Implementing a new Crypto Algorithm – MISTY1

Software: NetSim Standard v13.2, Visual Studio 2022, Wireshark

Project Download Link:

https://github.com/NetSim-TETCOS/MISTY_ENCRYPTION_v13.2/archive/refs/heads/main.zip

Follow the instructions specified in the following link to download and setup the Project in NetSim:

https://support.tetcos.com/en/support/solutions/articles/14000128666-downloading-and-setting-up-netsim-file-exchange-projects

Example

1. The **MISTY_ENCRYPTION_WorkSpace** comes with a sample network configuration that are already saved. To open this example, go to Your work in the Home screen of NetSim and click on the **MISTY_ENCRYPTION_Example** from the list of experiments.
2. The Network Scenario mainly consist of 2 Wired Nodes and 1 Router.

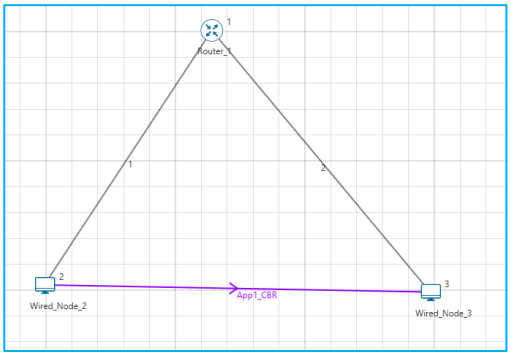


Figure 1: Network Scenario

3. Run Simulation with AES encryption enabled in the Application settings.

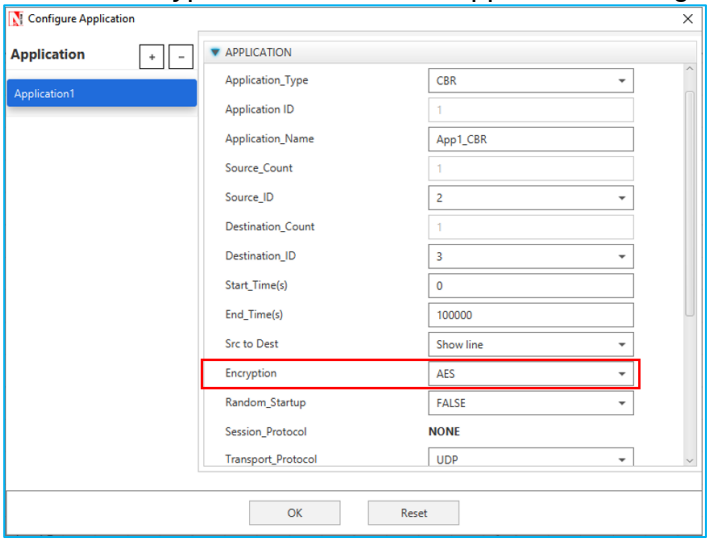


Figure 2: Application Configuration window

4. Now misty1 codes will be running instead of AES256.
5. Make sure to keep the Wireshark Online (wireShark window will open during the runtime)

6. If kept offline WireShark can be viewed from the results Dashboard

Results and discussion

After simulation open metrics window and observe the result.

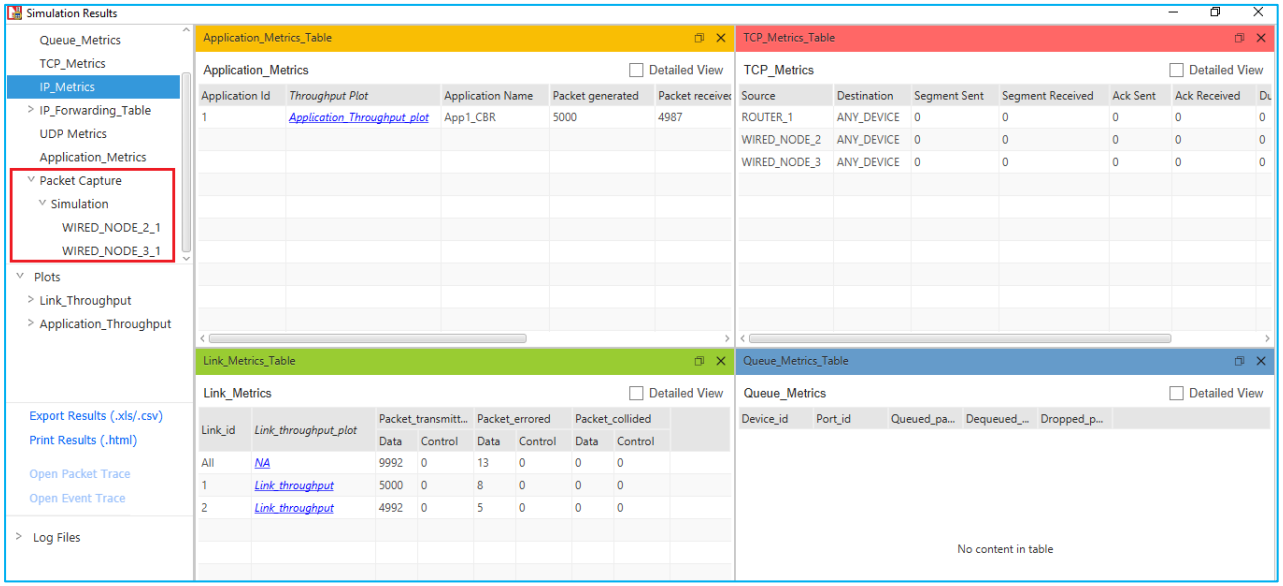


Figure 3: NetSim result dashboard.

You can see the encrypted payload in Wireshark either during simulation if online is set or after the simulation if offline is set in the source or destination nodes.

If Wireshark option is set to offline, then the capture files can be accessed from the results dashboard.

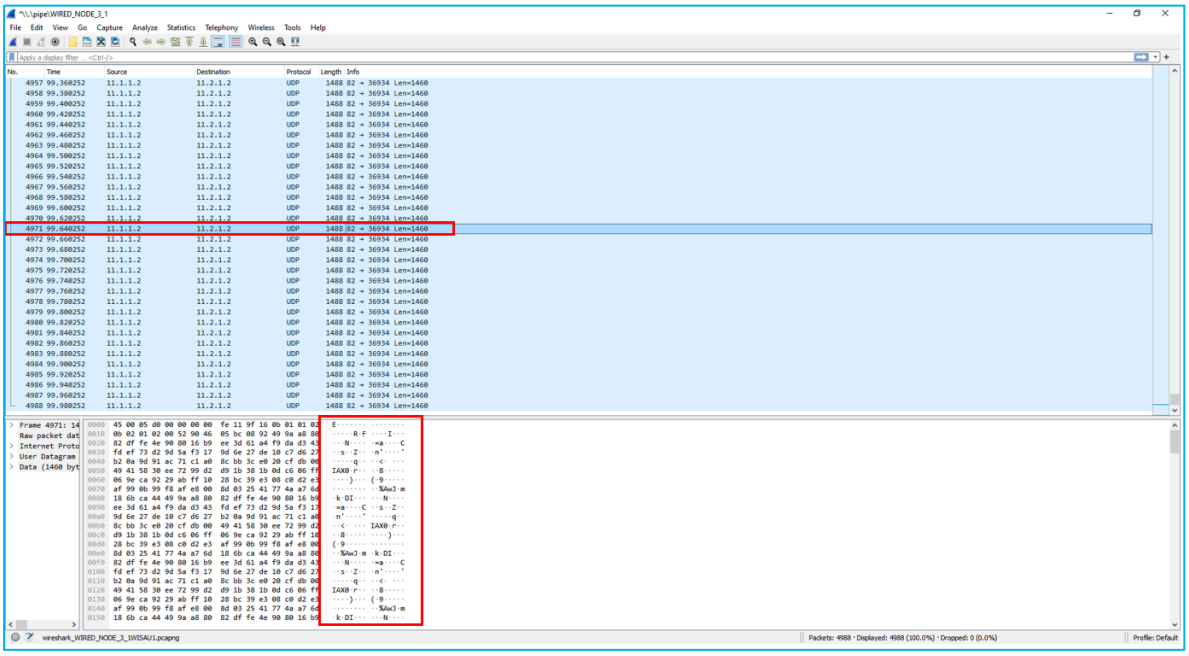


Figure 4: Wireshark window see the data is encrypted

Appendix: NetSim source code modifications

Added code in misty_run.c, within Application project

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "application.h"


void misty_run(char* str, int* len)
{
	int n;
	int l = *len;

	unsigned char buf[32];
	unsigned char key[32];

	for (n = 0; n < *len; n += 16, str += 16, l -= 16)
	{
		/* Set the plain-text */
		memcpy(buf, str, min(16, l));

		misty1_main(buf);
		memcpy(str, buf, 16);
	}
}
```

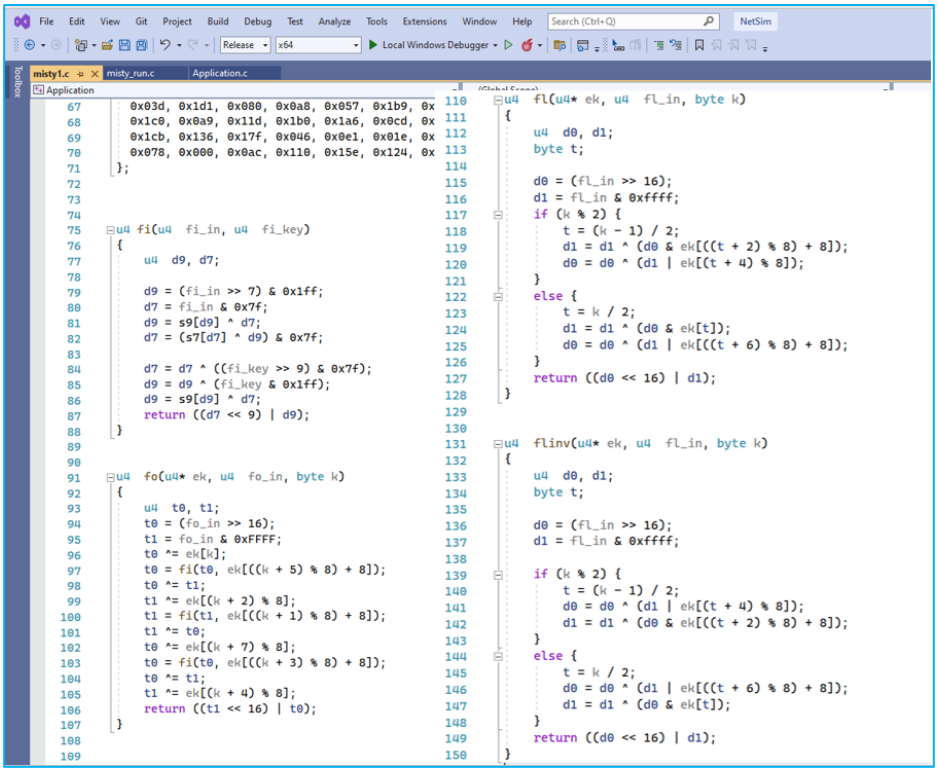
In the misty_run() function inside the misty_run.c file we pass the plain text in parts of 16 bytes each time to get it encrypted. This is done because the crypto algorithm accepts a 16-byte plaintext as input. Here the variable str contains the packet payload and len corresponds to the size of payload in bytes.

Added code in misty1.c, within Application project

A. Addition of #include<application.h> and #define uint8 unsigned char to the beginning of the misty1.c file

```
#include <stdlib.h>
#include <string.h>
#include "application.h"
typedef unsigned long u4;
typedef unsigned char byte;
#define MISTY1_KEYSIZE 32
#define uint8  unsigned char
```

B. Removed inline keyword that is present before the functions fi(), fo(), fl() and flinv().



```
misty1.c | misty_run.c | Application.c
Application

    0x03d, 0x1d1, 0x080, 0x0a8, 0x057, 0x1b9, 0x
    0x1c0, 0x0a9, 0x11d, 0x1b0, 0x1a6, 0x0cd, 0x
    0x1cb, 0x136, 0x17f, 0x046, 0x0e1, 0x01e, 0x
    0x078, 0x000, 0x0ac, 0x110, 0x15e, 0x124, 0x
};

u4 fi(u4  fi_in, u4  fi_key)
{
    u4  d9, d7;

    d9 = (fi_in >> 7) & 0x1ff;
    d7 = fi_in & 0x7f;
    d9 = s9[d9] ^ d7;
    d7 = (s7[d7] ^ d9) & 0x7f;

    d7 = d7 ^ ((fi_key >> 9) & 0x7f);
    d9 = d9 ^ (fi_key & 0x1ff);
    d9 = s9[d9] ^ d7;
    return ((d7 << 9) | d9);
}

u4  fo(u4* ek, u4  fo_in, byte k)
{
    u4  t0, t1;
    t0 = (fo_in >> 16);
    t1 = fo_in & 0xFFFF;
    t0 ^= ek[k];
    t0 = fi(t0, ek[((k + 5) % 8) + 8]);
    t0 ^= t1;
    t1 ^= ek[(k + 2) % 8];
    t1 = fi(t1, ek[((k + 1) % 8) + 8]);
    t1 ^= t0;
    t0 ^= ek[(k + 7) % 8];
    t0 = fi(t0, ek[((k + 3) % 8) + 8]);
    t0 ^= t1;
    t1 ^= ek[(k + 4) % 8];
    return ((t1 << 16) | t0);
}

u4  fl(u4* ek, u4  fl_in, byte k)
{
    u4  d0, d1;
    byte t;

    d0 = (fl_in >> 16);
    d1 = fl_in & 0xffff;
    if (k % 2) {
        t = (k - 1) / 2;
        d1 = d1 ^ (d0 & ek[((t + 2) % 8) + 8]);
        d0 = d0 ^ (d1 | ek[((t + 4) % 8)]);
    }
    else {
        t = k / 2;
        d1 = d1 ^ (d0 & ek[t]);
        d0 = d0 ^ (d1 | ek[((t + 6) % 8) + 8]);
    }
    return ((d0 << 16) | d1);
}

u4  flinv(u4* ek, u4  fl_in, byte k)
{
    u4  d0, d1;
    byte t;

    d0 = (fl_in >> 16);
    d1 = fl_in & 0xffff;

    if (k % 2) {
        t = (k - 1) / 2;
        d0 = d0 ^ (d1 | ek[((t + 4) % 8)]);
        d1 = d1 ^ (d0 & ek[((t + 2) % 8) + 8]);
    }
    else {
        t = k / 2;
        d0 = d0 ^ (d1 | ek[((t + 6) % 8) + 8]);
        d1 = d1 ^ (d0 & ek[t]);
    }
    return ((d0 << 16) | d1);
}
```

C. Now go to the main() function in the file and check the line #ifdef TESTMAIN was removed or commented before the main() function and also check the associated #endif at the end of the main() function.

D. main() function was renamed to unsigned char* misty1_main(uint8* input)

```
//#ifdef TESTMAIN
unsigned char* misty1_main(uint8* input)
{
        /*
          Key:        00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
          Plaintext:  01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
          Ciphertext: 8b 1d a5 f5 6a b3 d0 7c 04 b6 82 40 b1 3b e9 5d
        */


        u4  Key[] = { 0x00112233, 0x44556677, 0x8899aabb, 0xccddeeff };
        u4  Plaintext[4];
        // u4  Ciphertext[]= { 0x8b1da5f5, 0x6ab3d07c, 0x04b68240, 0xb13be95d};
        u4  ek_e[MISTY1_KEYSIZE], ek_d[MISTY1_KEYSIZE];
        u4  c[4];

        /* misty1_keyinit(ek_e,Key);
         misty1_encrypt_block(ek_e,&Plaintext[0],&c[0]);
         misty1_encrypt_block(ek_e,&Plaintext[2],&c[2]);

         if (!memcmp(c,Ciphertext,4 * sizeof(u4))) {
           printf("Encryption OK\n");
         }
```

```
else {
  printf("Encryption failed[0x%08lx 0x%08lx 0x%08lx 0x%08lx]\n",
          c[0],c[1],c[2],c[3]);
  exit(1);
}

misty1_keyinit(ek_d,Key);

if (memcmp(ek_e,ek_d,MISTY1_KEYSIZE*sizeof(u4))) {
  printf("Internal Error keysch is wrong\n");
  exit(1);
}

misty1_decrypt_block(ek_d,&Ciphertext[0],&c[0]);
misty1_decrypt_block(ek_d,&Ciphertext[2],&c[2]);


if (!memcmp(c,Plaintext,4 * sizeof(u4))) {
  printf("Decryption OK\n");
}
else {

  printf("Decryption failed[0x%08lx 0x%08lx 0x%08lx 0x%08lx]\n",
          c[0],c[1],c[2],c[3]);
  exit(1);
}
*/
```

E. Commented the declaration of Cipher text, Modify the declaration of Plaintext variable, as shown below:

```
u4  Key[] = { 0x00112233, 0x44556677, 0x8899aabb, 0xccddeeff };
u4  Plaintext[4];
// u4  Ciphertext[]= { 0x8b1da5f5, 0x6ab3d07c, 0x04b68240, 0xb13be95d};
u4  ek_e[MISTY1_KEYSIZE], ek_d[MISTY1_KEYSIZE];
u4  c[4];
```

F. Now check the commented lines starting from misty1_keyinit() to misty1_key_destroy() as shown below:

```
/* misty1_keyinit(ek_e,Key);
        misty1_encrypt_block(ek_e,&Plaintext[0],&c[0]);
        misty1_encrypt_block(ek_e,&Plaintext[2],&c[2]);

        if (!memcmp(c,Ciphertext,4 * sizeof(u4))) {
          printf("Encryption OK\n");
        }
        else {
          printf("Encryption failed[0x%08lx 0x%08lx 0x%08lx 0x%08lx]\n",
                  c[0],c[1],c[2],c[3]);
          exit(1);
        }

        misty1_keyinit(ek_d,Key);
```

```
 if (memcmp(ek_e,ek_d,MISTY1_KEYSIZE*sizeof(u4))) {
   printf("Internal Error keysch is wrong\n");
   exit(1);
 }

 misty1_decrypt_block(ek_d,&Ciphertext[0],&c[0]);
 misty1_decrypt_block(ek_d,&Ciphertext[2],&c[2]);


 if (!memcmp(c,Plaintext,4 * sizeof(u4))) {
   printf("Decryption OK\n");
 }
 else {

   printf("Decryption failed[0x%08lx 0x%08lx 0x%08lx 0x%08lx]\n",
            c[0],c[1],c[2],c[3]);
   exit(1);
 }
 */
```

G. Addition of the following lines of code just above the misty1_key_destroy(ek_e); statement as shown below:

```
// Memcpy is used to equate input which is Char to Plaintext
// which is Unsigned Long

memcpy(Plaintext, input, 2 * sizeof(u4));
memcpy(&Plaintext[2], &input[8], 2 * sizeof(u4));

misty1_keyinit(ek_e, Key);
misty1_encrypt_block(ek_e, Plaintext, &c[0]);
misty1_encrypt_block(ek_e, &Plaintext[2], &c[2]);

memcpy(input, c, 2 * sizeof(u4));
memcpy(&input[8], &c[2], 2 * sizeof(u4));

misty1_key_destroy(ek_e);
misty1_key_destroy(ek_d);
memset(Key, 0, 4 * sizeof(u4));
```

H. Inside the misty1_main function the above codes were modified to ensure that the plaintext is properly initialized with the 16 bytes of payload received, for the encryption to happen

I. Here, memcpy() is done initially to equate input received as which is char, to the plain text which is unsigned long.

```
memcpy(Plaintext,input,2*sizeof(u4));
memcpy(&Plaintext[2],&input[8],2*sizeof(u4));
```

J. After the calls to misty1_encrypt_block() memcpy() is done to equate the encrypted cipher text back to the input.

```
 memcpy(input, c, 2 * sizeof(u4));
```

memcpy(&input[8], &c[2], 2 * sizeof(u4));

K. Now double click on the application.c file and make a call to misty_run() function instead of the call to aes256, inside the copy_payload() function.

```
void copy_payload(UINT8 real[],NetSim_PACKET* packet,unsigned int* payload,
ptrAPPLICATION_INFO info)
{
u_short i;
uint32_t key = 16;
if (payload)
{
for (i = 0; i < *payload; i++)
{
if (info->encryption == Encryption_XOR)
real[i] = xor_encrypt('a' + i % 26, 16);
else
real[i] = 'a' + i % 26;
}
if (info->encryption == Encryption_TEA)
encryptBlock(real, payload, &key);
else if (info->encryption == Encryption_AES)
misty_run(real, payload);
//aes256(real,payload);
else if(info->encryption==Encryption_DES)
des(real,payload);
}
}
```