavi simmetriche sono generalmente più veloci ed efficienti delle chiavi asimmetriche (o chiavi pubbliche), poiché la complessità matematica coinvolta nella cifratura e nella decifratura è minore. Questo le rende adatte per la cifratura e la decifratura di grandi quantità di dati in tempo reale.

Inoltre, vengono spesso utilizzate nelle reti virtuali private (VPN) per garantire la privacy delle comunicazioni su Internet e nei protocolli di sicurezza delle connessioni Wi-Fi.

Il principale problema associato alle chiavi simmetriche è la necessità di condividere la chiave in modo sicuro tra le parti che comunicano. Se una chiave simmetrica cade nelle mani sbagliate, gli attaccanti possono accedere ai dati cifrati. Per mitigare questo rischio, vengono utilizzati protocolli di scambio chiave sicuri, come il protocollo Diffie-Hellman, per stabilire una chiave segreta condivisa in modo affidabile.

In sintesi, le chiavi simmetriche rappresentano un metodo efficiente ed efficace per la cifratura dei dati, ma richiedono una gestione attenta delle chiavi per garantire la sicurezza delle comunicazioni.

Le **chiavi asimmetriche**, conosciute anche come chiavi pubbliche e private, rappresentano un altro importante concetto nella crittografia e sono utilizzate per risolvere uno dei principali problemi associati alle chiavi simmetriche: la condivisione sicura delle chiavi.

La crittografia asimmetrica opera con una coppia di chiavi correlate, composta da una chiave pubblica e una chiave privata. Quest'ultime sono generate simultaneamente e sono strettamente legate tra loro, ma hanno funzioni diverse.

La chiave pubblica è accessibile a chiunque e può essere condivisa liberamente. È utilizzata per cifrare i dati o verificare le firme digitali. Tuttavia, non può essere utilizzata per decifrare i dati criptati.

La chiave privata, invece, è segreta e deve essere mantenuta in modo sicuro. È utilizzata per decifrare i dati cifrati con la chiave pubblica corrispondente o per firmare digitalmente i documenti. Solo il proprietario della chiave privata ha accesso a questa chiave.

Quando una persona A vuole inviare un messaggio o dati in modo sicuro a una persona B, utilizza la chiave pubblica di B per cifrare i dati. La chiave pubblica di B è nota e può essere condivisa apertamente. Quindi, chiunque può cifrare dati utilizzando la chiave pubblica di B, ma solo B può decifrarli con la sua chiave privata.

Quando B riceve i dati cifrati, li decifra utilizzando la sua chiave privata. Solo B possiede questa chiave e, pertanto, solo B può decifrare i dati. Questo processo garantisce la confidenzialità dei dati, poiché solo il destinatario legittimo può accedere alle informazioni originali.

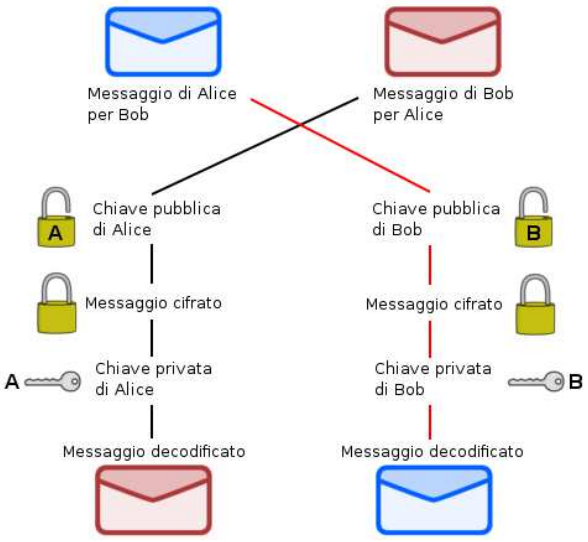

Figura 6. Funzionamento delle chiavi asimmetriche

La cifratura asimmetrica può essere utilizzata per garantire l'autenticazione tramite l'uso delle **firme digitali** per i documenti. Ad esempio, se A vuole inviare un messaggio autenticato a B, A può firmare digitalmente il messaggio utilizzando la sua chiave privata. Il destinatario B può quindi verificare la firma digitale utilizzando la chiave pubblica di A. Se la firma è valida, B può essere sicuro che il messaggio proviene realmente da A e che non è stato alterato durante il trasferimento.

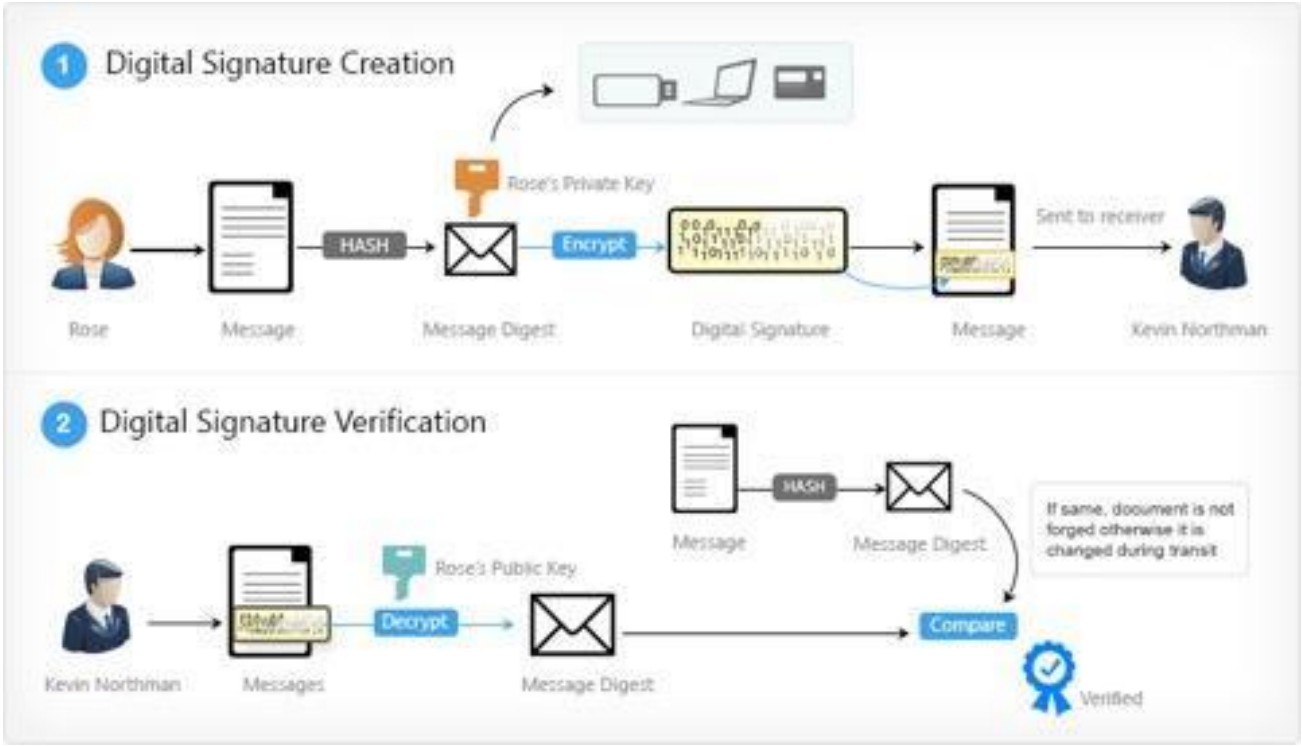

Figura 7. Funzionamento della firma digitale

La firma digitale garantisce anche l'integrità dei dati. Se anche un singolo bit del messaggio viene modificato, la firma digitale non sarà più valida, e la verifica di B rileverà l'alterazione. In questo modo, la cifratura asimmetrica aiuta a garantire che i dati non siano stati compromessi durante la trasmissione.

In sintesi, la cifratura asimmetrica permette di cifrare dati in modo che solo il destinatario legittimo possa decifrarli utilizzando la sua chiave privata. Inoltre, la firma digitale consente di autenticare l'origine dei dati e di verificare che i dati non siano stati alterati durante la trasmissione, contribuendo così a garantire l'autenticazione e l'integrità delle informazioni. Questa combinazione di crittografia asimmetrica e firma digitale è fondamentale per la sicurezza delle comunicazioni online e la protezione dei dati.

Il funzionamento delle chiavi asimmetriche si basa su operazioni matematiche complesse che coinvolgono numeri primi. La chiave pubblica è derivata da queste operazioni, ed è relativamente facile da calcolare, ma risalire alla chiave privata (fattorizzazione inversa) è estremamente difficile e richiede molto tempo, specialmente per chiavi molto lunghe.

Gli algoritmi di crittografia asimmetrica più noti includono RSA (Rivest–Shamir–Adleman) e ECC (Elliptic Curve Cryptography). Queste chiavi vengono utilizzate in scenari come la gestione delle chiavi SSL/TLS per la sicurezza delle connessioni web, la creazione di firme digitali e la gestione delle chiavi per l'accesso sicuro ai sistemi informatici.

2.5 Protocolli di scambio della chiave

La protezione delle chiavi di cifratura svolge un ruolo fondamentale nella crittografia. I protocolli di scambio chiave sono metodi e procedure utilizzati per consentire a due o più parti di stabilire una chiave di crittografia segreta condivisa in modo sicuro su una rete pubblica o non sicura. Questi protocolli sono fondamentali perché risolvono il problema critico della condivisione sicura delle chiavi, che è essenziale per garantire la confidenzialità e la sicurezza delle comunicazioni crittografate [8]. Di seguito viene illustrato uno dei protocolli di scambio chiave più utilizzato, Diffie-Hellman.

2.5.1 Diffie-Hellman (DH)

Il protocollo Diffie-Hellman è uno dei primi protocolli di scambio chiave ed è stato inventato nel 1976 da Whitfield Diffie e Martin Hellman. Fu il primo metodo pratico per due interlocutori di accordarsi su un segreto condiviso (la chiave) utilizzando un canale di comunicazione non protetto. Il suo obiettivo è consentire a due parti di stabilire una chiave segreta condivisa su una rete non sicura senza mai scambiare direttamente la chiave. La sua sicurezza si basa sulla complessità computazionale del calcolo del logaritmo discreto.

Diffie-Hellman agisce seguendo un funzionamento ben preciso:

1) Le due entità, Alice e Bob, si accordano su due numeri pubblici g e p, dove g è un generatore del gruppo moltiplicativo degli interi modulo p e p è un numero primo grande.
2) Alice sceglie un numero casuale a e calcola A = gˆa mod p. Alice invia A a Bob.
3) Bob sceglie un numero casuale b e calcola B = gˆb mod p. Bob invia B ad Alice.
4) Alice calcola la chiave condivisa K = Bˆa mod p.
5) Bob calcola la chiave condivisa K = Aˆb mod p.

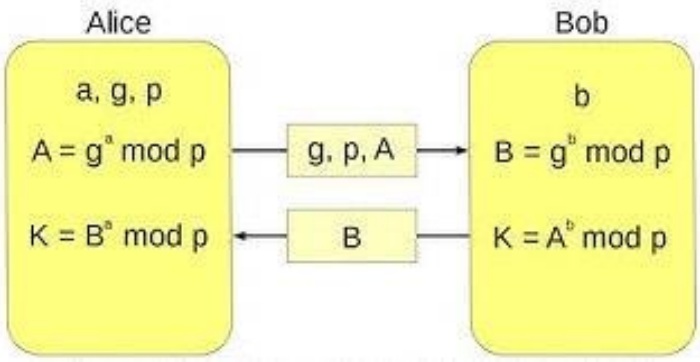

Figura 8. Procedura del protocollo Diffie-Hellman

Si può dimostrare che la chiave condivisa K è uguale per entrambe le entità e che un eventuale intruso che intercetta i messaggi A e B non può calcolare K senza conoscere a o b. Per aumentare la sicurezza dello scambio di chiavi DH, si possono usare varianti del protocollo che aggiungono una fase di autenticazione basata su password o su certificati digitali.

Il protocollo Diffie-Hellman è considerato molto sicuro a meno che non vengano utilizzate chiavi deboli o colpite da attacchi basati sulla crittoanalisi delle chiavi pubbliche.

I protocolli di scambio chiave sono essenziali per garantire che due o più parti possano comunicare in modo sicuro su reti pubbliche o non sicure. La chiave segreta condivisa generata attraverso questi protocolli può quindi essere utilizzata per cifrare e decifrare dati, proteggendo così la confidenzialità e la sicurezza delle comunicazioni.

Capitolo 3

5G-air-simulator e la sicurezza

In questo capitolo viene presentato 5G-air-simulator, ossia il simulatore utilizzato per svolgere gli esperimenti svolti in questo lavoro di tesi e dunque valutarne le prestazioni.
Nel dettaglio, vengono approfonditi i suoi componenti principali, i problemi da affrontare con la sicurezza, le vulnerabilità e le strategie adottate per affrontare i problemi di cybersecurity.

3.1 5G-air-simulator

5G-air-simulator è uno strumento open source e basato sugli eventi che modella gli elementi chiave dell'interfaccia aerea 5G da una prospettiva a livello di sistema.

Implementa diverse architetture di rete con più celle e utenti, diversi modelli di mobilità e di applicazione, un modello di collegamento al sistema calibrato per livelli fisici e di collegamento dati e un'ampia gamma di funzionalità standardizzate sia per il piano di controllo che per quello utente, nonché una serie di componenti tecnici recentemente progettati per l'interfaccia aerea 5G (come un massiccio Multiple Input Multiple Output, schemi di trasmissione multicast e broadcast estesi, antenne predittive, procedura di accesso casuale migliorata e NB-IoT).

Lo strumento è già stato utilizzato in diverse attività di ricerca per progettare e valutare le prestazioni di casi d'uso di riferimento abilitati al 5G. 5G-air-simulator è scritto in C++ [9].

3.2 Componenti principali di 5G-air-simulator

Il simulatore 5G-air-simulator è strutturato come un'applicazione basata su eventi: la classe *Calendar* contiene una lista di eventi da eseguire, con ciascun elemento che include il tempo di esecuzione richiesto, il metodo da eseguire, l'oggetto su cui deve essere chiamato il metodo ed eventualmente alcuni parametri.

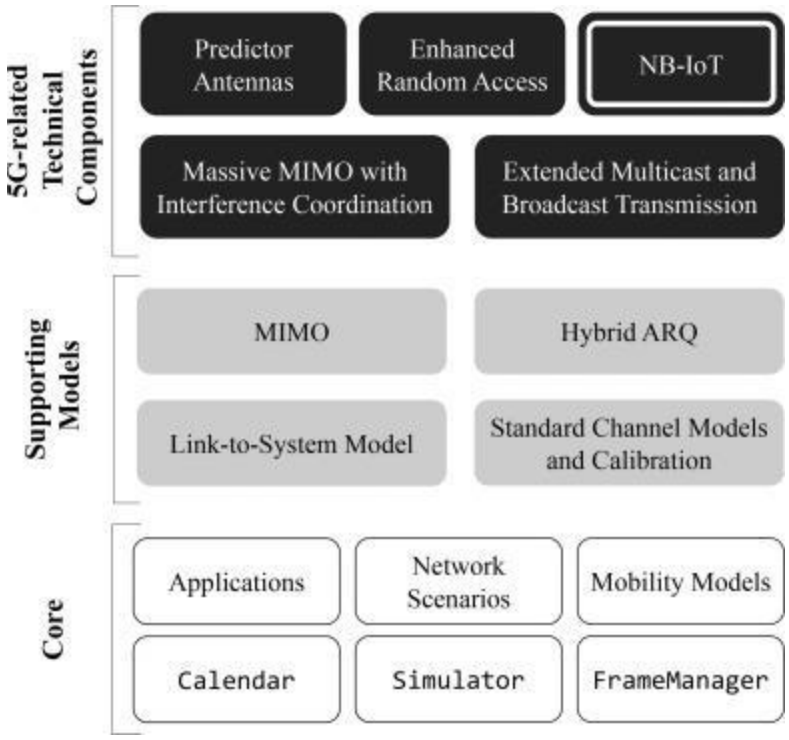

Figura 9: Building blocks del simulatore, NB-IoT è un componente indipendente

Altre classi importanti sono *Simulator* e *Framemanager*:

- *Simulator* è una classe singleton che esegue azioni globali, come avviare e interrompere la simulazione, aggiungere eventi al Calendario ed eseguirli.
- *FrameManager* traccia il flusso del tempo, incrementa i contatori legati ai frame e alle sotto-frame e differenzia i subframe dedicati a diverse funzioni, come downlink e uplink per la modalità Time Division Duplexing.

All'inizio del flusso del programma, viene selezionato uno degli scenari disponibili. Lo scenario è responsabile della creazione e dell'inizializzazione di molti elementi importanti dell'ambiente di simulazione, come le stazioni base, i terminali mobili e le realizzazioni dei canali.

In alcuni casi, l'inizializzazione di alcuni oggetti definisce alcuni eventi che vengono inseriti nella classe *Calendar* e verranno eseguiti in un secondo momento. Questo include, ad esempio, la generazione di pacchetti a livello di applicazione e il movimento dei dispositivi.

```
void
Simulator::Run (void)
{
  /*
   * This method start the whole simulation
   * The simulation will end when no events are into the
   * calendar list.
   */
  while (!m_calendar->IsEmpty () && !m_stop)
    {
      ProcessOneEvent ();
    }
}

void
Simulator::ProcessOneEvent(void)
{
  shared_ptr<Event> next = m_calendar->GetEvent();

  m_unscheduledEvents--;
  m_currentTs = next->GetTimeStamp()/1000.0;
  m_currentUid = next->GetUID();

  if (!next->IsDeleted())
    {
      next->RunEvent();
    }

  m_calendar->RemoveEvent();
}
```

Figura 10: Codice del metodo Run()

Dopo la configurazione iniziale, la simulazione effettiva viene avviata chiamando il metodo *Simulator*::*Run*(). A questo punto, la classe Calendario inizia a eseguire gli eventi registrati in ordine cronologico. Ciascun evento può comportare la generazione di nuovi eventi che vengono inseriti nel calendario, risultando in un continuo flusso di eventi da elaborare fino alla fine della simulazione. In particolare, alcuni tipi di eventi si ri-pianificano alla fine della loro esecuzione, ripetendosi periodicamente per tutto il tempo della simulazione.

La generazione ed esecuzione degli eventi continuano fino a quando non viene chiamato il metodo *Simulator*::*Stop*(), il quale fa sì che il calendario si fermi e scarti tutti gli eventi che potrebbero ancora essere in attesa, terminando infine il programma.

Di solito, l'orario di fine della simulazione viene impostato in anticipo tramite una chiamata a *Simulator*::*SetStop*() nello scenario, subito prima di chiamare *Simulator*::*Run*() [10][11].

```
void
Simulator::Stop (void)
{
  cout << " SIMULATOR_DEBUG: Stop (" << m_lastAssignedUid << " events)" << endl;
  m_stop = true;
}

void
Simulator::SetStop (double time)
{
  DoSchedule (time,
              MakeEvent (&Simulator::Stop, this));
}


shared_ptr<Event>
Simulator::DoSchedule (double time,
                       shared_ptr<Event> event)
{
  int timeStamp = round( (time + Now())*1000 );
  event->SetTimeStamp(timeStamp);
  event->SetUID(m_lastAssignedUid);

  m_lastAssignedUid++;
  m_unscheduledEvents++;

  m_calendar->InsertEvent(event);
  return event;
}
```

Figura 10: Codice dei metodi Stop() e SetStop()

3.3 Vulnerabilità di 5G-air-simulator

I rischi di sicurezza di 5G-air-simulator sono legati principalmente alle vulnerabilità del software, alla dipendenza da singoli fornitori, ai problemi di prestazioni e scalabilità e alle violazioni della licenza open-source. Per mitigare questi rischi, è necessario seguire le buone pratiche di sviluppo del software, verificare la validità e la coerenza dei risultati, ottimizzare i parametri di simulazione e rispettare le condizioni della licenza.

Le vulnerabilità di 5G-air-simulator sono legate principalmente alle seguenti categorie:

- **Vulnerabilità del software**: si tratta di errori o difetti nel codice sorgente o nella documentazione dell'applicazione che possono compromettere la sua funzionalità, affidabilità o sicurezza. Alcuni esempi di vulnerabilità del software sono i buffer overflow, gli injection attacks, i race conditions, i memory leaks, ecc. Per prevenire o correggere queste vulnerabilità, è necessario seguire le buone pratiche di sviluppo del software, testare e verificare il codice e applicare le patch di sicurezza quando disponibili.
- **Vulnerabilità della dipendenza**: si tratta di vulnerabilità che derivano dall'uso di librerie o componenti esterni all'applicazione che possono essere obsoleti, non aggiornati o non sicuri. Alcuni esempi di vulnerabilità della dipendenza sono i man-in-the-middle attacks, i denial-of-service attacks, i privilege escalation attacks, ecc. Per prevenire o correggere queste vulnerabilità, è necessario monitorare e aggiornare le dipendenze usate dall'applicazione e verificare la loro provenienza e integrità.
- **Vulnerabilità della prestazione**: si tratta di vulnerabilità che derivano dalla complessità computazionale o dalla memoria richiesta dall'applicazione che possono causare ritardi, blocchi o crash del sistema. Alcuni esempi di vulnerabilità della prestazione sono i resource exhaustion attacks, i memory corruption attacks, i timing attacks, ecc. Per prevenire o correggere queste vulnerabilità, è necessario ottimizzare i parametri di simulazione, usare algoritmi efficienti e gestire adeguatamente le eccezioni e gli errori.
- **Vulnerabilità della licenza**: si tratta di vulnerabilità che derivano dal mancato rispetto delle condizioni della licenza open-source dell'applicazione che possono comportare sanzioni legali o morali. Alcuni esempi di vulnerabilità della licenza sono la violazione del diritto d'autore, la violazione del copyleft, la violazione delle clausole di attribuzione o di non-commercialità, ecc [12].

Per prevenire o correggere queste vulnerabilità, è necessario rispettare le condizioni della licenza GNU GPL v3.0 quando si usa o si modifica il codice sorgente o gli eseguibili dell'applicazione [13].

3.4 Sistema crittografico in 5G-air-simulator

Non esiste un sistema crittografico specifico per il simulatore, in quanto la scelta dell'algoritmo di cifratura a chiave privata dipende da diversi fattori, come il livello di sicurezza richiesto, le prestazioni della macchina, la compatibilità con gli standard 5G e le preferenze personali. Tuttavia, si possono dare alcuni criteri generali per orientare la scelta:

- È consigliato utilizzare *algoritmi standardizzati* e riconosciuti come sicuri dalla comunità scientifica e industriale, come AES, 3DES, IDEA o CAST5.
- È preferibile utilizzare algoritmi che offrono una *lunghezza di chiave adeguata al livello di sicurezza* richiesto. In generale, maggiore è la lunghezza della chiave, maggiore è la resistenza agli attacchi di forza bruta. Tuttavia, una chiave troppo lunga può comportare una riduzione delle prestazioni. Un buon compromesso può essere una chiave di 128 bit, che offre un livello di sicurezza elevato senza sacrificare troppo le prestazioni.
- È necessario utilizzare algoritmi che offrono una *complessità computazionale bassa* o moderata per blocco. In generale, minore è la complessità computazionale, minore è il tempo necessario per cifrare o decifrare i dati. Tuttavia, una complessità troppo bassa può comportare una riduzione della sicurezza. Un buon compromesso può essere una complessità di qualche migliaio di operazioni elementari per blocco.
- È consigliato utilizzare *algoritmi che siano compatibili con gli standard 5G definiti dal 3GPP*. Questo garantisce una maggiore interoperabilità e conformità con le specifiche tecniche della rete 5G. Tra gli algoritmi a chiave privata standardizzati dal 3GPP per la rete 5G ci sono AES e ZUC.

Tenendo conto di questi criteri, un possibile algoritmo di cifratura a chiave privata da usare per 5G-air-simulator potrebbe essere AES con chiave di 128 bit. Questo algoritmo offre un elevato livello di sicurezza, una complessità computazionale moderata e una compatibilità con gli standard 5G. Inoltre, AES è uno degli algoritmi più diffusi e testati nel campo della crittografia simmetrica.

3.5 Algoritmo di scambio delle chiavi in 5G-air-simulator

Un algoritmo di scambio delle chiavi è un protocollo crittografico che consente a due entità di stabilire una chiave condivisa e segreta utilizzando un canale di comunicazione insicuro (pubblico) senza la necessità che le due parti si siano scambiate informazioni o si siano incontrate in precedenza. La chiave ottenuta mediante questo protocollo può essere successivamente impiegata per cifrare le comunicazioni successive tramite uno schema di **crittografia simmetrica**.

Per 5G-air-simulator, bisogna usare un algoritmo di scambio delle chiavi che sia compatibile con gli standard 5G definiti dal 3GPP, che offra un elevato livello di sicurezza e che sia efficiente dal punto di vista computazionale. Un possibile algoritmo di scambio delle chiavi da usare per 5G-air-simulator è lo **scambio di chiavi Diffie-Hellman (DH),** che è uno dei protocolli più diffusi e testati nel campo della crittografia asimmetrica [14].

L'utilizzo dell'algoritmo Diffie-Hellman come meccanismo di scambio chiave all'interno di 5G-air-simulator offre infatti diversi vantaggi:

- Sicurezza: Diffie-Hellman offre uno scambio di chiavi sicuro, che è fondamentale nelle reti 5G per proteggere la confidenzialità delle comunicazioni. Il protocollo Diffie-Hellman è resistente agli attacchi di tipo Man-in-the-Middle (MitM).
- Autenticazione: Diffie-Hellman può essere combinato con altri protocolli per garantire l'autenticazione reciproca tra le parti che stabiliscono una chiave condivisa. Questo aiuta a prevenire l'accesso non autorizzato alla rete 5G.
- Privacy: L'uso di Diffie-Hellman consente alle parti di stabilire una chiave di sessione senza dover condividere le chiavi effettive attraverso la rete. Ciò migliora la privacy delle comunicazioni 5G, rendendo più difficile per gli attaccanti intercettare o decifrare i dati.
- Scalabilità: Diffie-Hellman è adatto per scenari 5G con numerosi utenti e dispositivi. Poiché le chiavi di sessione vengono generate dinamicamente durante la negoziazione, l'algoritmo è altamente scalabile per reti di grandi dimensioni.
- Resistenza ai Futuri Avanzamenti Computazionali: Diffie-Hellman continua a essere robusto contro gli attacchi futuri basati su avanzamenti nella capacità di calcolo. Questo garantisce che le comunicazioni 5G rimangano sicure nel tempo.
- Conformità agli Standard: Diffie-Hellman è ampiamente accettato e utilizzato in tutto il mondo ed è conforme agli standard crittografici riconosciuti. Questa conformità facilita l'interoperabilità con altri dispositivi e sistemi.

- Complessità Minima nella Gestione delle Chiavi: Rispetto a molte altre tecniche di gestione delle chiavi, Diffie-Hellman richiede una complessità minima nella distribuzione e nella gestione delle chiavi.
- Adattabilità alle Esigenze di Sicurezza delle Reti 5G: Diffie-Hellman può essere configurato per soddisfare le specifiche esigenze di sicurezza delle reti 5G, inclusa la lunghezza delle chiavi e altre impostazioni di sicurezza.

In generale, Diffie-Hellman rappresenta una scelta affidabile e sicura per lo scambio di chiavi all'interno di 5G-air-simulator e offre una serie di vantaggi che lo rendono adatto alle esigenze di sicurezza e scalabilità delle moderne reti 5G.

Capitolo 4

Gli algoritmi di crittografia utilizzati nel contesto del 5G

In questo capitolo viene presentata una panoramica degli algoritmi crittografici impiegati abitualmente all'interno del contesto del 5G, tra cui figurano AES e ZUC. Per ciascuno di questi algoritmi, vengono presentate le loro caratteristiche principali, i benefici che comportano alla sicurezza e le loro eventuali limitazioni.

4.1 Sicurezza della rete 5G

La sicurezza all'interno di 5G-air-simulator dipende anche dalla sicurezza della rete 5G stessa, che presenta diverse sfide e opportunità rispetto alle generazioni precedenti. Alcuni dei rischi riguardanti la cybersecurity all'interno delle reti 5G sono:

- Una maggiore esposizione agli attacchi informatici e un aumento del numero dei potenziali punti di accesso per gli hacker, dato che le reti 5G si basano sempre più sul software e su una vasta gamma di applicazioni e servizi.
- La possibilità di inserire backdoor malevoli nei prodotti o nelle funzioni di rete da parte di fornitori non affidabili o compromessi.
- La vulnerabilità di alcuni elementi dell'apparecchiatura o determinate funzioni di rete che diventano più sensibili, come le stazioni base o le principali funzioni di gestione tecnica delle reti.
- Il rischio di interferenze nella privacy degli utenti o nella protezione dei dati personali, a causa della maggiore quantità e qualità dei dati trasmessi e raccolti dalle reti 5G.
- La complessità di garantire la sicurezza e la resilienza delle reti 5G in presenza di nuove tecnologie come il network slicing, il NFV e il SDN, che richiedono una maggiore flessibilità e dinamicità.

Per affrontare questi rischi, è necessario seguire gli standard di sicurezza 5G definiti dal 3GPP, che prevedono diverse misure per migliorare l'identificazione, l'autenticazione, la crittografia e la protezione dei dati nelle reti 5G.

Inoltre, è importante adottare un approccio coordinato a livello europeo per valutare e mitigare i rischi per la cybersicurezza delle reti 5G, come previsto dalla raccomandazione della Commissione europea del marzo 2019. Infine, è opportuno sensibilizzare gli utenti e i gestori delle reti 5G sui possibili rischi e sulle buone pratiche da seguire per garantire una maggiore sicurezza informatica [15].

4.2 Standard di sicurezza 5G definiti dal 3GPP

Gli standard di sicurezza 5G definiti dal 3GPP sono un insieme di specifiche tecniche che descrivono le misure di sicurezza per la rete 5G, sia a livello di accesso che di core e di servizio [16]. Queste specifiche coprono diversi aspetti, come la crittografia, l'autenticazione, la protezione dell'integrità, la privacy e la disponibilità della rete. Alcune delle principali specifiche di sicurezza 5G sono:

- **TS 33.501: Security architecture and procedures for 5G system**. Questa specifica definisce l'architettura di sicurezza generale per il sistema 5G, i requisiti di sicurezza, i meccanismi di sicurezza e le procedure per il controllo e il piano utente [16].
- **TS 33.502: Security Assurance Specification (SCAS) for the 5G network.** Questa specifica definisce i requisiti e i metodi per valutare e dimostrare il livello di sicurezza dei prodotti e delle funzioni di rete 5G.
- **TS 33.503: Security Assurance Specification (SCAS) for the user equipment (UE) and Universal Integrated Circuit Card (UICC)**. Questa specifica definisce i requisiti e i metodi per valutare e dimostrare il livello di sicurezza dell'equipaggiamento dell'utente (UE) e della scheda UICC che supportano il sistema 5G.
- **TS 33.504: Security Assurance Specification (SCAS) for the Interworking between the 5G system and external networks**. Questa specifica definisce i requisiti e i metodi per valutare e dimostrare il livello di sicurezza dell'interconnessione tra il sistema 5G e le reti esterne.

Queste e altre specifiche sono in continua evoluzione e aggiornamento per seguire lo sviluppo tecnologico e le esigenze del mercato del 5G. Il 3GPP è un'organizzazione internazionale che riunisce diversi organismi di standardizzazione, operatori, fornitori e altri stakeholder del settore delle telecomunicazioni per definire gli standard per le reti mobili.

4.3 Sistema crittografico nelle reti 5G

Esistono diversi algoritmi crittografici utilizzati per le reti 5G, dipendenti tutti dallo scenario di simulazione e dal livello di sicurezza richiesto. In generale, vengono utilizzati algoritmi standardizzati e riconosciuti come sicuri dalla comunità scientifica e industriale. Ad esempio, il 3GPP ha definito diversi algoritmi crittografici per la rete 5G, tra cui:

- **5G-AKA**: un algoritmo per l'autenticazione e la generazione delle chiavi tra l'UE e la rete 5G.
- **5G-EA1**: un algoritmo per la cifratura dei dati sul piano utente.
- **5G-EA2**: un algoritmo per la cifratura dei dati sul piano di controllo.
- **5G-EA3**: un algoritmo per la protezione dell'integrità dei dati sul piano utente e sul piano di controllo.

Questi algoritmi si basano su **primitive crittografiche** consolidate, come AES e ZUC.

Le primitive crittografiche sono le operazioni di base che vengono usate per costruire algoritmi e protocolli crittografici. Si tratta di funzioni matematiche che svolgono compiti come la generazione di chiavi, la cifratura, la decifratura, la firma digitale, la verifica di firma, l'hashing, ecc. Ad esempio:

- **AES**: Advanced Encryption Standard. È un algoritmo di cifratura a blocchi simmetrico che usa una chiave di 128, 192 o 256 bit per cifrare e decifrare i dati. È usato dagli algoritmi 5G-EA1 e 5G-EA2 per la cifratura dei dati sul piano utente e sul piano di controllo.
- **SNOW 3G**: Stream cipher with Non-linear Output Word. È un algoritmo di cifratura a flusso simmetrico che usa una chiave di 128 bit e un vettore di inizializzazione di 128 bit per generare una sequenza pseudo-casuale di bit. È usato dagli algoritmi 5G-EA1 e 5G-EA3 per la cifratura e la protezione dell'integrità dei dati sul piano utente e sul piano di controllo. Tuttavia, Snow 3G è un algoritmo di cifratura utilizzato principalmente nelle reti wireless 3G e 4G per garantire la sicurezza delle comunicazioni ed esistono algoritmi progettati appositamente per le reti 5G che offrono una maggiore sicurezza.
- **ZUC**: ZUC stream cipher. È un algoritmo di cifratura a flusso simmetrico che usa una chiave di 128 bit e un vettore di inizializzazione di 128 bit per generare una sequenza pseudo-casuale di bit. È usato dagli algoritmi 5G-EA1 e 5G-EA3 per la cifratura e la protezione dell'integrità dei dati sul piano utente e sul piano di controllo.
- **SHA-256**: Secure Hash Algorithm 256. È un algoritmo di hashing che produce un valore di hash di 256 bit a partire da un messaggio arbitrario. È usato dall'algoritmo 5G-AKA per l'autenticazione e la generazione delle chiavi tra l'UE e la rete 5G.

4.4. Complessità computazionale degli algoritmi di cifratura

Gli algoritmi a chiave privata sono algoritmi di cifratura simmetrica che usano la stessa chiave per cifrare e decifrare i dati. La complessità computazionale di questi algoritmi dipende dalla lunghezza della chiave, dal numero di operazioni elementari richieste per cifrare o decifrare un blocco di dati e dalla dimensione dei dati da cifrare o decifrare [17]. Alcuni esempi di algoritmi a chiave privata sono:

- **DES: Data Encryption Standard**. È un algoritmo di cifratura a blocchi che usa una chiave di 56 bit per cifrare e decifrare blocchi di 64 bit. La complessità computazionale di DES è di circa 2^56 operazioni elementari per blocco.
- **3DES: Triple Data Encryption Standard**. È un algoritmo di cifratura a blocchi che usa tre chiavi di 56 bit ciascuna per cifrare e decifrare blocchi di 64 bit. La complessità computazionale di 3DES è di circa 2^112 operazioni elementari per blocco.
- **IDEA: International Data Encryption Algorithm**. È un algoritmo di cifratura a blocchi che usa una chiave di 128 bit per cifrare e decifrare blocchi di 64 bit. La complessità computazionale di IDEA è di circa 2^128 operazioni elementari per blocco.

4.5 AES

AES, Advanced Encryption Standard, (noto anche come algoritmo Rijndael) è un algoritmo di cifratura a blocchi ampiamente utilizzato per proteggere dati sensibili tramite crittografia. È stato selezionato dal National Institute of Standards and Technology (NIST) degli Stati Uniti nel 2001 come standard di cifratura ufficiale, sostituendo il DES (Data Encryption Standard). Quando l'algoritmo DES fu formato e standardizzato, infatti, aveva senso per quella generazione di computer.

Chronology of DES Cracking	
Broken for the first time	1997
Broken in 56 hours	1998
Broken in 22 hours and 15 minutes	1999
Capable of broken in 5 minutes	2021

Figura 11: Cronologia di violazioni del DES

Al momento era necessario un algoritmo più robusto, con chiavi di dimensioni più lunghe e cifrature più potenti da violare. Venne quindi organizzato un concorso con l'obbiettivo di trovare un algoritmo più robusto ed efficiente.

L'algoritmo scelto è stato sviluppato da due crittografi belgi, Joan Daemen e Vincent Rijmen, che lo hanno presentato al processo di selezione per l'AES con il nome di "Rijndael", derivato dai nomi degli inventori.

AES è stato sviluppato per rispondere alla necessità di una crittografia più robusta e sicura. È basato su una struttura di rete di sostituzione permutazione iterata (SPN) e utilizza chiavi di diverse lunghezze (128, 192 o 256 bit) per la cifratura. AES è noto per la sua resistenza a una vasta gamma di attacchi crittografici, ed è diventato un pilastro della sicurezza informatica [19][21].

4.5.1 Componenti di AES

L'algoritmo AES è composto da diversi componenti chiave che contribuiscono alla sua struttura e al suo funzionamento. Ecco una panoramica dei principali componenti dell'AES:

- *S-Box* (*Substitution Box*): L'S-Box è una tabella di sostituzione non lineare utilizzata per sostituire i byte di dati durante il processo di cifratura. Questa tabella è fondamentale per l'introduzione di non linearità nell'algoritmo e rende più difficile la crittoanalisi.

		y															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
x	**0**	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Figura 12: Tabella S-Box

- *P-Box* (*Permutation Box*): La P-Box è utilizzata per riorganizzare i byte di dati all'interno di un blocco durante il processo di cifratura. Questa operazione di permutazione contribuisce a diffondere i dati e aumenta la sicurezza dell'algoritmo.
- *Chiave di cifratura*: L'AES utilizza chiavi di diverse lunghezze, tra cui 128, 192 o 256 bit. La chiave di cifratura è un elemento cruciale dell'algoritmo e viene utilizzata per mescolare e cifrare i dati.

	Key Length *(Nk words)*	Block Size *(Nb words)*	Number of Rounds *(Nr)*
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Figura 13: Lunghezza della chiave, dei blocchi e numero di round di AES

- *Round Keys*: AES genera round keys (chiavi di round) da una chiave di cifratura iniziale. Queste chiavi di round vengono utilizzate in ciascun round per mescolare e cifrare i dati.
- *Round Functions*: Gli "round functions" sono un insieme di operazioni matematiche che vengono applicate a ciascun round di cifratura. Queste operazioni includono la sostituzione dei byte, la permutazione dei byte, operazioni di shift e altre trasformazioni matematiche.
- *Rounds*: L'AES è una cifratura iterativa a blocchi che esegue un numero fisso di "rounds" (o cicli) a seconda della lunghezza della chiave. Per AES-128, vengono eseguiti 10 rounds, mentre AES-192 e AES-256 utilizzano rispettivamente 12 e 14 rounds.
- *AddRoundKey*: In ogni round, l'operazione AddRoundKey combina i dati del blocco con la chiave di round corrispondente. Questo processo aggiunge una componente chiave al blocco di dati e contribuisce all'efficacia della cifratura.
- *ShiftRows e MixColumns*: Queste operazioni vengono applicate in ciascun round per diffondere e mescolare i dati all'interno del blocco, aumentando così la sicurezza e la resistenza agli attacchi crittografici.
- *Inverse Operations*: Per la decifratura, l'AES utilizza operazioni inverse alle operazioni di cifratura per ripristinare i dati originali a partire dai dati cifrati.

Complessivamente, questi componenti lavorano insieme per eseguire la cifratura e la decifratura dei dati all'interno dell'AES, garantendo un alto livello di sicurezza e protezione delle informazioni.

4.5.2 Esecuzione di AES

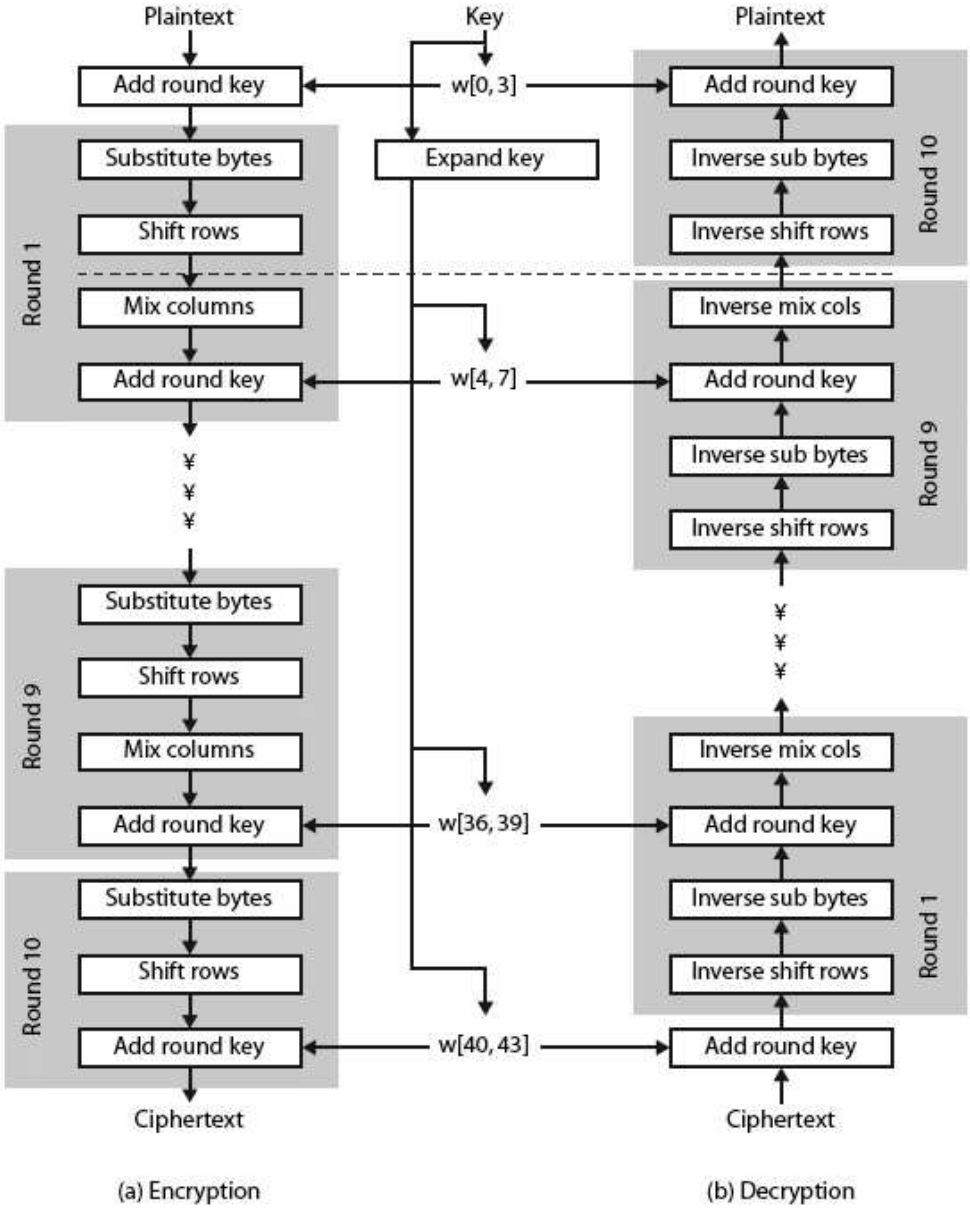

Figura 14: Encryption e Decryption di AES-128

L'esecuzione dell'algoritmo AES coinvolge una serie di passaggi chiave che vengono ripetuti un certo numero di volte, a seconda della lunghezza della chiave utilizzata (128, 192 o 256 bit).

Ecco una panoramica generale di come funziona l'esecuzione di AES:

1) *Preparazione delle chiavi di round*: Inizialmente, l'algoritmo AES prende in input la chiave di cifratura (128, 192 o 256 bit) e genera una serie di chiavi di round derivate da questa chiave principale. Queste chiavi di round vengono utilizzate in ciascun round del processo di cifratura.
2) *Inizializzazione*: Il blocco di dati da cifrare viene diviso in blocchi di dimensioni fisse (128 bit). Ora può iniziare il primo round.
3) *AddRoundKey*: In ciascun round, il blocco di dati corrente viene combinato con la chiave di round corrispondente utilizzando l'operazione di XOR (addizione modulo 2). Questo processo aggiunge una componente chiave al blocco di dati.
4) *SubBytes*: In questa fase, ogni byte del blocco di dati viene sostituito con un nuovo valore dalla S-Box (Substitution Box), una tabella di sostituzione non lineare. Questa operazione introduce non linearità nei dati.
5) *ShiftRows*: I byte all'interno di ciascuna riga del blocco di dati vengono spostati orizzontalmente in modo specifico. Questo passaggio aiuta a diffondere i dati all'interno del blocco.
6) *MixColumns* (escluso nell'ultimo round): Le colonne del blocco di dati vengono mescolate utilizzando una trasformazione matematica specifica. Questo processo contribuisce a ulteriormente mescolare i dati.
7) *AddRoundKey*: Si applica nuovamente l'operazione AddRoundKey, ma questa volta utilizzando una nuova chiave di round derivata dalla chiave principale.
8) *Ripetizione dei Rounds*: I passaggi dal 4 al 7 vengono ripetuti un certo numero di volte, a seconda della lunghezza della chiave. Per AES-128, vengono eseguiti 10 rounds; per AES-192, 12 rounds; e per AES-256, 14 rounds.
9) *Output del Blocco Cifrato*: Una volta completati tutti i rounds, il blocco di dati viene cifrato. Il risultato finale è il blocco di dati cifrati.

La decifratura con AES segue un processo simile, ma con l'uso di chiavi di round inverse e operazioni inverse per ripristinare il blocco di dati originale.

4.5.3 AES e attacchi crittografici

L'algoritmo AES è generalmente considerato molto robusto e resistente contro una vasta gamma di attacchi crittografici. Tuttavia, come qualsiasi algoritmo crittografico, è importante considerare le sue vulnerabilità, minacce potenziali e prestazioni. Di seguito sono riportati alcuni attacchi crittografici di cui AES è vulnerabile:

- *Attacchi Brute Force*: AES è vulnerabile agli attacchi brute force, in cui un aggressore cerca tutte le possibili chiavi finché non trova quella corretta. Tuttavia, con lunghezze di chiave di 128, 192 o 256 bit, AES richiede quantità immense di tempo e risorse computazionali per essere vulnerabile a tali attacchi. La lunghezza della chiave è direttamente proporzionale alla resistenza contro gli attacchi brute force.
- *Attacchi Side-Channel*: AES potrebbe essere vulnerabile agli attacchi side-channel, come gli attacchi basati su analisi del consumo di energia o sul tempo di esecuzione. Questi attacchi sfruttano informazioni ausiliarie, come il consumo di potenza o i tempi di risposta, per dedurre la chiave. Implementazioni hardware o software deboli possono aumentare la vulnerabilità a questi attacchi.
- *Attacchi a Testo in Chiaro Noti* (*Known-Plaintext Attacks*): In alcune situazioni in cui un aggressore ha accesso a dati cifrati e corrispondenti testi in chiaro, potrebbe cercare di dedurre la chiave crittografica. Tuttavia, AES è progettato per resistere a questo tipo di attacchi.

Nonostante AES sia vulnerabile a questi tipi di attacchi, possiede ottime prestazioni.

Difatti, AES è noto per le sue prestazioni efficienti, specialmente quando utilizzato con chiavi di 128 bit. È supportato da hardware dedicato nella maggior parte dei moderni processori, il che lo rende molto veloce nella cifratura e decifratura dei dati.

Esistono inoltre molte implementazioni ottimizzate di AES per vari tipi di hardware, dal software su CPU agli acceleratori hardware su FPGA (Field-Programmable Gate Array) e ASIC (Application-Specific Integrated Circuit). Queste implementazioni sono progettate per offrire prestazioni ottimali e sicurezza.

Infine, AES è ampiamente utilizzato in tutto il mondo per proteggere dati sensibili in applicazioni come la crittografia dei dati archiviati, le connessioni Internet sicure (HTTPS), le reti VPN e molto altro ancora. La sua diffusione testimonia la sua affidabilità e prestazioni.

In sintesi, AES è un algoritmo crittografico molto forte ed efficiente che resiste bene a molti tipi di attacchi crittografici noti. Tuttavia, è importante mantenere implementazioni sicure e prendere in considerazione le minacce potenziali, soprattutto in contesti in cui gli attacchi side-channel possono essere una preoccupazione. Nel complesso, AES è una scelta solida per la cifratura di dati sensibili.

4.5.4 Complessità computazionale di AES

La complessità computazionale di AES **dipende dalla lunghezza della chiave e dal numero di round di cifratura che vengono eseguiti**. Un round di cifratura consiste in quattro operazioni: *SubBytes, ShiftRows, MixColumns* e *AddRoundKey*. Il numero di round varia da 10 a 14 a seconda della lunghezza della chiave (128, 192 o 256 bit). Ogni round richiede un certo numero di operazioni elementari, come XOR, sostituzioni e moltiplicazioni in un campo finito. La complessità computazionale di AES può essere espressa come il numero di operazioni elementari necessarie per cifrare o decifrare un blocco di dati di 128 bit.

Secondo uno studio del 2009, la complessità computazionale di AES con chiave di 128 bit è di circa 13.000 operazioni elementari per blocco, mentre con chiave di 256 bit è di circa 20.000 operazioni elementari per blocco. Questi valori sono validi per una macchina RAM che usa istruzioni a 32 bit. Tuttavia, la complessità computazionale può variare a seconda dell'architettura della macchina, del linguaggio di programmazione e delle ottimizzazioni implementate. Ad esempio, usando istruzioni dedicate a 128 bit (come le SSE2), la complessità computazionale può essere ridotta a circa 3.000 operazioni elementari per blocco con chiave di 128 bit e a circa 4.000 operazioni elementari per blocco con chiave di 256 bit.

4.6 ZUC

ZUC è un algoritmo di cifratura a flusso sviluppato dall'azienda cinese Huawei. È stato scelto come algoritmo di cifratura per le reti di telecomunicazioni di quinta generazione (5G) in Cina. ZUC è stato progettato per garantire la sicurezza e la riservatezza dei dati nelle comunicazioni wireless ad alta velocità. L'algoritmo ZUC utilizza un generatore di numeri pseudo-casuali basato su una rete di automi cellulari [20][21].

4.6.1 Componenti di ZUC

È costituito da tre componenti principali: una funzione di inizializzazione, una funzione di produzione di bit pseudocasuali e una funzione di feedback.

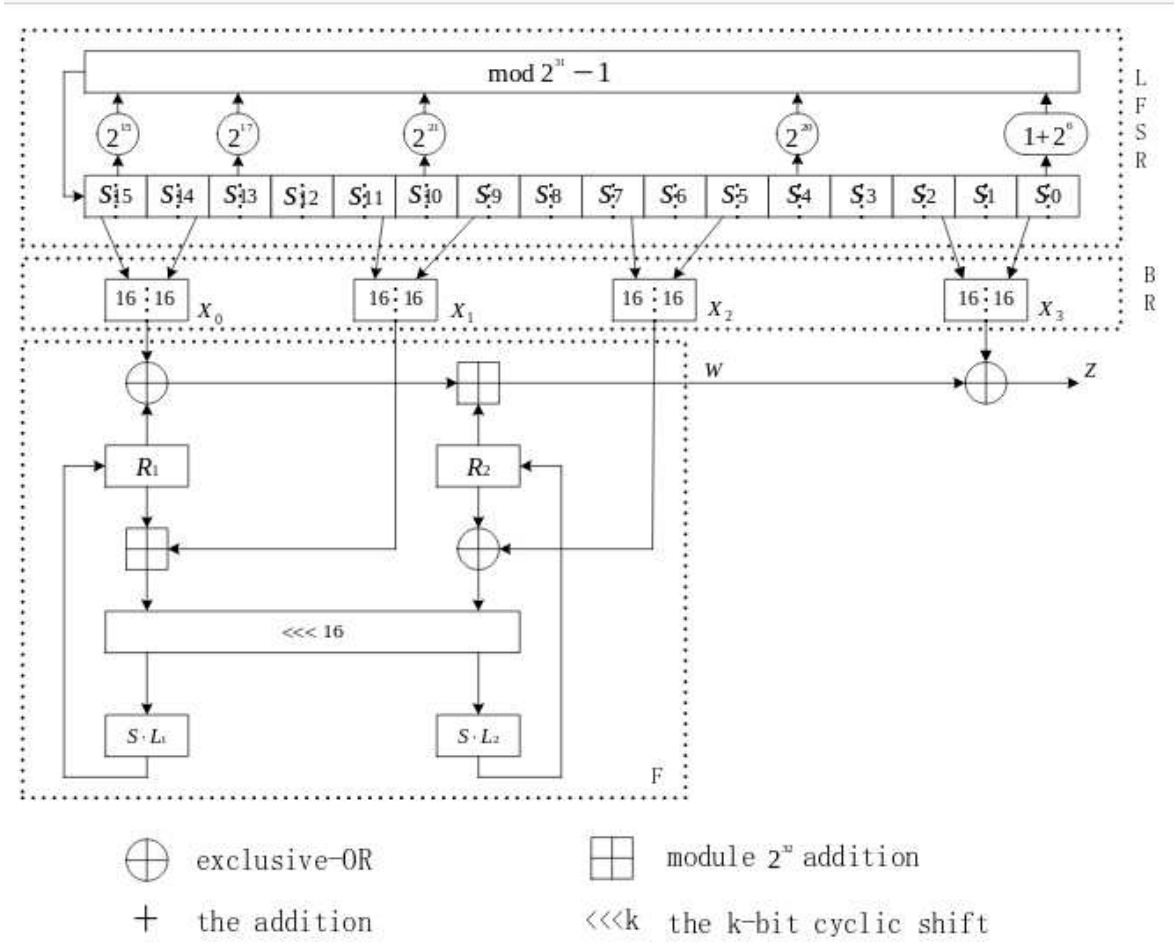

Figura 14: Struttura dell'algoritmo ZUC

In questa sezione, introduciamo brevemente l'ultimo algoritmo ZUC. ZUC è un cifrario world–orientedstream che prende una chiave segreta da 128 bit e un IV (Initialization Vector) da 128 bit come input e produce un flusso di chiavi di parole da 32 bit, che viene utilizzato per cifrare o decifrare i dati. Secondo la specifica ufficiale di ZUC, ZUC è composto da tre livelli logici, tra cui il livello superiore è un registro a scorrimento a retroazione lineare (LFSR per abbreviare) di 16 celle, il livello intermedio è l'operazione di riorganizzazione dei bit (BR per abbreviare) e il livello inferiore è una funzione non lineare F. La struttura di ZUC è illustrata nella figura in alto.

Il registro a scorrimento a retroazione lineare (LFSR) è composto da 16 celle di 31 bit (S0, S1, ..., S15). Ogni cella del registro (0 <= i <= 15) può assumere valori dall'insieme seguente: {1, 2, 3, ..., 2^31}. Il LFSR ha due modalità di funzionamento: la modalità di inizializzazione e la modalità di lavoro. La modalità di inizializzazione funziona come mostrato nell'Algoritmo 1.

Algorithm 1. *LFSRWithInitialisationMode(u){*

1. $v = 2^{15}s_{15} + 2^{17}s_{13} + 2^{21}s_{10} + 2^{20}s_4 + (1+2^8) \bmod (2^{31}-1)$;
2. $s_{16} = (u+v) \bmod (2^{31}-1)$;
3. If $s_{16} = 0$, then set $s_{16} = 2^{31}-1$;
4. $(s_1, s_2, ..., s_{15}, s_{16}) \rightarrow (s_0, s_1, \ldots, s_{14}, s_{15})$.

}

Nella modalità di lavoro, il LFSR non riceve alcun input e funziona come mostrato nell'Algoritmo 2.

Algorithm 2. *LFSRWithWorkMode(){*

1. $v = 2^{15}s_{15} + 2^{17}s_{13} + 2^{21}s_{10} + 2^{20}s_4 + (1+2^8) \bmod (2^{31}-1)$;
2. If $s_{16} = 0$, then set $s_{16} = 2^{31}-1$;
3. $(s_1, s_2, ..., s_{15}, s_{16}) \rightarrow (s_0, s_1, \ldots, s_{14}, s_{15})$.

}

Il livello intermedio di ZUC è la procedura di **riorganizzazione dei bit** (BR). Estrae 128 bit dalle celle del LFSR e forma quattro parole da 32 bit, di cui le prime tre parole verranno passate al livello successivo, la funzione non lineare F, mentre l'ultima parola sarà coinvolta nella produzione del flusso di chiavi. L'algoritmo per la riorganizzazione dei bit è descritto nell'Algoritmo 3.

Algorithm 3. *Bitreorganization*(){

1. $X_0 = s_{15H} \| s_{14L}$;
2. $X_1 = s_{11L} \| s_{9H}$;
3. $X_2 = s_{7L} \| s_{5H}$;
4. $X_3 = s_{2L} \| s_{0H}$.

}

La funzione non lineare F ha due celle di memoria da 32 bit, R1 e R2, e prende in input X0, X1 e X2, che sono le prime tre parole dell'output della procedura BR. Restituisce una parola da 32 bit W. Il processo dettagliato della funzione non lineare F è descritto nell'Algoritmo 4, in cui S è una S-box 32x32. Sono presenti anche trasformazioni lineari L1 e L2 di parole da 32 bit.

Algorithm 4. $F(X_0, X_1, X_2)$ {

1. $W=(X_0 \oplus X_1 \boxplus) R_{21}$;
2. $W_1=R_1 \boxplus X_1$;
3. $W_2 = R_2 \oplus X_2$;
4. $R_1 = S(L_1(W_{1L} \| W_{2H}))$;
5. $R_2 = S(L_2(W_{2L} \| W_{1H}))$.

}

Sia L1 che L2 sono trasformazioni lineari di parole da 32 bit e sono definite come segue:

- $L_1(X) = X\ (X<<<_{32}2) \oplus (X<<<_{32}10) \oplus (X<<<_{32}18) \oplus (X<<<_{32}24)$
- $L_2(X) = X\ (X<<<_{32}8) \oplus (X<<<_{32}14) \oplus (X<<<_{32}22) \oplus (X<<<_{32}30)$

La *procedura di caricamento delle chiavi* espande la chiave iniziale e il vettore iniziale di 128 bit in 16 interi da 31 bit come stato iniziale del LFSR. Siano la chiave iniziale di 128 bit k e il vettore iniziale iv, con $k = k_0 \| k_1 \| k_2 \| \dots \| k_{15}$ *e* $iv = iv_0 \| iv_1 \| iv_2 \| \dots \| iv_{15}$, dove k_i e iv_i, $0 \leq i \leq 15$, sono tutti i bytes. Dopodichè k_i e iv_i sono caricati nelle celle S con $s_i = k_i \| d_i \| iv_i$, dove d_i è una costante nota.

4.6.2 L'esecuzione di ZUC

L'esecuzione di ZUC è composta da due fasi: la fase di inizializzazione e la fase di lavoro. Durante la fase di inizializzazione, l'algoritmo esegue le operazioni descritte 32 volte per completare l'inizializzazione:

1) $Bitreorganization()$;
2) $W = F(X_0, X_1, X_2)$;
3) $LFSRWithInitialisationMode\ (w>>1)$.

Dopo la fase di inizializzazione, l'algoritmo passa alla fase di lavoro. Alla fine di questa fase, l'algoritmo passa alla fase di produzione del flusso di chiavi, in cui vengono eseguite le operazioni descritte per ciascuna iterazione e scarta l'output W della funzione non lineare F:

1) $Bitreorganization()$;
2) $W = F(X_0, X_1, X_2)$;
3) $LFSRWithInitialisationMode\ ()$.

Quindi l'algoritmo passa alla fase di generazione del flusso di chiavi, ovvero per ogni iterazione, le seguenti operazioni vengono eseguite una volta, e una parola da 32 bit Z viene prodotta come output:

1) $Bitreorganization()$;
2) $W = F(X_0, X_1, X_2) \oplus X_3$;
3) $LFSRWithInitialisationMode\ ()$.

4.6.3 ZUC e attacchi crittografici

ZUC è stato progettato per essere resistente a diversi tipi di attacchi crittografici, come gli attacchi di correlazione e gli attacchi differenziali. È stato incluso nello standard di cifratura per il 5G adottato dalla 3rd Generation Partnership Project (3GPP), un'organizzazione che sviluppa specifiche per le reti di telecomunicazioni mobili. Alcuni dei principali tipi di attacchi a cui ZUC è resistente includono:

- *Attacchi di forza bruta*: ZUC utilizza una chiave di cifratura a 128 bit, che offre un ampio spazio di ricerca per le possibili chiavi. Ciò rende estremamente difficile per un attaccante eseguire un attacco di forza bruta in cui si tenta ogni possibile combinazione di chiavi per rompere il cifrario.
- *Attacchi di crittanalisi lineare e differenziale*: ZUC è stato progettato per essere resistente ad attacchi di crittanalisi lineare e differenziale, che sono tipici attacchi crittografici che cercano di trovare correlazioni statistiche o differenze tra input e output crittografati per ricostruire la chiave segreta.
- *Attacchi di ricerca esaustiva delle chiavi*: ZUC è progettato per resistere agli attacchi di ricerca esaustiva delle chiavi, in cui l'attaccante cerca di dedurre la chiave di cifratura attraverso l'analisi delle relazioni tra la chiave, il testo in chiaro e il testo cifrato.
- *Attacchi basati su testo scelto e testo noto*: ZUC è progettato per essere resistente agli attacchi basati su testo scelto e testo noto, in cui l'attaccante ha accesso a testi in chiaro e corrispondenti testi cifrati.

L'algoritmo di cifratura di ZUC è progettato per rendere difficile dedurre la chiave o l'output corrispondente a testi specifici. Si noti che la resistenza di ZUC a questi attacchi dipende anche dall'implementazione specifica e dalle misure di sicurezza adottate nel contesto di utilizzo. È importante adottare le migliori pratiche di implementazione e configurazione per garantire la massima sicurezza quando si utilizza ZUC come algoritmo di cifratura.

4.6.4 Complessità computazionale di ZUC

La complessità computazionale di ZUC è stata progettata per garantire un equilibrio tra sicurezza ed efficienza.

La generazione della sequenza pseudo-casuale in ZUC richiede l'uso di registri di retroazione e la funzione di feedback. La complessità computazionale per generare una sequenza pseudo-casuale di lunghezza L è di $O(L)$, dove L è la lunghezza desiderata della sequenza.

La cifratura dei dati in ZUC coinvolge l'operazione di XOR tra la sequenza pseudo-casuale generata e i dati da cifrare. La complessità computazionale per eseguire l'operazione di XOR tra due sequenze di bit di lunghezza L è di $O(L)$.

Infine, la fase di inizializzazione di ZUC, che imposta gli stati iniziali dei registri di retroazione, richiede un tempo di inizializzazione fisso e indipendente dalla lunghezza dei dati da cifrare. Pertanto, la complessità computazionale dell'inizializzazione è considerata trascurabile rispetto alla generazione della sequenza pseudo-casuale e alla cifratura dei dati stessi.

Complessivamente, la complessità computazionale di ZUC è principalmente determinata dalla lunghezza della sequenza pseudo-casuale desiderata e dalla lunghezza dei dati da cifrare. La complessità computazionale è lineare rispetto alla lunghezza di questi parametri.

In generale, ZUC è stato progettato per essere efficiente e adatto per l'implementazione in dispositivi con risorse limitate, come telefoni cellulari e dispositivi IoT, pur garantendo una buona sicurezza crittografica.

Capitolo 5

Implementazione e risultati ottenuti

In questo capitolo verranno illustrate le implementazioni degli algoritmi di crittografia sviluppati. Tutti gli algoritmi riportati sono in grado di criptare un numero intero passato come argomento e decriptare il testo cifrato. Visti i vantaggi elencati nei capitoli precedenti, per la generazione e distribuzione delle chiavi è stato scelto l'algoritmo di Diffie-Hellman. Tutti i file prodotti sono contenuti all'interno della cartella *"Security"*, contenuta in *"protocolStack"* del simulatore, mostrata qua a fianco.

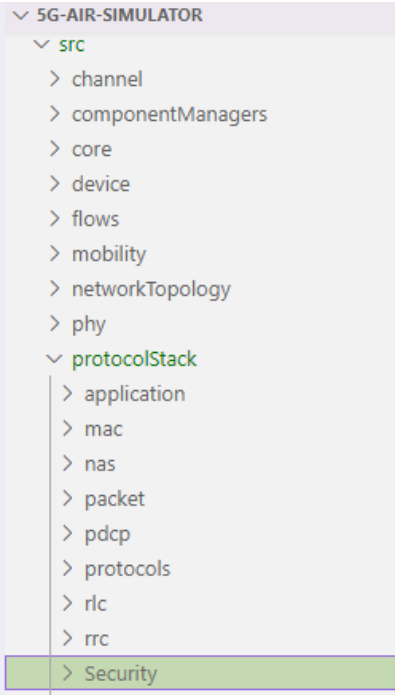

Figura 15: Cartella "Security" in 5G-air-simulator

5.1 Implementazione di AES

Lo sviluppo dell'algoritmo di cifratura AES si dirama in tre classi principali e un file *"miniz.c"*. Quest'ultimo file definisce le funzioni e le costanti per l'utilizzo della libreria di compressione/decompressione "miniz". La libreria fornisce un'implementazione leggera dello standard

di compressione zlib. Le definizioni presenti all'inizio del file consentono di disabilitare specifiche parti della libreria, come l'utilizzo di stdio per l'I/O dei file, le API per la gestione degli archivi ZIP e le API di compressione/decompressione zlib.
Nelle altre classi si trova l'effettiva realizzazione dell'algoritmo AES, responsabile di cifrare e decifrare un intero fornito come input.
In particolare, nella classe *"AES.cpp"* è contenuto l'algoritmo vero e proprio. Essa include funzioni per cifrare e decifrare dati utilizzando diversi modi di operare come ECB (Electronic Codebook), CBC (Cipher Block Chaining) e CFB (Cipher Feedback).

5.1.1 Cifratura a blocchi

ECB, CBC e CFB sono tre diverse modalità di cifratura utilizzate con algoritmi a blocchi come l'AES per crittografare dati.

ECB (Electronic Codebook):

In ECB, il testo in chiaro viene suddiviso in blocchi di dimensioni fisse (solitamente 128, 192 o 256 bit) e ciascun blocco viene cifrato separatamente utilizzando la stessa chiave. Questo significa che lo stesso blocco di testo in chiaro verrà cifrato nello stesso modo, indipendentemente dalla sua posizione all'interno del messaggio.
ECB è semplice da implementare ed è adatto per dati a blocchi indipendenti l'uno dall'altro. Tuttavia, è vulnerabile agli attacchi che sfruttano la ripetizione dei blocchi di cifrato, poiché blocchi identici di testo in chiaro producono blocchi di cifrato identici.

CBC (Cipher Block Chaining):

In CBC, il testo in chiaro viene suddiviso in blocchi e ciascun blocco viene messo in XOR con il blocco cifrato precedente prima di essere cifrato. Questa dipendenza tra i blocchi cifrati rende i risultati cifrati diversi anche per blocchi di testo in chiaro identici.
CBC è più sicuro di ECB perché introduce variabilità nell'output. CBC è ampiamente utilizzato in applicazioni di sicurezza informatica, come VPN, TLS/SSL per la crittografia delle comunicazioni su Internet e il cifrario di blocchi avanzato (AES) con CBC è uno standard crittografico comunemente adottato.

CFB (Cipher Feedback):

In CFB, il cifrario viene utilizzato per generare un flusso di bit casuale noto come "flusso di cifratura" (keystream). Il keystream viene quindi messo in XOR con il testo in chiaro per cifrarlo o decifrarlo. CFB opera a livello di bit anziché a livello di blocchi.

CFB è utilizzato in situazioni in cui la sincronizzazione tra mittente e destinatario è essenziale e la lunghezza dei dati da cifrare può variare. Può essere utilizzato per la cifratura di flussi di dati di lunghezza variabile.

In sintesi, queste modalità di cifratura sono utilizzate per proteggere dati sensibili in diverse situazioni. La scelta tra ECB, CBC e CFB dipende dalle specifiche dell'applicazione e dai requisiti di sicurezza. Nella classe "AES.cpp" vengono implementate tutte e tre le modalità di cifratura, ognuna con le sue caratteristiche. Per ognuna di esse esiste la corrispettiva modalità di decifratura. Ad esempio, di seguito, è riportato il codice della modalità CBC, una delle modalità che offre maggiore sicurezza.

```
unsigned char *AES::EncryptECB(const unsigned char in[], unsigned int inLen,
                               const unsigned char key[]) {
  std::cerr << "sto per controllare la lunghezza del packetsize" << std::endl;
  CheckLength(inLen);
  std::cerr << "finito di controllare la lunghezza del packetsize" << std::endl;
  unsigned char *out = new unsigned char[inLen];
  unsigned char *roundKeys = new unsigned char[4 * Nb * (Nr + 1)];
  KeyExpansion(key, roundKeys);
  for (unsigned int i = 0; i < inLen; i += blockBytesLen) {
    EncryptBlock(in + i, out + i, roundKeys);
  }

  delete[] roundKeys;

  return out;
}
unsigned char *AES::DecryptECB(const unsigned char in[], unsigned int inLen,
                               const unsigned char key[]) {
  std::cerr << "sto per controllare la lunghezza del packetsize" << std::endl;
  CheckLength(inLen);
  std::cerr << "fnito di controllare la lunghezza del packetsize" << std::endl;
  unsigned char *out = new unsigned char[inLen];
  unsigned char *roundKeys = new unsigned char[4 * Nb * (Nr + 1)];
  KeyExpansion(key, roundKeys);
  for (unsigned int i = 0; i < inLen; i += blockBytesLen) {
    DecryptBlock(in + i, out + i, roundKeys);
  }

  delete[] roundKeys;

  return out;
}
```

Figura 16: Implementazione di *CBC (Cipher Block Chaining)*

Ognuna di esse fa uso delle operazioni di KeyExpansion, EncryptBlock/DecryptBlock e XorBlocks.

Quest'ultima esegue l'operazione di XOR tra due blocchi di dati. L'operazione di XOR viene eseguita elemento per elemento tra i corrispondenti byte dei due array.

La funzione "*EncryptBlock*()" prende in input un blocco di dati da cifrare, lo divide in una matrice di stato 4xNb (dove Nb è il numero di colonne della matrice), applica una serie di operazioni di sostituzione, permutazione e combinazione dei dati nel blocco, e infine restituisce il blocco di dati cifrati. La funzione "*DecryptBlock*()" fa l'operazione inversa, prendendo in input un blocco di dati cifrati, applicando le operazioni inverse e restituendo il blocco di dati originali.

```
void AES::EncryptBlock(const unsigned char in[], unsigned char out[],
                       unsigned char *roundKeys) {
  unsigned char state[4][Nb];
  unsigned int i, j, round;

  for (i = 0; i < 4; i++) {
    for (j = 0; j < Nb; j++) {
      state[i][j] = in[i + 4 * j];
    }
  }

  AddRoundKey(state, roundKeys);

  for (round = 1; round <= Nr - 1; round++) {
    SubBytes(state);
    ShiftRows(state);
    MixColumns(state);
    AddRoundKey(state, roundKeys + round * 4 * Nb);
  }

  SubBytes(state);
  ShiftRows(state);
  AddRoundKey(state, roundKeys + Nr * 4 * Nb);

  for (i = 0; i < 4; i++) {
    for (j = 0; j < Nb; j++) {
      out[i + 4 * j] = state[i][j];
    }
  }
}
```

Figura 17: Implementazione della funzione EncryptBlock

Infine, la funzione "*KeyExpansion*()" implementa l'algoritmo di espansione della chiave AES. Prende in input una chiave di cifratura e restituisce una serie di sottochiavi utilizzate durante il processo di cifratura.

Durante la generazione delle sottochiavi, vengono utilizzate diverse operazioni come RotWord, SubWord e Rcon per manipolare i byte temporanei. Queste operazioni sono eseguite in base alla posizione corrente delle sottochiavi e al valore di Nk (lunghezza dei blocchi della chiave).

```
void AES::KeyExpansion(const unsigned char key[], unsigned char w[]) {
  unsigned char temp[4];
  unsigned char rcon[4];

  unsigned int i = 0;
  while (i < 4 * Nk) {
    w[i] = key[i];
    i++;
  }

  i = 4 * Nk;
  while (i < 4 * Nb * (Nr + 1)) {
    temp[0] = w[i - 4 + 0];
    temp[1] = w[i - 4 + 1];
    temp[2] = w[i - 4 + 2];
    temp[3] = w[i - 4 + 3];

    if (i / 4 % Nk == 0) {
      RotWord(temp);
      SubWord(temp);
      Rcon(rcon, i / (Nk * 4));
      XorWords(temp, rcon, temp);
    } else if (Nk > 6 && i / 4 % Nk == 4) {
      SubWord(temp);
    }

    w[i + 0] = w[i - 4 * Nk] ^ temp[0];
    w[i + 1] = w[i + 1 - 4 * Nk] ^ temp[1];
    w[i + 2] = w[i + 2 - 4 * Nk] ^ temp[2];
    w[i + 3] = w[i + 3 - 4 * Nk] ^ temp[3];
    i += 4;
  }
}
```

Figura 5.4 Implementazione della funzione KeyExpansion

5.1.2 Il costruttore

Il costruttore della classe *AES.cpp* è in grado di prendere in input una lunghezza della chiave a scelta tra 128, 192 e 256 bit in base alle necessità di sicurezza. Inoltre, esso inizializza le variabili Nk e Nr in base a questa lunghezza. Nk rappresenta il numero di parole da 32 bit nella chiave di cifratura (ossia la lunghezza dei blocchi), mentre Nr rappresenta il numero di round di cifratura da eseguire. Le variabili Nk e Nr vengono impostate in base al valore della lunghezza della chiave passata come parametro.

```
AES::AES(const AESKeyLength keyLength) {
  switch (keyLength) {
    case AESKeyLength::AES_128:
      this->Nk = 4;
      this->Nr = 10;
      break;
    case AESKeyLength::AES_192:
      this->Nk = 6;
      this->Nr = 12;
      break;
    case AESKeyLength::AES_256:
      this->Nk = 8;
      this->Nr = 14;
      break;
  }
}
```

Figura 18: Costruttore della classe "*AES.cpp*"

5.1.3 "AES_SRC.cpp" e "AES_DST.cpp"

AES_SRC.cpp e AES_DST.cpp costituiscono le altre due componenti del pacchetto AES. Queste classi includono le funzioni principali, ossia *main_AES_SRC* e *main_AES_DST*, che devono essere invocate dagli attori coinvolti nella comunicazione (mittente e destinatario) per iniziare il processo di cifratura e decifratura del payload del pacchetto inviato attraverso la rete.

Entrambe le classi istanziano un oggetto AES, configurando la lunghezza della chiave su 128 bit. La chiave segreta è stata generata utilizzando l'algoritmo Diffie-Hellman. Inizialmente, è stata scelta la modalità di cifratura ECB in quanto è una delle modalità più semplici da comprendere e

implementare. Tuttavia, è importante notare che è possibile cambiare la modalità di cifratura con facilità. Per farlo, è sufficiente definire un vettore di inizializzazione e chiamare la funzione *EncryptCBC*/*DecryptCBC* o *EncryptCFB*/*DecryptCFB*, passando il vettore di inizializzazione come argomento.

Un'altra caratteristica fondamentale di queste classi è la compressione del pacchetto prima della trasmissione attraverso la rete e la decompressione del pacchetto una volta che il destinatario lo ha ricevuto. Ciò consente di ridurre l'occupazione di banda durante la trasmissione dei dati, ottimizzando l'efficienza della comunicazione.

In sintesi, queste classi forniscono un'implementazione di AES con la possibilità di configurare la modalità di cifratura, garantendo al contempo la sicurezza nella comunicazione attraverso l'utilizzo di chiavi segrete generate mediante Diffie-Hellman e la compressione dei dati per una trasmissione più efficiente.

5.2 Implementazione di ZUC

L'implementazione dell'algoritmo di crittografia ZUC si articola in tre classi principali.

La classe *"ZUC.cpp"* contiene diverse funzioni che sono utilizzate per implementare l'algoritmo ZUC. In particolare:

- La funzione "*AddMod*" calcola la somma di due numeri interi a e b, e restituisce il risultato modulo 2^31-1. Se la somma supera il limite di 2^31-1, viene aggiunto 1 al risultato.
- La funzione "*PowMod*" calcola il valore di x*2^k modulo 2^31-1, dove x è un input e k è un esponente.
- Le funzioni "*L1*" e "*L2*" sono trasformazioni lineari utilizzate nell'algoritmo ZUC. Entrambe le funzioni prendono un input X e restituiscono il risultato di una serie di operazioni di "rotazione" (shift ciclico) su X.
- La funzione "*BitValue*" controlla se il valore di M nella posizione i è uguale a 0 o 1. Restituisce 0 se il valore è 0 e 1 se il valore è 1.
- La funzione "*GetWord*" restituisce una parola di 32 bit ki da una serie di bit k[i], k[i+1], ..., ovvero ki = k[i] || k[i+1] || ... || k[i+31].

- Le funzioni "*LFSRWithInitMode*" e "*LFSRWithWorkMode*" sono utilizzate per aggiornare lo stato corrente del LFSR (Linear Feedback Shift Register) durante la fase di inizializzazione e di lavoro dell'algoritmo ZUC.
- La funzione "*BR*" (Bit Reconstruction) ricostruisce le parole X0, X1, X2, X3 dallo stato corrente del LFSR.
- La funzione "*F*" è una funzione non lineare utilizzata nell'algoritmo ZUC. Prende in input le parole X0, X1, X2, X3 e i valori correnti R1, R2 e restituisce il valore di W.

Tuttavia, le funzioni principali sviluppate nell'algoritmo sono:

- La funzione "*Initialization*" è il processo di inizializzazione dell'algoritmo ZUC. Prende in input la chiave iniziale (k), il vettore iniziale (iv) e un intero w. Il codice inizializza gli elementi necessari per l'algoritmo di crittografia, espande la chiave, imposta le variabili e i registri a zero ed esegue un ciclo di inizializzazione per aggiornare gli elementi dell'array LFSR_S. Infine, viene eseguita una fase di lavoro per aggiornare nuovamente gli elementi dell'array LFSR_S. L'array LFSR_S in ZUC è un array di 16 elementi utilizzato per memorizzare lo stato dei registri a feedback lineare a scorrimento (LFSR) nell'algoritmo. Questi registri sono fondamentali per generare sequenze pseudo-casuali che vengono utilizzate per crittografare i dati.

```
void Initialization(unsigned char *k, unsigned char *iv) {
    unsigned int w;

    /* expand key */
    for(int i = 0; i < 16; ++i) {
        LFSR_S[i] = MAKEU31(k[i], EK_d[i], iv[i]);
    }
    /* set F_R1 and F_R2 to zero */
    F_R1 = 0;
    F_R2 = 0;
    unsigned int nCount = 32;
    while (nCount > 0){
        BitReorganization();
        w = F();
        LFSRWithInitializationMode(w >> 1);
        nCount--;
    }

    BitReorganization();
    F();
    LFSRWithWorkMode();
}
```

Figura 19: Implementazione della funzione *Initialization*

- La funzione "*GenerateKeyStream*" genera un flusso di chiavi pseudo-casuale utilizzando l'algoritmo ZUC. Durante ogni iterazione, vengono eseguite diverse operazioni, tra cui la riorganizzazione dei bit, il calcolo di un valore tramite la funzione "F", l'operazione di XOR con un elemento specifico e l'aggiornamento degli elementi dell'array LFSR_S. Il flusso di chiavi generato viene memorizzato nell'array "pKeyStream".

```
void GenerateKeyStream(unsigned int *pKeyStream, unsigned int KeyStreamLen){
    /* working cycles */
    for (int i = 0; i < KeyStreamLen; ++i){
        BitReorganization();
        pKeyStream[i] = F() ^ BRC_X[3];
        LFSRWithWorkMode();
    }
}
```

Figura 20: Implementazione della funzione *GenerateKeyStream*

- "main_*ZUC* ", ossia la funzione principale di ZUC. Si occupa di inizializzare il contesto dell'algoritmo utilizzando una chiave segreta e un vettore di inizializzazione, genera poi un flusso di chiavi e ne calcola la somma. Infine, restituisce la somma dei valori.

```
unsigned int main_ZUC(unsigned char secretKey[]){

    unsigned char iv[16] = {0x84, 0x31, 0x9a, 0xa8, 0xde, 0x69, 0x15, 0xca, 0x1f, 0x6b, 0xda, 0x6b, 0xfb, 0xd8, 0xc7, 0x66};

    // Converti l'array secretKey in una stringa key_string
    std::ostringstream key_stream;
    for (int i = 0; i < 16; ++i) {
        key_stream << std::hex << static_cast<int>(secretKey[i]);
    }
    std::string key_string = key_stream.str();

    unsigned char *key = (unsigned char*)key_string.c_str();

    int keyStreamSize = 0;
    std::cout << "Enter key stream size:\n>>";
    std::cin >> keyStreamSize;

    unsigned int *pKeyStream = new unsigned int[keyStreamSize];
    Initialization(key, iv);
    GenerateKeyStream(pKeyStream, keyStreamSize);

    std::cout << "Generated key stream:" << std::endl << std::endl;



    for (int i = 0; i < keyStreamSize; ++i){
        std::cout << std::dec <<  i << "\t0x" << std::oct << pKeyStream[i] << std::endl;
    }

    unsigned int result = 0;
    for (int i = 0; i < keyStreamSize; ++i) {
        result = result * 10 + pKeyStream[i];
    }

    return result;
}
```

Figura 21: Implementazione della funzione *main_ZUC*()

5.2.1 “ZUC_encrypt.cpp” e “ZUC_decrypt.cpp”

“ZUC_encrypt.cpp” e *“ZUC_decrypt.cpp”* costituiscono le altre due classi del pacchetto ZUC. Queste classi includono le funzioni principali, ossia *main_ZUC_SRC()* e *main_ZUC_DST()*, che devono essere invocate dagli attori coinvolti nella comunicazione (mittente e destinatario) per iniziare il processo di cifratura e decifratura del payload del pacchetto inviato attraverso la rete.

Entrambe le classi sono configurate per utilizzare una classe ZUC con una chiave di 128 bit.

Nella classe *“ZUC_encrypt.cpp”*, è presente una funzione denominata "*encrypt()*" che accetta un intero di plaintext e una chiave k come input. Questa funzione utilizza l'algoritmo di cifratura ZUC per cifrare l'intero e restituisce l'intero cifrato.
Inoltre, la classe contiene una funzione denominata "*main_ZUC_SRC()*" che sfrutta l'algoritmo Diffie-Hellman per generare una chiave segreta condivisa tra il mittente e il destinatario. Successivamente, utilizza questa chiave segreta per chiamare la funzione "*encrypt()*" e cifrare il payload del pacchetto.

D'altra parte, la classe *“ZUC_decrypt.cpp”* funziona in modo simile. Contiene una funzione chiamata "*decrypt()*" che decifra un intero cifrato utilizzando l'algoritmo di cifratura ZUC. Inoltre, possiede la funzione "*main_ZUC_DST()*" che genera una chiave segreta ZUC (di 128 bit) e utilizza questa chiave segreta per decifrare il dato cifrato.

5.3 Implementazione di Diffie-Hellman

L'implementazione dell'algoritmo di scambio chiave Diffie-Hellman è costituita da tre classi principali per la gestione dell'algoritmo e della comunicazione tra mittente e destinatario.

Nel dettaglio, "*DH.h*" possiede le funzioni per stabilire una comunicazione tra il server e il client per inviare e ricevere dati di tipo intero. *setUpClientSocket*() e *setUpServerSocket*() creano infatti le socket del client e del server, le configurano con l'indirizzo del server o del client a cui connettersi e si connettono ad esso sulla porta specificata.

```
void setUpServerSocket(int port) {
    // Creazione della socket server
    serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket == -1) {
        std::cerr << "Errore durante la creazione della socket server" << std::endl;
        exit(1);
    }

    int reuse = 1;
    setsockopt(serverSocket, SOL_SOCKET, SO_REUSEADDR, &reuse, sizeof(reuse));

    // Configurazione dell'indirizzo del server
    sockaddr_in serverAddress{};
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_addr.s_addr = INADDR_ANY;
    serverAddress.sin_port = htons(port); // Sostituisci con la porta desiderata

    // Bind della socket server all'indirizzo del server
    if (bind(serverSocket, reinterpret_cast<sockaddr*>(&serverAddress), sizeof(serverAddress)) == -1) {
        std::cerr << "Errore durante il binding della socket server" << std::endl;
        exit(1);
    }

    // In ascolto delle connessioni
    if (listen(serverSocket, 1) == -1) {
        std::cerr << "Errore durante l'ascolto delle connessioni" << std::endl;
        exit(1);
    }

    std::cout << "Server in ascolto sulla porta: " << port << std::endl;
}
```

Figura 22: Codice della funzione *setUpServerSocket*

La classe utilizza la libreria *sys/socket.h*; pertanto, è compilabile soltanto su sistemi operativi Linux e macOS.

Le altre funzioni implementate in "*DH.h*" sono:

- *sendLongLong*(): utilizzata per inviare al destinatario il dato da comunicare. Questo dato viene spedito grazie all'uso della funzione "*send*()" dei socket.

```
int sendLongLong(long long int dataToSend) {
    ssize_t bytesSent = send(clientSocket, &dataToSend, sizeof(dataToSend), 0);
    if (bytesSent == -1) {
        std::cerr << "Errore durante l'invio di long long int" << std::endl;
    }
    return 0;
}
```

Figura 23: Codice della funzione *sendLongLong*()

- *receiveLongLong*(): utilizzata per ricevere dal destinatario il dato da inviato. Questo dato si ottiene grazie all'uso della funzione "*recv*()" dei socket.

```
long long int receiveLongLong() {
    long long int receivedData;
    ssize_t bytesRead = recv(clientSocket, &receivedData, sizeof(receivedData), 0);
    if (bytesRead == -1) {
        std::cerr << "Errore durante la ricezione di long long int" << std::endl;
        return -1;
    }
    return receivedData;
}
```

Figura 24: Codice della funzione *receiveLongLong*()

- *getPublicKey*(): genera una chiave casuale estraendo un numero compreso tra 2 e 256. Per ottenere il valore viene utilizzata la funzione "*rand*()".

```
long long int getPublicKey (){
static bool srandCalled = false;

if (!srandCalled) {
    srand(time(NULL));
    srandCalled = true;
}

long long int A = rand() % 256 + 2;
return A;
}
```

Figura 25: Codice della funzione *getPublicKey*()

- *power*(): dati 3 numeri interi, G,a e P, restituisce il risultato dell'operazione G^a mod P.

```
long long int power(long long int G, long long int a, long long int P){
if (a == 0)
    return 1;

long long int result = 1;
G = G % P;

while (a > 0) {
    if (a % 2 == 1)
        result = (result * G) % P;

    a = a >> 1; // Dividi a per 2
    G = (G * G) % P;
}

return result;
```

Figura 26: Codice della funzione *getPublicKey*()

5.3.1 "DH_SRC.cpp" e "DH_DST.cpp"

"*DH_SRC.cpp*" e "*DH_DST.cpp*" sono le classi che implementano l'algoritmo di Diffie-Hellman, per la generazione di una chiave condivisa tra due parti, in questo caso mittente e destinatario.

In particolare, "DH_SRC.cpp" esegue i seguenti passi:

- Viene inizializzato un oggetto Diffie-Hellman e generata la chiave privata e pubblica.
- Viene configurata una socket client e inviata la chiave pubblica P al destinatario.
- Viene configurata una socket server e attesa la connessione da parte del destinatario.
- Viene ricevuta la chiave pubblica G dal destinatario.
- Viene calcolata la chiave a e la chiave condivisa x utilizzando la funzione power.
- Viene configurata una nuova socket server e attesa la connessione dal destinatario.
- Viene ricevuta la chiave y dal destinatario.
- Viene inviata la chiave x al destinatario utilizzando una socket client.
- Viene calcolata la chiave condivisa ka tra SRC e DST utilizzando la funzione power.
- La funzione restituisce la chiave condivisa ka.

"*DH_DST.cpp*" invece effettua le seguenti operazioni:

- Viene inizializzato un oggetto Diffie-Hellman e generata la chiave privata e pubblica.
- Viene configurata una socket client e inviata la chiave pubblica G al mittente.
- Viene configurata una socket server e attesa la connessione da parte del mittente.
- Viene ricevuta la chiave pubblica P da mittente.
- Viene calcolata la chiave b e la chiave condivisa y utilizzando la funzione power.
- Viene configurata una nuova socket server e attesa la connessione dal mittente.
- Viene inviata la chiave y al destinatario utilizzando una socket client.
- Viene ricevuta la chiave x dal mittente.
- Viene calcolata la chiave condivisa kb tra SRC e DST utilizzando la funzione power.

Parametri	AES	ZUC
Tipo di crittografia	AES è un algoritmo di crittografia a blocchi. Cifra e decifra i dati in blocchi di dimensioni fisse (128 bit) alla volta.	ZUC è un algoritmo di stream cipher. Cifra e decifra i dati un bit alla volta o in piccoli flussi di dati
Applicazioni principali	AES è ampiamente utilizzato in una vasta gamma di applicazioni, inclusi sistemi di crittografia dati, comunicazioni sicure su Internet, dispositivi di archiviazione sicura e altro ancora.	ZUC è principalmente utilizzato nelle reti cellulari, come parte degli standard 4G e 5G, per la cifratura dei dati utente nelle comunicazioni mobili.
Sicurezza	AES è noto per essere uno degli algoritmi di crittografia più sicuri ed è stato ampiamente valutato e adottato da organizzazioni governative e aziende in tutto il mondo.	ZUC ha dimostrato una buona sicurezza nelle applicazioni per cui è stato progettato, ma la sua adozione è stata più limitata rispetto ad AES e non è stato soggetto a un livello di scrutinio crittografico comparabile.
Dimensione delle chiavi	AES supporta chiavi di diverse lunghezze, tra cui 128, 192 e 256 bit, consentendo un adeguato controllo sulla sicurezza.	ZUC utilizza chiavi di 128 bit.
Velocità	La velocità di AES può variare in base all'implementazione e alla lunghezza della chiave utilizzata. In generale, AES è noto per le prestazioni efficienti, specialmente su hardware moderno con istruzioni crittografiche specializzate.	ZUC è noto per essere un algoritmo di stream cipher veloce ed efficiente in termini di prestazioni.
Storia e adozione	AES è stato selezionato come standard crittografico dal National Institute of Standards and Technology (NIST) degli Stati Uniti ed è ampiamente utilizzato in tutto il mondo.	ZUC è stato sviluppato principalmente per le reti cellulari e ha una diffusione più limitata in altre applicazioni.

Figura 27: Confronto tra AES e ZUC

5.4 Tempi di Simulazione e Analisi delle Prestazioni

La latenza di rete può variare per una serie di motivi, come il carico di rete, la congestione, i ritardi di instradamento e altro ancora.

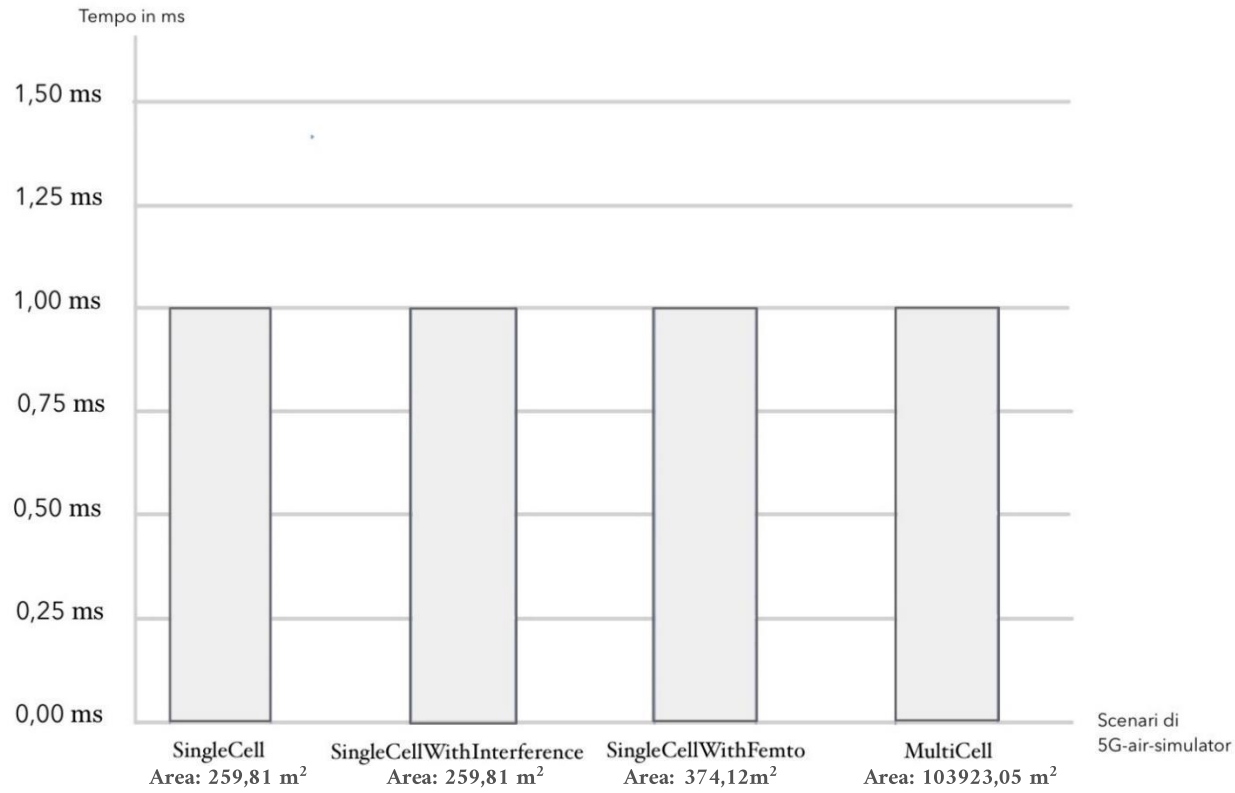

Figura 28: Ritardo medio dei pacchetti costante (1 ms) nei vari scenari di 5G-air-simulator.

La crittografia, tuttavia, non è la principale fonte di variabilità nella latenza di rete.

Pertanto, se i tempi di cifratura e di decifratura sono stati ottimizzati e sono brevi, il tempo che il pacchetto trascorre in rete rimane invariato rispetto a quando il pacchetto non è crittografato.

Viceversa, una crittografia inefficiente o complessa può comportare un aumento del ritardo, il che può essere problematico nelle applicazioni in cui è richiesta una latenza molto bassa, come le chiamate vocali o nelle operazioni real-time.

Inoltre, in reti con ritardo molto basso come il 5G, la ritrasmissione dei pacchetti crittografati potrebbe non essere tollerata, il che potrebbe portare alla perdita di dati in caso di ritardi eccessivi.

Per mitigare questi problemi, è stato necessario utilizzare algoritmi di crittografia efficienti e ottimizzati per le prestazioni.

Il tempo richiesto per crittografare un pacchetto dipende da diversi fattori, tra cui la complessità dell'algoritmo di crittografia utilizzato, la potenza di elaborazione del dispositivo e la dimensione dei dati da crittografare.
Tuttavia, un tempo di crittografia di pochi millisecondi per un singolo pacchetto è generalmente considerato abbastanza veloce e può indicare che si sta utilizzando una implementazione efficiente dell'algoritmo AES e ZUC.

Nel corso del lavoro di tesi, sono stati condotti una serie di esperimenti per valutare le prestazioni degli algoritmi implementati.
Durante questi esperimenti, sono stati raccolti i seguenti dati:

- Tempo "*real*":

  il tempo totale trascorso dall'inizio all'end dell'esecuzione del programma, compresi eventuali tempi di attesa o di inattività.

- Tempo "*user*":

  il tempo di CPU trascorso nell'esecuzione del codice dell'applicazione stessa, cioè il tempo durante il quale il sistema operativo ha eseguito il codice del programma in modalità utente.

- Tempo "*sys*":

  il tempo di CPU trascorso nell'esecuzione di chiamate di sistema del kernel del sistema operativo a causa del programma.

I dati di esecuzione sono stati raccolti utilizzando il comando "time" e sono espressi in secondi (s).

Questi tempi sono stati misurati su una macchina di prova Ubuntu 20.04 LTS con CPU Intel Core i7-8700K e architettura a 64 bit.
I risultati delle misurazioni sono presentati nella Tabella 1 e Tabella 2. Tutti i dati in essa riportati sono ottenuti dalla media di tre misurazioni disgiunte.

Algoritmo		Tempo medio (s)
Cifratura AES con Diffie-Hellman	real	0,059
	user	0,006
	sys	0,011
Decifratura AES con Diffie-Hellman	real	0,062
	user	0,004
	sys	0,002
Cifratura AES senza Diffie-Hellman	real	0,007
	user	0,005
	sys	0,002
Decifratura AES senza Diffie-Hellman	real	0,008
	user	0,004
	sys	0,003

Tabella 1: Risultati delle simulazioni eseguite sull'algoritmo di AES

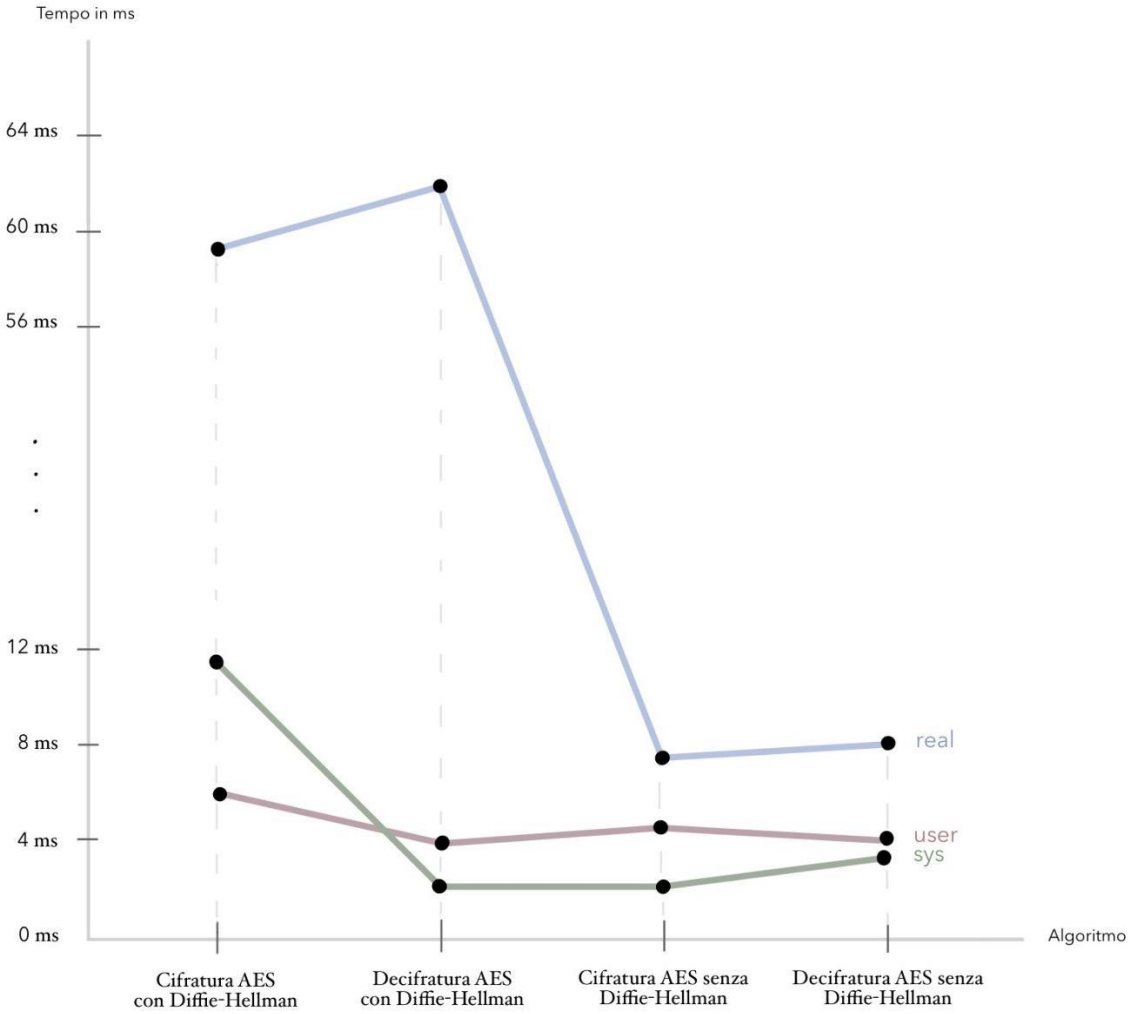

Figura 29: Tempo real, user e sys medio in millisecondi impiegato da AES.

Algoritmo		Tempo medio (s)
Cifratura ZUC con Diffie-Hellman	user	0,007
	sys	0,002
Decifratura ZUC con Diffie-Hellman	user	0,004
	sys	0,003
Cifratura ZUC senza Diffie-Hellman	user	0,004
	sys	0,001
Decifratura ZUC senza Diffie-Hellman	user	0,004
	sys	0,001

Tabella 2: Risultati delle simulazioni eseguite sull'algoritmo di ZUC

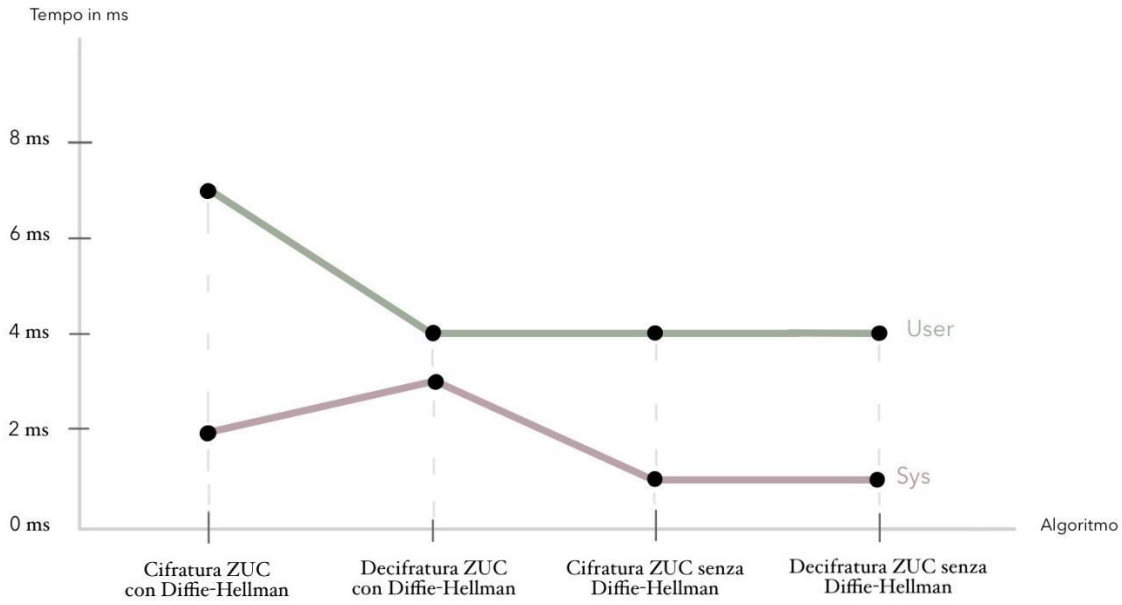

Figura 30: Tempo user e sys medio in millisecondi impiegato da ZUC.

Come mostrato dai risultati, le prestazioni degli algoritmi implementati sono ottimizzate e il tempo di crittografia è limitato a pochi secondi. Tuttavia, bisogna tenere conto di alcune osservazioni:

- Nel protocollo di scambio di chiavi di Diffie-Hellman, prima di crittografare un pacchetto, le due parti coinvolte devono generare una chiave condivisa segreta. Questo processo coinvolge una serie di scambi di dati tra le parti, che implica un aumento dei tempi di esecuzione.

- Nel caso dell'algoritmo ZUC, il tempo 'real' può variare in base alla lunghezza del keystream specificata dall'utente. Ad esempio, con una dimensione del keystream di 100000 byte, il tempo ‘user’ si aggira intorno ai 0,0045 s e il tempo ‘sys’ ai 0,3000 s, peggiorando quindi le prestazioni dell’algoritmo. Tuttavia, la lunghezza tipica nella maggior parte delle implementazioni di ZUC del keystream è di 32 byte (256 bit). Con quest’ultima dimensione si ottengono dei resultati del tutto simili a quelli elencati nella tabella 1.
- Il tempo totale di esecuzione di ZUC può variare in base alla tempistica con cui l'utente specifica la dimensione del keystream desiderato. Per fare un esempio, se l'utente richiede 20 secondi per fornire questa informazione, il tempo complessivo "real" sarà di poco superiore ai 20 secondi.

Per stabilire quindi la latenza end-to-end di un pacchetto criptato, bisogna sommare al ritardo medio dei pacchetti del simulatore il tempo impiegato per criptare e decriptare il messaggio.
Infatti, siccome le operazioni di encryption e decryption vengono svolte in locale ed in modo ottimizzato da ognuno dei due utenti coinvolti nella comunicazione, il tempo trascorso in rete dal pacchetto rimane invariato.

Dai dati raccolti emerge che le implementazioni di AES e ZUC, che sfruttano il protocollo di scambio delle chiavi Diffie-Hellman, sono state realizzate con efficienza e ottimizzazione. Inoltre, tali implementazioni non compromettono le prestazioni complessive di 5G-air-simulator, confermando così la loro capacità di operare in modo efficiente all'interno dell'ambiente simulato.

Capitolo 6

Conclusioni

Il lavoro di tesi è stato diviso in due parti. La prima ha fornito una panoramica delle caratteristiche principali del 5G e della crittografia, incentrandosi in particolare sulle vulnerabilità e sulle possibili soluzioni con sistemi crittografici. Nello specifico, sono stati approfonditi diversi argomenti riguardo la sicurezza informatica, come l'uso di chiavi simmetriche o asimmetriche, i diversi attacchi e vulnerabilità delle reti 5G, gli algoritmi di crittografia utilizzati nel contesto del 5G e le complessità computazionali di ognuno di essi.
La seconda parte si è occupata di descrivere lo sviluppo delle primitive crittografiche utilizzate all'interno di 5G-air-simulator per risolvere i problemi di sicurezza dei pacchetti in rete. Per ognuna di esse sono state elencate le caratteristiche e le scelte implementative effettuate, fornendo uno scenario completo del progetto.

In conclusione, l'evoluzione verso il 5G rappresenta un passo significativo nell'ambito delle telecomunicazioni e dell'interconnessione globale. Tuttavia, con l'aumento della complessità e della potenza delle reti, emergono nuove sfide in termini di sicurezza informatica. La tesi ha evidenziato l'importanza di implementare correttamente le primitive crittografiche nel contesto del 5G al fine di garantire la riservatezza e l'integrità delle comunicazioni.

Nonostante il forte interesse per le opportunità offerte dalla rete 5G, è fondamentale riconoscere che essa non è immune da minacce e vulnerabilità. L'aumento del numero di dispositivi connessi, la diffusione dell'Internet delle cose e la crescente complessità delle reti mobile costituiscono una superficie d'attacco più ampia che richiede una rigorosa attenzione alla sicurezza.

La tesi ha fornito un approfondimento per la comprensione delle implementazioni di primitive crittografiche nel contesto del 5G e ha evidenziato l'importanza di una cultura della sicurezza in continua evoluzione. L'applicazione di protocolli di sicurezza durante le simulazioni ha dimostrato di essere uno strumento essenziale per valutare l'impatto sulle performance delle reti 5G.

In questo scenario in continua evoluzione, il lavoro futuro potrebbe concentrarsi ulteriormente sull'ottimizzazione delle implementazioni crittografiche per le reti 5G e sull'identificazione e mitigazione delle nuove minacce emergenti, fornendo ulteriori ottimizzazioni degli algoritmi, valutazioni di altri algoritmi di crittografia o esperimenti su diverse condizioni di rete.

La sicurezza rimane una priorità fondamentale per garantire che il 5G continui a essere un protocollo di comunicazione affidabile e sicuro nel contesto della crescente connettività globale.

Bibliografia

[1] 5G: una nuova era per le telecomunicazioni (parte 1), 5 marzo 2021
URL: https://www.bipxtech.it/it/5g-nuova-era-telecomunicazioni-parte-1/

[2] Tutto sul 5G, benefici, vantaggi ed evoluzione, 12 febbraio 2021
URL: https://blog.osservatori.net/it_it/reti-5g-cosa-sono

[3] Cox, Christopher. An Introduction to 5G: The New Radio, 5G Network and Beyond. John Wiley & Sons, 2020.

[4] 5G System Overview - 3gpp.org. 8 agosto 2022
URL: https://www.3gpp.org/technologies/5g-system-overview

[5] 5G – Tecnologie e Prospettive, 18 aprile 2021
URL: https://cs.unibg.it/martignon/documenti/reti/RetiCellulariNuovaGenerazione.pdf

[6] William Stallings, "Cryptography and Network Security" Principles and Practice, Global Edition, Pearson Education Limited, 2016

[7] Processi crittografici: una panoramica
URL: https://www.ionos.it/digitalguide/server/sicurezza/processi-crittografici-una-panoramica/

[8] SCAMBIO DI CHIAVI CRITTOGRAFICHE
URL: https://www.c3t.it/awareness/downloads/brochure_key_exchange_v5.0.pdf

[9] 5G-air-simulator - poliba.it.
URL: https://telematics.poliba.it/5G-air-simulator

[10] S. Martiradonna, A. Grassi, G. Piro, and G. Boggia,"5G-air-simulator: an open-source tool modeling the 5G air interface", Computer Networks (Elsevier), 2020
URL: https://github.com/telematics-lab/5g-air-simulator

[11] telematics-lab/5G-air-simulator - Github. 1 marzo 2021
URL: https://github.com/telematics-lab/5G-air-simulator

[12] 5G-air-simulator | source system-level simulator modeling - Open Weaver.
URL: https://kandi.openweaver.com/c++/telematics-lab/5G-air-simulator

[13] Guida rapida alla GPLv3, 5 gennaio 2022
URL: https://www.gnu.org/licenses/quick-guide-gplv3.it.html

[14] Algoritmo di crittografia Diffie-Hellman - Computer Security. 26 dicembre 2018
URL: https://www.computersec.it/2018/12/26/algoritmo-di-crittografia-diffie-hellman/#:~:text=L'algoritmo%20Diffie%2DHellman%20permette,ricavare%20la%20chiave%20KC.

[15] Relazione speciale sulla sicurezza delle reti 5G, marzo 2022
URL: https://op.europa.eu/webpub/eca/special-reports/security-5g-networks-03-2022/it/index.html

[16] Seminario 5G: standardizzazione, aspetti di sicurezza e la strada verso il 6G, 12 dicembre 2022
URL: https://atc.mise.gov.it/images/documenti/seminari/5g_standardizzazione_evoluzione.pdf

[17] Specification # 33.501, 6 gennaio 2023
URL:https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3169

[18] La complessità degli algoritmi - Andrea Minini. 19 settembre 2019
URL: https://www.andreaminini.com/informatica/algoritmo/complessita-algoritmo

[19] 128-bit AES encryptor and decryptor in C++, 2023
URL: https://coderspacket.com/128-bit-aes-encryptor-and-decryptor-in-c

[20] Stream cipher ZUC - core component in the 3GPP confidentiality and integrity algorithms, 28 luglio 2019
URL: https://github.com/luminousmen/ZUC/blob/master/ZUC.cpp

[21] Languasco Alessandro; Zaccagnini Alessandro, "Manuale di crittografia", *Teoria, algoritmi e protocolli,* 2015