Документација за симулацијата на CCMP протоколот

За направената симулација на CCMP протоколот ги имплементирав следниве класи:

AESAlgorithm, ClearTextFrame, MICCalculator, Nonce, EncryptedFrame, Encryption, Decryption, Sender, Receiver, DecryptionOrIntegrityException, Test, CounterPreload.

AESAlgorithm

Во оваа класа креирам инстанца на AES во ECB mode без padding. Потоа правам специјализација на AES клучот за на крај да го иницијализирам cipher-от во енкрипциски мод и да го вратам иницјализираниот cipher. Со ова се осигурам дека само тие со правилниот клуч можат да ги дешифрираат пратените пораки.

ClearTextFrame

Во оваа класа ги чувам податоците на неекриптираната рамка која се користи за имплементација на CCMP. Овде се чуваат податоците за рамката frameHeader(associated data) која би требало да ги чува информациите за адресата, протоколот и слично, packageNumber односно бројот на пакетот кој не игра голема улога во делот за енкрипција и декрипција и на крај се чува data која всушност претставува податокот кој треба да се пренесе. Креирани се и три getters за истите.

MICCalculator

Оваа класа го содржи делот од кодот кој треба да направи симулација на MIC Calculation делот од CCMP.

Тука го чувам објектот за шифрирање кој ќе се користи за обработка на податоците и исто така се чува претходниот блок кој е потебен за XOR операцијата.

Во конструкторот претходникот блок потребен за XOR операцијата го иницијализирам на nonce.

За симулација на MIC Calculation, најпрво креирам метод processData каде ќе ги обработам frameHeader и data. Поради тоа што frameHeader и data се обработуваат на ист начин, доволен е само еден метод за таа обработка, а редоследот се запазува при повикување на овој метод во класата Sender.

Во методот се дели frameHeader/data на 128 битни блокови (16 бајти). На почетокот се прави XOR операција на nonce и на првиот блок од frameHeader. Резултатот од таа операција се шифрира со помош на AES и тој резултат се чува во previousBlock, кој е потребен за обработка на следниот блок се до крај.

Доколку последниот блок не е целосно исполнет односно нема 128 бита, се пополнува со 0 за да го исполниме блокот.

Методот getFinalMIC служи за да се извадат најбитните односно првите 64 бита добиени од XOR операцијата на последниот блок од data и претходно добиениот резултат, кои се потребни во делот за енкрипција.

Nonce

Класата Nonce е дефинирана според инструкциите во документацијата на IEEE 802.11i стандардот, познат и како WPA2. Оваа класа служи за генерирање на уникатна низа од бајти (nonce) која е од клучно значење во процесот на енкрипција и верификација.

Во конструкторот се иницијализира низа од 16 бајти за nonce. Првите 48 бита се пополнуваат со бројот на пакет, следните 48 бита се пополнуваат со MAC адресата на испраќачот, следните 8 се пополнуваат QoS вредност потребна за управување на мрежниот сообраќај. Останатие 16 бита се пополнуваат со 0 за да дојдеме до должина од 128 бита.

EncryptedFrame

Во оваа класа се чуваат податоците кои треба да бидат испратени од страна на испраќачот односно енкриптираниот дел и MIC, затоа и се чуваат тие податоци заедно до getters.

Encryption

По завршување на пресметката на MIC, следно што треба да се направи е да се имплементира делот за енкрипција. Како и во делот за пресметка на MIC, така и овде имаме параметар што се вика counter preload.

Во оваа класа чуваме инстанца на cipher-от која ни е потребна за пресметка на XOR операцијата, но овој пат на counter preload и на data. Повторно data ја делиме на блокови од 128 бита, counter preload го инкрементираме за 1, го ставаме во AES и потоа на добиениот резултат му правиме XOR со блокот од data. Потоа го инкременирање counter preload за 1 и повторно го повторуваме претходиот процес се

додека не го изминеме и последниот блок од data. Сите добиени блокови ги конкатенираме и на нив го додаваме 64 MIC добиен од MICCalculation.

Decryption

За да се осигураме дека добиената порака е пораката која е испратена потребно е да се направи декрипција. Во оваа класа правам декрипција и верификација на добиената порака.

Бидејќи го имам ciphertext-от и генерираниот counter preload, на сличен начин како во Encryption, но во обратна насока се прави XOR и на крај го добивам plaintext-от.

Овде повторно правам пресметка на MIC, користејќи го frame header-от, за да направам верификација на MIC, односно дали добиениот MIC од фазата на декрипција е истиот MIC со тој што го добивме од фазата на енкрипција.

Sender и Receiver

Бидејќи, во описот на задачата има барање за симулација на испраќање енкриптиран дел и MIC, креирав две класи Sender и Receiver.

Во класта Sender креирам објеки од класата MICCalculator и Encryption со цел да ги добијам подоците кои се потребни за да испратам еден објект од класата EncryptedFrame.

На сличен начин во класата Receiver, креирам објект од Decryption и го повикувам методот за верификација каде се проверуваат дали се исти MIC од делот за енкрипција и делот за декрипција и доколу не се се печати порака за грешка.