ДОДАТОК А

Код програми «Lamda»

Файл base.c

```
#include "base.h"
#include <time.h>
#include <stdio.h>
#include <string.h>


miracl* MIP;
big curve_a, curve_b;
big curve_a1, curve_a2b;
big tmp0, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7, tmp8, tmp9, tmp10, tmp11, tmp12,
tmp13, tmp14, tmp15, tmp16, tmp17, tmp18, tmp19;
point_p* ld_tmp;
point_l_projective* lamda_tmp;
point_p* table_ld[16];
point_l_projective* table_lambda[16];
csprng strrng;
csprng strrng_copy;


point_l_projective* mypoint_init()
{
	point_l_projective* p = malloc(sizeof(point_l_projective));
	p->x = mirvar(0);
	p->l = mirvar(0);
	p->z = mirvar(0);
	return p;
}

point_p* ld_point_init()
{
	point_p* p = malloc(sizeof(point_p));
	p->x = mirvar(0);
	p->y = mirvar(0);
	p->z = mirvar(0);
	return p;
}

void mypoint_kill(point_l_projective* p)
{
	mirkill(p->x);
	mirkill(p->l);
	mirkill(p->z);
	free(p);
}

void ld_point_kill(point_p* p)
{
	mirkill(p->x);
	mirkill(p->y);
	mirkill(p->z);
	free(p);
}
```

```
void mypoint_set(point_l_projective* p, big x, big y)
{
	copy(x, p->x);
	inverse2(x, p->l); //x^-1
	modmult2(p->l, y, p->l); // y/x
	add2(p->l, x, p->l); //x+y/x
	p->z->len = 1;
	p->z->w[0] = 1;
}

void ld_point_set(point_p* p, big x, big y)
{
	copy(x, p->x);
	copy(y, p->y);
	p->z->len = 1;
	p->z->w[0] = 1;
}

void mypoint_get(point_l_projective* p, big x, big y)
{
	inverse2(p->z, tmp0); //z^-1
	modmult2(p->x, tmp0, x); //x
	modmult2(p->l, tmp0, y); //x+y/x
	add2(y, x, y); // y/x
	modmult2(y, x, y); //y
}

void ld_point_get(point_p* p, big x, big y)
{
	inverse2(p->z, tmp0);
	modmult2(p->x, tmp0, x);
	modsquare2(tmp0, tmp0);
	modmult2(p->y, tmp0, y);
}

void mypoint_normalize(point_l_projective* p)
{
	inverse2(p->z, p->z);
	modmult2(p->x, p->z, p->x);
	modmult2(p->l, p->z, p->l);
	p->z->len = 1;
	p->z->w[0] = 1;
}

void ld_point_normalize(point_p* p)
{
	inverse2(p->z, p->z);
	modmult2(p->x, p->z, p->x);
	modmult2(p->y, p->z, p->y);
	modmult2(p->y, p->z, p->y);
	p->z->len = 1;
	p->z->w[0] = 1;
}

void mypoint_copy(point_l_projective* source, point_l_projective* dest)
{
	copy(source->x, dest->x);
	copy(source->l, dest->l);
	copy(source->z, dest->z);
}
```

```
void ld_point_copy(point_p* source, point_p* dest)
{
	copy(source->x, dest->x);
	copy(source->y, dest->y);
	copy(source->z, dest->z);
}


void ld_point_double(point_p* p, point_p* p2)
{
	modsquare2(p->x, tmp1);
	modsquare2(p->z, tmp2);
	modmult2(tmp1, tmp2, p2->z); //z = x2z2
	modsquare2(tmp1, tmp0);
	modsquare2(tmp2, tmp4);
	modmult2(curve_b, tmp4, tmp1);
	copy(tmp1, tmp6);
	add2(tmp0, tmp6, p2->x);
	modmult2(tmp6, p2->z, tmp5);
	modmult2(curve_a, p2->z, tmp1);
	modsquare2(p->y, tmp2);
	add2(tmp1, tmp2, tmp3);
	add2(tmp3, tmp6, tmp1);
	modmult2(p2->x, tmp1, tmp2);
	add2(tmp5, tmp2, p2->y);
}

void point_double_1mp(point_l_projective* p, point_l_projective* p2) //4m + 1mp + 4s +
5a
{
	modmult2(p->l, p->z, tmp1);
	modsquare2(p->z, tmp2);
	modsquare2(p->l, tmp3); //L^2
	add2(tmp3, tmp1, tmp5); //L^2 + L*Z
	modmult2(curve_a, tmp2, tmp4); //a*Z^2
	add2(tmp5, tmp4, tmp0); //T ready
	modmult2(p->x, p->z, tmp3); //X*Z
	modsquare2(tmp0, p2->x); //X ready
	modmult2(tmp0, tmp2, p2->z); //Z ready
	modsquare2(tmp3, tmp5); //(X*Z)^2
	add2(tmp5, p2->x, tmp3); //(X*Z)^2 + X
	modmult2(tmp0, tmp1, tmp4);
	add2(tmp4, p2->z, tmp5);
	add2(tmp3, tmp5, p2->l); //L ready
}

void ld_point_add(point_p* p1, point_p* p2, point_p* p3)
{
	modmult2(p1->x, p2->z, tmp0);
	modmult2(p2->x, p1->z, tmp1);
	modsquare2(tmp0, tmp2);
	modsquare2(tmp1, tmp3);
	add2(tmp0, tmp1, tmp4);
	add2(tmp2, tmp3, tmp5);
	modsquare2(p2->z, tmp10); //z2^2
	modmult2(p1->y, tmp10, tmp6);
	modsquare2(p1->z, tmp10); //z1^2
	modmult2(p2->y, tmp10, tmp7);
	add2(tmp6, tmp7, tmp8);
	modmult2(tmp8, tmp4, tmp9);
	modmult2(tmp5, p1->z, tmp10); //F*z1
	modmult2(tmp10, p2->z, p3->z); //z
```

```
	add2(tmp7, tmp3, tmp10); //H+D
	modmult2(tmp0, tmp10, tmp11); //A(H+D)
	add2(tmp2, tmp6, tmp10); //C+G
	modmult2(tmp1, tmp10, tmp12); //B(C+G)
	add2(tmp11, tmp12, p3->x); //x
	modmult2(tmp0, tmp9, tmp10); //A*J
	modmult2(tmp5, tmp6, tmp11); //F*G
	add2(tmp10, tmp11, tmp12); //A*J + F*G
	modmult2(tmp12, tmp5, tmp10); //(A*J + F*G)*F
	add2(tmp9, p3->z, tmp11); //J+z3
	modmult2(tmp11, p3->x, tmp12); //(J+z3)*x3
	add2(tmp10, tmp12, p3->y); //y
}

void point_add(point_l_projective* p1, point_l_projective* p2, point_l_projective* p3)
{
	modmult2(p1->l, p2->z, tmp1); //Lp*Zq
	modmult2(p2->l, p1->z, tmp2); //Lq*Zp
	add2(tmp1, tmp2, tmp5); //A
	add2(p1->l, p1->z, tmp1); //Lp + Zp
	modmult2(p1->x, p2->z, tmp0); //Xp*Zq
	modmult2(p2->x, p1->z, tmp4); //Xq*Zp
	add2(tmp0, tmp4, tmp6); //Xp*Zq + Xq*Zp
	modsquare2(tmp6, tmp6); //B
	modmult2(tmp4, tmp5, tmp4); //Xq*Zp*A
	modmult2(tmp0, tmp5, p3->x); //Xp*Zq*A  !!!
	modmult2(p3->x, tmp4, p3->x); //X  !!!
	add2(tmp4, tmp6, p3->l); //A*Xq*Zp + B  !!!
	modsquare2(p3->l, p3->l); //(A*Xq*Zp + B)^2  !!!
	modmult2(tmp5, tmp6, tmp7); //A*B
	modmult2(tmp7, p2->z, tmp7); //A*B*Zq
	modmult2(tmp1, tmp7, tmp1); //A*B*Zq * (Lp + Zp)
	add2(p3->l, tmp1, p3->l); //L  !!!
	modmult2(tmp7, p1->z, p3->z); //Z  !!!
}

void ld_point_mixed_add(point_p* p1, point_p* p2, point_p* p3)
{
	//p2 in afine coord
	modsquare2(p1->z, tmp0); //E1(tmp0) = Z_1 ^ 2
	modmult2(p2->y, tmp0, tmp1); //E(tmp1) = Y_2 * E1
	modmult2(p2->x, p1->z, tmp2); //F(tmp2) = X_2 * Z_1
	add2(p1->y, tmp1, tmp3); //A(tmp3) = Y_1 + E
	add2(p1->x, tmp2, tmp4); //B(tmp4) = X_1 + F
	modmult2(tmp4, p1->z, tmp5); //C(tmp5) = B * Z_1
	modsquare2(tmp5, p3->z); //Z = C ^ 2 !!!
	modmult2(p2->x, p3->z, tmp6); //D(tmp6) = X_2 * Z
	modsquare2(tmp3, tmp7); //G1(tmp7) = A ^ 2
	modsquare2(tmp4, tmp8); //G2(tmp8) = B ^ 2
	add2(tmp3, tmp8, tmp9); //H1(tmp9) = A + G2
	modmult2(curve_a, tmp5, tmp10); //H2(tmp10) = a * C
	add2(tmp9, tmp10, tmp11); //H(tmp11) = H1 + H2
	modmult2(tmp5, tmp11, tmp12); //I(tmp12) = C * H
	add2(tmp7, tmp12, p3->x); //X = G1 + I !!!
	add2(tmp6, p3->x, tmp13); //J(tmp13) = D + X
	modsquare2(p3->z, tmp14); //K1(tmp14) = Z ^ 2
	add2(p2->y, p2->x, tmp15); //K2(tmp15) = Y_2 + X_2
	modmult2(tmp14, tmp15, tmp16); //K(tmp16) = K1 * K2
	modmult2(tmp3, tmp5, tmp17); //L1(tmp17) = A * C
	add2(tmp17, p3->z, tmp18); //L(tmp18) = L1 + Z
	modmult2(tmp13, tmp18, tmp19); //M(tmp19) = J * L
	add2(tmp19, tmp16, p3->y); //Y = M + K !!!
```

```
}

void point_mixed_add(point_l_projective* p1, point_l_projective* p2,
point_l_projective* p3)
{
	modmult2(p2->l, p1->z, tmp0); //D(tmp0) = L_2 * Z_1
	add2(p1->l, tmp0, tmp1); //A(tmp1) = L_1 + D
	modmult2(p2->x, p1->z, tmp2); //F(tmp2) = X_2 * Z_1
	add2(p1->x, tmp2, tmp3); //B1(tmp3) = X_1 + F
	modsquare2(tmp3, tmp4); //B(tmp4) = B1 ^ 2
	modmult2(tmp1, tmp2, tmp5); //G(tmp5) = A * F
	modmult2(tmp1, tmp4, tmp6); //V(tmp6) = A * B
	modmult2(tmp1, p1->x, tmp7); //C(tmp7) = A * X_1
	modmult2(tmp7, tmp5, p3->x); //X = C * G !!!
	add2(tmp5, tmp4, tmp8); //E1(tmp8) = G + B
	modsquare2(tmp8, tmp9); //E(tmp9) = E1 ^ 2
	add2(p1->l, p1->z, tmp10); //H1(tmp10) = L_1 + Z_1
	modmult2(tmp6, tmp10, tmp11); //H(tmp11) = V * H1
	add2(tmp9, tmp11, p3->l); //L = E + H !!!
	modmult2(tmp6, p1->z, p3->z); //Z = V * Z_1 !!!
}


void binar_scalar_ld(big k, point_p* p, point_p* result)
{
	ld_point_copy(p, result);

	int i, j = k->len - 1;
	for (i = 63; ((k->w[j] >> i) & 1) == 0; --i);
	--i;

	for (; j >= 0; --j)
	{
		for (; i >= 0; --i)
		{
			ld_point_double(result, result);
			if ((k->w[j] >> i) & 1)
				ld_point_mixed_add(result, p, result);
		}
		i = 63;
	}
}

void binar_scalar_lamda(big k, point_l_projective* p, point_l_projective* result)
{
	mypoint_copy(p, result);

	int i, j = k->len - 1;
	for (i = 63; ((k->w[j] >> i) & 1) == 0; --i);
	--i;

	for (; j >= 0; --j)
	{
		for (; i >= 0; --i)
		{
			point_double_1mp(result, result);
			if ((k->w[j] >> i) & 1)
				point_mixed_add(result, p, result);
		}
		i = 63;
	}
}
```

```
void block_scalar_ld(big k, point_p* p, point_p* result) //for w-4
{
	//generating precomputable table
	ld_point_copy(p, table_ld[1]);
	for (int i = 2; i < 16; ++i)
	{
		if (i % 2)
			ld_point_mixed_add(table_ld[i - 1], p, table_ld[i]);
		else
			ld_point_double(table_ld[i / 2], table_ld[i]);
	}
	for (int i = 2; i < 16; ++i)
		ld_point_normalize(table_ld[i]);

	//multiplication itself
	int i = k->len - 1, offset;
	for (offset = 60; ((k->w[i] >> offset) & 0xF) == 0; offset -= 4);
	ld_point_copy(table_ld[(k->w[i] >> offset) & 0xF], result);
	offset -= 4;

	for (; i >= 0; --i)
	{
		for (; offset >= 0; offset -= 4)
		{
			int d = (k->w[i] >> offset) & 0xF;
			for (int t = 0; t < 4; ++t)
				ld_point_double(result, result);
			if (d != 0)
				ld_point_mixed_add(result, table_ld[d], result);
		}
		offset = 60;
	}
}

void block_scalar_lambda(big k, point_l_projective* p, point_l_projective* result)
//for w-4
{
	//generating precomputable table
	mypoint_copy(p, table_lambda[1]);
	for (int i = 2; i < 16; ++i)
	{
		if (i % 2)
			point_mixed_add(table_lambda[i - 1], p, table_lambda[i]);
		else
			point_double_1mp(table_lambda[i / 2], table_lambda[i]);
	}
	for (int i = 2; i < 16; ++i)
		mypoint_normalize(table_lambda[i]);

	//multiplication itself
	int i = k->len - 1, offset;
	for (offset = 60; ((k->w[i] >> offset) & 0xF) == 0; offset -= 4);
	mypoint_copy(table_lambda[(k->w[i] >> offset) & 0xF], result);
	offset -= 4;

	for (; i >= 0; --i)
	{
		for (; offset >= 0; offset -= 4)
		{
			int d = (k->w[i] >> offset) & 0xF;
			for (int t = 0; t < 4; ++t)
```

```
                        point_double_1mp(result, result);
                if (d != 0)
                        point_mixed_add(result, table_lambda[d], result);
            }
            offset = 60;
        }
}

void montgomery_scalar_ld(big k, point_p* p, point_p* result)
{
        ld_point_copy(p, result);
        ld_point_double(p, ld_tmp);

        int i, j = k->len - 1;
        for (i = 63; ((k->w[j] >> i) & 1) == 0; --i);
        --i;

        for (; j >= 0; --j)
        {
            for (; i >= 0; --i)
            {
                if ((k->w[j] >> i) & 1)
                {
                    ld_point_add(result, ld_tmp, result);
                    ld_point_double(ld_tmp, ld_tmp);
                }
                else
                {
                    ld_point_add(result, ld_tmp, ld_tmp);
                    ld_point_double(result, result);
                }
            }
            i = 63;
        }
}

void montgomery_scalar_lamda(big k, point_l_projective* p, point_l_projective* result)
{
        mypoint_copy(p, result);
        point_double_1mp(p, lamda_tmp);

        int i, j = k->len - 1;
        for (i = 63; ((k->w[j] >> i) & 1) == 0; --i);
        --i;

        for (; j >= 0; --j)
        {
            for (; i >= 0; --i)
            {
                if ((k->w[j] >> i) & 1)
                {
                    point_add(result, lamda_tmp, result);
                    point_double_1mp(lamda_tmp, lamda_tmp);
                }
                else
                {
                    point_add(result, lamda_tmp, lamda_tmp);
                    point_double_1mp(result, result);
                }
            }
            i = 63;
        }
```

```
}


void init_data_for_scalar(int m, int k3, int k2, int k1, const char* a_s, const char*
b_s)
{
	tmp0 = mirvar(0);
	tmp1 = mirvar(0);
	tmp2 = mirvar(0);
	tmp3 = mirvar(0);
	tmp4 = mirvar(0);
	tmp5 = mirvar(0);
	tmp6 = mirvar(0);
	tmp7 = mirvar(0);
	tmp8 = mirvar(0);
	tmp9 = mirvar(0);
	tmp10 = mirvar(0);
	tmp11 = mirvar(0);
	tmp12 = mirvar(0);
	tmp13 = mirvar(0);
	tmp14 = mirvar(0);
	tmp15 = mirvar(0);
	tmp16 = mirvar(0);
	tmp17 = mirvar(0);
	tmp18 = mirvar(0);
	tmp19 = mirvar(0);

	printf("setup curve settings... ");
	curve_a = mirvar(0);
	cinstr(curve_a, a_s);
	curve_b = mirvar(0);
	cinstr(curve_b, b_s);
	if (ecurve2_init(m, k3, k2, k1, curve_a, curve_b, TRUE, MR_PROJECTIVE))
//MR_PROJECTIVE
		printf("OK\n");
	else
		printf("FAIL\n");

	//setup precalculated data for lamda
	curve_a1 = mirvar(0);
	incr2(curve_a, 0, curve_a1);
	curve_a2b = mirvar(0);
	modsquare2(curve_a, curve_a2b);
	add2(curve_a2b, curve_b, curve_a2b);

	//setup temporary points
	ld_tmp = ld_point_init();
	lamda_tmp = mypoint_init();

	for (int i = 1; i < 16; ++i)
		table_lambda[i] = mypoint_init();
	for (int i = 1; i < 16; ++i)
		table_ld[i] = ld_point_init();
}

void free_data_for_scalar()
{
	mirkill(tmp0);
	mirkill(tmp1);
	mirkill(tmp2);
	mirkill(tmp3);
	mirkill(tmp4);
```

```
	mirkill(tmp5);
	mirkill(tmp6);
	mirkill(tmp7);
	mirkill(tmp8);
	mirkill(tmp9);
	mirkill(tmp10);
	mirkill(tmp11);
	mirkill(tmp12);
	mirkill(tmp13);
	mirkill(tmp14);
	mirkill(tmp15);
	mirkill(tmp16);
	mirkill(tmp17);
	mirkill(tmp18);
	mirkill(tmp19);

	mirkill(curve_a);
	mirkill(curve_b);

	mirkill(curve_a1);
	mirkill(curve_a2b);

	ld_point_kill(ld_tmp);
	mypoint_kill(lamda_tmp);

	for (int i = 1; i < 16; ++i)
		mypoint_kill(table_lambda[i]);
	for (int i = 1; i < 16; ++i)
		ld_point_kill(table_ld[i]);
}
```

Файл main.c

```
#include "base.h"
#include <stdio.h>

miracl* MIP;
csprng strrng;
csprng strrng_copy;

timetype COUNT_AVARAGES = 5; //250
timetype COUNT_MINIMA = 20; //100

//internal variables
big e, Fe, r, temp1, s, x, y;
point_p *sP_ld, *rQ_ld, *R_ld;
point_l_projective *sP_lambda, *rQ_lambda, *R_lambda;


int get_bit_length(big x)
{
	int len;
	for (len = 31; (x->w[x->len-1] & (1 << len)) == 0; --len);
	++len;

	len += x->len * 64;
	return len;
}

void generate_random_digit(big n, big a) //n - base point order
{
```

```
	strong_bigrand(&strrng, n, a);
}

void truncate(big x, int new_bit_len)
{
	int word_len = new_bit_len / 64 + 1;
	if (x->len < word_len)
		return;

	x->len = word_len;
	x->w[x->len - 1] &= (((long long)1) << (new_bit_len % 64)) - 1;
}

inline bool is_zero(big x)
{
	return x->len == 0 || (x->len == 1 && x->w[0] == 0);
}

void save_strrng()
{
	strrng_copy.borrow = strrng.borrow;
	strrng_copy.pool_ptr = strrng.pool_ptr;
	strrng_copy.rndptr = strrng.rndptr;
	memcpy(strrng_copy.ira, strrng.ira, NK * sizeof(mr_unsign32));
	memcpy(strrng_copy.pool, strrng.pool, MR_HASH_BYTES);
}

void load_strrng()
{
	strrng.borrow = strrng_copy.borrow;
	strrng.pool_ptr = strrng_copy.pool_ptr;
	strrng.rndptr = strrng_copy.rndptr;
	memcpy(strrng.ira, strrng_copy.ira, NK * sizeof(mr_unsign32));
	memcpy(strrng.pool, strrng_copy.pool, MR_HASH_BYTES);
}

void ld_point_negate(point_p* p)
{
	ld_point_normalize(p);
	add2(p->y, p->x, p->y);
}

void mypoint_negate(point_l_projective* p)
{
	big x = mirvar(0);
	big y = mirvar(0);
	mypoint_get(p, x, y);
	add2(y, x, y);
	mypoint_set(p, x, y);
	mirkill(x);
	mirkill(y);
}


void init_internal_variables()
{
	e = mirvar(0);
	Fe = mirvar(0);
	r = mirvar(0);
	temp1 = mirvar(0);
	s = mirvar(0);
	x = mirvar(0);
```

```
	y = mirvar(0);

	sP_ld = ld_point_init();
	rQ_ld = ld_point_init();
	R_ld = ld_point_init();

	sP_lambda = mypoint_init();
	rQ_lambda = mypoint_init();
	R_lambda = mypoint_init();
}

void free_internal_variables()
{
	mirkill(e);
	mirkill(Fe);
	mirkill(r);
	mirkill(temp1);
	mirkill(s);
	mirkill(x);
	mirkill(y);

	ld_point_kill(sP_ld);
	ld_point_kill(rQ_ld);
	ld_point_kill(R_ld);

	mypoint_kill(sP_lambda);
	mypoint_kill(rQ_lambda);
	mypoint_kill(R_lambda);
}


inline void scalar_ld(big k, point_p* p, point_p* result)
{
	binar_scalar_ld(k, p, result);
	//block_scalar_ld(k, p, result);
}

inline void scalar_lambda(big k, point_l_projective* p, point_l_projective* result)
{
	binar_scalar_lamda(k, p, result);
	//block_scalar_lambda(k, p, result);
}

void generate_private_public_keys(big n, point_p* P_ld, point_l_projective* P_lambda,
big d, point_p* Q_ld, point_l_projective* Q_lambda)
{
	do
		generate_random_digit(n, d);
	while (is_zero(d));

	scalar_ld(d, P_ld, Q_ld);
	ld_point_negate(Q_ld);

	scalar_lambda(d, P_lambda, Q_lambda);
	mypoint_negate(Q_lambda);
}


void generate_presignature_ld(big n, point_p* P, big res_e, big res_Fe)
{
	do
	{
```

```
            generate_random_digit(n, res_e);
            scalar_ld(res_e, P, R_ld);
            ld_point_get(R_ld, res_Fe, y);
      } while (is_zero(res_Fe));
}

void generate_presignature_lambda(big n, point_l_projective* P, big res_e, big res_Fe)
{
      do
      {
            generate_random_digit(n, res_e);
            scalar_lambda(res_e, P, R_lambda);
            mypoint_get(R_lambda, res_Fe, y);
      } while (is_zero(res_Fe));
}


void generate_signature_ld(int m, big n, point_p* P, big d, big h, big* res_r, big*
res_s)
{
      do
      {
            do
            {
                  generate_presignature_ld(n, P, e, Fe);
                  modmult2(h, Fe, r);
                  truncate(r, get_bit_length(n));
            } while (is_zero(r));

            multiply(d, r, temp1); //dr
            divide(temp1, n, n); //dr % n
            add(e, temp1, s); //e + dr
            divide(s, n, n); //(e + dr) % n
      } while (is_zero(s));

      *res_r = r;
      *res_s = s;
}

void generate_signature_lambda(int m, big n, point_l_projective* P, big d, big h, big*
res_r, big* res_s)
{
      do
      {
            do
            {
                  generate_presignature_lambda(n, P, e, Fe);
                  modmult2(h, Fe, r);
                  truncate(r, get_bit_length(n));
            } while (is_zero(r));

            multiply(d, r, temp1); //dr
            divide(temp1, n, n); //dr % n
            add(e, temp1, s); //e + dr
            divide(s, n, n); //(e + dr) % n
      } while (is_zero(s));

      *res_r = r;
      *res_s = s;
}

bool check_signature_ld(big r, big s, int m, big n, point_p* P, point_p* Q, big h)
```

```
{
	scalar_ld(s, P, sP_ld);
	scalar_ld(r, Q, rQ_ld);
	ld_point_add(sP_ld, rQ_ld, R_ld);
	ld_point_get(R_ld, x, y);
	modmult2(h, x, y); //y = h*Xr

	truncate(y, get_bit_length(n));
	return mr_compare(y, r) == 0;
}

bool check_signature_lambda(big r, big s, int m, big n, point_l_projective* P,
point_l_projective* Q, big h)
{
	scalar_lambda(s, P, sP_lambda);
	scalar_lambda(r, Q, rQ_lambda);
	point_add(sP_lambda, rQ_lambda, R_lambda);
	mypoint_get(R_lambda, x, y);
	modmult2(h, x, y); //y = h*Xr

	truncate(y, get_bit_length(n));
	return mr_compare(y, r) == 0;
}


void test_and_print_signature(int m, big n, point_p* P_ld, point_l_projective*
P_lambda)
{
	big pow2m = mirvar(1);
	sftbit(pow2m, m, pow2m);
	decr(pow2m, 1, pow2m);
	big sig_r = 0, sig_s = 0;
	big h = mirvar(0);
	big d = mirvar(0);
	point_p* Q_ld = ld_point_init();
	point_l_projective* Q_lambda = mypoint_init();

	timetype summ_sign_ld = 0;
	timetype summ_check_ld = 0;
	timetype summ_sign_lambda = 0;
	timetype summ_check_lambda = 0;
	int count_errors = 0;

	for (int i = 0; i < COUNT_AVARAGES; ++i)
	{
		timetype begin, end, res;
		timetype minimum_sign_miracl = (unsigned __int64)(-1);
		timetype minimum_check_miracl = (unsigned __int64)(-1);
		timetype minimum_sign_ld = (unsigned __int64)(-1);
		timetype minimum_check_ld = (unsigned __int64)(-1);
		timetype minimum_sign_lambda = (unsigned __int64)(-1);
		timetype minimum_check_lambda = (unsigned __int64)(-1);

		generate_private_public_keys(n, P_ld, P_lambda, d, Q_ld, Q_lambda);
		generate_random_digit(pow2m, h);
		save_strrng();

		for (int j = 0; j < COUNT_MINIMA; ++j)
		{
			bool sig_is_valid;

			load_strrng();
```

```
                    begin = getTSC();
                    generate_signature_ld(m, n, P_ld, d, h, &sig_r, &sig_s);
                    end = getTSC();
                    res = end - begin;
                    if (res < minimum_sign_ld)
                          minimum_sign_ld = res;

                    begin = getTSC();
                    sig_is_valid = check_signature_ld(sig_r, sig_s, m, n, P_ld, Q_ld,
h);
                    end = getTSC();
                    res = end - begin;
                    if (res < minimum_check_ld)
                          minimum_check_ld = res;
                    if (!sig_is_valid)
                          ++count_errors;

                    load_strrng();
                    begin = getTSC();
                    generate_signature_lambda(m, n, P_lambda, d, h, &sig_r, &sig_s);
                    end = getTSC();
                    res = end - begin;
                    if (res < minimum_sign_lambda)
                          minimum_sign_lambda = res;

                    begin = getTSC();
                    sig_is_valid = check_signature_lambda(sig_r, sig_s, m, n, P_lambda,
Q_lambda, h);
                    end = getTSC();
                    res = end - begin;
                    if (res < minimum_check_lambda)
                          minimum_check_lambda = res;
                    if (!sig_is_valid)
                          ++count_errors;
             }

             summ_sign_ld += minimum_sign_ld;
             summ_check_ld += minimum_check_ld;
             summ_sign_lambda += minimum_sign_lambda;
             summ_check_lambda += minimum_check_lambda;
      }

      timetype time_sign_ld = summ_sign_ld / COUNT_AVARAGES;
      timetype time_check_ld = summ_check_ld / COUNT_AVARAGES;
      timetype time_sign_lambda = summ_sign_lambda / COUNT_AVARAGES;
      timetype time_check_lambda = summ_check_lambda / COUNT_AVARAGES;
      printf("time of signing in lopez-dahab: %10lld\n", time_sign_ld);
      printf("time of check in lopez-dahab:   %10lld\n", time_check_ld);
      printf("time of signing in lambda:      %10lld\n", time_sign_lambda);
      printf("time of check in lambda:        %10lld\n", time_check_lambda);
      printf("count errors: %d\n", count_errors);

      mirkill(pow2m);
      mirkill(h);
      mirkill(d);
      ld_point_kill(Q_ld);
      mypoint_kill(Q_lambda);
}

void test_on_curve(const char* name, int m, int k3, int k2, int k1, const char* a_s,
const char* b_s, const char* gx_s, const char* gy_s, const char* n_s)
{
```

```
	printf("====================================[%s]================================
===\n", name);

	MIP = mirsys(128, 0);
	MIP->IOBASE = 16;

	char* vi = "adsfadsf";
	strong_init(&strrng, strlen(vi), vi, rand(time(0)));
	save_strrng();
	init_data_for_scalar(m, k3, k2, k1, a_s, b_s);
	init_internal_variables();

	printf("---[Testing curve settings: m=%d, k3=%d, k2=%d, k1=%d]---\n", m, k3, k2,
k1);

	big gx = mirvar(0);
	cinstr(gx, gx_s);
	big gy = mirvar(0);
	cinstr(gy, gy_s);
	big n = mirvar(0);
	cinstr(n, n_s);

	point_p* P_ld = ld_point_init();
	ld_point_set(P_ld, gx, gy);
	point_l_projective* P_lambda = mypoint_init();
	mypoint_set(P_lambda, gx, gy);

	test_and_print_signature(m, n, P_ld, P_lambda);

	ld_point_kill(P_ld);
	mypoint_kill(P_lambda);
	mirkill(n);
	mirkill(gy);
	mirkill(gx);

	free_internal_variables();
	free_data_for_scalar();
	mirexit();
	printf("============================================================================\
n");
}


int main()
{
	test_on_curve("U-163", 163, 7, 6, 3, "1",
"5FF6108462A2DC8210AB403925E638A19C1455D21",
		"", "",
		"4000000000000000000002BEC12BE2262D39BCF14D");

	test_on_curve("U-167", 167, 6, 0, 0, "1",
"6EE3CEEB230811759F20518A0930F1A4315A827DAC",
		"", "",
		"3FFFFFFFFFFFFFFFFFFFFFB12EBCC7D7F29FF7701F");

	test_on_curve("U-173", 173, 10, 2, 1, "0",
"108576C80499DB2FC16EDDF6853BBB278F6B6FB437D9",
		"", "",
		"800000000000000000000189B4E67606E3825BB2831");

	test_on_curve("U-179", 179, 4, 2, 1, "1",
"4A6E0856526436F2F88DD07A341E32D04184572BEB710",
```

```
            "", "",
            "3FFFFFFFFFFFFFFFFFFFFFFB981960435FE5AB64236EF");

      test_on_curve("U-191", 191, 9, 0, 0, "1",
"7BC86E2102902EC4D5890E8B6B4981ff27E0482750FEFC03",
            "", "",
            "4000000000000000000000069A779CAC1DABC6788F7474F");

      test_on_curve("U-233", 233, 9, 4, 1, "1",
"06973B15095675534C7CF7E64A21BD54EF5DD3B8A0326AA936ECE454D2C",
            "", "",
            "1000000000000000000000000013E974E72F8A6922031D2603CFE0D7");

      test_on_curve("B-283", 283, 12, 7, 5, "1",
"27b680ac8b8596da5a4af8a19a0303fca97fd7645309fa2a581485af6263e313b79a2f5",
            "",
            "",

      "3ffffffffffffffffffffffffffffffffef90399660fc938a90165b042a7cefadb307");

      test_on_curve("U-307", 307, 8, 4, 2, "1",
"393C7F7D53666B5054B5E6C6D3DE94F4296C0C599E2E2E241050DF18B6090BDC90186904968BB",
            "",
            "",

      "3FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC079C2F3825DA70D390FBBA588D4604022B7B7");

      test_on_curve("U-367", 367, 21, 0, 0, "1",
"43FC8AD242B0B7A6F3D1627AD5654447556B47BF6AA4A64B0C2AFE42CADAB8F93D92394C79A79755437B56
995136",
            "",
            "",

      "4000000000000000000000000000000000000000009C300B75A3FA824F22428FD28CE8812245
EF44049B2D49");

      test_on_curve("B-409", 409, 87, 0, 0, "1",

      "021a5c2c8ee9feb5c4b9a753b7b476b7fd6422ef1f3dd674761fa99d6ac27c8a9a197b272822f6cd
57a55aa4f50ae317b13545f",
            "",
            "",

      "1000000000000000000000000000000000000000000000001e2aad6a612f33307be5fa47c3c9
e052f838164cd37d9a21173");

      test_on_curve("U-431", 431, 5, 3, 1, "1",
"03CE10490F6A708FC26DFE8C3D27C4F94E690134D5BFF988D8D28AAEAEDE975936C66BAC536B18AE2DC312
CA493117DAA469C640CAF3",
            "",
            "",

      "3FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFBA3175458009A8C0A724F02F81
AA8A1FCBAF80D90C7A95110504CF");

      test_on_curve("B-571", 571, 10, 5, 2, "1",

      "2f40e7e2221f295de297117b7f3d62f5c6a97ffcb8ceff1cd6ba8ce4a9a18ad84ffabbd8efa59332
be7ad6756a66e294afd185a78ff12aa520e4de739baca0c7ffeff7f2955727a",
            "",
            "",
```

```
	"3fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffe661ce18f
f55987308059b186823851ec7dd9ca1161de93d5174d66e8382e9bb2fe84e47");

	printf("program end");
	getchar();
	return 0;
}
```

Файл base.h

```
#ifndef HEADER_BASE
#define HEADER_BASE

#include "miracl/miracl.h"


typedef unsigned long long timetype;
typedef unsigned char byte;
typedef int bool;


inline timetype getTSC()
{
	return __rdtsc();
}


typedef struct
{
	big x, l, z;
} point_l_projective;

typedef struct
{
	big x, y, z;
} point_p;


point_l_projective* mypoint_init();
point_p* ld_point_init();
void mypoint_kill(point_l_projective* p);
void ld_point_kill(point_p* p);

void mypoint_set(point_l_projective* p, big x, big y);
void ld_point_set(point_p* p, big x, big y);
void mypoint_get(point_l_projective* p, big x, big y);
void ld_point_get(point_p* p, big x, big y);
void mypoint_normalize(point_l_projective* p);
void ld_point_normalize(point_p* p);
void mypoint_copy(point_l_projective* source, point_l_projective* dest);
void ld_point_copy(point_p* source, point_p* dest);

void ld_point_double(point_p* p, point_p* p2);
void point_double_1mp(point_l_projective* p, point_l_projective* p2); //4m + 1mp + 4s +
5a
void ld_point_add(point_p* p1, point_p* p2, point_p* p3);
void point_add(point_l_projective* p1, point_l_projective* p2, point_l_projective* p3);
void ld_point_mixed_add(point_p* p1, point_p* p2, point_p* p3);
void point_mixed_add(point_l_projective* p1, point_l_projective* p2,
point_l_projective* p3);
```

```
void binar_scalar_ld(big k, point_p* p, point_p* result);
void binar_scalar_lamda(big k, point_l_projective* p, point_l_projective* result);
void block_scalar_lambda(big k, point_l_projective* p, point_l_projective* result);
//for w-4
void montgomery_scalar_ld(big k, point_p* p, point_p* result);
void montgomery_scalar_lamda(big k, point_l_projective* p, point_l_projective* result);

void init_data_for_scalar(int m, int k3, int k2, int k1, const char* a_s, const char*
b_s);
void free_data_for_scalar();

#endif
```

ДОДАТОК Б

Результати роботи програми «Lamda»

```
==================================[U-163]==================================
setup curve settings... OK
---[Testing curve settings: m=163, k3=7, k2=6, k1=3]---
time of signing in lopez-dahab:    1239732
time of check in lopez-dahab:      2431737
time of signing in lambda:         1132292
time of check in lambda:           2206252
count errors: 0
===========================================================================
==================================[U-167]==================================
setup curve settings... OK
---[Testing curve settings: m=167, k3=6, k2=0, k1=0]---
time of signing in lopez-dahab:    1159338
time of check in lopez-dahab:      2245786
time of signing in lambda:         1052332
time of check in lambda:           2039972
count errors: 0
===========================================================================
==================================[U-173]==================================
setup curve settings... OK
---[Testing curve settings: m=173, k3=10, k2=2, k1=1]---
time of signing in lopez-dahab:    1257650
time of check in lopez-dahab:      2453828
time of signing in lambda:         1143616
time of check in lambda:           2227076
count errors: 0
===========================================================================
==================================[U-179]==================================
setup curve settings... OK
---[Testing curve settings: m=179, k3=4, k2=2, k1=1]---
time of signing in lopez-dahab:    1350554
time of check in lopez-dahab:      2639552
time of signing in lambda:         1229823
time of check in lambda:           2393636
count errors: 0
===========================================================================
==================================[U-191]==================================
setup curve settings... OK
---[Testing curve settings: m=191, k3=9, k2=0, k1=0]---
time of signing in lopez-dahab:    1289266
time of check in lopez-dahab:      2526933
time of signing in lambda:         1181309
time of check in lambda:           2313069
count errors: 0
===========================================================================
==================================[U-233]==================================
setup curve settings... OK
---[Testing curve settings: m=233, k3=9, k2=4, k1=1]---
```

```
time of signing in lopez-dahab:    2180532
time of check in lopez-dahab:      4318532
time of signing in lambda:         2012492
time of check in lambda:           3970463
count errors: 0
====================================================================
===================================[U-257]==================================
setup curve settings... OK
---[Testing curve settings: m=257, k3=12, k2=0, k1=0]---
time of signing in lopez-dahab:    2738202
time of check in lopez-dahab:      5365371
time of signing in lambda:         2527436
time of check in lambda:           4961499
count errors: 281
====================================================================
===================================[B-283]==================================
setup curve settings... OK
---[Testing curve settings: m=283, k3=12, k2=7, k1=5]---
time of signing in lopez-dahab:    3264194
time of check in lopez-dahab:      6496606
time of signing in lambda:         3033849
time of check in lambda:           6019470
count errors: 0
====================================================================
===================================[U-307]==================================
setup curve settings... OK
---[Testing curve settings: m=307, k3=8, k2=4, k1=2]---
time of signing in lopez-dahab:    3609151
time of check in lopez-dahab:      7118107
time of signing in lambda:         3339337
time of check in lambda:           6579866
count errors: 0
====================================================================
===================================[U-367]==================================
setup curve settings... OK
---[Testing curve settings: m=367, k3=21, k2=0, k1=0]---
time of signing in lopez-dahab:    4917164
time of check in lopez-dahab:      9790633
time of signing in lambda:         4597206
time of check in lambda:           9163301
count errors: 0
====================================================================
===================================[B-409]==================================
setup curve settings... OK
---[Testing curve settings: m=409, k3=87, k2=0, k1=0]---
time of signing in lopez-dahab:    6765020
time of check in lopez-dahab:     13413237
time of signing in lambda:         6390594
time of check in lambda:          12667567
count errors: 0
====================================================================
===================================[U-431]==================================
setup curve settings... OK
---[Testing curve settings: m=431, k3=5, k2=3, k1=1]---
time of signing in lopez-dahab:    7707669
time of check in lopez-dahab:     15293576
```

```
time of signing in lambda:         7233704
time of check in lambda:          14347022
count errors: 0
================================================================================
===================================[B-571]======================================
setup curve settings... OK
---[Testing curve settings: m=571, k3=10, k2=5, k1=2]---
time of signing in lopez-dahab:   13552958
time of check in lopez-dahab:     26942844
time of signing in lambda:        12763705
time of check in lambda:          25345179
count errors: 0
================================================================================
program end
```

ДОДАТОК В

Публікації

УДК 004.421

ОЦЕНКА ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ ЭЛЕКТРОННОЙ ЦИФРОВОЙ ПОДПИСИ, ОСНОВАННОЙ НА ЭЛЛИПТИЧЕСКИХ КРИВЫХ, ПРИ ИСПОЛЬЗОВАНИИ λ-КООРДИНАТ

Мельникова Оксана Анатольевна

к.т.н., доцент

Назарук Роман Русланович

студент

Харьковский национальный университет радиоэлектроники

г. Харьков, Украина

Аннотация: исследовано влияние применения λ-координат на вычислительную сложность алгоритмов формирования и проверки электронной цифровой подписи (ЭЦП), реализуемой согласно требованиям стандарта ДСТУ 4145 [1]. Сравниваются теоретические и экспериментальные оценки вычислительной сложности алгоритмов формирования и проверки ЕЦП при использовании λ-координат и проективных координат Лопеза-Дахаба.

Ключевые слова: λ-координаты, проективные координаты Лопеза-Дахаба, электронная цифровая подпись, эллиптическое скалярное умножение, несимметричная криптография.

В криптографических алгоритмах, основанных на эллиптических кривых (ЭК), основной операцией является скалярное умножение точки ЭК на многоразрядное целое число (скалярный множитель). В наиболее общем виде, данная операция сводиться к последовательности выполнения сложений и

удвоений точек эллиптической кривой. В классическом аффинном представлении точка ЭК задается парой чисел (x, y). При применении этих координат для сложения и удвоения точек ЭК необходимо находить мультипликативно обратный элемент, что является вычислительно сложной задачей [2]. Поэтому различными авторами предлагалось множество более эффективных типов проективных координат, не требующих использования операции мультипликативной инверсии в ходе промежуточных вычислений. В последнее время большинство специализированных библиотек использует проективные координаты Лопеза-Дахаба и их модифицированные версии, как наименее затратные в вычислительном плане. В проективных координатах Лопеза-Дахаба точка задается 3 координатами (X, Y, Z), которые соотносятся с аффинными как $x = \frac{X}{Z}$, $y = \frac{Y}{Z^2}$ [3].

Относительно недавно был предложен новый тип координат, так называемые λ-координаты [4]. Они разделяются на λ-аффинные и λ-проективные. В λ-аффинных координатах точка задается парой чисел (x, λ), где $\lambda = x - \frac{y}{x}$, а (x, y) — традиционное аффинное представление точки ЭК. В λ-проективных координатах точка задается 3 координатами (X, L, Z), которые связаны с λ-аффинными следующим образом: $x = \frac{X}{Z}$, $\lambda = \frac{L}{Z}$ [4, с. 5].

При реализации сложения и удвоения точек ЭК используются операции умножения, возведения в квадрат и сложения элементов базового поля, например $GF(2^m)$ согласно ДСТУ 4145. При этом вычислительная сложность операций возведения в квадрат и сложения может считаться пренебрежимо малой в сравнении с вычислительной сложностью операции умножения элементов поля $GF(2^m)$, то есть эти операции можно не учитывать в упрощенных оценках. Как видно из теоретических оценок вычислительной сложности операций сложения и удвоения точек ЭК, приведенных в табл. 1 для

различных типов координат, λ-координаты являются предпочтительным вариантом.

Для получения экспериментальных оценок эффективности применения проективных λ-координат были реализованы программно алгоритмы формирования и проверки электронной цифровой подписи согласно ДСТУ 4145 [1].

Таблица 1

Упрощённые теоретические оценки сложности выполнения базовых операций над точками ЭК

Тип координат	Сложение	Удвоение	Смешанное сложение
Проективные Лопеза-Дахаба	13m	3m + 2mp	8m + 1mp
λ-проективные координаты	11m	4m + 1mp	8m

Примечания:

m — оценка вычислительной сложности операции умножения элементов базового поля $GF(2^m)$;

mp — оценка вычислительной сложности операции умножения элементов базового поля $GF(2^m)$, где одним из множителей является параметром ЭК, который может иметь вырожденное значение, то есть $mp < m$.

При оценке времени выполнения алгоритмов не учитывался этап формирования хеш-значения и приведения его к элементу базового поля. Для выполнения операции эллиптического скалярного умножения в рамках реализованных криптографических алгоритмов применен блочный метод в λ-координатах с размером блока w = 4. Ниже, в табл. 2, представлены результаты экспериментальных оценок.

Как видно из табл. 2, использование λ-координат позволяет уменьшить вычислительную сложность как алгоритма формирования, так и алгоритма проверки ЭЦП примерно на 7,5%. Потенциально этот результат может быть улучшен за счет реализации эллиптического скалярного умножения, оптимизированного именно для применения с λ-координатами.

Таблица 2

Экспериментальные оценки вычислительной сложности алгоритмов формирования и проверки подписи с использованием различных типов координат, приведены в тактах процессора[1]

№	Вариант эллиптической кривой[3]	Формирование подписи		Проверка подписи	
		λ	Лопез-Дахаб	λ	Лопез-Дахаб
1	U-163	1132292	1239732	2206252	2431737
2	U-167	1052332	1159338	2039972	2245786
3	U-173	1143616	1257650	2227076	2453828
4	U-179	1229823	1350554	2393636	2639552
5	U-191	1181309	1289266	2313069	2526933
6	U-233	2012492	2180532	3970463	4318532
7	B-283	3033849	3264194	6019470	6496606
8	U-307	3339337	3609151	6579866	7118107
9	U-367	4597206	4917164	9163301	9790633
10	B-409	6390594	6765020	12667567	13413237
11	U-431	7233704	7707669	14347022	15293576
12	B-571	12763705	13552958	25345179	26942844
Усредненные оценки[2]		92,5%	100%	92,5%	100%

Примечания:

[1] эксперименты проводились на Intel Core i7-8750H 2.2GHz;

[2] усредненные оценки приведены в %, отдельно для алгоритмов формирования и проверки ЭЦП; причем для удобства интерпретации данных за 100% приняты усредненные экспериментальные значения вычислительной сложности алгоритмов, использовавших координаты Лопеза-Дахаба;

[3] тестовые параметры для ЭК U-163, U-167, …, U-431 взяты из рекомендаций стандарта [1, с. 36-37]; а также, для большей детализации экспериментов, дополнительные варианты ЭК B-283, B-409, B-571 взяты из стандарта [5, с. 94-101].

Так, например, необходимы дальнейшие эксперименты с реализацией методов эллиптического скалярного умножения различных классов, в том числе метода Монтгомери для проективных λ-координат, методов со значительным объемом предвычислений для выполнения операций при фиксированной базовой точке (для формирования предподписей), например, вариантов comb-метода в смешанных λ-координатах, вариантов метода Шамира для реализации одновременного двукратного скалярного умножения при проверке ЭЦП и т.п.

СПИСОК ЛИТЕРАТУРЫ

1. Інформаційні технології. Криптографічний захист інформації. Цифровий підпис, що ґрунтується на еліптичних кривих. Формування та перевірка: ДСТУ 4145-2002. – [Чинний від 2003-07-01]. – Київ: Держстандарт України, 2003. – 92 с. – (Національний стандарт України).
2. Hankrson D., Lopez Hernandez J., Menezes A. Software implementation of elliptic curve cryptography over binary fields // Advances in Cryptology Crypto'99. – P. 37 – 46.
3. EFD / Ordinary genus-1 binary / short Weierstrass curves [Электронный ресурс]: Режим доступа: https://www.hyperelliptic.org/EFD/g12o/auto-shortw.html (07.04.2018).

4. Thomaz Oliveira, Julio Lopez, Diego F. Aranha, and Francisco Rodriguez-Henriquez. Lambda coordinates for binary elliptic curves. CHES «Cryptographic Hardware and Embedded Systems». 2013. №15 p.311-330.
5. Federal information processing standards publication. Digital Signature Standard: FIPS 186-4 – [Issued July 2013], 2013. – 130p.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ

МАТЕРІАЛИ 22-го МІЖНАРОДНОГО МОЛОДІЖНОГО ФОРУМУ

«РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ У XXI СТОЛІТТІ»

17 – 19 квітня 2018 р.

Том 5

КОНФЕРЕНЦІЯ «ВІРТУАЛЬНИЙ ТА ФІЗИЧНИЙ КОМП'ЮТІНГ»

Харків 2018

ОЦЕНКИ ЭФФЕКТИВНОСТИ ПРИМЕНЕНИЯ λ-КООРДИНАТ ПРИ РЕАЛИЗАЦИИ КРИПТОСИСТЕМ В ГРУППАХ ТОЧЕК ЭЛЛИПТИЧЕСКИХ КРИВЫХ НАД GF(2^m)

Назарук Р.Р.
Научный руководитель – к.т.н., Мельникова О.А.
Харьковский национальный университет радиоэлектроники
(61166, Харьков, просп. Научный,14, каф. Безопасности информационных технологий)
e-mail: 97roman@gmail.com, тел. +380503460596

In this study was reviewed ways to implement basic operations of elliptic curves over Galois fields in different coordinate systems. In particular, in the projective coordinates of Lopez-Dahab and λ-coordinates. An experimental evaluation of the operability and speed of the λ-coordinates was performed in comparison with the Lopez-Dahab coordinates. According to the results obtained, the basic operations in λ-coordinates are more effective by 3% upon addition and by 17% upon doubling.

Сейчас в шифровании, цифровой подписи, аутентификации и других сферах информационной безопасности активно применяется криптография на эллиптических кривых. В криптографии используются только эллиптические кривые над конечными полями. Это означает, что для эллиптической кривой понимается набор точек, чьи координаты принадлежат конечному полю. В данном докладе использовались кривые над двоичным расширенным полем GF(2^m), оно же поле Галуа.

Основными операциями в криптографии на эллиптических кривых являются сложение и удвоение точек. Вычислительная сложность данных операций зависит от того в каких координатах представлены точки. В докладе было рассмотрено новое представление точек – λ-координаты, базовые операции в которых, как утверждают авторы, имеют меньшую вычислительную сложность нежели в других известных на данный момент координатах, в частности проективных координаты Лопеза-Дахаба.

Λ-координаты основаны на λ-представлении точки, которое делится в свою очередь на λ-аффинное $P = (x, \lambda)$, где $\lambda = x + \frac{y}{x}$ и λ-проективное $P = (X, L, Z)$ что соответствует точке $\left(\frac{X}{Z}, \quad \frac{L}{Z}\right)$ в аффинных координатах) представления. Уравнение кривой Вейерштрасса в λ-проективных координатах имеет вид $(L2 + LZ + aZ2)X2 = X4 + bZ4$.

По теоретическим оценкам вычислительная сложность операции сложения точек составляет $I_{add} = 11I_m + 2I_s + 5I_a$, сложность удвоения и зависимости от параметров a и b составляет

$I_{double} = 4I_m + I_{m(a)} + 4I_s + 5I_a$ или
$I_{double} = 3I_m + 2I_{m(a)} + I_{m(a^2*b)} + 5I_s + 8I_a$.

Что меньше вычислительной сложности аналогичных операций для точек представленных в проективных координаты Лопеза-Дахаба, которая в свою очередь составляет $I_{add} = 13I_m + 4I_s + 9I_a$ для сложения и $I_{double} = 3I_m + I_{m(a)} + I_{m(b)} + 5I_s + 4I_a$ для удвоения.

В приведенных формулах $I_m, I_{m(a)}, \quad I_{m(b)}, \quad I_{s}, \quad I_a$ – вычислительные сложности операций умножения, умножения на параметр кривой a, умножение на параметр кривой b, возведения в квадрат и сложения элементов поля соответственно.

Как видно из выше описанных формул, выполнение сложения/удвоение точек на эллиптической в λ-проективных координатах требует меньше операций нежели в проективных координатах Лопеза-Дахаба.

Целью данного доклада было проверить правильность λ -координат на практике и сравнить скорость выполнения базовых операций над точками в сравнении с проективными координатами Лопеза-Дахаба. Для тестирования было произведено 1000 сложений и удвоений различных точек эллиптической кривой. Для всех случаев результаты операций были правильными. Так что можно сделать вывод что λ-координаты могут применяться в криптографии на эллиптических кривых.

Для замера времени выполнения операций был использован метод усреднения по минимумах. Усреднение проводилось по 1000 значений, каждое из которых являлось усреднением 10 000 результатов. Согласно экспериментально полученным результатам операция удвоения точки в λ-координатах работает быстрее на 3% нежели в проективных координатах Лопеза-Дахаба, а сложение на 17% быстрее.

Исходя из проведенного исследования было установлено что λ-координаты являются перспективным направлением развития эллиптической криптографии и могут быть достойной заменой используемым ныне проективным координатам Лопеза-Дахаба.

Список источников:

1. Thomaz Oliveira, Julio Lopez, Diego F. Aranha, and Francisco Rodriguez-Henriquez – Two is the fastest prime. – 2013. – 21c.
2. Don Johnson, Alfred Menezes, Scott Vanstone — The Elliptic Curve Digital Signature Algorithm. – 2000. – 46 c.
3. Lopez-Dahab coordinates for short Weierstrass curves [Электронный ресурс]: Режим доступа: https://hyperelliptic.org/EFD/g12o/auto-shortw-lopezdahab.html#addition-add-2005-dl
4. Lawrence Washington — Elliptic curves, Number theory and Cryptography. — 2008. — 513 с.

Черкаський державний
технологічний університет

Військова Академія Збройних Сил
Азербайджанської республіки

Університет технології і гуманітарних наук
(м. Бельсько-Бяла, Польща)

Полтавський національний технічний університет
імені Юрія Кондратюка

ПРОБЛЕМИ ІНФОРМАТИЗАЦІЇ

ТЕЗИ ДОПОВІДЕЙ П'ЯТОЇ МІЖНАРОДНОЇ
НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

13 – 15 листопада 2017 року

Черкаси – Баку – Бельсько-Бяла – Полтава – 2017

найбільш ефективні заходи реагування на них з метою їх усунення. Для досягнення мети в роботі було вдосконалено математичні моделі для визначення характеристик управління ризиками, що, на відміну від існуючих, дозволяють оцінювати параметри функціонування в умовах діючих інформаційних систем; модифіковано математичні моделі характеристик оцінювання ризиків. Проведено грунтовий аналіз існуючих аналогів та вказано на їх основні недоліки та переваги. Розроблено програмний комплекс для розрахунку ризиків інформаційної безпеки, що базується на використанні експертних оцінок та теорії нечітких множин.

6. ПЕРСПЕКТИВЫ СТАНДАРТИЗАЦИИ ЦИФРОВЫХ ПОДПИСЕЙ НА КРИВЫХ ЭДВАРДСА

к.т.н. Мельникова О.А., Джурик О.В., ХНУРЭ, Харьков

Анализ на примере RFC 8080 (2017 г), определяющего использование ключей и цифровых подписей (ЦП) DSA на эллиптических кривых (ЭК) Эдвардса для аутентификации данных DNS (DNSSEC). Применяется EdDSA из RFC 8032, варианты над полем GF(P): Ed25519 и Ed448. Открытый ключ Ed448 —456 бит, Ed25519 — 256 бит. Личные ключи, используемые для DNSSEC, не применимы в других целях для защищенности от кросс-протокольных и кросс-реализационных атак. Длина ЦП Ed448 — 912 бит, Ed25519 — 512 бит. Эквивалентный уровень безопасности Ed25519 — 128 бит, Ed448 — 224 бит. По прогнозам развития традиционной ВТ, Ed25519 считается безопасной. Ed448 предназначена для приложений со сниженными требованиями, но необходимостью защиты от аналитических атак, применимых к обычным ЭК. Оценки может ухудшить создание более эффективных криптоатак. Разработка мощного квантового компьютера угрожает безопасности обоих вариантов, как и любой криптосистемы на ЭК. Однако, сегодня кривые Эдвардса все ещё используются в ряде криптографических библиотек, протоколах и ПО.

7. АНАЛИЗ ЭФФЕКТИВНОСТИ ИСПОЛЬЗОВАНИЯ λ-КООРДИНАТ ПРИ РЕАЛИЗАЦИИ ОПЕРАЦИЙ В ГРУППАХ ТОЧЕК ЭК НАД GF(2^m)

к.т.н. Мельникова О.А., Назарук Р.Р., ХНУРЭ, Харьков

В докладе проанализировано использование различных типов координат при реализации базовых операций в группах точек эллиптических кривых (ЭК) над GF(2^m). В частности, проективных координат Лопеза-Дахаба и λ-координат. Последние основаны на λ-представлении точки, которое подразделяется на λ-аффинное представление $P = (x, \lambda)$, где $\lambda = x + y/x$, а также λ-проективное представление $P = (X, L, Z)$, соответствующее аффинной точке $(X/Z, L/Z)$. Уравнение кривой Вейерштрасса в λ-проективных координатах имеет вид: $(L^2 + LZ + aZ^2)X^2 = X^4 + bZ^4$. По теоретическим оценкам, вычислительная сложность сложения / удвоения точек ЭК в λ-проективных координатах меньше, чем в проективных Лопеза-Дахаба. Согласно полученным экспериментальным оценкам, среднее время удвоения точки в λ-координатах на 3% меньше, чем в проективных Лопеза-Дахаба, а сложения – на 17% меньше. Переход на новый тип λ-координат позволит снизить вычислительную сложность современных криптопреобразований в группах точек ЭК.

8. ПОСТРОЕНИЕ АЛГОРИТМОВ ПРИВЕДЕНИЯ ПО ФИКСИРОВАННЫМ МОДУЛЯМ $F(X) \in Z_2[X]$ ПРИ 32-БИТОВОЙ РАЗРЯДНОСТИ СЛОВ

к.т.н. Мельникова О.А., Масленникова А.О., ХНУРЭ, Харьков

В докладе рассмотрена предложенная авторами методика расчета параметров для построения эффективных алгоритмов приведения по многоразрядным фиксированным модулям $f(x) \in Z_2[x]$ при использовании вычислительной техники с 32-битовой разрядностью. Подобные модули используются в криптографических стандартах, основанных

ВІЙСЬКОВА АКАДЕМІЯ ЗБРОЙНИХ СИЛ
АЗЕРБАЙДЖАНСЬКОЇ РЕСПУБЛІКИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
"ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ"

ДП "ХАРКІВСЬКИЙ НАУКОВО-ДОСЛІДНИЙ ІНСТИТУТ
ТЕХНОЛОГІЇ МАШИНОБУДУВАННЯ"

УНІВЕРСИТЕТ МІСТА ЖИЛІНА

СУЧАСНІ НАПРЯМИ РОЗВИТКУ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЗАСОБІВ УПРАВЛІННЯ

МАТЕРІАЛИ ДЕВ'ЯТОЇ МІЖНАРОДНОЇ
НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ
11 – 12 квітня 2019 року

Баку – Харків – Жиліна – 2019

соответствовать устройство такого типа, проанализированы основные атаки и угрозы, которые применимы к аппаратным холодным кошелькам, а также оценены современные аппаратные холодные кошельки, представленные на рынке, их соответствие выдвинутым требованиям и защищенность от предполагаемых угроз и атак.

27. ГЕНЕРАЦИЯ ПРИМИТИВНЫХ ПОЛИНОМОВ СО СВОЙСТВАМИ, УМЕНЬШАЮЩИМИ ВЫЧИСЛИТЕЛЬНУЮ СЛОЖНОСТЬ ОДНОВРЕМЕННОЙ РЕДУКЦИИ

к.т.н. Мельникова О.А., Назарук Р.Р., ХНУРЭ, Харьков

Во многих криптографических алгоритмах используется умножение элементов GF(2m) по модулю примитивного (неприводимого) полинома f(t), образующего поле. Вычислительную сложность "побитовых" алгоритмов умножения по модулю можно значительно уменьшить, используя одновременную редукцию. Она эффективна, если меньшие ненулевые коэффициенты f(t) сгруппированы в младшем слове представления полинома. В этой ситуации редукция сводится к двум операциям xor, над старшим и младшим словами операнда, и нескольким дополнительным элементарным операциям. Авторами был реализован алгоритм генерации примитивных полиномов f(t) с указанными свойствами. В некоторых режимах блочных симметричных шифров умножение полиномов по модулю используется как часть операции $(g(t))^i \bmod f(t)$ при $g(t) = t$. При этом дополнительным требованием является использование значений степеней f(t), выравненных на границу размера блока, т.е. $m = 2L$ (например, $L = 128, 256, 512$ для ДСТУ 7624:2014). В результате исследований найдены примитивные триномы и пентаномы, удовлетворяющие вышеописанным требованиям.

28. СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПРИЛОЖЕНИЙ ДЛЯ УПРАВЛЕНИЯ БИБЛИОГРАФИЧЕСКОЙ ИНФОРМАЦИЕЙ

Ринас А.А., к.т.н. Иващенко Г.С., ХНУРЭ, Харьков

При работе со списками источников во время написания научных работ, актуальной проблемой остается отображение ссылок и их преобразование к различным государственным или региональным стандартам хранения библиографических описаний. Вне зависимости от используемого стандарта, библиографические описания содержат общие элементы, для хранения которых целесообразно применение специального программного обеспечения. В работе представлены результаты сравнительного анализа существующих систем для управления библиографической информацией. Рассмотрены наиболее известные приложения, такие как Zotero, Citavi и BibTeX. Среди их недостатков следует отметить невозможность автоматического импорта библиографических данных из существующих документов, содержащих списки источников. Предложено программное обеспечение, в котором пользователю доступен импорт из существующих списков литературы с преобразованием полученных данных к различным стандартам в соответствии с заранее определенными стилевыми файлами.

29. ПРЕИМУЩЕСТВА, ОБЕСПЕЧИВАЕМЫЕ СТРУКТУРОЙ GPV В РАМКАХ АЛГОРИТМА FALCON

к.т.н. Мельникова О.А., Черныш Д.И., ХНУРЭ, Харьков

Использование структуры Gentry-Peikert-Vaikuntanathan обеспечивает ряд преимуществ алгоритма FALCON. К примеру, структура GPV безопасна в модели случайного оракула при использовании модульных решеток («NTRU-SIS»). Также, было доказано, что структура GPV безопасна в модели квантового оракула. Дополнительным преимуществом является возможность преобразования существующей версии FALCON для использования в режиме восстановления сообщения. Несмотря на то, что требуется подпись двойной длины, обеспечивается полное восстановление сообщения из подписи. Возможность применения такого режима делает FALCON еще более компактным с точки зрения количест-

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ

МАТЕРІАЛИ 23-го МІЖНАРОДНОГО МОЛОДІЖНОГО ФОРУМУ

«РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ У XXI СТОЛІТТІ»

16 – 18 квітня 2019 р.

Том 5

КОНФЕРЕНЦІЯ «ВІРТУАЛЬНИЙ ТА ФІЗИЧНИЙ КОМП'ЮТИНГ»

Харків 2019

ДОСЛІДЖЕННЯ ВЛАСТИВОСТЕЙ НЕЗВІДНОСТІ ТА ПРИМІТИВНОСТІ ПОЛІНОМІВ

Назарук Р.Р.
Науковий керівник – к.т.н., доцент, Мельникова О.А.
Харківський національний університет радіоелектроніки
(61166, Харків, просп. Науки,14, каф. Безпеки інформаційних технологій,
тел. (057) 702-14-25)
e-mail: 97roman@gmail.com, факс (050) 346-05-96

Irreducible and primitive polynomials in $Z_2[x]$ are widely used in modern cryptography. For example many stream and block ciphers use such polynomials with big degrees of the form 2^k. Reduction with modulus of this type can be performed faster if smaller non-zero coefficients are placed in one computer word, mostly the lowest one. Algorithms for generating polynomials with such properties are considered in this paper. As a result, it was found 596 primitive pentanoms of power 128, 271 primitive pentanoms of power 256, and 145 primitive pentanoms of power 512 whose smaller non-zero coefficients are less than 64.

В ряді криптографічних алгоритмів і стандартів, у тому числі потокових та блокових шифрах (наприклад, ДСТУ 7624:2014 [1]), використовуються незвідні та примітивні поліноми.

У даному досліджені був проведений пошук примітивних та незвідних поліномів у кільці $Z_2[x]$. Для цього використовувався адаптований алгоритм перевірки властивостей поліному з [2]:

1. $g(x) = x$;
2. *for* $(i = 0; i < l/2; ++i)$
   2.1 $g(x) = g(x)^2 \bmod f(x)$;
   2.2 $d(x) = GCD(f(x), g(x)+x)$;
   2.3 *if* $(d(x) \neq 1)$
   ret "поліном не незвідний (і не примітивний)";
3. $T = 2^l - 1 = q_1 \times q_2 \times \ldots \times q_k$;
4. *for* $(i = 1; i \leq k; ++i)$
   4.1 $d(x) = x^{T/q_i} \bmod f(x)$;
   4.3 *if* $(d(x) == 1)$
   ret "поліном не примітивний (але незвідний)";

ret "поліном примітивний (та незвідний)";

В цьому алгоритмі: l — степінь поліному $f(x)$, який тестується, GCD — найбільший спільний дільник, q_i — прості множники числа T.

На 3 кроці алгоритму використовується факторизація великого числа T. Факторизація є складною розрахунковою задачею, однак у криптографічних алгоритмах потокових та блочних шифрів [1] використовуються поліноми, степені яких являються степенями числа 2

(наприклад, 128, 256, 512). Такі числа можна розкласти на множники за різницею квадратів ($a^2 - b^2 = (a + b) \times (a - b)$).

Наприклад, для поліному степеню $l = 128$ маємо:

$T = 2^{128} - 1 = (2^{64})^2 - 1^2 = (2^{64} + 1) \times (2^{64} - 1) = F_6 \times (2^{64} - 1)$;

де F_6 — 6-те число Ферма (F_6 = 0x42f01 × 0x3d30f19cd101), а $(2^{64} - 1)$ далі розкладується за різницею квадратів:

$(2^{64} - 1) = (2^{32})^2 - 1^2 = (2^{32} + 1) \times (2^{32} - 1) = F_5 \times (2^{32} - 1)$;

F_5 = 0x281 × 0x663d81;

$(2^{32} - 1) = (2^{16})^2 - 1^2 = (2^{16} + 1) \times (2^{16} - 1) = F_4 \times (2^{16} - 1)$;

F_4 = 0x10001;

$(2^{16} - 1) = (2^8)^2 - 1^2 = (2^8 + 1) \times (2^8 - 1) = F_3 \times (2^8 - 1)$;

F_3 = 0x101;

$(2^8 - 1) = (2^4)^2 - 1^2 = (2^4 + 1) \times (2^4 - 1) = F_2 \times (2^4 - 1)$;

F_2 = 0x11;

$(2^4 - 1) = 15$ = 0x5 × 0x3;

Дані про числа Ферма та результати їх факторизації/доказу простоти чисел взяті з [3].

В результаті маємо 9 співмножників у факторизації значення $T = 2^{128} - 1$ = 0x42f01 × 0x3d30f19cd101 × 0x281 × 0x663d81 × 0x10001 × 0x101 × 0x11 × 0x5 × 0x3.

Операції приведення за модулем $f(x)$ виконуються значно швидше, коли його менші ненульові коефіцієнти розташовані у молодшому слові, тобто на бітових позиціях менших 32 або 64, в залежності від розрядності обчислювальної техніки. Алгоритми одночасної редукції особливо ефективні, якщо степінь поліному $f(x)$ вирівняна по границі слова, тобто є степеню числа 2. У роботі сформовані варіанти пентаномів та триномів з бітовими позиціями ненульових коефіцієнтів менших 64.

За результатами дослідження було виявлено 596 примітивних пентаномів степеню 128, 271 примітивних пентаномів степеню 256, та 145 примітивних пентаномів степеню 512.

Список джерел:

1. ДСТУ 7624: 2014. Інформаційні технології. Криптографічний захист інформації. Алгоритм симетричного блокового перетворення. — Перше видання; Введ. 01.07.2015. — К.: Мінекономрозвитку України, 2015 р. — 238 с.

2. ДСТУ 4145 – 2002. Інформаційні технології. Криптографічний захист інформації. Цифровий підпис, що грунтується на еліптичних кривих. Формування та перевіряння. — Перше видання; Введ. 1.07.2003. — К.: Державний комітет України з питань технічного регулювання та споживчої політики, 2003 р. — 36 с.

3. Fermat factoring status [Електронний ресурс]: Режим доступу: http://www.prothsearch.com/fermat.html (24.01.2019).

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ

МАТЕРІАЛИ 20-го ЮВІЛЕЙНОГО
МІЖНАРОДНОГО МОЛОДІЖНОГО ФОРУМУ

**«РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ
У XXI СТОЛІТТІ»**

19 – 21 квітня 2016 р.

Том 5

**КОНФЕРЕНЦІЯ
«ВІРТУАЛЬНИЙ ТА ФІЗИЧНИЙ КОМП'ЮТІНГ»**

Харків 2016

УМЕНЬШЕНИЕ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ ПРИВЕДЕНИЯ ПО МОДУЛЮ ЗА СЧЕТ ИСПОЛЬЗОВАНИЯ ВАРИАНТОВ АЛГОРИТМА БАРРЕТТА

Лаухин Ю.Р., Назарук Р.Р.
Научный руководитель – доц. Мельникова О.А.
Харьковский национальный университет радиоэлектронники
(61166, Харьков, пр. Научный,14, каф. БИТ)
E-mail: yurii.laukhin@nure.ua, тел. +380632389020

In this study it is shown how modular multiplication with Barrett reductions over certain finite fields of characteristic 2 can be implemented efficiently without using a pre-computational phase. Analyzing of several Barrett algorithm versions is provided. Barrett reduction precomputation optimization can lower computational complexity of public key elliptic curve cryptosystems.

В традиционной несимметричной криптографии, использующей преобразования целочисленных значений, основной операцией является вычислительно сложное возведение в степень по большому модулю. Одним из подходов к уменьшению ее вычислительной сложности является создание / улучшение методов и алгоритмов, снижающих вычислительную сложность соответствующих базовых операций. А именно, умножения по модулю и возведения в квадрат по модулю. Одним из подобных подходов является использование вариантов алгоритма Барретта [1, 2].

Варианты алгоритма Барретта основываются на оценке частного x/p с помощью операций, которые либо могут быть предвычислены, либо являются вычислительно более простыми. Например, остаток r от деления x/p может быть вычислен следующим образом: $r = x - p(x/p)$. Операция деления целочисленных значений большой размерности является вычислительно сложной, но x/p может быть представлено в виде $x/2^n$. А деление на число вида 2^n может быть заменено эквивалентной операцией сдвига вправо на n бит, имеющей значительно меньшую вычислительную сложность. Деление на число 2^n может быть представлено в виде [1]:

$$r = x - p\left\lfloor \frac{\frac{2^n}{p}}{p \times \frac{2^n}{p}} \right\rfloor = x - p\left\lfloor \frac{\mu}{2^n} \right\rfloor,$$

где μ — предвычисляемая константа;
x — приводимое по модулю число;
p — модуль;
n — степень двойки.

В современной несимметричной криптографии чаще используются криптосистемы, основанные на преобразованиях в группах точек эллиптических кривых (ЭК). Для них основной операцией является скалярное умножение P = d*G, где P, G — точки эллиптической кривой, d — целочисленное значение. Данная операция может быть представлена в виде

последовательности операций сложения G+G и удвоения 2G точек ЭК. То есть, при реализации криптосистем этого типа модулярные операции используются на более "низком уровне". А именно, при модульном умножении и возведении в квадрат координат точек ЭК. Но эти операции, в отличие от традиционной целочисленной криптографии, не являются многократно повторяемыми, то есть использование обычных предвычислений является неэффективным. Соответственно, метод Барретта «напрямую» не применим, требуется уменьшение вычислительной сложности этапа предвычислений.

При этом, если ЭК определена над полем GF(p), то могут использоваться вариации ранее упомянутого метода [1] при условии, что удастся уменьшить вычислительную сложность предвычислений. Для ЭК над полем $GF(2^n)$ подобные варианты упрощенных предвычислений уже рассматриваются [3], но требуют более детального анализа вариантов реализации и экспериментальных оценок.

В докладе рассматриваются математические основы приведения по модулю методом Барретта, а также различные варианты его реализации для элементов поля $GF(2^n)$, традиционный и с предвычислениями [3]. Проводится анализ и сравнение вычислительных характеристик различных версий алгоритмов.

Оптимизированные по предвычислениям варианты алгоритма Барретта для элементов $GF(2^n)$ могут позволить уменьшить вычислительную сложность криптопреобразований в группах точек ЭК над $GF(2^n)$. А также, ускорят алгоритмы формирования криптографических параметров этих систем. Особенно эффективно их применения при проверке примитивности образующих полиномов для $GF(2^n)$. Поскольку в данной ситуации используется операция возведения в степень по модулю именно для элементов $GF(2^n)$, то есть многократно повторяются операции умножения по модулю и возведения в квадрат по модулю элементов $GF(2^n)$. Следовательно, результат предвычислений может использоваться повторно, в отличие от соответствующих операций при реализации сложений и удвоений точек ЭК.

Список литературы:

1. Barrett P. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor // Proceedings CRYPTO'86. – P. 311-323.

2. Menezes Alfred, Handbook of Applied Cryptography. / Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone // CRC Press. – 2001. – 780 p.

3. Modular Reduction in $GF(2^n)$ without Pre-computational Phase / M. Kneˇzevi´c1, K. Sakiyama, J. Fan, and I. Verbauwhede // Arithmetic of Finite Fields. – 2008. – 77-87 p.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ

МАТЕРІАЛИ 20-го ЮВІЛЕЙНОГО МІЖНАРОДНОГО МОЛОДІЖНОГО ФОРУМУ

«РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ У XXI СТОЛІТТІ»

19 – 21 квітня 2016 р.

Том 5

КОНФЕРЕНЦІЯ «ВІРТУАЛЬНИЙ ТА ФІЗИЧНИЙ КОМП'ЮТІНГ»

Харків 2016

ЗМЕНШЕННЯ ОБЧИСЛЮВАЛЬНОЇ СКЛАДНОСТІ ОПЕРАЦІЙ ЗА МОДУЛЕМ З ВИКОРИСТАННЯМ АЛГОРИТМУ МОНТГОМЕРІ

Назарук Р.Р., Лаухін Ю.Р.
Науковий керівник – доц. Мельникова О.А.
Харківський національний університет радіоелектроніки
(61166, Харків, просп. Науковий,14, каф. Безпеки інформаційних технологій)
e-mail: 97roman@gmail.com, тел. +380503460596

In this study it is shown how modular multiplication with Montgomery reductions can be used in elliptic curve cryptography for computation complexity reducing, particularly over certain finite fields of characteristic 2 (Galois fields). Experimental computational characteristics of various algorithms over fields using Montgomery reduction were compared.

Класичні криптоалгоритми з відкритим ключем отримали широке розповсюдження в засобах захисту інформації в різних інформаційно-телекомунікаційних системах. Більшість з них базується на перетвореннях над скінченими полями і основною операцією є піднесення до степеня за модулем. Зменшення обчислювальної складності даної операції зводиться до пошуку нестандартних алгоритмів піднесення до степеня (бінарний метод, блочний метод, та інш.) або (та) зменшення складності базових операцій (множення та піднесення до квадрату за модулем). І саме для зменшення складності базових операцій за модулем використовується метод Монтгомері.

При множенні Монтгомері [1] ми маємо множники a та b, менші за модуль m. Вводиться ще одне ціле число r, яке повинно бути більше m та НСД(r, m) = 1. По суті, метод замінює операції за модулем m на операції за модулем r. Зазвичай r обирають як степінь числа 2, отже приведення по модулю зводиться до простих операцій з бітовими масками. Оскільки r є степінь 2, то щоб задовольнити умову НСД, підійде будь яке непарне значення m.

Класичний алгоритм Монтгомері має наступний вигляд.

1) Знайти два цілих числа r^{-1} та m', такі що
$$rr^{-1} - mm' = 1$$
2) Перевести множники у «форму Монтгомері» домноживши їх на r (побітовий зсув вліво, якщо r – степінь 2) і привівши за модулем m
$$\bar{a} = ar \bmod m$$
$$\bar{b} = br \bmod m$$
3) Крок множення. Ця операція виконується над множниками у формі Монтгомері і результат також знаходиться у даній формі ($\bar{c} = \bar{a}\bar{b}$).

Фактично, результат це $abr \,(mod\, m)$, який обраховується наступним чином:

$$t = \bar{a}\bar{b}$$
$$u = (t + (tm' \bmod r)m)/r$$
$$if\ (u \geq m)\ \bar{c} = u - m; else\ \bar{c} = u$$

4) Зворотні обчислення для переведення результату у звичайне представлення
$$c = \bar{c}r^{-1} \bmod m,$$
c також можна обрахувати за допомогою алгоритму з кроку 3, просто замінивши значення $t = \bar{a}\bar{b}$ на $t = \bar{c}$

У сучасній несиметричній криптографії використовуються криптосистеми, які базуються на перетвореннях в групах точок еліптичних кривих. Більшість еліптичних кривих будуються над полями GF(P) та $GF(2^n)$. У першому випадку обчислення проводяться над цілими числами і до них можливе застосування вище зазначеного алгоритму Монтгомері [1]. Модифікації даного алгоритму для ЕК над полями $GF(2^n)$ вже були розглянуті [2,3]. На відміну від класичної криптографії, де основною операцією є піднесення до степені, у криптографії, яка базується на ЕК, основною операцією є множення точки на число. Тому, у більшості випадків, класичний метод Монтгомері не дасть виграшу у часі за рахунок багаторазового множення. Більш практичне застосування цей метод може знайти при перевірці примітивності поліномів, оскільки при цьому виконується піднесення до степеня.

У доповіді розглядаються математичні основи алгоритму Монтгомері, різні варіанти його реалізації , а саме: традиційний варіант над полями $GF(2^n)$ [2], варіанти зі зменшеною складністю попередніх обчислень [3]. А також, проводиться порівняльний аналіз їх обчислювальної складності.

Список літератури:

1. Montgomery, Peter L. "Modular Multiplication Without Trial Division." Mathematics of Computation 4A, 170 (April 1985), 519-521.
2. Koc, C.K., Acar, T.: Montgomery multiplication in $GF(2^k)$. Designs, Codes and Cryptography 14, 57–69 (1998)
3. Modular Reduction in $GF(2^n)$ without Pre-computational Phase / M. Kneˇzevi´c1, K. Sakiyama, J. Fan, and I. Verbauwhede // Arithmetic of Finite Fields. – 2008. – 77-87 p.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ

МАТЕРІАЛИ 21-го МІЖНАРОДНОГО МОЛОДІЖНОГО ФОРУМУ

«РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ У XXI СТОЛІТТІ»

25 – 27 квітня 2017 р.

Том 5

КОНФЕРЕНЦІЯ «ВІРТУАЛЬНИЙ ТА ФІЗИЧНИЙ КОМП'ЮТІНГ»

Харків 2017

АНАЛІЗ АТАК НА ГЕШ-ФУНКЦІЮ АЛГОРИТМУ ЕЛЕКТРОННОГО ЦИФРОВОГО ПІДПИСУ

Назарук Р.Р., Лаухін Ю.Р.
Науковий керівник – к.т.н., доцент Гріненко Т.О.
Харківський національний університет радіоелектроніки
(61166, Харків, просп. Науковий,14, каф. Безпеки інформаційних технологій)
e-mail: 97roman@gmail.com, тел. +380503460596

In this study it was reviewed a type of attack on digital signature – an attack on a hash function. Were analyzed a few possible attacks on the hash function and given hash cracking time estimate depending on the hash bit length. Based on this research and the current pace of growth in computing machines it is recommended to use a long hash of at least 256 bits.

Надійність цифрового підпису визначається стійкістю до крипто аналітичних атак двох її компонентів: геш-функції та самого алгоритму ЕЦП. Стійка схема цифрового підпису повинна використовувати геш-функцію, яка відповідає таким вимогам [1]:

1. Однобічність. Нехай дано геш-значення H(M) деякого невідомого повідомлення M. Розрахунково неможливо визначити M по наявному H(M).

2. Стійкість до колізій. Нехай дано повідомлення M і його геш-значення H(M). Розрахунково неможливо визначити M' таке, що H(M)=H(M'). Ця властивість еквівалентна властивості однобічності.

3. Сувора стійкість до колізій. Розрахунково неможливо знайти два довільних повідомлення M та M', для яких H(M)=H(M').

Оцінимо ймовірність взлому геш-функції.

Для "лобової" атаки на однобічні геш-функції використовують два методи. Перший направлений на взлом другої властивості, тобто по відомому значенню геш-функції H(M) зловмисник хоче створити інший документ M', такий, що H(M)=H(M'). Другий метод направлений на взлом третьої властивості тобто зловмисник хоче знайти два випадкових повідомлення M та M', таких, що H(M)=H(M').

Припустимо, що однобічна геш-функція надійна і найкращий метод її розкриття – "в лоб". Вихід даної геш-функції – послідовність з m біт. Тоді кількість вихідних значень геш-функції H становить n=2m.

Позначимо P(n, k) – ймовірність, що для конкретного значення X і хоча б для одного Yi із значень Y1…Yk виконується рівність H(X)=H(Y).

Знайдемо значення k, при якому ймовірність P(n, k)≥0,5.

Для одного значення Y ймовірність того, що H(X)=H(Y), становить 1/n. Відповідно ймовірність того, що H(X)≠H(Y), становить 1-1/n. Для k значень імовірність того, що не буде жодного збігу становить $(1-1/n)^k$, відповідно ймовірність щонайменше одного збігу становить P(n, k)=1-$(1-1/n)^k$. При P(n, k)=0,5 отримаємо $k=n/2=2^{m-1}$.

Таким чином, для знаходження повідомлення, яке має даний геш, знадобилося б загешувати 2^{m-1} випадкових повідомлень. Імовірність взлому геш-функції при цьому становить $P=1/2^{m-1}=2^{1-m}$.

За допомогою аналогічних розрахунків знайдемо кількість повідомлень, при яких ймовірність появи двух однакових повідомлень буде більше 0,5, вона становить $P=2^{-m/2}$ [1, 2].

Наведемо оцінку, наскільки успішними можуть бути атаки, які базуються на двох описаних вище методах. Нехай одна MIPS (Million Instruction Per Second) машина обчислює мільйон геш-значень за секунду. У табл. 1 наведено оцінку ймовірності взлому геш-функції для розглянутих методів атаки при різних значеннях довжини гешу.

Таблиця 1 – Час атаки на геш-функцію для різних довжин гешів.

Довжина гешу, m (біт)	Перший метод		Другий метод	
	Ймовірність взлому, P	Тривалість взлому, MIPS-років	Ймовірність взлому, P	Тривалість взлому, MIPS-років
64	$1,08 * 10^{-19}$	300000	$2,33 * 10^{-10}$	1,19 годин
128	$5,88 * 10^{-39}$	$5,4 * 10^{24}$	$5,42 * 10^{-20}$	600000
256	$1,73 * 10^{-77}$	$1,8 * 10^{63}$	$2,94 * 10^{-39}$	$1,1 * 10^{25}$
512	$1,49 * 10^{-154}$	$2,1 * 10^{140}$	$8,64 * 10^{-78}$	$3,7 * 10^{63}$

Щоб забезпечити необхідну ймовірність взлому геш-функції, необхідно використовувати більшу довжину геш-значення. Так, наприклад, для забезпечення ймовірності взлому не більш ніж 10^{-30} необхідно використовувати 256-бітне значення гешу.

Отже, одним із варіантів атак на ЕЦП є атака на геш-функцію. При реалізації відповідного алгоритму геш-функцію слід обирати зважаючи на важливість даних, які будуть підписуватися, та час, який вони будуть актуальними. Враховуючи сучасні темпи росту обчислювальної здатності комп'ютерних систем не рекомендується обирати геш-функції з довжиною геша менше 256 біт.

Список літератури:

1. Горбенко Ю.І. Побудування та аналіз систем, протоколів і засобів криптографічного захисту інформації: монографія. – Частина 1: Методи побудування та аналізу, стандартизація та застосування криптографічних систем /За заг. ред. д.т.н., професора І.Д. Горбенко / Ю.І. Горбенко // Харків, Видавництво «Форт», 2016. – 960 с.

2. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. – М.: ТРИУМФ, 2002 – 816 с.

3. Методи захисту. Криптографічні методи, що ґрунтуються на еліптичних кривих. Частина 2: Цифрові підписи (ISO/IEC 15946–2:2002, IDT) – [Чинний від 2002–12–05] . – Київ : Держспоживстандарт України, 2002. – 58 с. – (Національний стандарт України).

№ док.	*Позначення*	*Найменування*	*Дод. відом.*
1		Текстові документи	
	ГЮІК.ХХХХХХ.1649Ст.10ПЗ	Пояснювальна записка	96 с.
2		Графічні матеріали	
		Презентаційний матеріал	15 слайдів
3		Інші документи	
		Рецензія	1 с.
		Відгук керівника	1 с.
		Електрона версія ПЗ	1 шт.

Зм.	Арк.	№ докум.	Підпис	Дата	ГЮІК.ХХХХХХ.1649Ст.10ПЗ		
Розробив		Назарук Р.Р.			Електронний цифровий підпис на основі λ-координат	Літ.	Арк. 1 / Арк. 1
Перевірив.		Мельникова О.А.					
Н. контр.		Конєва Н.Ф.				ХНУРЕ Кафедра БІТ	
Затвердив		Халімов Г.З.					