Code

```
#include <stdlib.h>
//#include <windows.h>
#include <time.h>
#include <stdio.h>
#include <Math.h>


typedef unsigned int uint32_t;


void initRandomizer();
uint32_t getRandomUint32(uint32_t length);

void f2m_rand(uint32_t t, uint32_t m, uint32_t *A);
void f2m_print(uint32_t t, uint32_t *A);
void f2m_assign(uint32_t t, uint32_t *A, uint32_t *B);
void f2m_and(uint32_t t, uint32_t *A, uint32_t *B, uint32_t *C);
void f2m_or(uint32_t t, uint32_t *A, uint32_t *B, uint32_t *C);
void f2m_xor(uint32_t t, uint32_t *A, uint32_t *B, uint32_t *C);
void f2m_shiftleft(uint32_t t, uint32_t *A, uint32_t *B);
void f2m_shiftright(uint32_t t, uint32_t *A, uint32_t *B);
uint32_t getbit(uint32_t U, uint32_t i);
uint32_t f2m_bitm(uint32_t t, uint32_t m, uint32_t *A);

uint32_t f2m_biti(uint32_t t,uint32_t i,uint32_t *A);
void f2m_one(uint32_t t,uint32_t *A);
void f2m_zero(uint32_t t,uint32_t *A);
uint32_t f2m_is_equal(uint32_t t,uint32_t *A,uint32_t *B);
uint32_t f2m_is_one(uint32_t t,uint32_t *A);
uint32_t f2m_is_zero(uint32_t t,uint32_t *A);
void f2m_mul(uint32_t t,uint32_t m,uint32_t *A,uint32_t *B,uint32_t *F,uint32_t
*C,uint32_t *T1,uint32_t *T2);
void f2m_inv(uint32_t t,uint32_t m,uint32_t *A,uint32_t *F,uint32_t *B,uint32_t
*U,uint32_t *V,uint32_t *T1,uint32_t *T2);
uint32_t f2m_deg(uint32_t,uint32_t,uint32_t*);
void build_f(uint32_t,uint32_t,uint32_t*);

void f2m_montgomery(uint32_t t,uint32_t m,uint32_t* k,uint32_t* xP,uint32_t* F,uint32_t*
b,uint32_t* X1,uint32_t* Z1,uint32_t* X2,
uint32_t* Z2,uint32_t* T,uint32_t* S,uint32_t* M1,uint32_t* M2);
uint32_t f2m_reconstruct_point(
       uint32_t t,
       uint32_t m,
       uint32_t* F,
       uint32_t* xP,
       uint32_t* yP,
       uint32_t* X1,
       uint32_t* Z1,
       uint32_t* X2,
       uint32_t* Z2,
       uint32_t* xQ,
       uint32_t* yQ,
       uint32_t* A1,
```

```
	uint32_t* A2,
	uint32_t* A3,
	uint32_t* A4
);
bool f2m_diffie(uint32_t t,uint32_t m,uint32_t* F,uint32_t* b,uint32_t *xP,uint32_t*
yP,uint32_t* k,uint32_t *d);

int main(int argc, char* argv[])
{
	initRandomizer();
	uint32_t m = 163;
	double x = static_cast<double>( ((double)m) / 32);
	uint32_t t = 6; //static_cast<uint32_t>(ceil(x));
	uint32_t *X1,*Z1,*X2,*Z2,*T,*S,*M1,*M2,*F,*b,*xP,*yP,*k,*xQ,*yQ,*tmpInv,*x1,*x2;
	long s = sizeof(uint32_t);
	long bytes = s*t;
	X1 = (uint32_t*) malloc(bytes);
	Z1 = (uint32_t*) malloc(bytes);
	X2 = (uint32_t*) malloc(bytes);
	Z2 = (uint32_t*) malloc(bytes);
	F = (uint32_t*) malloc(bytes);
	T = (uint32_t*) malloc(bytes);
	S = (uint32_t*) malloc(bytes);
	M1 = (uint32_t*) malloc(bytes);
	M2 = (uint32_t*) malloc(bytes);
	b = (uint32_t*) malloc(bytes);
	xP = (uint32_t*) malloc(bytes);
	yP = (uint32_t*) malloc(bytes);
	k = (uint32_t*) malloc(bytes);
	xQ = (uint32_t*) malloc(bytes);
	yQ = (uint32_t*) malloc(bytes);
	tmpInv = (uint32_t*) malloc(bytes);
	x1 = (uint32_t*) malloc(bytes);
	x2 = (uint32_t*) malloc(bytes);

	// Test-Cases
	/*
	f2m_zero(t,A);
	f2m_print(t,A);
	if(f2m_is_zero(t,A))
		printf("A is zero \n");

	f2m_one(t,A);
	f2m_print(t,A);
	if(f2m_is_one(t,A))
		printf("A is one \n");

	uint32_t bit;
	bit = f2m_biti(t,0,A);
	if(bit == 1)
		printf("Bit 1 of A is 1\n");
	else
		printf("Bit 1 of A is %u\n",bit);

	if (f2m_is_equal(t,A,A) )
		printf("A equal A\n");
	else
		printf("A not equal A ?!\n");
	*/
```

```
//build_f(t,m,F);
//uint32_t degF = f2m_deg(t,m,F);
//printf("Deg(F) = %u\n",degF);
/*
bool hasError = false;
for(uint32_t n = 0;n<10000;n++) {
	f2m_rand(t,m,xP);
	//f2m_rand(t,m,B);
	//f2m_rand(t,m,C);

	//f2m_mul(t,m,A,B,F,R1,T1,T2);
	//f2m_mul(t,m,R1,C,F,R2,T1,T2);

	//f2m_mul(t,m,B,C,F,R3,T1,T2);
	//f2m_mul(t,m,A,R3,F,R4,T1,T2);
	//
	//if(f2m_is_equal(t,R2,R4))
	//	printf("Mul works correctly\n");
	//else
	//	printf("Mul is not working correctly\n");

	f2m_inv(t,m,xP,F,S,T,X1,M1,M2);
	f2m_mul(t,m,xP,S,F,X1,M1,M2);

	if(n%100 == 0){
		printf("%u\n",n );
	}

	if(1 != f2m_is_one(t,X1))
	{
		hasError = true;
		break;
	}
}
if(!hasError)
	printf("Inv works correctly\n");
else
	printf("Inv is not working correctly\n");
*/
//f2m_assign(t,A,B);
//printf("\n\n\n");
//f2m_print(t,B);

/*

f2m_and(t,A,A,C);

printf("A&A: \n");
f2m_print(t,C);

printf("A: \n");
f2m_print(t,A);

printf("B: \n");
f2m_print(t,B);

f2m_and(t,A,B,C);

printf("A&B: \n");
```

```
	f2m_print(t,C);

	printf("A << 1: \n");
	f2m_shiftleft(t,A,B);
	f2m_print(t,B);
	uint32_t test[] = {0x1,0x0};
	f2m_print(2,&test[0]);
	for(int i=0;i<63;i++) {
		f2m_shiftleft(2,&test[0],&test[0]);
		f2m_print(2,&test[0]);
	}
	for(int i=0;i<63;i++) {
		f2m_shiftright(2,&test[0],&test[0]);
		f2m_print(2,&test[0]);
	}
	printf("0 OR 1: \n");
	test[0] = 0x0;
	test[1] = 0x0;
	uint32_t test2[] = {0x1,0x0};
	uint32_t test3[2];
	f2m_or(2,&test[0],&test2[0],&test3[0]);
	f2m_print(2,&test3[0]);

	printf("1 XOR 1: \n");
	test[0] = 0x1;
	test[1] = 0x0;
	test2[0] = 0x1;
	test2[1] = 0x0;
	f2m_xor(2,&test[0],&test2[0],&test3[0]);
	f2m_print(2,&test3[0]);
	*/

	// Prakt 4 Verify Montgomery impl.
	build_f(t,m,F);
	b[5] = 0x00000002; b[4] = 0x0A601907; b[3]=0xB8C953CA; b[2]=0x1481EB10;
b[1]=0x512F7874; b[0]=0x4A3205FD;
	xP[5] = 0x00000003; xP[4] = 0xF0EBA162; xP[3]=0x86A2D57E; xP[2]=0xA0991168;
xP[1]=0xD4994637; xP[0]=0xE8343E36;
	yP[5] = 0x00000000; yP[4] = 0xD51FBC6C; yP[3]=0x71A0094F; yP[2]=0xA2CDD545;
yP[1]=0xB11C5C0C; yP[0]=0x797324F1;
	k[5] = 0x00000001; k[4] = 0x367D5331; k[3]=0x3A3A028C; k[2]=0x0BF1F5B4;
k[1]=0x34B7C146; k[0]=0x627801B3;

	f2m_montgomery(t,m,k,xP,F,b,X1,Z1,X2,Z2,T,S,M1,M2);

	f2m_inv(t,m,Z1,F,tmpInv,T,S,M1,M2);
	f2m_mul(t,m,X1,tmpInv,F,x1,T,S);
	printf("\nx1 = X1/Z1:\n");
	f2m_print(t,x1);

	f2m_inv(t,m,Z2,F,tmpInv,T,S,M1,M2);
	f2m_mul(t,m,X2,tmpInv,F,x2,T,S);
	printf("\nx2 = X2/Z2:\n");
	f2m_print(t,x2);

	f2m_reconstruct_point(t,m,F,xP,yP,X1,Z1,X2,Z2,xQ,yQ,T,S,M1,M2);
	printf("\nxQ: ");
	f2m_print(t,xQ);
	printf("\nyQ: ");
```

```
	f2m_print(t,yQ);
	printf("\n");

	// Prakt 4

	// Prakt 5
	uint32_t *d;
	d = (uint32_t*) malloc(sizeof(uint32_t)*t);
	k = (uint32_t*) malloc(sizeof(uint32_t)*t);
	f2m_rand(t, m, d);
	f2m_rand(t, m, k);

	printf("\n Diffie: %s",(f2m_diffie(t,m,F,b,xP,yP,k,d)?"true":"false"));

	free(d);
	// Prakt 5

	getchar();
	free(X1);
	free(Z1);
	free(X2);
	free(Z2);
	free(F);
	free(b);
	free(T);
	free(S);
	free(M1);
	free(M2);
	free(k);
	free(xP);
	free(yP);
	free(xQ);
	free(yQ);
	free(tmpInv);
	free(x1);
	free(x2);
	return 0;
}

void initRandomizer()
{
	srand(static_cast<unsigned int>(time(NULL)));
}
uint32_t getRandomUint32()
{
	uint32_t resLow = rand();
	uint32_t resHigh = rand();
	uint32_t res;
	res = resHigh << 15;
	res = res | (resLow);
	return res;
}

void f2m_rand(uint32_t t, uint32_t m, uint32_t *A)
{
	for(int i=t-1;i>=0;i--)
	{
		A[i]= getRandomUint32();
	}
```

```
	uint32_t mask = 0x0;
	uint32_t unEvenBytes = m%32;
	if (unEvenBytes != 0) {
		for(uint32_t i=0;i<(unEvenBytes)-1;i++) {
			mask |= (1<<i);
		}
		A[t-1] &= mask;
	}
}

void f2m_print(uint32_t t, uint32_t *A)
{
	for(int i=t-1;i>=0;i--)
	{
		printf("%08X ",A[i]);
		//if(!(i%4))
		//	printf("\n");
	}
}

void f2m_assign(uint32_t t, uint32_t *A, uint32_t *B)
{
	for(int i=t-1;i>=0;i--)
	{
		B[i]=A[i];
	}
}
void f2m_and(uint32_t t, uint32_t *A, uint32_t *B, uint32_t *C)
{
	for(int i=t-1;i>=0;i--)
	{
		C[i] = (B[i] & A[i]);
	}
}

void f2m_or(uint32_t t, uint32_t *A, uint32_t *B, uint32_t *C)
{
	for(int i=t-1;i>=0;i--)
	{
		C[i] = (B[i] | A[i]);
	}
}

void f2m_xor(uint32_t t, uint32_t *A, uint32_t *B, uint32_t *C)
{
	for(int i=t-1;i>=0;i--)
	{
		C[i] = (B[i] ^ A[i]);
	}
}

void f2m_shiftleft(uint32_t t, uint32_t *A, uint32_t *B)
{
	for(int i=t-1;i>0;i--)
	{
		B[i] = ((A [i] << 1)| getbit(A[i-1],31));
	}
	B[0] = A [0] << 1;
```

```
}
void f2m_shiftright(uint32_t t, uint32_t *A, uint32_t *B)
{
	for(uint32_t n=0;n<(t-1);n++)
	{
		B[n] = ((A[n]>> 1)| (getbit(A[n+1],0)<<31));
	}
	B[t-1] = A[t-1]>> 1;
}

uint32_t getbit(uint32_t U, uint32_t i)
{
	if(i==0)
		return U & 0x1;
	else
		return ((U>> i)& 0x1);
}

uint32_t f2m_bitm(uint32_t t, uint32_t m, uint32_t *A)
{
	/*
	int modul = m%32;
	if(modul!=0)
		return getbit(A[t-1],(m%32)-1);
	else
		return getbit(A[t-1],31);
	*/
	return f2m_biti(t,m,A);
}

uint32_t f2m_biti(uint32_t t,uint32_t i,uint32_t *A)
{
	return getbit(A[i / 32],i % 32);
}
void f2m_one(uint32_t t,uint32_t *A)
{
	for(uint32_t i=1; i<t; i++)
	{
		A[i] = 0;
	}
	A[0]=1;
}
void f2m_zero(uint32_t t,uint32_t *A)
{
	for(uint32_t i=0; i<t; i++)
	{
		A[i] = 0;
	}
}
uint32_t f2m_is_equal(uint32_t t,uint32_t *A,uint32_t *B)
{
	for(uint32_t i=0; i<t; i++)
	{
		if(A[i] != B[i])
			return 0;
	}
	return 1;
}
uint32_t f2m_is_one(uint32_t t,uint32_t *A)
```

```
{
	for(uint32_t i=1; i<t; i++)
	{
		if(A[i] != 0)
			return 0;
	}
	if(A[0] != 1)
		return 0;
	return 1;
}
uint32_t f2m_is_zero(uint32_t t,uint32_t *A)
{
	for(uint32_t i=0; i<t; i++)
	{
		if(A[i] != 0)
			return 0;
	}
	return 1;
}
void f2m_mul(uint32_t t,uint32_t m,uint32_t *A,uint32_t *B,uint32_t *F,uint32_t
*C,uint32_t *T1,uint32_t *T2)
{
	f2m_assign(t,B,T1);
	f2m_zero(t,T2);
	for(uint32_t i=0;i<m;i++){
		if( f2m_biti(t,m,T1) == 1 )
		{
			f2m_xor(t,T1, F ,T1);
		}
		if ( f2m_biti(t,i,A) == 1)
		{
			f2m_xor(t,T2,T1,T2);
		}
		f2m_shiftleft(t,T1,T1);
	}
	f2m_assign(t,T2,C);
}

uint32_t f2m_deg(uint32_t t,uint32_t m, uint32_t* A)
{
	/*
	for(uint32_t i = t-1; i >= 0; i--)
	{
		if(A[i] != 0)
		{
			for(uint32_t x=31;x>=0;x--)
			{
				if( getbit(A[i],x) == 0x1 )
				{
					return (i*32)+x+1;
				}
			}
		}
	}
	*/
	for(uint32_t pos=m;pos>=0;pos--)
	{
		if(f2m_biti(t,pos,A) == 0x1)
		{
```

```
                        return pos;
                }
        }
        return 0;
}

void f2m_inv(uint32_t t,uint32_t m,uint32_t *A,uint32_t *F,uint32_t *B,uint32_t
*U,uint32_t *V,uint32_t *T1,uint32_t *T2)
{
        f2m_assign(t,A,U);
        f2m_assign(t,F,V);
        f2m_one(t,T1);
        f2m_zero(t,T2);
        while( !f2m_is_one(t,U) && !f2m_is_one(t,V) )
        {
                while(f2m_biti(t,0,U) == 0 &&  !f2m_is_zero(t,U) )
                {
                        f2m_shiftright(t,U,U);
                        if(f2m_biti(t,0,T1) == 0 && !f2m_is_zero(t,T1) )
                        {
                                f2m_shiftright(t,T1,T1);
                        }
                        else
                        {
                                f2m_xor(t,T1,F,T1);
                                f2m_shiftright(t,T1,T1);
                        }
                }
                while(f2m_biti(t,0,V) == 0 &&  !f2m_is_zero(t,V) )
                {
                        f2m_shiftright(t,V,V);
                        if(f2m_biti(t,0,T2) == 0 && !f2m_is_zero(t,T2) )
                        {
                                f2m_shiftright(t,T2,T2);
                        }
                        else
                        {
                                f2m_xor(t,T2,F,T2);
                                f2m_shiftright(t,T2,T2);
                        }
                }
                if(f2m_deg(t,m,U) > f2m_deg(t,m,V))
                {
                        f2m_xor(t,U,V,U);
                        f2m_xor(t,T1,T2,T1);
                }
                else
                {
                        f2m_xor(t,U,V,V);
                        f2m_xor(t,T1,T2,T2);
                }
        }
        if(f2m_is_one(t,U))
        {
                f2m_assign(t,T1,B);
        }
        else
        {
                f2m_assign(t,T2,B);
```

```
	}
}
void build_f(uint32_t t,uint32_t m,uint32_t* F)
{
	f2m_zero(t,F);
	F[5] = 1<<3;
	F[0] = 1<<7|1<<6|1<<3|1;
	//f2m_print(t,F);
}

void f2m_montgomery(uint32_t t,uint32_t m,uint32_t* k,uint32_t* xP,uint32_t* F
                                ,uint32_t* b,uint32_t* X1,uint32_t* Z1,uint32_t* X2,
uint32_t* Z2,uint32_t* T,uint32_t* S,uint32_t* M1,uint32_t* M2)
{
	f2m_one(t,X1); f2m_zero(t,Z1);                  /*1*/
	f2m_assign(t,xP,X2); f2m_one(t,Z2);             /*2*/
	for(int i = m;i >= 1; i--)                      /*3*/
	{
		if(f2m_biti(t,i-1,k) == 1)              /*4*/
		{
			f2m_mul(t,m,X1,Z2,F,T,M1,M2);   /*5*/
			f2m_mul(t,m,X2,Z1,F,S,M1,M2);   /*6*/
			f2m_mul(t,m,T,S,F,X1,M1,M2);    /*7*/
			f2m_xor(t,T,S,S);                       /*8*/
			f2m_mul(t,m,S,S,F,Z1,M1,M2);    /*9*/
			f2m_mul(t,m,xP,Z1,F,T,M1,M2);   /*10*/
			f2m_xor(t,X1,T,X1);                     /*11*/
			f2m_mul(t,m,X2,X2,F,T,M1,M2);   /*12*/
			f2m_mul(t,m,Z2,Z2,F,S,M1,M2);   /*13*/
			f2m_mul(t,m,S,T,F,Z2,M1,M2);    /*14*/
			f2m_mul(t,m,S,S,F,S,M1,M2);     /*15*/
			f2m_mul(t,m,S,b,F,S,M1,M2);     /*16*/
			f2m_mul(t,m,T,T,F,X2,M1,M2);    /*17*/
			f2m_xor(t,X2,S,X2);                     /*18*/
		}
		else                                                    /*19*/
		{
			f2m_mul(t,m,X2,Z1,F,T,M1,M2);   /*20*/
			f2m_mul(t,m,X1,Z2,F,S,M1,M2);   /*21*/
			f2m_mul(t,m,T,S,F,X2,M1,M2);    /*22*/
			f2m_xor(t,T,S,S);                       /*23*/
			f2m_mul(t,m,S,S,F,Z2,M1,M2);    /*24*/
			f2m_mul(t,m,xP,Z2,F,T,M1,M2);   /*25*/
			f2m_xor(t,X2,T,X2);                     /*26*/
			f2m_mul(t,m,X1,X1,F,T,M1,M2);   /*27*/
			f2m_mul(t,m,Z1,Z1,F,S,M1,M2);   /*28*/
			f2m_mul(t,m,S,T,F,Z1,M1,M2);    /*29*/
			f2m_mul(t,m,S,S,F,S,M1,M2);     /*30*/
			f2m_mul(t,m,S,b,F,S,M1,M2);     /*31*/
			f2m_mul(t,m,T,T,F,X1,M1,M2);    /*32*/
			f2m_xor(t,X1,S,X1);                     /*33*/
		}
	}
}
uint32_t f2m_reconstruct_point(
	uint32_t t,
	uint32_t m,
	uint32_t* F,
	uint32_t* xP,
```

```
	uint32_t* yP,
	uint32_t* X1,
	uint32_t* Z1,
	uint32_t* X2,
	uint32_t* Z2,
	uint32_t* xQ,
	uint32_t* yQ,
	uint32_t* A1,
	uint32_t* A2,
	uint32_t* A3,
	uint32_t* A4
)
{
	uint32_t step = 0;

	if( f2m_is_zero(t,Z1) == 1 )
		return 0;
	if( f2m_is_zero(t,Z2) == 1 )
	{
		f2m_assign(t,xP,xQ);
		f2m_xor(t,xP,yP,yQ);
		return 1;
	}

	uint32_t* R1 = (uint32_t*) malloc(sizeof(uint32_t) * t);
	f2m_mul(t,m,Z1,Z2,F,xQ,A1,A2);
	f2m_mul(t,m,xP,xP,F,yQ,A1,A2);
	f2m_xor(t,yQ,yP,yQ);
	f2m_mul(t,m,xQ,yQ,F,yQ,A1,A2);
	f2m_mul(t,m,xQ,xP,F,xQ,A1,A2);
	f2m_inv(t,m,xQ,F,R1,A1,A2,A3,A4);
	f2m_assign(t,R1,xQ);
	f2m_mul(t,m,Z2,xP,F,Z2,A1,A2);
	f2m_xor(t,Z2,X2,Z2);
	f2m_mul(t,m,Z1,xP,F,X2,A1,A2);
	f2m_xor(t,X2,X1,X2);
	f2m_mul(t,m,X2,Z2,F,Z2,A1,A2);
	f2m_xor(t,yQ,Z2,yQ);
	f2m_mul(t,m,yQ,xQ,F,yQ,A1,A2);
	f2m_inv(t,m,Z1,F,xQ,A1,A2,A3,A4);
	f2m_mul(t,m,xQ,X1,F,xQ,A1,A2);
	f2m_xor(t,xP,xQ,Z2);
	f2m_mul(t,m,yQ,Z2,F,yQ,A1,A2);
	f2m_xor(t,yQ,yP,yQ);

	free(R1);

	return 1;
}
bool f2m_diffie(uint32_t t,uint32_t m,uint32_t* F,uint32_t* b,uint32_t *xP,uint32_t*
yP,uint32_t* k,uint32_t *d)
{
	bool retVal = false;
	uint32_t *X1,*Z1,*X2,*Z2,*T,*S,*M1,*M2,*xQ1,*yQ1,*xQ2,*yQ2,*xTmp,*yTmp;
	X1 = (uint32_t*) malloc(sizeof(uint32_t)*t);
	Z1 = (uint32_t*) malloc(sizeof(uint32_t)*t);
	X2 = (uint32_t*) malloc(sizeof(uint32_t)*t);
	Z2 = (uint32_t*) malloc(sizeof(uint32_t)*t);
	T = (uint32_t*) malloc(sizeof(uint32_t)*t);
```

```
	S = (uint32_t*) malloc(sizeof(uint32_t)*t);
	M1 = (uint32_t*) malloc(sizeof(uint32_t)*t);
	M2 = (uint32_t*) malloc(sizeof(uint32_t)*t);

	xQ1 = (uint32_t*) malloc(sizeof(uint32_t)*t);
	yQ1 = (uint32_t*) malloc(sizeof(uint32_t)*t);
	xQ2 = (uint32_t*) malloc(sizeof(uint32_t)*t);
	yQ2 = (uint32_t*) malloc(sizeof(uint32_t)*t);
	xTmp = (uint32_t*) malloc(sizeof(uint32_t)*t);
	yTmp = (uint32_t*) malloc(sizeof(uint32_t)*t);

	f2m_montgomery(t,m,d,xP,F,b,X1,Z1,X2,Z2,T,S,M1,M2);
	f2m_reconstruct_point(t,m,F,xP,yP,X1,Z1,X2,Z2,xTmp,yTmp,T,S,M1,M2);
	f2m_montgomery(t,m,k,xTmp,F,b,X1,Z1,X2,Z2,T,S,M1,M2);
	f2m_reconstruct_point(t,m,F,xTmp,yTmp,X1,Z1,X2,Z2,xQ1,yQ1,T,S,M1,M2);

	f2m_montgomery(t,m,k,xP,F,b,X1,Z1,X2,Z2,T,S,M1,M2);
	f2m_reconstruct_point(t,m,F,xP,yP,X1,Z1,X2,Z2,xTmp,yTmp,T,S,M1,M2);
	f2m_montgomery(t,m,d,xTmp,F,b,X1,Z1,X2,Z2,T,S,M1,M2);
	f2m_reconstruct_point(t,m,F,xTmp,yTmp,X1,Z1,X2,Z2,xQ2,yQ2,T,S,M1,M2);

	if ( (1 == f2m_is_equal(t,xQ1,xQ2)) && (1 == f2m_is_equal(t,yQ1,yQ2)) )
	{
		retVal = true;
	}

	free(X1);
	free(Z1);
	free(X2);
	free(Z2);
	free(T);
	free(S);
	free(M1);
	free(M2);
	free(xQ1);
	free(yQ1);
	free(xQ2);
	free(yQ2);
	free(xTmp);
	free(yTmp);

	return retVal;
}
```

Ergebnis Diffie-Hellman