Secure AODV in MANET

Software Recommended: NetSim Standard v11.0, Microsoft Visual Studio 2015/2017

Project Download Link:

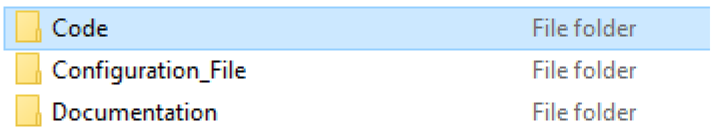
https://github.com/NetSim-TETCOS/SECURE_AODV_v11.0/archive/master.zip

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.

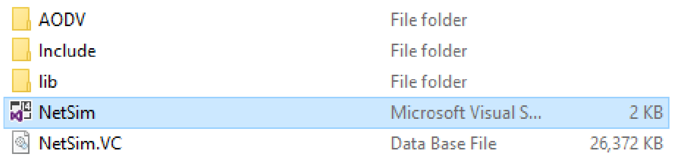
To implement SAODV, we have added **Secure AODV.c, RSA.c and Malicious.c** files in AODV project. RSA.c file is used to generate keys, encrypt and decrypt the data. Users can implement their own encryption algorithms by changing RSA.c file. Malicious.c file is used to identify malicious nodes present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

Steps:

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

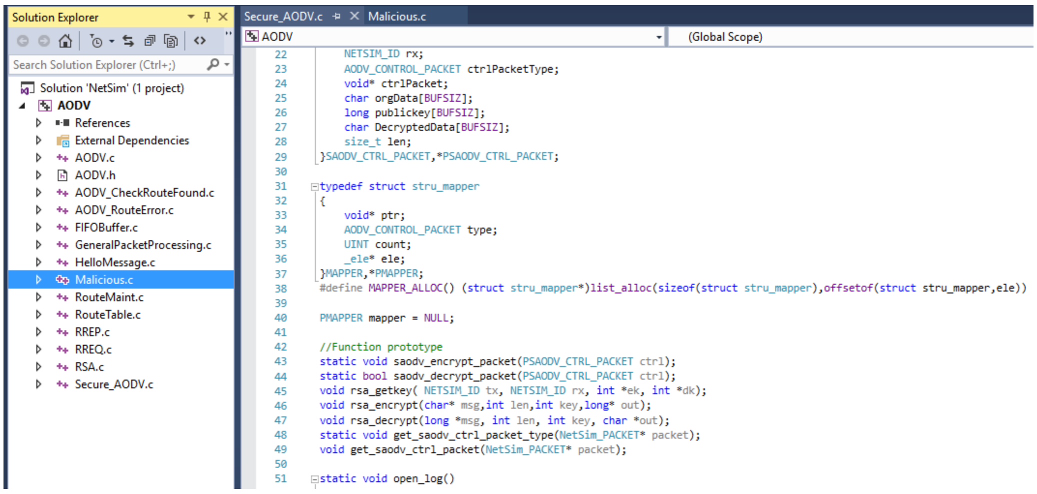


Code	File folder
Configuration_File	File folder
Documentation	File folder



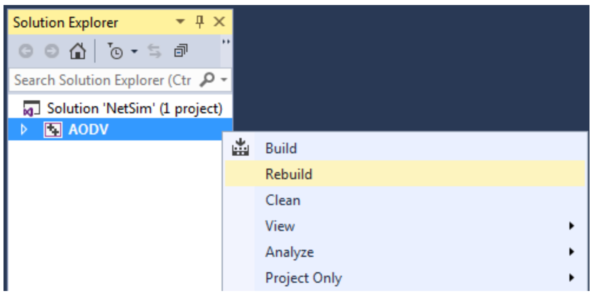
AODV	File folder	
Include	File folder	
lib	File folder	
NetSim	Microsoft Visual S...	2 KB
NetSim.VC	Data Base File	26,372 KB

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



```
        NETSIM_ID rx;
        AODV_CONTROL_PACKET ctrlPacketType;
        void* ctrlPacket;
        char orgData[BUFSIZ];
        long publickey[BUFSIZ];
        char DecryptedData[BUFSIZ];
        size_t len;
    }SAODV_CTRL_PACKET,*PSAODV_CTRL_PACKET;

    typedef struct stru_mapper
    {
        void* ptr;
        AODV_CONTROL_PACKET type;
        UINT count;
        _ele* ele;
    }MAPPER,*PMAPPER;
    #define MAPPER_ALLOC() (struct stru_mapper*)list_alloc(sizeof(struct stru_mapper),offsetof(struct stru_mapper,ele))

    PMAPPER mapper = NULL;

    //Function prototype
    static void saodv_encrypt_packet(PSAODV_CTRL_PACKET ctrl);
    static bool saodv_decrypt_packet(PSAODV_CTRL_PACKET ctrl);
    void rsa_getkey( NETSIM_ID tx, NETSIM_ID rx, int *ek, int *dk);
    void rsa_encrypt(char* msg,int len,int key,long* out);
    void rsa_decrypt(long *msg, int len, int key, char *out);
    static void get_saodv_ctrl_packet_type(NetSim_PACKET* packet);
    void get_saodv_ctrl_packet(NetSim_PACKET* packet);

    static void open_log()
```

3. Right click on AODV in Solution Explorer and select rebuild.



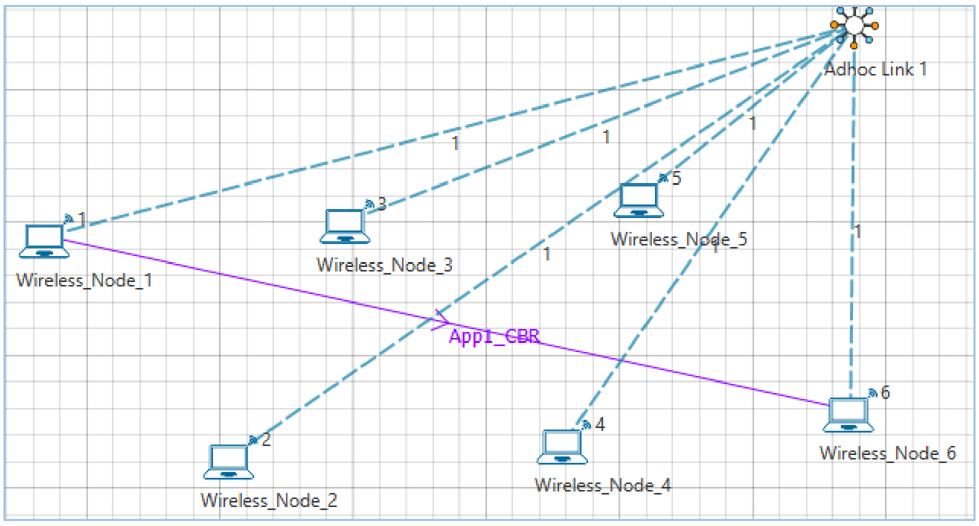
4. Upon rebuilding, libAODV.dll will get created in the DLL folder.

5. 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"**.

6. 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.

7. 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

8. Run NetSim and open **Configuration.netsim** file present inside the zip folder



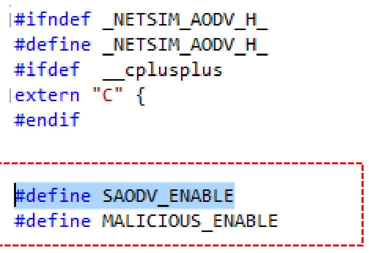
9. Run the Simulation

10. 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)



```
#ifndef _NETSIM_AODV_H_
#define _NETSIM_AODV_H_
#ifdef  __cplusplus
extern "C" {
#endif

#define SAODV_ENABLE
#define MALICIOUS_ENABLE
```

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()`

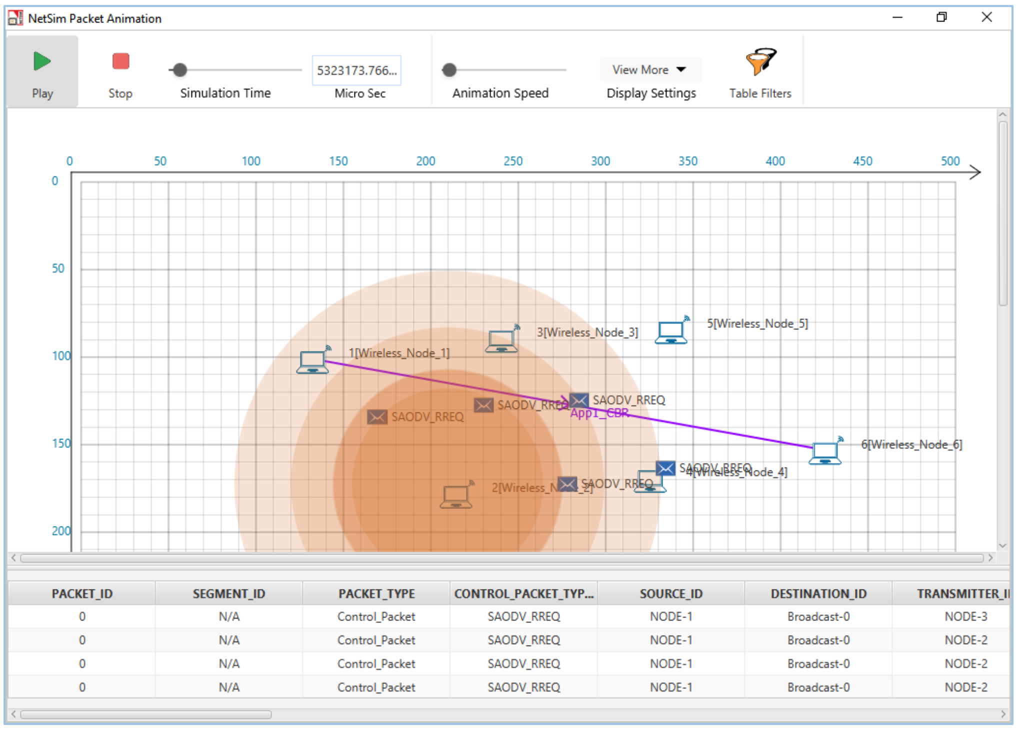
  This function is called whenever a new control packet is generated

- `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.



PACKET_ID	SEGMENT_ID	PACKET_TYPE	CONTROL_PACKET_TYP...	SOURCE_ID	DESTINATION_ID	TRANSMITTER_II
0	N/A	Control_Packet	SAODV_RREQ	NODE-1	Broadcast-0	NODE-3
0	N/A	Control_Packet	SAODV_RREQ	NODE-1	Broadcast-0	NODE-2
0	N/A	Control_Packet	SAODV_RREQ	NODE-1	Broadcast-0	NODE-2
0	N/A	Control_Packet	SAODV_RREQ	NODE-1	Broadcast-0	NODE-2

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

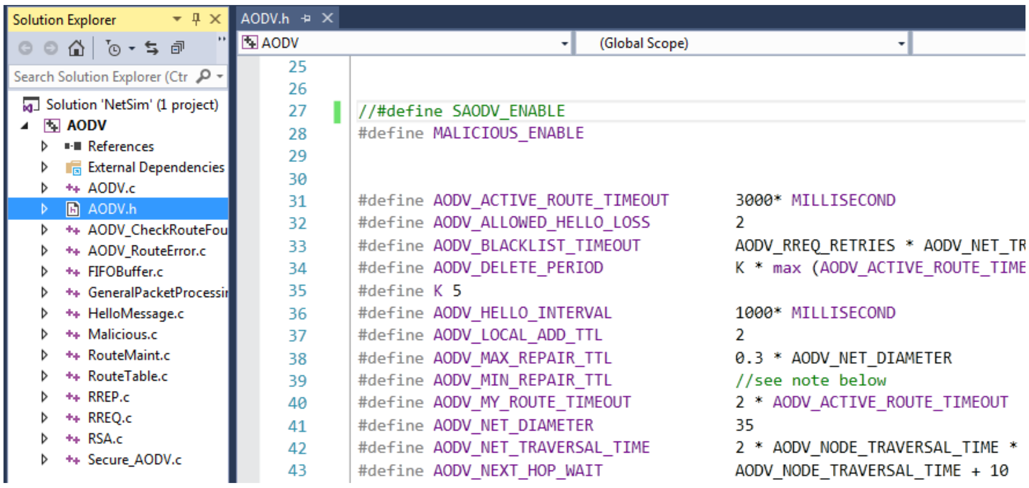
Encrypted Data = *-Ÿ-*-¡*¡*¡-Ÿ-**¡*¡*¡*-***

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

…………………………………………………………………………

2. 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.



```
//#define SAODV_ENABLE
#define MALICIOUS_ENABLE


#define AODV_ACTIVE_ROUTE_TIMEOUT        3000* MILLISECOND
#define AODV_ALLOWED_HELLO_LOSS          2
#define AODV_BLACKLIST_TIMEOUT           AODV_RREQ_RETRIES * AODV_NET_TR
#define AODV_DELETE_PERIOD               K * max (AODV_ACTIVE_ROUTE_TIME
#define K 5
#define AODV_HELLO_INTERVAL              1000* MILLISECOND
#define AODV_LOCAL_ADD_TTL               2
#define AODV_MAX_REPAIR_TTL              0.3 * AODV_NET_DIAMETER
#define AODV_MIN_REPAIR_TTL              //see note below
#define AODV_MY_ROUTE_TIMEOUT            2 * AODV_ACTIVE_ROUTE_TIMEOUT
#define AODV_NET_DIAMETER                35
#define AODV_NET_TRAVERSAL_TIME          2 * AODV_NODE_TRAVERSAL_TIME *
#define AODV_NEXT_HOP_WAIT               AODV_NODE_TRAVERSAL_TIME + 10
```

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:

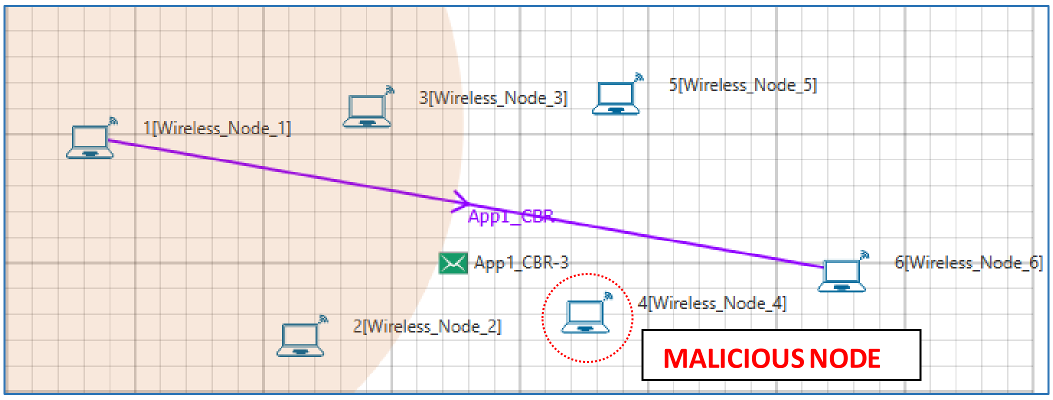
- `IsMaliciousNode ()`
  This function is used to identify whether a current device is malicious or not in-order to establish malicious behavior.

- `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.



3. 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(pstruEventDet
	ails);
```

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
```