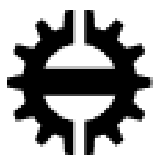
Faster Binary Curve Software: A Case Study

NordSec 2015, Stockholm

B. B. Brumley

Department of Pervasive Computing
Tampere University of Technology, Finland
billy.brumley AT tut.fi

20 Oct 2015



TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Elliptic curves over binary fields

Fix m and consider all of the (x, y) solutions over F_{2^m} to the following equation:

$$E : y^2 + xy = x^3 + a_2x^2 + a_6$$

Standardized curves

```
$ openssl ecparam -list_curves
...
  sect163k1 : NIST/SECG/WTLS curve over a 163 bit binary field
  sect163r2 : NIST/SECG curve over a 163 bit binary field
  sect233k1 : NIST/SECG/WTLS curve over a 233 bit binary field
  sect233r1 : NIST/SECG/WTLS curve over a 233 bit binary field
  sect239k1 : SECG curve over a 239 bit binary field
  sect283k1 : NIST/SECG curve over a 283 bit binary field
  sect283r1 : NIST/SECG curve over a 283 bit binary field
  sect409k1 : NIST/SECG curve over a 409 bit binary field
  sect409r1 : NIST/SECG curve over a 409 bit binary field
  sect571k1 : NIST/SECG curve over a 571 bit binary field
  sect571r1 : NIST/SECG curve over a 571 bit binary field
...
```

Carryless multiplication

projects / openssl.git / commit

summary | shortlog | log | commit | commitdiff | tree
(parent: b5c6aab) | patch

x86_64 assembler pack: add x86_64-gf2m module.

author	Andy Polyakov <appro@openssl.org>	
	Mon, 16 May 2011 20:46:45 +0300 (17:46 +0000)	
committer	Andy Polyakov <appro@openssl.org>	
	Mon, 16 May 2011 20:46:45 +0300 (17:46 +0000)	
commit	afebe623c52d2067d1c34d4461ee92924371621d	
tree	96e677f78ce827b6af6f288c6025919e60e0623e	tree | snapshot
parent	b5c6aab57e71015c00d5149f1cf6d03dab1d67ec	commit | diff

x86_64 assembler pack: add x86_64-gf2m module.

projects / openssl.git / commit

summary | shortlog | log | commit | commitdiff | tree
(parent: 3514154) | patch

Add ARMv8 Montgomery multiplication module.

author	Andy Polyakov <appro@openssl.org>	
	Sat, 21 Mar 2015 15:54:17 +0300 (13:54 +0100)	
committer	Andy Polyakov <appro@openssl.org>	
	Mon, 20 Apr 2015 15:39:34 +0300 (14:39 +0200)	
commit	cb2ed545828065e4099137ba3e3568328348f473	
tree	ac7ef190bdfc1572c3beb1e9f52b674d8636533f	tree | snapshot
parent	35141544e2994f0f3b87be7d7c9a43ea3cd9840a	commit | diff

Add ARMv8 Montgomery multiplication module.

Reviewed-by: Rich Salz <rsalz@openssl.org>

Point multiplication

Algorithm 3.36 Window NAF method for point multiplication

INPUT: Window width w, positive integer k, $P \in E(\mathbb{F}_q)$.
OUTPUT: kP.

1. Use Algorithm 3.35 to compute $\mathrm{NAF}_w(k) = \sum_{i=0}^{l-1} k_i 2^i$,
2. Compute $P_i = iP$ for $i \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$.
3. $Q \leftarrow \infty$.
4. For i from $l-1$ downto 0 do
   4.1 $Q \leftarrow 2Q$.
   4.2 If $k_i \neq 0$ then:
   If $k_i > 0$ then $Q \leftarrow Q + P_{k_i}$;
   Else $Q \leftarrow Q - P_{-k_i}$.
5. Return(Q).

Affine coordinates

OpenSSL implements curve operations as written in P1363.

Addition

Let $P = (x_1, y_1)$, $Q = (x_2, y_2)$ such that $P \neq \pm Q$. Then $P + Q = (x_3, y_3)$ is given by

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a_2$$
$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1$$
$$\lambda = \frac{y_1 + y_2}{x_1 + x_2}$$

Doubling

Let $P = (x_1, y_1)$ then $2P = (x_3, y_3)$, where

$$x_3 = \lambda^2 + \lambda + a_2$$
$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1$$
$$\lambda = x_1 + \frac{y_1}{x_1}$$

Lambda coordinates

Affine (Knudsen 1999)

Short affine point $P = (x, y)$ is (x, λ) where $\lambda = x + y/x$.

Projective (Oliveira et al. 2014)

With projective equation

$$(L^2 + LZ + a_2 Z^2)X^2 = X^4 + a_6 Z^4.$$

the λ-projective point $(X_1 : L_1 : Z_1)$ corresponds to the λ-affine point $(X_1/Z_1, L_1/Z_1)$.
The inverse of $(X_1 : L_1 : Z_1)$ is $(X_1 : L_1 + Z_1 : Z_1)$.

Computational costs

Computational costs of elliptic curve operations in various coordinate systems w.r.t. finite field inversions (I), multiplications (M), and squarings (S).

Coordinates	double	add	negate
affine	$1I + 2M + 1S$	$1I + 2M + 1S$	–
LD-projective (mixed)	$4M + 5S$	$8M + 5S$	$1M$
λ-projective (mixed)	$4M + 4S$	$8M + 2S$	–
λ-projective	$4M + 4S$	$11M + 2S$	–

ECC in OpenSSL

```
struct ec_method_st {
...
    int (*point_set_affine_coordinates) (const EC_GROUP *, EC_POINT *,
                                         const BIGNUM *x, const BIGNUM *y,
                                         BN_CTX *);
    int (*point_get_affine_coordinates) (const EC_GROUP *, const EC_POINT *,
                                         BIGNUM *x, BIGNUM *y, BN_CTX *);
...
    int (*add) (const EC_GROUP *, EC_POINT *r, const EC_POINT *a,
                const EC_POINT *b, BN_CTX *);
    int (*dbl) (const EC_GROUP *, EC_POINT *r, const EC_POINT *a, BN_CTX *);
    int (*invert) (const EC_GROUP *, EC_POINT *, BN_CTX *);
...
    int (*is_on_curve) (const EC_GROUP *, const EC_POINT *, BN_CTX *);
...
    int (*make_affine) (const EC_GROUP *, EC_POINT *, BN_CTX *);
    int (*points_make_affine) (const EC_GROUP *, size_t num, EC_POINT *[],
                               BN_CTX *);
...
    int (*mul) (const EC_GROUP *group, EC_POINT *r, const BIGNUM *scalar,
                size_t num, const EC_POINT *points[], const BIGNUM *scalars[],
                BN_CTX *);
    int (*precompute_mult) (EC_GROUP *group, BN_CTX *);
    int (*have_precompute_mult) (const EC_GROUP *group);
    int (*field_mul) (const EC_GROUP *, BIGNUM *r, const BIGNUM *a,
                      const BIGNUM *b, BN_CTX *);
    int (*field_sqr) (const EC_GROUP *, BIGNUM *r, const BIGNUM *a, BN_CTX *);
    int (*field_div) (const EC_GROUP *, BIGNUM *r, const BIGNUM *a,
                      const BIGNUM *b, BN_CTX *);
...
} /* EC_METHOD */ ;
```

Scalar multiplication in OpenSSL

```
/** Computes r = generator * n sum_{i=0}^{num-1} p[i] * m[i]
 *  \param  group  underlying EC_GROUP object
 *  \param  r      EC_POINT object for the result
 *  \param  n      BIGNUM with the multiplier for the group generator (optional)
 *  \param  num    number futher summands
 *  \param  p      array of size num of EC_POINT objects
 *  \param  m      array of size num of BIGNUM objects
 *  \param  ctx    BN_CTX object (optional)
 *  \return 1 on success and 0 if an error occurred
 */
int EC_POINTs_mul(const EC_GROUP *group, EC_POINT *r, const BIGNUM *n,
                  size_t num, const EC_POINT *p[], const BIGNUM *m[],
                  BN_CTX *ctx);
```

Montgomery's Ladder in OpenSSL

```
/*-
 * Computes scalar*point and stores the result in r.
 * point can not equal r.
 * Uses a modified algorithm 2P of
 *     Lopez, J. and Dahab, R.  "Fast multiplication on elliptic curves over
 *     GF(2^m) without precomputation" (CHES '99, LNCS 1717).
 *
 * To protect against side-channel attack the function uses constant time swap,
 * avoiding conditional branches.
 */
static int ec_GF2m_montgomery_point_multiply(const EC_GROUP *group,
                                             EC_POINT *r,
                                             const BIGNUM *scalar,
                                             const EC_POINT *point,
                                             BN_CTX *ctx)
```

Bug attacks and projective randomization

- Bug attacks (Biham et al. 2008) target implementation errors to steal keys.
- Pick β at random, and at the beginning of scalar multiplication set the accumulator to $(\beta X : \beta L : \beta Z)$.
- Observe
  $(\beta X : \beta L : \beta Z) \mapsto ((\beta X)/(\beta Z), (\beta L)/(\beta Z)) = (X/Z, L/Z)$.

ECDH performance

ECDH operations per second. Intel Celeron 2955U 1.40GHz.

curve	stock	modified	gain
nistk163	2107.7	2022.6	-4.0%
nistk233	1675.2	1670.2	-0.3%
nistk283	929.3	921.0	-0.9%
nistk409	589.5	563.8	-4.4%
nistk571	248.7	244.9	-1.5%
nistb163	2043.9	2011.4	-1.6%
nistb233	1600.9	1640.6	2.5%
nistb283	891.6	903.9	1.4%
nistb409	551.9	559.4	1.4%
nistb571	229.1	243.5	6.3%

ECDSA performance

ECDSA operations per second. Intel Celeron 2955U 1.40GHz.

curve	stock (sign)	modified (sign)	gain (sign)	stock (verify)	modified (verify)	gain (verify)
nistk163	2304.1	6723.4	191.8%	1022.9	1617.6	58.1%
nistk233	1146.2	5147.5	349.1%	791.8	1313.5	65.9%
nistk283	770.6	3136.7	307.0%	442.6	744.2	68.1%
nistk409	341.0	1969.2	477.5%	280.2	456.4	62.9%
nistk571	158.2	896.0	466.4%	120.2	199.0	65.6%
nistb163	2300.3	6684.2	190.6%	983.1	1635.9	66.4%
nistb233	1174.2	5227.7	345.2%	765.0	1280.2	67.3%
nistb283	771.3	3142.4	307.4%	420.1	735.1	75.0%
nistb409	339.8	1952.7	474.7%	262.4	446.5	70.2%
nistb571	157.6	858.8	444.9%	111.1	197.7	77.9%

Conclusion

- Source code patches: RT 4013
  `http://marc.info/?l=openssl-dev&m=144008703808363`
- Leverages existing arch, not a(nother) full stack
- Crypto in a vacuum has dubious utility

Future work

Specialized finite field arithmetic (Bluhm & Gueron 2015)