|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  File name: zuc.c  Version: V1.1  Date: Oct 28,2016  Description: This code provide the implement of ZUC algorithm,which consist of three parts:key  stream generation,confidentiality algorithm  and integrity algorithm.  Function List:  1.AddMod // calculate a+b mod 2^31-1  2.PowMod // calculate x\*2^k mod 2^31-1  3.L1 // linear transformation L1:X^(X<<< 2)^(X<<<10)^(X<<<18)^(X<<<24)  4.L2 // linear transformation L2:X^(X<<< 8)^(X<<<14)^(X<<<22)^(X<<<30)  5.BitValue // test if the value of M at the position i equals 0  6.GetWord // get a 32bit word ki from bit strings k[i],k[i+1]..., // namely ki=k[i]||k[i+1]||…||k[i+31] 7.LFSRWithInitMode // Initialisation mode,refresh the current state of LFSR  8.LFSRWithWorkMode // working mode,refresh the current state of LFSR  9.BR // Bit Reconstruction  10.F // nonlinear function  11.ZUC\_Init // Initialisation process of ZUC  12.ZUC\_Work // working stage of ZUC  13.ZUC\_GenKeyStream // generate key stream  14.ZUC\_Confidentiality // the ZUC-based condifentiality algorithm  15.ZUC\_Integrity // the ZUC-based integrity algorithm    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include "zuc.h"  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Function: AddMod  Description: calculate a+b mod 2^31-1  Calls:  Called By: LFSRWithInitMode  Input: a,b: unsigned int(32bit)  Output:  Return: c, c=a+b mod 2^31-1  Others:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  unsigned int AddMod(unsigned int a, unsigned int b)  {  unsigned int c = a + b;  if (c >> 31)  {  c = (c & 0x7fffffff) + 1;  }  return c;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Function: PowMod  Description: calculate x\*2^k mod 2^31-1  Calls: Called By: LFSRWithInitMode  Input: x: input  k: exponential  Output:  Return: x\*2^k mod 2^31-1  Others:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  unsigned int PowMod(unsigned int x, unsigned int k)  {  return (((x << k) | (x >> (31 - k))) & 0x7fffffff);  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Function: L1  Description: linear transformation L1  Calls:  Called By: F  Input: X: input  Output:  Return: X^(X<<< 2)^(X<<<10)^(X<<<18)^(X<<<24)  Others:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  unsigned int L1(unsigned int X)  {  return X ^ ZUC\_rotl32(X, 2) ^ ZUC\_rotl32(X, 10) ^ ZUC\_rotl32(X, 18) ^ ZUC\_rotl32(X, 24);  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Function: L2  Description: linear transformation L2  Calls:  Called By: F  Input: X: input  Output:  Return: X^(X<<< 8)^(X<<<14)^(X<<<22)^(X<<<30)  Others:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  unsigned int L2(unsigned int X)  {  return X ^ ZUC\_rotl32(X, 8) ^ ZUC\_rotl32(X, 14) ^ ZUC\_rotl32(X, 22) ^ ZUC\_rotl32(X, 30);  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Function: BitValue  Description: test if the value of M at the position i equals 0  Calls:  Called By: ZUC\_Integrity  Input: M: message  i: the position i  Output:  Return: 0:the value of M at the position i equals 0  1:the value of M at the position i equals 1  Others:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  unsigned char BitValue(unsigned int M[], unsigned int i)  {  int j, k;  j = i >> 5;  k = i & 0x1f;  if (M[j] & (0x1 << (31 - k)))  return 1;  else  return 0;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Function: GetWord  Description: get a 32bit word ki from bit strings k[i],k[i+1]...,namely ki=k[i]||k[i+1]||…||k[i+31] Calls:  Called By: ZUC\_Integrity  Input: k[]:  i: the position i  Output: Return: ki=k[i]||k[i+1]||…||k[i+31]  Others:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  unsigned int GetWord(unsigned int k[], unsigned int i) //获取字串中的从第i个比特值开始的字  {  int j, m;  unsigned int word;  j = i >> 5;  m = i & 0x1f;  if (m == 0)  word = k[j];  else  word = (k[j] << m) | (k[j + 1] >> (32 - m));  return word;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Function: LFSRWithInitMode  Description: Initialisation mode,refresh the current state of LFSR  Calls: AddMod,PowMod  Called By: ZUC\_Init  Input: LFSR\_S:current state of LFSR  u:u=W>>1  Output: Null  Return: Null  Others:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void LFSRWithInitMode(unsigned int LFSR\_S[], unsigned int u)//LFSR初始化模式  {  unsigned int v = LFSR\_S[0], i;  v = AddMod(v, PowMod(LFSR\_S[15], 15));  v = AddMod(v, PowMod(LFSR\_S[13], 17));  v = AddMod(v, PowMod(LFSR\_S[10], 21));  v = AddMod(v, PowMod(LFSR\_S[4], 20));  v = AddMod(v, PowMod(LFSR\_S[0], 8));  for (i = 0; i < 15; i++)  {  LFSR\_S[i] = LFSR\_S[i + 1];  }  LFSR\_S[15] = AddMod(v, u);  if (!LFSR\_S[15])  {  LFSR\_S[15] = 0x7fffffff;  }  };  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Function: LFSRWithWorkMode  Description: working mode,refresh the current state of LFSR  Calls: AddMod,PowMod  Called By: ZUC\_Work  Input: LFSR\_S:current state of LFSR  Output: Null  Return: Null  Others:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void LFSRWithWorkMode(unsigned int LFSR\_S[])//LFSR工作模式  {  unsigned int v = LFSR\_S[0], i;  v = AddMod(v, PowMod(LFSR\_S[15], 15));  v = AddMod(v, PowMod(LFSR\_S[13], 17));  v = AddMod(v, PowMod(LFSR\_S[10], 21));  v = AddMod(v, PowMod(LFSR\_S[4], 20));  v = AddMod(v, PowMod(LFSR\_S[0], 8));  for (i = 0; i < 15; i++)  {  LFSR\_S[i] = LFSR\_S[i + 1];  }  LFSR\_S[15] = v;  if (!LFSR\_S[15])  {  LFSR\_S[15] = 0x7fffffff;  }  };  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Function: BR  Description: Bit Reconstruction  Calls:  Called By: ZUC\_Init,ZUC\_Work  Input: LFSR\_S:current state of LFSR  Output: BR\_X[]:achieve X0,X1,X2,X3  Return: Null  Others:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void BR(unsigned int LFSR\_S[], unsigned int BR\_X[])//比特重组  {  BR\_X[0] = ((LFSR\_S[15] & 0x7fff8000) << 1) | (LFSR\_S[14] & 0x0000ffff);  BR\_X[1] = ((LFSR\_S[11] & 0x0000ffff) << 16) | ((LFSR\_S[9] & 0x7fff8000) >> 15);  BR\_X[2] = ((LFSR\_S[7] & 0x0000ffff) << 16) | ((LFSR\_S[5] & 0x7fff8000) >> 15);  BR\_X[3] = ((LFSR\_S[2] & 0x0000ffff) << 16) | ((LFSR\_S[0] & 0x7fff8000) >> 15);  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Function: F  Description: nonlinear function  Calls:  Called By: ZUC\_Init,ZUC\_Work  Input: BR\_X[]:words X0,X1,X2,X3 from BR  F\_R[]:F\_R[0]=R1,F\_R[1]=R2  Output:  Return: W  Others:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  unsigned int F(unsigned int BR\_X[], unsigned int F\_R[])//非线性函数F  {  unsigned int W, W1, W2;  W = (BR\_X[0] ^ F\_R[0]) + F\_R[1];  W1 = F\_R[0] + BR\_X[1];  W2 = F\_R[1] ^ BR\_X[2];  F\_R[0] = L1((W1 << 16) | (W2 >> 16));  F\_R[0] = (ZUC\_S0[(F\_R[0] >> 24) & 0xFF]) << 24 | (ZUC\_S1[(F\_R[0] >> 16) & 0xFF]) << 16 | (ZUC\_S0[(F\_R[0] >> 8) & 0xFF]) << 8 | (ZUC\_S1[F\_R[0] & 0xFF]);  F\_R[1] = L2((W2 << 16) | (W1 >> 16));  F\_R[1] = (ZUC\_S0[(F\_R[1] >> 24) & 0xFF]) << 24 | (ZUC\_S1[(F\_R[1] >> 16) & 0xFF]) << 16 | (ZUC\_S0[(F\_R[1] >> 8) & 0xFF]) << 8 | (ZUC\_S1[F\_R[1] & 0xFF]);  return W;  };  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Function: ZUC\_Init  Description: Initialisation process of ZUC  Calls: ZUC\_LinkToS,BR,F,LFSRWithInitMode  Called By: ZUC\_GenKeyStream  Input: k:initial key  iv:initial vector  Output: LFSR\_S[]:the state of LFSR after initialisation:s0,s1,s2,..s15  BR\_X[] : the current value:X0,X1,X2,X3  F\_R[]:the current value:R1,R2,F\_R[0]=R1,F\_R[1]=R2  Return: Null  Others:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void ZUC\_Init(unsigned char k[], unsigned char iv[], unsigned int LFSR\_S[], unsigned int BR\_X[], unsigned int F\_R[])//祖冲之密码的运行：初始化阶段  {  unsigned char count = 32;  int i;  //loading key to the LFSR s0,s1,s2....s15  printf("\ninitial state of LFSR: S[0]-S[15]\n");  for (i = 0; i < 16; i++)  {  LFSR\_S[i] = ZUC\_LinkToS(k[i], ZUC\_d[i], iv[i]);  printf("%08x ", LFSR\_S[i]);  }  F\_R[0] = 0x00; //R1  F\_R[1] = 0x00; //R2  while (count) //32 times  {  unsigned int W;  BR(LFSR\_S, BR\_X); //BitReconstruction  W = F(BR\_X, F\_R); //nonlinear function  LFSRWithInitMode(LFSR\_S, W >> 1);  count--;  }  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Function: ZUC\_work  Description: working stage of ZUC  Calls: BR,F,LFSRWithWorkMode  Called By: ZUC\_GenKeyStream  Input: LFSR\_S[]:the state of LFSR after initialisation:s0,s1,s2,..s15  BR\_X[] : X0,X1,X2,X3  F\_R[]:R1,R2  Output: pKeyStream[]:key stream  KeyStreamLen:the length of KeyStream,exporting 32bit for a beat  Return: Null  Others:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void ZUC\_Work(unsigned int LFSR\_S[], unsigned int BR\_X[], unsigned int F\_R[], unsigned int pKeyStream[], int KeyStreamLen)//祖冲之算法的运行：工作阶段  {  int i = 0;  BR(LFSR\_S, BR\_X);  F(BR\_X, F\_R);  LFSRWithWorkMode(LFSR\_S);  while (i < KeyStreamLen)  {  BR(LFSR\_S, BR\_X);  pKeyStream[i] = F(BR\_X, F\_R) ^ BR\_X[3];  LFSRWithWorkMode(LFSR\_S);  i++;  }  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Function: ZUC\_GenKeyStream  Description: generate key stream  Calls: ZUC\_Init,ZUC\_Work  Called By: ZUC\_SelfCheck  Input: k[] //initial key,128bit  iv[] //initial iv,128bit  KeyStreamLen //the byte length of KeyStream,exporting 32bit for a beat  Output: KeyStream[] // key strem to be outputed  Return: null  Others:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void ZUC\_GenKeyStream(unsigned char k[], unsigned char iv[], unsigned int KeyStream[], int KeyStreamLen)//祖冲之算法的运行  {  unsigned int LFSR\_S[16]; //LFSR state s0,s1,s2,...s15  unsigned int BR\_X[4]; //Bit Reconstruction X0,X1,X2,X3  unsigned int F\_R[2]; //R1,R2,variables of nonlinear function F  int i;  //Initialisation  ZUC\_Init(k, iv, LFSR\_S, BR\_X, F\_R);  printf("\nstate of LFSR after executing initialization: S[0]-S[15]\n");  for (i = 0; i < 16; i++)  {  printf("%08x ", LFSR\_S[i]);  }  printf("\ninternal state of Finite State Machine:\n");  printf("R1=%08x\n", F\_R[0]);  printf("R2=%08x\n", F\_R[1]);  //Working  ZUC\_Work(LFSR\_S, BR\_X, F\_R, KeyStream, KeyStreamLen);  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Function: ZUC\_Confidentiality  Description: the ZUC-based condifentiality algorithm  Calls: ZUC\_GenKeyStream  Called By: ZUC\_SelfCheck  Input: CK[] //initial key,128bit,uesed to gain the key of ZUC KeyStream  generation algorithm  COUNT //128bit  BEARER //5bit,bearing layer identification,  DIRECTION //1bit  IBS[] //input bit stream,  LENGTH //the bit length of IBS  Output: OBS[] //output bit stream,  Return: null  Others:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void ZUC\_Confidentiality(unsigned char CK[], unsigned int COUNT, unsigned char BEARER, unsigned char DIRECTION, unsigned int IBS[], int LENGTH, unsigned int OBS[])  {  unsigned int \*k;  int L, i, t;  unsigned char iv[16];  //generate vector iv1,iv2,...iv15  iv[0] = (unsigned char)(COUNT >> 24);  iv[1] = (unsigned char)((COUNT >> 16) & 0xff);  iv[2] = (unsigned char)((COUNT >> 8) & 0xff);  iv[3] = (unsigned char)(COUNT & 0xff);  iv[4] = (((BEARER << 3) | (DIRECTION << 2)) & 0xfc);  iv[5] = 0x00;  iv[6] = 0x00;  iv[7] = 0x00;  iv[8] = iv[0];  iv[9] = iv[1];  iv[10] = iv[2];  iv[11] = iv[3];  iv[12] = iv[4];  iv[13] = iv[5];  iv[14] = iv[6];  iv[15] = iv[7];  //L,the length of key stream,taking 32bit as a unit  L = (LENGTH + 31) / 32;  k = malloc(sizeof(unsigned int) \* L);  //generate key stream k  ZUC\_GenKeyStream(CK, iv, k, L); //generate key stream  //OBS=IBS^k  for (i = 0; i < L; i++)  {  OBS[i] = IBS[i] ^ k[i];  }  t = LENGTH % 32;  if (t)  {  OBS[L - 1] = ((OBS[L - 1] >> (32 - t)) << (32 - t));  }  free(k);  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Function: ZUC\_Integrity  Description: the ZUC-based integrity algorithm  Calls: ZUC\_GenKeyStream,BitValue,GetWord  Called By: ZUC\_SelfCheck  Input: IK[] //integrity key,128bit,uesed to gain the key of ZUC KeyStream  generation algorithm  COUNT //128bit  BEARER //5bit,bearing layer identification,  DIRECTION //1bit  M[] //message  LENGTH //the bit length of M  Output:  Return: MAC //message authentication code  Others:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  unsigned int ZUC\_Integrity(unsigned char IK[], unsigned int COUNT, unsigned char BEARER, unsigned char DIRECTION, unsigned int M[], int LENGTH)  {  unsigned int \*k, ki, MAC;  int L, i;  unsigned char iv[16];  unsigned int T = 0;  //generate vector iv1,iv2,...iv15  iv[0] = (unsigned char)(COUNT >> 24);  iv[1] = (unsigned char)((COUNT >> 16) & 0xff);  iv[2] = (unsigned char)((COUNT >> 8) & 0xff);  iv[3] = (unsigned char)(COUNT & 0xff);  iv[4] = BEARER << 3;  iv[5] = 0x00;  iv[6] = 0x00;  iv[7] = 0x00;  iv[8] = iv[0] ^ (DIRECTION << 7);  iv[9] = iv[1];  iv[10] = iv[2];  iv[11] = iv[3];  iv[12] = iv[4];  iv[13] = iv[5];  iv[14] = iv[6] ^ (DIRECTION << 7);  iv[15] = iv[7];  //L,the length of key stream,taking 32bit as a unit  L = (LENGTH + 31) / 32 + 2;  k = malloc(sizeof(unsigned int) \* L);  //generate key stream k  ZUC\_GenKeyStream(IK, iv, k, L);  //T=T^ki  for (i = 0; i < LENGTH; i++)  {  if (BitValue(M, i))  {  ki = GetWord(k, i);  T = T ^ ki;  }  }  //T=T^kLENGTH  ki = GetWord(k, LENGTH);  T = T ^ ki;  //MAC=T^k(32\*(L-1))  ki = GetWord(k, 32 \* (L - 1));  MAC = T ^ ki;  free(k);  return MAC;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Function: ZUC\_SelfCheck  Description: Self-check with standard data  Calls: ZUC\_GenKeyStream,ZUC\_Confidentiality,ZUC\_Integrity  Called By:  Input:  Output:  Return: 0:success  1:error  Others:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int ZUC\_SelfCheck()//总程序，自检  {  int i;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* KeyStream generation validation data \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  // (all 0)  /\* unsigned char  k[16]={0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};  unsigned char  iv[16]={0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};  unsigned int Std\_Keystream[2]={0x27bede74,0x018082da};\*/  //(all 1)  /\*unsigned char  k[16]={0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff};  unsigned char  iv[16]={0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff};  unsigned int Std\_Keystream[2]={0x0657cfa0,0x7096398b};\*/  //(random)  unsigned char  k[16] = {0x3d, 0x4c, 0x4b, 0xe9, 0x6a, 0x82, 0xfd, 0xae, 0xb5, 0x8f, 0x64, 0x1d, 0xb1, 0x7b, 0x45, 0x5b};  unsigned char  iv[16] = {0x84, 0x31, 0x9a, 0xa8, 0xde, 0x69, 0x15, 0xca, 0x1f, 0x6b, 0xda, 0x6b, 0xfb, 0xd8, 0xc7, 0x66};  unsigned int Std\_Keystream[2] = {0x14f1c272, 0x3279c419};  int KeystreamLen = 2; //the length of key stream  unsigned int Keystream[2];  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Confidentiality validation data \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  unsigned char key[16] =  {0x17, 0x3d, 0x14, 0xba, 0x50, 0x03, 0x73, 0x1d, 0x7a, 0x60, 0x04, 0x94, 0x70, 0xf0, 0x0a, 0x29};  unsigned int COUNT = 0x66035492;  unsigned char BEARER = 0x0f;  unsigned char DIRECTION = 0x00;  unsigned int plain[7] =  {0x6cf65340, 0x735552ab, 0x0c9752fa, 0x6f9025fe, 0x0bd675d9, 0x005875b2, 0x00000000};  unsigned int Std\_cipher[7] =  {0xa6c85fc6, 0x6afb8533, 0xaafc2518, 0xdfe78494, 0x0ee1e4b0, 0x30238cc8, 0x00000000};  int plainlen = 0xc1;  unsigned int cipher[7];  //2  //unsigned char key[16] = {0xe5,0xbd,0x3e,0xa0,0xeb,0x55,0xad,0xe8,0x66,0xc6,0xac,0x58,0xbd,0x54,0x30,0x2a};  //unsigned int COUNT=0x00056823;  //unsigned char BEARER=0x18;  //unsigned char DIRECTION=0x01;  //unsigned int plain[25] = {0x14a8ef69,0x3d678507,0xbbe7270a,0x7f67ff50,0x06c3525b,0x9807e467,0xc4e56000,  // 0xba338f5d,0x42955903,0x67518222,0x46c80d3b,0x38f07f4b,0xe2d8ff58,0x05f51322,0x29bde93b,0xbbdcaf38,  // 0x2bf1ee97,0x2fbf9977,0xbada8945,0x847a2a6c,0x9ad34a66,0x7554e04d,0x1f7fa2c3,0x3241bd8f,0x01ba220d};  //unsigned int Std\_cipher[25] = {0x131d43e0,0xdea1be5c,0x5a1bfd97,0x1d852cbf,0x712d7b4f,0x57961fea,0x3208afa8,  // 0xbca433f4,0x56ad09c7,0x417e58bc,0x69cf8866,0xd1353f74,0x865e8078,0x1d202dfb,0x3ecff7fc,0xbc3b190f,  // 0xe82a204e,0xd0e350fc,0x0f6f2613,0xb2f2bca6,0xdf5a473a,0x57a4a00d,0x985ebad8,0x80d6f238,0x64a07b01};  //int plainlen = 0x0320;  //unsigned int cipher[25];  //3  //unsigned char key[16] = {0xe1,0x3f,0xed,0x21,0xb4,0x6e,0x4e,0x7e,0xc3,0x12,0x53,0xb2,0xbb,0x17,0xb3,0xe0};  //unsigned int COUNT=0x2738cdaa;  //unsigned char BEARER=0x1a;  //unsigned char DIRECTION=0x00;  //unsigned int plain[126] = {0x8d74e20d,0x54894e06,0xd3cb13cb,0x3933065e,0x8674be62,0xadb1c72b,0x3a646965,  // 0xab63cb7b,0x7854dfdc,0x27e84929,0xf49c64b8,0x72a490b1,0x3f957b64,0x827e71f4,0x1fbd4269,0xa42c97f8,  // 0x24537027,0xf86e9f4a,0xd82d1df4,0x51690fdd,0x98b6d03f,0x3a0ebe3a,0x312d6b84,0x0ba5a182,0x0b2a2c97,  // 0x09c090d2,0x45ed267c,0xf845ae41,0xfa975d33,0x33ac3009,0xfd40eba9,0xeb5b8857,0x14b768b6,0x97138baf,  // 0x21380eca,0x49f644d4,0x8689e421,0x5760b906,0x739f0d2b,0x3f091133,0xca15d981,0xcbe401ba,0xf72d05ac,  // 0xe05cccb2,0xd297f4ef,0x6a5f58d9,0x1246cfa7,0x7215b892,0xab441d52,0x78452795,0xccb7f5d7,0x9057a1c4,  // 0xf77f80d4,0x6db2033c,0xb79bedf8,0xe60551ce,0x10c667f6,0x2a97abaf,0xabbcd677,0x2018df96,0xa282ea73,  // 0x7ce2cb33,0x1211f60d,0x5354ce78,0xf9918d9c,0x206ca042,0xc9b62387,0xdd709604,0xa50af16d,0x8d35a890,  // 0x6be484cf,0x2e74a928,0x99403643,0x53249b27,0xb4c9ae29,0xeddfc7da,0x6418791a,0x4e7baa06,0x60fa6451,  // 0x1f2d685c,0xc3a5ff70,0xe0d2b742,0x92e3b8a0,0xcd6b04b1,0xc790b8ea,0xd2703708,0x540dea2f,0xc09c3da7,  // 0x70f65449,0xe84d817a,0x4f551055,0xe19ab850,0x18a0028b,0x71a144d9,0x6791e9a3,0x57793350,0x4eee0060,  // 0x340c69d2,0x74e1bf9d,0x805dcbcc,0x1a6faa97,0x6800b6ff,0x2b671dc4,0x63652fa8,0xa33ee509,0x74c1c21b,  // 0xe01eabb2,0x16743026,0x9d72ee51,0x1c9dde30,0x797c9a25,0xd86ce74f,0x5b961be5,0xfdfb6807,0x814039e7,  // 0x137636bd,0x1d7fa9e0,0x9efd2007,0x505906a5,0xac45dfde,0xed7757bb,0xee745749,0xc2963335,0x0bee0ea6,  // 0xf409df45,0x80160000};  //unsigned int Std\_cipher[126] = {0x94eaa4aa,0x30a57137,0xddf09b97,0xb25618a2,0x0a13e2f1,0x0fa5bf81,0x61a879cc,  // 0x2ae797a6,0xb4cf2d9d,0xf31debb9,0x905ccfec,0x97de605d,0x21c61ab8,0x531b7f3c,0x9da5f039,0x31f8a064,  // 0x2de48211,0xf5f52ffe,0xa10f392a,0x04766998,0x5da454a2,0x8f080961,0xa6c2b62d,0xaa17f33c,0xd60a4971,  // 0xf48d2d90,0x9394a55f,0x48117ace,0x43d708e6,0xb77d3dc4,0x6d8bc017,0xd4d1abb7,0x7b7428c0,0x42b06f2f,  // 0x99d8d07c,0x9879d996,0x00127a31,0x985f1099,0xbbd7d6c1,0x519ede8f,0x5eeb4a61,0x0b349ac0,0x1ea23506,  // 0x91756bd1,0x05c974a5,0x3eddb35d,0x1d4100b0,0x12e522ab,0x41f4c5f2,0xfde76b59,0xcb8b96d8,0x85cfe408,  // 0x0d1328a0,0xd636cc0e,0xdc05800b,0x76acca8f,0xef672084,0xd1f52a8b,0xbd8e0993,0x320992c7,0xffbae17c,  // 0x408441e0,0xee883fc8,0xa8b05e22,0xf5ff7f8d,0x1b48c74c,0x468c467a,0x028f09fd,0x7ce91109,0xa570a2d5,  // 0xc4d5f4fa,0x18c5dd3e,0x4562afe2,0x4ef77190,0x1f59af64,0x5898acef,0x088abae0,0x7e92d52e,0xb2de5504,  // 0x5bb1b7c4,0x164ef2d7,0xa6cac15e,0xeb926d7e,0xa2f08b66,0xe1f759f3,0xaee44614,0x725aa3c7,0x482b3084,  // 0x4c143ff8,0x5b53f1e5,0x83c50125,0x7dddd096,0xb81268da,0xa303f172,0x34c23335,0x41f0bb8e,0x190648c5,  // 0x807c866d,0x71932286,0x09adb948,0x686f7de2,0x94a802cc,0x38f7fe52,0x08f5ea31,0x96d0167b,0x9bdd02f0,  // 0xd2a5221c,0xa508f893,0xaf5c4b4b,0xb9f4f520,0xfd84289b,0x3dbe7e61,0x497a7e2a,0x584037ea,0x637b6981,  // 0x127174af,0x57b471df,0x4b2768fd,0x79c1540f,0xb3edf2ea,0x22cb69be,0xc0cf8d93,0x3d9c6fdd,0x645e8505,  // 0x91cca3d6,0x2c0cc000};  //int plainlen = 0x0fb3;  //unsigned int cipher[126];  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Integrity validation data \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  //1  unsigned char IK[16] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};  unsigned int counter = 0x00000000;  unsigned char bear = 0x00;  unsigned char direc = 0x00;  unsigned int message[1] = {0x00000000};  int length = 1;  unsigned int Std\_MAC = 0xc8a9595e;  //2  //unsigned char IK[16] = {0xc9,0xe6,0xce,0xc4,0x60,0x7c,0x72,0xdb,0x00,0x0a,0xef,0xa8,0x83,0x85,0xab,0x0a};  //unsigned int counter=0xa94059da;  //unsigned char bear=0x0a;  //unsigned char direc=0x01;  //unsigned int message[19] = {0x983b41d4,0x7d780c9e,0x1ad11d7e,0xb70391b1,0xde0b35da,0x2dc62f83,0xe7b78d63,  // 0x06ca0ea0,0x7e941b7b,0xe91348f9,0xfcb170e2,0x217fecd9,0x7f9f68ad,0xb16e5d7d,0x21e569d2,0x80ed775c,  // 0xebde3f40,0x93c53881,0x00000000};  //int length = 0x0241;  //unsigned int Std\_MAC=0xfae8ff0b;  //3  /\* unsigned char IK[16] =  {0x6b,0x8b,0x08,0xee,0x79,0xe0,0xb5,0x98,0x2d,0x6d,0x12,0x8e,0xa9,0xf2,0x20,0xcb};  unsigned int counter=0x561eb2dd;  unsigned char bear=0x1c;  unsigned char direc=0x00;  unsigned int message[178] =  {0x5bad7247,0x10ba1c56,0xd5a315f8,0xd40f6e09,0x3780be8e,0x8de07b69,0x92432018,    0xe08ed96a,0x5734af8b,0xad8a575d,0x3a1f162f,0x85045cc7,0x70925571,0xd9f5b94e,0x454a77c1,0x6e  72936b,    0xf016ae15,0x7499f054,0x3b5d52ca,0xa6dbeab6,0x97d2bb73,0xe41b8075,0xdce79b4b,0x86044f66,0x1d  4485a5,    0x43dd7860,0x6e0419e8,0x059859d3,0xcb2b67ce,0x0977603f,0x81ff839e,0x33185954,0x4cfbc8d0,0x0f  ef1a4c,    0x8510fb54,0x7d6b06c6,0x11ef44f1,0xbce107cf,0xa45a06aa,0xb360152b,0x28dc1ebe,0x6f7fe09b,0x05  16f9a5,    0xb02a1bd8,0x4bb0181e,0x2e89e19b,0xd8125930,0xd178682f,0x3862dc51,0xb636f04e,0x720c47c3,0xce  51ad70,    0xd94b9b22,0x55fbae90,0x6549f499,0xf8c6d399,0x47ed5e5d,0xf8e2def1,0x13253e7b,0x08d0a76b,0x6b  fc68c8,    0x12f375c7,0x9b8fe5fd,0x85976aa6,0xd46b4a23,0x39d8ae51,0x47f680fb,0xe70f978b,0x38effd7b,0x2f  7866a2,    0x2554e193,0xa94e98a6,0x8b74bd25,0xbb2b3f5f,0xb0a5fd59,0x887f9ab6,0x8159b717,0x8d5b7b67,0x7c  b546bf,    0x41eadca2,0x16fc1085,0x0128f8bd,0xef5c8d89,0xf96afa4f,0xa8b54885,0x565ed838,0xa950fee5,0xf1  c3b0a4,    0xf6fb71e5,0x4dfd169e,0x82cecc72,0x66c850e6,0x7c5ef0ba,0x960f5214,0x060e71eb,0x172a75fc,0x14  86835c,    0xbea65344,0x65b055c9,0x6a72e410,0x52241823,0x25d83041,0x4b40214d,0xaa8091d2,0xe0fb010a,0xe1  5c6de9,    0x0850973b,0xdf1e423b,0xe148a237,0xb87a0c9f,0x34d4b476,0x05b803d7,0x43a86a90,0x399a4af3,0x96  d3a120,    0x0a62f3d9,0x507962e8,0xe5bee6d3,0xda2bb3f7,0x237664ac,0x7a292823,0x900bc635,0x03b29e80,0xd6  3f6067,    0xbf8e1716,0xac25beba,0x350deb62,0xa99fe031,0x85eb4f69,0x937ecd38,0x7941fda5,0x44ba67db,0x09  117749,    0x38b01827,0xbcc69c92,0xb3f772a9,0xd2859ef0,0x03398b1f,0x6bbad7b5,0x74f7989a,0x1d10b2df,0x79  8e0dbf,    0x30d65874,0x64d24878,0xcd00c0ea,0xee8a1a0c,0xc753a279,0x79e11b41,0xdb1de3d5,0x038afaf4,0x9f  5c682c,    0x3748d8a3,0xa9ec54e6,0xa371275f,0x1683510f,0x8e4f9093,0x8f9ab6e1,0x34c2cfdf,0x4841cba8,0x8e  0cff2b,    0x0bcc8e6a,0xdcb71109,0xb5198fec,0xf1bb7e5c,0x531aca50,0xa56a8a3b,0x6de59862,0xd41fa113,0xd9  cd9578,    0x08f08571,0xd9a4bb79,0x2af271f6,0xcc6dbb8d,0xc7ec36e3,0x6be1ed30,0x8164c31c,0x7c0afc54,0x1c  000000};  int length = 0x1626;  unsigned int Std\_MAC=0x0ca12792;\*/  unsigned int MAC;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* KeyStream generation testing \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  ZUC\_GenKeyStream(k, iv, Keystream, KeystreamLen);  for (i = 0; i < KeystreamLen; i++)  {  printf("%s", "z = ");  printf("%08x\n", Keystream[i]);  }  if (memcmp(Keystream, Std\_Keystream, KeystreamLen \* 8))  return 1;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Confidentialitym testing \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*confidentiality validation\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  ZUC\_Confidentiality(key, COUNT, BEARER, DIRECTION, plain, plainlen, cipher);  printf("\nIBS:\n");  for (i = 0; i < (plainlen + 31) / 32; i++)  {  printf("%08x ", plain[i]);  }  printf("\nOBS:\n");  for (i = 0; i < (plainlen + 31) / 32; i++)  {  printf("%08x ", cipher[i]);  }  if (memcmp(cipher, Std\_cipher, (plainlen + 31) / 32))  return 1;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Integrity testing \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Integrity validation\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  MAC = ZUC\_Integrity(IK, counter, bear, direc, message, length);  printf("\nMAC = %08x ", MAC);  if (MAC != Std\_MAC)  return 1;  return 0;  }  void EEA3(u8 \*CK, u32 COUNT, u32 BEARER, u32 DIRECTION, u32 LENGTH, u32 \*M, u32 \*C)  {  u32 \*z, L, i;  u8 IV[16];  L = (LENGTH + 31) / 32;  z = (u32 \*)malloc(L \* sizeof(u32));  IV[0] = (COUNT >> 24) & 0xFF;  IV[1] = (COUNT >> 16) & 0xFF;  IV[2] = (COUNT >> 8) & 0xFF;  IV[3] = COUNT & 0xFF;  IV[4] = ((BEARER << 3) | ((DIRECTION & 1) << 2)) & 0xFC;  IV[5] = 0;  IV[6] = 0;  IV[7] = 0;  IV[8] = IV[0];  IV[9] = IV[1];  IV[10] = IV[2];  IV[11] = IV[3];  IV[12] = IV[4];  IV[13] = IV[5];  IV[14] = IV[6];  IV[15] = IV[7];  ZUC\_GenKeyStream(CK, IV, z, L);  for (i = 0; i < L; i++)  {  //printf("%X\n", z[i]);  C[i] = M[i] ^ z[i];  }  free(z);  }  /\*EEA3\_Test1 :????EEA3?? :  Key = (hex) 17 3d 14 ba 50 03 73 1d 7a 60 04 94 70 f0 0a 29  Count =(hex)66035492  Bearer = (hex) f  Direction = (hex) 0  Direction = (bin) 0  Length = 193 bits  Plaintext:(hex) 6cf65340 735552ab 0c9752fa 6f9025fe 0bd675d9 005875b2 00000000  Ciphertext:  (hex) a6c85fc6 6afb8533 aafc2518 dfe78494 0ee1e4b0 30238cc8 00000000  \*/  int main() {  int i;  printf("EEA3\n");  u8 CK[16] = {0x17,0x3d,0x14,0xba,0x50,0x03,0x73,0x1d,0x7a,0x60,0x04,0x94,0x70,0xf0,0x0a,0x29};  u32 COUNT = 0x66035492;  u32 BEARER = 0xf;  u32 DIRECTION = 0x0;  u32 LENGTH = 193;  u32 M[] = {0x6cf65340,0x735552ab,0x0c9752fa,0x6f9025fe,0x0bd675d9,0x005875b2,0x00000000};  u32 C\_real[]= {0xa6c85fc6,0x6afb8533,0xaafc2518,0xdfe78494,0x0ee1e4b0,0x30238cc8,0x00000000};  u32 C[8];  u32 D[8];  EEA3(CK, COUNT, BEARER, DIRECTION, LENGTH, M, C);  printf("A\n");  for(i = 0; i < 6; i++) {  printf("%08x,", C[i]);  }  printf("\n");  EEA3(CK, COUNT, BEARER, DIRECTION, LENGTH, C, D);  printf("B\n");  for (i = 0; i < 6; i++) {  printf("%08x,", D[i]);  }  system("pause");  } |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  File name: zuc.h  Version: V1.1  Date: Oct 28,2016  Description: This headfile provide macro defination,parameter definition and function  declaration needed in ZUC stream cipher algorithm implementation.  Function List:  1.AddMod // calculate a+b mod 2^31-1  2.PowMod // calculate x\*2^k mod 2^31-1  3.L1 // linear transformation L1:X^(X<<< 2)^(X<<<10)^(X<<<18)^(X<<<24)  4.L2 // linear transformation L2:X^(X<<< 8)^(X<<<14)^(X<<<22)^(X<<<30)  5.BitValue // test if the value of M at the position i equals 0  6.GetWord // get a 32bit word ki from bit strings k[i],k[i+1]...,  // namely ki=k[i]||k[i+1]||...||k[i+31]  7.LFSRWithInitMode // Initialisation mode,refresh the current state of LFSR  8.LFSRWithWorkMode // working mode,refresh the current state of LFSR  9.BR // Bit Reconstruction  10.F // nonlinear function  11.ZUC\_Init // Initialisation process of ZUC  12.ZUC\_Work // working stage of ZUC  13.ZUC\_GenKeyStream // generate key stream  14.ZUC\_Confidentiality // the ZUC-based condifentiality algorithm  15.ZUC\_Integrity // the ZUC-based integrity algorithm  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  typedef unsigned char u8;  typedef unsigned int u32;  unsigned char ZUC\_S0[256] =  {0x3e, 0x72, 0x5b, 0x47, 0xca, 0xe0, 0x00, 0x33, 0x04, 0xd1, 0x54, 0x98, 0x09, 0xb9, 0x6d, 0xcb,  0x7b, 0x1b, 0xf9, 0x32, 0xaf, 0x9d, 0x6a, 0xa5, 0xb8, 0x2d, 0xfc, 0x1d, 0x08, 0x53, 0x03, 0x90,  0x4d, 0x4e, 0x84, 0x99, 0xe4, 0xce, 0xd9, 0x91, 0xdd, 0xb6, 0x85, 0x48, 0x8b, 0x29, 0x6e, 0xac,  0xcd, 0xc1, 0xf8, 0x1e, 0x73, 0x43, 0x69, 0xc6, 0xb5, 0xbd, 0xfd, 0x39, 0x63, 0x20, 0xd4, 0x38,  0x76, 0x7d, 0xb2, 0xa7, 0xcf, 0xed, 0x57, 0xc5, 0xf3, 0x2c, 0xbb, 0x14, 0x21, 0x06, 0x55, 0x9b,  0xe3, 0xef, 0x5e, 0x31, 0x4f, 0x7f, 0x5a, 0xa4, 0x0d, 0x82, 0x51, 0x49, 0x5f, 0xba, 0x58, 0x1c,  0x4a, 0x16, 0xd5, 0x17, 0xa8, 0x92, 0x24, 0x1f, 0x8c, 0xff, 0xd8, 0xae, 0x2e, 0x01, 0xd3, 0xad,  0x3b, 0x4b, 0xda, 0x46, 0xeb, 0xc9, 0xde, 0x9a, 0x8f, 0x87, 0xd7, 0x3a, 0x80, 0x6f, 0x2f, 0xc8,  0xb1, 0xb4, 0x37, 0xf7, 0x0a, 0x22, 0x13, 0x28, 0x7c, 0xcc, 0x3c, 0x89, 0xc7, 0xc3, 0x96, 0x56,  0x07, 0xbf, 0x7e, 0xf0, 0x0b, 0x2b, 0x97, 0x52, 0x35, 0x41, 0x79, 0x61, 0xa6, 0x4c, 0x10, 0xfe,  0xbc, 0x26, 0x95, 0x88, 0x8a, 0xb0, 0xa3, 0xfb, 0xc0, 0x18, 0x94, 0xf2, 0xe1, 0xe5, 0xe9, 0x5d,  0xd0, 0xdc, 0x11, 0x66, 0x64, 0x5c, 0xec, 0x59, 0x42, 0x75, 0x12, 0xf5, 0x74, 0x9c, 0xaa, 0x23,  0x0e, 0x86, 0xab, 0xbe, 0x2a, 0x02, 0xe7, 0x67, 0xe6, 0x44, 0xa2, 0x6c, 0xc2, 0x93, 0x9f, 0xf1,  0xf6, 0xfa, 0x36, 0xd2, 0x50, 0x68, 0x9e, 0x62, 0x71, 0x15, 0x3d, 0xd6, 0x40, 0xc4, 0xe2, 0x0f,  0x8e, 0x83, 0x77, 0x6b, 0x25, 0x05, 0x3f, 0x0c, 0x30, 0xea, 0x70, 0xb7, 0xa1, 0xe8, 0xa9, 0x65,  0x8d, 0x27, 0x1a, 0xdb, 0x81, 0xb3, 0xa0, 0xf4, 0x45, 0x7a, 0x19, 0xdf, 0xee, 0x78, 0x34, 0x60};  unsigned char ZUC\_S1[256] =  {0x55, 0xc2, 0x63, 0x71, 0x3b, 0xc8, 0x47, 0x86, 0x9f, 0x3c, 0xda, 0x5b, 0x29, 0xaa, 0xfd, 0x77,  0x8c, 0xc5, 0x94, 0x0c, 0xa6, 0x1a, 0x13, 0x00, 0xe3, 0xa8, 0x16, 0x72, 0x40, 0xf9, 0xf8, 0x42,  0x44, 0x26, 0x68, 0x96, 0x81, 0xd9, 0x45, 0x3e, 0x10, 0x76, 0xc6, 0xa7, 0x8b, 0x39, 0x43, 0xe1,  0x3a, 0xb5, 0x56, 0x2a, 0xc0, 0x6d, 0xb3, 0x05, 0x22, 0x66, 0xbf, 0xdc, 0x0b, 0xfa, 0x62, 0x48,  0xdd, 0x20, 0x11, 0x06, 0x36, 0xc9, 0xc1, 0xcf, 0xf6, 0x27, 0x52, 0xbb, 0x69, 0xf5, 0xd4, 0x87,  0x7f, 0x84, 0x4c, 0xd2, 0x9c, 0x57, 0xa4, 0xbc, 0x4f, 0x9a, 0xdf, 0xfe, 0xd6, 0x8d, 0x7a, 0xeb,  0x2b, 0x53, 0xd8, 0x5c, 0xa1, 0x14, 0x17, 0xfb, 0x23, 0xd5, 0x7d, 0x30, 0x67, 0x73, 0x08, 0x09,  0xee, 0xb7, 0x70, 0x3f, 0x61, 0xb2, 0x19, 0x8e, 0x4e, 0xe5, 0x4b, 0x93, 0x8f, 0x5d, 0xdb, 0xa9,  0xad, 0xf1, 0xae, 0x2e, 0xcb, 0x0d, 0xfc, 0xf4, 0x2d, 0x46, 0x6e, 0x1d, 0x97, 0xe8, 0xd1, 0xe9,  0x4d, 0x37, 0xa5, 0x75, 0x5e, 0x83, 0x9e, 0xab, 0x82, 0x9d, 0xb9, 0x1c, 0xe0, 0xcd, 0x49, 0x89,  0x01, 0xb6, 0xbd, 0x58, 0x24, 0xa2, 0x5f, 0x38, 0x78, 0x99, 0x15, 0x90, 0x50, 0xb8, 0x95, 0xe4,  0xd0, 0x91, 0xc7, 0xce, 0xed, 0x0f, 0xb4, 0x6f, 0xa0, 0xcc, 0xf0, 0x02, 0x4a, 0x79, 0xc3, 0xde,  0xa3, 0xef, 0xea, 0x51, 0xe6, 0x6b, 0x18, 0xec, 0x1b, 0x2c, 0x80, 0xf7, 0x74, 0xe7, 0xff, 0x21,  0x5a, 0x6a, 0x54, 0x1e, 0x41, 0x31, 0x92, 0x35, 0xc4, 0x33, 0x07, 0x0a, 0xba, 0x7e, 0x0e, 0x34,  0x88, 0xb1, 0x98, 0x7c, 0xf3, 0x3d, 0x60, 0x6c, 0x7b, 0xca, 0xd3, 0x1f, 0x32, 0x65, 0x04, 0x28,  0x64, 0xbe, 0x85, 0x9b, 0x2f, 0x59, 0x8a, 0xd7, 0xb0, 0x25, 0xac, 0xaf, 0x12, 0x03, 0xe2, 0xf2};  //D value in key loading  unsigned int ZUC\_d[16] = {0x44D7, 0x26BC, 0x626B, 0x135E, 0x5789, 0x35E2, 0x7135, 0x09AF,  0x4D78, 0x2F13, 0x6BC4, 0x1AF1, 0x5E26, 0x3C4D, 0x789A, 0x47AC};  //rotate n bits to the left in a 32bit buffer  #define ZUC\_rotl32(x, k) (((x) << k) | ((x) >> (32 - k)))  //si = ki ¬¡ di ¬¡ ivi,in key loading  #define ZUC\_LinkToS(a, b, c) (((unsigned int)(a) << 23) | ((unsigned int)(b) << 8) | (unsigned int)(c))  unsigned int AddMod(unsigned int a, unsigned int b);  unsigned int PowMod(unsigned int x, unsigned int k);  unsigned int L1(unsigned int X);  unsigned int L2(unsigned int X);  unsigned char BitValue(unsigned int M[], unsigned int i);  unsigned int GetWord(unsigned int k[], unsigned int i);  void LFSRWithInitMode(unsigned int LFSR\_S[], unsigned int u);  void LFSRWithWorkMode(unsigned int LFSR\_S[]);  void BR(unsigned int LFSR\_S[], unsigned int BR\_X[]);  unsigned int F(unsigned int BR\_X[], unsigned int F\_R[]);  void ZUC\_Init(unsigned char k[], unsigned char iv[], unsigned int LFSR\_S[], unsigned int BR\_X[], unsigned int F\_R[]);  void ZUC\_Work(unsigned int LFSR\_S[], unsigned int BR\_X[], unsigned int F\_R[], unsigned int pKeyStream[], int KeyStreamLen);  void ZUC\_GenKeyStream(unsigned char k[], unsigned char iv[], unsigned int KeyStream[], int KeyStreamLen);  void EEA3(u8 \*CK, u32 COUNT, u32 BEARER, u32 DIRECTION, u32 LENGTH, u32 \*M, u32 \*C);  void ZUC(u8 \*k, u8 \*iv, u32 \*ks, int len);  void ZUC\_Confidentiality(unsigned char CK[], unsigned int COUNT, unsigned char BEARER, unsigned char DIRECTION, unsigned int IBS[], int LENGTH, unsigned int OBS[]);  unsigned int ZUC\_Integrity(unsigned char IK[], unsigned int COUNT, unsigned char BEARER, unsigned char DIRECTION, unsigned int M[], int LENGTH); |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*FileName:EEA3.cpp  \*Author:TFflyer  \*Version:V1.0  \*Date:2019-03-28  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include "ZUC.h"  #include <malloc.h>  #include <iostream>  #include <stdio.h>  #include <stdlib.h>  typedef unsigned char u8;  typedef unsigned int u32;  void ZUC(u8 \*k, u8 \*iv, u32 \*ks, int len)  {  ZUC\_GenKeyStream(k, iv, ks, len);  }  void EEA3(u8 \*CK, u32 COUNT, u32 BEARER, u32 DIRECTION, u32 LENGTH, u32 \*M, u32 \*C)  {  u32 \*z, L, i;  u8 IV[16];  L = (LENGTH + 31) / 32;  z = (u32 \*)malloc(L \* sizeof(u32));  IV[0] = (COUNT >> 24) & 0xFF;  IV[1] = (COUNT >> 16) & 0xFF;  IV[2] = (COUNT >> 8) & 0xFF;  IV[3] = COUNT & 0xFF;  IV[4] = ((BEARER << 3) | ((DIRECTION & 1) << 2)) & 0xFC;  IV[5] = 0;  IV[6] = 0;  IV[7] = 0;  IV[8] = IV[0];  IV[9] = IV[1];  IV[10] = IV[2];  IV[11] = IV[3];  IV[12] = IV[4];  IV[13] = IV[5];  IV[14] = IV[6];  IV[15] = IV[7];  ZUC(CK, IV, z, L);  for (i = 0; i < L; i++)  {  //printf("%X\n", z[i]);  C[i] = M[i] ^ z[i];  }  free(z);  }  /\*EEA3\_Test1 :????EEA3?? :  Key = (hex) 17 3d 14 ba 50 03 73 1d 7a 60 04 94 70 f0 0a 29  Count =(hex)66035492  Bearer = (hex) f  Direction = (hex) 0  Direction = (bin) 0  Length = 193 bits  Plaintext:(hex) 6cf65340 735552ab 0c9752fa 6f9025fe 0bd675d9 005875b2 00000000  Ciphertext:  (hex) a6c85fc6 6afb8533 aafc2518 dfe78494 0ee1e4b0 30238cc8 00000000  \*/  int main() {  int i;  printf("EE3A\n");  u8 CK[16] = {0x17,0x3d,0x14,0xba,0x50,0x03,0x73,0x1d,0x7a,0x60,0x04,0x94,0x70,0xf0,0x0a,0x29};  u32 COUNT = 0x66035492;  u32 BEARER = 0xf;  u32 DIRECTION = 0x0;  u32 LENGTH = 193;  u32 M[] = {0x6cf65340,0x735552ab,0x0c9752fa,0x6f9025fe,0x0bd675d9,0x005875b2,0x00000000};  u32 C\_real[]= {0xa6c85fc6,0x6afb8533,0xaafc2518,0xdfe78494,0x0ee1e4b0,0x30238cc8,0x00000000};  u32 C[8];  u32 D[8];  EEA3(CK, COUNT, BEARER, DIRECTION, LENGTH, M, C);  printf("A\n");  for(i = 0; i < 6; i++) {  printf("%08x,", C[i]);  }  printf("\n");  EEA3(CK, COUNT, BEARER, DIRECTION, LENGTH, C, D);  printf("B\n");  for (i = 0; i < 6; i++) {  printf("%08x,", D[i]);  }  system("pause");  } |