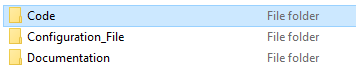
**Secure AODV in MANET**

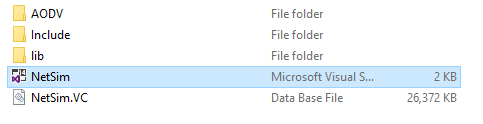
**Software Recommended:** NetSim Standard v11.0 (32-bit), Microsoft Visual Studio 2015, 2017

SAODV is an extension of the AODV routing protocol that can be used to protect the route discovery mechanism providing security features like integrity and authentication. The reason only route discovery is secured by AODV is because data messages can be protected using a point-to-point security protocol like IPSec. SAODV uses a key management system and each node maintains public keys, encryption keys and decryption keys.

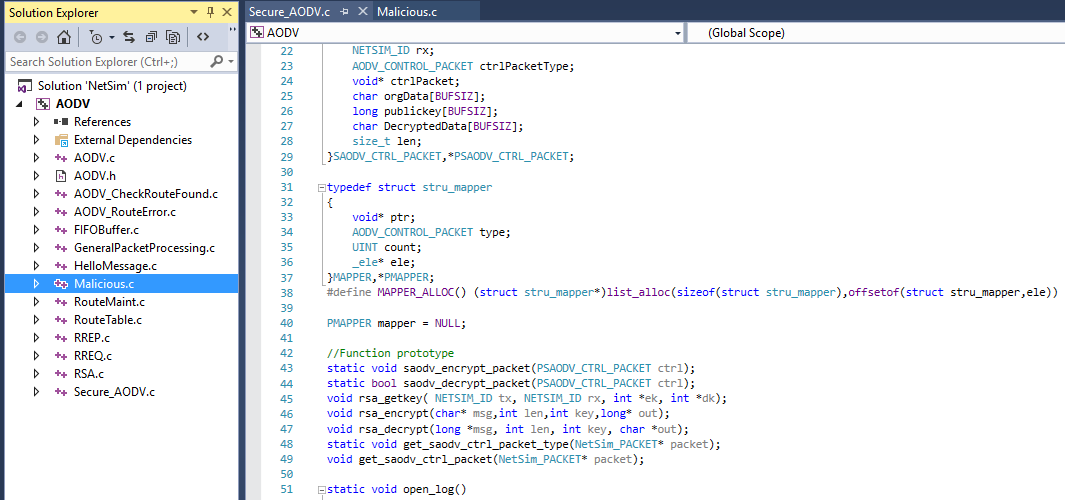
To implement SAODV, we have added **Secure AODV.c, RSA.c and Malicious.c** files in AODV project. RSA.c file is used to generate keys, encrypt and decrypt the data. Users can implement their own encryption algorithms by changing RSA.c file. Malicious.c file is used to identify malicious nodes present in the network.

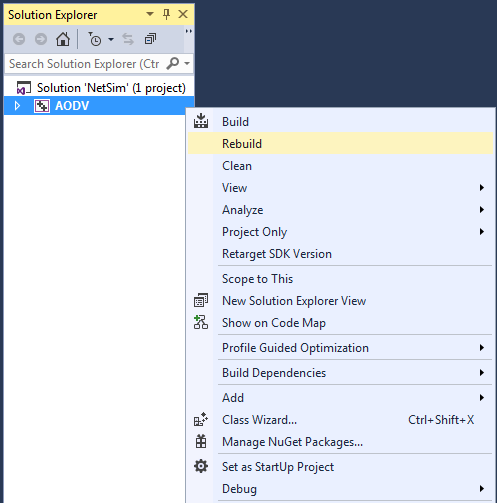
**Steps:**

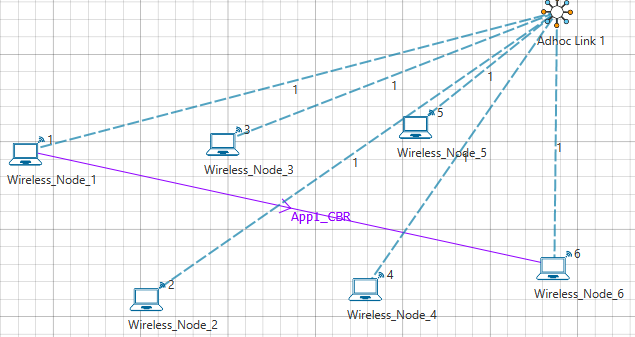
1. After you unzip the file the folder would look like



1. Open the NetSim.sln file inside the Simulation folder through Visual Studio 2015, 2017



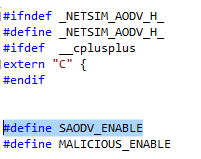
1. Right click on AODV in Solution Explorer and select rebuild.
2. Upon rebuilding, libAODV.dll will get created in the DLL folder.
3. Now copy the libAODV.dll from this DLL folder and paste it in NetSim bin folder present in the NetSim installation directory. The NetSim install directory would look something like **“C:\Program Files\NetSim Standard\bin”**.
4. Note that there exists a libAODV.dll in this bin folder. This is the default file being shipped with NetSim. The user is replacing this file with the newly built file.
5. Therefore, take care to rename the original libAODV.dll file, so that it isn’t lost. For example, you may rename it as libAODV\_default.dll
6. Run NetSim and open **Configuration.netsim** file present inside the zip folder



1. Run the Simulation
2. Secure AODV logs **Secure\_AODV.txt** file in the bin folder present in NetSim’s installed directory. This can be explained in next section

**USE CASES:**

1. **Secure AODV implementation**

Here users can enable Secure AODV (Open **AODV.h** file)

A Secure\_AODV.c file is added to the AODV project which contains the following important functions:

* + saodv\_encrypt\_packet()

This function is used to encrypt the control packet data

* + saodv\_decrypt\_packet()

This function is used to decrypt the control packet data

* + get\_rrep\_str\_data()

This function is used to get the route reply data from AODV\_RREP control packet

* + get\_rreq\_str\_data()

This function is used to get the route request data from AODV\_RREQ control packet

* + get\_saodv\_ctrl\_packet\_type()

This function is used to change the control packet type from AODV (AODV\_RREQ, AODV\_RREP) to SAODV (SAODV\_RREQ, SAODV\_RREP)

* + get\_saodv\_ctrl\_packet()

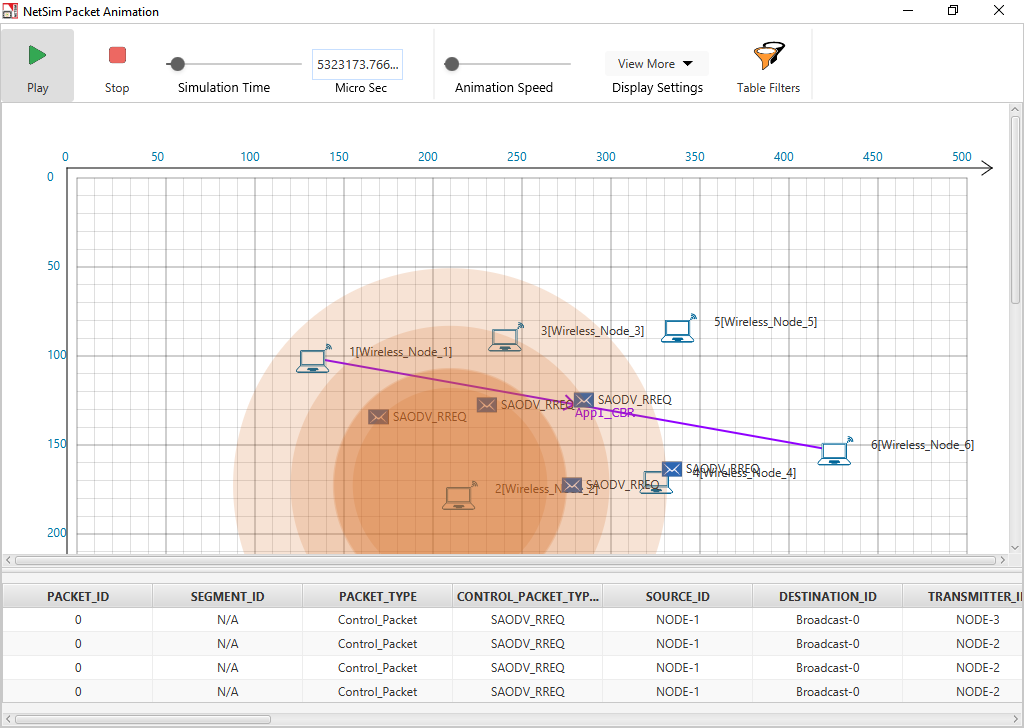
This function is called whenever a new control packet is generated

* + get\_aodv\_ctrl\_packet()

This function is called while processing the control packets

Comment the line #define MALICIOUS\_ENABLE present in AODV.h file. Rebuild the solution and replace the dlls as explained before and run the simulation.

After simulation of the given Configuration file, open packet animation. In the packet users can notice **SAODV\_RREQ** and **SAODV\_RREP** control packets.



The SAODV codes also logs certain details in SAODVlog.txt. The format of the log file is such that each control packet is logged. The first line represents the packet type and the numbering used in a NetSim internal numbering system where by **30701 is RREQ and 30702 is RREP**. The second line is the message which is encrypted. The third line contains the encrypted message after running the RSA encryption algorithm. The fourth line is after decryption and if everything is OK, the 2nd and 4th lines must match

**.........................................**

**Packet Type = 30701**

**Org Data = 1,0,1,11.1.1.6,0,11.1.1.1,1**

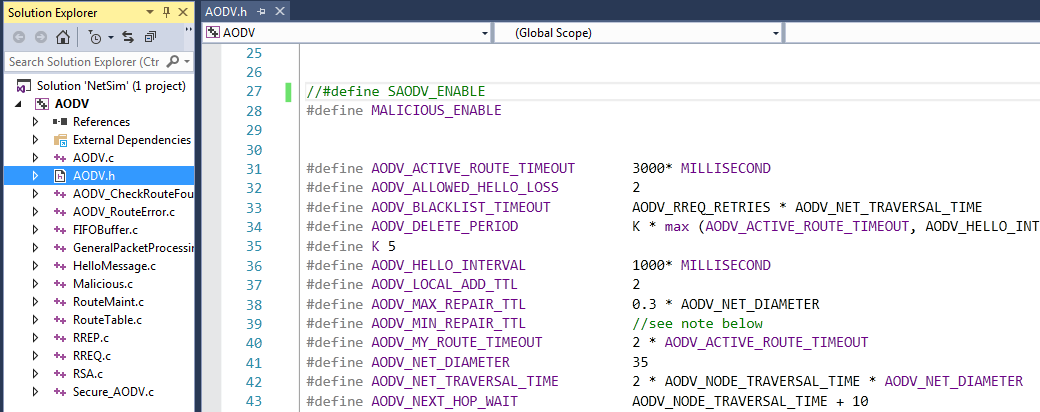
**Encrypted Data = \*‑Ÿ‑\*‑\*\*¡\*¡\*¡‑Ÿ‑\*\*¡\*¡\*¡\*‑\***

**Decrypted Data = 1,0,1,11.1.1.6,0,11.1.1.1,1**

**..................................................................................**

1. **Malicious node implementation**

Here users can enable code to malicious node problem. Enable #define MALICIOUS\_ENABLE and comment #define SAODV\_ENABLE that are present inside AODV.h file.



Malicious node advertises wrong routing information to produce itself as a specific node and receives whole network traffic.

After receiving whole network traffic it can either modify the packet information or drop them to make the network complicated

In packet animation, users can notice that malicious node will take all the packets and drops without forwarding to destination

A file **malicious.c** is added to the AODV project which contains the following functions:

* IsMaliciousNode ()

This function is used to identify whether a current device is malicious or not in-order to establish malicious behavior.

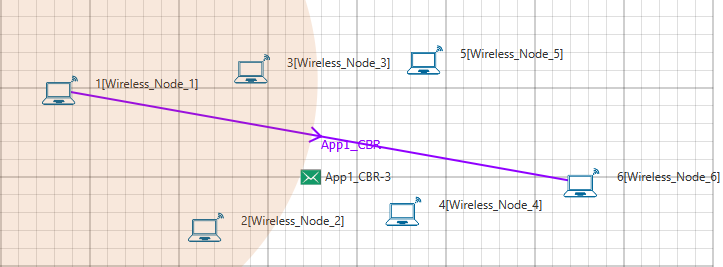
* fn\_NetSim\_AODV\_MaliciousRouteAddToTable()

This function is used to add a fake route entry into the route table of the malicious device with its next hop as the destination.

* fn\_NetSim\_AODV\_MaliciousProcessSourceRouteOption()

This function is used to drop the received packets if the device is malicious, instead of forwarding the packet to the next hop.

You can set any device as a malicious node and you can have more than one malicious node in a scenario. Device id’s of malicious nodes can be set using malicious\_node [ ] array present in malicious.c file. Comment the line #define SAODV\_ENABLE present in AODV.h file. Rebuild the solution and replace the dlls as explained before and run the simulation. If we run simulation without SAODV, we will get zero throughput because malicious node gets all the packets and drops without forwarding to destination. You can notice this in NetSim packet animation.



**MALICIOUS NODE**

1. **Both Secure AODV and Malicious node implementation**

Enable the below mentioned lines of code present in AODV.h file.

#define SAODV\_ENABLE

#define MALICIOUS\_ENABLE

Rebuild the solution and replace the dlls as explained before and run the simulation. Packets will be transmitted to the destination, since SAODV helps in overcoming the Malicious Node problem. Route reply RREP from malicious node 4 will not be accepted by Node 1. It takes the Route reply from node 2 and forms the route.

The SAODV logs certain details in **Secure\_AODV.txt**. The first line represents the packet type 30701 = RREQ. The second line is the message logged by SAODV when malicious node tries to decrypt the message

**.........................................**

**Packet Type = 30702**

**Encryption and decryption fails. This could be a malicious node**

**.........................................**

**Packet Type = 30702**

**Encryption and decryption fails. This could be a malicious node**

**.........................................**

**Code modifications done:**

Please note that in this project we have added Secure\_AODV.c, RSA.c and Malicious.c files

We have added the following macros in AODV.h file

#define SAODV\_ENABLE

#define MALICIOUS\_ENABLE

Then we have added the following lines of code in enum\_AODV\_Ctrl\_Packet in AODV.h file

//#ifdef SAODV\_ENABLE

SAODV\_RREQ,

SAODV\_RREP,

SAODV\_RERR,

//#endif

Then we have added the following function prototypes in AODV.h file

#ifdef SAODV\_ENABLE

void get\_saodv\_ctrl\_packet(NetSim\_PACKET\* packet);

void get\_aodv\_ctrl\_packet(NetSim\_PACKET\* packet);

void saodv\_copy\_packet(NetSim\_PACKET\* dest, NetSim\_PACKET\* src);

void saodv\_free\_packet(NetSim\_PACKET\* packet);

void remove\_from\_mapper(void\* ptr,bool isfree);

#endif

bool IsMaliciousNode(NETSIM\_ID devId);

We have added the following function prototypes in AODV.c file

bool IsMaliciousNode(NETSIM\_ID devId);

int fn\_NetSim\_AODV\_MaliciousRouteAddToTable(NetSim\_EVENTDETAILS\*);

int fn\_NetSim\_AODV\_MaliciousProcessSourceRouteOption(NetSim\_EVENTDETAILS\*);

Then we have added the following lines of code in NETWORK\_IN event in fn\_NetSim\_AODV\_Run() function present in AODV.c file

#ifdef SAODV\_ENABLE

switch(pstruEventDetails->pPacket->nControlDataType)

{

case SAODV\_RREQ:

case SAODV\_RREP:

case SAODV\_RERR:

get\_aodv\_ctrl\_packet(pstruEventDetails->pPacket);

break;

}

if(pstruEventDetails->pPacket == NULL)

{

return -1; //Decryption fail.

}

#endif

We have added the following lines of code in AODVctrlPacket\_RREQ and default cases in NETWORK\_IN event to check the current node is malicious or not

if(IsMaliciousNode(pstruEventDetails->nDeviceId))

fn\_NetSim\_AODV\_MaliciousProcessSourceRouteOption(pstruEventDetails);

Then we have added the following code in fn\_NetSim\_AODV\_CopyPacket () function present in AODV.c file

#ifdef SAODV\_ENABLE

switch(srcPacket->nControlDataType)

{

case SAODV\_RERR:

case SAODV\_RREQ:

case SAODV\_RREP:

saodv\_copy\_packet(destPacket,srcPacket);

return 0;

break;

default:

#endif

return fn\_NetSim\_AODV\_CopyPacket\_F(destPacket,srcPacket);

#ifdef SAODV\_ENABLE

break;

}

#endif

Then we have added the following code in int fn\_NetSim\_AODV\_FreePacket () present in the AODV.c file

#ifdef SAODV\_ENABLE

switch(packet->nControlDataType)

{

case SAODV\_RERR:

case SAODV\_RREQ:

case SAODV\_RREP:

saodv\_free\_packet(packet);

return 0;

break;

default:

remove\_from\_mapper(packet->pstruNetworkData->Packet\_RoutingProtocol, true);

return 0;

break;

}

#endif

Then we have added the following function calls in fn\_NetSim\_AODV\_GenerateRREQ (), fn\_NetSim\_AODV\_RetryRREQ () and fn\_NetSim\_AODV\_ForwardRREQ () functions present in RREQ.c file

#ifdef SAODV\_ENABLE

get\_saodv\_ctrl\_packet(packet);

#endif

Then we have added the following function calls in fn\_NetSim\_AODV\_GenerateRREP(), fn\_NetSim\_AODV\_ForwardRREP () and fn\_NetSim\_AODV\_GenerateRREPByIntermediate ()

functions present in RREP.c file

#ifdef SAODV\_ENABLE

get\_saodv\_ctrl\_packet(packet);

#endif