

Debreceni Egyetem  
Informatikai Kar

Kulcskiosztásos technológiák  
szenzorhálózatokban

Témavezető:

Dr. Pethő Attila

Egyetemi tanár, docens, tanszékvezető

Készítette:

Driszkó Károly

Programtervező-Matematikus

Debrecen

2009

## Tartalomjegyzék

|                                                              |    |
|--------------------------------------------------------------|----|
| 1. Bevezetés.....                                            | 4  |
| 2. Szenzorok, szenzorhálózat .....                           | 6  |
| 2.1 Szenzor definíciója, tulajdonságai, felépítése.....      | 6  |
| 2.2 Szenzorhálózat definíciója.....                          | 10 |
| 3. Kulcskiosztás.....                                        | 12 |
| 3.1 Általánosan a kulcskiosztás problémájáról.....           | 12 |
| 3.2 Szenzorhálózatok korlátozásai.....                       | 12 |
| 3.3 Biztonságos betöltés problémája szenzorhálózatokban..... | 13 |
| 3.4 Kiértékelési metrika.....                                | 14 |
| 3.5 Teljes hálózatra kiterjedő kulcs használata.....         | 14 |
| 3.6 Aszimmetrikus kriptográfia használata.....               | 16 |
| 4. Az AES blokkrejtjelező.....                               | 19 |
| 5. TinyOS operációs rendszer és Tossim szimulátor.....       | 23 |
| 5.1 TinyOS operációs rendszer lehetőségei, használata.....   | 23 |
| 5.2 Tossim szimulátor bemutatása.....                        | 28 |
| 5.2.1 Tossim szimulátor fordítása.....                       | 28 |
| 5.2.2 Tossim szimulátor és a Python nyelv használata.....    | 29 |
| 5.2.3 Tossim hálózati beállításai.....                       | 31 |
| 6. A SensorWatcher alkalmazás bemutatása.....                | 33 |
| 6.1 Az alkalmazásról.....                                    | 33 |
| 6.2 Szolgáltatás orientált programozás.....                  | 34 |
| 6.3 Fejlesztő környezet.....                                 | 34 |
| 6.4 Titkosítás.....                                          | 35 |
| 6.5 Szenzorok .....                                          | 37 |
| 6.6 Adatbázis kezelés.....                                   | 38 |
| 6.7 Felület – grafika.....                                   | 40 |
| 7. Összefoglalás .....                                       | 43 |
| 8. Függelék.....                                             | 44 |
| Irodalomjegyzék.....                                         | 46 |

*Ezúton szeretnék köszönetet mondani témavezetőmnek, Dr. Pethő Attila tanárúrnak a szakmai irányításért, és hogy lehetőséget biztosított dolgozatom elkészítéséhez. Köszönöm a rengeteg türelmet, és a hasznos tanácsokat, amik nélkül ez a dolgozat nem jöhetett volna létre.*

## 1. Bevezetés

A XXI. század első évtizedeiben megjelenő és az egyik leggyorsabban fejlődő technológia jelensége az érzékeléstechnika. A szenzorok egyre nagyobb információfeldolgozó-képességgel és tárolási lehetőséggel rendelkeznek, melyek hálózatba rendezve különösen eredményesen tevékenykednek.

A vezeték nélküli hálózatokba rendezett szenzorok szerteágazó alkalmazásaira számíthatunk: Intelligens Otthonok alapjait képezhetik, adatokat gyűjtenek a környezetszennyezésről, a közlekedésről, hőmérsékletről, az emberek vérnyomásáról, szenzor vezérelt robotok dolgoznak a gyártásban.

Bevezetésként kettő, 2005-ben indult fejlesztést is szeretnék megemlíteni példaként, melyek jól illusztrálják, hogy milyen célokat szolgálhatnak a szenzorhálózatok: Kanadában kamerákkal és szenzorhálózatokkal egybeépített rendszerrel vették fel a harcot a terrorizmus ellen. Az optikai és infravörös érzékelők, műholdak valós idejű képei adatokat szolgáltatnak, miközben a kamerák autópályákat, épületeket, hajókat monitoroznak. Természetesen a kiépített rendszer nem csak a terrorizmus ellen, hanem pl. természeti katasztrófák vagy súlyos balesetek esetén is segítséget nyújt. A 2008-ra kiépülő rendszert több kanadai cég, egyetem és az ország legnagyobb katonai laboratóriuma együttesen fejlesztette ki.

Hasonlóan jártak el a zürichi és baseli egyetem valamint a zürichi műszaki főiskola PermaSense elnevezésű projektje során. A projekt keretein belül egy két éve működő vezeték nélküli szenzorhálózatot építettek ki a Matterhornon (4478 m), amelynek segítségével a kutatók adatokat gyűjtenek arról, hogy milyen hatással van a klímaváltozás a sziklaomlásokra. Ezzel a rendszerrel valós időben tudják ellenőrizni azokat a területeket, ahol a hegyomlások és földcsuszamlások veszélye különösen nagy. A szenzorok folyamatosan szolgáltatnak adatokat, és különlegességük, hogy -40°C-on

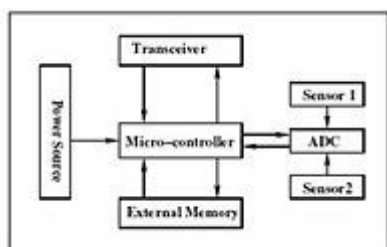
is kifogástalanul működnek, valamint legalább hároméves élettartalmú elemmel vannak ellátva, mivel a szenzorok cseréje elhelyezkedésükből adódóan nagyon költséges.

Dolgozatom célja, hogy egy konkrét példán keresztül bemutassam egy szenzorhálózat működését. A szenzorhálózatok működésének megértéséhez az első fejezetben részletesen bemutatom egy szenzor felépítését, kapcsolódási lehetőségeit, valamint magát a szenzorhálózatok felépítését, fejlődésének menetét. A második fejezet a dolgozatom fő témájáról, a kulcskiosztás problémájáról, a szenzorhálózatok korlátozásairól, illetve megoldási lehetőségekről szól. A harmadik fejezetben az egyik legelterjedtebb és szenzorhálózatok esetében is szívesen használt titkosítási eljárást, az AES blokkrejtjelezőt mutatom be részletesen. Szenzorhálózatok irányításához számos operációs rendszert fejlesztettek ki, melyek közül a TinyOS operációs rendszert választottam. Sajnos, hálózatépítéshez szükséges szenzorok nem álltak a rendelkezésemre, így egy hálózati működést szimuláló Tossim nevű szimulációs rendszerrel helyettesítettem őket. Az ötödik fejezetben az operációs rendszert és a hozzá tartozó szimulátor lehetőségeit mutatom be a fontosabb tulajdonságaikon, lehetőségeiken keresztül. Az utolsó fejezetben pedig magáról a fejlesztésről, körülményeiről számolok be.

Szakdolgozatom témaválasztásában nagy szerepet játszott, hogy egy még friss, az informatika világában is újdonságnak számító területtel foglalkozhassak, illetve a fejlesztés során új programozási nyelvekkel, esetleg új gondolkodásmóddal ismerkedhessek meg.

## 2. Szenzorok, szenzorhálózat

### 2.1 Szenzor definíciója, tulajdonságai, felépítése



1. ábra Szenzor általános felépítése

[9] A szenzorcsomópont egy olyan csomópont a drótnélküli szenzorhálózatokban, ami alkalmas folyamatok végrehajtására, szenzoros információk gyűjtésére, és kommunikációra más, a hálózatba kapcsolódott csomóponttal. A szenzor csomópont egy jellegzetes felépítése az 1. ábrán látható.

Fő komponensei egy szenzorcsomópontnak a mikrokontroller, adó-vevő, külső memória, áramforrás és egy vagy több érzékelő.

[14]. A mikrokontroller egyetlen lapkára integrált, általában vezérlési feladatokra optimalizált számítógép. A mikrokontroller hajtja végre a feladatokat, adatfeldolgozást és irányítja a szenzorcsomópont többi alkotóelemének a működését. A mikrokontroller a legjobb választás a beágyazott rendszerekhez, mert rugalmasan tud kapcsolódni más eszközökhöz, programozható, áramfelvétele kevés, mivel ezek az eszközök kikapcsolhatók és a mikrokontroller csak néhány része lehet aktív.

A szenzorcsomópontok az ISM<sup>1</sup> frekvenciasávot használják, mely szabad sugárzást biztosít, széles spektrumú felhasználással és globális elérhetőséggel. A drótnélküli adatátviteli közeg különböző lehetőségei a rádió frekvencia, lézeres és infravörös kommunikáció. A lézervfény kevesebb energiát igényel, viszont szükség van

---

<sup>1</sup> ISM:(Industrial, Scientific, Medical) A nemzetközi szervezetek a 2.4Ghz-es frekvenciasávot jelölték ki engedély nélküli használatra. Ez a sáv szabadon igénybe vehető a világon, mindenki számára bárhol könnyen hozzáférhető vezeték nélküli számítógépes hálózati rendszer létrehozására.

egy irányvonalra és érzékeny a légköri feltételekre is. Az infravörösnek a lézerhez hasonlóan nincs szüksége antennára, de korlátozott az adattovábbítási kapacitása. A rádió frekvencia alapú kommunikáció a legjobb választás, amit a legtöbb szenzorhálózati alkalmazás is használ. A vezeték nélküli szenzorhálózatok 433Mhz és 2.4Ghz közötti kommunikációs frekvenciát használnak, amelyben az adó-vevőknek szüksége van egy egyedi azonosítóra.

Memória tekintetében 2 típust használ egy szenzorcsomópont, egy mikrokontrolleren belülit és egy FLASH<sup>2</sup> memóriát. A FLASH memória a költségei és tárolókapacitása miatt megfelelően használható. Memória szükséglete nagyon sokban függ az alkalmazásoktól. Két típusát különböztetjük meg: felhasználói memória tárolja az alkalmazáshoz kapcsolódó és személyes adatokat, míg a program memóriát az eszköz programozhatóságára használják.

Áramfelvétele az érzékelésnek, kommunikációnak és az adat-továbbításnak van. A legtöbb energiára az adattovábbításnak van szüksége. 1 Kb adatátvitelének energia költsége 100 méteres távolságba megközelítőleg megegyezik egy 3 millió utasítást elvégző processzor energiaszükségletével, ahol a processzor 1 Watt teljesítményt fogyaszt 100 millió művelet 1 másodperc alatt történő elvégzése közben. Az energia vagy elemekben, vagy kondenzátorban van tárolva. Az elemek a legfontosabb energiaellátói a szenzorcsomópontoknak, mivel léteznek feltölthető és nem újra-tölthető elemek is. 2 fontos energiatakarékosági eljárást különböztetünk meg: DPM (Dynamic Power Management) ill. DVS (Dynamic Voltage Scaling). A DPM gondoskodik a szenzorcsomópont azon részeinek a kikapcsolásáról, melyeket pillanatnyilag nem használ, vagy nem aktívak. A DVS séma módosítja az energiaszintet a nemdeterminisztikus munkamennyiségtől függően.

Érzékelők olyan hardver eszközök, melyek olyan értékeket mérnek, amelyek megtudják változtatni fizikai állapotaikat, mint pl. a hőmérséklet vagy a nyomás. A szenzorok a megfigyelt terület fizikai adatait mérik vagy érzékelik. A folyamatosan érzékelt analóg jeleket egy Analóg-digitális átalakítóval digitális jellé alakítja a szenzor és további feldolgozásra a mikrokontrollerhez küldi.

---

2 Flash memória: nem felejtő memória, mely az adatokat a tápfeszültség kikapcsolása után is megőrzi(non-volatile), és melyet elektromosan lehet törölni és újraprogramozni, miközben a memória a felhasznált alkalmazásban marad.

Három kategóriába osztályozhatók a szenzorok:

- Passzív, irányítatlan szenzorok: A passzív szenzorok mérik az adatokat anélkül, hogy valójában megváltoztatnák a környezetet az aktív figyeléssel. Önmaguk termelik az energiát, azaz energiára csak ahhoz van szükségük, hogy felerősítsék analóg jeleiket.
- Passzív, keskeny sávú - sugárnyaláb szenzorok: Ezek a szenzorok is passzívak, viszont jól definiált fogalma van a mérés utasításáról. Jellegzetes példa a kamera.
- Aktív szenzorok: Szenzorok ezen csoportja aktívan figyeli a környezetét, mint pl. szonár vagy radar szenzor vagy földrengési szenzorok néhány típusa, melyek kisebb robbanások által okozott lökéshullámokról adnak információt.

A vezeték nélküli szenzorhálózatokban mindenre kiterjedő elméleti munkákhoz a passzív, irányítatlan szenzorokat részesítik előnyben. Szenzorcsomópontok térbeli sűrűsége a megfigyelési területeken magasabb lehet, mint  $20 \text{ csomópont/m}^3$ .

Megkülönböztetjük a csomópontok között még az átjáró csomópontokat, melyek biztosítják a kapcsolatot a szenzorhálózat és a külvilág között.

Az 1. Függelékben, táblázatba foglalva látható a manapság is forgalomban lévő szenzorcsomópontok listája műszaki adatokkal.

A szenzorok közül szeretném kiemelni és részletesen bemutatni a micaZ nevű szenzor felépítését, számítási és kommunikációs kapacitását.



**2. ábra micaZ szenzor**

[15]. A micaZ egy 2.4GHz-es csomópont modul, melyet alacsony energiafelvételt engedélyező vezeték nélküli szenzorhálózatokban használnak.

A szenzor jellemzői:

- 2.4-től 2.48GHz-ig terjedő ISM-el teljesen kompatibilis frekvenciasáv

- IEEE 802.15.4<sup>3</sup> szabványnak megfelelő RF adó-vevő

<sup>3</sup> IEEE 802.15.4: kis adatrátájú, kis komplexitású, nem engedély köteles rádió szabványa, mely a hálózati rétegek közül a fizikai és az adatkapcsolati réteget foglalja magába.

- Irányított, sorba rendezett szórt spektrumú rádió, mely ellenálló az RF interferenciákkal szemben és adatbiztonságot is nyújt
- 250 kbps –os adatátviteli sebesség
- Plug and Play a Crossbow érzékelő áramköreivel, adatgyűjtő áram-köreivel, átjáróival illetve a MoteWorks szoftverei engedélyezik érzékelő alkalmazások fejlesztését és különösen alacsony energiájú, elemmel működő hálózatok optimalizálását. A MoteWorks a nyílt forráskódú TinyOS operációs rendszerre épül, és biztosít megbízható ad-hoc szerteágazó hálózatokat, fejlesztetőkészítőket, szerveroldali szoftvereket vállalati hálózatok integrációjához és felhasználói felületet az analízishoz és a konfigurációhoz.

#### Processzor és Rádió Platform (MPR2400CA)

Az MPR2400CA az Atmega által gyártott ATmega128L-en alapszik. Az ATmega128L egy alacsony energiafelvételű mikrokontroller, mely MoteWorks szoftvert futtat a belső flash memóriáját használva. Az egyetlen, telepített processzor áramkörei (MPR2400) beállíthatóak a szenzoron bármilyen alkalmazás futtatásával, a folyamatok és a hálózati /rádiós kommunikációk egyidejűleg halmozhatóak. Az 51 tűs bővítő csatlakozó támogatja az Analóg Input, Digitális I/O, I2C<sup>4</sup>, SPI<sup>5</sup> és UART<sup>6</sup> interfészeket. Ezek az interfészek egyszerűvé teszik, hogy külső perifériák széles választékát csatlakoztathassuk. A szenzorokkal egyik legfontosabb elvárás manapság, hogy megtudják határozni saját helyzetüket. Ezt a problémát sikerült kiküszöbölni a számos csatlakozó és gps eszközök használatával. Manapság már a gps eszközök is alacsony áron hozzáférhetőek, és a csatlakozási lehetőségeik révén könnyedén használhatóak már szenzor hálózatok esetén is. A micaZ (MPR2400) IEEE 802.15.4 rádió nagy sebességet és hardver biztonságot (AES-128) is nyújt.

---

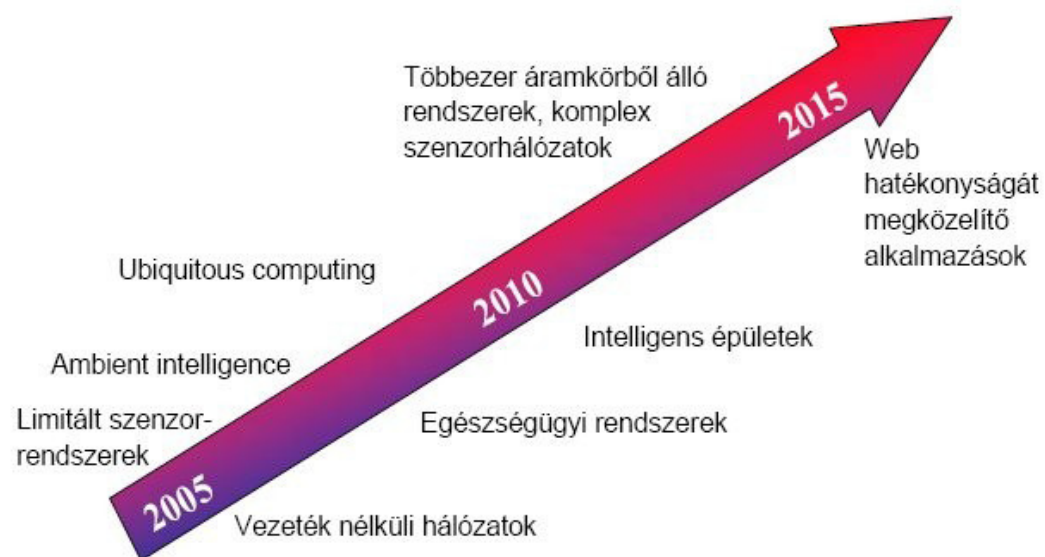
4 I2C: kétvezetékes szinkron adatátviteli rendszer integrált áramkörök összekapcsolá-sára. Félduplex, két vezetékből áll: SLC(az órajel) és SDA, az adatvezeték.

5 SPI:(Serial Peripheral Interface) egy nagy sebességű soros szinkron I/O rendszer. Az SPI alkalmas egy CPU és kiegészítő áramkörei összekapcsolására, de több processzor együttműködését is lehetővé teszi.

6 UART:(Universal Asynchronous Receiver/Transmitter) A számítógép egy olyan eszköze, mely adatot továbbít a párhuzamos és soros portok között. Az UART-ot általában más kommunikációs szabványokkal használják együtt, mint pl. EIA RS-232.

## 2.2 Szenzorhálózat definíciója

[5] A vezeték nélküli szenzorhálózat olyan hálózat, mely térben szétszórta, önálló működésű eszközökből áll, melyek szenzorokat használnak, hogy együttműködve figyeljék a fizikai és környezeti feltételeket, mint pl. hőmérséklet, hang, rezgés, nyomás, mozgás vagy szennyezettség. Vezeték nélküli szenzorhálózatok fejlesztését eredetileg olyan katonai feladatok indították el, mint pl. a harcmező figyelése. Azonban ma már számos civil alkalmazási területen használják, magába foglalva környezet és lakhely figyelését, egészségügyi alkalmazások, házi automatizálást, és forgalomirányítást. Egy szenzorhálózat általában egy vezeték nélküli ad-hoc hálózatot képez, ami azt jelenti, hogy minden szenzor támogat egy több-ugrásos routing algoritmust. (néhány csomópont továbbíthat adatokat a központi állomásnak). A vezeték nélküli ad hoc hálózatok egymással kommunikáló független csomópontok és terminálok decentralizált, elosztott rendszerei. Két főbb típust különböztetünk meg: intelligens szenzor (smart sensor networks) és mobil ad hoc hálózatokat (mobile ad hoc networks, MANETs). Előbbi egy adott földrajzi területre kiterjedő, a jelfeldolgozáshoz és a kommunikációhoz elegendő „intelligenciával” rendelkező érzékelők rendszere, utóbbi a mobil felhasználók sávszélesség által behatárolt vezeték nélküli decentralizált hálózata, melyben a mozgékony csomópontok töltik be a főszerepet. A jövőt tekintve megjelennek az önmagukat telepítő hálózatok. A hely-érzékeny szenzorok a web méretét és hatékonyságát megközelítő/elérő alkalmazásokban valósídják megfigyeléseket és méréseket szolgáltatnak a világról. Hálózataik mind gyakrabban funkcionálnak adatbázisokként is. Ehhez azonban a közeljövőben még számos olyan problémát kell megoldani, mint pl. a zaj, elhalkulás vagy interferencia, illetve ahhoz hogy az, hogy a csomópontok kiszolgálóként és útvonalválasztóként is jól működjenek, hatékonyabb routing-protokollokat kell kidolgozni.



**3. ábra: Várható fejlődés 2005-2015 között**

### 3. Kulcskiosztás

#### 3.1 Általánosan a kulcskiosztás problémájáról

Az általános kulcskiosztás problémája titkos kulcsok szétosztásának feladatát jelenti két kommunikáló fél között azzal a céllal, hogy megkönnyítse a biztonsági feladatokat, úgymint titkos közlés és hitelesítés.

Szenzorhálózatokban a kulcskiosztás rendszerint kombinált egy kezdeti kommunikációs szervezettel, melynek szerepe egy biztonságos kommunikációs infrastruktúra betöltése, miközben a telepített szenzor csomópontok összegyűjtése történik. Ezeknek a csomópontoknak újra kezdeti érték állítható be néhány titkos információval de nem kell előzetesen közvetlen kapcsolatban állniuk egymással. A kulcskiosztás ilyen fajta kombinált problémáját és a biztonságos kommunikáció létesítését betöltési problémának nevezzük.

A betöltési protokollnak nem csak engedélyeznie kell egy újonnan telepített szenzorhálózatban, hogy megnyisson egy biztonságos infrastruktúrát, hanem csomópontok későbbi csatlakozását is engedélyeznie kell a hálózathoz. A szenzorhálózatok hardver és szoftver korlátozásainak köszönhető ez a roppant komoly probléma.

#### 3.2 Szenzorhálózatok korlátozásai

A szenzorhálózatok következő jellegzetességei komplikálják biztonsági protokoll tervezését.

- A csomópontok sebezhetőek fizikai elhelyezkedésük alapján, azaz a szenzor csomópontok számos alkalmazásban lehetnek nyilvános vagy ellenséges területen telepítve. Továbbá a nagyszámú csomópontok, melyek telepítve vannak, sejtetik, hogy olcsónak kell lenniük, ami miatt a gyártónak nehéz úgy legyártania, hogy lopás ellen ellenállóak legyenek. Ezek az ellenség által fizikai támadásnak kitett csomópontok.

A legrosszabb esetben egy támadó átveheti az irányítást egy szenzor felett és veszélyeztetheti a titkosító kulcsokat.

- Hiányzik az előzetes ismeret a hálózat konfigurációjának változásairól. Ha egy szenzorhálózat véletlen szórással van telepítve (pl. egy repülővel), a szenzorhálózat protokollja nem tudhatja előre, hogy melyik csomópont van a kommunikációs tartományán belül. Még akkor sem, ha a csomópontok kézzel lettek telepítve. A csomópontok nagy számának bonyolultsága teszi drágává, hogy előre meghatározzuk minden egyes csomópont helyzetét. Ezentúl a biztonsági protokoll nem feltételezheti, hogy melyik csomópontok szomszédosak a hálózatban.

- Korlátozott sávszélesség és átviteli képesség: a tipikus szenzorhálózat platformok nagyon alacsony sávszélességűek Pl.: az UC Berkeley mica platformjának 10 Kbps a sávszélessége, és kb. 30 byte a csomag mérete. Az átvitel megbízhatósága gyakran alacsony, az adatok nagy blokkú kommunikációját elkészíteni pedig meglehetősen drága.

### 3.3 Biztonságos betöltés problémája szenzorhálózatokban

Egy betöltési sémának szenzorhálózatok esetén a következő követelményeket kell teljesítenie:

- A telepített csomópontokhoz ki kell tudni építenie egy biztonságos csomópont-csomópont közötti kommunikációs kapcsolatot.
- Létre kell tudni hoznia egy biztonságos kapcsolatot a már telepített csomópontok és a később, szabályosan telepíteni kívánt csomópontok között
- A jogosulatlan csomópontok nem nyerhetnek belépést a hálózatba
- A sémának olyan előzetes tudás nélkül kell dolgoznia, mely megmondja, hogy melyik csomópontok léptek be egymás kommunikációs tartományába
- A séma számítási és tárolási követelményeinek alacsonynak kell lennie, és a sémának ellenállónak kell lennie a külső forrásból érkező DoS<sup>7</sup> támadásokkal szemben

---

<sup>7</sup> DoS támadás:(Denial of Service attack): Egy adott szolgáltatást az azt biztosító hardver vagy szoftver megbénításával vagy működésének zavarásával elérhetetlenné tevő támadás. Leggyakrabban a korlátozott erőforrások kimerítésével, vagy a kommunikációban résztvevő komponensek működésének zavarásával viszik végbe.

### 3.4 Kiértékelési metrika

A szenzorhálózatoknak számos olyan tulajdonsága van, amely sebezhetővé teszi őket a hagyományos számító berendezésekkel szemben.

- Rugalmasság a csomópont rögzítéssel szemben. Feltételezhető, hogy ha egy támadó végre tud hajtani egy fizikai támadást egy szenzor csomópont ellen, miután telepítették, titkos információkhoz juthat hozzá a memóriából.
- Ellenállás a csomópont másolással szemben. Akár az ellenség is be tud helyezni néhány csomópontot a hálózatba, miután megszerzett néhány titkos információt. Ez egy komoly támadási lehetőség abban az esetben, ha a támadó teli tudja telepíteni a hálózatot a rögzített csomópontok klónjaival olyan mértékben, hogy a szabályosan telepített csomópontok számbelileg kisebbségben legyenek, így a támadó megszerezheti a hálózat teljes irányítását.
- Visszavonás: amikor egy észlelt betolakodó csomópontot el lehet távolítani a rendszerből
- Skálázhatóság: amint növekszik a hálózatban a csomópontok száma, a biztonsági jellemzők csökkennek

Minden betöltési protokoll magába foglalja a következő néhány lépést. Az inicializáló folyamatban hajtódnak végre a szenzor csomópontok inicializálási telepítés előtt. Miután a szenzor csomópontok telepítve lettek, a kulcs beállító folyamat hajtódik végre a csomópontokon, hogy fel lehessen építeni az osztott kulcsokat minden egyes szomszédos csomópont között, mellyel megalapozzák a biztonságos kapcsolatot.

### 3.5 Teljes hálózatra kiterjedő kulcs használata

A kulcs kiosztás legegyszerűbb módja, előre beolvasunk egy hálózatra kiterjedő kulcsot minden csomópontra telepítés előtt. Telepítés után a csomópontok létrehozzák a kommunikációt minden más szomszédos csomóponttal, amelyek szintén rendelkeznek a kiosztott hálózati kulccsal. Ez könnyen elérhető minden kommunikáció kódolásával a szétszórott, teljes hálózatra kiterjedő kulccsal, és egy üzenet hitelesítő kódot is hozzáfűz

a sértetlenség biztosítása érdekében. Az egyetlen, teljes hálózatra kiterjedő kulcs szemléletmódjának tulajdonságai a következők:

- Minimális memória tár szükséges: egyetlen (kriptográfiai) kulcs tárolása szükséges a memóriában
- Nincs utólagos/további protokoll lépésekre szükség: a protokollnak kulcs felfedés, vagy csere végrehajtása nélkül kell működnie
- Ellenállás a DoS típusú támadással szemben: az üzenethitelesítő kódok feladata, hogy megakadályozzák, hogy egy támadó hozzáférhessen a hálózatra kiterjedő teljes kulcshoz. Ebből adódóan az egyik módszer arra, hogy lelassítsák a hálózatot, hogy a támadó üzeneteket küld a csomópontoknak, melyeknek így számos üzenethitelesítő ellenőrzést kell végrehajtaniuk. Azonban, a szimmetrikus kulcsú kriptográfiai rendszerek üzenethitelesítő kódok ellenőrzése nagyon gyors és egy DoS típusú támadás nem elég hatékony, hogy megakadályozza a normális működést.

A teljes hálózatra kiterjedő kulcs használatának egyetlen hátránya, hogy ha csak egyetlen csomópont veszélyeztetett, a teljes hálózat elveszti a biztonságát, mivel a hálózatra kiterjedő kulcs már ismert a támadó számára.

Lehetséges védekezések a támadásokkal szemben: az egyik lehetőség, hogy a csomópontokat úgy helyezik el, hogy a támadók ne férhessenek hozzá. Például: az autópályákon beépítik a szenzorokat az úttestbe.

A védekezés másik módja, hogy nem engedélyezett újabb csomópont hozzákapcsolása a hálózathoz. Ekkor a teljes hálózatra kiterjedő kulccsal a szomszédos csomópontok egyedi kapcsolatkulcsokat hozhatnak létre, mellyel titkosítják a küldendő üzenetet. A dinamikusan létrehozott kapcsolatkulcsok használatának számos előnye van, köztük a következők:

- A kapcsolatkulcs rövid élettartama miatt egy lehallgató támadónak csak korlátozott mennyiségű, ugyanolyan kulccsal rejtjelezett üzenet áll rendelkezésére, ami nehezíti a kulcs megfejtését.

- Ha egy kapcsolatkulcs mégis kompromittálódik, akkor az csak korlátozott mennyiségű üzenet megfejtését teszi lehetővé. Továbbá, megfelelő tervezés esetén, a kapcsolatkulcsok kompromittálódása nem befolyásolja a hosszú élettartamú mesterkulcsok biztonságát.
- A kapcsolatkulcsok alkalmazása függetlenné teszi egymástól a felek közötti különböző kapcsolatokat.

A kapcsolatkulcsokat megőrzi a csomópontok, a teljes hálózatra kiterjedő kulcsot törlik a memóriából. Így minden csomópont csak a szomszédjával tud kommunikálni, ha feltöri a kapcsolatkulcsot, a hálózat még nincs veszélyeztetve. A hátránya ennek a módszernek, hogy ha egy szenzor meghibásodik, vagy lemerül, nem lehet pótolni. Egy lehetséges módja új szenzor beszúrásának a hálózatba, hogy folyamatos ellenőrzést kell végezni, nincs-e támadás indítva valamelyik csomópont ellen. Amennyiben biztonságosnak tűnik a hálózat, be lehet illeszteni a hálózatba egy új csomópontot. Természetesen a folyamatos ellenőrzés olyan mértékben lelassítja a hálózat működését, hogy gyakorlatban ezt a változatot sem alkalmazzák.

### 3.6 Aszimmetrikus kriptográfia használata

A kulcskiosztás kedvelt módszere a legtöbb modern számítógépes rendszerben az aszimmetrikus kriptográfia, mely nyilvános kulcsú módszerként is ismert. A nyilvános kulcsú kriptográfia alapja, hogy a kódoló és a dekódoló transzformáció, illetve az azokat paraméterező nyilvános és titkos kulcsok nem azonosak, sőt egyiket a másikból kiszámítani nagy komplexitású feladat. Ezért a két kulcs közül az egyiket, általában a kódoló kulcsot, nyilvánosságra lehet hozni. Ez azért hasznos, mert ezáltal leegyszerűsödni látszik a kulcscsere probléma, ugyanis a titkos kommunikáció megvalósításához nincsen szükség egy titkos (szimmetrikus) kapcsolatkulcs létrehozására a küldő és a vevő között; elegendő a rejteni kívánt üzenetet a vevő nyilvános kódoló kulcsával titkosítani. Azonban, gondoskodni kell a nyilvános kulcsok hitelességéről, azaz a küldőnek meg kell tudnia győződnie arról, hogy a használni kívánt nyilvános kulcs valóban a vevő nyilvános kulcsa, és nem egy harmadik, feltehetően egy rossz-szándékú félé. A nyilvános kulcsú kriptográfia titkos csatorna helyett hiteles

csatorna létezését követeli meg a vevő és a küldő között, melyen a vevő eljuttathatja nyilvános kulcsát a küldőnek.

Ha a szenzor csomópont hardveresen támogatja az aszimmetrikus kriptográfia működését, akkor ez egy életképes lehetőség a kulcskiosztás megoldására.

A telepítést megelőzően egy master nyilvános és privát ( $k_m$ ,  $k_m^{-1}$ ) kulcspár készül. Ez a kulcspár A csomópont memóriájában lesz letárolva  $k_m$  privát kulccsal együtt és a privát kulcs aláírása A nyilvános kulcsa. Ezután minden A csomóponthoz egy nyilvános/privát kulcspár generálódik. Mihelyt minden csomópont inicializálódott ebben a módszerben, készen állnak a telepítésre.

Amint a nyilvános kulcsa el lett fogadva, egy szimmetrikus kapcsolatkulcs készül el és elküldi ezt, mely a nyilvános kulcs által van titkosítva. Az éppen aktuális kulcs felvételével a kulcs készítése befejeződött, és a két csomópont kommunikálhat a szimmetrikus kapcsolatkulcs használatával.

Ezen megközelítés tulajdonságai a következők:

- Teljesen rugalmas a megtámadott csomópontokkal szemben  
Csomópontok nagy számú elfogása nem leplezhet le fontos információkat a hálózatról, mivel ezeknek a csomópontoknak nincs ismerete a többi titkos kapcsolatkulcsról amellet az egy mellett, amelyet aktuálisan használnak.
- Lehetőség van visszavonni ismert veszélyeztetett kulcspárokat

Amennyiben nagyon sok kulcspár lett visszavonva, lehetőség van arra, hogy új privát kulcsot osszunk szét, melyet a régi kulccsal titkosítunk. Ezzel elérhetjük, hogy a még érvényesen működő csomópontoktól a már veszélyeztetett vagy visszavont csomópontok semmilyen információhoz ne jussanak hozzá, hiszen az új kulcsokkal már nem rendelkeznek.

Az aszimmetrikus kriptográfia hátránya

Minden ismert aszimmetrikus kriptográfiai séma magas számítási igényű műveleteket foglal magába. Ugyanakkor a szenzorok processzora rendkívül korlátozott,

gyakran hiányzik belőle az egész szorzási utasítás is. Ahhoz, hogy az aszimmetrikus titkosítás végrehajtható legyen egy szenzoron, szükséges mindegyiknek implementálnia egy dedikált kriptográfiai hardvert a szenzorokon, ennek következtében viszont növekszik a szenzorok ára.

A másik lehetőség, hogy matematikai függvényeket kódolnak a szoftverbe, ekkor viszont a mérésekhez szükséges memória mennyisége csökken.

Kapcsolatteremtés közben a szenzoroknak kötelezően ellenőrizniük kell az aláírás helyességét is. Jelentős támadási felületet biztosít a támadók számára, hiszen, ha folyamatosan illegális üzeneteket küldenek egy csomópontnak, az aláírás ellenőrzése miatt annyira lelassulhat, hogy nem képes ellátni a feladatát, kiesik a hálózathoz.

Természetesen, igazán komoly kárt tenni, illetve fontos információkhoz hozzájutni a támadó csak a kulcs létrehozás fázisa alatt tud, ekkor a legsebezhetőbb a hálózat.

## 4. Az AES blokkrejtjelező

[2] A Rijndael szimmetrikus kulcsú blokkrejtjelező algoritmus a DES algoritmust lecserélő új rejtjelező szabvány az USA-ban (Advanced Encryption Standard AES). Feladat – továbbra is – a bizalmas kereskedelmi információk rejtjelezéssel történő védelme. Számos versengő algoritmus közül a Rijndael lett a győztes, mivel több követelményt kiegyensúlyozottan valósít meg. Ezen követelmények legfontosabbika természetesen a biztonság, de nagy súllyal számított a hatékony implementálhatóság (sebesség) különböző platformokon, valamint a blokkméretek (üzenet, kulcs) variálhatósága, azaz a flexibilitás.

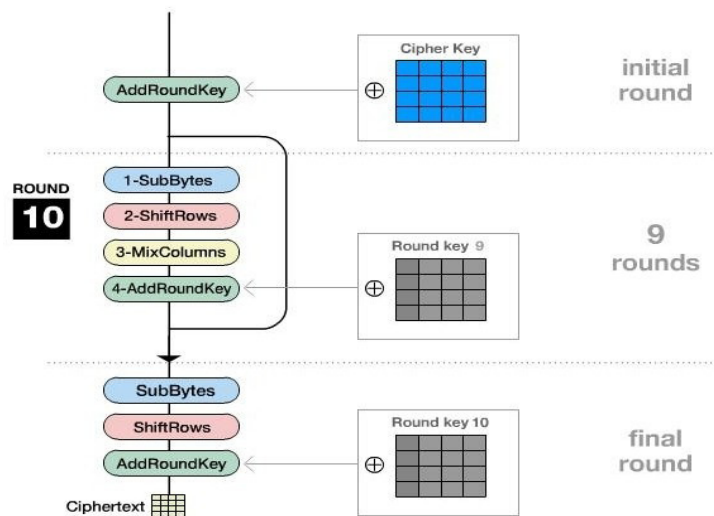
A biztonság kritérium tekintetében számos versenytársa lényegében azonos minőséget nyújtott; a Rijndael előnye az, hogy ezt a biztonságot hatékonyabban tudja megvalósítani. A biztonság vonatkozásában fontos bírálati szempont volt az algoritmus részletekben is világos, áttekinthető struktúrája, s bizonyíthatósága. Az AES tiszta algebrai struktúrája  $GF(2^8)$  aritmetikát alkalmaz, bájt a legkisebb kezelt egység. Az üzenet és kulcs bemenetpár és a rejtjeles üzenet kimenet között  $GF(2^8)$  feletti nemlineáris egyenletrendszer írható fel.

Az AES iteratív blokkrejtjelező, amelynek rétegfelépítése követi a shannoni helyettesítéses-permutációs elveket, így az iterációs rétegekben (round) megtalálható a bájt elemekkel dolgozó nemlineáris S-boxokból álló és a permutációs alréteg is. Az alrétegek külön-külön invertálhatók.

Az üzenet (rejtett üzenet) és a kulcs  $N_b$ , illetve  $N_k$  számú 32 bites (4 bájt) szóból áll. Ennek megfelelően az AES blokkméret választéka 128, 192, illetve 256 bit. Az iterációs rétegek számára  $N_r$  jelölést használva,  $N_r = 10$ , ha  $N_b = N_k = 4$ ;  $N_r = 12$ , ha  $N_b$  és  $N_k$  4 és 6 közötti (de nem 4 mindegyikük); egyéb esetekben  $N_r = 14$ . (Összehasonlításképpen a DES paraméterei, az adott jelölésben:  $N_b = 2$ ,  $N_k = 2$  (56 bit),  $N_r = 16$ .)

Egy iterációs réteg (round) szerkezete az alábbi C kód szerint:

```
Round (State, Roundkey) {
    SubBytes(State);
    ShiftRow(State);
    MixColumn(State);
    AddRoundKey(State); }
```



4. ábra: Titkosítás folyamata

A SubBytes bájt helyettesítő transzformáció alréteg nyújtja a nemlinearitást (S-boxok alrétege). Az alréteg bemeneti  $4 \cdot N_b$  bájtjára bájtonként alkalmazzuk az  $S: GF(2^8) \rightarrow GF(2^8)$  helyettesítő transzformációt:

$$S(x) = Bx^{-1} + b,$$

ahol  $B$  egy  $8 \times 8$  dimenziós invertálható bináris mátrix,  $b$  egy 8 bites vektor,  $x^{-1}$  az inverze (bináris koordinátákban). Így az  $S$  helyettesítő transzformáció (S-box) a nemlinearitását ezen utóbbi invertálásból nyeri. Az  $S$  transzformáció egyszerűen invertálható:  $S^{-1}(y) = (B^{-1}(y-b))^{-1}$ .



$d_{0,j}$  polinomot és egy rögzített  $E(x) = e_3x^3 + e_2x^2 + e_1x + e_0$ , ( $e_3 = 0x03$ ,  $e_2 = 0x01$ ,  $e_1 = 0x01$ ,  $e_0 = 0x02$ ) polinomot szorozzuk össze mod  $(x^4 + 1)$ :

$$D_j(x)E(x) \pmod{x^4 + 1},$$

azaz elvégezzük a szokásos polinomszorzást, majd a szorzat 4,5,6 fokszámú tagjait redukáljuk, rendre 0, 1, 2 fokszámúvá. Ez a moduláris szorzás invertálható:  $x^4 + 1$  nem irreducibilis polinom  $GF(2^8)$  felett, de  $E(x)$  úgy lett választva, hogy létezzen inverze, azaz létezik  $E^{-1}(x)$  polinom, amelyre  $E^{-1}(x)E(x) = 1 \pmod{x^4 + 1}$ .

Az AddRoundKey transzformáció-alréteg végzi a rétegkulcs lineáris bevitelét, a rétegkulcsot bitenként mod 2 hozzáadva a transzformáció bemeneti blokkja megfelelő bitjeihez.

A kulcsütemezési algoritmus (KeyExpansion) bemenete az AES rejtjelező  $K = K_0, \dots, K_{N_k-1}$  kulcsa, kimenete a  $W = (W_0, \dots, W_{Nb(Nr+1)-1})$  kiterjesztett kulcs, ahol  $W_i$ ,  $K_i$  32 bites szavak. A kulcsütemezési algoritmus a SubByte és a RotByte transzformációkat alkalmazza, ahol

$$SubByte(a,b,c,d) = (S(a), S(b), S(c), S(d))$$

$$RotByte(a,b,c,d) = (b,c,d,a)$$

A kulcsütemezési algoritmus két verziója közül az első az  $N_k \leq 6$ , a második az  $N_k > 6$  esetben kerül felhasználásra. Mindegyiknek 2 fázisa van: az inicializálás, és az ezt követő kiterjesztési fázis.

A kódolandó üzenet és az első iterációs réteg között egy külön AddRoundKey kulcsbeviteli művelet is szerepel.

A Rijndael-rejtjelező szabványként való elfogadása óta, a kriptóanalízis szakértők egyik értékes célpontja, ugyanúgy, ahogy közel harminc évvel ezelőtt a DES rejtjelező volt. Az algoritmus ellenállt a differenciális és lineáris analízis kísérleteknek.

## 5. TinyOS operációs rendszer és Tossim szimulátor

### 5.1 TinyOS operációs rendszer lehetőségei, használata

[8] A TinyOS operációs rendszert a kaliforniai Berkeley egyetemen fejlesztették ki, speciálisan a szenzorhálózati alkalmazások igényeihez igazítva. Egy ingyenesen elérhető, nyílt forráskódú komponens alapú operációs rendszer és platform wireless szenzorhálózatokhoz. Szorosan kötődik hozzá a nesC programnyelv, mellyel a teljes beágyazott operációs rendszer készült. A nesC nyelv segítségével lehet alkalmazásokat írni az operációs rendszer alá, amely irányítja a feladatok és folyamatok együttműködését. A nesC programnyelv a közismert C programozási nyelv egy komponens-orientált, eseményvezérelt szemléletű, bővített változata, melyet a több szempontból is kis kapacitású szenzorok működtetéséhez fejlesztettek ki. A TinyOS a Kaliforniai Berkeley Egyetem és az Intel Kutatási részlege közötti együttműködésből indult, és azóta nötte ki magát egy nemzetközi konzorciummá, a TinyOS Szövetséggé.

[13] A dolgozatom egyik céljának tűztem ki a TinyOS és nesC nyelv bemutatását. Az általam írt SensorWatcher alkalmazásból kivett kódrészletekkel is szeretném illusztrálni a TinyOS rendszer bizonyos részeit.

[7] A TinyOS a szabványos C nyelvből jól ismert int, long vagy char típusok helyett részletesebb típuselnevezést használ. A különböző típusok a platform specifikusság miatt alakultak ki. Pl. egy micaZ, vagy Telos szenzor esetében az int típus 16 biten tárolódik, míg IntelMote2-es típusú csomópont esetén 32 biten. Ennek megfelelően különböző elnevezéseket használ:

|          | 8 bit   | 16 bit   | 32 bit   | 64 bit   |
|----------|---------|----------|----------|----------|
| Signed   | int8_t  | int16_t  | int32_t  | int64_t  |
| Unsigned | uint8_t | uint16_t | uint32_t | uint64_t |

A nesC nyelven írt programok komponenseknek egy halmaza, melyek kapcsolatban állnak egymással. A komponensek megvalósítanak valamilyen feladatot, a többi

komponens felé szolgáltatásokat biztosítanak, és a feladatuk végrehajtásához maguk is igénybe vehetnek más komponenseket. Ez az architektúra teszi lehetővé a szolgáltatás szétválasztásának és cserélhetőségének lehetőségét. A *components* kulcsszóval kezdődő sorok határozzák meg a komponensek halmazát. A *SensorWatcherAppC* komponens a *SensorWatherC*, *MainC*, *TimerC*, *ActiveMessagesC*, *AMSenderC* és *AMReceiverC* komponensből áll.

A TinyOS egyik szolgáltatásai közé tartozik, hogy a nesdoc segítségével automatikusan dokumentáció generálatható a forráskódból. A nesdoc megmutatja a struktúráját és összetételét a konfigurációnak. A dokumentáció generálása *make platform docs* alkalmazás futtatásával történik.

A komponensek parancsok ( *command* ) és események ( *event* ) segítségével kommunikálnak egymással. A parancs és az esemény olyan függvények, melyeket az egyik komponens implementál, és ezt más komponens meghívhatja. A különbség ott jelentkezik, hogy ha ezeket egy interfészbe fogjuk össze, akkor különböző irányú kommunikációt tesznek lehetővé. A komponensek általában interfészeket nyújtanak (*provide* kulcsszó) vagy használnak (*uses* kulcsszó) az objektum-orientált szemléletmód jegyében. A komponensek ezeken keresztül kapcsolódnak össze. Ha egy komponens szolgáltat valamilyen interfészt, akkor az megvalósítja az interfészben definiált parancsokat, valamint a *signal* kulcsszóval kiválthatja az eseményeket. Ha egy másik komponens használ valamilyen interfészt, akkor az azt jelenti, hogy a *call* kulcsszó segítségével meghívhatja a benne levő parancsokat, cserébe viszont kezelnie kell a benne definiált eseményeket.

Pl.: A *SensorWatcherC.nc* fájl a következő sorokat tartalmazza:

```
uses interface Boot;
```

Ehhez az implementációban a következő eseményt kötelezően le kell kezelni:

```
event void Boot.booted() {  
    call AMControl.start();  
    call Timer0.startPeriodic(1000);  
}
```

A komponensek tehát nyújtanak és használnak interfészeket, de ehhez meg kell adni, hogy hogyan kapcsolódnak egymáshoz. Ez az összeköttetés a *configuration* kulcsszóval kezdődő konfigurációkban történik meg. Ezeket a fájlokat konfigurációs fájloknak nevezzük.

*configuration SensorWatcherAppC {}*

A konfigurációk maguk is komponensek, tehát úgy néznek ki, hogy maguk is nyújthatnak és használhatnak interfészeket, de felépítésük olyan, hogy belül nem imperatív programkódot tartalmaznak, hanem más komponenseket, melyeket a megfelelő interfészeik mentén összekötnek.

Egy komponens deklaráció bármilyen állapota privát: nincs másik komponens, mely hivatkozhatna rá, vagy közvetlenül hozzáférhetne. Egy hierarchikus rendszer kialakítását teszik lehetővé. Az aktuális konfiguráció kapcsos zárójelek között az *implementation* kulcsszót követően van implementálva.

A komponensek másik típusa az úgynevezett modulok(*module*). A modulok egyszerű belső szerkezetűek, a hierarchikus felépítés alját képezik, mivel ezek a modulok tartalmazzák a parancsok és eseménykezelők konkrét implementációit. A függvények megvalósítása C nyelven történhet néhány kikötéstől eltekintve: használhatóak a következő kulcsszavak: *call*, *signal* és *post*, valamint nem lehet használni memória-allokációt és függvénymutatókat. Modulok deklarálhatnak állapot változókat. A modulok függvényeken kívül még taszkokat (*task*) is tartalmazhatnak. A taszk egy speciális függvény, a *task* kulcsszóval deklarálható, visszatérési értéke és argumentuma nincs. A TinyOS ütemezője ilyen taszkokat futtat, egymás után FIFO ütemezéssel, nem preemptív módon. A *post* kulcsszóval indítható el egy taszk, és végrehajtásra kerül, amint rá kerül a sor.

*task void taskname(){} - egy taszk deklarációjának szintaxisa*

*post taskname() - egy taszk elküldése a futtatási sorba*

A *post* művelet visszatérési értéke *error\_t*, aminek az értéke lehet *SUCCESS* vagy *FAIL*.

[8] Amikor egy szinkronizációs kód elkezd futni, ez a kód nem mond le a processzorról egy másik kód miatt, amíg be nem fejezte a futását. Ez az egyszerű mechanizmus megengedi a TinyOS ütemezőjének, hogy minimalizálja a RAM

fogyasztását. Azonban, ez azt jelenti, hogyha egy szinkronizációs kód sokáig fut, ez megakadályozza másik szinkronizációs kód futását, mely ellentétesen hathat a rendszer hatékonyságára. Egy komponensnek lehetőséget kell biztosítania arra, hogy nagyobb számítási igényű feladat esetén kisebb részekre bontható legyen, melyek bizonyos időközönként futtathatóak. A TinyOS fejlesztői a következő megoldást választották az ütemezés problémájára. A rendszer prioritás nélküli, nem preemptív ütemezőt használ, mely valós idejű rendszerek készítésére nem a legalkalmasabb. Azonban a rádió keresztüli kommunikáció bizonytalansága miatt nem várhatóak el szigorú követelmények az időzítésre. A preemptív ütemezőnél jelentkező szinkronizációs problémákat viszont ez a megoldás kiküszöböli, mert minden taszk egymástól teljesen függetlenül fut, nem szakítják meg egymás futását. A taszkok képessé teszik a komponenseket arra, hogy általános célú háttérfeldolgozással működjenek egy alkalmazásban.

TinyOS számos interfészt szolgáltat kommunikációt megalapozó szolgáltatásokhoz és számos komponens, melyek ezen interfészek implementációi. Minden interfész és komponens egy közös absztrakt üzenet buffert használ, melyet *message\_t*-nek hívnak és (C nyelv struktúrájához hasonlóan) nesC struktúraként van implementálva.

Alap kommunikációs interfészek:

*Packet* – alap hozzáférést szolgáltat a *message\_t* absztrakt adattípushoz.

*Send* – alap üzenet küldő interfészt szolgáltat. Parancsokat ad üzenet küldéshez és várakozó üzenetek törléséhez. Egy eseményt biztosít annak jelzésére, hogy az üzenet sikeresen ki lett-e küldve. Olyan kényelmi funkciókat is szolgáltat, melyek megadják az üzenet maximális hasznos súlyát valamint egy mutatót az üzenet hasznos területére.

*Receive* – alap szolgáltatást nyújt az üzenetek fogadására.

*PacketAcknowledgements* – egy mechanizmust szolgáltat a csomag alapú visszajelzések fogadásához.

*RadioTimeStamping* – időbélyegzés-információt biztosít rádió közvetítés és fogadás számára.

Az üzenet küldéshez definiálni kell a küldendő üzenet formátumát, a C nyelvben lévő struktúrához hasonló hozható létre, ahol a struktúra egy vagy több, esetleg különböző típusú változó együttese, mely önálló névvel rendelkezik. A definiált struktúra a szenzor csomópont azonosítóját, és a mért értéket küldi el:

```
typedef nx_struct SensorWatcherRadioMsg {
    nx_uint16_t nodeid;
    nx_uint16_t counter;
} SensorWatcherRadioMsg;
```

nesC nyelv különlegessége a *nx\_* prefixum, melynek jelentése, hogy a *struct* és a *uint16\_t* külső típusok. A külső típusoknak minden platformon ugyanolyan a reprezentációja. Az üzenet küldéshez szükséges lépések: először is szükséges azonosítani azokat az interfészeket, és komponenseket, melyek hozzáférést biztosítanak a rádióhoz, és engedélyezik a *message\_t* típus változtatását. Definiálni kell azokat a használható utasításokat, változókat, melyekre szükség lehet, inicializálni kell néhány start/stop kódot az interfészekhez és komponensekhez. Az *AMSend* interfészt kell használni a csomag küldéséhez, és *AMPacket* interfészt a *message\_t* absztrakt adattípus elfogadásához. A *SplitControl* interfész egy általános megoldás a starting és stopping komponensek használatára.

Az *AMSender* mintájára egy *AMReceiver* komponensre is szükség van, ill. az ehhez tartozó függvények, interfészek felvételére, melynek célja, hogy egy szenzor ne csak küldeni, hanem fogadni is tudjon üzeneteket.

Egy példán keresztül szeretném megmutatni, hogy ezeket a függvényeket használva milyen adatcsomagok mennek át a szenzorok között.

*Destination address: (2 bytes)*

- *Message length: (1 byte)*
- *Group ID (1 byte)*
- *Active Message handler type (1 byte)*
- *Payload (up to 28 bytes):*
  - *source mote ID (2 bytes)*
  - *sample counter (2 bytes)*

| dest addr | msg len | groupID | handlerID | source addr | counter |
|-----------|---------|---------|-----------|-------------|---------|
| ff ff     | 04      | 22      | 06        | 00 02       | 00 0B   |

## 5.2 Tossim szimulátor bemutatása

[8] Hardver eszközök hiányában a TinyOS biztosít a fejlesztők számára egy Tossim nevű szimulátort, ahol bizonyos korlátok között a fejlesztők ki tudják próbálni a szenzorokra telepítendő alkalmazásokat. Tekintettel arra, hogy a dolgozat keletkezése alatt nem állt rendelkezésemre ilyen eszköz, én magam is ezt a szimulátort használtam. A TinyOS operációs rendszer esetéhez hasonlóan szeretném a szimulátor néhány fontosabb elemét, illetve működését bemutatni.

A szimulátor komponensek cserélgetésével és szimulációk implementálásával működik. Rugalmasan dönti el, hogy melyik komponenst kell cserélni. A Tossim tud helyettesíteni egy csomag szintű kommunikációs komponenst egy csomag szintű szimulációra, vagy helyettesíthet egy alacsony szintű rádióchip komponenst egy a kód futtatásához pontosabb szimulációval. Tossim egy diszkrét esemény szimulátor. Futás közben az eseményeket egy esemény sorba helyezi és érkezési sorrend szerint hajtja végre őket. A szimuláció szintjétől függően a szimulált események ábrázolhatnak hardvermegszakítást, vagy magas szintű rendszer-eseményeket. A Tossim 2 programozói interfészt szolgáltat: a Python és a C++.

### 5.2.1 Tossim szimulátor fordítása

[8] A Tossim egy TinyOS könyvtár. A forráskód a *tos/lib/tossim* könyvtárban található. Minden TinyOS forrás könyvtárban opcionálisan lehet egy *sim* alkönyvtár, mely a csomagoknak a szimulációs implementációit tartalmazza. Például a *SensorWatcherApp* alkalmazás fordítása esetén a következő utasításokat kell kiadnunk a fordításhoz:

```
$ cd apps/SensorWatcherApp
```

```
$ make micaz sim
```

Ekkor a *SensorWatcherApp* könyvtárban létrejön a szimulációhoz egy *sim* alkönyvtár. Jelen esetben az egyetlen platform, melyen a szimuláció működik a *micaz*.

A Tossim szolgáltatásai közé tartozik a változók állandó tárolása. A Tossim szimulátor egy XML fájlban tárolja a változók minden tulajdonságát. Ezzel egyfajta ellenőrzést lehet elérni, folyamatosan nyomon követhető a változók értékeinek változása. Ehhez egy Python csomagot kell importálni, mely felelős az XML fájl kezelésért:

```
from tinyos.tossim.TossimApp import *  
n = NescApp( "Unknown App", "app.xml")
```

A *NescApp()* függvénynek 2 opcionális argumentuma van, az első a betöltött alkalmazás neve, a második pedig annak az XML fájlnek a neve, melyben az információ mentődik.

A *NescApp()* által tárolt változók listáját a *variables* mezőn meghívott *variables()* függvény segítségével futási időben is megkaphatjuk:

```
vars = n.variables.variables()
```

Ezzel engedélyezhető a szimulátornak a változók figyelése, melynek következménye, hogy a szimulátor ezután már ismerni fogja minden változó nevét méretét és típusát.

```
t = Tossim(vars)
```

Ezek az információk adnak lehetőséget arra, hogy a szimulátor a C-s változókat helyesen konvertálja át Python változókra.

Tossim szolgáltatásai közé tartozik, hogy dinamikusan Python csomagok adhatók a hálózathoz. A csomagok érkezésének ideje bármikorra időzíthető. Ha a megadott idő már elmúlt, a csomagok azonnali betöltésre kerülnek.

### 5.2.2 Tossim szimulátor és a Python nyelv használata

[3, 4] A szimuláció Python fejlesztőkörnyezetben is futtatható vagy egy szkript segítségével, vagy interaktív módon. Az első lépés egy Tossim objektum létrehozása a Tossim csomag importálása után:

```
>>> from Tossim import *  
>>> t = Tossim([])
```

A kapcsos zárójelek között opcionális argumentum adható meg. A *runNextEvent()* függvény segítségével futtatható a Tossim szimulátor.

```
>>> t.runNextEvent()
```

```
>>>0
```

A Tossim visszatérési értéke azt jelenti, hogy mennyi futásra várakozó esemény van a sorban. A szimulátornak van néhány hasznos függvénye. Pythonban a *dir()* függvény segítségével pl. láthatóak egy objektum tulajdonságai, függvényei.

```
>>>dir(t)
```

```
['_class_', '__del__', '__delattr__', '__dict__', '__doc__', '__getattr__', '__getattribute__', '__hash__', '__init__', '__module__',  
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__str__', '__swig_getmethods__', '__swig_setmethods__',  
'__weakref__', 'addChannel', 'currentNode', 'getNode', 'init', 'mac', 'newPacket', 'radio', 'removeChannel', 'runNextEvent',  
'setCurrentNode', 'setTime', 'this', 'thisown', 'ticksPerSecond', 'time', 'timeStr']
```

A leggyakrabban használt segéd függvények:

*currentNode()*: visszaadja az aktuális csomópont ID értékét

*getNode(id)*: visszaad egy objektumot, mely egy speciális csomópontot reprezentál

*runNextEvent()*: futtat egy szimulált eseményt

*time()*: visszaadja szimuláció aktuális idejét *long* integer formátumban

*timeStr()*: visszaadja az aktuális idő *String* formátumban

*init()*: Tossim inicializálása

*mac()*: visszaad egy objektumot, mely a közeg hozzáférési réteget reprezentálja

*radio()*: visszaad egy objektumot, mely a rádióvevő modellt reprezentálja

*addChannel(ch, output)*: hozzáadja *ch*-t kimeneti csatornaként

*removeChannel(ch, output)*: eltávolítja a *ch* kimeneti csatornát

*ticksPerSecond()*: visszaadja, mennyi szimulációs beosztás van egy szimulációs másodpercben

Tossim rendelkezik egy hibakeresési kimeneti rendszerrel, amit *dbg*-nek hívnak:

*dbg*: megjelenít egy debug állapotot a csomópont ID-val az elején

*dbg\_clear*: megjelenít egy debug állapotot, amely nem tartalmazza a csomópontok ID értékét.

*dbgerror*, *dbgerror\_clear*: hiba állapot megjelenítése csomópont ID-val, ill. csomópont ID nélkül.

A csomagok közül az egyik legfontosabb a *sys* csomag, importálásával lehetővé válik, hogy egy csatorna kimenete a standard kimenetre íródjon, vagy egy fájlba:

```
>>> import sys
>>> t.addChannel("Boot", sys.stdout)
```

### 5.2.3 Tossim hálózati beállításai

[8] A Tossim indításakor a csomópontok még nem tudnak kommunikálni egymással. Azon célból, hogy hálózati viselkedés szimulálható legyen, definiálni kell egy hálózati topológiát. Belül, a Tossim strukturált rendszer, így könnyű megváltoztatni a rádió szimulációt. Az általános Tossim rádió-modell jel intenzitás alapú. Meghatározható a zajszint és a fogadó intenzitás is. Természetesen egy Python *Radio* objektummal ellenőrizhető a rádió szimuláció:

```
>>> from TOSSIM import *
>>> t = Tossim([])
>>> r = t.radio()
>>> dir(r)
['__class__', '__del__', '__delattr__', '__dict__', '__doc__',
 '__getattr__', '__getattribute__', '__hash__', '__init__',
 '__module__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__setattr__', '__str__', '__swig_getmethods__',
 '__swig_setmethods__', '__weakref__', 'add', 'connected',
 'gain', 'remove', 'setNoise', 'this', 'thisown',
]
```

Még néhány fontosabb függvény:

*add(src, dest, gain)*: kapcsolat az *src* forrástól, a *dest* célig *gain* erősséggel. Amikor *src* továbbít, *dest* fogadja a csomagot *gain* erősség értékkel csillapítva.

*connected(src, dest)*: visszatérési érték megadja, hogy van-e kapcsolat a *src* forrás és *dest* cél között

*gain(src, dest)*: visszatérési értéke az *src* forrás és *dest* cél közötti kapcsolat erősségi értéke

*threshold()*: visszatérési értéke a CCA határérték

*setThreshold(val)*: CCA határérték beállítása decibelben. Alapértelmezett: -72dBm

A Tossim rádió-modell alap értékei a CC2420-as rádión alapulnak, melyet a micaZ-ben, a telos családban, és az imote2-ben is használnak. Tossim képes szimulálni

az RF zajt, ill. vételi zavart tud okozni csomópontok lehallgatása esetén, melyhez egy Closest Pattern Matching (CPM) algoritmust használ. Bemenetként vesz egy zajnyomot, és generál egy statisztikai modellt ebből. El tudja fogni az interferencia változásokat, és más összefüggő jelenségeket, így ez javítja az RF szimulálás minőségét. A CPM konfigurálásához meg kell adni egy zajnyomot, melyhez az *addNoiseTraceReading()* függvényt kell meghívni egy csomópont objektumon. Egy fájlban is megadható egy beolvasott zaj sorozat, soronként egy érték. A szenzoroknak a zajszintje általában -97dB körül van. A *Radio* objektum csak a fizikai réteggel foglalkozik. A MAC objektum az adatkapcsolati réteggel, csomag hosszával, és a rádió sávszélességével foglalkozik. Az általános Tossim MAC objektum egy CSMA protokollt biztosít.

```
>>>mac = t.mac()
```

[1] Mivel a rádió összekapcsolhatósági gráf egy szkripttel megadható, a topológiát könnyen lehet tárolni egy fájlban, mely bármikor betölthető. A példában egy gráf kapcsolatai láthatóak, soronként tárolva 3 értékkel, az első a forrás, aztán a cél és a csillapítás értéke. Egy részlet a alkalmazásban szereplő *topologia.txt* fájlból, mely a szenzorok topológiáját tartalmazza:

|         |         |           |          |          |
|---------|---------|-----------|----------|----------|
| 1 2 -54 | 4 5 -54 | 8 11 -60  | 3 6 -62  | 7 10 -60 |
| 2 1 -55 | 5 4 -55 | 11 8 -59  | 6 3 -58  | 10 7 -60 |
| 1 3 -54 | 5 8 -60 | 10 11 -54 | 4 7 -59  | 6 5 -55  |
| 3 1 -54 | 8 5 -58 | 11 10 -55 | 7 4 -55  | 7 8 -55  |
| 1 5 -60 | 8 9 -54 | 11 12 -55 | 6 9 -61  | 4 2 -61  |
| 5 1 -55 | 9 8 -55 | 12 11 -54 | 9 6 -60  | 12 9 -56 |
| 5 6 -54 | 8 7 -54 | 2 4 -62   | 9 12 -55 |          |

## 6. A SensorWatcher alkalmazás bemutatása

### 6.1 Az alkalmazásról

Diplomamunkám célja: egy szenzorhálózat modellezése konkrét példán keresztül. A szenzorhálózat tartalmaz egy központi (szerver) számítógépet és 12 db szenzort. A hálózatot gráf segítségével modellezem, ahol van egy kitüntetett szerepű csomópont, mely a központi számítógép. A csomópontok 1-1 szenzort jelképeznek.

A szenzorok feladata: adatok gyűjtése, majd egy jól meghatározott módszerrel titkosítása és elküldése a központi számítógépnek. Ebben a modellben a titkosított információ nem azonnal a központi számítógéphez érkezik, hanem a szenzorokon keresztül jut el az információ. A problémát az okozza, hogy a modellben a szenzorok sugárzása kis hatótávolságú, csak néhány másik szenzorral képesek kommunikálni.

A központi számítógép feladat: a kulcskiosztás, kulcsgenerálás, fogadni az információkat, illetve a titkosított információk dekódolása. Feladatai közé tartozik még a hálózat figyelése, karbantartása is.

A könnyebb modellezés érdekében egy konkrét példán keresztül mutatnám be a szenzorhálózat működését. A szenzorhálózatom az Áramszolgáltató Műveknek szolgáltató információt.

Működése: 1 központi számítógépből és 12 db szenzorból áll a hálózat.

A szenzorok feladata, hogy megmérjék a villanyóra pillanatnyi állását. Az inicializáláskor kapott privát kulcs segítségével 128 bites AES kódolást használva kódolják az üzenetet és a tulajdonos nevét hitelesítésnek.

A központi számítógép megkapja a titkosított információkat, majd dekódolja, ellenőrzi a helyességét a kapott eredménynek. Valóban jó információ érkezett-e, nem sérültek-e az adatok. Feladatai közé tartozik a privát kulcs kiosztása inicializálás esetén illetve a tárolása.

A hálózat figyelése: áramszünet után a hálózat újra aktiválása, illetve jelezni a szakembereknek, ha egy szenzortól nem érkezik információ, mely történhet meghibásodás vagy lopás miatt.

A program: indulása után egy grafikus felület jelenik meg, mely 3 egymás alatt elhelyezkedő ablakra osztja a monitort. Az első ablakban az érkező üzenetek eredeti formájukban jelennek meg, a második ablakban az üzenetek dekódolt alakja jelenik meg, míg az utolsó ablakban az elvégzett műveletek logja látható, mellyel ellenőrizhető a hálózat megfelelő működése.

## 6.2 Szolgáltatás orientált programozás

Tekintettel arra, hogy a mai programfejlesztői világban a szolgáltatás orientáció felé haladunk, amely épp napjainkban alakul ki, úgy döntöttem, hogy megpróbálom én is ebből a szempontból közelíteni a problémát. Ezt úgy próbálom elérni, hogy a szimulátor valóban úgy viselkedjen, mintha valódi rendszer lenne a háttérben, és az elsődleges célja a felhasználók kiszolgálása. Több felhasználó használat, biztos működésű felhasználói felület, súgó, stb..

Programozás technikai szempontból négy elemet emelhetünk ki: Adatbázis kezelés, párhuzamos programozás a szenzorok szimulálásához melyet jelen esetben a Tossim szimulátor végez, felület-grafika, és nem utolsó sorban titkosítás. Ezeknek megfelelően a programot is 4 különböző csomagra lehet felbontani, mely elősegíti később az újrafelhasználást.

## 6.3 Fejlesztő környezet

A TinyOS rendszer Linux/Unix vagy cygwin környezetben használható, így Linux-os, Windows-os és MacOS X-es operációs rendszerre is telepíthető. Telepítés után kialakul egy könyvtárszerkezet, amiben az operációs rendszer forrás állományai és a fordításhoz szükséges segédprogramok valamint konfigurációs állományok találhatók. A diplomamunkám írásának kezdete óta a rendszer telepítése, felépítése sokat változott, eleinte problémát okozott a rendszer telepítése.

Van lehetőség külön csomagokból történő telepítésre mind Linux-os, mind Windows-os cygwin környezetben, illetve külön operációs rendszer telepítésére, mely már tartalmazza a TinyOS rendszer telepített változatát. A fejlesztéshez a Stanfordi Egyetem által készített XubuntuOS rendszerét választottam, melynek tartalma:

- Xubuntu Feisty Fawn (7.04) operációs rendszer
- Tinyos-1.x és Tinyos-2.0.2 verziójú rendszer

[16] A fejlesztés során Eclipse fejlesztőkörnyezetet is használtam TinyOS plugin és PyDev plugin segítségével a nesc és python nyelvű programrészek fejlesztéséhez.

## 6.4 Titkosítás

A diplomamunka szempontjából a legfontosabb része a programnak. A már említett AES blokkrejtjelező titkosítási módszer használatára lesz lehetőség. Egy szenzor titkosítja az adott ügyfél azonosítót (név, esetleg cím tartalommal), a mért adatot, ill. milyen műveletet hajtott végre a szenzor. Természetesen a szenzor azonosítóját használjuk fel a titkosításhoz, aláíráshoz szükséges. Miután megérkezik a titkosított üzenet a szerverhez, megjelenik minden adat a titkosításról, megtekinthetjük a kódolt, ill. dekódolt üzenetet.

[11] A CC2420 típusú chip széles körben elterjedt micaZ szenzorokból szerelt drótnélküli szenzor hálózatokban, illetve más drótnélküli adó-vevőkben is. A CC2420 chipet IEEE 802.15.4 MAC hardver biztonsági műveletei jellemzik, mely magába foglal egy független AES titkosítást is, mellyel egy 128 bites eredeti szöveg 128 bites rejtjelezett szöveggé titkosítható. A hardveres AES kódoló sebessége sokkal gyorsabb, mint egy szoftveres implementációé, ezért választottam én is ennek a módszernek az alkalmazását. A szenzorokon sikeresen teljesítenek a CC2420 független AES kódoló funkciói, miközben nagy kényelmet biztosítanak a csomópontok közötti kölcsönös hitelesítéshez, és a blokk-titkosítási alapú *hash* függvények tervezésében.

A titkosításhoz az *Encryption* interfészt kell alkalmazni, mely 3 utasítást és 2 eseményt tartalmaz:

```
interface Encrypt {  
    command error_t setKey ( uint8_t * key );
```

```

        event void setKeyDone ( uint8_t * key );
        command error_t clrKey ( uint8_t * key );
        command error_t putPlain ( uint8_t * plaintext, uint8_t *
ciphertext );
        event void getCipher (uint8_t * plaintext, uint8_t * ciphertext);
    }

```

A *setKey(key)* utasítással lehet tárolni a kulcsot a CC2420 egy speciális RAM területén. A kulcsnak 128 bitesnek (16 byte) kell lennie. Ha az AES függvény nincs használatban és a kulcs tárolás folyamata elkezdődik, *SUCCESS* a visszatérési értéke. Viszont, ha az AES függvény használatban van, *EBUSY* a visszatérési értéke. A kulcs tárolás folyamatának végét a *setKeyDone(key)* esemény jelzi. A *putPlain(plaintext, ciphertext)* parancsot kell használni az eredeti szöveg tartalmának a CC2420 egy meghatározott RAM területére másoláshoz, és a titkosítás elkezdéséhez. A titkosított eredeti szöveg egy tömbben lesz tárolva. Az eredeti és a titkosított szövegnek is 128 bitesnek kell lennie. A titkosítás sikeres végét a *getCipher(plaintext, ciphertext)* esemény jelzi.

A titkosító kulcs változtatásához a *clrKey(key)* utasítást kell meghívni először, mely *FAIL* értékkel tér vissza, ha a titkosító eljárásban jelenleg használt kulccsal nem egyezik meg a *key* paraméter. Ezzel a tervezéssel elkerülhetőek a konfliktusok a különböző esetekben.

[12] Az AES titkosítás használatához alap szinten ismernünk kell a CC2420 chip felépítését, illetve regisztereinek nevét. Az AES titkosítás aktiválásához a következő regiszterek használata szükséges: CC2420\_RAM\_KEY0, CC2420\_RAM\_KEY1, CC2420\_RAM\_SABUF, CC2420\_SECCTRL0, CC2420\_SECCTRL1, CC2420\_SAES, CC2420\_SNOP.

A regiszterek használatához a megfelelő interfészek használatára is szükség van: pl.:  
*uses interface CC2420Register as SECCTRL1;*

Negatív tapasztalatom volt a fejlesztés során, hogy a szimulátor nem tudta lefordítani és használni a CC2420.h header fájlt változtatások nélkül, mellyel kapcsolatban a kutatásaim azt mutatják, hogy verziófüggő probléma.

A titkosítandó üzenet a következőképpen épül fel. Mindössze 16byte lehetséges, ami azt jelenti, hogy 16 karakterből állhat. A 16 karakterből az első 10 karakter azonosítóként szolgál, 11-14-ig a mért adatot tartalmazza, míg az utolsó kettő bájt a szenzorral kapcsolatos információ közölhető.

[17] A visszafejtéshez a program egy nyílt forráskódú AES Calculator nevű programot használ módosításokkal, mivel az eredeti program 32/48/64 bájtos üzenetek dekódolására megfelelő csak.

A dolgozatban bemutatott Tossim szimulátorral kapcsolatban szeretném kiemelni egyik negatív tapasztalatomat. Az AES titkosítás programozása során számos problémába ütköztem, melyeknek egyik forrása maga a szimulátor volt. Bár nem állt a rendelkezésemre a fejlesztés alatt valódi micaZ szenzor, van lehetőség a szenzor használata nélküli fordításra. Ugyanaz a programkód ebben az esetben hiba nélkül lefordult és elindult a program futása. Természetesen a kapcsolat hívása után megállt a program futása, mivel nem volt a számítógéphez csatlakoztatva szenzor. Azonban a szimulátor opcióval futtatva nem fordult le a program, mely arra enged következtetni és a kutatásaim is azt mutatják, hogy a Tossim szimulátorral nem lehet teljes mértékben helyettesíteni egy valódi szenzor működését.

## 6.5 Szenzorok

Egy-egy szenzort külön szálak reprezentálnak. Egy valódi szenzorhálózatban a szenzorok külön-külön, önálló egységként dolgoznak. Ebben az esetben az áramfogyasztást kell szimulálni, azaz a szenzorok azonos időben, véletlenszerűen egy számlálót növelnek, „külön életet élnek”. A számláló növelése mellett a szenzor feladata az adattovábbítás, azaz egy szenzortól érkező adatot tovább kell küldeni a központi számítógép felé, ellenőrzés, azaz bizonyos időközökben a szenzoroknak maguk kell ellenőrizniük, hogy a szomszédos szenzorokkal meg van-e a kapcsolat, valamint a titkosítást is meg kell oldania.

A következő adatokat kell tartalmaznia: egy számláló a fogyasztás szimulálásához, adatokat az ügyfélről (azonosító), a szomszédos szenzorokról adatok,

akikkel kapcsolatot tart (azonosítók, ill. egy kiemelt azonosító, mely azt mondja meg, melyik a leggyorsabb irány a központi számítógép felé).

A szenzorok inicializálása az adatbázisban tárolt adatokkal történik, megkapjuk a legutóbbi elmentett értékeket, milyen szomszédos szenzorok vannak, akikkel kapcsolatot teremthetnek, és meghatározzák a gráfban a megfelelő helyüket.

A szenzorok önálló működésének hátránya a karbantartás: lehetséges, hogy egy szenzor tönkremegy, vagy áramkimaradás történik. Ebben az esetben jelezni kell a központ felé a meghibásodást. Ezt a szomszédos szenzoroknak kell megtenniük, ha nem tudnak kapcsolatot teremteni, az adatok változásáról (azaz, hogy milyen szomszédos szenzorokkal rendelkeznek) küldenek egy üzenetet a központi számítógép felé.

## 6.6 Adatbázis kezelés

A hálózat felépítése szerver – kliens típusú, ahol a szerver adatbázis kezelési feladatot lát el. A Szakember táblában van tárolva a szakemberekhez tartozó:

- Szakember Azonosító
- Felhasználó név
- Jelszó (belépéskor ellenőrzésre kerül, és eldől, hogy milyen jogosultsággal rendelkezik)
- Munkahely
- Lakcím, elérhetőség

Szakemberekhez külön tartozik egy tábla, melyben minden egyes cselekedete tárolásra kerül, a tábla tartalma:

- Szakember Azonosító (szakemberé)
- Végrehajtott feladat
- Dátum

Szenzor tábla az összes szenzor adatát tartalmazza:

- Szenzor Azonosító
- Ügyfél Azonosító

- Szomszédos szenzorok Szenzor azonosítói

Szenzorokhoz külön tartozik egy tábla, melyben a szenzorok állapotváltozásait, ill. elvégzett feladatait tároljuk. Állapotváltozás közé tartozik az inicializálás, bekapcsolás, a szomszédos szenzorral a kapcsolat elvesztése, új kapcsolat létrehozása, míg elvégzett feladatok közé sorolhatjuk az adatok lekérdezését. Nem írjuk be az adatbázisba továbbítás műveletét.

- Szenzor Azonosító
- Végrehajtott művelet
- Dátum

Ügyfél táblában a következő adatokat tároljuk:

- Szenzor Azonosító
- Ügyfél Azonosító
- Ügyfél neve
- Ügyfél címe
- Legutolsó fogyasztási érték
- Mért fogyasztási érték
- Tartozás

Adatbázis kezeléshez használt programok:

MySQL 5.0 – adatbázis létrehozásához

MySQL-Python csomagok – a Python és az adatbázis közti kapcsolat létesítéséhez

[3] Az adatbázis kezeléséhez a következő függvények használhatók. Meg kell határozni, hogy melyik adatbázishoz akar az alkalmazás csatlakozni, majd létre kell hozni a kapcsolatot. Ehhez a MySQL modul *connect()* metódusával egy kapcsolattartó objektum hozható létre.

```
>>>> kapcsolat = MySQLdb.connect( host = "localhost", db = "ugyfel", port =
8000, username = administrator_name, passwd = administrator_passwd )
```

A kapcsolat objektum létrehozása után egy kurzor objektumra is szükség van, mely egy fajta mutatóként használható az adatbázisban való keresés során. A kurzor lehetővé teszi, hogy az alkalmazásnak ne kelljen az utasítások által visszaadott teljes eredménykészletet dolgoznia, lehetőség van csak egyetlen, vagy csak néhány sor feldolgozására is.

```
>>>> kurzor = kapcsolat.cursor();
```

Miután létrejött a kapcsolat az alkalmazás és az adatbázis között, a kurzor objektum metódusaival és attribútumaival lekérdezések küldhetők az adatbázisra.

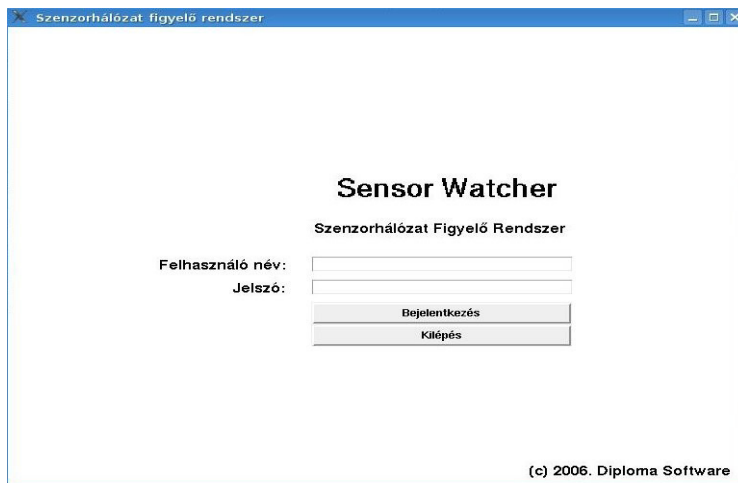
Az *execute()* metódus előkészít és végrehajt egy adatbázis parancsot, vagy lekérdezést az adatbázisban. A következő függvény egy új ügyfélnek az adatbázisba történő felvételét mutatja be:

```
def new_ugyfel(szenzor_azonosito, ugyfel_azonosito, jelszo, ugyfel_neve, ugyfel_cime):  
    sql_utasitas = 'insert into ugyfel values( "%s", "%s",  
        "%d", "%d", "%d", "%s") % (ugyfel_neve, ugyfel_cime, 0, 0, 0,  
        jelszo))';  
    try  
        kurzor.execute ( sql_utasitas );  
        logolas ( " Uj ugyfel felvetele sikeresen letrejott. ");  
    except  
        logolas ( " Uj ugyfel felvetele nem jott létre. ");  
    kurzor.close();
```

## 6.7 Felület – grafika

A szakember egy grafikus felületen keresztül kommunikál a rendszerrel. Kizárólag ennyit lát a rendszerből. A rendszer állandó figyelést kíván, ezért több felhasználós rendszerről van szó, mely azt jelenti, hogy a szakember munkakezdés előtt

egy, az adminisztrátortól kapott felhasználónévvel és jelszóval bejelentkezik a rendszerbe. Az adatbázisban tárolva vannak az adatai, ill. minden egyes tette elmentésre kerül. Ezzel a bekövetkezett hibák előzményeit kereshetjük vissza:

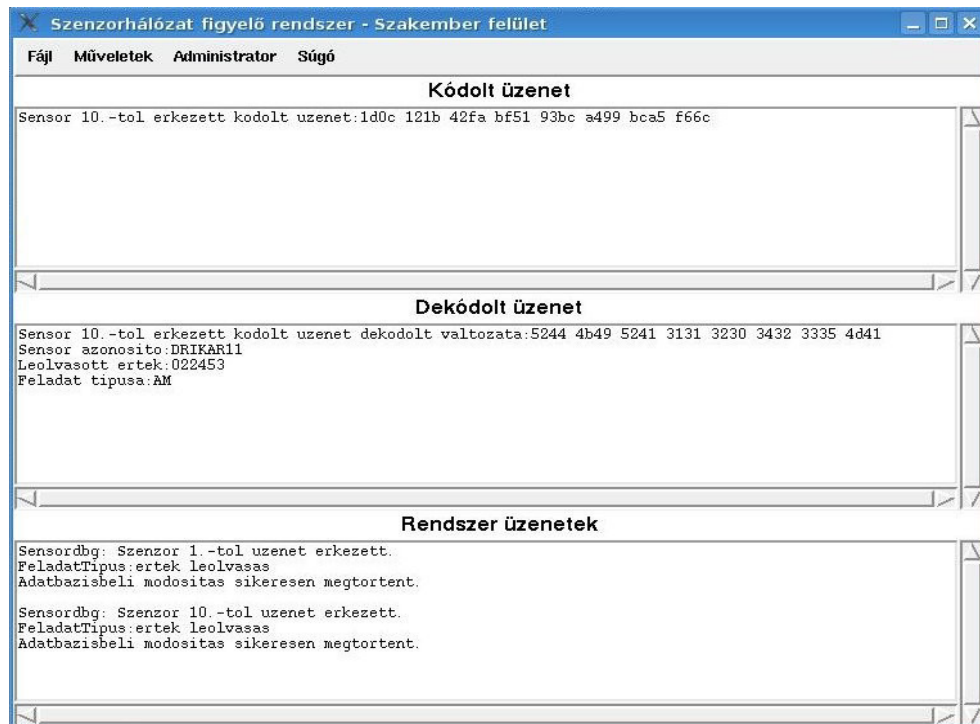


6. ábra: Bejelentkező ablak

Bejelentkezés után ellenőrzi a rendszer a szakember jogosultságát és elindul a rendszer. A felhasználói felületen csak bizonyos művelet elvégzésére van lehetősége a szakembernek. Azok a műveletek, melyek nem elérhetőek a szakemberek számára, inaktívak.

A szakemberek számára a menüsorból a **Műveletek** és a **Súgó** menü érhető el. A műveletek között van lehetőség a teljes rendszer újraindítására, ill. ellenőrzésére, adatok lekérdezésére az egyes ügyfelekről, kijelentkezésre. A súgó segítséget nyújt a rendszer használatához.

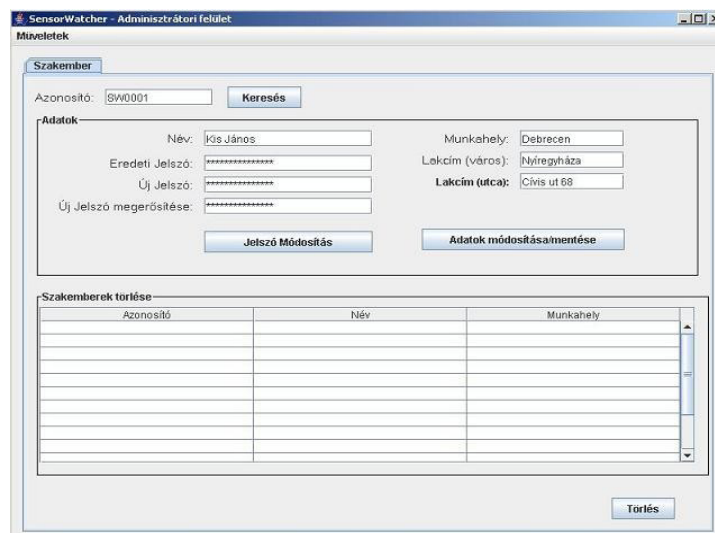
A felület három darab ablakra oszlik. Az első ablakban az érkező kódolt üzenetek jelennek meg, a második ablakban láthatóak a dekódolt üzenetből az információk széttagolt állapotban, míg a harmadik ablakban a rendszerrel kapcsolatos információk érkeznek. Amennyiben figyelmeztetés érkezik, felugró ablakban is figyelmezteti a rendszer kezelőjét a jelentkezett problémáról.



7. ábra: Szakemberek kezelő felülete

A menüsorból az *Adminisztrátor* menüpont csak az adminisztrátor számára érhető el.

Az adminisztrátor jogosultsága közé tartozik a rendszer frissítése, a szakemberek adatainak kezelése, adatok módosítása, szakember felvétele, ill. törlése a rendszerből, az adatbázishoz való hozzáférés, ellenőrzés, módosítása.



8. ábra: Adminisztrátori kezelőfelület

## 7. Összefoglalás

Céлом egy szenzorhálózat implementálása volt, egy felhasználhatóság szempontjából teljes alkalmazás elkészítésével. A célok közül a rendszer nehéz telepítése és a szimulátor hibás működése miatt nem valósult meg minden funkciója az alkalmazásnak. A diplomamunkám segítségével viszont szerettem volna felhívni a figyelmet a szenzorhálózatok jelentőségére. A szenzorhálózatok már ott vannak a mindennapjainkban, néhol tudunk nélkül is segítik a munkánkat, védik az életünket. Egy olyan tudományág van kialakulóban mely szorosan összekötődik az informatikával, és idővel az informatika bizonyos területeit alapjaiban fogja megváltoztatni, ezért a mi egyetemünkön is érdekesebb lenne tüzetesebben foglalkozni ezzel a témakörrel fejlesztői szempontból.

A program írása során számos új tapasztalatra tettem szert. Ezek közül szeretném kiemelni a megfelelő fejlesztő környezet kialakításának fontosságát, mivel a rossz fejlesztő környezet jelentős mértékben hátráltathatja a fejlesztés menetét. Egy másik fontos tapasztalatnak tartom a különböző programnyelvek együttes használatát egyetlen probléma megoldására, melynek nagy hasznát fogom venni további programozási problémák megoldásakor.

## 8. Függelék

[14]. Szenzor csomópontok táblázata:

| Sensor Node Name | Microcontroller                     | Tranceiver                                                   | Program + Data Memory | External Memory           | Programming            | Remarks                                                       |
|------------------|-------------------------------------|--------------------------------------------------------------|-----------------------|---------------------------|------------------------|---------------------------------------------------------------|
| <b>BEAN</b>      | MSP430F169                          | CC1000 ( 300 – 1000 Mhz ) with 78.6 kbit/s                   |                       | 4 Mbit                    |                        | YATOS support                                                 |
| <b>BTnode</b>    | Atmel Atmega 128 L ( 8Mhz @ 8MIPS ) | Chipcon CC1000 (433-915 MHz) and Bluetooth (2.4 GHz)         | 64 + 180 K RAM        | 128K FLASH ROM, 4K EPROM  | C and nesC Programming | BTnut and TinyOS support                                      |
| <b>COTS</b>      | ATMEL Microcontroller 916 Mhz       |                                                              |                       |                           |                        |                                                               |
| <b>Dot</b>       | ATMEGA 163                          |                                                              | 1K RAM                | 8-16K Flash               | weC                    |                                                               |
| <b>Eyes</b>      | MSP430F149                          | TR1001                                                       |                       | 8Mbit                     |                        | PeerOS support                                                |
| <b>IMote</b>     | ARM core 12 Mhz                     | Bluetooth with the range of 30 m                             | 64K SRAM              | 512KB Flash               |                        | TinyOS support                                                |
| <b>IMote 1.0</b> | ARM 7TDMI 12-48 Mhz                 | Bluetooth with the range of 30 m                             | 64K SRAM              | 512KB Flash               |                        | TinyOS support                                                |
| <b>IMote 2.0</b> | Marvel PXA271 ARM 11-400 Mhz        | TI CC 2420 802.15.4/Zigbee compliant radio                   | 32MB SRAM             | 32 Flash                  |                        | Microsoft, .NET Micro, TinyOS support                         |
| <b>Iris</b>      | Atmega1281                          | Atmel AT86RF230 802.15.4/ZigBee compliant radio              | 8K RAM                | 128K Flash                | nesC                   | TinyOS, MoteWorks Support                                     |
| <b>KMote</b>     | TI MSP430 microcontroller           | 250 kbit/s 2.4Ghz IEEE 802.15.4 Chipcon Wireless Transceiver | 10k RAM               | 48k Flash                 |                        | TinyOS and SOS Support                                        |
| <b>Mica</b>      | Atmel ATMEGA 103 4 Mhz 8-bit CPU    | RFM TR1000 radio 50 kbit/s                                   | 128+4K RAM            | 512K Flash                | nesC Programming       | TinyOS Support                                                |
| <b>Mica2</b>     | ATMEGA 128L                         | Chipcon 868/916 MHz                                          | 4K RAM                | 128K Flash                |                        | TinyOS, SOS and MantisOS Support                              |
| <b>Mica2Dot</b>  | ATMEGA 128                          |                                                              | 4K RAM                | 128K Flash                |                        |                                                               |
| <b>MicaZ</b>     | ATMEGA 128                          | TI CC2420 802.15.4/Zigbee compliant radio                    | 4K RAM                | 128K Flash                | nesC                   | TinyOS, SOS, MantisOS, and Nano-RK Support                    |
| <b>Mulle</b>     | Renesas M16C                        | Bluetooth 2.0                                                | 31K RAM               | 384K+4K Flash, 2MB EEPROM | C programming          | TCP/IP and Bluetooth, Profiles: LAP, DUN, PAN and SPP Support |
| <b>Nymph</b>     | ATMEGA 128L                         | CC1000                                                       |                       | 64 KB EEPROM              |                        | MantisOS Support                                              |
| <b>Rene</b>      | ATMEL8535                           | 916Mhz radio with bandwidth of 10 kbit/s                     | 512 bytes RAM         | 8K Flash                  |                        | TinyOS Support                                                |
| <b>SenseNode</b> | MSP430F1611                         | Chipcon CC2420                                               | 10K RAM               | 48K Flash                 | C and nesC Programming | GenOS and TinyOS Support                                      |
| <b>SunSPOT</b>   | ARM 920T                            | 802.15.4                                                     | 512K RAM              | 4MB Flash                 | Java                   | Squawk J2ME Virtual Machine                                   |
| <b>Telos</b>     | Motorola HCS08                      |                                                              | 4K RAM                |                           |                        |                                                               |
| <b>TelosB</b>    | Texas Instruments MSP430            | 250kbit/s 2.4 GHz IEEE 802.15.4 Chipcon Wireless             | 10K RAM               | 48K Flash                 |                        | Contiki, TinyOS, SOS and MantisOS Support                     |

|                   |                                          |                                                             |         |                           |                        |                                          |
|-------------------|------------------------------------------|-------------------------------------------------------------|---------|---------------------------|------------------------|------------------------------------------|
|                   | microcontroller                          | Transevier                                                  |         |                           |                        |                                          |
| <b>T-Mote Sky</b> | Texas Instruments MSP430 microcontroller | 250kbit/s 2.4 GHz IEEE 802.15.4 Chipcon Wireless Transevier | 10K RAM | 48K Flash                 |                        | Contiki, TinyOS, SOS and MantiOS Support |
| <b>weC</b>        | Atmel AVR AT90S2313                      | RFM TR 1000 RF                                              |         |                           |                        |                                          |
| <b>XYZ</b>        | ML67 series ARM/THUMB microcontroller    | CC2420 Zigbee compliant radio from Chipcon                  | 32K RAM | 256K Flash                | C Programming          | SOS Operating System                     |
| <b>FireFly</b>    | Atmel ATmega 1281                        | Chipcon CC2420                                              | 8K RAM  | 128K Flash ROM, 4K EEPROM | C Programming          | Nano-RK RTOS Support                     |
| <b>Iris Mote</b>  | Atmel ATmega 1281                        | Atmel rf230                                                 | 8K RAM  | 128K Flash ROM, 4K EEPROM | C and nesC Programming | TinyOS Support                           |

Pl. egy átvjáró csomópont:

|                 | Micro-controller | Tranceiver | Interface (USB/Serial/Wifi/Ethernet) | Program Memory | External Memory |
|-----------------|------------------|------------|--------------------------------------|----------------|-----------------|
| <b>Stargate</b> | IntelPXA255      | 802.11     | serial connection to WSN             | 64MB SDRAM     | 32MB Flash      |

## Irodalomjegyzék

- [1]. Hajnal Péter: Gráfelmélet Polygon Jegyzettár Szeged 2003  
Szerkeszti: Kurusa Árpád, Kiadja az SZTE BOLYAI INTÉZET
- [2]. Buttyán Levente, Vajda István: Kriptográfia és alkalmazásai,  
Typotex kiadó, 2004
- [3]. Brad Dayley: Python zsebkönyv, Kiskapu kiadó, Budapest, 2008
- [4]. Rashi Gupta: Mindentudó Python, Kossuth Kiadó, 2003
- [5]. Kömlődi Ferenc: SzenzorRendszerek 2005  
<<http://www.nhit-it3.hu>>
- [6]. David Gay, Phil Levis, Rob von Behren, Matt Welsh, Eric Brewer,  
David Culler: The nesC Language: A Holistic Approach to  
Networked Embedded Systems  
<<http://www.tinyos.net/papers/nesc.pdf>>
- [7]. TinyOS Tutorial:  
<<http://www.tinyos.net/tinyos-1.x/doc/tutorial/>>
- [8]. TinyOS 1.1 Dokumentáció, 2003:  
<<http://www.tinyos.net/tinyos-1.x/doc/index.html>>
- [9]. Wikipédia, A szabad enciklopédia  
<<http://hu.wikipedia.org/wiki/>>
- [10]. VIK Wiki, BME wiki oldala:  
<<https://wiki.sch.bme.hu/bin/view>>
- [11]. The Standalone AES Encryption of CC2420 (TinyOS and MICAz):  
<[http://cis.sjtu.edu.cn/index.php/The\\_Standalone\\_AES\\_Encryption\\_of\\_CC2420\\_\(TinyOS\\_2.10\\_and\\_MICAz\)](http://cis.sjtu.edu.cn/index.php/The_Standalone_AES_Encryption_of_CC2420_(TinyOS_2.10_and_MICAz))>
- [12]. CC2420 2.4Ghz IEEE802.15.4 / Zigbee-ready RF trasnceiver :  
<<http://inst.eecs.berkeley.edu/~cs150/Documents/CC2420.pdf>>
- [13]. Philip Levis: TinyOS/nesc Programming Reference Manual: 2006  
<<http://www.csie.isu.edu.tw/homepage/ccchen/WSNs&Apps/TinyOS&NesC.pdf>>

- [14]. WikiPedia The Free Encyclopedia  
<[http://en.wikipedia.org/wiki/Sensor\\_node](http://en.wikipedia.org/wiki/Sensor_node)>
- [15]. MICAz Wireless Measurement System:  
<[http://www.xbow.com/Products/Product\\_pdf\\_files/Wireless\\_pdf/MICAz\\_Datasheet.pdf](http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf)>
- [16]. TinyOS 2.x Eclipse Plugin:  
<<http://tos-ide.ethz.ch/wiki/pmwiki.php?>>>
- [17]. AES calculator: Lawrie Brown, 2005  
<<http://www.unsw.adfa.edu.au/~lpb/src/AEScalc/index.html>>
- [18]. aptana Pydev – Python Development Environment  
<<http://pydev.sourceforge.net/>>