

AMCL

Generated by Doxygen 1.8.13



# Contents

- 1 Apache Milagro Crypto Library (AMCL) 1**
  - 1.1 Project page . . . . . 1
  - 1.2 License . . . . . 1
  - 1.3 Platforms . . . . . 1
  - 1.4 Downloads . . . . . 2
  - 1.5 Installation . . . . . 2
  
- 2 Linux 3**
  
- 3 Mac OS 5**
  
- 4 Windows 7**
  
- 5 Data Structure Index 9**
  - 5.1 Data Structures . . . . . 9
  
- 6 File Index 11**
  - 6.1 File List . . . . . 11

---

<b>7 Data Structure Documentation</b>	<b>13</b>
7.1 amcl_aes Struct Reference	13
7.1.1 Field Documentation	13
7.1.1.1 f	13
7.1.1.2 fkey	13
7.1.1.3 mode	14
7.1.1.4 Nk	14
7.1.1.5 Nr	14
7.1.1.6 rkey	14
7.2 csprng Struct Reference	14
7.2.1 Field Documentation	14
7.2.1.1 borrow	15
7.2.1.2 ira	15
7.2.1.3 pool	15
7.2.1.4 pool_ptr	15
7.2.1.5 rndptr	15
7.3 ECP2_BLS381 Struct Reference	15
7.3.1 Field Documentation	16
7.3.1.1 x	16
7.3.1.2 y	16
7.3.1.3 z	16
7.4 ECP_BLS381 Struct Reference	16
7.4.1 Field Documentation	16
7.4.1.1 x	17
7.4.1.2 y	17
7.4.1.3 z	17
7.5 FP12_BLS381 Struct Reference	17
7.5.1 Field Documentation	17
7.5.1.1 a	17
7.5.1.2 b	18

---

7.5.1.3	c	18
7.5.1.4	type	18
7.6	FP2_BLS381 Struct Reference	18
7.6.1	Field Documentation	18
7.6.1.1	a	18
7.6.1.2	b	19
7.7	FP4_BLS381 Struct Reference	19
7.7.1	Field Documentation	19
7.7.1.1	a	19
7.7.1.2	b	19
7.8	FP_BLS381 Struct Reference	19
7.8.1	Field Documentation	20
7.8.1.1	g	20
7.8.1.2	XES	20
7.9	gcm Struct Reference	20
7.9.1	Field Documentation	20
7.9.1.1	a	21
7.9.1.2	lenA	21
7.9.1.3	lenC	21
7.9.1.4	stateX	21
7.9.1.5	status	21
7.9.1.6	table	21
7.9.1.7	Y_0	21
7.10	hash256 Struct Reference	22
7.10.1	Field Documentation	22
7.10.1.1	h	22
7.10.1.2	hlen	22
7.10.1.3	length	22
7.10.1.4	w	22
7.11	hash512 Struct Reference	23

---

---

7.11.1	Field Documentation	23
7.11.1.1	h	23
7.11.1.2	hlen	23
7.11.1.3	length	23
7.11.1.4	w	23
7.12	octet Struct Reference	24
7.12.1	Field Documentation	24
7.12.1.1	len	24
7.12.1.2	max	24
7.12.1.3	val	24
7.13	PAILLIER_private_key Struct Reference	24
7.13.1	Field Documentation	25
7.13.1.1	invp	25
7.13.1.2	invq	25
7.13.1.3	lp	25
7.13.1.4	lq	25
7.13.1.5	mp	25
7.13.1.6	mq	26
7.13.1.7	p	26
7.13.1.8	p2	26
7.13.1.9	q	26
7.13.1.10	q2	26
7.14	PAILLIER_public_key Struct Reference	26
7.14.1	Field Documentation	27
7.14.1.1	g	27
7.14.1.2	n	27
7.14.1.3	n2	27
7.15	pktype Struct Reference	27
7.15.1	Field Documentation	27
7.15.1.1	curve	28

---

---

7.15.1.2	hash	28
7.15.1.3	type	28
7.16	rsa_private_key_2048 Struct Reference	28
7.16.1	Field Documentation	28
7.16.1.1	c	28
7.16.1.2	dp	29
7.16.1.3	dq	29
7.16.1.4	p	29
7.16.1.5	q	29
7.17	rsa_public_key_2048 Struct Reference	29
7.17.1	Field Documentation	29
7.17.1.1	e	30
7.17.1.2	n	30
7.18	sha3 Struct Reference	30
7.18.1	Field Documentation	30
7.18.1.1	len	30
7.18.1.2	length	30
7.18.1.3	rate	30
7.18.1.4	S	30
<b>8</b>	<b>File Documentation</b>	<b>31</b>
8.1	arch.h File Reference	31
8.1.1	Detailed Description	31
8.1.2	Macro Definition Documentation	31
8.1.2.1	byte	32
8.1.2.2	CHUNK	32
8.1.2.3	chunk	32
8.1.2.4	sign16	32
8.1.2.5	sign32	32
8.1.2.6	sign64	32
8.1.2.7	sign8	32

---

8.1.2.8	uchar	33
8.1.2.9	unsign32	33
8.1.2.10	unsign64	33
8.2	big_1024_58.h File Reference	33
8.2.1	Detailed Description	36
8.2.2	Macro Definition Documentation	37
8.2.2.1	BIGBITS_1024_58	37
8.2.2.2	BMASK_1024_58	37
8.2.2.3	DNLEN_1024_58	37
8.2.2.4	HBITS_1024_58	37
8.2.2.5	HMASK_1024_58	37
8.2.2.6	NEXCESS_1024_58	37
8.2.2.7	NLEN_1024_58	37
8.2.3	Typedef Documentation	38
8.2.3.1	BIG_1024_58	38
8.2.3.2	DBIG_1024_58	38
8.2.4	Function Documentation	38
8.2.4.1	BIG_1024_58_add()	38
8.2.4.2	BIG_1024_58_bit()	38
8.2.4.3	BIG_1024_58_cmove()	39
8.2.4.4	BIG_1024_58_comp()	39
8.2.4.5	BIG_1024_58_copy()	39
8.2.4.6	BIG_1024_58_cswap()	40
8.2.4.7	BIG_1024_58_dadd()	40
8.2.4.8	BIG_1024_58_dcmove()	40
8.2.4.9	BIG_1024_58_dcomp()	41
8.2.4.10	BIG_1024_58_dcopy()	41
8.2.4.11	BIG_1024_58_ddiv()	41
8.2.4.12	BIG_1024_58_dec()	42
8.2.4.13	BIG_1024_58_dfromBytesLen()	42



---

8.2.4.14	BIG_1024_58_diszilch()	42
8.2.4.15	BIG_1024_58_div3()	43
8.2.4.16	BIG_1024_58_dmod()	44
8.2.4.17	BIG_1024_58_dmod2m()	44
8.2.4.18	BIG_1024_58_dnbits()	44
8.2.4.19	BIG_1024_58_dnorm()	45
8.2.4.20	BIG_1024_58_doutput()	45
8.2.4.21	BIG_1024_58_drawoutput()	45
8.2.4.22	BIG_1024_58_dscopy()	46
8.2.4.23	BIG_1024_58_dshl()	46
8.2.4.24	BIG_1024_58_dshr()	46
8.2.4.25	BIG_1024_58_dsub()	46
8.2.4.26	BIG_1024_58_dscopy()	47
8.2.4.27	BIG_1024_58_dzero()	47
8.2.4.28	BIG_1024_58_fromBytes()	47
8.2.4.29	BIG_1024_58_fromBytesLen()	48
8.2.4.30	BIG_1024_58_fshl()	48
8.2.4.31	BIG_1024_58_fshr()	48
8.2.4.32	BIG_1024_58_imul()	49
8.2.4.33	BIG_1024_58_inc()	49
8.2.4.34	BIG_1024_58_invmod2m()	49
8.2.4.35	BIG_1024_58_invmodp()	49
8.2.4.36	BIG_1024_58_isunity()	50
8.2.4.37	BIG_1024_58_iszilch()	50
8.2.4.38	BIG_1024_58_jacobi()	50
8.2.4.39	BIG_1024_58_lastbits()	51
8.2.4.40	BIG_1024_58_mod()	51
8.2.4.41	BIG_1024_58_mod2m()	51
8.2.4.42	BIG_1024_58_moddiv()	52
8.2.4.43	BIG_1024_58_modmul()	52

---

8.2.4.44	BIG_1024_58_modneg()	53
8.2.4.45	BIG_1024_58_modsqr()	53
8.2.4.46	BIG_1024_58_monty()	53
8.2.4.47	BIG_1024_58_mul()	54
8.2.4.48	BIG_1024_58_nbits()	54
8.2.4.49	BIG_1024_58_norm()	54
8.2.4.50	BIG_1024_58_one()	55
8.2.4.51	BIG_1024_58_or()	55
8.2.4.52	BIG_1024_58_output()	55
8.2.4.53	BIG_1024_58_parity()	55
8.2.4.54	BIG_1024_58_pmul()	56
8.2.4.55	BIG_1024_58_pxmud()	56
8.2.4.56	BIG_1024_58_random()	56
8.2.4.57	BIG_1024_58_randomnum()	57
8.2.4.58	BIG_1024_58_rawoutput()	57
8.2.4.59	BIG_1024_58_rcopy()	57
8.2.4.60	BIG_1024_58_sdcopy()	58
8.2.4.61	BIG_1024_58_sdiv()	58
8.2.4.62	BIG_1024_58_sducopy()	58
8.2.4.63	BIG_1024_58_shl()	58
8.2.4.64	BIG_1024_58_shr()	59
8.2.4.65	BIG_1024_58_smul()	59
8.2.4.66	BIG_1024_58_split()	59
8.2.4.67	BIG_1024_58_sqr()	60
8.2.4.68	BIG_1024_58_ssn()	60
8.2.4.69	BIG_1024_58_sub()	60
8.2.4.70	BIG_1024_58_toBytes()	61
8.2.4.71	BIG_1024_58_zero()	61
8.3	big_384_58.h File Reference	61
8.3.1	Detailed Description	65

---

---

8.3.2	Macro Definition Documentation	65
8.3.2.1	BIGBITS_384_58	65
8.3.2.2	BMASK_384_58	65
8.3.2.3	DNLEN_384_58	65
8.3.2.4	HBITS_384_58	65
8.3.2.5	HMASK_384_58	65
8.3.2.6	NEXCESS_384_58	66
8.3.2.7	NLEN_384_58	66
8.3.3	Typedef Documentation	66
8.3.3.1	BIG_384_58	66
8.3.3.2	DBIG_384_58	66
8.3.4	Function Documentation	66
8.3.4.1	BIG_384_58_add()	66
8.3.4.2	BIG_384_58_bit()	67
8.3.4.3	BIG_384_58_cmove()	67
8.3.4.4	BIG_384_58_comp()	67
8.3.4.5	BIG_384_58_copy()	68
8.3.4.6	BIG_384_58_cswap()	68
8.3.4.7	BIG_384_58_dadd()	68
8.3.4.8	BIG_384_58_dcmove()	69
8.3.4.9	BIG_384_58_dcomp()	69
8.3.4.10	BIG_384_58_dcopy()	69
8.3.4.11	BIG_384_58_ddiv()	70
8.3.4.12	BIG_384_58_dec()	70
8.3.4.13	BIG_384_58_dfromBytesLen()	70
8.3.4.14	BIG_384_58_diszilch()	71
8.3.4.15	BIG_384_58_div3()	71
8.3.4.16	BIG_384_58_dmod()	71
8.3.4.17	BIG_384_58_dmod2m()	72
8.3.4.18	BIG_384_58_dnbits()	72

---

8.3.4.19	<code>BIG_384_58_dnorm()</code>	72
8.3.4.20	<code>BIG_384_58_doutput()</code>	72
8.3.4.21	<code>BIG_384_58_drawoutput()</code>	73
8.3.4.22	<code>BIG_384_58_dscopy()</code>	73
8.3.4.23	<code>BIG_384_58_dshl()</code>	73
8.3.4.24	<code>BIG_384_58_dshr()</code>	73
8.3.4.25	<code>BIG_384_58_dsub()</code>	74
8.3.4.26	<code>BIG_384_58_dscopy()</code>	74
8.3.4.27	<code>BIG_384_58_dzero()</code>	74
8.3.4.28	<code>BIG_384_58_fromBytes()</code>	75
8.3.4.29	<code>BIG_384_58_fromBytesLen()</code>	75
8.3.4.30	<code>BIG_384_58_fshl()</code>	75
8.3.4.31	<code>BIG_384_58_fshr()</code>	76
8.3.4.32	<code>BIG_384_58_imul()</code>	76
8.3.4.33	<code>BIG_384_58_inc()</code>	76
8.3.4.34	<code>BIG_384_58_invmod2m()</code>	77
8.3.4.35	<code>BIG_384_58_invmodp()</code>	77
8.3.4.36	<code>BIG_384_58_issunity()</code>	77
8.3.4.37	<code>BIG_384_58_iszilch()</code>	77
8.3.4.38	<code>BIG_384_58_jacobi()</code>	78
8.3.4.39	<code>BIG_384_58_lastbits()</code>	78
8.3.4.40	<code>BIG_384_58_mod()</code>	78
8.3.4.41	<code>BIG_384_58_mod2m()</code>	79
8.3.4.42	<code>BIG_384_58_moddiv()</code>	79
8.3.4.43	<code>BIG_384_58_modmul()</code>	79
8.3.4.44	<code>BIG_384_58_modneg()</code>	80
8.3.4.45	<code>BIG_384_58_modsqr()</code>	80
8.3.4.46	<code>BIG_384_58_monty()</code>	81
8.3.4.47	<code>BIG_384_58_mul()</code>	81
8.3.4.48	<code>BIG_384_58_nbits()</code>	81

---

8.3.4.49	<a href="#">BIG_384_58_norm()</a>	82
8.3.4.50	<a href="#">BIG_384_58_one()</a>	82
8.3.4.51	<a href="#">BIG_384_58_or()</a>	82
8.3.4.52	<a href="#">BIG_384_58_output()</a>	82
8.3.4.53	<a href="#">BIG_384_58_parity()</a>	83
8.3.4.54	<a href="#">BIG_384_58_pmul()</a>	83
8.3.4.55	<a href="#">BIG_384_58_pxmulo()</a>	83
8.3.4.56	<a href="#">BIG_384_58_random()</a>	84
8.3.4.57	<a href="#">BIG_384_58_randomnum()</a>	84
8.3.4.58	<a href="#">BIG_384_58_rawoutput()</a>	84
8.3.4.59	<a href="#">BIG_384_58_rcopy()</a>	84
8.3.4.60	<a href="#">BIG_384_58_sdcopy()</a>	85
8.3.4.61	<a href="#">BIG_384_58_sdiv()</a>	85
8.3.4.62	<a href="#">BIG_384_58_sducopy()</a>	85
8.3.4.63	<a href="#">BIG_384_58_shl()</a>	86
8.3.4.64	<a href="#">BIG_384_58_shr()</a>	86
8.3.4.65	<a href="#">BIG_384_58_smulo()</a>	86
8.3.4.66	<a href="#">BIG_384_58_split()</a>	86
8.3.4.67	<a href="#">BIG_384_58_sqr()</a>	87
8.3.4.68	<a href="#">BIG_384_58_ssn()</a>	87
8.3.4.69	<a href="#">BIG_384_58_sub()</a>	88
8.3.4.70	<a href="#">BIG_384_58_toBytes()</a>	88
8.3.4.71	<a href="#">BIG_384_58_zero()</a>	88
8.4	<a href="#">big_512_60.h File Reference</a>	88
8.4.1	<a href="#">Detailed Description</a>	92
8.4.2	<a href="#">Macro Definition Documentation</a>	92
8.4.2.1	<a href="#">BIGBITS_512_60</a>	92
8.4.2.2	<a href="#">BMASK_512_60</a>	92
8.4.2.3	<a href="#">DNLEN_512_60</a>	92
8.4.2.4	<a href="#">HBITS_512_60</a>	93

---

8.4.2.5	HMASK_512_60	93
8.4.2.6	NEXCESS_512_60	93
8.4.2.7	NLEN_512_60	93
8.4.3	Typedef Documentation	93
8.4.3.1	BIG_512_60	93
8.4.3.2	DBIG_512_60	93
8.4.4	Function Documentation	93
8.4.4.1	BIG_512_60_add()	93
8.4.4.2	BIG_512_60_bit()	94
8.4.4.3	BIG_512_60_cmove()	94
8.4.4.4	BIG_512_60_comp()	94
8.4.4.5	BIG_512_60_copy()	95
8.4.4.6	BIG_512_60_cswap()	95
8.4.4.7	BIG_512_60_dadd()	95
8.4.4.8	BIG_512_60_dcmove()	96
8.4.4.9	BIG_512_60_dcomp()	96
8.4.4.10	BIG_512_60_dcopy()	96
8.4.4.11	BIG_512_60_ddiv()	97
8.4.4.12	BIG_512_60_dec()	97
8.4.4.13	BIG_512_60_dfromBytesLen()	97
8.4.4.14	BIG_512_60_diszilch()	98
8.4.4.15	BIG_512_60_div3()	98
8.4.4.16	BIG_512_60_dmod()	98
8.4.4.17	BIG_512_60_dmod2m()	99
8.4.4.18	BIG_512_60_dnbits()	99
8.4.4.19	BIG_512_60_dnorm()	99
8.4.4.20	BIG_512_60_doutput()	99
8.4.4.21	BIG_512_60_drawoutput()	100
8.4.4.22	BIG_512_60_dscopy()	100
8.4.4.23	BIG_512_60_dshl()	100

---

8.4.4.24	<a href="#">BIG_512_60_dshr()</a>	101
8.4.4.25	<a href="#">BIG_512_60_dsub()</a>	101
8.4.4.26	<a href="#">BIG_512_60_dscopy()</a>	101
8.4.4.27	<a href="#">BIG_512_60_dzero()</a>	101
8.4.4.28	<a href="#">BIG_512_60_fromBytes()</a>	102
8.4.4.29	<a href="#">BIG_512_60_fromBytesLen()</a>	102
8.4.4.30	<a href="#">BIG_512_60_fshl()</a>	102
8.4.4.31	<a href="#">BIG_512_60_fshr()</a>	103
8.4.4.32	<a href="#">BIG_512_60_imul()</a>	103
8.4.4.33	<a href="#">BIG_512_60_inc()</a>	103
8.4.4.34	<a href="#">BIG_512_60_invmod2m()</a>	104
8.4.4.35	<a href="#">BIG_512_60_invmodp()</a>	104
8.4.4.36	<a href="#">BIG_512_60_isunity()</a>	104
8.4.4.37	<a href="#">BIG_512_60_iszilch()</a>	105
8.4.4.38	<a href="#">BIG_512_60_jacobi()</a>	106
8.4.4.39	<a href="#">BIG_512_60_lastbits()</a>	106
8.4.4.40	<a href="#">BIG_512_60_mod()</a>	106
8.4.4.41	<a href="#">BIG_512_60_mod2m()</a>	107
8.4.4.42	<a href="#">BIG_512_60_moddiv()</a>	107
8.4.4.43	<a href="#">BIG_512_60_modmul()</a>	107
8.4.4.44	<a href="#">BIG_512_60_modneg()</a>	108
8.4.4.45	<a href="#">BIG_512_60_modsqr()</a>	108
8.4.4.46	<a href="#">BIG_512_60_monty()</a>	109
8.4.4.47	<a href="#">BIG_512_60_mul()</a>	109
8.4.4.48	<a href="#">BIG_512_60_nbits()</a>	109
8.4.4.49	<a href="#">BIG_512_60_norm()</a>	110
8.4.4.50	<a href="#">BIG_512_60_one()</a>	110
8.4.4.51	<a href="#">BIG_512_60_or()</a>	110
8.4.4.52	<a href="#">BIG_512_60_output()</a>	110
8.4.4.53	<a href="#">BIG_512_60_parity()</a>	111

---

8.4.4.54	BIG_512_60_pmul()	111
8.4.4.55	BIG_512_60_pxmud()	111
8.4.4.56	BIG_512_60_random()	112
8.4.4.57	BIG_512_60_randomnum()	112
8.4.4.58	BIG_512_60_rawoutput()	112
8.4.4.59	BIG_512_60_rcopy()	112
8.4.4.60	BIG_512_60_sdcopy()	113
8.4.4.61	BIG_512_60_sdiv()	113
8.4.4.62	BIG_512_60_sducopy()	113
8.4.4.63	BIG_512_60_shl()	114
8.4.4.64	BIG_512_60_shr()	114
8.4.4.65	BIG_512_60_smul()	114
8.4.4.66	BIG_512_60_split()	114
8.4.4.67	BIG_512_60_sqr()	115
8.4.4.68	BIG_512_60_ssn()	115
8.4.4.69	BIG_512_60_sub()	116
8.4.4.70	BIG_512_60_toBytes()	116
8.4.4.71	BIG_512_60_zero()	116
8.5	bls_BLS381.h File Reference	116
8.5.1	Detailed Description	117
8.5.2	Macro Definition Documentation	117
8.5.2.1	BFS_BLS381	117
8.5.2.2	BGS_BLS381	118
8.5.2.3	BLS_FAIL	118
8.5.2.4	BLS_INVALID_G1	118
8.5.2.5	BLS_INVALID_G2	118
8.5.2.6	BLS_OK	118
8.5.3	Function Documentation	118
8.5.3.1	BLS_BLS381_ADD_G1()	118
8.5.3.2	BLS_BLS381_ADD_G2()	119



---

8.5.3.3	BLS_BLS381_KEY_PAIR_GENERATE()	119
8.5.3.4	BLS_BLS381_MAKE_SHARES()	119
8.5.3.5	BLS_BLS381_RECOVER_SECRET()	120
8.5.3.6	BLS_BLS381_RECOVER_SIGNATURE()	120
8.5.3.7	BLS_BLS381_SIGN()	121
8.5.3.8	BLS_BLS381_VERIFY()	121
8.6	config_big_1024_58.h File Reference	122
8.6.1	Detailed Description	122
8.6.2	Macro Definition Documentation	122
8.6.2.1	BASEBITS_1024_58	122
8.6.2.2	MODBYTES_1024_58	122
8.7	config_big_384_58.h File Reference	122
8.7.1	Detailed Description	123
8.7.2	Macro Definition Documentation	123
8.7.2.1	BASEBITS_384_58	123
8.7.2.2	MODBYTES_384_58	123
8.8	config_big_512_60.h File Reference	123
8.8.1	Detailed Description	123
8.8.2	Macro Definition Documentation	124
8.8.2.1	BASEBITS_512_60	124
8.8.2.2	MODBYTES_512_60	124
8.9	config_ff_2048.h File Reference	124
8.9.1	Detailed Description	124
8.9.2	Macro Definition Documentation	124
8.9.2.1	FFLEN_2048	124
8.10	config_ff_4096.h File Reference	125
8.10.1	Detailed Description	125
8.10.2	Macro Definition Documentation	125
8.10.2.1	FFLEN_4096	125
8.11	ecdh_BLS381.h File Reference	125

---

8.11.1	Detailed Description	126
8.11.2	Macro Definition Documentation	126
8.11.2.1	ECDH_ERROR	126
8.11.2.2	ECDH_INVALID	126
8.11.2.3	ECDH_INVALID_PUBLIC_KEY	126
8.11.2.4	ECDH_OK	127
8.11.2.5	EFS_BLS381	127
8.11.2.6	EGS_BLS381	127
8.11.3	Function Documentation	127
8.11.3.1	ECP_BLS381_ECIES_DECRYPT()	127
8.11.3.2	ECP_BLS381_ECIES_ENCRYPT()	128
8.11.3.3	ECP_BLS381_KEY_PAIR_GENERATE()	128
8.11.3.4	ECP_BLS381_PUBLIC_KEY_VALIDATE()	129
8.11.3.5	ECP_BLS381_SP_DSA()	129
8.11.3.6	ECP_BLS381_SVDP_DH()	129
8.11.3.7	ECP_BLS381_VP_DSA()	130
8.12	ecdh_support.h File Reference	130
8.12.1	Detailed Description	131
8.12.2	Function Documentation	131
8.12.2.1	AES_CBC_IV0_DECRYPT()	131
8.12.2.2	AES_CBC_IV0_ENCRYPT()	132
8.12.2.3	ehashit()	132
8.12.2.4	HASH()	132
8.12.2.5	HMAC()	133
8.12.2.6	KDF2()	133
8.12.2.7	PBKDF2()	134
8.13	ecp2_BLS381.h File Reference	134
8.13.1	Detailed Description	136
8.13.2	Function Documentation	136
8.13.2.1	ECP2_BLS381_add()	136

---

8.13.2.2	ECP2_BLS381_affine()	136
8.13.2.3	ECP2_BLS381_copy()	136
8.13.2.4	ECP2_BLS381_dbl()	137
8.13.2.5	ECP2_BLS381_equals()	137
8.13.2.6	ECP2_BLS381_frob()	137
8.13.2.7	ECP2_BLS381_fromOctet()	137
8.13.2.8	ECP2_BLS381_generator()	138
8.13.2.9	ECP2_BLS381_get()	138
8.13.2.10	ECP2_BLS381_inf()	138
8.13.2.11	ECP2_BLS381_isinf()	139
8.13.2.12	ECP2_BLS381_mapit()	139
8.13.2.13	ECP2_BLS381_mul()	139
8.13.2.14	ECP2_BLS381_mul4()	140
8.13.2.15	ECP2_BLS381_neg()	140
8.13.2.16	ECP2_BLS381_output()	140
8.13.2.17	ECP2_BLS381_outputxyz()	140
8.13.2.18	ECP2_BLS381_rhs()	141
8.13.2.19	ECP2_BLS381_set()	141
8.13.2.20	ECP2_BLS381_setx()	141
8.13.2.21	ECP2_BLS381_sub()	142
8.13.2.22	ECP2_BLS381_toOctet()	142
8.13.3	Variable Documentation	142
8.13.3.1	CURVE_A_BLS381	142
8.13.3.2	CURVE_B_BLS381	142
8.13.3.3	CURVE_B_I_BLS381	143
8.13.3.4	CURVE_Bnx_BLS381	143
8.13.3.5	CURVE_Cof_BLS381	143
8.13.3.6	CURVE_Gx_BLS381	143
8.13.3.7	CURVE_Gy_BLS381	143
8.13.3.8	CURVE_Order_BLS381	143

---

8.13.3.9	CURVE_Pxa_BLS381	143
8.13.3.10	CURVE_Pxb_BLS381	143
8.13.3.11	CURVE_Pya_BLS381	144
8.13.3.12	CURVE_Pyb_BLS381	144
8.13.3.13	Fra_BLS381	144
8.13.3.14	Frb_BLS381	144
8.14	ecp_BLS381.h File Reference	144
8.14.1	Detailed Description	146
8.14.2	Function Documentation	147
8.14.2.1	ECP_BLS381_add()	147
8.14.2.2	ECP_BLS381_affine()	147
8.14.2.3	ECP_BLS381_cfp()	147
8.14.2.4	ECP_BLS381_copy()	147
8.14.2.5	ECP_BLS381_dbl()	148
8.14.2.6	ECP_BLS381_equals()	148
8.14.2.7	ECP_BLS381_fromOctet()	148
8.14.2.8	ECP_BLS381_generator()	149
8.14.2.9	ECP_BLS381_get()	149
8.14.2.10	ECP_BLS381_inf()	149
8.14.2.11	ECP_BLS381_isinf()	150
8.14.2.12	ECP_BLS381_mapit()	150
8.14.2.13	ECP_BLS381_mul()	150
8.14.2.14	ECP_BLS381_mul2()	150
8.14.2.15	ECP_BLS381_neg()	151
8.14.2.16	ECP_BLS381_output()	151
8.14.2.17	ECP_BLS381_outputxyz()	151
8.14.2.18	ECP_BLS381_pinmul()	152
8.14.2.19	ECP_BLS381_rawoutput()	152
8.14.2.20	ECP_BLS381_rhs()	152
8.14.2.21	ECP_BLS381_set()	152

---

8.14.2.22	ECP_BLS381_setx()	153
8.14.2.23	ECP_BLS381_sub()	153
8.14.2.24	ECP_BLS381_toOctet()	153
8.14.3	Variable Documentation	155
8.14.3.1	CURVE_A_BLS381	155
8.14.3.2	CURVE_B_BLS381	155
8.14.3.3	CURVE_B_I_BLS381	155
8.14.3.4	CURVE_BB_BLS381	155
8.14.3.5	CURVE_Bnx_BLS381	155
8.14.3.6	CURVE_Cof_BLS381	155
8.14.3.7	CURVE_Cof_I_BLS381	156
8.14.3.8	CURVE_Cru_BLS381	156
8.14.3.9	CURVE_Gx_BLS381	156
8.14.3.10	CURVE_Gy_BLS381	156
8.14.3.11	CURVE_Order_BLS381	156
8.14.3.12	CURVE_Pxa_BLS381	156
8.14.3.13	CURVE_Pxaa_BLS381	156
8.14.3.14	CURVE_Pxaaa_BLS381	156
8.14.3.15	CURVE_Pxaab_BLS381	157
8.14.3.16	CURVE_Pxab_BLS381	157
8.14.3.17	CURVE_Pxaba_BLS381	157
8.14.3.18	CURVE_Pxab_b_BLS381	157
8.14.3.19	CURVE_Pxb_BLS381	157
8.14.3.20	CURVE_Pxba_BLS381	157
8.14.3.21	CURVE_Pxbaa_BLS381	157
8.14.3.22	CURVE_Pxbab_BLS381	157
8.14.3.23	CURVE_Pxbb_BLS381	158
8.14.3.24	CURVE_Pxbba_BLS381	158
8.14.3.25	CURVE_Pxbbb_BLS381	158
8.14.3.26	CURVE_Pya_BLS381	158

---

---

8.14.3.27 CURVE_Pyaa_BLS381 . . . . .	158
8.14.3.28 CURVE_Pyaaa_BLS381 . . . . .	158
8.14.3.29 CURVE_Pyaab_BLS381 . . . . .	158
8.14.3.30 CURVE_Pyab_BLS381 . . . . .	158
8.14.3.31 CURVE_Pyaba_BLS381 . . . . .	159
8.14.3.32 CURVE_Pyabb_BLS381 . . . . .	159
8.14.3.33 CURVE_Pyb_BLS381 . . . . .	159
8.14.3.34 CURVE_Pyba_BLS381 . . . . .	159
8.14.3.35 CURVE_Pybaa_BLS381 . . . . .	159
8.14.3.36 CURVE_Pybab_BLS381 . . . . .	159
8.14.3.37 CURVE_Pybb_BLS381 . . . . .	159
8.14.3.38 CURVE_Pybba_BLS381 . . . . .	159
8.14.3.39 CURVE_Pybbb_BLS381 . . . . .	160
8.14.3.40 CURVE_SB_BLS381 . . . . .	160
8.14.3.41 CURVE_W_BLS381 . . . . .	160
8.14.3.42 CURVE_WB_BLS381 . . . . .	160
8.14.3.43 Fra_BLS381 . . . . .	160
8.14.3.44 Frb_BLS381 . . . . .	160
8.15 ff_2048.h File Reference . . . . .	160
8.15.1 Detailed Description . . . . .	162
8.15.2 Macro Definition Documentation . . . . .	162
8.15.2.1 HFLEN_2048 . . . . .	162
8.15.2.2 P_EXCESS_2048 . . . . .	163
8.15.2.3 P_FEXCESS_2048 . . . . .	163
8.15.2.4 P_MBITS_2048 . . . . .	163
8.15.2.5 P_TBITS_2048 . . . . .	163
8.15.3 Function Documentation . . . . .	163
8.15.3.1 FF_2048_add() . . . . .	163
8.15.3.2 FF_2048_cfactor() . . . . .	164
8.15.3.3 FF_2048_comp() . . . . .	164

---

8.15.3.4	FF_2048_copy()	164
8.15.3.5	FF_2048_crt()	165
8.15.3.6	FF_2048_dec()	165
8.15.3.7	FF_2048_dmod()	165
8.15.3.8	FF_2048_fromOctet()	166
8.15.3.9	FF_2048_inc()	166
8.15.3.10	FF_2048_init()	166
8.15.3.11	FF_2048_invmod2m()	167
8.15.3.12	FF_2048_invmodp()	167
8.15.3.13	FF_2048_iszilch()	167
8.15.3.14	FF_2048_lastbits()	168
8.15.3.15	FF_2048_mod()	168
8.15.3.16	FF_2048_mul()	168
8.15.3.17	FF_2048_norm()	169
8.15.3.18	FF_2048_one()	169
8.15.3.19	FF_2048_output()	169
8.15.3.20	FF_2048_parity()	170
8.15.3.21	FF_2048_pow()	170
8.15.3.22	FF_2048_pow2()	170
8.15.3.23	FF_2048_power()	171
8.15.3.24	FF_2048_prime()	171
8.15.3.25	FF_2048_random()	172
8.15.3.26	FF_2048_randomnum()	172
8.15.3.27	FF_2048_rawoutput()	172
8.15.3.28	FF_2048_shl()	173
8.15.3.29	FF_2048_shr()	173
8.15.3.30	FF_2048_skpow()	173
8.15.3.31	FF_2048_skpow2()	174
8.15.3.32	FF_2048_skspow()	174
8.15.3.33	FF_2048_sqr()	174

---

8.15.3.34	FF_2048_sub()	175
8.15.3.35	FF_2048_toOctet()	175
8.15.3.36	FF_2048_zero()	176
8.16	ff_4096.h File Reference	176
8.16.1	Detailed Description	178
8.16.2	Macro Definition Documentation	178
8.16.2.1	HFLen_4096	178
8.16.2.2	P_EXCESS_4096	178
8.16.2.3	P_FEXCESS_4096	178
8.16.2.4	P_MBITS_4096	178
8.16.2.5	P_TBITS_4096	178
8.16.3	Function Documentation	179
8.16.3.1	FF_4096_add()	179
8.16.3.2	FF_4096_cfactor()	179
8.16.3.3	FF_4096_comp()	179
8.16.3.4	FF_4096_copy()	180
8.16.3.5	FF_4096_crt()	180
8.16.3.6	FF_4096_dec()	181
8.16.3.7	FF_4096_dmod()	181
8.16.3.8	FF_4096_fromOctet()	181
8.16.3.9	FF_4096_inc()	182
8.16.3.10	FF_4096_init()	182
8.16.3.11	FF_4096_invmod2m()	182
8.16.3.12	FF_4096_invmodp()	183
8.16.3.13	FF_4096_iszilch()	183
8.16.3.14	FF_4096_lastbits()	183
8.16.3.15	FF_4096_mod()	184
8.16.3.16	FF_4096_mul()	184
8.16.3.17	FF_4096_norm()	184
8.16.3.18	FF_4096_one()	185



8.16.3.19	FF_4096_output()	185
8.16.3.20	FF_4096_parity()	185
8.16.3.21	FF_4096_pow()	185
8.16.3.22	FF_4096_pow2()	186
8.16.3.23	FF_4096_power()	186
8.16.3.24	FF_4096_prime()	187
8.16.3.25	FF_4096_random()	187
8.16.3.26	FF_4096_randomnum()	187
8.16.3.27	FF_4096_rawoutput()	188
8.16.3.28	FF_4096_shl()	188
8.16.3.29	FF_4096_shr()	188
8.16.3.30	FF_4096_skpow()	189
8.16.3.31	FF_4096_skpow2()	189
8.16.3.32	FF_4096_skspow()	189
8.16.3.33	FF_4096_sqr()	190
8.16.3.34	FF_4096_sub()	190
8.16.3.35	FF_4096_toOctet()	191
8.16.3.36	FF_4096_zero()	191
8.17	fp12_BLS381.h File Reference	191
8.17.1	Detailed Description	193
8.17.2	Function Documentation	193
8.17.2.1	FP12_BLS381_cmove()	193
8.17.2.2	FP12_BLS381_compow()	193
8.17.2.3	FP12_BLS381_conj()	194
8.17.2.4	FP12_BLS381_copy()	194
8.17.2.5	FP12_BLS381_equals()	194
8.17.2.6	FP12_BLS381_frob()	195
8.17.2.7	FP12_BLS381_from_FP4()	195
8.17.2.8	FP12_BLS381_from_FP4s()	195
8.17.2.9	FP12_BLS381_fromOctet()	196

---

8.17.2.10	FP12_BLS381_inv()	196
8.17.2.11	FP12_BLS381_isunity()	196
8.17.2.12	FP12_BLS381_iszilch()	197
8.17.2.13	FP12_BLS381_mul()	198
8.17.2.14	FP12_BLS381_norm()	198
8.17.2.15	FP12_BLS381_one()	198
8.17.2.16	FP12_BLS381_output()	199
8.17.2.17	FP12_BLS381_pinpow()	199
8.17.2.18	FP12_BLS381_pow()	199
8.17.2.19	FP12_BLS381_pow4()	199
8.17.2.20	FP12_BLS381_reduce()	200
8.17.2.21	FP12_BLS381_smul()	200
8.17.2.22	FP12_BLS381_sqr()	200
8.17.2.23	FP12_BLS381_ssmul()	201
8.17.2.24	FP12_BLS381_toOctet()	201
8.17.2.25	FP12_BLS381_trace()	201
8.17.2.26	FP12_BLS381_usqr()	201
8.17.2.27	FP12_BLS381_zero()	202
8.17.3	Variable Documentation	202
8.17.3.1	Fra_BLS381	202
8.17.3.2	Frb_BLS381	202
8.18	fp2_BLS381.h File Reference	202
8.18.1	Detailed Description	204
8.18.2	Function Documentation	204
8.18.2.1	FP2_BLS381_add()	204
8.18.2.2	FP2_BLS381_cmove()	204
8.18.2.3	FP2_BLS381_conj()	205
8.18.2.4	FP2_BLS381_copy()	205
8.18.2.5	FP2_BLS381_div2()	205
8.18.2.6	FP2_BLS381_div_ip()	206

---

8.18.2.7	FP2_BLS381_div_ip2()	206
8.18.2.8	FP2_BLS381_equals()	206
8.18.2.9	FP2_BLS381_from_BIG()	207
8.18.2.10	FP2_BLS381_from_BIGs()	207
8.18.2.11	FP2_BLS381_from_FP()	207
8.18.2.12	FP2_BLS381_from_FPs()	207
8.18.2.13	FP2_BLS381_imul()	208
8.18.2.14	FP2_BLS381_inv()	208
8.18.2.15	FP2_BLS381_ismunity()	208
8.18.2.16	FP2_BLS381_iszilch()	209
8.18.2.17	FP2_BLS381_mul()	209
8.18.2.18	FP2_BLS381_mul_ip()	209
8.18.2.19	FP2_BLS381_neg()	210
8.18.2.20	FP2_BLS381_norm()	210
8.18.2.21	FP2_BLS381_one()	210
8.18.2.22	FP2_BLS381_output()	210
8.18.2.23	FP2_BLS381_pmul()	211
8.18.2.24	FP2_BLS381_pow()	211
8.18.2.25	FP2_BLS381_rawoutput()	211
8.18.2.26	FP2_BLS381_reduce()	211
8.18.2.27	FP2_BLS381_sqr()	212
8.18.2.28	FP2_BLS381_sqrt()	212
8.18.2.29	FP2_BLS381_sub()	212
8.18.2.30	FP2_BLS381_times_i()	213
8.18.2.31	FP2_BLS381_zero()	213
8.19	fp4_BLS381.h File Reference	213
8.19.1	Detailed Description	215
8.19.2	Function Documentation	215
8.19.2.1	FP4_BLS381_add()	215
8.19.2.2	FP4_BLS381_cmove()	215

8.19.2.3	FP4_BLS381_conj()	217
8.19.2.4	FP4_BLS381_copy()	217
8.19.2.5	FP4_BLS381_div2()	217
8.19.2.6	FP4_BLS381_div_2i()	218
8.19.2.7	FP4_BLS381_div_i()	218
8.19.2.8	FP4_BLS381_equals()	218
8.19.2.9	FP4_BLS381_frob()	219
8.19.2.10	FP4_BLS381_from_FP2()	219
8.19.2.11	FP4_BLS381_from_FP2H()	219
8.19.2.12	FP4_BLS381_from_FP2s()	219
8.19.2.13	FP4_BLS381_imul()	221
8.19.2.14	FP4_BLS381_inv()	221
8.19.2.15	FP4_BLS381_isreal()	221
8.19.2.16	FP4_BLS381_issunity()	222
8.19.2.17	FP4_BLS381_issilch()	222
8.19.2.18	FP4_BLS381_mul()	222
8.19.2.19	FP4_BLS381_nconj()	223
8.19.2.20	FP4_BLS381_neg()	223
8.19.2.21	FP4_BLS381_norm()	223
8.19.2.22	FP4_BLS381_one()	223
8.19.2.23	FP4_BLS381_output()	224
8.19.2.24	FP4_BLS381_pmul()	224
8.19.2.25	FP4_BLS381_pow()	224
8.19.2.26	FP4_BLS381_qmul()	225
8.19.2.27	FP4_BLS381_rawoutput()	225
8.19.2.28	FP4_BLS381_reduce()	225
8.19.2.29	FP4_BLS381_sqr()	225
8.19.2.30	FP4_BLS381_sqrt()	226
8.19.2.31	FP4_BLS381_sub()	226
8.19.2.32	FP4_BLS381_times_i()	226

8.19.2.33 FP4_BLS381_xtr_A()	227
8.19.2.34 FP4_BLS381_xtr_D()	227
8.19.2.35 FP4_BLS381_xtr_pow()	227
8.19.2.36 FP4_BLS381_xtr_pow2()	228
8.19.2.37 FP4_BLS381_zero()	228
8.20 fp_BLS381.h File Reference	228
8.20.1 Detailed Description	230
8.20.2 Macro Definition Documentation	230
8.20.2.1 FEXCESS_BLS381	230
8.20.2.2 MODBITS_BLS381	230
8.20.2.3 OMASK_BLS381	230
8.20.2.4 TBITS_BLS381	231
8.20.2.5 TMASK_BLS381	231
8.20.3 Function Documentation	231
8.20.3.1 FP_BLS381_add()	231
8.20.3.2 FP_BLS381_cmove()	231
8.20.3.3 FP_BLS381_copy()	232
8.20.3.4 FP_BLS381_cswap()	232
8.20.3.5 FP_BLS381_div2()	232
8.20.3.6 FP_BLS381_equals()	232
8.20.3.7 FP_BLS381_imul()	233
8.20.3.8 FP_BLS381_inv()	233
8.20.3.9 FP_BLS381_iszilch()	233
8.20.3.10 FP_BLS381_mod()	234
8.20.3.11 FP_BLS381_mul()	234
8.20.3.12 FP_BLS381_neg()	234
8.20.3.13 FP_BLS381_norm()	235
8.20.3.14 FP_BLS381_nres()	235
8.20.3.15 FP_BLS381_one()	235
8.20.3.16 FP_BLS381_output()	235

8.20.3.17	FP_BLS381_pow()	236
8.20.3.18	FP_BLS381_qr()	236
8.20.3.19	FP_BLS381_rawoutput()	236
8.20.3.20	FP_BLS381_rcopy()	236
8.20.3.21	FP_BLS381_redc()	237
8.20.3.22	FP_BLS381_reduce()	237
8.20.3.23	FP_BLS381_sqr()	237
8.20.3.24	FP_BLS381_sqrt()	238
8.20.3.25	FP_BLS381_sub()	238
8.20.3.26	FP_BLS381_zero()	238
8.20.4	Variable Documentation	238
8.20.4.1	MConst_BLS381	238
8.20.4.2	Modulus_BLS381	239
8.20.4.3	R2modp_BLS381	239
8.21	mpin_BLS381.h File Reference	239
8.21.1	Detailed Description	240
8.21.2	Macro Definition Documentation	241
8.21.2.1	M_SIZE_BLS381	241
8.21.2.2	MAXPIN	241
8.21.2.3	MESSAGE_SIZE	241
8.21.2.4	MPIN_BAD_PIN	241
8.21.2.5	MPIN_INVALID_POINT	241
8.21.2.6	MPIN_OK	241
8.21.2.7	MPIN_PAS	241
8.21.2.8	PBLEN	242
8.21.2.9	PFS_BLS381	242
8.21.2.10	PGS_BLS381	242
8.21.3	Function Documentation	242
8.21.3.1	MPIN_BLS381_CLIENT()	242
8.21.3.2	MPIN_BLS381_CLIENT_1()	243

---

8.21.3.3	MPIN_BLS381_CLIENT_2()	244
8.21.3.4	MPIN_BLS381_CLIENT_KEY()	244
8.21.3.5	MPIN_BLS381_DECODING()	245
8.21.3.6	MPIN_BLS381_ENCODING()	245
8.21.3.7	MPIN_BLS381_EXTRACT_FACTOR()	245
8.21.3.8	MPIN_BLS381_EXTRACT_PIN()	246
8.21.3.9	MPIN_BLS381_GET_CLIENT_PERMIT()	246
8.21.3.10	MPIN_BLS381_GET_CLIENT_SECRET()	247
8.21.3.11	MPIN_BLS381_GET_DVS_KEYPAIR()	247
8.21.3.12	MPIN_BLS381_GET_G1_MULTIPLE()	247
8.21.3.13	MPIN_BLS381_GET_G2_MULTIPLE()	248
8.21.3.14	MPIN_BLS381_GET_SERVER_SECRET()	248
8.21.3.15	MPIN_BLS381_GET_Y()	249
8.21.3.16	MPIN_BLS381_KANGAROO()	249
8.21.3.17	MPIN_BLS381_PRECOMPUTE()	249
8.21.3.18	MPIN_BLS381_RANDOM_GENERATE()	250
8.21.3.19	MPIN_BLS381_RECOMBINE_G1()	250
8.21.3.20	MPIN_BLS381_RECOMBINE_G2()	251
8.21.3.21	MPIN_BLS381_RESTORE_FACTOR()	251
8.21.3.22	MPIN_BLS381_SERVER()	251
8.21.3.23	MPIN_BLS381_SERVER_1()	252
8.21.3.24	MPIN_BLS381_SERVER_2()	253
8.21.3.25	MPIN_BLS381_SERVER_KEY()	254
8.22	paillier.h File Reference	254
8.22.1	Macro Definition Documentation	255
8.22.1.1	FS_2048	255
8.22.1.2	FS_4096	255
8.22.1.3	HFS_2048	256
8.22.1.4	HFS_4096	256
8.22.2	Function Documentation	256

---

8.22.2.1	PAILLIER_ADD()	256
8.22.2.2	PAILLIER_DECRYPT()	256
8.22.2.3	PAILLIER_ENCRYPT()	257
8.22.2.4	PAILLIER_KEY_PAIR()	257
8.22.2.5	PAILLIER_MULT()	258
8.22.2.6	PAILLIER_PK_fromOctet()	259
8.22.2.7	PAILLIER_PK_toOctet()	259
8.22.2.8	PAILLIER_PRIVATE_KEY_KILL()	259
8.23	pair_BLS381.h File Reference	259
8.23.1	Detailed Description	260
8.23.2	Function Documentation	260
8.23.2.1	PAIR_BLS381_another()	260
8.23.2.2	PAIR_BLS381_ate()	261
8.23.2.3	PAIR_BLS381_double_ate()	261
8.23.2.4	PAIR_BLS381_fexp()	261
8.23.2.5	PAIR_BLS381_G1mul()	263
8.23.2.6	PAIR_BLS381_G2mul()	263
8.23.2.7	PAIR_BLS381_GTmember()	263
8.23.2.8	PAIR_BLS381_GTpow()	264
8.23.2.9	PAIR_BLS381_initmp()	264
8.23.2.10	PAIR_BLS381_miller()	264
8.23.2.11	PAIR_BLS381_nbits()	264
8.23.3	Variable Documentation	265
8.23.3.1	CURVE_BB_BLS381	265
8.23.3.2	CURVE_Bnx_BLS381	265
8.23.3.3	CURVE_Cru_BLS381	265
8.23.3.4	CURVE_SB_BLS381	265
8.23.3.5	CURVE_W_BLS381	265
8.23.3.6	CURVE_WB_BLS381	266
8.24	pbcsupport.h File Reference	266



8.24.1	Detailed Description	266
8.24.2	Macro Definition Documentation	266
8.24.2.1	TIME_SLOT_MINUTES	267
8.24.3	Function Documentation	267
8.24.3.1	AES_GCM_DECRYPT()	267
8.24.3.2	AES_GCM_ENCRYPT()	267
8.24.3.3	GET_TIME()	268
8.24.3.4	HASH_ALL()	268
8.24.3.5	HASH_ID()	268
8.24.3.6	mhashit()	269
8.24.3.7	today()	269
8.25	randapi.h File Reference	269
8.25.1	Detailed Description	270
8.25.2	Function Documentation	270
8.25.2.1	CREATE_CSPRNG()	270
8.25.2.2	KILL_CSPRNG()	270
8.26	rsa_2048.h File Reference	270
8.26.1	Detailed Description	271
8.26.2	Macro Definition Documentation	271
8.26.2.1	HASH_TYPE_RSA_2048	271
8.26.2.2	RFS_2048	271
8.26.3	Function Documentation	271
8.26.3.1	RSA_2048_DECRYPT()	271
8.26.3.2	RSA_2048_ENCRYPT()	272
8.26.3.3	RSA_2048_fromOctet()	272
8.26.3.4	RSA_2048_KEY_PAIR()	272
8.26.3.5	RSA_2048_PRIVATE_KEY_KILL()	273
8.27	rsa_support.h File Reference	273
8.27.1	Detailed Description	273
8.27.2	Macro Definition Documentation	274

8.27.2.1	MAX_RSA_BYTES	274
8.27.3	Function Documentation	274
8.27.3.1	OAEP_DECODE()	274
8.27.3.2	OAEP_ENCODE()	274
8.27.3.3	PKCS15()	275
8.28	utils.c File Reference	275
8.28.1	Detailed Description	276
8.28.2	Function Documentation	276
8.28.2.1	amcl_bin2hex()	276
8.28.2.2	amcl_hex2bin()	277
8.28.2.3	amcl_print_hex()	277
8.28.2.4	generateOTP()	277
8.28.2.5	generateRandom()	278
8.29	utils.h File Reference	278
8.29.1	Detailed Description	278
8.29.2	Function Documentation	278
8.29.2.1	amcl_bin2hex()	278
8.29.2.2	amcl_hex2bin()	279
8.29.2.3	amcl_print_hex()	279
8.29.2.4	generateOTP()	279
8.29.2.5	generateRandom()	280
8.30	version.c File Reference	280
8.30.1	Detailed Description	280
8.30.2	Function Documentation	281
8.30.2.1	amcl_version()	281
8.31	wcc_BLS381.h File Reference	281
8.31.1	Detailed Description	282
8.31.2	Macro Definition Documentation	282
8.31.2.1	PIV	282
8.31.2.2	PTAG	282

8.31.2.3	TIME_SLOT_MINUTES	282
8.31.2.4	WCC_INVALID_POINT	283
8.31.2.5	WCC_OK	283
8.31.2.6	WCC_PFS_BLS381	283
8.31.2.7	WCC_PGS_BLS381	283
8.31.3	Function Documentation	283
8.31.3.1	WCC_BLS381_GET_G1_MULTIPLE()	283
8.31.3.2	WCC_BLS381_GET_G2_MULTIPLE()	284
8.31.3.3	WCC_BLS381_Hq()	284
8.31.3.4	WCC_BLS381_RANDOM_GENERATE()	285
8.31.3.5	WCC_BLS381_RECEIVER_KEY()	285
8.31.3.6	WCC_BLS381_RECOMBINE_G1()	286
8.31.3.7	WCC_BLS381_RECOMBINE_G2()	286
8.31.3.8	WCC_BLS381_SENDER_KEY()	287
8.32	x509.h File Reference	287
8.32.1	Detailed Description	288
8.32.2	Function Documentation	288
8.32.2.1	X509_extract_cert()	288
8.32.2.2	X509_extract_cert_sig()	288
8.32.2.3	X509_extract_public_key()	289
8.32.2.4	X509_find_entity_property()	289
8.32.2.5	X509_find_expiry_date()	290
8.32.2.6	X509_find_issuer()	290
8.32.2.7	X509_find_start_date()	290
8.32.2.8	X509_find_subject()	291
8.32.2.9	X509_find_validity()	291
<b>Index</b>		<b>293</b>



# Chapter 1

## Apache Milagro Crypto Library (AMCL)

AMCL is a standards compliant C cryptographic library with no external dependencies, specifically designed to support the Internet of Things (IoT).

AMCL is provided in C language but includes a Python wrapper for some components as an aid for development work.

### 1.1 Project page

The official project page is hosted at [Apache Milagro \(incubating\)](#)

### 1.2 License

Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

### 1.3 Platforms

The software can be compiled and installed for these operating systems;

- Linux
- Windows
- Mac OS

## 1.4 Downloads

The source code is available from here;

```
git clone https://github.com/milagro-crypto/milagro-crypto-c
```

## 1.5 Installation

There are instructions for building for [Linux](#), [Mac OS](#) and [Windows](#).

## Chapter 2

# Linux

### Software dependencies

CMake is required to build the library and can usually be installed from the operating system package manager.

- `sudo apt-get install cmake`

If not, then you can download it from [www.cmake.org](http://www.cmake.org)

In order to use the Python language wrapper install [Python](#)

The C Foreign Function Interface for Python [CFFI](#) module is also required if you wish to use the Python module.

- `sudo pip install cffi`

In order to build the documentation [doxygen](#) is required.

### Quick Start

A Makefile is present at the project root that reads the options defined in `config.mk`. Change these options and then type `make` to build and test the library.

If [docker](#) is installed then type `make dbuild` to build and test the library in a docker container.

## Manual build

The default build is for 64 bit machines, Elliptic curve BN254CX and curve type Weierstrass

1. `mkdir target/build`
2. `cd target/build`
3. `cmake -D CMAKE_INSTALL_PREFIX=/opt/amcl ../..`
4. `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./`
5. `make`
6. `make test`
7. `make doc`
8. `sudo make install`

The build can be configured using by setting flags on the command line i.e.

1. `cmake -D CMAKE_INSTALL_PREFIX=/opt/amcl -D WORD_LENGTH=32 ../..`

list available CMake options

1. `cmake -LH`

## Uninstall software

- `sudo make uninstall`

## Building an installer

After having built the libraries you can build a binary installer and a source distribution by running this command

- `make package`



## Chapter 3

# Mac OS

### Software dependencies

Install [Homebrew](#)

- `ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`

Install [cmake](#)

- `brew install cmake`

In order to use the Python language wrapper install [Python](#)

The C Foreign Function Interface for Python [CFFI](#) module is also required if you wish to use the Python module.

- `brew install pkg-config libffi`
- `sudo pip install cffi`

In order to build the documentation [doxygen](#) is required.

- `brew install doxygen`

### Build Instructions

The default build is for 64 bit machines, Elliptic curve BN254CX and curve type Weierstrass

1. `mkdir -p target/build`
2. `cd target/build`
3. `cmake ../..`
4. `make`
5. `make test`
6. `make doc`
7. `sudo make install`

The build can be configured using by setting flags on the command line i.e.

1. `cmake -DWORD_LENGTH=32 ../..`

### Uninstall software

- `sudo make uninstall`



# Chapter 4

## Windows

### Software dependencies

Minimalist GNU for Windows **MinGW** provides the tool set used to build the library and should be installed. When the MinGW installer starts select the mingw32-base and mingw32-gcc-g++ components. From the menu select "Installation" -> "Apply Changes", then click "Apply". Finally add C:\MinGW\bin to the PATH variable.

CMake is required to build the library and can be downloaded from [www.cmake.org](http://www.cmake.org)

In order to use the Python language wrapper install **Python**

The C Foreign Function Interface for Python **CFFI** module is also required, if you wish to use the Python module.

- pip install cffi

In order to build the documentation **doxygen** is required.

### Build Instructions

Start a command prompt as an administrator

The default build is for 64 bit machines, Elliptic curve BN254CX and curve type Weierstrass

1. mkdir target\build
2. cd target\build
3. cmake -G "MinGW Makefiles" -D WORD\_SIZE=64 ..\..
4. mingw32-make
5. mingw32-make test
6. mingw32-make doc
7. mingw32-make install

Post install append the PATH system variable to point to the install ./lib.

My Computer -> Properties -> Advanced > Environment Variables

The build can be configured using by setting flags on the command line i.e.

1. cmake -G "MinGW Makefiles" -D WORD\_SIZE=64 -D BUILD\_PYTHON=on ..\..

### Uninstall software

- `mingw32-make uninstall`

### Building an installer

After having built the libraries you can build a Windows installer using this command

- `sudo mingw32-make package`

In order for this to work [NSIS](#) has to have been installed

# Chapter 5

## Data Structure Index

### 5.1 Data Structures

Here are the data structures with brief descriptions:

- [amcl\\_aes](#)
  - AES instance . . . . . 13
- [csprng](#)
  - Cryptographically secure pseudo-random number generator instance . . . . . 14
- [ECP2\\_BLS381](#)
  - ECP2 Structure - Elliptic Curve Point over quadratic extension field . . . . . 15
- [ECP\\_BLS381](#)
  - ECP structure - Elliptic Curve Point over base field . . . . . 16
- [FP12\\_BLS381](#)
  - FP12 Structure - towered over three FP4 . . . . . 17
- [FP2\\_BLS381](#)
  - FP2 Structure - quadratic extension field . . . . . 18
- [FP4\\_BLS381](#)
  - FP4 Structure - towered over two FP2 . . . . . 19
- [FP\\_BLS381](#)
  - FP Structure - quadratic extension field . . . . . 19
- [gcm](#)
  - GCM mode instance, using AES internally . . . . . 20
- [hash256](#)
  - SHA256 hash function instance . . . . . 22
- [hash512](#)
  - SHA384-512 hash function instance . . . . . 23
- [octet](#)
  - Portable representation of a big positive number . . . . . 24
- [PAILLIER\\_private\\_key](#)
  - Paillier Private Key . . . . . 24
- [PAILLIER\\_public\\_key](#)
  - Paillier Public Key . . . . . 26
- [pktype](#)
  - Public key type . . . . . 27
- [rsa\\_private\\_key\\_2048](#)
  - Integer Factorisation Private Key . . . . . 28
- [rsa\\_public\\_key\\_2048](#)
  - Integer Factorisation Public Key . . . . . 29
- [sha3](#)
  - SHA3 hash function instance . . . . . 30



# Chapter 6

## File Index

### 6.1 File List

Here is a list of all documented files with brief descriptions:

<b>amcl.h</b>	.....	<b>??</b>
<a href="#">arch.h</a>	Architecture Header File	31
<a href="#">big_1024_58.h</a>	BIG Header File	33
<a href="#">big_384_58.h</a>	BIG Header File	61
<a href="#">big_512_60.h</a>	BIG Header File	88
<a href="#">bls_BLS381.h</a>	BLS Header file	116
<a href="#">config_big_1024_58.h</a>	Config BIG Header File	122
<a href="#">config_big_384_58.h</a>	Config BIG Header File	122
<a href="#">config_big_512_60.h</a>	Config BIG Header File	123
<b>config_curve_BLS381.h</b>	.....	<b>??</b>
<a href="#">config_ff_2048.h</a>	COnfig FF Header File	124
<a href="#">config_ff_4096.h</a>	COnfig FF Header File	125
<b>config_field_BLS381.h</b>	.....	<b>??</b>
<b>config_test.h</b>	.....	<b>??</b>
<a href="#">ecdh_BLS381.h</a>	ECDH Header file for implementation of standard EC protocols	125
<a href="#">ecdh_support.h</a>	ECDH Support Header File	130
<a href="#">ecp2_BLS381.h</a>	ECP2 Header File	134
<a href="#">ecp_BLS381.h</a>	ECP Header File	144
<a href="#">ff_2048.h</a>	FF Header File	160
<a href="#">ff_4096.h</a>	FF Header File	176

<a href="#">fp12_BLS381.h</a>		
	FP12 Header File . . . . .	191
<a href="#">fp2_BLS381.h</a>		
	FP2 Header File . . . . .	202
<a href="#">fp4_BLS381.h</a>		
	FP4 Header File . . . . .	213
<a href="#">fp_BLS381.h</a>		
	FP Header File . . . . .	228
<a href="#">mpin_BLS381.h</a>		
	M-Pin Header file . . . . .	239
<a href="#">paillier.h</a>		
	Paillier declarations . . . . .	254
<a href="#">pair_BLS381.h</a>		
	PAIR Header File . . . . .	259
<a href="#">pbc_support.h</a>		
	Auxiliary functions for Pairing-based protocols . . . . .	266
<a href="#">randapi.h</a>		
	PRNG API File . . . . .	269
<a href="#">rsa_2048.h</a>		
	RSA Header file for implementation of RSA protocol . . . . .	270
<a href="#">rsa_support.h</a>		
	RSA Support Header File . . . . .	273
<a href="#">utils.c</a>		
	AMCL Support functions for M-Pin servers . . . . .	275
<a href="#">utils.h</a>		
	Utility functions Header File . . . . .	278
<a href="#">version.c</a>		
	AMCL version support function . . . . .	280
<b>version.h</b>		<b>??</b>
<a href="#">wcc_BLS381.h</a>		
	WCC Header File . . . . .	281
<a href="#">x509.h</a>		
	X509 function Header File . . . . .	287



# Chapter 7

## Data Structure Documentation

### 7.1 amcl\_aes Struct Reference

AES instance.

```
#include <amcl.h>
```

#### Data Fields

- int `Nk`
- int `Nr`
- int `mode`
- `unsign32` `fkey` [60]
- `unsign32` `rkey` [60]
- char `f` [16]

#### 7.1.1 Field Documentation

##### 7.1.1.1 `f`

```
char amcl_aes::f[16]
```

buffer for chaining vector

##### 7.1.1.2 `fkey`

```
unsign32 amcl_aes::fkey[60]
```

subkeys for encrypton

### 7.1.1.3 mode

```
int amcl_aes::mode
```

AES mode of operation

### 7.1.1.4 Nk

```
int amcl_aes::Nk
```

AES Key Length

### 7.1.1.5 Nr

```
int amcl_aes::Nr
```

AES Number of rounds

### 7.1.1.6 rkey

```
unsign32 amcl_aes::rkey[60]
```

subkeys for decrypton

The documentation for this struct was generated from the following file:

- amcl.h

## 7.2 csprng Struct Reference

Cryptographically secure pseudo-random number generator instance.

```
#include <amcl.h>
```

### Data Fields

- [unsign32 ira](#) [NK]
- [int rndptr](#)
- [unsign32 borrow](#)
- [int pool\\_ptr](#)
- [char pool](#) [32]

### 7.2.1 Field Documentation

#### 7.2.1.1 borrow

```
unsign32 csprng::borrow
```

borrow as a result of subtraction

#### 7.2.1.2 ira

```
unsign32 csprng::ira[NK]
```

random number array

#### 7.2.1.3 pool

```
char csprng::pool[32]
```

random pool

#### 7.2.1.4 pool\_ptr

```
int csprng::pool_ptr
```

pointer into random pool

#### 7.2.1.5 rndptr

```
int csprng::rndptr
```

pointer into array

The documentation for this struct was generated from the following file:

- `amcl.h`

## 7.3 ECP2\_BLS381 Struct Reference

ECP2 Structure - Elliptic Curve Point over quadratic extension field.

```
#include <ecp2_BLS381.h>
```

### Data Fields

- [FP2\\_BLS381 x](#)
- [FP2\\_BLS381 y](#)
- [FP2\\_BLS381 z](#)

## 7.3.1 Field Documentation

### 7.3.1.1 x

`FP2_BLS381 ECP2_BLS381::x`

x-coordinate of point

### 7.3.1.2 y

`FP2_BLS381 ECP2_BLS381::y`

y-coordinate of point

### 7.3.1.3 z

`FP2_BLS381 ECP2_BLS381::z`

z-coordinate of point

The documentation for this struct was generated from the following file:

- [ecp2\\_BLS381.h](#)

## 7.4 ECP\_BLS381 Struct Reference

ECP structure - Elliptic Curve Point over base field.

```
#include <ecp_BLS381.h>
```

### Data Fields

- [FP\\_BLS381 x](#)
- [FP\\_BLS381 y](#)
- [FP\\_BLS381 z](#)

### 7.4.1 Field Documentation

#### 7.4.1.1 x

`FP_BLS381 ECP_BLS381::x`

x-coordinate of point

#### 7.4.1.2 y

`FP_BLS381 ECP_BLS381::y`

y-coordinate of point. Not needed for Montgomery representation

#### 7.4.1.3 z

`FP_BLS381 ECP_BLS381::z`

z-coordinate of point

The documentation for this struct was generated from the following file:

- [ecp\\_BLS381.h](#)

## 7.5 FP12\_BLS381 Struct Reference

FP12 Structure - towered over three FP4.

```
#include <fp12_BLS381.h>
```

### Data Fields

- [FP4\\_BLS381 a](#)
- [FP4\\_BLS381 b](#)
- [FP4\\_BLS381 c](#)
- [int type](#)

### 7.5.1 Field Documentation

#### 7.5.1.1 a

`FP4_BLS381 FP12_BLS381::a`

first part of FP12

### 7.5.1.2 b

```
FP4_BLS381 FP12_BLS381::b
```

second part of FP12

### 7.5.1.3 c

```
FP4_BLS381 FP12_BLS381::c
```

third part of FP12

### 7.5.1.4 type

```
int FP12_BLS381::type
```

Type

The documentation for this struct was generated from the following file:

- [fp12\\_BLS381.h](#)

## 7.6 FP2\_BLS381 Struct Reference

FP2 Structure - quadratic extension field.

```
#include <fp2_BLS381.h>
```

### Data Fields

- [FP\\_BLS381 a](#)
- [FP\\_BLS381 b](#)

### 7.6.1 Field Documentation

#### 7.6.1.1 a

```
FP_BLS381 FP2_BLS381::a
```

real part of FP2

### 7.6.1.2 b

`FP_BLS381 FP2_BLS381::b`

imaginary part of FP2

The documentation for this struct was generated from the following file:

- [fp2\\_BLS381.h](#)

## 7.7 FP4\_BLS381 Struct Reference

FP4 Structure - towered over two FP2.

```
#include <fp4_BLS381.h>
```

### Data Fields

- [FP2\\_BLS381 a](#)
- [FP2\\_BLS381 b](#)

### 7.7.1 Field Documentation

#### 7.7.1.1 a

`FP2_BLS381 FP4_BLS381::a`

real part of FP4

#### 7.7.1.2 b

`FP2_BLS381 FP4_BLS381::b`

imaginary part of FP4

The documentation for this struct was generated from the following file:

- [fp4\\_BLS381.h](#)

## 7.8 FP\_BLS381 Struct Reference

FP Structure - quadratic extension field.

```
#include <fp_BLS381.h>
```

## Data Fields

- [BIG\\_384\\_58](#) g
- [sign32](#) XES

### 7.8.1 Field Documentation

#### 7.8.1.1 g

[BIG\\_384\\_58](#) FP\_BLS381::g

Big representation of field element

#### 7.8.1.2 XES

[sign32](#) FP\_BLS381::XES

Excess

The documentation for this struct was generated from the following file:

- [fp\\_BLS381.h](#)

## 7.9 gcm Struct Reference

GCM mode instance, using AES internally.

```
#include <amcl.h>
```

## Data Fields

- [unsign32](#) table [128][4]
- [uchar](#) stateX [16]
- [uchar](#) Y\_0 [16]
- [unsign32](#) lenA [2]
- [unsign32](#) lenC [2]
- [int](#) status
- [amcl\\_aes](#) a

### 7.9.1 Field Documentation



### 7.9.1.1 a

```
amcl_aes gcm::a
```

Internal Instance of AMCL\_AES cipher

### 7.9.1.2 lenA

```
unsign32 gcm::lenA[2]
```

GCM 64-bit length of header

### 7.9.1.3 lenC

```
unsign32 gcm::lenC[2]
```

GCM 64-bit length of ciphertext

### 7.9.1.4 stateX

```
uchar gcm::stateX[16]
```

GCM Internal State

### 7.9.1.5 status

```
int gcm::status
```

GCM Status

### 7.9.1.6 table

```
unsign32 gcm::table[128][4]
```

2k byte table

### 7.9.1.7 Y\_0

```
uchar gcm::Y_0[16]
```

GCM Internal State

The documentation for this struct was generated from the following file:

- amcl.h

## 7.10 hash256 Struct Reference

SHA256 hash function instance.

```
#include <amcl.h>
```

### Data Fields

- [unsign32 length](#) [2]
- [unsign32 h](#) [8]
- [unsign32 w](#) [80]
- [int hlen](#)

### 7.10.1 Field Documentation

#### 7.10.1.1 h

```
unsign32 hash256::h[8]
```

Internal state

#### 7.10.1.2 hlen

```
int hash256::hlen
```

Hash length in bytes

#### 7.10.1.3 length

```
unsign32 hash256::length[2]
```

64-bit input length

#### 7.10.1.4 w

```
unsign32 hash256::w[80]
```

Internal state

The documentation for this struct was generated from the following file:

- [amcl.h](#)

## 7.11 hash512 Struct Reference

SHA384-512 hash function instance.

```
#include <amcl.h>
```

### Data Fields

- [unsign64 length](#) [2]
- [unsign64 h](#) [8]
- [unsign64 w](#) [80]
- [int hlen](#)

### 7.11.1 Field Documentation

#### 7.11.1.1 h

```
unsign64 hash512::h[8]
```

Internal state

#### 7.11.1.2 hlen

```
int hash512::hlen
```

Hash length in bytes

#### 7.11.1.3 length

```
unsign64 hash512::length[2]
```

64-bit input length

#### 7.11.1.4 w

```
unsign64 hash512::w[80]
```

Internal state

The documentation for this struct was generated from the following file:

- [amcl.h](#)

## 7.12 octet Struct Reference

Portable representation of a big positive number.

```
#include <amcl.h>
```

### Data Fields

- int [len](#)
- int [max](#)
- char \* [val](#)

### 7.12.1 Field Documentation

#### 7.12.1.1 len

```
int octet::len
```

length in bytes

#### 7.12.1.2 max

```
int octet::max
```

max length allowed - enforce truncation

#### 7.12.1.3 val

```
char* octet::val
```

byte array

The documentation for this struct was generated from the following file:

- [amcl.h](#)

## 7.13 PAILLIER\_private\_key Struct Reference

Paillier Private Key.

```
#include <paillier.h>
```

## Data Fields

- [BIG\\_1024\\_58 p](#) [HFLEN\_2048]
- [BIG\\_1024\\_58 q](#) [HFLEN\_2048]
- [BIG\\_1024\\_58 lp](#) [HFLEN\_2048]
- [BIG\\_1024\\_58 lq](#) [HFLEN\_2048]
- [BIG\\_1024\\_58 invp](#) [FFLEN\_2048]
- [BIG\\_1024\\_58 invq](#) [FFLEN\_2048]
- [BIG\\_1024\\_58 p2](#) [FFLEN\_2048]
- [BIG\\_1024\\_58 q2](#) [FFLEN\_2048]
- [BIG\\_1024\\_58 mp](#) [HFLEN\_2048]
- [BIG\\_1024\\_58 mq](#) [HFLEN\_2048]

### 7.13.1 Field Documentation

#### 7.13.1.1 invp

[BIG\\_1024\\_58 PAILLIER\\_private\\_key::invp](#)[FFLEN\_2048]

Precomputed  $p^{-1} \pmod{2^m}$

#### 7.13.1.2 invq

[BIG\\_1024\\_58 PAILLIER\\_private\\_key::invq](#)[FFLEN\_2048]

Precomputed  $q^{-1} \pmod{2^m}$

#### 7.13.1.3 lp

[BIG\\_1024\\_58 PAILLIER\\_private\\_key::lp](#)[HFLEN\_2048]

Private Key modulo  $p$  (Euler totient of  $p$ )

#### 7.13.1.4 lq

[BIG\\_1024\\_58 PAILLIER\\_private\\_key::lq](#)[HFLEN\_2048]

Private Key modulo  $q$  (Euler totient of  $q$ )

#### 7.13.1.5 mp

[BIG\\_1024\\_58 PAILLIER\\_private\\_key::mp](#)[HFLEN\_2048]

Precomputed  $L(g^{lp} \pmod{p^2})^{-1}$

### 7.13.1.6 mq

[BIG\\_1024\\_58](#) PAILLIER\_private\_key::mq[HFLEN\_2048]

Precomputed  $L(g^{mq} \pmod{q^2})^{-1}$

### 7.13.1.7 p

[BIG\\_1024\\_58](#) PAILLIER\_private\_key::p[HFLEN\_2048]

Secret Prime

### 7.13.1.8 p2

[BIG\\_1024\\_58](#) PAILLIER\_private\_key::p2[FFLEN\_2048]

Precomputed  $p^2$

### 7.13.1.9 q

[BIG\\_1024\\_58](#) PAILLIER\_private\_key::q[HFLEN\_2048]

Secret Prime

### 7.13.1.10 q2

[BIG\\_1024\\_58](#) PAILLIER\_private\_key::q2[FFLEN\_2048]

Precomputed  $q^2$

The documentation for this struct was generated from the following file:

- [paillier.h](#)

## 7.14 PAILLIER\_public\_key Struct Reference

Paillier Public Key.

```
#include <paillier.h>
```

### Data Fields

- [BIG\\_512\\_60 n](#) [FFLEN\_4096]
- [BIG\\_512\\_60 g](#) [FFLEN\_4096]
- [BIG\\_512\\_60 n2](#) [FFLEN\_4096]

## 7.14.1 Field Documentation

### 7.14.1.1 g

[BIG\\_512\\_60](#) PAILLIER\_public\_key::g [FFLEN\_4096]

Public Base -  $g = n + 1$

### 7.14.1.2 n

[BIG\\_512\\_60](#) PAILLIER\_public\_key::n [FFLEN\_4096]

Paillier Modulus -  $n = pq$

### 7.14.1.3 n2

[BIG\\_512\\_60](#) PAILLIER\_public\_key::n2 [FFLEN\_4096]

Precomputed  $n^2$

The documentation for this struct was generated from the following file:

- [paillier.h](#)

## 7.15 pktype Struct Reference

Public key type.

```
#include <x509.h>
```

### Data Fields

- int [type](#)
- int [hash](#)
- int [curve](#)

## 7.15.1 Field Documentation

### 7.15.1.1 curve

```
int pktype::curve
```

elliptic curve used or RSA key length in bits

### 7.15.1.2 hash

```
int pktype::hash
```

hash type

### 7.15.1.3 type

```
int pktype::type
```

signature type (ECC or RSA)

The documentation for this struct was generated from the following file:

- [x509.h](#)

## 7.16 rsa\_private\_key\_2048 Struct Reference

Integer Factorisation Private Key.

```
#include <rsa_2048.h>
```

### Data Fields

- [BIG\\_1024\\_58 p](#) [FFLEN\_2048/2]
- [BIG\\_1024\\_58 q](#) [FFLEN\_2048/2]
- [BIG\\_1024\\_58 dp](#) [FFLEN\_2048/2]
- [BIG\\_1024\\_58 dq](#) [FFLEN\_2048/2]
- [BIG\\_1024\\_58 c](#) [FFLEN\_2048/2]

### 7.16.1 Field Documentation

#### 7.16.1.1 c

```
BIG\_1024\_58 rsa_private_key_2048::c [FFLEN_2048/2]
```

$1/p \bmod q$



### 7.16.1.2 `dp`

[BIG\\_1024\\_58](#) `rsa_private_key_2048::dp` [`FFLEN_2048/2`]

decrypting exponent mod (p-1)

### 7.16.1.3 `dq`

[BIG\\_1024\\_58](#) `rsa_private_key_2048::dq` [`FFLEN_2048/2`]

decrypting exponent mod (q-1)

### 7.16.1.4 `p`

[BIG\\_1024\\_58](#) `rsa_private_key_2048::p` [`FFLEN_2048/2`]

secret prime p

### 7.16.1.5 `q`

[BIG\\_1024\\_58](#) `rsa_private_key_2048::q` [`FFLEN_2048/2`]

secret prime q

The documentation for this struct was generated from the following file:

- [rsa\\_2048.h](#)

## 7.17 `rsa_public_key_2048` Struct Reference

Integer Factorisation Public Key.

```
#include <rsa_2048.h>
```

### Data Fields

- [sign32 e](#)
- [BIG\\_1024\\_58 n](#) [`FFLEN_2048`]

### 7.17.1 Field Documentation

### 7.17.1.1 e

```
sign32 rsa_public_key_2048::e
```

RSA exponent (typically 65537)

### 7.17.1.2 n

```
BIG_1024_58 rsa_public_key_2048::n[FFLEN_2048]
```

An array of BIGs to store public key

The documentation for this struct was generated from the following file:

- [rsa\\_2048.h](#)

## 7.18 sha3 Struct Reference

SHA3 hash function instance.

```
#include <amcl.h>
```

### Data Fields

- [unsign64 length](#)
- [unsign64 S \[5\]\[5\]](#)
- [int rate](#)
- [int len](#)

### 7.18.1 Field Documentation

#### 7.18.1.1 len

```
int sha3::len
```

Hash length in bytes

#### 7.18.1.2 length

```
unsign64 sha3::length
```

64-bit input length

#### 7.18.1.3 rate

```
int sha3::rate
```

TODO

#### 7.18.1.4 S

```
unsign64 sha3::S[5][5]
```

Internal state

The documentation for this struct was generated from the following file:

- [amcl.h](#)

# Chapter 8

## File Documentation

### 8.1 arch.h File Reference

Architecture Header File.

#### Macros

- #define [CHUNK](#) 64
- #define [byte](#) unsigned char
- #define [sign32](#) \_\_int32
- #define [sign8](#) signed char
- #define [sign16](#) short int
- #define [sign64](#) long long
- #define [unsign32](#) unsigned \_\_int32
- #define [unsign64](#) unsigned long long
- #define [uchar](#) unsigned char
- #define [chunk](#) \_\_int64

#### 8.1.1 Detailed Description

##### Author

Mike Scott

##### Date

23rd February 2016 Specify Processor Architecture

#### 8.1.2 Macro Definition Documentation

### 8.1.2.1 byte

```
#define byte unsigned char
```

8-bit unsigned integer

### 8.1.2.2 CHUNK

```
#define CHUNK 64
```

size of chunk in bits = wordlength of computer = 16, 32 or 64. Note not all curve options are supported on 16-bit processors - see rom.c

### 8.1.2.3 chunk

```
#define chunk __int64
```

C type corresponding to word length Note - no 128-bit type available

### 8.1.2.4 sign16

```
#define sign16 short int
```

16-bit signed integer

### 8.1.2.5 sign32

```
#define sign32 __int32
```

32-bit signed integer

### 8.1.2.6 sign64

```
#define sign64 long long
```

64-bit signed integer

### 8.1.2.7 sign8

```
#define sign8 signed char
```

8-bit signed integer

### 8.1.2.8 uchar

```
#define uchar unsigned char
```

Unsigned char

### 8.1.2.9 unsign32

```
#define unsign32 unsigned __int32
```

32-bit unsigned integer

### 8.1.2.10 unsign64

```
#define unsign64 unsigned long long
```

64-bit unsigned integer

## 8.2 big\_1024\_58.h File Reference

BIG Header File.

```
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include "arch.h"
#include "amcl.h"
#include "config_big_1024_58.h"
```

### Macros

- #define `BIGBITS_1024_58` (8\*MODBYTES\_1024\_58)
- #define `NLEN_1024_58` (1+((8\*MODBYTES\_1024\_58-1)/BASEBITS\_1024\_58))
- #define `DNLEN_1024_58` 2\*NLEN\_1024\_58
- #define `BMASK_1024_58` (((chunk)1<<<BASEBITS\_1024\_58)-1)
- #define `NEXCESS_1024_58` (1<<<(CHUNK-BASEBITS\_1024\_58-1))
- #define `HBITS_1024_58` (BASEBITS\_1024\_58/2)
- #define `HMASK_1024_58` (((chunk)1<<<HBITS\_1024\_58)-1)

### Typedefs

- typedef `chunk` `BIG_1024_58`[`NLEN_1024_58`]
- typedef `chunk` `DBIG_1024_58`[`DNLEN_1024_58`]

## Functions

- int [BIG\\_1024\\_58\\_iszilch](#) ([BIG\\_1024\\_58](#) x)  
*Tests for BIG equal to zero.*
- int [BIG\\_1024\\_58\\_isunity](#) ([BIG\\_1024\\_58](#) x)  
*Tests for BIG equal to one.*
- int [BIG\\_1024\\_58\\_diszilch](#) ([DBIG\\_1024\\_58](#) x)  
*Tests for DBIG equal to zero.*
- void [BIG\\_1024\\_58\\_output](#) ([BIG\\_1024\\_58](#) x)  
*Outputs a BIG number to the console.*
- void [BIG\\_1024\\_58\\_rawoutput](#) ([BIG\\_1024\\_58](#) x)  
*Outputs a BIG number to the console in raw form (for debugging)*
- void [BIG\\_1024\\_58\\_cswap](#) ([BIG\\_1024\\_58](#) x, [BIG\\_1024\\_58](#) y, int s)  
*Conditional constant time swap of two BIG numbers.*
- void [BIG\\_1024\\_58\\_cmove](#) ([BIG\\_1024\\_58](#) x, [BIG\\_1024\\_58](#) y, int s)  
*Conditional copy of BIG number.*
- void [BIG\\_1024\\_58\\_dcmove](#) ([BIG\\_1024\\_58](#) x, [BIG\\_1024\\_58](#) y, int s)  
*Conditional copy of DBIG number.*
- void [BIG\\_1024\\_58\\_toBytes](#) (char \*a, [BIG\\_1024\\_58](#) x)  
*Convert from BIG number to byte array.*
- void [BIG\\_1024\\_58\\_fromBytes](#) ([BIG\\_1024\\_58](#) x, char \*a)  
*Convert to BIG number from byte array.*
- void [BIG\\_1024\\_58\\_fromBytesLen](#) ([BIG\\_1024\\_58](#) x, char \*a, int s)  
*Convert to BIG number from byte array of given length.*
- void [BIG\\_1024\\_58\\_dfromBytesLen](#) ([DBIG\\_1024\\_58](#) x, char \*a, int s)  
*Convert to DBIG number from byte array of given length.*
- void [BIG\\_1024\\_58\\_doutput](#) ([DBIG\\_1024\\_58](#) x)  
*Outputs a DBIG number to the console.*
- void [BIG\\_1024\\_58\\_drawoutput](#) ([DBIG\\_1024\\_58](#) x)  
*Outputs a DBIG number to the console.*
- void [BIG\\_1024\\_58\\_rcopy](#) ([BIG\\_1024\\_58](#) x, const [BIG\\_1024\\_58](#) y)  
*Copy BIG from Read-Only Memory to a BIG.*
- void [BIG\\_1024\\_58\\_copy](#) ([BIG\\_1024\\_58](#) x, [BIG\\_1024\\_58](#) y)  
*Copy BIG to another BIG.*
- void [BIG\\_1024\\_58\\_dcopy](#) ([DBIG\\_1024\\_58](#) x, [DBIG\\_1024\\_58](#) y)  
*Copy DBIG to another DBIG.*
- void [BIG\\_1024\\_58\\_dsucopy](#) ([DBIG\\_1024\\_58](#) x, [BIG\\_1024\\_58](#) y)  
*Copy BIG to upper half of DBIG.*
- void [BIG\\_1024\\_58\\_dscopy](#) ([DBIG\\_1024\\_58](#) x, [BIG\\_1024\\_58](#) y)  
*Copy BIG to lower half of DBIG.*
- void [BIG\\_1024\\_58\\_sdcopy](#) ([BIG\\_1024\\_58](#) x, [DBIG\\_1024\\_58](#) y)  
*Copy lower half of DBIG to a BIG.*
- void [BIG\\_1024\\_58\\_sducopy](#) ([BIG\\_1024\\_58](#) x, [DBIG\\_1024\\_58](#) y)  
*Copy upper half of DBIG to a BIG.*
- void [BIG\\_1024\\_58\\_zero](#) ([BIG\\_1024\\_58](#) x)  
*Set BIG to zero.*
- void [BIG\\_1024\\_58\\_dzero](#) ([DBIG\\_1024\\_58](#) x)  
*Set DBIG to zero.*
- void [BIG\\_1024\\_58\\_one](#) ([BIG\\_1024\\_58](#) x)  
*Set BIG to one (unity)*
- void [BIG\\_1024\\_58\\_invmod2m](#) ([BIG\\_1024\\_58](#) x)

- Set BIG to inverse mod  $2^{256}$ .*

  - void `BIG_1024_58_add` (`BIG_1024_58` x, `BIG_1024_58` y, `BIG_1024_58` z)
- Set BIG to sum of two BIGs - output not normalised.*

  - void `BIG_1024_58_or` (`BIG_1024_58` x, `BIG_1024_58` y, `BIG_1024_58` z)
- Set BIG to logical or of two BIGs - output normalised.*

  - void `BIG_1024_58_inc` (`BIG_1024_58` x, int i)
- Increment BIG by a small integer - output not normalised.*

  - void `BIG_1024_58_sub` (`BIG_1024_58` x, `BIG_1024_58` y, `BIG_1024_58` z)
- Set BIG to difference of two BIGs.*

  - void `BIG_1024_58_dec` (`BIG_1024_58` x, int i)
- Decrement BIG by a small integer - output not normalised.*

  - void `BIG_1024_58_dadd` (`DBIG_1024_58` x, `DBIG_1024_58` y, `DBIG_1024_58` z)
- Set DBIG to sum of two DBIGs.*

  - void `BIG_1024_58_dsub` (`DBIG_1024_58` x, `DBIG_1024_58` y, `DBIG_1024_58` z)
- Set DBIG to difference of two DBIGs.*

  - void `BIG_1024_58_imul` (`BIG_1024_58` x, `BIG_1024_58` y, int i)
- Multiply BIG by a small integer - output not normalised.*

  - `chunk` `BIG_1024_58_pmul` (`BIG_1024_58` x, `BIG_1024_58` y, int i)
- Multiply BIG by not-so-small small integer - output normalised.*

  - int `BIG_1024_58_div3` (`BIG_1024_58` x)
- Divide BIG by 3 - output normalised.*

  - void `BIG_1024_58_pxm` (`DBIG_1024_58` x, `BIG_1024_58` y, int i)
- Multiply BIG by even bigger small integer resulting in a DBIG - output normalised.*

  - void `BIG_1024_58_mul` (`DBIG_1024_58` x, `BIG_1024_58` y, `BIG_1024_58` z)
- Multiply BIG by another BIG resulting in DBIG - inputs normalised and output normalised.*

  - void `BIG_1024_58_smul` (`BIG_1024_58` x, `BIG_1024_58` y, `BIG_1024_58` z)
- Multiply BIG by another BIG resulting in another BIG - inputs normalised and output normalised.*

  - void `BIG_1024_58_sqr` (`DBIG_1024_58` x, `BIG_1024_58` y)
- Square BIG resulting in a DBIG - input normalised and output normalised.*

  - void `BIG_1024_58_monty` (`BIG_1024_58` a, `BIG_1024_58` md, `chunk` MC, `DBIG_1024_58` d)
- Montgomery reduction of a DBIG to a BIG - input normalised and output normalised.*

  - void `BIG_1024_58_shl` (`BIG_1024_58` x, int s)
- Shifts a BIG left by any number of bits - input must be normalised, output normalised.*

  - int `BIG_1024_58_fshl` (`BIG_1024_58` x, int s)
- Fast shifts a BIG left by a small number of bits - input must be normalised, output will be normalised.*

  - void `BIG_1024_58_dshl` (`DBIG_1024_58` x, int s)
- Shifts a DBIG left by any number of bits - input must be normalised, output normalised.*

  - void `BIG_1024_58_shr` (`BIG_1024_58` x, int s)
- Shifts a BIG right by any number of bits - input must be normalised, output normalised.*

  - int `BIG_1024_58_ssn` (`BIG_1024_58` r, `BIG_1024_58` a, `BIG_1024_58` m)
- Fast time-critical combined shift by 1 bit, subtract and normalise.*

  - int `BIG_1024_58_fshr` (`BIG_1024_58` x, int s)
- Fast shifts a BIG right by a small number of bits - input must be normalised, output will be normalised.*

  - void `BIG_1024_58_dshr` (`DBIG_1024_58` x, int s)
- Shifts a DBIG right by any number of bits - input must be normalised, output normalised.*

  - `chunk` `BIG_1024_58_split` (`BIG_1024_58` x, `BIG_1024_58` y, `DBIG_1024_58` z, int s)
- Splits a DBIG into two BIGs - input must be normalised, outputs normalised.*

  - `chunk` `BIG_1024_58_norm` (`BIG_1024_58` x)
- Normalizes a BIG number - output normalised.*

  - void `BIG_1024_58_dnorm` (`DBIG_1024_58` x)
- Normalizes a DBIG number - output normalised.*

- int `BIG_1024_58_comp` (`BIG_1024_58 x`, `BIG_1024_58 y`)  
*Compares two BIG numbers. Inputs must be normalised externally.*
- int `BIG_1024_58_dcomp` (`DBIG_1024_58 x`, `DBIG_1024_58 y`)  
*Compares two DBIG numbers. Inputs must be normalised externally.*
- int `BIG_1024_58_nbits` (`BIG_1024_58 x`)  
*Calculate number of bits in a BIG - output normalised.*
- int `BIG_1024_58_dnbits` (`DBIG_1024_58 x`)  
*Calculate number of bits in a DBIG - output normalised.*
- void `BIG_1024_58_mod` (`BIG_1024_58 x`, `BIG_1024_58 n`)  
*Reduce  $x \bmod n$  - input and output normalised.*
- void `BIG_1024_58_sdiv` (`BIG_1024_58 x`, `BIG_1024_58 n`)  
*Divide  $x$  by  $n$  - output normalised.*
- void `BIG_1024_58_dmod` (`BIG_1024_58 x`, `DBIG_1024_58 y`, `BIG_1024_58 n`)  
 *$x=y \bmod n$  - output normalised*
- void `BIG_1024_58_ddiv` (`BIG_1024_58 x`, `DBIG_1024_58 y`, `BIG_1024_58 n`)  
 *$x=y/n$  - output normalised*
- int `BIG_1024_58_parity` (`BIG_1024_58 x`)  
*return parity of BIG, that is the least significant bit*
- int `BIG_1024_58_bit` (`BIG_1024_58 x`, int `i`)  
*return  $i$ -th of BIG*
- int `BIG_1024_58_lastbits` (`BIG_1024_58 x`, int `n`)  
*return least significant bits of a BIG*
- void `BIG_1024_58_random` (`BIG_1024_58 x`, `csprng *r`)  
*Create a random BIG from a random number generator.*
- void `BIG_1024_58_randomnum` (`BIG_1024_58 x`, `BIG_1024_58 n`, `csprng *r`)  
*Create an unbiased random BIG from a random number generator, reduced with respect to a modulus.*
- void `BIG_1024_58_modmul` (`BIG_1024_58 x`, `BIG_1024_58 y`, `BIG_1024_58 z`, `BIG_1024_58 n`)  
*Calculate  $x=y*z \bmod n$ .*
- void `BIG_1024_58_moddiv` (`BIG_1024_58 x`, `BIG_1024_58 y`, `BIG_1024_58 z`, `BIG_1024_58 n`)  
*Calculate  $x=y/z \bmod n$ .*
- void `BIG_1024_58_modsq` (`BIG_1024_58 x`, `BIG_1024_58 y`, `BIG_1024_58 n`)  
*Calculate  $x=y^2 \bmod n$ .*
- void `BIG_1024_58_modneg` (`BIG_1024_58 x`, `BIG_1024_58 y`, `BIG_1024_58 n`)  
*Calculate  $x=-y \bmod n$ .*
- int `BIG_1024_58_jacobi` (`BIG_1024_58 x`, `BIG_1024_58 y`)  
*Calculate jacobi Symbol ( $x/y$ )*
- void `BIG_1024_58_invmodp` (`BIG_1024_58 x`, `BIG_1024_58 y`, `BIG_1024_58 n`)  
*Calculate  $x=1/y \bmod n$ .*
- void `BIG_1024_58_mod2m` (`BIG_1024_58 x`, int `m`)  
*Calculate  $x=x \bmod 2^m$ .*
- void `BIG_1024_58_dmod2m` (`DBIG_1024_58 x`, int `m`)  
*Calculate  $x=x \bmod 2^m$ .*

### 8.2.1 Detailed Description

#### Author

Mike Scott



## 8.2.2 Macro Definition Documentation

### 8.2.2.1 BIGBITS\_1024\_58

```
#define BIGBITS_1024_58 (8*MODBYTES_1024_58)
```

Length in bits

### 8.2.2.2 BMASK\_1024\_58

```
#define BMASK_1024_58 (((chunk)1<<BASEBITS_1024_58)-1)
```

Mask =  $2^{\text{BASEBITS}-1}$

### 8.2.2.3 DNLEN\_1024\_58

```
#define DNLEN_1024_58 2*NLEN_1024_58
```

Double length in bytes

### 8.2.2.4 HBITS\_1024\_58

```
#define HBITS_1024_58 (BASEBITS_1024_58/2)
```

Number of bits in number base divided by 2

### 8.2.2.5 HMASK\_1024\_58

```
#define HMASK_1024_58 (((chunk)1<<HBITS_1024_58)-1)
```

Mask =  $2^{\text{HBITS}-1}$

### 8.2.2.6 NEXCESS\_1024\_58

```
#define NEXCESS_1024_58 (1<<(CHUNK-BASEBITS_1024_58-1))
```

$2^{(\text{CHUNK}-\text{BASEBITS}-1)}$  - digit cannot be multiplied by more than this before normalisation

### 8.2.2.7 NLEN\_1024\_58

```
#define NLEN_1024_58 (1+((8*MODBYTES_1024_58-1)/BASEBITS_1024_58))
```

length in bytes

## 8.2.3 Typedef Documentation

### 8.2.3.1 BIG\_1024\_58

```
typedef chunk BIG_1024_58[NLEN_1024_58]
```

Define type BIG as array of chunks

### 8.2.3.2 DBIG\_1024\_58

```
typedef chunk DBIG_1024_58[DNLEN_1024_58]
```

Define type DBIG as array of chunks

## 8.2.4 Function Documentation

### 8.2.4.1 BIG\_1024\_58\_add()

```
void BIG_1024_58_add (  
    BIG_1024_58 x,  
    BIG_1024_58 y,  
    BIG_1024_58 z )
```

#### Parameters

<i>x</i>	BIG number, sum of other two
<i>y</i>	BIG number
<i>z</i>	BIG number

### 8.2.4.2 BIG\_1024\_58\_bit()

```
int BIG_1024_58_bit (  
    BIG_1024_58 x,  
    int i )
```

#### Parameters

<i>x</i>	BIG number
<i>i</i>	the bit of x to be returned

**Returns**

0 or 1

**8.2.4.3 BIG\_1024\_58\_cmove()**

```
void BIG_1024_58_cmove (
    BIG_1024_58 x,
    BIG_1024_58 y,
    int s )
```

Conditionally copies second parameter to the first (without branching)

**Parameters**

<i>x</i>	a BIG number
<i>y</i>	another BIG number
<i>s</i>	copy takes place if not equal to 0

**8.2.4.4 BIG\_1024\_58\_comp()**

```
int BIG_1024_58_comp (
    BIG_1024_58 x,
    BIG_1024_58 y )
```

**Parameters**

<i>x</i>	first BIG number to be compared
<i>y</i>	second BIG number to be compared

**Returns**-1 is  $x < y$ , 0 if  $x = y$ , 1 if  $x > y$ **8.2.4.5 BIG\_1024\_58\_copy()**

```
void BIG_1024_58_copy (
    BIG_1024_58 x,
    BIG_1024_58 y )
```

**Parameters**

<i>x</i>	BIG number
<i>y</i>	BIG number to be copied

#### 8.2.4.6 BIG\_1024\_58\_cswap()

```
void BIG_1024_58_cswap (
    BIG_1024_58 x,
    BIG_1024_58 y,
    int s )
```

Conditionally swaps parameters in constant time (without branching)

##### Parameters

<i>x</i>	a BIG number
<i>y</i>	another BIG number
<i>s</i>	swap takes place if not equal to 0

#### 8.2.4.7 BIG\_1024\_58\_dadd()

```
void BIG_1024_58_dadd (
    DBIG_1024_58 x,
    DBIG_1024_58 y,
    DBIG_1024_58 z )
```

##### Parameters

<i>x</i>	DBIG number, sum of other two - output not normalised
<i>y</i>	DBIG number
<i>z</i>	DBIG number

#### 8.2.4.8 BIG\_1024\_58\_dcmove()

```
void BIG_1024_58_dcmove (
    BIG_1024_58 x,
    BIG_1024_58 y,
    int s )
```

Conditionally copies second parameter to the first (without branching)

##### Parameters

<i>x</i>	a DBIG number
<i>y</i>	another DBIG number
<i>s</i>	copy takes place if not equal to 0

#### 8.2.4.9 BIG\_1024\_58\_dcomp()

```
int BIG_1024_58_dcomp (
    DBIG_1024_58 x,
    DBIG_1024_58 y )
```

##### Parameters

<i>x</i>	first DBIG number to be compared
<i>y</i>	second DBIG number to be compared

##### Returns

-1 if  $x < y$ , 0 if  $x = y$ , 1 if  $x > y$

#### 8.2.4.10 BIG\_1024\_58\_dcopy()

```
void BIG_1024_58_dcopy (
    DBIG_1024_58 x,
    DBIG_1024_58 y )
```

##### Parameters

<i>x</i>	DBIG number
<i>y</i>	DBIG number to be copied

#### 8.2.4.11 BIG\_1024\_58\_ddiv()

```
void BIG_1024_58_ddiv (
    BIG_1024_58 x,
    DBIG_1024_58 y,
    BIG_1024_58 n )
```

Slow but rarely used. *y* is destroyed.

##### Parameters

<i>x</i>	BIG number, on exit = $y/n$
<i>y</i>	DBIG number
<i>n</i>	Modulus

#### 8.2.4.12 BIG\_1024\_58\_dec()

```
void BIG_1024_58_dec (
    BIG_1024_58 x,
    int i )
```

##### Parameters

<i>x</i>	BIG number to be decremented
<i>i</i>	integer

#### 8.2.4.13 BIG\_1024\_58\_dfromBytesLen()

```
void BIG_1024_58_dfromBytesLen (
    DBIG_1024_58 x,
    char * a,
    int s )
```

##### Parameters

<i>x</i>	DBIG number
<i>a</i>	byte array
<i>s</i>	byte array length

#### 8.2.4.14 BIG\_1024\_58\_diszilch()

```
int BIG_1024_58_diszilch (
    DBIG_1024_58 x )
```

##### Parameters

<i>x</i>	a DBIG number
----------	---------------

##### Returns

1 if zero, else returns 0

## 8.2.4.15 BIG\_1024\_58\_div3()

```
int BIG_1024_58_div3 (  
    BIG_1024_58 x )
```

**Parameters**

$x$	BIG number
-----	------------

**Returns**

Remainder

**8.2.4.16 BIG\_1024\_58\_dmod()**

```
void BIG_1024_58_dmod (
    BIG_1024_58 x,
    DBIG_1024_58 y,
    BIG_1024_58 n )
```

Slow but rarely used.  $y$  is destroyed.

**Parameters**

$x$	BIG number, on exit = $y \bmod n$
$y$	DBIG number
$n$	Modulus

**8.2.4.17 BIG\_1024\_58\_dmod2m()**

```
void BIG_1024_58_dmod2m (
    DBIG_1024_58 x,
    int m )
```

**Truncation****Parameters**

$x$	DBIG number, on reduced mod $2^m$
$m$	new truncated size

**8.2.4.18 BIG\_1024\_58\_dnbits()**

```
int BIG_1024_58_dnbits (
    DBIG_1024_58 x )
```



**Parameters**

<i>x</i>	DBIG number
----------	-------------

**Returns**

Number of bits in *x*

**8.2.4.19 BIG\_1024\_58\_dnorm()**

```
void BIG_1024_58_dnorm (  
    DBIG_1024_58 x )
```

All digits of the input DBIG are reduced mod  $2^{\text{BASEBITS}}$

**Parameters**

<i>x</i>	DBIG number to be normalised
----------	------------------------------

**8.2.4.20 BIG\_1024\_58\_doutput()**

```
void BIG_1024_58_doutput (  
    DBIG_1024_58 x )
```

**Parameters**

<i>x</i>	a DBIG number
----------	---------------

**8.2.4.21 BIG\_1024\_58\_drawoutput()**

```
void BIG_1024_58_drawoutput (  
    DBIG_1024_58 x )
```

**Parameters**

<i>x</i>	a DBIG number
----------	---------------

#### 8.2.4.22 BIG\_1024\_58\_dscopy()

```
void BIG_1024_58_dscopy (
    DBIG_1024_58 x,
    BIG_1024_58 y )
```

##### Parameters

<i>x</i>	DBIG number
<i>y</i>	BIG number to be copied

#### 8.2.4.23 BIG\_1024\_58\_dshl()

```
void BIG_1024_58_dshl (
    DBIG_1024_58 x,
    int s )
```

##### Parameters

<i>x</i>	DBIG number to be shifted
<i>s</i>	Number of bits to shift

#### 8.2.4.24 BIG\_1024\_58\_dshr()

```
void BIG_1024_58_dshr (
    DBIG_1024_58 x,
    int s )
```

##### Parameters

<i>x</i>	DBIG number to be shifted
<i>s</i>	Number of bits to shift

#### 8.2.4.25 BIG\_1024\_58\_dsub()

```
void BIG_1024_58_dsub (
    DBIG_1024_58 x,
    DBIG_1024_58 y,
    DBIG_1024_58 z )
```

**Parameters**

<i>x</i>	DBIG number, difference of other two - output not normalised
<i>y</i>	DBIG number
<i>z</i>	DBIG number

**8.2.4.26 BIG\_1024\_58\_dsucopy()**

```
void BIG_1024_58_dsucopy (
    DBIG_1024_58 x,
    BIG_1024_58 y )
```

**Parameters**

<i>x</i>	DBIG number
<i>y</i>	BIG number to be copied

**8.2.4.27 BIG\_1024\_58\_dzero()**

```
void BIG_1024_58_dzero (
    DBIG_1024_58 x )
```

**Parameters**

<i>x</i>	DBIG number to be set to zero
----------	-------------------------------

**8.2.4.28 BIG\_1024\_58\_fromBytes()**

```
void BIG_1024_58_fromBytes (
    BIG_1024_58 x,
    char * a )
```

**Parameters**

<i>x</i>	BIG number
<i>a</i>	byte array

#### 8.2.4.29 BIG\_1024\_58\_fromBytesLen()

```
void BIG_1024_58_fromBytesLen (
    BIG_1024_58 x,
    char * a,
    int s )
```

##### Parameters

<i>x</i>	BIG number
<i>a</i>	byte array
<i>s</i>	byte array length

#### 8.2.4.30 BIG\_1024\_58\_fshl()

```
int BIG_1024_58_fshl (
    BIG_1024_58 x,
    int s )
```

The number of bits to be shifted must be less than BASEBITS

##### Parameters

<i>x</i>	BIG number to be shifted
<i>s</i>	Number of bits to shift

##### Returns

Overflow bits

#### 8.2.4.31 BIG\_1024\_58\_fshr()

```
int BIG_1024_58_fshr (
    BIG_1024_58 x,
    int s )
```

The number of bits to be shifted must be less than BASEBITS

##### Parameters

<i>x</i>	BIG number to be shifted
<i>s</i>	Number of bits to shift

**Returns**

Shifted out bits

**8.2.4.32 BIG\_1024\_58\_imul()**

```
void BIG_1024_58_imul (
    BIG_1024_58 x,
    BIG_1024_58 y,
    int i )
```

**Parameters**

<i>x</i>	BIG number, product of other two
<i>y</i>	BIG number
<i>i</i>	small integer

**8.2.4.33 BIG\_1024\_58\_inc()**

```
void BIG_1024_58_inc (
    BIG_1024_58 x,
    int i )
```

**Parameters**

<i>x</i>	BIG number to be incremented
<i>i</i>	integer

**8.2.4.34 BIG\_1024\_58\_invmod2m()**

```
void BIG_1024_58_invmod2m (
    BIG_1024_58 x )
```

**Parameters**

<i>x</i>	BIG number to be inverted
----------	---------------------------

**8.2.4.35 BIG\_1024\_58\_invmodp()**

```
void BIG_1024_58_invmodp (
```

```

BIG_1024_58 x,
BIG_1024_58 y,
BIG_1024_58 n )

```

Modular Inversion - This is slow. Uses binary method.

#### Parameters

$x$	BIG number, on exit = $1/y \pmod n$
$y$	BIG number
$n$	The BIG Modulus

#### 8.2.4.36 BIG\_1024\_58\_isunity()

```

int BIG_1024_58_isunity (
    BIG_1024_58 x )

```

#### Parameters

$x$	a BIG number
-----	--------------

#### Returns

1 if one, else returns 0

#### 8.2.4.37 BIG\_1024\_58\_iszilch()

```

int BIG_1024_58_iszilch (
    BIG_1024_58 x )

```

#### Parameters

$x$	a BIG number
-----	--------------

#### Returns

1 if zero, else returns 0

#### 8.2.4.38 BIG\_1024\_58\_jacobi()

```

int BIG_1024_58_jacobi (
    BIG_1024_58 x,
    BIG_1024_58 y )

```

**Parameters**

<i>x</i>	BIG number
<i>y</i>	BIG number

**Returns**

Jacobi symbol, -1,0 or 1

**8.2.4.39 BIG\_1024\_58\_lastbits()**

```
int BIG_1024_58_lastbits (
    BIG_1024_58 x,
    int n )
```

**Parameters**

<i>x</i>	BIG number
<i>n</i>	number of bits to return. Assumed to be less than BASEBITS.

**Returns**

least significant n bits as an integer

**8.2.4.40 BIG\_1024\_58\_mod()**

```
void BIG_1024_58_mod (
    BIG_1024_58 x,
    BIG_1024_58 n )
```

Slow but rarely used

**Parameters**

<i>x</i>	BIG number to be reduced mod n
<i>n</i>	The modulus

**8.2.4.41 BIG\_1024\_58\_mod2m()**

```
void BIG_1024_58_mod2m (
    BIG_1024_58 x,
    int m )
```

## Truncation

## Parameters

$x$	BIG number, on reduced mod $2^m$
$m$	new truncated size

8.2.4.42 `BIG_1024_58_moddiv()`

```
void BIG_1024_58_moddiv (
    BIG_1024_58 x,
    BIG_1024_58 y,
    BIG_1024_58 z,
    BIG_1024_58 n )
```

Slow method for modular division

## Parameters

$x$	BIG number, on exit = $y/z \bmod n$
$y$	BIG number
$z$	BIG number
$n$	The BIG Modulus

8.2.4.43 `BIG_1024_58_modmul()`

```
void BIG_1024_58_modmul (
    BIG_1024_58 x,
    BIG_1024_58 y,
    BIG_1024_58 z,
    BIG_1024_58 n )
```

brief return NAF (Non-Adjacent-Form) value as +/- 1, 3 or 5, inputs must be normalised

Given  $x$  and  $3*x$  extracts NAF value from given bit position, and returns number of bits processed, and number of trailing zeros detected if any param  $x$  BIG number param  $x3$  BIG number, three times  $x$  param  $i$  bit position param  $nbs$  pointer to integer returning number of bits processed param  $nzs$  pointer to integer returning number of trailing 0s return + or - 1, 3 or 5 Slow method for modular multiplication

## Parameters

$x$	BIG number, on exit = $y*z \bmod n$
$y$	BIG number
$z$	BIG number
$n$	The BIG Modulus



## 8.2.4.44 BIG\_1024\_58\_modneg()

```
void BIG_1024_58_modneg (
    BIG_1024_58 x,
    BIG_1024_58 y,
    BIG_1024_58 n )
```

Modular negation

## Parameters

$x$	BIG number, on exit = $-y \bmod n$
$y$	BIG number
$n$	The BIG Modulus

## 8.2.4.45 BIG\_1024\_58\_modsqr()

```
void BIG_1024_58_modsqr (
    BIG_1024_58 x,
    BIG_1024_58 y,
    BIG_1024_58 n )
```

Slow method for modular squaring

## Parameters

$x$	BIG number, on exit = $y^2 \bmod n$
$y$	BIG number
$n$	The BIG Modulus

## 8.2.4.46 BIG\_1024\_58\_monty()

```
void BIG_1024_58_monty (
    BIG_1024_58 a,
    BIG_1024_58 md,
    chunk MC,
    DBIG_1024_58 d )
```

## Parameters

$a$	BIG number, reduction of a BIG
$md$	BIG number, the modulus
$MC$	the Montgomery Constant
$d$	DBIG number to be reduced

#### 8.2.4.47 BIG\_1024\_58\_mul()

```
void BIG_1024_58_mul (
    DBIG_1024_58 x,
    BIG_1024_58 y,
    BIG_1024_58 z )
```

##### Parameters

x	DBIG number, product of other two
y	BIG number
z	BIG number

#### 8.2.4.48 BIG\_1024\_58\_nbits()

```
int BIG_1024_58_nbits (
    BIG_1024_58 x )
```

##### Parameters

x	BIG number
---	------------

##### Returns

Number of bits in x

#### 8.2.4.49 BIG\_1024\_58\_norm()

```
chunk BIG_1024_58_norm (
    BIG_1024_58 x )
```

All digits of the input BIG are reduced mod  $2^{\text{BASEBITS}}$

##### Parameters

x	BIG number to be normalised
---	-----------------------------

#### 8.2.4.50 BIG\_1024\_58\_one()

```
void BIG_1024_58_one (
    BIG_1024_58 x )
```

##### Parameters

x	BIG number to be set to one.
---	------------------------------

#### 8.2.4.51 BIG\_1024\_58\_or()

```
void BIG_1024_58_or (
    BIG_1024_58 x,
    BIG_1024_58 y,
    BIG_1024_58 z )
```

##### Parameters

x	BIG number, or of other two
y	BIG number
z	BIG number

#### 8.2.4.52 BIG\_1024\_58\_output()

```
void BIG_1024_58_output (
    BIG_1024_58 x )
```

##### Parameters

x	a BIG number
---	--------------

#### 8.2.4.53 BIG\_1024\_58\_parity()

```
int BIG_1024_58_parity (
    BIG_1024_58 x )
```

##### Parameters

x	BIG number
---	------------

**Returns**

0 or 1

**8.2.4.54 BIG\_1024\_58\_pmul()**

```
chunk BIG_1024_58_pmul (
    BIG_1024_58 x,
    BIG_1024_58 y,
    int i )
```

**Parameters**

<i>x</i>	BIG number, product of other two
<i>y</i>	BIG number
<i>i</i>	small integer

**Returns**

Overflowing bits

**8.2.4.55 BIG\_1024\_58\_pxmuls()**

```
void BIG_1024_58_pxmuls (
    DBIG_1024_58 x,
    BIG_1024_58 y,
    int i )
```

**Parameters**

<i>x</i>	DBIG number, product of other two
<i>y</i>	BIG number
<i>i</i>	small integer

**8.2.4.56 BIG\_1024\_58\_random()**

```
void BIG_1024_58_random (
    BIG_1024_58 x,
    csprng * r )
```

Assumes that the random number generator has been suitably initialised

## Parameters

$x$	BIG number, on exit a random number
$r$	A pointer to a Cryptographically Secure Random Number Generator

## 8.2.4.57 BIG\_1024\_58\_randomnum()

```
void BIG_1024_58_randomnum (
    BIG_1024_58 x,
    BIG_1024_58 n,
    csprng * r )
```

Assumes that the random number generator has been suitably initialised

## Parameters

$x$	BIG number, on exit a random number
$n$	The modulus
$r$	A pointer to a Cryptographically Secure Random Number Generator

## 8.2.4.58 BIG\_1024\_58\_rawoutput()

```
void BIG_1024_58_rawoutput (
    BIG_1024_58 x )
```

## Parameters

$x$	a BIG number
-----	--------------

## 8.2.4.59 BIG\_1024\_58\_rcopy()

```
void BIG_1024_58_rcopy (
    BIG_1024_58 x,
    const BIG_1024_58 y )
```

## Parameters

$x$	BIG number
$y$	BIG number in ROM

#### 8.2.4.60 BIG\_1024\_58\_sdcopy()

```
void BIG_1024_58_sdcopy (
    BIG_1024_58 x,
    DBIG_1024_58 y )
```

##### Parameters

<i>x</i>	BIG number
<i>y</i>	DBIG number to be copied

#### 8.2.4.61 BIG\_1024\_58\_sdiv()

```
void BIG_1024_58_sdiv (
    BIG_1024_58 x,
    BIG_1024_58 n )
```

Slow but rarely used

##### Parameters

<i>x</i>	BIG number to be divided by n
<i>n</i>	The Divisor

#### 8.2.4.62 BIG\_1024\_58\_sducopy()

```
void BIG_1024_58_sducopy (
    BIG_1024_58 x,
    DBIG_1024_58 y )
```

##### Parameters

<i>x</i>	BIG number
<i>y</i>	DBIG number to be copied

#### 8.2.4.63 BIG\_1024\_58\_shl()

```
void BIG_1024_58_shl (
    BIG_1024_58 x,
    int s )
```

## Parameters

<i>x</i>	BIG number to be shifted
<i>s</i>	Number of bits to shift

## 8.2.4.64 BIG\_1024\_58\_shr()

```
void BIG_1024_58_shr (
    BIG_1024_58 x,
    int s )
```

## Parameters

<i>x</i>	BIG number to be shifted
<i>s</i>	Number of bits to shift

## 8.2.4.65 BIG\_1024\_58\_smul()

```
void BIG_1024_58_smul (
    BIG_1024_58 x,
    BIG_1024_58 y,
    BIG_1024_58 z )
```

Note that the product must fit into a BIG, and x must be distinct from y and z

## Parameters

<i>x</i>	BIG number, product of other two
<i>y</i>	BIG number
<i>z</i>	BIG number

## 8.2.4.66 BIG\_1024\_58\_split()

```
chunk BIG_1024_58_split (
    BIG_1024_58 x,
    BIG_1024_58 y,
    DBIG_1024_58 z,
    int s )
```

Internal function. The value of s must be approximately in the middle of the DBIG. Typically used to extract  $z \bmod 2^{\text{MODBITS}}$  and  $z/2^{\text{MODBITS}}$

**Parameters**

<i>x</i>	BIG number, top half of z
<i>y</i>	BIG number, bottom half of z
<i>z</i>	DBIG number to be split in two.
<i>s</i>	Bit position at which to split

**Returns**

carry-out from top half

**8.2.4.67 BIG\_1024\_58\_sqr()**

```
void BIG_1024_58_sqr (
    DBIG_1024_58 x,
    BIG_1024_58 y )
```

**Parameters**

<i>x</i>	DBIG number, square of a BIG
<i>y</i>	BIG number to be squared

**8.2.4.68 BIG\_1024\_58\_ssn()**

```
int BIG_1024_58_ssn (
    BIG_1024_58 r,
    BIG_1024_58 a,
    BIG_1024_58 m )
```

**Parameters**

<i>r</i>	BIG number normalised output
<i>a</i>	BIG number to be subtracted from
<i>m</i>	BIG number to be shifted and subtracted

**Returns**

sign of r

**8.2.4.69 BIG\_1024\_58\_sub()**

```
void BIG_1024_58_sub (
    BIG_1024_58 x,
```



```

BIG_1024_58 y,
BIG_1024_58 z )

```

**Parameters**

<i>x</i>	BIG number, difference of other two - output not normalised
<i>y</i>	BIG number
<i>z</i>	BIG number

**8.2.4.70 BIG\_1024\_58\_toBytes()**

```

void BIG_1024_58_toBytes (
    char * a,
    BIG_1024_58 x )

```

**Parameters**

<i>a</i>	byte array
<i>x</i>	BIG number

**8.2.4.71 BIG\_1024\_58\_zero()**

```

void BIG_1024_58_zero (
    BIG_1024_58 x )

```

**Parameters**

<i>x</i>	BIG number to be set to zero
----------	------------------------------

**8.3 big\_384\_58.h File Reference**

BIG Header File.

```

#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include "arch.h"
#include "amcl.h"
#include "config_big_384_58.h"

```

## Macros

- #define `BIGBITS_384_58` ( $8 * \text{MODBYTES\_384\_58}$ )
- #define `NLEN_384_58` ( $1 + ((8 * \text{MODBYTES\_384\_58} - 1) / \text{BASEBITS\_384\_58})$ )
- #define `DNLEN_384_58`  $2 * \text{NLEN\_384\_58}$
- #define `BMASK_384_58` ( $((\text{chunk})1 \ll \text{BASEBITS\_384\_58}) - 1$ )
- #define `NEXCESS_384_58` ( $1 \ll (\text{CHUNK} - \text{BASEBITS\_384\_58} - 1)$ )
- #define `HBITS_384_58` ( $\text{BASEBITS\_384\_58} / 2$ )
- #define `HMASK_384_58` ( $((\text{chunk})1 \ll \text{HBITS\_384\_58}) - 1$ )

## Typedefs

- typedef `chunk BIG_384_58[NLEN_384_58]`
- typedef `chunk DBIG_384_58[DNLEN_384_58]`

## Functions

- int `BIG_384_58_iszilch` (`BIG_384_58` x)  
*Tests for BIG equal to zero.*
- int `BIG_384_58_unity` (`BIG_384_58` x)  
*Tests for BIG equal to one.*
- int `BIG_384_58_diszilch` (`DBIG_384_58` x)  
*Tests for DBIG equal to zero.*
- void `BIG_384_58_output` (`BIG_384_58` x)  
*Outputs a BIG number to the console.*
- void `BIG_384_58_rawoutput` (`BIG_384_58` x)  
*Outputs a BIG number to the console in raw form (for debugging)*
- void `BIG_384_58_cswap` (`BIG_384_58` x, `BIG_384_58` y, int s)  
*Conditional constant time swap of two BIG numbers.*
- void `BIG_384_58_cmove` (`BIG_384_58` x, `BIG_384_58` y, int s)  
*Conditional copy of BIG number.*
- void `BIG_384_58_dcmove` (`BIG_384_58` x, `BIG_384_58` y, int s)  
*Conditional copy of DBIG number.*
- void `BIG_384_58_toBytes` (char \*a, `BIG_384_58` x)  
*Convert from BIG number to byte array.*
- void `BIG_384_58_fromBytes` (`BIG_384_58` x, char \*a)  
*Convert to BIG number from byte array.*
- void `BIG_384_58_fromBytesLen` (`BIG_384_58` x, char \*a, int s)  
*Convert to BIG number from byte array of given length.*
- void `BIG_384_58_dfromBytesLen` (`DBIG_384_58` x, char \*a, int s)  
*Convert to DBIG number from byte array of given length.*
- void `BIG_384_58_doutput` (`DBIG_384_58` x)  
*Outputs a DBIG number to the console.*
- void `BIG_384_58_drawoutput` (`DBIG_384_58` x)  
*Outputs a DBIG number to the console.*
- void `BIG_384_58_rcopy` (`BIG_384_58` x, const `BIG_384_58` y)  
*Copy BIG from Read-Only Memory to a BIG.*
- void `BIG_384_58_copy` (`BIG_384_58` x, `BIG_384_58` y)  
*Copy BIG to another BIG.*
- void `BIG_384_58_dcopy` (`DBIG_384_58` x, `DBIG_384_58` y)

- Copy DBIG to another DBIG.*

  - void [BIG\\_384\\_58\\_dsucopy](#) ([DBIG\\_384\\_58](#) x, [BIG\\_384\\_58](#) y)
- Copy BIG to upper half of DBIG.*

  - void [BIG\\_384\\_58\\_dscopy](#) ([DBIG\\_384\\_58](#) x, [BIG\\_384\\_58](#) y)
- Copy BIG to lower half of DBIG.*

  - void [BIG\\_384\\_58\\_sdcopy](#) ([BIG\\_384\\_58](#) x, [DBIG\\_384\\_58](#) y)
- Copy lower half of DBIG to a BIG.*

  - void [BIG\\_384\\_58\\_sducopy](#) ([BIG\\_384\\_58](#) x, [DBIG\\_384\\_58](#) y)
- Copy upper half of DBIG to a BIG.*

  - void [BIG\\_384\\_58\\_zero](#) ([BIG\\_384\\_58](#) x)
- Set BIG to zero.*

  - void [BIG\\_384\\_58\\_dzero](#) ([DBIG\\_384\\_58](#) x)
- Set DBIG to zero.*

  - void [BIG\\_384\\_58\\_one](#) ([BIG\\_384\\_58](#) x)
- Set BIG to one (unity)*

  - void [BIG\\_384\\_58\\_invmod2m](#) ([BIG\\_384\\_58](#) x)
- Set BIG to inverse mod  $2^{256}$ .*

  - void [BIG\\_384\\_58\\_add](#) ([BIG\\_384\\_58](#) x, [BIG\\_384\\_58](#) y, [BIG\\_384\\_58](#) z)
- Set BIG to sum of two BIGs - output not normalised.*

  - void [BIG\\_384\\_58\\_or](#) ([BIG\\_384\\_58](#) x, [BIG\\_384\\_58](#) y, [BIG\\_384\\_58](#) z)
- Set BIG to logical or of two BIGs - output normalised.*

  - void [BIG\\_384\\_58\\_inc](#) ([BIG\\_384\\_58](#) x, int i)
- Increment BIG by a small integer - output not normalised.*

  - void [BIG\\_384\\_58\\_sub](#) ([BIG\\_384\\_58](#) x, [BIG\\_384\\_58](#) y, [BIG\\_384\\_58](#) z)
- Set BIG to difference of two BIGs.*

  - void [BIG\\_384\\_58\\_dec](#) ([BIG\\_384\\_58](#) x, int i)
- Decrement BIG by a small integer - output not normalised.*

  - void [BIG\\_384\\_58\\_dadd](#) ([DBIG\\_384\\_58](#) x, [DBIG\\_384\\_58](#) y, [DBIG\\_384\\_58](#) z)
- Set DBIG to sum of two DBIGs.*

  - void [BIG\\_384\\_58\\_dsub](#) ([DBIG\\_384\\_58](#) x, [DBIG\\_384\\_58](#) y, [DBIG\\_384\\_58](#) z)
- Set DBIG to difference of two DBIGs.*

  - void [BIG\\_384\\_58\\_imul](#) ([BIG\\_384\\_58](#) x, [BIG\\_384\\_58](#) y, int i)
- Multiply BIG by a small integer - output not normalised.*

  - [chunk](#) [BIG\\_384\\_58\\_pmul](#) ([BIG\\_384\\_58](#) x, [BIG\\_384\\_58](#) y, int i)
- Multiply BIG by not-so-small small integer - output normalised.*

  - int [BIG\\_384\\_58\\_div3](#) ([BIG\\_384\\_58](#) x)
- Divide BIG by 3 - output normalised.*

  - void [BIG\\_384\\_58\\_pxm](#) ([DBIG\\_384\\_58](#) x, [BIG\\_384\\_58](#) y, int i)
- Multiply BIG by even bigger small integer resulting in a DBIG - output normalised.*

  - void [BIG\\_384\\_58\\_mul](#) ([DBIG\\_384\\_58](#) x, [BIG\\_384\\_58](#) y, [BIG\\_384\\_58](#) z)
- Multiply BIG by another BIG resulting in DBIG - inputs normalised and output normalised.*

  - void [BIG\\_384\\_58\\_smul](#) ([BIG\\_384\\_58](#) x, [BIG\\_384\\_58](#) y, [BIG\\_384\\_58](#) z)
- Multiply BIG by another BIG resulting in another BIG - inputs normalised and output normalised.*

  - void [BIG\\_384\\_58\\_sqr](#) ([DBIG\\_384\\_58](#) x, [BIG\\_384\\_58](#) y)
- Square BIG resulting in a DBIG - input normalised and output normalised.*

  - void [BIG\\_384\\_58\\_monty](#) ([BIG\\_384\\_58](#) a, [BIG\\_384\\_58](#) md, [chunk](#) MC, [DBIG\\_384\\_58](#) d)
- Montgomery reduction of a DBIG to a BIG - input normalised and output normalised.*

  - void [BIG\\_384\\_58\\_shl](#) ([BIG\\_384\\_58](#) x, int s)
- Shifts a BIG left by any number of bits - input must be normalised, output normalised.*

  - int [BIG\\_384\\_58\\_fshl](#) ([BIG\\_384\\_58](#) x, int s)
- Fast shifts a BIG left by a small number of bits - input must be normalised, output will be normalised.*

- void `BIG_384_58_dshl` (`DBIG_384_58` x, int s)  
*Shifts a DBIG left by any number of bits - input must be normalised, output normalised.*
- void `BIG_384_58_shr` (`BIG_384_58` x, int s)  
*Shifts a BIG right by any number of bits - input must be normalised, output normalised.*
- int `BIG_384_58_ssn` (`BIG_384_58` r, `BIG_384_58` a, `BIG_384_58` m)  
*Fast time-critical combined shift by 1 bit, subtract and normalise.*
- int `BIG_384_58_fshr` (`BIG_384_58` x, int s)  
*Fast shifts a BIG right by a small number of bits - input must be normalised, output will be normalised.*
- void `BIG_384_58_dshr` (`DBIG_384_58` x, int s)  
*Shifts a DBIG right by any number of bits - input must be normalised, output normalised.*
- `chunk` `BIG_384_58_split` (`BIG_384_58` x, `BIG_384_58` y, `DBIG_384_58` z, int s)  
*Splits a DBIG into two BIGs - input must be normalised, outputs normalised.*
- `chunk` `BIG_384_58_norm` (`BIG_384_58` x)  
*Normalizes a BIG number - output normalised.*
- void `BIG_384_58_dnorm` (`DBIG_384_58` x)  
*Normalizes a DBIG number - output normalised.*
- int `BIG_384_58_comp` (`BIG_384_58` x, `BIG_384_58` y)  
*Compares two BIG numbers. Inputs must be normalised externally.*
- int `BIG_384_58_dcomp` (`DBIG_384_58` x, `DBIG_384_58` y)  
*Compares two DBIG numbers. Inputs must be normalised externally.*
- int `BIG_384_58_nbits` (`BIG_384_58` x)  
*Calculate number of bits in a BIG - output normalised.*
- int `BIG_384_58_dnbits` (`DBIG_384_58` x)  
*Calculate number of bits in a DBIG - output normalised.*
- void `BIG_384_58_mod` (`BIG_384_58` x, `BIG_384_58` n)  
*Reduce x mod n - input and output normalised.*
- void `BIG_384_58_sdiv` (`BIG_384_58` x, `BIG_384_58` n)  
*Divide x by n - output normalised.*
- void `BIG_384_58_dmod` (`BIG_384_58` x, `DBIG_384_58` y, `BIG_384_58` n)  
*x=y mod n - output normalised*
- void `BIG_384_58_ddiv` (`BIG_384_58` x, `DBIG_384_58` y, `BIG_384_58` n)  
*x=y/n - output normalised*
- int `BIG_384_58_parity` (`BIG_384_58` x)  
*return parity of BIG, that is the least significant bit*
- int `BIG_384_58_bit` (`BIG_384_58` x, int i)  
*return i-th of BIG*
- int `BIG_384_58_lastbits` (`BIG_384_58` x, int n)  
*return least significant bits of a BIG*
- void `BIG_384_58_random` (`BIG_384_58` x, `csprng` \*r)  
*Create a random BIG from a random number generator.*
- void `BIG_384_58_randomnum` (`BIG_384_58` x, `BIG_384_58` n, `csprng` \*r)  
*Create an unbiased random BIG from a random number generator, reduced with respect to a modulus.*
- void `BIG_384_58_modmul` (`BIG_384_58` x, `BIG_384_58` y, `BIG_384_58` z, `BIG_384_58` n)  
*Calculate x=y\*z mod n.*
- void `BIG_384_58_moddiv` (`BIG_384_58` x, `BIG_384_58` y, `BIG_384_58` z, `BIG_384_58` n)  
*Calculate x=y/z mod n.*
- void `BIG_384_58_modsq` (`BIG_384_58` x, `BIG_384_58` y, `BIG_384_58` n)  
*Calculate x=y<sup>2</sup> mod n.*
- void `BIG_384_58_modneg` (`BIG_384_58` x, `BIG_384_58` y, `BIG_384_58` n)  
*Calculate x=-y mod n.*
- int `BIG_384_58_jacobi` (`BIG_384_58` x, `BIG_384_58` y)

*Calculate jacobi Symbol (x/y)*

- void [BIG\\_384\\_58\\_invmodp](#) ([BIG\\_384\\_58](#) x, [BIG\\_384\\_58](#) y, [BIG\\_384\\_58](#) n)

*Calculate  $x=1/y \bmod n$ .*

- void [BIG\\_384\\_58\\_mod2m](#) ([BIG\\_384\\_58](#) x, int m)

*Calculate  $x=x \bmod 2^m$ .*

- void [BIG\\_384\\_58\\_dmod2m](#) ([DBIG\\_384\\_58](#) x, int m)

*Calculate  $x=x \bmod 2^m$ .*

### 8.3.1 Detailed Description

#### Author

Mike Scott

### 8.3.2 Macro Definition Documentation

#### 8.3.2.1 BIGBITS\_384\_58

```
#define BIGBITS_384_58 (8*MODBYTES_384_58)
```

Length in bits

#### 8.3.2.2 BMASK\_384\_58

```
#define BMASK_384_58 (((chunk)1<<BASEBITS_384_58)-1)
```

Mask =  $2^{\text{BASEBITS}-1}$

#### 8.3.2.3 DNLEN\_384\_58

```
#define DNLEN_384_58 2*NLEN_384_58
```

Double length in bytes

#### 8.3.2.4 HBITS\_384\_58

```
#define HBITS_384_58 (BASEBITS_384_58/2)
```

Number of bits in number base divided by 2

#### 8.3.2.5 HMASK\_384\_58

```
#define HMASK_384_58 (((chunk)1<<HBITS_384_58)-1)
```

Mask =  $2^{\text{HBITS}}-1$

### 8.3.2.6 NEXCESS\_384\_58

```
#define NEXCESS_384_58 (1<<(CHUNK-BASEBITS_384_58-1))
```

$2^{(CHUNK-BASEBITS-1)}$  - digit cannot be multiplied by more than this before normalisation

### 8.3.2.7 NLEN\_384\_58

```
#define NLEN_384_58 (1+((8*MODBYTES_384_58-1)/BASEBITS_384_58))
```

length in bytes

## 8.3.3 Typedef Documentation

### 8.3.3.1 BIG\_384\_58

```
typedef chunk BIG_384_58[NLEN_384_58]
```

Define type BIG as array of chunks

### 8.3.3.2 DBIG\_384\_58

```
typedef chunk DBIG_384_58[DNLEN_384_58]
```

Define type DBIG as array of chunks

## 8.3.4 Function Documentation

### 8.3.4.1 BIG\_384\_58\_add()

```
void BIG_384_58_add (
    BIG_384_58 x,
    BIG_384_58 y,
    BIG_384_58 z )
```

#### Parameters

<i>x</i>	BIG number, sum of other two
<i>y</i>	BIG number
<i>z</i>	BIG number

#### 8.3.4.2 BIG\_384\_58\_bit()

```
int BIG_384_58_bit (
    BIG_384_58 x,
    int i )
```

##### Parameters

<i>x</i>	BIG number
<i>i</i>	the bit of x to be returned

##### Returns

0 or 1

#### 8.3.4.3 BIG\_384\_58\_cmove()

```
void BIG_384_58_cmove (
    BIG_384_58 x,
    BIG_384_58 y,
    int s )
```

Conditionally copies second parameter to the first (without branching)

##### Parameters

<i>x</i>	a BIG number
<i>y</i>	another BIG number
<i>s</i>	copy takes place if not equal to 0

#### 8.3.4.4 BIG\_384\_58\_comp()

```
int BIG_384_58_comp (
    BIG_384_58 x,
    BIG_384_58 y )
```

##### Parameters

<i>x</i>	first BIG number to be compared
<i>y</i>	second BIG number to be compared

**Returns**

-1 if  $x < y$ , 0 if  $x = y$ , 1 if  $x > y$

**8.3.4.5 BIG\_384\_58\_copy()**

```
void BIG_384_58_copy (
    BIG_384_58 x,
    BIG_384_58 y )
```

**Parameters**

<i>x</i>	BIG number
<i>y</i>	BIG number to be copied

**8.3.4.6 BIG\_384\_58\_cswap()**

```
void BIG_384_58_cswap (
    BIG_384_58 x,
    BIG_384_58 y,
    int s )
```

Conditionally swaps parameters in constant time (without branching)

**Parameters**

<i>x</i>	a BIG number
<i>y</i>	another BIG number
<i>s</i>	swap takes place if not equal to 0

**8.3.4.7 BIG\_384\_58\_dadd()**

```
void BIG_384_58_dadd (
    DBIG_384_58 x,
    DBIG_384_58 y,
    DBIG_384_58 z )
```

**Parameters**

<i>x</i>	DBIG number, sum of other two - output not normalised
<i>y</i>	DBIG number
<i>z</i>	DBIG number



#### 8.3.4.8 BIG\_384\_58\_dcmove()

```
void BIG_384_58_dcmove (
    BIG_384_58 x,
    BIG_384_58 y,
    int s )
```

Conditionally copies second parameter to the first (without branching)

##### Parameters

<i>x</i>	a DBIG number
<i>y</i>	another DBIG number
<i>s</i>	copy takes place if not equal to 0

#### 8.3.4.9 BIG\_384\_58\_dcomp()

```
int BIG_384_58_dcomp (
    DBIG_384_58 x,
    DBIG_384_58 y )
```

##### Parameters

<i>x</i>	first DBIG number to be compared
<i>y</i>	second DBIG number to be compared

##### Returns

-1 is  $x < y$ , 0 if  $x = y$ , 1 if  $x > y$

#### 8.3.4.10 BIG\_384\_58\_dcopy()

```
void BIG_384_58_dcopy (
    DBIG_384_58 x,
    DBIG_384_58 y )
```

##### Parameters

<i>x</i>	DBIG number
<i>y</i>	DBIG number to be copied

### 8.3.4.11 BIG\_384\_58\_ddiv()

```
void BIG_384_58_ddiv (
    BIG_384_58 x,
    DBIG_384_58 y,
    BIG_384_58 n )
```

Slow but rarely used. *y* is destroyed.

#### Parameters

<i>x</i>	BIG number, on exit = <i>y/n</i>
<i>y</i>	DBIG number
<i>n</i>	Modulus

### 8.3.4.12 BIG\_384\_58\_dec()

```
void BIG_384_58_dec (
    BIG_384_58 x,
    int i )
```

#### Parameters

<i>x</i>	BIG number to be decremented
<i>i</i>	integer

### 8.3.4.13 BIG\_384\_58\_dfromBytesLen()

```
void BIG_384_58_dfromBytesLen (
    DBIG_384_58 x,
    char * a,
    int s )
```

#### Parameters

<i>x</i>	DBIG number
<i>a</i>	byte array
<i>s</i>	byte array length

#### 8.3.4.14 BIG\_384\_58\_diszilch()

```
int BIG_384_58_diszilch (
    DBIG_384_58 x )
```

##### Parameters

<i>x</i>	a DBIG number
----------	---------------

##### Returns

1 if zero, else returns 0

#### 8.3.4.15 BIG\_384\_58\_div3()

```
int BIG_384_58_div3 (
    BIG_384_58 x )
```

##### Parameters

<i>x</i>	BIG number
----------	------------

##### Returns

Remainder

#### 8.3.4.16 BIG\_384\_58\_dmod()

```
void BIG_384_58_dmod (
    BIG_384_58 x,
    DBIG_384_58 y,
    BIG_384_58 n )
```

Slow but rarely used. *y* is destroyed.

##### Parameters

<i>x</i>	BIG number, on exit = $y \bmod n$
<i>y</i>	DBIG number
<i>n</i>	Modulus

**8.3.4.17 BIG\_384\_58\_dmod2m()**

```
void BIG_384_58_dmod2m (
    DBIG_384_58 x,
    int m )
```

Truncation

Parameters

<i>x</i>	DBIG number, on reduced mod $2^m$
<i>m</i>	new truncated size

**8.3.4.18 BIG\_384\_58\_dnbits()**

```
int BIG_384_58_dnbits (
    DBIG_384_58 x )
```

Parameters

<i>x</i>	DBIG number
----------	-------------

Returns

Number of bits in *x*

**8.3.4.19 BIG\_384\_58\_dnorm()**

```
void BIG_384_58_dnorm (
    DBIG_384_58 x )
```

All digits of the input DBIG are reduced mod  $2^{\text{BASEBITS}}$

Parameters

<i>x</i>	DBIG number to be normalised
----------	------------------------------

**8.3.4.20 BIG\_384\_58\_doutput()**

```
void BIG_384_58_doutput (
    DBIG_384_58 x )
```

**Parameters**

<i>x</i>	a DBIG number
----------	---------------

**8.3.4.21 BIG\_384\_58\_drawoutput()**

```
void BIG_384_58_drawoutput (
    DBIG_384_58 x )
```

**Parameters**

<i>x</i>	a DBIG number
----------	---------------

**8.3.4.22 BIG\_384\_58\_dscopy()**

```
void BIG_384_58_dscopy (
    DBIG_384_58 x,
    BIG_384_58 y )
```

**Parameters**

<i>x</i>	DBIG number
<i>y</i>	BIG number to be copied

**8.3.4.23 BIG\_384\_58\_dshl()**

```
void BIG_384_58_dshl (
    DBIG_384_58 x,
    int s )
```

**Parameters**

<i>x</i>	DBIG number to be shifted
<i>s</i>	Number of bits to shift

**8.3.4.24 BIG\_384\_58\_dshr()**

```
void BIG_384_58_dshr (
```

```

    DBIG_384_58 x,
    int s )

```

**Parameters**

<i>x</i>	DBIG number to be shifted
<i>s</i>	Number of bits to shift

**8.3.4.25 BIG\_384\_58\_dsub()**

```

void BIG_384_58_dsub (
    DBIG_384_58 x,
    DBIG_384_58 y,
    DBIG_384_58 z )

```

**Parameters**

<i>x</i>	DBIG number, difference of other two - output not normalised
<i>y</i>	DBIG number
<i>z</i>	DBIG number

**8.3.4.26 BIG\_384\_58\_dsucopy()**

```

void BIG_384_58_dsucopy (
    DBIG_384_58 x,
    BIG_384_58 y )

```

**Parameters**

<i>x</i>	DBIG number
<i>y</i>	BIG number to be copied

**8.3.4.27 BIG\_384\_58\_dzero()**

```

void BIG_384_58_dzero (
    DBIG_384_58 x )

```

**Parameters**

<i>x</i>	DBIG number to be set to zero
----------	-------------------------------

#### 8.3.4.28 BIG\_384\_58\_fromBytes()

```
void BIG_384_58_fromBytes (
    BIG_384_58 x,
    char * a )
```

##### Parameters

<i>x</i>	BIG number
<i>a</i>	byte array

#### 8.3.4.29 BIG\_384\_58\_fromBytesLen()

```
void BIG_384_58_fromBytesLen (
    BIG_384_58 x,
    char * a,
    int s )
```

##### Parameters

<i>x</i>	BIG number
<i>a</i>	byte array
<i>s</i>	byte array length

#### 8.3.4.30 BIG\_384\_58\_fshl()

```
int BIG_384_58_fshl (
    BIG_384_58 x,
    int s )
```

The number of bits to be shifted must be less than BASEBITS

##### Parameters

<i>x</i>	BIG number to be shifted
<i>s</i>	Number of bits to shift

##### Returns

Overflow bits

**8.3.4.31 BIG\_384\_58\_fshr()**

```
int BIG_384_58_fshr (
    BIG_384_58 x,
    int s )
```

The number of bits to be shifted must be less than BASEBITS

**Parameters**

<i>x</i>	BIG number to be shifted
<i>s</i>	Number of bits to shift

**Returns**

Shifted out bits

**8.3.4.32 BIG\_384\_58\_imul()**

```
void BIG_384_58_imul (
    BIG_384_58 x,
    BIG_384_58 y,
    int i )
```

**Parameters**

<i>x</i>	BIG number, product of other two
<i>y</i>	BIG number
<i>i</i>	small integer

**8.3.4.33 BIG\_384\_58\_inc()**

```
void BIG_384_58_inc (
    BIG_384_58 x,
    int i )
```

**Parameters**

<i>x</i>	BIG number to be incremented
<i>i</i>	integer



#### 8.3.4.34 BIG\_384\_58\_invmod2m()

```
void BIG_384_58_invmod2m (  
    BIG_384_58 x )
```

##### Parameters

$x$	BIG number to be inverted
-----	---------------------------

#### 8.3.4.35 BIG\_384\_58\_invmodp()

```
void BIG_384_58_invmodp (  
    BIG_384_58 x,  
    BIG_384_58 y,  
    BIG_384_58 n )
```

Modular Inversion - This is slow. Uses binary method.

##### Parameters

$x$	BIG number, on exit = $1/y \pmod n$
$y$	BIG number
$n$	The BIG Modulus

#### 8.3.4.36 BIG\_384\_58\_isunity()

```
int BIG_384_58_isunity (  
    BIG_384_58 x )
```

##### Parameters

$x$	a BIG number
-----	--------------

##### Returns

1 if one, else returns 0

#### 8.3.4.37 BIG\_384\_58\_iszilch()

```
int BIG_384_58_iszilch (  
    BIG_384_58 x )
```

**Parameters**

<i>x</i>	a BIG number
----------	--------------

**Returns**

1 if zero, else returns 0

**8.3.4.38 BIG\_384\_58\_jacobi()**

```
int BIG_384_58_jacobi (
    BIG_384_58 x,
    BIG_384_58 y )
```

**Parameters**

<i>x</i>	BIG number
<i>y</i>	BIG number

**Returns**

Jacobi symbol, -1,0 or 1

**8.3.4.39 BIG\_384\_58\_lastbits()**

```
int BIG_384_58_lastbits (
    BIG_384_58 x,
    int n )
```

**Parameters**

<i>x</i>	BIG number
<i>n</i>	number of bits to return. Assumed to be less than BASEBITS.

**Returns**

least significant n bits as an integer

**8.3.4.40 BIG\_384\_58\_mod()**

```
void BIG_384_58_mod (
    BIG_384_58 x,
    BIG_384_58 n )
```

Slow but rarely used

#### Parameters

$x$	BIG number to be reduced mod $n$
$n$	The modulus

#### 8.3.4.41 BIG\_384\_58\_mod2m()

```
void BIG_384_58_mod2m (
    BIG_384_58 x,
    int m )
```

Truncation

#### Parameters

$x$	BIG number, on reduced mod $2^m$
$m$	new truncated size

#### 8.3.4.42 BIG\_384\_58\_moddiv()

```
void BIG_384_58_moddiv (
    BIG_384_58 x,
    BIG_384_58 y,
    BIG_384_58 z,
    BIG_384_58 n )
```

Slow method for modular division

#### Parameters

$x$	BIG number, on exit = $y/z \bmod n$
$y$	BIG number
$z$	BIG number
$n$	The BIG Modulus

#### 8.3.4.43 BIG\_384\_58\_modmul()

```
void BIG_384_58_modmul (
    BIG_384_58 x,
    BIG_384_58 y,
```

```

BIG_384_58 z,
BIG_384_58 n )

```

brief return NAF (Non-Adjacent-Form) value as +/- 1, 3 or 5, inputs must be normalised

Given  $x$  and  $3*x$  extracts NAF value from given bit position, and returns number of bits processed, and number of trailing zeros detected if any param  $x$  BIG number param  $x3$  BIG number, three times  $x$  param  $i$  bit position param  $nbs$  pointer to integer returning number of bits processed param  $nzs$  pointer to integer returning number of trailing 0s return + or - 1, 3 or 5 Slow method for modular multiplication

#### Parameters

$x$	BIG number, on exit = $y*z \bmod n$
$y$	BIG number
$z$	BIG number
$n$	The BIG Modulus

#### 8.3.4.44 BIG\_384\_58\_modneg()

```

void BIG_384_58_modneg (
    BIG_384_58 x,
    BIG_384_58 y,
    BIG_384_58 n )

```

Modular negation

#### Parameters

$x$	BIG number, on exit = $-y \bmod n$
$y$	BIG number
$n$	The BIG Modulus

#### 8.3.4.45 BIG\_384\_58\_modsqr()

```

void BIG_384_58_modsqr (
    BIG_384_58 x,
    BIG_384_58 y,
    BIG_384_58 n )

```

Slow method for modular squaring

#### Parameters

$x$	BIG number, on exit = $y^2 \bmod n$
$y$	BIG number
$n$	The BIG Modulus

#### 8.3.4.46 BIG\_384\_58\_monty()

```
void BIG_384_58_monty (
    BIG_384_58 a,
    BIG_384_58 md,
    chunk MC,
    DBIG_384_58 d )
```

##### Parameters

<i>a</i>	BIG number, reduction of a BIG
<i>md</i>	BIG number, the modulus
<i>MC</i>	the Montgomery Constant
<i>d</i>	DBIG number to be reduced

#### 8.3.4.47 BIG\_384\_58\_mul()

```
void BIG_384_58_mul (
    DBIG_384_58 x,
    BIG_384_58 y,
    BIG_384_58 z )
```

##### Parameters

<i>x</i>	DBIG number, product of other two
<i>y</i>	BIG number
<i>z</i>	BIG number

#### 8.3.4.48 BIG\_384\_58\_nbits()

```
int BIG_384_58_nbits (
    BIG_384_58 x )
```

##### Parameters

<i>x</i>	BIG number
----------	------------

##### Returns

Number of bits in x

**8.3.4.49 BIG\_384\_58\_norm()**

```
chunk BIG_384_58_norm (
    BIG_384_58 x )
```

All digits of the input BIG are reduced mod  $2^{\text{BASEBITS}}$

**Parameters**

<i>x</i>	BIG number to be normalised
----------	-----------------------------

**8.3.4.50 BIG\_384\_58\_one()**

```
void BIG_384_58_one (
    BIG_384_58 x )
```

**Parameters**

<i>x</i>	BIG number to be set to one.
----------	------------------------------

**8.3.4.51 BIG\_384\_58\_or()**

```
void BIG_384_58_or (
    BIG_384_58 x,
    BIG_384_58 y,
    BIG_384_58 z )
```

**Parameters**

<i>x</i>	BIG number, or of other two
<i>y</i>	BIG number
<i>z</i>	BIG number

**8.3.4.52 BIG\_384\_58\_output()**

```
void BIG_384_58_output (
    BIG_384_58 x )
```

**Parameters**

<i>x</i>	a BIG number
----------	--------------

#### 8.3.4.53 BIG\_384\_58\_parity()

```
int BIG_384_58_parity (  
    BIG_384_58 x )
```

##### Parameters

<i>x</i>	BIG number
----------	------------

##### Returns

0 or 1

#### 8.3.4.54 BIG\_384\_58\_pmul()

```
chunk BIG_384_58_pmul (  
    BIG_384_58 x,  
    BIG_384_58 y,  
    int i )
```

##### Parameters

<i>x</i>	BIG number, product of other two
<i>y</i>	BIG number
<i>i</i>	small integer

##### Returns

Overflowing bits

#### 8.3.4.55 BIG\_384\_58\_pxmuls()

```
void BIG_384_58_pxmuls (  
    DBIG_384_58 x,  
    BIG_384_58 y,  
    int i )
```

##### Parameters

<i>x</i>	DBIG number, product of other two
<i>y</i>	BIG number
<i>i</i>	small integer

**8.3.4.56 BIG\_384\_58\_random()**

```
void BIG_384_58_random (
    BIG_384_58 x,
    csprng * r )
```

Assumes that the random number generator has been suitably initialised

**Parameters**

<i>x</i>	BIG number, on exit a random number
<i>r</i>	A pointer to a Cryptographically Secure Random Number Generator

**8.3.4.57 BIG\_384\_58\_randomnum()**

```
void BIG_384_58_randomnum (
    BIG_384_58 x,
    BIG_384_58 n,
    csprng * r )
```

Assumes that the random number generator has been suitably initialised

**Parameters**

<i>x</i>	BIG number, on exit a random number
<i>n</i>	The modulus
<i>r</i>	A pointer to a Cryptographically Secure Random Number Generator

**8.3.4.58 BIG\_384\_58\_rawoutput()**

```
void BIG_384_58_rawoutput (
    BIG_384_58 x )
```

**Parameters**

<i>x</i>	a BIG number
----------	--------------

**8.3.4.59 BIG\_384\_58\_rcopy()**

```
void BIG_384_58_rcopy (
```



```
BIG_384_58 x,
const BIG_384_58 y )
```

**Parameters**

<i>x</i>	BIG number
<i>y</i>	BIG number in ROM

**8.3.4.60 BIG\_384\_58\_sdcopy()**

```
void BIG_384_58_sdcopy (
    BIG_384_58 x,
    DBIG_384_58 y )
```

**Parameters**

<i>x</i>	BIG number
<i>y</i>	DBIG number to be copied

**8.3.4.61 BIG\_384\_58\_sdiv()**

```
void BIG_384_58_sdiv (
    BIG_384_58 x,
    BIG_384_58 n )
```

Slow but rarely used

**Parameters**

<i>x</i>	BIG number to be divided by n
<i>n</i>	The Divisor

**8.3.4.62 BIG\_384\_58\_sducopy()**

```
void BIG_384_58_sducopy (
    BIG_384_58 x,
    DBIG_384_58 y )
```

**Parameters**

<i>x</i>	BIG number
<i>y</i>	DBIG number to be copied

**8.3.4.63 BIG\_384\_58\_shl()**

```
void BIG_384_58_shl (
    BIG_384_58 x,
    int s )
```

**Parameters**

<i>x</i>	BIG number to be shifted
<i>s</i>	Number of bits to shift

**8.3.4.64 BIG\_384\_58\_shr()**

```
void BIG_384_58_shr (
    BIG_384_58 x,
    int s )
```

**Parameters**

<i>x</i>	BIG number to be shifted
<i>s</i>	Number of bits to shift

**8.3.4.65 BIG\_384\_58\_smul()**

```
void BIG_384_58_smul (
    BIG_384_58 x,
    BIG_384_58 y,
    BIG_384_58 z )
```

Note that the product must fit into a BIG, and x must be distinct from y and z

**Parameters**

<i>x</i>	BIG number, product of other two
<i>y</i>	BIG number
<i>z</i>	BIG number

**8.3.4.66 BIG\_384\_58\_split()**

```
chunk BIG_384_58_split (
```

```

BIG_384_58 x,
BIG_384_58 y,
DBIG_384_58 z,
int s )

```

Internal function. The value of  $s$  must be approximately in the middle of the DBIG. Typically used to extract  $z \bmod 2^{\text{MODBITS}}$  and  $z/2^{\text{MODBITS}}$

#### Parameters

$x$	BIG number, top half of $z$
$y$	BIG number, bottom half of $z$
$z$	DBIG number to be split in two.
$s$	Bit position at which to split

#### Returns

carry-out from top half

#### 8.3.4.67 BIG\_384\_58\_sqr()

```

void BIG_384_58_sqr (
    DBIG_384_58 x,
    BIG_384_58 y )

```

#### Parameters

$x$	DBIG number, square of a BIG
$y$	BIG number to be squared

#### 8.3.4.68 BIG\_384\_58\_ssn()

```

int BIG_384_58_ssn (
    BIG_384_58 r,
    BIG_384_58 a,
    BIG_384_58 m )

```

#### Parameters

$r$	BIG number normalised output
$a$	BIG number to be subtracted from
$m$	BIG number to be shifted and subtracted

**Returns**

sign of r

**8.3.4.69 BIG\_384\_58\_sub()**

```
void BIG_384_58_sub (
    BIG_384_58 x,
    BIG_384_58 y,
    BIG_384_58 z )
```

**Parameters**

<i>x</i>	BIG number, difference of other two - output not normalised
<i>y</i>	BIG number
<i>z</i>	BIG number

**8.3.4.70 BIG\_384\_58\_toBytes()**

```
void BIG_384_58_toBytes (
    char * a,
    BIG_384_58 x )
```

**Parameters**

<i>a</i>	byte array
<i>x</i>	BIG number

**8.3.4.71 BIG\_384\_58\_zero()**

```
void BIG_384_58_zero (
    BIG_384_58 x )
```

**Parameters**

<i>x</i>	BIG number to be set to zero
----------	------------------------------

**8.4 big\_512\_60.h File Reference**

BIG Header File.

```
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include "arch.h"
#include "amcl.h"
#include "config_big_512_60.h"
```

## Macros

- #define [BIGBITS\\_512\\_60](#) (8\*MODBYTES\_512\_60)
- #define [NLEN\\_512\\_60](#) (1+((8\*MODBYTES\_512\_60-1)/BASEBITS\_512\_60))
- #define [DNLEN\\_512\\_60](#) 2\*NLEN\_512\_60
- #define [BMASK\\_512\\_60](#) (((chunk)1<<BASEBITS\_512\_60)-1)
- #define [NEXCESS\\_512\\_60](#) (1<<(CHUNK-BASEBITS\_512\_60-1))
- #define [HBITS\\_512\\_60](#) (BASEBITS\_512\_60/2)
- #define [HMASK\\_512\\_60](#) (((chunk)1<<HBITS\_512\_60)-1)

## Typedefs

- typedef [chunk](#) [BIG\\_512\\_60](#)[[NLEN\\_512\\_60](#)]
- typedef [chunk](#) [DBIG\\_512\\_60](#)[[DNLEN\\_512\\_60](#)]

## Functions

- int [BIG\\_512\\_60\\_iszilch](#) ([BIG\\_512\\_60](#) x)  
*Tests for BIG equal to zero.*
- int [BIG\\_512\\_60\\_isunity](#) ([BIG\\_512\\_60](#) x)  
*Tests for BIG equal to one.*
- int [BIG\\_512\\_60\\_diszilch](#) ([DBIG\\_512\\_60](#) x)  
*Tests for DBIG equal to zero.*
- void [BIG\\_512\\_60\\_output](#) ([BIG\\_512\\_60](#) x)  
*Outputs a BIG number to the console.*
- void [BIG\\_512\\_60\\_rawoutput](#) ([BIG\\_512\\_60](#) x)  
*Outputs a BIG number to the console in raw form (for debugging)*
- void [BIG\\_512\\_60\\_cswap](#) ([BIG\\_512\\_60](#) x, [BIG\\_512\\_60](#) y, int s)  
*Conditional constant time swap of two BIG numbers.*
- void [BIG\\_512\\_60\\_cmove](#) ([BIG\\_512\\_60](#) x, [BIG\\_512\\_60](#) y, int s)  
*Conditional copy of BIG number.*
- void [BIG\\_512\\_60\\_dcmove](#) ([BIG\\_512\\_60](#) x, [BIG\\_512\\_60](#) y, int s)  
*Conditional copy of DBIG number.*
- void [BIG\\_512\\_60\\_toBytes](#) (char \*a, [BIG\\_512\\_60](#) x)  
*Convert from BIG number to byte array.*
- void [BIG\\_512\\_60\\_fromBytes](#) ([BIG\\_512\\_60](#) x, char \*a)  
*Convert to BIG number from byte array.*
- void [BIG\\_512\\_60\\_fromBytesLen](#) ([BIG\\_512\\_60](#) x, char \*a, int s)  
*Convert to BIG number from byte array of given length.*
- void [BIG\\_512\\_60\\_dfromBytesLen](#) ([DBIG\\_512\\_60](#) x, char \*a, int s)  
*Convert to DBIG number from byte array of given length.*
- void [BIG\\_512\\_60\\_doutput](#) ([DBIG\\_512\\_60](#) x)

- Outputs a DBIG number to the console.*

  - void `BIG_512_60_drawoutput` (`DBIG_512_60 x`)
- Outputs a DBIG number to the console.*

  - void `BIG_512_60_rcopy` (`BIG_512_60 x`, `const BIG_512_60 y`)
- Copy BIG from Read-Only Memory to a BIG.*

  - void `BIG_512_60_copy` (`BIG_512_60 x`, `BIG_512_60 y`)
- Copy BIG to another BIG.*

  - void `BIG_512_60_dcopy` (`DBIG_512_60 x`, `DBIG_512_60 y`)
- Copy DBIG to another DBIG.*

  - void `BIG_512_60_dsucopy` (`DBIG_512_60 x`, `BIG_512_60 y`)
- Copy BIG to upper half of DBIG.*

  - void `BIG_512_60_dscopy` (`DBIG_512_60 x`, `BIG_512_60 y`)
- Copy BIG to lower half of DBIG.*

  - void `BIG_512_60_sdcopy` (`BIG_512_60 x`, `DBIG_512_60 y`)
- Copy lower half of DBIG to a BIG.*

  - void `BIG_512_60_sducopy` (`BIG_512_60 x`, `DBIG_512_60 y`)
- Copy upper half of DBIG to a BIG.*

  - void `BIG_512_60_zero` (`BIG_512_60 x`)
- Set BIG to zero.*

  - void `BIG_512_60_dzero` (`DBIG_512_60 x`)
- Set DBIG to zero.*

  - void `BIG_512_60_one` (`BIG_512_60 x`)
- Set BIG to one (unity)*

  - void `BIG_512_60_invmod2m` (`BIG_512_60 x`)
- Set BIG to inverse mod  $2^{256}$ .*

  - void `BIG_512_60_add` (`BIG_512_60 x`, `BIG_512_60 y`, `BIG_512_60 z`)
- Set BIG to sum of two BIGs - output not normalised.*

  - void `BIG_512_60_or` (`BIG_512_60 x`, `BIG_512_60 y`, `BIG_512_60 z`)
- Set BIG to logical or of two BIGs - output normalised.*

  - void `BIG_512_60_inc` (`BIG_512_60 x`, `int i`)
- Increment BIG by a small integer - output not normalised.*

  - void `BIG_512_60_sub` (`BIG_512_60 x`, `BIG_512_60 y`, `BIG_512_60 z`)
- Set BIG to difference of two BIGs.*

  - void `BIG_512_60_dec` (`BIG_512_60 x`, `int i`)
- Decrement BIG by a small integer - output not normalised.*

  - void `BIG_512_60_dadd` (`DBIG_512_60 x`, `DBIG_512_60 y`, `DBIG_512_60 z`)
- Set DBIG to sum of two DBIGs.*

  - void `BIG_512_60_dsub` (`DBIG_512_60 x`, `DBIG_512_60 y`, `DBIG_512_60 z`)
- Set DBIG to difference of two DBIGs.*

  - void `BIG_512_60_imul` (`BIG_512_60 x`, `BIG_512_60 y`, `int i`)
- Multiply BIG by a small integer - output not normalised.*

  - `chunk` `BIG_512_60_pmul` (`BIG_512_60 x`, `BIG_512_60 y`, `int i`)
- Multiply BIG by not-so-small small integer - output normalised.*

  - `int` `BIG_512_60_div3` (`BIG_512_60 x`)
- Divide BIG by 3 - output normalised.*

  - void `BIG_512_60_pxmud` (`DBIG_512_60 x`, `BIG_512_60 y`, `int i`)
- Multiply BIG by even bigger small integer resulting in a DBIG - output normalised.*

  - void `BIG_512_60_mul` (`DBIG_512_60 x`, `BIG_512_60 y`, `BIG_512_60 z`)
- Multiply BIG by another BIG resulting in DBIG - inputs normalised and output normalised.*

  - void `BIG_512_60_smul` (`BIG_512_60 x`, `BIG_512_60 y`, `BIG_512_60 z`)
- Multiply BIG by another BIG resulting in another BIG - inputs normalised and output normalised.*

- void `BIG_512_60_sqr` (`DBIG_512_60` x, `BIG_512_60` y)  
*Square BIG resulting in a DBIG - input normalised and output normalised.*
- void `BIG_512_60_monty` (`BIG_512_60` a, `BIG_512_60` md, `chunk` MC, `DBIG_512_60` d)  
*Montgomery reduction of a DBIG to a BIG - input normalised and output normalised.*
- void `BIG_512_60_shl` (`BIG_512_60` x, int s)  
*Shifts a BIG left by any number of bits - input must be normalised, output normalised.*
- int `BIG_512_60_fshl` (`BIG_512_60` x, int s)  
*Fast shifts a BIG left by a small number of bits - input must be normalised, output will be normalised.*
- void `BIG_512_60_dshl` (`DBIG_512_60` x, int s)  
*Shifts a DBIG left by any number of bits - input must be normalised, output normalised.*
- void `BIG_512_60_shr` (`BIG_512_60` x, int s)  
*Shifts a BIG right by any number of bits - input must be normalised, output normalised.*
- int `BIG_512_60_ssn` (`BIG_512_60` r, `BIG_512_60` a, `BIG_512_60` m)  
*Fast time-critical combined shift by 1 bit, subtract and normalise.*
- int `BIG_512_60_fshr` (`BIG_512_60` x, int s)  
*Fast shifts a BIG right by a small number of bits - input must be normalised, output will be normalised.*
- void `BIG_512_60_dshr` (`DBIG_512_60` x, int s)  
*Shifts a DBIG right by any number of bits - input must be normalised, output normalised.*
- `chunk` `BIG_512_60_split` (`BIG_512_60` x, `BIG_512_60` y, `DBIG_512_60` z, int s)  
*Splits a DBIG into two BIGs - input must be normalised, outputs normalised.*
- `chunk` `BIG_512_60_norm` (`BIG_512_60` x)  
*Normalizes a BIG number - output normalised.*
- void `BIG_512_60_dnorm` (`DBIG_512_60` x)  
*Normalizes a DBIG number - output normalised.*
- int `BIG_512_60_comp` (`BIG_512_60` x, `BIG_512_60` y)  
*Compares two BIG numbers. Inputs must be normalised externally.*
- int `BIG_512_60_dcomp` (`DBIG_512_60` x, `DBIG_512_60` y)  
*Compares two DBIG numbers. Inputs must be normalised externally.*
- int `BIG_512_60_nbits` (`BIG_512_60` x)  
*Calculate number of bits in a BIG - output normalised.*
- int `BIG_512_60_dnbits` (`DBIG_512_60` x)  
*Calculate number of bits in a DBIG - output normalised.*
- void `BIG_512_60_mod` (`BIG_512_60` x, `BIG_512_60` n)  
*Reduce x mod n - input and output normalised.*
- void `BIG_512_60_sdiv` (`BIG_512_60` x, `BIG_512_60` n)  
*Divide x by n - output normalised.*
- void `BIG_512_60_dmod` (`BIG_512_60` x, `DBIG_512_60` y, `BIG_512_60` n)  
 *$x=y \bmod n$  - output normalised*
- void `BIG_512_60_ddiv` (`BIG_512_60` x, `DBIG_512_60` y, `BIG_512_60` n)  
 *$x=y/n$  - output normalised*
- int `BIG_512_60_parity` (`BIG_512_60` x)  
*return parity of BIG, that is the least significant bit*
- int `BIG_512_60_bit` (`BIG_512_60` x, int i)  
*return i-th of BIG*
- int `BIG_512_60_lastbits` (`BIG_512_60` x, int n)  
*return least significant bits of a BIG*
- void `BIG_512_60_random` (`BIG_512_60` x, `csprng` \*r)  
*Create a random BIG from a random number generator.*
- void `BIG_512_60_randomnum` (`BIG_512_60` x, `BIG_512_60` n, `csprng` \*r)  
*Create an unbiased random BIG from a random number generator, reduced with respect to a modulus.*
- void `BIG_512_60_modmul` (`BIG_512_60` x, `BIG_512_60` y, `BIG_512_60` z, `BIG_512_60` n)

- Calculate  $x=y*z \bmod n$ .*

  - void [BIG\\_512\\_60\\_moddiv](#) ([BIG\\_512\\_60](#) x, [BIG\\_512\\_60](#) y, [BIG\\_512\\_60](#) z, [BIG\\_512\\_60](#) n)
- Calculate  $x=y/z \bmod n$ .*

  - void [BIG\\_512\\_60\\_modsqr](#) ([BIG\\_512\\_60](#) x, [BIG\\_512\\_60](#) y, [BIG\\_512\\_60](#) n)
- Calculate  $x=y^2 \bmod n$ .*

  - void [BIG\\_512\\_60\\_modneg](#) ([BIG\\_512\\_60](#) x, [BIG\\_512\\_60](#) y, [BIG\\_512\\_60](#) n)
- Calculate  $x=-y \bmod n$ .*

  - int [BIG\\_512\\_60\\_jacobi](#) ([BIG\\_512\\_60](#) x, [BIG\\_512\\_60](#) y)
- Calculate jacobi Symbol ( $x/y$ )*

  - void [BIG\\_512\\_60\\_invmodp](#) ([BIG\\_512\\_60](#) x, [BIG\\_512\\_60](#) y, [BIG\\_512\\_60](#) n)
- Calculate  $x=1/y \bmod n$ .*

  - void [BIG\\_512\\_60\\_mod2m](#) ([BIG\\_512\\_60](#) x, int m)
- Calculate  $x=x \bmod 2^m$ .*

  - void [BIG\\_512\\_60\\_dmod2m](#) ([DBIG\\_512\\_60](#) x, int m)
- Calculate  $x=x \bmod 2^m$ .*

### 8.4.1 Detailed Description

#### Author

Mike Scott

### 8.4.2 Macro Definition Documentation

#### 8.4.2.1 BIGBITS\_512\_60

```
#define BIGBITS_512_60 (8*MODBYTES_512_60)
```

Length in bits

#### 8.4.2.2 BMASK\_512\_60

```
#define BMASK_512_60 (((chunk)1<<BASEBITS_512_60)-1)
```

Mask =  $2^{\text{BASEBITS}}-1$

#### 8.4.2.3 DNLEN\_512\_60

```
#define DNLEN_512_60 2*NLEN_512_60
```

Double length in bytes



#### 8.4.2.4 HBITS\_512\_60

```
#define HBITS_512_60 (BASEBITS_512_60/2)
```

Number of bits in number base divided by 2

#### 8.4.2.5 HMASK\_512\_60

```
#define HMASK_512_60 (((chunk)1<<HBITS_512_60)-1)
```

Mask =  $2^{\text{HBITS}-1}$

#### 8.4.2.6 NEXCESS\_512\_60

```
#define NEXCESS_512_60 (1<<(CHUNK-BASEBITS_512_60-1))
```

$2^{(\text{CHUNK}-\text{BASEBITS}-1)}$  - digit cannot be multiplied by more than this before normalisation

#### 8.4.2.7 NLEN\_512\_60

```
#define NLEN_512_60 (1+((8*MODBYTES_512_60-1)/BASEBITS_512_60))
```

length in bytes

### 8.4.3 Typedef Documentation

#### 8.4.3.1 BIG\_512\_60

```
typedef chunk BIG_512_60[NLEN_512_60]
```

Define type BIG as array of chunks

#### 8.4.3.2 DBIG\_512\_60

```
typedef chunk DBIG_512_60[DNLEN_512_60]
```

Define type DBIG as array of chunks

### 8.4.4 Function Documentation

#### 8.4.4.1 BIG\_512\_60\_add()

```
void BIG_512_60_add (  
    BIG_512_60 x,  
    BIG_512_60 y,  
    BIG_512_60 z )
```

**Parameters**

<i>x</i>	BIG number, sum of other two
<i>y</i>	BIG number
<i>z</i>	BIG number

**8.4.4.2 BIG\_512\_60\_bit()**

```
int BIG_512_60_bit (
    BIG_512_60 x,
    int i )
```

**Parameters**

<i>x</i>	BIG number
<i>i</i>	the bit of x to be returned

**Returns**

0 or 1

**8.4.4.3 BIG\_512\_60\_cmove()**

```
void BIG_512_60_cmove (
    BIG_512_60 x,
    BIG_512_60 y,
    int s )
```

Conditionally copies second parameter to the first (without branching)

**Parameters**

<i>x</i>	a BIG number
<i>y</i>	another BIG number
<i>s</i>	copy takes place if not equal to 0

**8.4.4.4 BIG\_512\_60\_comp()**

```
int BIG_512_60_comp (
    BIG_512_60 x,
    BIG_512_60 y )
```

**Parameters**

<i>x</i>	first BIG number to be compared
<i>y</i>	second BIG number to be compared

**Returns**

-1 if  $x < y$ , 0 if  $x = y$ , 1 if  $x > y$

**8.4.4.5 BIG\_512\_60\_copy()**

```
void BIG_512_60_copy (
    BIG_512_60 x,
    BIG_512_60 y )
```

**Parameters**

<i>x</i>	BIG number
<i>y</i>	BIG number to be copied

**8.4.4.6 BIG\_512\_60\_cswap()**

```
void BIG_512_60_cswap (
    BIG_512_60 x,
    BIG_512_60 y,
    int s )
```

Conditionally swaps parameters in constant time (without branching)

**Parameters**

<i>x</i>	a BIG number
<i>y</i>	another BIG number
<i>s</i>	swap takes place if not equal to 0

**8.4.4.7 BIG\_512\_60\_dadd()**

```
void BIG_512_60_dadd (
    DBIG_512_60 x,
    DBIG_512_60 y,
    DBIG_512_60 z )
```

**Parameters**

<i>x</i>	DBIG number, sum of other two - output not normalised
<i>y</i>	DBIG number
<i>z</i>	DBIG number

**8.4.4.8 BIG\_512\_60\_dcmove()**

```
void BIG_512_60_dcmove (
    BIG_512_60 x,
    BIG_512_60 y,
    int s )
```

Conditionally copies second parameter to the first (without branching)

**Parameters**

<i>x</i>	a DBIG number
<i>y</i>	another DBIG number
<i>s</i>	copy takes place if not equal to 0

**8.4.4.9 BIG\_512\_60\_dcomp()**

```
int BIG_512_60_dcomp (
    DBIG_512_60 x,
    DBIG_512_60 y )
```

**Parameters**

<i>x</i>	first DBIG number to be compared
<i>y</i>	second DBIG number to be compared

**Returns**

-1 is  $x < y$ , 0 if  $x = y$ , 1 if  $x > y$

**8.4.4.10 BIG\_512\_60\_dcopy()**

```
void BIG_512_60_dcopy (
    DBIG_512_60 x,
    DBIG_512_60 y )
```

## Parameters

<i>x</i>	DBIG number
<i>y</i>	DBIG number to be copied

## 8.4.4.11 BIG\_512\_60\_ddiv()

```
void BIG_512_60_ddiv (
    BIG_512_60 x,
    DBIG_512_60 y,
    BIG_512_60 n )
```

Slow but rarely used. *y* is destroyed.

## Parameters

<i>x</i>	BIG number, on exit = <i>y</i> / <i>n</i>
<i>y</i>	DBIG number
<i>n</i>	Modulus

## 8.4.4.12 BIG\_512\_60\_dec()

```
void BIG_512_60_dec (
    BIG_512_60 x,
    int i )
```

## Parameters

<i>x</i>	BIG number to be decremented
<i>i</i>	integer

## 8.4.4.13 BIG\_512\_60\_dfromBytesLen()

```
void BIG_512_60_dfromBytesLen (
    DBIG_512_60 x,
    char * a,
    int s )
```

## Parameters

<i>x</i>	DBIG number
<i>a</i>	byte array
<i>s</i>	byte array length

#### 8.4.4.14 BIG\_512\_60\_diszilch()

```
int BIG_512_60_diszilch (  
    DBIG_512_60 x )
```

##### Parameters

<i>x</i>	a DBIG number
----------	---------------

##### Returns

1 if zero, else returns 0

#### 8.4.4.15 BIG\_512\_60\_div3()

```
int BIG_512_60_div3 (  
    BIG_512_60 x )
```

##### Parameters

<i>x</i>	BIG number
----------	------------

##### Returns

Remainder

#### 8.4.4.16 BIG\_512\_60\_dmod()

```
void BIG_512_60_dmod (  
    BIG_512_60 x,  
    DBIG_512_60 y,  
    BIG_512_60 n )
```

Slow but rarely used. *y* is destroyed.

##### Parameters

<i>x</i>	BIG number, on exit = $y \bmod n$
<i>y</i>	DBIG number
<i>n</i>	Modulus

#### 8.4.4.17 BIG\_512\_60\_dmod2m()

```
void BIG_512_60_dmod2m (
    DBIG_512_60 x,
    int m )
```

Truncation

Parameters

<i>x</i>	DBIG number, on reduced mod $2^m$
<i>m</i>	new truncated size

#### 8.4.4.18 BIG\_512\_60\_dnbits()

```
int BIG_512_60_dnbits (
    DBIG_512_60 x )
```

Parameters

<i>x</i>	DBIG number
----------	-------------

Returns

Number of bits in x

#### 8.4.4.19 BIG\_512\_60\_dnorm()

```
void BIG_512_60_dnorm (
    DBIG_512_60 x )
```

All digits of the input DBIG are reduced mod  $2^{\text{BASEBITS}}$

Parameters

<i>x</i>	DBIG number to be normalised
----------	------------------------------

#### 8.4.4.20 BIG\_512\_60\_doutput()

```
void BIG_512_60_doutput (
```

```
DBIG_512_60 x )
```

**Parameters**

<i>x</i>	a DBIG number
----------	---------------

**8.4.4.21 BIG\_512\_60\_drawoutput()**

```
void BIG_512_60_drawoutput (  
    DBIG_512_60 x )
```

**Parameters**

<i>x</i>	a DBIG number
----------	---------------

**8.4.4.22 BIG\_512\_60\_dscopy()**

```
void BIG_512_60_dscopy (  
    DBIG_512_60 x,  
    BIG_512_60 y )
```

**Parameters**

<i>x</i>	DBIG number
<i>y</i>	BIG number to be copied

**8.4.4.23 BIG\_512\_60\_dshl()**

```
void BIG_512_60_dshl (  
    DBIG_512_60 x,  
    int s )
```

**Parameters**

<i>x</i>	DBIG number to be shifted
<i>s</i>	Number of bits to shift



#### 8.4.4.24 BIG\_512\_60\_dshr()

```
void BIG_512_60_dshr (
    DBIG_512_60 x,
    int s )
```

##### Parameters

<i>x</i>	DBIG number to be shifted
<i>s</i>	Number of bits to shift

#### 8.4.4.25 BIG\_512\_60\_dsub()

```
void BIG_512_60_dsub (
    DBIG_512_60 x,
    DBIG_512_60 y,
    DBIG_512_60 z )
```

##### Parameters

<i>x</i>	DBIG number, difference of other two - output not normalised
<i>y</i>	DBIG number
<i>z</i>	DBIG number

#### 8.4.4.26 BIG\_512\_60\_dsucopy()

```
void BIG_512_60_dsucopy (
    DBIG_512_60 x,
    BIG_512_60 y )
```

##### Parameters

<i>x</i>	DBIG number
<i>y</i>	BIG number to be copied

#### 8.4.4.27 BIG\_512\_60\_dzero()

```
void BIG_512_60_dzero (
    DBIG_512_60 x )
```

## Parameters

<i>x</i>	DBIG number to be set to zero
----------	-------------------------------

## 8.4.4.28 BIG\_512\_60\_fromBytes()

```
void BIG_512_60_fromBytes (
    BIG_512_60 x,
    char * a )
```

## Parameters

<i>x</i>	BIG number
<i>a</i>	byte array

## 8.4.4.29 BIG\_512\_60\_fromBytesLen()

```
void BIG_512_60_fromBytesLen (
    BIG_512_60 x,
    char * a,
    int s )
```

## Parameters

<i>x</i>	BIG number
<i>a</i>	byte array
<i>s</i>	byte array length

## 8.4.4.30 BIG\_512\_60\_fshl()

```
int BIG_512_60_fshl (
    BIG_512_60 x,
    int s )
```

The number of bits to be shifted must be less than BASEBITS

## Parameters

<i>x</i>	BIG number to be shifted
<i>s</i>	Number of bits to shift

**Returns**

Overflow bits

**8.4.4.31 BIG\_512\_60\_fshr()**

```
int BIG_512_60_fshr (  
    BIG_512_60 x,  
    int s )
```

The number of bits to be shifted must be less than BASEBITS

**Parameters**

<i>x</i>	BIG number to be shifted
<i>s</i>	Number of bits to shift

**Returns**

Shifted out bits

**8.4.4.32 BIG\_512\_60\_imul()**

```
void BIG_512_60_imul (  
    BIG_512_60 x,  
    BIG_512_60 y,  
    int i )
```

**Parameters**

<i>x</i>	BIG number, product of other two
<i>y</i>	BIG number
<i>i</i>	small integer

**8.4.4.33 BIG\_512\_60\_inc()**

```
void BIG_512_60_inc (  
    BIG_512_60 x,  
    int i )
```

**Parameters**

<i>x</i>	BIG number to be incremented
<i>i</i>	integer

#### 8.4.4.34 BIG\_512\_60\_invmod2m()

```
void BIG_512_60_invmod2m (  
    BIG_512_60 x )
```

##### Parameters

$x$	BIG number to be inverted
-----	---------------------------

#### 8.4.4.35 BIG\_512\_60\_invmodp()

```
void BIG_512_60_invmodp (  
    BIG_512_60 x,  
    BIG_512_60 y,  
    BIG_512_60 n )
```

Modular Inversion - This is slow. Uses binary method.

##### Parameters

$x$	BIG number, on exit = $1/y \pmod n$
$y$	BIG number
$n$	The BIG Modulus

#### 8.4.4.36 BIG\_512\_60\_isunity()

```
int BIG_512_60_isunity (  
    BIG_512_60 x )
```

##### Parameters

$x$	a BIG number
-----	--------------

##### Returns

1 if one, else returns 0

#### 8.4.4.37 BIG\_512\_60\_iszilch()

```
int BIG_512_60_iszilch (  
    BIG_512_60 x )
```

**Parameters**

<i>x</i>	a BIG number
----------	--------------

**Returns**

1 if zero, else returns 0

**8.4.4.38 BIG\_512\_60\_jacobi()**

```
int BIG_512_60_jacobi (
    BIG_512_60 x,
    BIG_512_60 y )
```

**Parameters**

<i>x</i>	BIG number
<i>y</i>	BIG number

**Returns**

Jacobi symbol, -1,0 or 1

**8.4.4.39 BIG\_512\_60\_lastbits()**

```
int BIG_512_60_lastbits (
    BIG_512_60 x,
    int n )
```

**Parameters**

<i>x</i>	BIG number
<i>n</i>	number of bits to return. Assumed to be less than BASEBITS.

**Returns**

least significant n bits as an integer

**8.4.4.40 BIG\_512\_60\_mod()**

```
void BIG_512_60_mod (
    BIG_512_60 x,
    BIG_512_60 n )
```

Slow but rarely used

#### Parameters

$x$	BIG number to be reduced mod $n$
$n$	The modulus

#### 8.4.4.41 BIG\_512\_60\_mod2m()

```
void BIG_512_60_mod2m (
    BIG_512_60 x,
    int m )
```

Truncation

#### Parameters

$x$	BIG number, on reduced mod $2^m$
$m$	new truncated size

#### 8.4.4.42 BIG\_512\_60\_moddiv()

```
void BIG_512_60_moddiv (
    BIG_512_60 x,
    BIG_512_60 y,
    BIG_512_60 z,
    BIG_512_60 n )
```

Slow method for modular division

#### Parameters

$x$	BIG number, on exit = $y/z \bmod n$
$y$	BIG number
$z$	BIG number
$n$	The BIG Modulus

#### 8.4.4.43 BIG\_512\_60\_modmul()

```
void BIG_512_60_modmul (
    BIG_512_60 x,
    BIG_512_60 y,
```

```

BIG_512_60 z,
BIG_512_60 n )

```

brief return NAF (Non-Adjacent-Form) value as +/- 1, 3 or 5, inputs must be normalised

Given  $x$  and  $3*x$  extracts NAF value from given bit position, and returns number of bits processed, and number of trailing zeros detected if any param  $x$  BIG number param  $x3$  BIG number, three times  $x$  param  $i$  bit position param  $nbs$  pointer to integer returning number of bits processed param  $nzs$  pointer to integer returning number of trailing 0s return + or - 1, 3 or 5 Slow method for modular multiplication

#### Parameters

$x$	BIG number, on exit = $y*z \bmod n$
$y$	BIG number
$z$	BIG number
$n$	The BIG Modulus

#### 8.4.4.44 BIG\_512\_60\_modneg()

```

void BIG_512_60_modneg (
    BIG_512_60 x,
    BIG_512_60 y,
    BIG_512_60 n )

```

Modular negation

#### Parameters

$x$	BIG number, on exit = $-y \bmod n$
$y$	BIG number
$n$	The BIG Modulus

#### 8.4.4.45 BIG\_512\_60\_modsqr()

```

void BIG_512_60_modsqr (
    BIG_512_60 x,
    BIG_512_60 y,
    BIG_512_60 n )

```

Slow method for modular squaring

#### Parameters

$x$	BIG number, on exit = $y^2 \bmod n$
$y$	BIG number
$n$	The BIG Modulus



#### 8.4.4.46 BIG\_512\_60\_monty()

```
void BIG_512_60_monty (
    BIG_512_60 a,
    BIG_512_60 md,
    chunk MC,
    DBIG_512_60 d )
```

##### Parameters

<i>a</i>	BIG number, reduction of a BIG
<i>md</i>	BIG number, the modulus
<i>MC</i>	the Montgomery Constant
<i>d</i>	DBIG number to be reduced

#### 8.4.4.47 BIG\_512\_60\_mul()

```
void BIG_512_60_mul (
    DBIG_512_60 x,
    BIG_512_60 y,
    BIG_512_60 z )
```

##### Parameters

<i>x</i>	DBIG number, product of other two
<i>y</i>	BIG number
<i>z</i>	BIG number

#### 8.4.4.48 BIG\_512\_60\_nbits()

```
int BIG_512_60_nbits (
    BIG_512_60 x )
```

##### Parameters

<i>x</i>	BIG number
----------	------------

##### Returns

Number of bits in x

**8.4.4.49 BIG\_512\_60\_norm()**

```
chunk BIG_512_60_norm (
    BIG_512_60 x )
```

All digits of the input BIG are reduced mod  $2^{\text{BASEBITS}}$

**Parameters**

<i>x</i>	BIG number to be normalised
----------	-----------------------------

**8.4.4.50 BIG\_512\_60\_one()**

```
void BIG_512_60_one (
    BIG_512_60 x )
```

**Parameters**

<i>x</i>	BIG number to be set to one.
----------	------------------------------

**8.4.4.51 BIG\_512\_60\_or()**

```
void BIG_512_60_or (
    BIG_512_60 x,
    BIG_512_60 y,
    BIG_512_60 z )
```

**Parameters**

<i>x</i>	BIG number, or of other two
<i>y</i>	BIG number
<i>z</i>	BIG number

**8.4.4.52 BIG\_512\_60\_output()**

```
void BIG_512_60_output (
    BIG_512_60 x )
```

**Parameters**

<i>x</i>	a BIG number
----------	--------------

#### 8.4.4.53 BIG\_512\_60\_parity()

```
int BIG_512_60_parity (  
    BIG_512_60 x )
```

##### Parameters

<i>x</i>	BIG number
----------	------------

##### Returns

0 or 1

#### 8.4.4.54 BIG\_512\_60\_pmul()

```
chunk BIG_512_60_pmul (  
    BIG_512_60 x,  
    BIG_512_60 y,  
    int i )
```

##### Parameters

<i>x</i>	BIG number, product of other two
<i>y</i>	BIG number
<i>i</i>	small integer

##### Returns

Overflowing bits

#### 8.4.4.55 BIG\_512\_60\_pxmuls()

```
void BIG_512_60_pxmuls (  
    DBIG_512_60 x,  
    BIG_512_60 y,  
    int i )
```

##### Parameters

<i>x</i>	DBIG number, product of other two
<i>y</i>	BIG number
<i>i</i>	small integer

**8.4.4.56 BIG\_512\_60\_random()**

```
void BIG_512_60_random (
    BIG_512_60 x,
    csprng * r )
```

Assumes that the random number generator has been suitably initialised

**Parameters**

<i>x</i>	BIG number, on exit a random number
<i>r</i>	A pointer to a Cryptographically Secure Random Number Generator

**8.4.4.57 BIG\_512\_60\_randomnum()**

```
void BIG_512_60_randomnum (
    BIG_512_60 x,
    BIG_512_60 n,
    csprng * r )
```

Assumes that the random number generator has been suitably initialised

**Parameters**

<i>x</i>	BIG number, on exit a random number
<i>n</i>	The modulus
<i>r</i>	A pointer to a Cryptographically Secure Random Number Generator

**8.4.4.58 BIG\_512\_60\_rawoutput()**

```
void BIG_512_60_rawoutput (
    BIG_512_60 x )
```

**Parameters**

<i>x</i>	a BIG number
----------	--------------

**8.4.4.59 BIG\_512\_60\_rcopy()**

```
void BIG_512_60_rcopy (
```

```
BIG_512_60 x,  
const BIG_512_60 y )
```

**Parameters**

<i>x</i>	BIG number
<i>y</i>	BIG number in ROM

**8.4.4.60 BIG\_512\_60\_sdcopy()**

```
void BIG_512_60_sdcopy (  
    BIG_512_60 x,  
    DBIG_512_60 y )
```

**Parameters**

<i>x</i>	BIG number
<i>y</i>	DBIG number to be copied

**8.4.4.61 BIG\_512\_60\_sdiv()**

```
void BIG_512_60_sdiv (  
    BIG_512_60 x,  
    BIG_512_60 n )
```

Slow but rarely used

**Parameters**

<i>x</i>	BIG number to be divided by n
<i>n</i>	The Divisor

**8.4.4.62 BIG\_512\_60\_sducopy()**

```
void BIG_512_60_sducopy (  
    BIG_512_60 x,  
    DBIG_512_60 y )
```

**Parameters**

<i>x</i>	BIG number
<i>y</i>	DBIG number to be copied

**8.4.4.63 BIG\_512\_60\_shl()**

```
void BIG_512_60_shl (
    BIG_512_60 x,
    int s )
```

**Parameters**

<i>x</i>	BIG number to be shifted
<i>s</i>	Number of bits to shift

**8.4.4.64 BIG\_512\_60\_shr()**

```
void BIG_512_60_shr (
    BIG_512_60 x,
    int s )
```

**Parameters**

<i>x</i>	BIG number to be shifted
<i>s</i>	Number of bits to shift

**8.4.4.65 BIG\_512\_60\_smul()**

```
void BIG_512_60_smul (
    BIG_512_60 x,
    BIG_512_60 y,
    BIG_512_60 z )
```

Note that the product must fit into a BIG, and x must be distinct from y and z

**Parameters**

<i>x</i>	BIG number, product of other two
<i>y</i>	BIG number
<i>z</i>	BIG number

**8.4.4.66 BIG\_512\_60\_split()**

```
chunk BIG_512_60_split (
```

```

BIG_512_60 x,
BIG_512_60 y,
DBIG_512_60 z,
int s )

```

Internal function. The value of  $s$  must be approximately in the middle of the DBIG. Typically used to extract  $z \bmod 2^{\text{MODBITS}}$  and  $z/2^{\text{MODBITS}}$

#### Parameters

$x$	BIG number, top half of $z$
$y$	BIG number, bottom half of $z$
$z$	DBIG number to be split in two.
$s$	Bit position at which to split

#### Returns

carry-out from top half

#### 8.4.4.67 BIG\_512\_60\_sqr()

```

void BIG_512_60_sqr (
    DBIG_512_60 x,
    BIG_512_60 y )

```

#### Parameters

$x$	DBIG number, square of a BIG
$y$	BIG number to be squared

#### 8.4.4.68 BIG\_512\_60\_ssn()

```

int BIG_512_60_ssn (
    BIG_512_60 r,
    BIG_512_60 a,
    BIG_512_60 m )

```

#### Parameters

$r$	BIG number normalised output
$a$	BIG number to be subtracted from
$m$	BIG number to be shifted and subtracted

**Returns**

sign of r

**8.4.4.69 BIG\_512\_60\_sub()**

```
void BIG_512_60_sub (
    BIG_512_60 x,
    BIG_512_60 y,
    BIG_512_60 z )
```

**Parameters**

<i>x</i>	BIG number, difference of other two - output not normalised
<i>y</i>	BIG number
<i>z</i>	BIG number

**8.4.4.70 BIG\_512\_60\_toBytes()**

```
void BIG_512_60_toBytes (
    char * a,
    BIG_512_60 x )
```

**Parameters**

<i>a</i>	byte array
<i>x</i>	BIG number

**8.4.4.71 BIG\_512\_60\_zero()**

```
void BIG_512_60_zero (
    BIG_512_60 x )
```

**Parameters**

<i>x</i>	BIG number to be set to zero
----------	------------------------------

**8.5 bls\_BLS381.h File Reference**

BLS Header file.



```
#include "pair_BLS381.h"
```

## Macros

- #define [BGS\\_BLS381\\_MODBYTES\\_384\\_58](#)
- #define [BFS\\_BLS381\\_MODBYTES\\_384\\_58](#)
- #define [BLS\\_OK](#) 0
- #define [BLS\\_FAIL](#) 41
- #define [BLS\\_INVALID\\_G1](#) 42
- #define [BLS\\_INVALID\\_G2](#) 43

## Functions

- int [BLS\\_BLS381\\_KEY\\_PAIR\\_GENERATE](#) (csprng \*RNG, octet \*S, octet \*W)  
*Generate Key Pair.*
- int [BLS\\_BLS381\\_SIGN](#) (octet \*SIG, octet \*M, octet \*S)  
*Calculate a signature.*
- int [BLS\\_BLS381\\_VERIFY](#) (octet \*SIG, octet \*M, octet \*W)  
*Verify a signature.*
- int [BLS\\_BLS381\\_ADD\\_G1](#) (octet \*R1, octet \*R2, octet \*R)  
*Add two members from the group G1.*
- int [BLS\\_BLS381\\_ADD\\_G2](#) (octet \*W1, octet \*W2, octet \*W)  
*Add two members from the group G2.*
- int [BLS\\_BLS381\\_MAKE\\_SHARES](#) (int k, int n, csprng \*RNG, octet \*X, octet \*Y, octet \*SKI, octet \*SKO)  
*Use Shamir's secret sharing to distribute BLS secret keys.*
- int [BLS\\_BLS381\\_RECOVER\\_SECRET](#) (int k, octet \*X, octet \*Y, octet \*SK)  
*Use Shamir's secret sharing to recover a BLS secret key.*
- int [BLS\\_BLS381\\_RECOVER\\_SIGNATURE](#) (int k, octet \*X, octet \*Y, octet \*SIG)  
*Use Shamir's secret sharing to recover a BLS signature.*

### 8.5.1 Detailed Description

#### Author

Mike Scott

#### Date

28th Novemebr 2018 Allows some user configuration defines structures declares functions

### 8.5.2 Macro Definition Documentation

#### 8.5.2.1 BFS\_BLS381

```
#define BFS_BLS381_MODBYTES_384_58
```

BLS Field Size

### 8.5.2.2 BGS\_BLS381

```
#define BGS_BLS381 MODBYTES_384_58
```

BLS Group Size

### 8.5.2.3 BLS\_FAIL

```
#define BLS_FAIL 41
```

Invalid signature

### 8.5.2.4 BLS\_INVALID\_G1

```
#define BLS_INVALID_G1 42
```

Not a valid G1 point on the curve

### 8.5.2.5 BLS\_INVALID\_G2

```
#define BLS_INVALID_G2 43
```

Not a valid G2 point on the curve

### 8.5.2.6 BLS\_OK

```
#define BLS_OK 0
```

Function completed without error

## 8.5.3 Function Documentation

### 8.5.3.1 BLS\_BLS381\_ADD\_G1()

```
int BLS_BLS381_ADD_G1 (
    octet * R1,
    octet * R2,
    octet * R )
```

#### Parameters

<i>R1</i>	member of G1
<i>R2</i>	member of G1
<i>R</i>	member of G1. $R = R1+R2$

**Returns**

Zero for success or else an error code

**8.5.3.2 BLS\_BLS381\_ADD\_G2()**

```
int BLS_BLS381_ADD_G2 (
    octet * W1,
    octet * W2,
    octet * W )
```

**Parameters**

<i>W1</i>	member of G2
<i>W2</i>	member of G2
<i>W</i>	member of G2. $W = W1+W2$

**Returns**

Zero for success or else an error code

**8.5.3.3 BLS\_BLS381\_KEY\_PAIR\_GENERATE()**

```
int BLS_BLS381_KEY_PAIR_GENERATE (
    csprng * RNG,
    octet * S,
    octet * W )
```

**Parameters**

<i>RNG</i>	Pointer to a cryptographically secure random number generator
<i>S</i>	Private key. Generated externally if RNG set to NULL
<i>W</i>	Public Key. $W = S \cdot G$ , where G is fixed generator

**Returns**

Zero for success or else an error code

**8.5.3.4 BLS\_BLS381\_MAKE\_SHARES()**

```
int BLS_BLS381_MAKE_SHARES (
    int k,
```

```

    int n,
    csprng * RNG,
    octet * X,
    octet * Y,
    octet * SKI,
    octet * SKO )

```

#### Parameters

<i>k</i>	Threshold
<i>n</i>	Number of shares
<i>RNG</i>	Pointer to a cryptographically secure random number generator
<i>X</i>	X values
<i>Y</i>	Y values. Valid BLS secret keys
<i>SKI</i>	Input secret key to be shared. Ignored if set to NULL
<i>SKO</i>	Secret key that is shared

#### Returns

Zero for success or else an error code

#### 8.5.3.5 BLS\_BLS381\_RECOVER\_SECRET()

```

int BLS_BLS381_RECOVER_SECRET (
    int k,
    octet * X,
    octet * Y,
    octet * SK )

```

#### Parameters

<i>k</i>	Threshold
<i>X</i>	X values
<i>Y</i>	Y values. Valid BLS secret keys
<i>SK</i>	Secret key that is recovered

#### Returns

Zero for success or else an error code

#### 8.5.3.6 BLS\_BLS381\_RECOVER\_SIGNATURE()

```

int BLS_BLS381_RECOVER_SIGNATURE (
    int k,
    octet * X,
    octet * Y,
    octet * SIG )

```

## Parameters

<i>k</i>	Threshold
<i>X</i>	X values
<i>Y</i>	Y values. Valid BLS signatures
<i>SIG</i>	Signature that is recovered

## Returns

Zero for success or else an error code

## 8.5.3.7 BLS\_BLS381\_SIGN()

```
int BLS_BLS381_SIGN (
    octet * SIG,
    octet * M,
    octet * S )
```

## Parameters

<i>SIG</i>	signature
<i>M</i>	message to be signed
<i>S</i>	Private key

## Returns

Zero for success or else an error code

## 8.5.3.8 BLS\_BLS381\_VERIFY()

```
int BLS_BLS381_VERIFY (
    octet * SIG,
    octet * M,
    octet * W )
```

## Parameters

<i>SIG</i>	signature
<i>M</i>	message whose signature is to be verified.
<i>W</i>	Public key

**Returns**

Zero for success or else an error code

## 8.6 config\_big\_1024\_58.h File Reference

Config BIG Header File.

```
#include "amcl.h"
```

**Macros**

- #define [MODBYTES\\_1024\\_58](#) 128
- #define [BASEBITS\\_1024\\_58](#) 58

### 8.6.1 Detailed Description

**Author**

Mike Scott

### 8.6.2 Macro Definition Documentation

#### 8.6.2.1 BASEBITS\_1024\_58

```
#define BASEBITS_1024_58 58
```

Numbers represented to base  $2 * \text{BASEBITS}$

#### 8.6.2.2 MODBYTES\_1024\_58

```
#define MODBYTES_1024_58 128
```

Number of bytes in Modulus

## 8.7 config\_big\_384\_58.h File Reference

Config BIG Header File.

```
#include "amcl.h"
```

## Macros

- #define [MODBYTES\\_384\\_58](#) 48
- #define [BASEBITS\\_384\\_58](#) 58

### 8.7.1 Detailed Description

#### Author

Mike Scott

### 8.7.2 Macro Definition Documentation

#### 8.7.2.1 BASEBITS\_384\_58

```
#define BASEBITS_384_58 58
```

Numbers represented to base  $2^{\text{BASEBITS}}$

#### 8.7.2.2 MODBYTES\_384\_58

```
#define MODBYTES_384_58 48
```

Number of bytes in Modulus

## 8.8 config\_big\_512\_60.h File Reference

Config BIG Header File.

```
#include "amcl.h"
```

## Macros

- #define [MODBYTES\\_512\\_60](#) 64
- #define [BASEBITS\\_512\\_60](#) 60

### 8.8.1 Detailed Description

#### Author

Mike Scott

## 8.8.2 Macro Definition Documentation

### 8.8.2.1 BASEBITS\_512\_60

```
#define BASEBITS_512_60 60
```

Numbers represented to base  $2^{\text{BASEBITS}}$

### 8.8.2.2 MODBYTES\_512\_60

```
#define MODBYTES_512_60 64
```

Number of bytes in Modulus

## 8.9 config\_ff\_2048.h File Reference

COnfig FF Header File.

```
#include "amcl.h"  
#include "config_big_1024_58.h"
```

### Macros

- `#define FFLEN_2048 2`

### 8.9.1 Detailed Description

#### Author

Mike Scott

## 8.9.2 Macro Definition Documentation

### 8.9.2.1 FFLEN\_2048

```
#define FFLEN_2048 2
```

$2^n$  multiplier of BIGBITS to specify supported Finite Field size, e.g  $2048=256*2^3$  where BIGBITS=256



## 8.10 config\_ff\_4096.h File Reference

COnfig FF Header File.

```
#include "amcl.h"
#include "config_big_512_60.h"
```

### Macros

- `#define FFLEN_4096 8`

### 8.10.1 Detailed Description

#### Author

Mike Scott

### 8.10.2 Macro Definition Documentation

#### 8.10.2.1 FFLEN\_4096

```
#define FFLEN_4096 8
```

$2^n$  multiplier of BIGBITS to specify supported Finite Field size, e.g  $2048=256*2^3$  where BIGBITS=256

## 8.11 ecdh\_BLS381.h File Reference

ECDH Header file for implementation of standard EC protocols.

```
#include "ecp_BLS381.h"
#include "ecdh_support.h"
```

### Macros

- `#define EGS_BLS381 MODBYTES_384_58`
- `#define EFS_BLS381 MODBYTES_384_58`
- `#define ECDH_OK 0`
- `#define ECDH_INVALID_PUBLIC_KEY -2`
- `#define ECDH_ERROR -3`
- `#define ECDH_INVALID -4`

## Functions

- int `ECP_BLS381_KEY_PAIR_GENERATE` (`csprng *R`, `octet *s`, `octet *W`)  
*Generate an ECC public/private key pair.*
- int `ECP_BLS381_PUBLIC_KEY_VALIDATE` (`octet *W`)  
*Validate an ECC public key.*
- int `ECP_BLS381_SVDP_DH` (`octet *s`, `octet *W`, `octet *K`)  
*Generate Diffie-Hellman shared key.*
- void `ECP_BLS381_ECIES_ENCRYPT` (`int h`, `octet *P1`, `octet *P2`, `csprng *R`, `octet *W`, `octet *M`, `int len`, `octet *V`, `octet *C`, `octet *T`)  
*ECIES Encryption.*
- int `ECP_BLS381_ECIES_DECRYPT` (`int h`, `octet *P1`, `octet *P2`, `octet *V`, `octet *C`, `octet *T`, `octet *U`, `octet *M`)  
*ECIES Decryption.*
- int `ECP_BLS381_SP_DSA` (`int h`, `csprng *R`, `octet *k`, `octet *s`, `octet *M`, `octet *c`, `octet *d`)  
*ECDSA Signature.*
- int `ECP_BLS381_VP_DSA` (`int h`, `octet *W`, `octet *M`, `octet *c`, `octet *d`)  
*ECDSA Signature Verification.*

### 8.11.1 Detailed Description

#### Author

Mike Scott

### 8.11.2 Macro Definition Documentation

#### 8.11.2.1 ECDH\_ERROR

```
#define ECDH_ERROR -3
```

ECDH Internal Error

#### 8.11.2.2 ECDH\_INVALID

```
#define ECDH_INVALID -4
```

ECDH Internal Error

#### 8.11.2.3 ECDH\_INVALID\_PUBLIC\_KEY

```
#define ECDH_INVALID_PUBLIC_KEY -2
```

Public Key is Invalid

## 8.11.2.4 ECDH\_OK

```
#define ECDH_OK 0
```

Function completed without error

## 8.11.2.5 EFS\_BLS381

```
#define EFS_BLS381 MODBYTES_384_58
```

ECC Field Size in bytes

## 8.11.2.6 EGS\_BLS381

```
#define EGS_BLS381 MODBYTES_384_58
```

ECC Group Size in bytes

## 8.11.3 Function Documentation

## 8.11.3.1 ECP\_BLS381\_ECIES\_DECRYPT()

```
int ECP_BLS381_ECIES_DECRYPT (
    int h,
    octet * P1,
    octet * P2,
    octet * V,
    octet * C,
    octet * T,
    octet * U,
    octet * M )
```

IEEE-1363 ECIES Decryption

## Parameters

<i>h</i>	is the hash type
<i>P1</i>	input Key Derivation parameters
<i>P2</i>	input Encoding parameters
<i>V</i>	component of the input ciphertext
<i>C</i>	the input ciphertext
<i>T</i>	the input HMAC tag, part of the ciphertext
<i>U</i>	the input private key for decryption
<i>M</i>	the output plaintext message

**Returns**

1 if successful, else 0

**8.11.3.2 ECP\_BLS381\_ECIES\_ENCRYPT()**

```
void ECP_BLS381_ECIES_ENCRYPT (
    int h,
    octet * P1,
    octet * P2,
    csprng * R,
    octet * W,
    octet * M,
    int len,
    octet * V,
    octet * C,
    octet * T )
```

## IEEE-1363 ECIES Encryption

**Parameters**

<i>h</i>	is the hash type
<i>P1</i>	input Key Derivation parameters
<i>P2</i>	input Encoding parameters
<i>R</i>	is a pointer to a cryptographically secure random number generator
<i>W</i>	the input public key of the receiving party
<i>M</i>	is the plaintext message to be encrypted
<i>len</i>	the length of the HMAC tag
<i>V</i>	component of the output ciphertext
<i>C</i>	the output ciphertext
<i>T</i>	the output HMAC tag, part of the ciphertext

**8.11.3.3 ECP\_BLS381\_KEY\_PAIR\_GENERATE()**

```
int ECP_BLS381_KEY_PAIR_GENERATE (
    csprng * R,
    octet * s,
    octet * W )
```

**Parameters**

<i>R</i>	is a pointer to a cryptographically secure random number generator
<i>s</i>	the private key, an output internally randomly generated if R!=NULL, otherwise must be provided as an input
<i>W</i>	the output public key, which is s.G, where G is a fixed generator

**Returns**

0 or an error code

**8.11.3.4 ECP\_BLS381\_PUBLIC\_KEY\_VALIDATE()**

```
int ECP_BLS381_PUBLIC_KEY_VALIDATE (
    octet * W )
```

**Parameters**

<i>W</i>	the input public key to be validated
----------	--------------------------------------

**Returns**

0 if public key is OK, or an error code

**8.11.3.5 ECP\_BLS381\_SP\_DSA()**

```
int ECP_BLS381_SP_DSA (
    int h,
    csprng * R,
    octet * k,
    octet * s,
    octet * M,
    octet * c,
    octet * d )
```

**IEEE-1363 ECDSA Signature****Parameters**

<i>h</i>	is the hash type
<i>R</i>	is a pointer to a cryptographically secure random number generator
<i>k</i>	Ephemeral key. This value is used when R=NULL
<i>s</i>	the input private signing key
<i>M</i>	the input message to be signed
<i>c</i>	component of the output signature
<i>d</i>	component of the output signature

**8.11.3.6 ECP\_BLS381\_SVDP\_DH()**

```
int ECP_BLS381_SVDP_DH (
    octet * s,
```

```

    octet * W,
    octet * K )

```

IEEE-1363 Diffie-Hellman shared secret calculation

#### Parameters

<i>s</i>	is the input private key,
<i>W</i>	the input public key of the other party
<i>K</i>	the output shared key, in fact the x-coordinate of $s.W$

#### Returns

0 or an error code

#### 8.11.3.7 ECP\_BLS381\_VP\_DSA()

```

int ECP_BLS381_VP_DSA (
    int h,
    octet * W,
    octet * M,
    octet * c,
    octet * d )

```

IEEE-1363 ECDSA Signature Verification

#### Parameters

<i>h</i>	is the hash type
<i>W</i>	the input public key
<i>M</i>	the input message
<i>c</i>	component of the input signature
<i>d</i>	component of the input signature

#### Returns

0 or an error code

## 8.12 ecdh\_support.h File Reference

ECDH Support Header File.

```
#include "amcl.h"
```

## Functions

- void `ehashit` (int sha, `octet` \*p, int n, `octet` \*x, `octet` \*w, int pad)  
*general purpose hash function  $w = \text{hash}(p|n|x|y)$*
- void `HASH` (int h, `octet` \*I, `octet` \*O)  
*hash an octet into another octet*
- int `HMAC` (int h, `octet` \*M, `octet` \*K, int len, `octet` \*tag)  
*HMAC of message M using key K to create tag of length len in octet tag.*
- void `KDF2` (int h, `octet` \*Z, `octet` \*P, int len, `octet` \*K)  
*Key Derivation Function - generates key K from inputs Z and P*
- void `PBKDF2` (int h, `octet` \*P, `octet` \*S, int rep, int len, `octet` \*K)  
*Password Based Key Derivation Function - generates key K from password, salt and repeat counter.*
- void `AES_CBC_IV0_ENCRYPT` (`octet` \*K, `octet` \*P, `octet` \*C)  
*AES encrypts a plaintext to a ciphertext.*
- int `AES_CBC_IV0_DECRYPT` (`octet` \*K, `octet` \*C, `octet` \*P)  
*AES encrypts a plaintext to a ciphertext.*

### 8.12.1 Detailed Description

#### Author

Mike Scott

### 8.12.2 Function Documentation

#### 8.12.2.1 `AES_CBC_IV0_DECRYPT()`

```
int AES_CBC_IV0_DECRYPT (
    octet * K,
    octet * C,
    octet * P )
```

IEEE-1363 `AES_CBC_IV0_DECRYPT` function. Decrypts in CBC mode with a zero IV.

#### Parameters

<i>K</i>	AES key
<i>C</i>	input ciphertext octet
<i>P</i>	output plaintext octet

#### Returns

0 if bad input, else 1

### 8.12.2.2 AES\_CBC\_IV0\_ENCRYPT()

```
void AES_CBC_IV0_ENCRYPT (
    octet * K,
    octet * P,
    octet * C )
```

IEEE-1363 AES\_CBC\_IV0\_ENCRYPT function. Encrypts in CBC mode with a zero IV, padding as necessary to create a full final block.

#### Parameters

<i>K</i>	AES key
<i>P</i>	input plaintext octet
<i>C</i>	output ciphertext octet

### 8.12.2.3 ehashit()

```
void ehashit (
    int sha,
    octet * p,
    int n,
    octet * x,
    octet * w,
    int pad )
```

#### Parameters

<i>sha</i>	is the hash type
<i>p</i>	first octect involved in the hash
<i>n</i>	integer involved in the hash
<i>x</i>	second octect involved in the hash
<i>w</i>	output
<i>pad</i>	padding

### 8.12.2.4 HASH()

```
void HASH (
    int h,
    octet * I,
    octet * O )
```

#### Parameters

<i>h</i>	is the hash type
<i>I</i>	input octet
<i>O</i>	output octet -
	H(I)



## 8.12.2.5 HMAC()

```
int HMAC (
    int h,
    octet * M,
    octet * K,
    int len,
    octet * tag )
```

IEEE-1363 MAC1 function. Uses SHA256 internally.

## Parameters

<i>h</i>	is the hash type
<i>M</i>	input message octet
<i>K</i>	input encryption key
<i>len</i>	is output desired length of HMAC tag
<i>tag</i>	is the output HMAC

## Returns

0 for bad parameters, else 1

## 8.12.2.6 KDF2()

```
void KDF2 (
    int h,
    octet * Z,
    octet * P,
    int len,
    octet * K )
```

IEEE-1363 KDF2 Key Derivation Function. Uses SHA256 internally.

## Parameters

<i>h</i>	is the hash type
<i>Z</i>	input octet
<i>P</i>	input key derivation parameters - can be NULL
<i>len</i>	is output desired length of key
<i>K</i>	is the derived key

### 8.12.2.7 PBKDF2()

```
void PBKDF2 (
    int h,
    octet * P,
    octet * S,
    int rep,
    int len,
    octet * K )
```

PBKDF2 Password Based Key Derivation Function. Uses SHA256 internally.

#### Parameters

<i>h</i>	is the hash type
<i>P</i>	input password
<i>S</i>	input salt
<i>rep</i>	Number of times to be iterated.
<i>len</i>	is output desired length
<i>K</i>	is the derived key

## 8.13 ecp2\_BLS381.h File Reference

ECP2 Header File.

```
#include "fp2_BLS381.h"
#include "config_curve_BLS381.h"
```

### Data Structures

- struct [ECP2\\_BLS381](#)  
*ECP2 Structure - Elliptic Curve Point over quadratic extension field.*

### Functions

- int [ECP2\\_BLS381\\_isinf](#) ([ECP2\\_BLS381](#) \*P)  
*Tests for ECP2 point equal to infinity.*
- void [ECP2\\_BLS381\\_copy](#) ([ECP2\\_BLS381](#) \*P, [ECP2\\_BLS381](#) \*Q)  
*Copy ECP2 point to another ECP2 point.*
- void [ECP2\\_BLS381\\_inf](#) ([ECP2\\_BLS381](#) \*P)  
*Set ECP2 to point-at-infinity.*
- int [ECP2\\_BLS381\\_equals](#) ([ECP2\\_BLS381](#) \*P, [ECP2\\_BLS381](#) \*Q)  
*Tests for equality of two ECP2s.*
- void [ECP2\\_BLS381\\_affine](#) ([ECP2\\_BLS381](#) \*P)  
*Converts an ECP2 point from Projective (x,y,z) coordinates to affine (x,y) coordinates.*
- int [ECP2\\_BLS381\\_get](#) ([FP2\\_BLS381](#) \*x, [FP2\\_BLS381](#) \*y, [ECP2\\_BLS381](#) \*P)  
*Extract x and y coordinates of an ECP2 point P.*

- void `ECP2_BLS381_output` (`ECP2_BLS381 *P`)  
*Formats and outputs an ECP2 point to the console, converted to affine coordinates.*
- void `ECP2_BLS381_outputxyz` (`ECP2_BLS381 *P`)  
*Formats and outputs an ECP2 point to the console, in projective coordinates.*
- void `ECP2_BLS381_toOctet` (`octet *S`, `ECP2_BLS381 *P`)  
*Formats and outputs an ECP2 point to an octet string.*
- int `ECP2_BLS381_fromOctet` (`ECP2_BLS381 *P`, `octet *S`)  
*Creates an ECP2 point from an octet string.*
- void `ECP2_BLS381_rhs` (`FP2_BLS381 *r`, `FP2_BLS381 *x`)  
*Calculate Right Hand Side of curve equation  $y^2=f(x)$*
- int `ECP2_BLS381_set` (`ECP2_BLS381 *P`, `FP2_BLS381 *x`, `FP2_BLS381 *y`)  
*Set ECP2 to point(x,y) given x and y.*
- int `ECP2_BLS381_setx` (`ECP2_BLS381 *P`, `FP2_BLS381 *x`)  
*Set ECP to point(x,[y]) given x.*
- void `ECP2_BLS381_neg` (`ECP2_BLS381 *P`)  
*Negation of an ECP2 point.*
- int `ECP2_BLS381_dbl` (`ECP2_BLS381 *P`)  
*Doubles an ECP2 instance P.*
- int `ECP2_BLS381_add` (`ECP2_BLS381 *P`, `ECP2_BLS381 *Q`)  
*Adds ECP2 instance Q to ECP2 instance P.*
- void `ECP2_BLS381_sub` (`ECP2_BLS381 *P`, `ECP2_BLS381 *Q`)  
*Subtracts ECP instance Q from ECP2 instance P.*
- void `ECP2_BLS381_mul` (`ECP2_BLS381 *P`, `BIG_384_58 b`)  
*Multiplies an ECP2 instance P by a BIG, side-channel resistant.*
- void `ECP2_BLS381_frob` (`ECP2_BLS381 *P`, `FP2_BLS381 *f`)  
*Multiplies an ECP2 instance P by the internal modulus p, using precalculated Frobenius constant f.*
- void `ECP2_BLS381_mul4` (`ECP2_BLS381 *P`, `ECP2_BLS381 *Q`, `BIG_384_58 *b`)  
*Calculates  $P=b[0]*Q[0]+b[1]*Q[1]+b[2]*Q[2]+b[3]*Q[3]$ .*
- void `ECP2_BLS381_mapit` (`ECP2_BLS381 *P`, `octet *w`)  
*Maps random BIG to curve point of correct order.*
- void `ECP2_BLS381_generator` (`ECP2_BLS381 *G`)  
*Get Group Generator from ROM.*

## Variables

- const int `CURVE_A_BLS381`
- const int `CURVE_B_I_BLS381`
- const `BIG_384_58 CURVE_B_BLS381`
- const `BIG_384_58 CURVE_Order_BLS381`
- const `BIG_384_58 CURVE_Cof_BLS381`
- const `BIG_384_58 CURVE_Bnx_BLS381`
- const `BIG_384_58 Fra_BLS381`
- const `BIG_384_58 Frb_BLS381`
- const `BIG_384_58 CURVE_Gx_BLS381`
- const `BIG_384_58 CURVE_Gy_BLS381`
- const `BIG_384_58 CURVE_Pxa_BLS381`
- const `BIG_384_58 CURVE_Pxb_BLS381`
- const `BIG_384_58 CURVE_Pya_BLS381`
- const `BIG_384_58 CURVE_Pyb_BLS381`

### 8.13.1 Detailed Description

#### Author

Mike Scott

### 8.13.2 Function Documentation

#### 8.13.2.1 ECP2\_BLS381\_add()

```
int ECP2_BLS381_add (
    ECP2_BLS381 * P,
    ECP2_BLS381 * Q )
```

##### Parameters

<i>P</i>	ECP2 instance, on exit =P+Q
<i>Q</i>	ECP2 instance to be added to P

#### 8.13.2.2 ECP2\_BLS381\_affine()

```
void ECP2_BLS381_affine (
    ECP2_BLS381 * P )
```

##### Parameters

<i>P</i>	ECP2 instance to be converted to affine form
----------	--

#### 8.13.2.3 ECP2\_BLS381\_copy()

```
void ECP2_BLS381_copy (
    ECP2_BLS381 * P,
    ECP2_BLS381 * Q )
```

##### Parameters

<i>P</i>	ECP2 instance, on exit = Q
<i>Q</i>	ECP2 instance to be copied

## 8.13.2.4 ECP2\_BLS381\_dbl()

```
int ECP2_BLS381_dbl (
    ECP2_BLS381 * P )
```

## Parameters

<i>P</i>	ECP2 instance, on exit =2*P
----------	-----------------------------

## 8.13.2.5 ECP2\_BLS381\_equals()

```
int ECP2_BLS381_equals (
    ECP2_BLS381 * P,
    ECP2_BLS381 * Q )
```

## Parameters

<i>P</i>	ECP2 instance to be compared
<i>Q</i>	ECP2 instance to be compared

## Returns

1 if P=Q, else returns 0

## 8.13.2.6 ECP2\_BLS381\_frob()

```
void ECP2_BLS381_frob (
    ECP2_BLS381 * P,
    FP2_BLS381 * f )
```

Fast point multiplication using Frobenius

## Parameters

<i>P</i>	ECP2 instance, on exit = p*P
<i>f</i>	FP2 precalculated Frobenius constant

## 8.13.2.7 ECP2\_BLS381\_fromOctet()

```
int ECP2_BLS381_fromOctet (
    ECP2_BLS381 * P,
    octet * S )
```

The octet string is in the form  $x|y$ . The real and imaginary parts of the  $x$  and  $y$  coordinates are in big-endian base 256 form.

#### Parameters

$P$	ECP2 instance to be created from the octet string
$S$	input octet string return 1 if octet string corresponds to a point on the curve, else 0

#### 8.13.2.8 ECP2\_BLS381\_generator()

```
void ECP2_BLS381_generator (
    ECP2_BLS381 * G )
```

#### Parameters

$G$	ECP2 instance
-----	---------------

#### 8.13.2.9 ECP2\_BLS381\_get()

```
int ECP2_BLS381_get (
    FP2_BLS381 * x,
    FP2_BLS381 * y,
    ECP2_BLS381 * P )
```

If  $x=y$ , returns only  $x$

#### Parameters

$x$	FP2 on exit = $x$ coordinate of point
$y$	FP2 on exit = $y$ coordinate of point (unless $x=y$ )
$P$	ECP2 instance ( $x,y$ )

#### Returns

-1 if  $P$  is point-at-infinity, else 0

#### 8.13.2.10 ECP2\_BLS381\_inf()

```
void ECP2_BLS381_inf (
    ECP2_BLS381 * P )
```

## Parameters

<i>P</i>	ECP2 instance to be set to infinity
----------	-------------------------------------

## 8.13.2.11 ECP2\_BLS381\_isinf()

```
int ECP2_BLS381_isinf (
    ECP2_BLS381 * P )
```

## Parameters

<i>P</i>	ECP2 point to be tested
----------	-------------------------

## Returns

1 if infinity, else returns 0

## 8.13.2.12 ECP2\_BLS381\_mapit()

```
void ECP2_BLS381_mapit (
    ECP2_BLS381 * P,
    octet * w )
```

## Parameters

<i>P</i>	ECP2 instance of correct order
<i>w</i>	OCTET byte array to be mapped

## 8.13.2.13 ECP2\_BLS381\_mul()

```
void ECP2_BLS381_mul (
    ECP2_BLS381 * P,
    BIG_384_58 b )
```

Uses fixed sized windows.

## Parameters

<i>P</i>	ECP2 instance, on exit =b*P
<i>b</i>	BIG number multiplier

## 8.13.2.14 ECP2\_BLS381\_mul4()

```
void ECP2_BLS381_mul4 (
    ECP2_BLS381 * P,
    ECP2_BLS381 * Q,
    BIG_384_58 * b )
```

## Parameters

<i>P</i>	ECP2 instance, on exit = $b[0]*Q[0]+b[1]*Q[1]+b[2]*Q[2]+b[3]*Q[3]$
<i>Q</i>	ECP2 array of 4 points
<i>b</i>	BIG array of 4 multipliers

## 8.13.2.15 ECP2\_BLS381\_neg()

```
void ECP2_BLS381_neg (
    ECP2_BLS381 * P )
```

## Parameters

<i>P</i>	ECP2 instance, on exit = -P
----------	-----------------------------

## 8.13.2.16 ECP2\_BLS381\_output()

```
void ECP2_BLS381_output (
    ECP2_BLS381 * P )
```

## Parameters

<i>P</i>	ECP2 instance to be printed
----------	-----------------------------

## 8.13.2.17 ECP2\_BLS381\_outputxyz()

```
void ECP2_BLS381_outputxyz (
    ECP2_BLS381 * P )
```

## Parameters

<i>P</i>	ECP2 instance to be printed
----------	-----------------------------



## 8.13.2.18 ECP2\_BLS381\_rhs()

```
void ECP2_BLS381_rhs (
    FP2_BLS381 * r,
    FP2_BLS381 * x )
```

Function  $f(x)=x^3+Ax+B$  Used internally.

## Parameters

<i>r</i>	FP2 value of $f(x)$
<i>x</i>	FP2 instance

## 8.13.2.19 ECP2\_BLS381\_set()

```
int ECP2_BLS381_set (
    ECP2_BLS381 * P,
    FP2_BLS381 * x,
    FP2_BLS381 * y )
```

Point P set to infinity if no such point on the curve.

## Parameters

<i>P</i>	ECP2 instance to be set (x,y)
<i>x</i>	FP2 x coordinate of point
<i>y</i>	FP2 y coordinate of point

## Returns

1 if point exists, else 0

## 8.13.2.20 ECP2\_BLS381\_setx()

```
int ECP2_BLS381_setx (
    ECP2_BLS381 * P,
    FP2_BLS381 * x )
```

Point P set to infinity if no such point on the curve. Otherwise y coordinate is calculated from x.

## Parameters

<i>P</i>	ECP instance to be set (x,[y])
<i>x</i>	FP2 x coordinate of point

**Returns**

1 if point exists, else 0

**8.13.2.21 ECP2\_BLS381\_sub()**

```
void ECP2_BLS381_sub (
    ECP2_BLS381 * P,
    ECP2_BLS381 * Q )
```

**Parameters**

<i>P</i>	ECP2 instance, on exit =P-Q
<i>Q</i>	ECP2 instance to be subtracted from P

**8.13.2.22 ECP2\_BLS381\_toOctet()**

```
void ECP2_BLS381_toOctet (
    octet * S,
    ECP2_BLS381 * P )
```

The octet string is created in the form x|y. Convert the real and imaginary parts of the x and y coordinates to big-endian base 256 form.

**Parameters**

<i>S</i>	output octet string
<i>P</i>	ECP2 instance to be converted to an octet string

**8.13.3 Variable Documentation****8.13.3.1 CURVE\_A\_BLS381**

```
const int CURVE_A_BLS381
```

Elliptic curve A parameter

**8.13.3.2 CURVE\_B\_BLS381**

```
const BIG_384_58 CURVE_B_BLS381
```

Elliptic curve B parameter

### 8.13.3.3 CURVE\_B\_I\_BLS381

```
const int CURVE_B_I_BLS381
```

Elliptic curve B parameter

### 8.13.3.4 CURVE\_Bnx\_BLS381

```
const BIG_384_58 CURVE_Bnx_BLS381
```

Elliptic curve parameter

### 8.13.3.5 CURVE\_Cof\_BLS381

```
const BIG_384_58 CURVE_Cof_BLS381
```

Elliptic curve cofactor

### 8.13.3.6 CURVE\_Gx\_BLS381

```
const BIG_384_58 CURVE_Gx_BLS381
```

x-coordinate of generator point in group G1

### 8.13.3.7 CURVE\_Gy\_BLS381

```
const BIG_384_58 CURVE_Gy_BLS381
```

y-coordinate of generator point in group G1

### 8.13.3.8 CURVE\_Order\_BLS381

```
const BIG_384_58 CURVE_Order_BLS381
```

Elliptic curve group order

### 8.13.3.9 CURVE\_Pxa\_BLS381

```
const BIG_384_58 CURVE_Pxa_BLS381
```

real part of x-coordinate of generator point in group G2

### 8.13.3.10 CURVE\_Pxb\_BLS381

```
const BIG_384_58 CURVE_Pxb_BLS381
```

imaginary part of x-coordinate of generator point in group G2

### 8.13.3.11 CURVE\_Pya\_BLS381

```
const BIG_384_58 CURVE_Pya_BLS381
```

real part of y-coordinate of generator point in group G2

### 8.13.3.12 CURVE\_Pyb\_BLS381

```
const BIG_384_58 CURVE_Pyb_BLS381
```

imaginary part of y-coordinate of generator point in group G2

### 8.13.3.13 Fra\_BLS381

```
const BIG_384_58 Fra_BLS381
```

real part of BN curve Frobenius Constant

### 8.13.3.14 Frb\_BLS381

```
const BIG_384_58 Frb_BLS381
```

imaginary part of BN curve Frobenius Constant

## 8.14 ecp\_BLS381.h File Reference

ECP Header File.

```
#include "fp_BLS381.h"  
#include "config_curve_BLS381.h"
```

### Data Structures

- struct [ECP\\_BLS381](#)

*ECP structure - Elliptic Curve Point over base field.*

## Functions

- int `ECP_BLS381_isingf` (`ECP_BLS381 *P`)  
*Tests for ECP point equal to infinity.*
- int `ECP_BLS381_equals` (`ECP_BLS381 *P`, `ECP_BLS381 *Q`)  
*Tests for equality of two ECPs.*
- void `ECP_BLS381_copy` (`ECP_BLS381 *P`, `ECP_BLS381 *Q`)  
*Copy ECP point to another ECP point.*
- void `ECP_BLS381_neg` (`ECP_BLS381 *P`)  
*Negation of an ECP point.*
- void `ECP_BLS381_inf` (`ECP_BLS381 *P`)  
*Set ECP to point-at-infinity.*
- void `ECP_BLS381_rhs` (`FP_BLS381 *r`, `FP_BLS381 *x`)  
*Calculate Right Hand Side of curve equation  $y^2=f(x)$*
- int `ECP_BLS381_set` (`ECP_BLS381 *P`, `BIG_384_58 x`, `BIG_384_58 y`)  
*Set ECP to point(x,y) given x and y.*
- int `ECP_BLS381_get` (`BIG_384_58 x`, `BIG_384_58 y`, `ECP_BLS381 *P`)  
*Extract x and y coordinates of an ECP point P.*
- void `ECP_BLS381_add` (`ECP_BLS381 *P`, `ECP_BLS381 *Q`)  
*Adds ECP instance Q to ECP instance P.*
- void `ECP_BLS381_sub` (`ECP_BLS381 *P`, `ECP_BLS381 *Q`)  
*Subtracts ECP instance Q from ECP instance P.*
- int `ECP_BLS381_setx` (`ECP_BLS381 *P`, `BIG_384_58 x`, int s)  
*Set ECP to point(x,y) given just x and sign of y.*
- void `ECP_BLS381_cfp` (`ECP_BLS381 *Q`)  
*Multiplies Point by curve co-factor.*
- void `ECP_BLS381_mapit` (`ECP_BLS381 *Q`, `octet *w`)  
*Maps random BIG to curve point of correct order.*
- void `ECP_BLS381_affine` (`ECP_BLS381 *P`)  
*Converts an ECP point from Projective (x,y,z) coordinates to affine (x,y) coordinates.*
- void `ECP_BLS381_outputxyz` (`ECP_BLS381 *P`)  
*Formats and outputs an ECP point to the console, in projective coordinates.*
- void `ECP_BLS381_output` (`ECP_BLS381 *P`)  
*Formats and outputs an ECP point to the console, converted to affine coordinates.*
- void `ECP_BLS381_rawoutput` (`ECP_BLS381 *P`)  
*Formats and outputs an ECP point to the console.*
- void `ECP_BLS381_toOctet` (`octet *S`, `ECP_BLS381 *P`, bool c)  
*Formats and outputs an ECP point to an octet string The octet string is normally in the standard form 0x04|x|y Here x (and y) are the x and y coordinates in left justified big-endian base 256 form. For Montgomery curve it is 0x06|x If c is true, only the x coordinate is provided as in 0x2|x if y is even, or 0x3|x if y is odd.*
- int `ECP_BLS381_fromOctet` (`ECP_BLS381 *P`, `octet *S`)  
*Creates an ECP point from an octet string.*
- void `ECP_BLS381_dbl` (`ECP_BLS381 *P`)  
*Doubles an ECP instance P.*
- void `ECP_BLS381_pinmul` (`ECP_BLS381 *P`, int i, int b)  
*Multiplies an ECP instance P by a small integer, side-channel resistant.*
- void `ECP_BLS381_mul` (`ECP_BLS381 *P`, `BIG_384_58 b`)  
*Multiplies an ECP instance P by a BIG, side-channel resistant.*
- void `ECP_BLS381_mul2` (`ECP_BLS381 *P`, `ECP_BLS381 *Q`, `BIG_384_58 e`, `BIG_384_58 f`)  
*Calculates double multiplication  $P=e*P+f*Q$ , side-channel resistant.*
- void `ECP_BLS381_generator` (`ECP_BLS381 *G`)  
*Get Group Generator from ROM.*

## Variables

- const int CURVE\_A\_BLS381
- const int CURVE\_Cof\_I\_BLS381
- const int CURVE\_B\_I\_BLS381
- const BIG\_384\_58 CURVE\_B\_BLS381
- const BIG\_384\_58 CURVE\_Order\_BLS381
- const BIG\_384\_58 CURVE\_Cof\_BLS381
- const BIG\_384\_58 CURVE\_Gx\_BLS381
- const BIG\_384\_58 CURVE\_Gy\_BLS381
- const BIG\_384\_58 CURVE\_Pxa\_BLS381
- const BIG\_384\_58 CURVE\_Pxb\_BLS381
- const BIG\_384\_58 CURVE\_Pya\_BLS381
- const BIG\_384\_58 CURVE\_Pyb\_BLS381
- const BIG\_384\_58 CURVE\_Pxaa\_BLS381
- const BIG\_384\_58 CURVE\_Pxab\_BLS381
- const BIG\_384\_58 CURVE\_Pxba\_BLS381
- const BIG\_384\_58 CURVE\_Pxbb\_BLS381
- const BIG\_384\_58 CURVE\_Pyaa\_BLS381
- const BIG\_384\_58 CURVE\_Pyab\_BLS381
- const BIG\_384\_58 CURVE\_Pyba\_BLS381
- const BIG\_384\_58 CURVE\_Pybb\_BLS381
- const BIG\_384\_58 CURVE\_Pxaaa\_BLS381
- const BIG\_384\_58 CURVE\_Pxaab\_BLS381
- const BIG\_384\_58 CURVE\_Pxaba\_BLS381
- const BIG\_384\_58 CURVE\_Pxab\_bLS381
- const BIG\_384\_58 CURVE\_Pxbaa\_BLS381
- const BIG\_384\_58 CURVE\_Pxbab\_BLS381
- const BIG\_384\_58 CURVE\_Pxbba\_BLS381
- const BIG\_384\_58 CURVE\_Pxbbb\_BLS381
- const BIG\_384\_58 CURVE\_Pyaaa\_BLS381
- const BIG\_384\_58 CURVE\_Pyaab\_BLS381
- const BIG\_384\_58 CURVE\_Pyaba\_BLS381
- const BIG\_384\_58 CURVE\_Pyabb\_BLS381
- const BIG\_384\_58 CURVE\_Pybaa\_BLS381
- const BIG\_384\_58 CURVE\_Pybab\_BLS381
- const BIG\_384\_58 CURVE\_Pybba\_BLS381
- const BIG\_384\_58 CURVE\_Pybbb\_BLS381
- const BIG\_384\_58 CURVE\_Bnx\_BLS381
- const BIG\_384\_58 CURVE\_Cru\_BLS381
- const BIG\_384\_58 Fra\_BLS381
- const BIG\_384\_58 Frb\_BLS381
- const BIG\_384\_58 CURVE\_W\_BLS381 [2]
- const BIG\_384\_58 CURVE\_SB\_BLS381 [2][2]
- const BIG\_384\_58 CURVE\_WB\_BLS381 [4]
- const BIG\_384\_58 CURVE\_BB\_BLS381 [4][4]

### 8.14.1 Detailed Description

#### Author

Mike Scott

## 8.14.2 Function Documentation

### 8.14.2.1 ECP\_BLS381\_add()

```
void ECP_BLS381_add (
    ECP_BLS381 * P,
    ECP_BLS381 * Q )
```

#### Parameters

<i>P</i>	ECP instance, on exit =P+Q
<i>Q</i>	ECP instance to be added to P

### 8.14.2.2 ECP\_BLS381\_affine()

```
void ECP_BLS381_affine (
    ECP_BLS381 * P )
```

#### Parameters

<i>P</i>	ECP instance to be converted to affine form
----------	---

### 8.14.2.3 ECP\_BLS381\_cfp()

```
void ECP_BLS381_cfp (
    ECP_BLS381 * Q )
```

#### Parameters

<i>Q</i>	ECP instance
----------	--------------

### 8.14.2.4 ECP\_BLS381\_copy()

```
void ECP_BLS381_copy (
    ECP_BLS381 * P,
    ECP_BLS381 * Q )
```

## Parameters

<i>P</i>	ECP instance, on exit = <i>Q</i>
<i>Q</i>	ECP instance to be copied

## 8.14.2.5 ECP\_BLS381\_dbl()

```
void ECP_BLS381_dbl (
    ECP_BLS381 * P )
```

## Parameters

<i>P</i>	ECP instance, on exit =2* <i>P</i>
----------	------------------------------------

## 8.14.2.6 ECP\_BLS381\_equals()

```
int ECP_BLS381_equals (
    ECP_BLS381 * P,
    ECP_BLS381 * Q )
```

## Parameters

<i>P</i>	ECP instance to be compared
<i>Q</i>	ECP instance to be compared

## Returns

1 if *P*=*Q*, else returns 0

## 8.14.2.7 ECP\_BLS381\_fromOctet()

```
int ECP_BLS381_fromOctet (
    ECP_BLS381 * P,
    octet * S )
```

The octet string is normally in the standard form 0x04|x|y Here x (and y) are the x and y coordinates in left justified big-endian base 256 form. For Montgomery curve it is 0x06|x If in compressed form only the x coordinate is provided as in 0x2|x if y is even, or 0x3|x if y is odd

## Parameters

<i>P</i>	ECP instance to be created from the octet string
<i>S</i>	input octet string return 1 if octet string corresponds to a point on the curve, else 0



### 8.14.2.8 ECP\_BLS381\_generator()

```
void ECP_BLS381_generator (
    ECP_BLS381 * G )
```

#### Parameters

<i>G</i>	ECP instance
----------	--------------

### 8.14.2.9 ECP\_BLS381\_get()

```
int ECP_BLS381_get (
    BIG_384_58 x,
    BIG_384_58 y,
    ECP_BLS381 * P )
```

If  $x=y$ , returns only  $x$

#### Parameters

<i>x</i>	BIG on exit = x coordinate of point
<i>y</i>	BIG on exit = y coordinate of point (unless $x=y$ )
<i>P</i>	ECP instance (x,y)

#### Returns

sign of  $y$ , or -1 if  $P$  is point-at-infinity

### 8.14.2.10 ECP\_BLS381\_inf()

```
void ECP_BLS381_inf (
    ECP_BLS381 * P )
```

#### Parameters

<i>P</i>	ECP instance to be set to infinity
----------	------------------------------------

#### 8.14.2.11 ECP\_BLS381\_isinf()

```
int ECP_BLS381_isinf (
    ECP_BLS381 * P )
```

##### Parameters

<i>P</i>	ECP point to be tested
----------	------------------------

##### Returns

1 if infinity, else returns 0

#### 8.14.2.12 ECP\_BLS381\_mapit()

```
void ECP_BLS381_mapit (
    ECP_BLS381 * Q,
    octet * w )
```

##### Parameters

<i>Q</i>	ECP instance of correct order
<i>w</i>	OCTET byte array to be mapped

#### 8.14.2.13 ECP\_BLS381\_mul()

```
void ECP_BLS381_mul (
    ECP_BLS381 * P,
    BIG_384_58 b )
```

Uses Montgomery ladder for Montgomery curves, otherwise fixed sized windows.

##### Parameters

<i>P</i>	ECP instance, on exit =b*P
<i>b</i>	BIG number multiplier

#### 8.14.2.14 ECP\_BLS381\_mul2()

```
void ECP_BLS381_mul2 (
    ECP_BLS381 * P,
```

```

    ECP_BLS381 * Q,
    BIG_384_58 e,
    BIG_384_58 f )

```

**Parameters**

<i>P</i>	ECP instance, on exit = $e*P+f*Q$
<i>Q</i>	ECP instance
<i>e</i>	BIG number multiplier
<i>f</i>	BIG number multiplier

**8.14.2.15 ECP\_BLS381\_neg()**

```

void ECP_BLS381_neg (
    ECP_BLS381 * P )

```

**Parameters**

<i>P</i>	ECP instance, on exit = $-P$
----------	------------------------------

**8.14.2.16 ECP\_BLS381\_output()**

```

void ECP_BLS381_output (
    ECP_BLS381 * P )

```

**Parameters**

<i>P</i>	ECP instance to be printed
----------	----------------------------

**8.14.2.17 ECP\_BLS381\_outputxyz()**

```

void ECP_BLS381_outputxyz (
    ECP_BLS381 * P )

```

**Parameters**

<i>P</i>	ECP instance to be printed
----------	----------------------------

## 8.14.2.18 ECP\_BLS381\_pinmul()

```
void ECP_BLS381_pinmul (
    ECP_BLS381 * P,
    int i,
    int b )
```

## Parameters

<i>P</i>	ECP instance, on exit =i*P
<i>i</i>	small integer multiplier
<i>b</i>	maximum number of bits in multiplier

## 8.14.2.19 ECP\_BLS381\_rawoutput()

```
void ECP_BLS381_rawoutput (
    ECP_BLS381 * P )
```

## Parameters

<i>P</i>	ECP instance to be printed
----------	----------------------------

## 8.14.2.20 ECP\_BLS381\_rhs()

```
void ECP_BLS381_rhs (
    FP_BLS381 * r,
    FP_BLS381 * x )
```

Function f(x) depends on form of elliptic curve, Weierstrass, Edwards or Montgomery. Used internally.

## Parameters

<i>r</i>	BIG n-residue value of f(x)
<i>x</i>	BIG n-residue x

## 8.14.2.21 ECP\_BLS381\_set()

```
int ECP_BLS381_set (
    ECP_BLS381 * P,
    BIG_384_58 x,
    BIG_384_58 y )
```

Point P set to infinity if no such point on the curve.

## Parameters

<i>P</i>	ECP instance to be set (x,y)
<i>x</i>	BIG x coordinate of point
<i>y</i>	BIG y coordinate of point

## Returns

1 if point exists, else 0

## 8.14.2.22 ECP\_BLS381\_setx()

```
int ECP_BLS381_setx (
    ECP_BLS381 * P,
    BIG_384_58 x,
    int s )
```

Point P set to infinity if no such point on the curve. If x is on the curve then y is calculated from the curve equation. The correct y value (plus or minus) is selected given its sign s.

## Parameters

<i>P</i>	ECP instance to be set (x,[y])
<i>x</i>	BIG x coordinate of point
<i>s</i>	an integer representing the "sign" of y, in fact its least significant bit.

## 8.14.2.23 ECP\_BLS381\_sub()

```
void ECP_BLS381_sub (
    ECP_BLS381 * P,
    ECP_BLS381 * Q )
```

## Parameters

<i>P</i>	ECP instance, on exit =P-Q
<i>Q</i>	ECP instance to be subtracted from P

## 8.14.2.24 ECP\_BLS381\_toOctet()

```
void ECP_BLS381_toOctet (
    octet * S,
```

```
ECP_BLS381 * P,  
bool c )
```

## Parameters

<i>c</i>	compression required, true or false
<i>S</i>	output octet string
<i>P</i>	ECP instance to be converted to an octet string

### 8.14.3 Variable Documentation

#### 8.14.3.1 CURVE\_A\_BLS381

```
const int CURVE_A_BLS381
```

Elliptic curve A parameter

#### 8.14.3.2 CURVE\_B\_BLS381

```
const BIG\_384\_58 CURVE_B_BLS381
```

Elliptic curve B parameter

#### 8.14.3.3 CURVE\_B\_I\_BLS381

```
const int CURVE_B_I_BLS381
```

Elliptic curve B<sub>i</sub> parameter

#### 8.14.3.4 CURVE\_BB\_BLS381

```
const BIG\_384\_58 CURVE_BB_BLS381[4][4]
```

BN curve constant for GS decomposition

#### 8.14.3.5 CURVE\_Bnx\_BLS381

```
const BIG\_384\_58 CURVE_Bnx_BLS381
```

BN curve x parameter

#### 8.14.3.6 CURVE\_Cof\_BLS381

```
const BIG\_384\_58 CURVE_Cof_BLS381
```

Elliptic curve cofactor

#### 8.14.3.7 CURVE\_Cof\_I\_BLS381

```
const int CURVE_Cof_I_BLS381
```

Elliptic curve cofactor

#### 8.14.3.8 CURVE\_Cru\_BLS381

```
const BIG_384_58 CURVE_Cru_BLS381
```

BN curve Cube Root of Unity

#### 8.14.3.9 CURVE\_Gx\_BLS381

```
const BIG_384_58 CURVE_Gx_BLS381
```

x-coordinate of generator point in group G1

#### 8.14.3.10 CURVE\_Gy\_BLS381

```
const BIG_384_58 CURVE_Gy_BLS381
```

y-coordinate of generator point in group G1

#### 8.14.3.11 CURVE\_Order\_BLS381

```
const BIG_384_58 CURVE_Order_BLS381
```

Elliptic curve group order

#### 8.14.3.12 CURVE\_Pxa\_BLS381

```
const BIG_384_58 CURVE_Pxa_BLS381
```

real part of x-coordinate of generator point in group G2

#### 8.14.3.13 CURVE\_Pxaa\_BLS381

```
const BIG_384_58 CURVE_Pxaa_BLS381
```

real part of x-coordinate of generator point in group G2

#### 8.14.3.14 CURVE\_Pxaaa\_BLS381

```
const BIG_384_58 CURVE_Pxaaa_BLS381
```

real part of x-coordinate of generator point in group G2



**8.14.3.15 CURVE\_Pxaab\_BLS381**

```
const BIG_384_58 CURVE_Pxaab_BLS381
```

imaginary part of x-coordinate of generator point in group G2

**8.14.3.16 CURVE\_Pxab\_BLS381**

```
const BIG_384_58 CURVE_Pxab_BLS381
```

imaginary part of x-coordinate of generator point in group G2

**8.14.3.17 CURVE\_Pxaba\_BLS381**

```
const BIG_384_58 CURVE_Pxaba_BLS381
```

real part of x-coordinate of generator point in group G2

**8.14.3.18 CURVE\_Pxab\_b\_BLS381**

```
const BIG_384_58 CURVE_Pxab_b_BLS381
```

imaginary part of x-coordinate of generator point in group G2

**8.14.3.19 CURVE\_Pxb\_BLS381**

```
const BIG_384_58 CURVE_Pxb_BLS381
```

imaginary part of x-coordinate of generator point in group G2

**8.14.3.20 CURVE\_Pxba\_BLS381**

```
const BIG_384_58 CURVE_Pxba_BLS381
```

real part of x-coordinate of generator point in group G2

**8.14.3.21 CURVE\_Pxbaa\_BLS381**

```
const BIG_384_58 CURVE_Pxbaa_BLS381
```

real part of x-coordinate of generator point in group G2

**8.14.3.22 CURVE\_Pxbab\_BLS381**

```
const BIG_384_58 CURVE_Pxbab_BLS381
```

imaginary part of x-coordinate of generator point in group G2

**8.14.3.23 CURVE\_Pxbb\_BLS381**

```
const BIG_384_58 CURVE_Pxbb_BLS381
```

imaginary part of x-coordinate of generator point in group G2

**8.14.3.24 CURVE\_Pxbba\_BLS381**

```
const BIG_384_58 CURVE_Pxbba_BLS381
```

real part of x-coordinate of generator point in group G2

**8.14.3.25 CURVE\_Pxbbb\_BLS381**

```
const BIG_384_58 CURVE_Pxbbb_BLS381
```

imaginary part of x-coordinate of generator point in group G2

**8.14.3.26 CURVE\_Pya\_BLS381**

```
const BIG_384_58 CURVE_Pya_BLS381
```

real part of y-coordinate of generator point in group G2

**8.14.3.27 CURVE\_Pyaa\_BLS381**

```
const BIG_384_58 CURVE_Pyaa_BLS381
```

real part of y-coordinate of generator point in group G2

**8.14.3.28 CURVE\_Pyaaa\_BLS381**

```
const BIG_384_58 CURVE_Pyaaa_BLS381
```

real part of y-coordinate of generator point in group G2

**8.14.3.29 CURVE\_Pyaab\_BLS381**

```
const BIG_384_58 CURVE_Pyaab_BLS381
```

imaginary part of y-coordinate of generator point in group G2

**8.14.3.30 CURVE\_Pyab\_BLS381**

```
const BIG_384_58 CURVE_Pyab_BLS381
```

imaginary part of y-coordinate of generator point in group G2

**8.14.3.31 CURVE\_Pyaba\_BLS381**

```
const BIG_384_58 CURVE_Pyaba_BLS381
```

real part of y-coordinate of generator point in group G2

**8.14.3.32 CURVE\_Pyabb\_BLS381**

```
const BIG_384_58 CURVE_Pyabb_BLS381
```

imaginary part of y-coordinate of generator point in group G2

**8.14.3.33 CURVE\_Pyb\_BLS381**

```
const BIG_384_58 CURVE_Pyb_BLS381
```

imaginary part of y-coordinate of generator point in group G2

**8.14.3.34 CURVE\_Pyba\_BLS381**

```
const BIG_384_58 CURVE_Pyba_BLS381
```

real part of y-coordinate of generator point in group G2

**8.14.3.35 CURVE\_Pybaa\_BLS381**

```
const BIG_384_58 CURVE_Pybaa_BLS381
```

real part of y-coordinate of generator point in group G2

**8.14.3.36 CURVE\_Pybab\_BLS381**

```
const BIG_384_58 CURVE_Pybab_BLS381
```

imaginary part of y-coordinate of generator point in group G2

**8.14.3.37 CURVE\_Pybb\_BLS381**

```
const BIG_384_58 CURVE_Pybb_BLS381
```

imaginary part of y-coordinate of generator point in group G2

**8.14.3.38 CURVE\_Pybba\_BLS381**

```
const BIG_384_58 CURVE_Pybba_BLS381
```

real part of y-coordinate of generator point in group G2

#### 8.14.3.39 CURVE\_Pybbb\_BLS381

```
const BIG_384_58 CURVE_Pybbb_BLS381
```

imaginary part of y-coordinate of generator point in group G2

#### 8.14.3.40 CURVE\_SB\_BLS381

```
const BIG_384_58 CURVE_SB_BLS381[2][2]
```

BN curve constant for GLV decomposition

#### 8.14.3.41 CURVE\_W\_BLS381

```
const BIG_384_58 CURVE_W_BLS381[2]
```

BN curve constant for GLV decomposition

#### 8.14.3.42 CURVE\_WB\_BLS381

```
const BIG_384_58 CURVE_WB_BLS381[4]
```

BN curve constant for GS decomposition

#### 8.14.3.43 Fra\_BLS381

```
const BIG_384_58 Fra_BLS381
```

real part of BN curve Frobenius Constant

#### 8.14.3.44 Frb\_BLS381

```
const BIG_384_58 Frb_BLS381
```

imaginary part of BN curve Frobenius Constant

## 8.15 ff\_2048.h File Reference

FF Header File.

```
#include "big_1024_58.h"  
#include "config_ff_2048.h"
```

## Macros

- `#define HFLEN_2048 (FFLEN_2048/2)`
- `#define P_MBITS_2048 (MODBYTES_1024_58*8)`
- `#define P_TBITS_2048 (P_MBITS_2048%BASEBITS_1024_58)`
- `#define P_EXCESS_2048(a) (((a[NLEN_1024_58-1])>>(P_TBITS_2048))+1)`
- `#define P_FEXCESS_2048 ((chunk)1<<((BASEBITS_1024_58*NLEN_1024_58-P_MBITS_2048)-1))`

## Functions

- `void FF_2048_copy (BIG_1024_58 *x, BIG_1024_58 *y, int n)`  
*Copy one FF element of given length to another.*
- `void FF_2048_init (BIG_1024_58 *x, sign32 m, int n)`  
*Initialize an FF element of given length from a 32-bit integer m.*
- `void FF_2048_zero (BIG_1024_58 *x, int n)`  
*Set FF element of given size to zero.*
- `int FF_2048_iszilch (BIG_1024_58 *x, int n)`  
*Tests for FF element equal to zero.*
- `int FF_2048_parity (BIG_1024_58 *x)`  
*return parity of an FF, that is the least significant bit*
- `int FF_2048_lastbits (BIG_1024_58 *x, int m)`  
*return least significant m bits of an FF*
- `void FF_2048_one (BIG_1024_58 *x, int n)`  
*Set FF element of given size to unity.*
- `int FF_2048_comp (BIG_1024_58 *x, BIG_1024_58 *y, int n)`  
*Compares two FF numbers. Inputs must be normalised externally.*
- `void FF_2048_add (BIG_1024_58 *x, BIG_1024_58 *y, BIG_1024_58 *z, int n)`  
*addition of two FFs*
- `void FF_2048_sub (BIG_1024_58 *x, BIG_1024_58 *y, BIG_1024_58 *z, int n)`  
*subtraction of two FFs*
- `void FF_2048_inc (BIG_1024_58 *x, int m, int n)`  
*increment an FF by an integer, and normalise*
- `void FF_2048_dec (BIG_1024_58 *x, int m, int n)`  
*Decrement an FF by an integer, and normalise.*
- `void FF_2048_norm (BIG_1024_58 *x, int n)`  
*Normalises the components of an FF.*
- `void FF_2048_shl (BIG_1024_58 *x, int n)`  
*Shift left an FF by 1 bit.*
- `void FF_2048_shr (BIG_1024_58 *x, int n)`  
*Shift right an FF by 1 bit.*
- `void FF_2048_output (BIG_1024_58 *x, int n)`  
*Formats and outputs an FF to the console.*
- `void FF_2048_rawoutput (BIG_1024_58 *x, int n)`  
*Formats and outputs an FF to the console, in raw form.*
- `void FF_2048_toOctet (octet *S, BIG_1024_58 *x, int n)`  
*Formats and outputs an FF instance to an octet string.*
- `void FF_2048_fromOctet (BIG_1024_58 *x, octet *S, int n)`  
*Populates an FF instance from an octet string.*
- `void FF_2048_mul (BIG_1024_58 *x, BIG_1024_58 *y, BIG_1024_58 *z, int n)`  
*Multiplication of two FFs.*
- `void FF_2048_mod (BIG_1024_58 *x, BIG_1024_58 *p, int n)`

- Reduce FF mod a modulus.*

  - void `FF_2048_sqr` (`BIG_1024_58 *x`, `BIG_1024_58 *y`, int n)
- Square an FF.*

  - void `FF_2048_dmod` (`BIG_1024_58 *x`, `BIG_1024_58 *y`, `BIG_1024_58 *z`, int n)
- Reduces a double-length FF with respect to a given modulus.*

  - void `FF_2048_invmodp` (`BIG_1024_58 *x`, `BIG_1024_58 *y`, `BIG_1024_58 *z`, int n)
- Invert an FF mod a prime modulus.*

  - void `FF_2048_invmod2m` (`BIG_1024_58 U[]`, `BIG_1024_58 a[]`, int n)
- Invert an FF mod  $2^{(n \cdot \text{BIGBITS})}$*

  - void `FF_2048_random` (`BIG_1024_58 *x`, `csprng *R`, int n)
- Create an FF from a random number generator.*

  - void `FF_2048_randomnum` (`BIG_1024_58 *x`, `BIG_1024_58 *y`, `csprng *R`, int n)
- Create a random FF less than a given modulus from a random number generator.*

  - void `FF_2048_skpow` (`BIG_1024_58 *r`, `BIG_1024_58 *x`, `BIG_1024_58 *e`, `BIG_1024_58 *p`, int n, int en)
- Calculate  $r = x^e \bmod p$ , side channel resistant.*

  - void `FF_2048_skspow` (`BIG_1024_58 *r`, `BIG_1024_58 *x`, `BIG_1024_58 e`, `BIG_1024_58 *p`, int n)
- Calculate  $r = x^e \bmod p$ , side channel resistant.*

  - void `FF_2048_skpow2` (`BIG_1024_58 *r`, `BIG_1024_58 *x`, `BIG_1024_58 *e`, `BIG_1024_58 *y`, `BIG_1024_58 *f`, `BIG_1024_58 *p`, int n, int en)
- Calculate  $r = x^e \cdot y^f \bmod p$  for big e and f, side channel resistant.*

  - void `FF_2048_power` (`BIG_1024_58 *r`, `BIG_1024_58 *x`, int e, `BIG_1024_58 *p`, int n)
- Calculate  $r = x^e \bmod p$ .*

  - void `FF_2048_pow` (`BIG_1024_58 *r`, `BIG_1024_58 *x`, `BIG_1024_58 *e`, `BIG_1024_58 *p`, int n)
- Calculate  $r = x^e \bmod p$ .*

  - void `FF_2048_pow2` (`BIG_1024_58 *r`, `BIG_1024_58 *x`, `BIG_1024_58 e`, `BIG_1024_58 *y`, `BIG_1024_58 f`, `BIG_1024_58 *m`, int n)
- Calculate  $r = x^e \cdot y^f \bmod m$ .*

  - int `FF_2048_cfactor` (`BIG_1024_58 *x`, `sign32 s`, int n)
- Test if an FF has factor in common with integer s.*

  - int `FF_2048_prime` (`BIG_1024_58 *x`, `csprng *R`, int n)
- Test if an FF is prime.*

  - void `FF_2048 crt` (`BIG_1024_58 *r`, `BIG_1024_58 *rp`, `BIG_1024_58 *rq`, `BIG_1024_58 *p`, `BIG_1024_58 *q`, int n)
- Combine rp and rq using the Chinese Remainder Theorem.*

### 8.15.1 Detailed Description

#### Author

Mike Scott

### 8.15.2 Macro Definition Documentation

#### 8.15.2.1 HFLEN\_2048

```
#define HFLEN_2048 (FFLEN_2048/2)
```

Useful for half-size RSA private key operations

### 8.15.2.2 P\_EXCESS\_2048

```
#define P_EXCESS_2048(
    a ) (((a[NLEN_1024_58-1])>>(P_TBITS_2048))+1)
```

TODO

### 8.15.2.3 P\_FEXCESS\_2048

```
#define P_FEXCESS_2048 ((chunk)1<<(BASEBITS_1024_58*NLEN_1024_58-P_MBITS_2048-1))
```

TODO

### 8.15.2.4 P\_MBITS\_2048

```
#define P_MBITS_2048 (MODBYTES_1024_58*8)
```

Number of bits in modulus

### 8.15.2.5 P\_TBITS\_2048

```
#define P_TBITS_2048 (P_MBITS_2048%BASEBITS_1024_58)
```

TODO

## 8.15.3 Function Documentation

### 8.15.3.1 FF\_2048\_add()

```
void FF_2048_add (
    BIG_1024_58 * x,
    BIG_1024_58 * y,
    BIG_1024_58 * z,
    int n )
```

#### Parameters

<i>x</i>	FF instance, on exit = y+z
<i>y</i>	FF instance
<i>z</i>	FF instance
<i>n</i>	size of FF in BIGs

### 8.15.3.2 FF\_2048\_cfactor()

```
int FF_2048_cfactor (
    BIG_1024_58 * x,
    sign32 s,
    int n )
```

#### Parameters

<i>x</i>	FF instance to be tested
<i>s</i>	the supplied integer
<i>n</i>	size of FF in BIGs

#### Returns

1 if  $\text{gcd}(x,s) \neq 1$ , else return 0

### 8.15.3.3 FF\_2048\_comp()

```
int FF_2048_comp (
    BIG_1024_58 * x,
    BIG_1024_58 * y,
    int n )
```

#### Parameters

<i>x</i>	first FF number to be compared
<i>y</i>	second FF number to be compared
<i>n</i>	size of FF in BIGs

#### Returns

-1 is  $x < y$ , 0 if  $x = y$ , 1 if  $x > y$

### 8.15.3.4 FF\_2048\_copy()

```
void FF_2048_copy (
    BIG_1024_58 * x,
    BIG_1024_58 * y,
    int n )
```

#### Parameters

<i>x</i>	FF instance to be copied to, on exit = y
<i>y</i>	FF instance to be copied from
<i>n</i>	size of FF in BIGs



## 8.15.3.5 FF\_2048\_crt()

```
void FF_2048_crt (
    BIG_1024_58 * r,
    BIG_1024_58 * rp,
    BIG_1024_58 * rq,
    BIG_1024_58 * p,
    BIG_1024_58 * q,
    int n )
```

## Parameters

<i>r</i>	FF instance, on exit the solution of the system
<i>rp</i>	FF instance, solution modulo p
<i>rq</i>	FF instance, solution modulo q
<i>p</i>	FF instance, MUST be coprime with q
<i>q</i>	FF instance, MUST be coprime with p
<i>n</i>	size of p and q in BIGs

## 8.15.3.6 FF\_2048\_dec()

```
void FF_2048_dec (
    BIG_1024_58 * x,
    int m,
    int n )
```

## Parameters

<i>x</i>	FF instance, on exit = x-m
<i>m</i>	an integer to be subtracted from x
<i>n</i>	size of FF in BIGs

## 8.15.3.7 FF\_2048\_dmod()

```
void FF_2048_dmod (
    BIG_1024_58 * x,
    BIG_1024_58 * y,
    BIG_1024_58 * z,
    int n )
```

This is slow

## Parameters

<i>x</i>	FF instance, on exit = $y \bmod z$
<i>y</i>	FF instance, of double length $2*n$
<i>z</i>	FF modulus
<i>n</i>	size of FF in BIGs

## 8.15.3.8 FF\_2048\_fromOctet()

```
void FF_2048_fromOctet (
    BIG_1024_58 * x,
    octet * S,
    int n )
```

Creates FF from big-endian base 256 form.

## Parameters

<i>x</i>	FF instance to be created from an octet string
<i>S</i>	input octet string
<i>n</i>	size of FF in BIGs

## 8.15.3.9 FF\_2048\_inc()

```
void FF_2048_inc (
    BIG_1024_58 * x,
    int m,
    int n )
```

## Parameters

<i>x</i>	FF instance, on exit = $x+m$
<i>m</i>	an integer to be added to $x$
<i>n</i>	size of FF in BIGs

## 8.15.3.10 FF\_2048\_init()

```
void FF_2048_init (
    BIG_1024_58 * x,
    sign32 m,
    int n )
```

## Parameters

<i>x</i>	FF instance to be copied to, on exit = m
<i>m</i>	integer
<i>n</i>	size of FF in BIGs

## 8.15.3.11 FF\_2048\_invmod2m()

```
void FF_2048_invmod2m (
    BIG_1024_58 u[],
    BIG_1024_58 a[],
    int n )
```

## Parameters

<i>U</i>	FF instance, on exit $1/a \bmod 2^{(n \cdot \text{BIGBITS})}$
<i>a</i>	FF instance
<i>n</i>	size of FF in BIGs

## 8.15.3.12 FF\_2048\_invmodp()

```
void FF_2048_invmodp (
    BIG_1024_58 * x,
    BIG_1024_58 * y,
    BIG_1024_58 * z,
    int n )
```

## Parameters

<i>x</i>	FF instance, on exit = $1/y \bmod z$
<i>y</i>	FF instance
<i>z</i>	FF prime modulus
<i>n</i>	size of FF in BIGs

## 8.15.3.13 FF\_2048\_iszilch()

```
int FF_2048_iszilch (
    BIG_1024_58 * x,
    int n )
```

**Parameters**

$x$	FF number to be tested
$n$	size of FF in BIGs

**Returns**

1 if zero, else returns 0

**8.15.3.14 FF\_2048\_lastbits()**

```
int FF_2048_lastbits (
    BIG_1024_58 * x,
    int m )
```

**Parameters**

$x$	FF number
$m$	number of bits to return. Assumed to be less than BASEBITS.

**Returns**

least significant  $n$  bits as an integer

**8.15.3.15 FF\_2048\_mod()**

```
void FF_2048_mod (
    BIG_1024_58 * x,
    BIG_1024_58 * p,
    int n )
```

This is slow

**Parameters**

$x$	FF instance to be reduced mod $p$ - on exit = $x \bmod p$
$p$	FF modulus
$n$	size of FF in BIGs

**8.15.3.16 FF\_2048\_mul()**

```
void FF_2048_mul (
    BIG_1024_58 * x,
```

```
BIG_1024_58 * y,  
BIG_1024_58 * z,  
int n )
```

Uses Karatsuba method internally

#### Parameters

<i>x</i>	FF instance, on exit = $y*z$
<i>y</i>	FF instance
<i>z</i>	FF instance
<i>n</i>	size of FF in BIGs

#### 8.15.3.17 FF\_2048\_norm()

```
void FF_2048_norm (  
    BIG_1024_58 * x,  
    int n )
```

#### Parameters

<i>x</i>	FF instance to be normalised
<i>n</i>	size of FF in BIGs

#### 8.15.3.18 FF\_2048\_one()

```
void FF_2048_one (  
    BIG_1024_58 * x,  
    int n )
```

#### Parameters

<i>x</i>	FF instance to be set to unity
<i>n</i>	size of FF in BIGs

#### 8.15.3.19 FF\_2048\_output()

```
void FF_2048_output (  
    BIG_1024_58 * x,  
    int n )
```

## Parameters

$x$	FF instance to be printed
$n$	size of FF in BIGs

## 8.15.3.20 FF\_2048\_parity()

```
int FF_2048_parity (
    BIG_1024_58 * x )
```

## Parameters

$x$	FF number
-----	-----------

## Returns

0 or 1

## 8.15.3.21 FF\_2048\_pow()

```
void FF_2048_pow (
    BIG_1024_58 * r,
    BIG_1024_58 * x,
    BIG_1024_58 * e,
    BIG_1024_58 * p,
    int n )
```

## Parameters

$r$	FF instance, on exit = $x^e \bmod p$
$x$	FF instance
$e$	FF exponent
$p$	FF modulus
$n$	size of FF in BIGs

## 8.15.3.22 FF\_2048\_pow2()

```
void FF_2048_pow2 (
    BIG_1024_58 * r,
    BIG_1024_58 * x,
    BIG_1024_58 e,
```

```

    BIG_1024_58 * y,
    BIG_1024_58 f,
    BIG_1024_58 * m,
    int n )

```

**Parameters**

<i>r</i>	FF instance, on exit = $x^e y^f \bmod p$
<i>x</i>	FF instance
<i>e</i>	BIG exponent
<i>y</i>	FF instance
<i>f</i>	BIG exponent
<i>m</i>	FF modulus
<i>n</i>	size of FF in BIGs

**8.15.3.23 FF\_2048\_power()**

```

void FF_2048_power (
    BIG_1024_58 * r,
    BIG_1024_58 * x,
    int e,
    BIG_1024_58 * p,
    int n )

```

For very short integer exponent

**Parameters**

<i>r</i>	FF instance, on exit = $x^e \bmod p$
<i>x</i>	FF instance
<i>e</i>	integer exponent
<i>p</i>	FF modulus
<i>n</i>	size of FF in BIGs

**8.15.3.24 FF\_2048\_prime()**

```

int FF_2048_prime (
    BIG_1024_58 * x,
    csprng * R,
    int n )

```

Uses Miller-Rabin Method

**Parameters**

<i>x</i>	FF instance to be tested
<i>R</i>	an instance of a Cryptographically Secure Random Number Generator
<i>n</i>	size of FF in BIGs

**Returns**

1 if  $x$  is (almost certainly) prime, else return 0

**8.15.3.25 FF\_2048\_random()**

```
void FF_2048_random (
    BIG_1024_58 * x,
    csprng * R,
    int n )
```

**Parameters**

$x$	FF instance, on exit $x$ is a random number of length $n$ BIGs with most significant bit a 1
$R$	an instance of a Cryptographically Secure Random Number Generator
$n$	size of FF in BIGs

**8.15.3.26 FF\_2048\_randomnum()**

```
void FF_2048_randomnum (
    BIG_1024_58 * x,
    BIG_1024_58 * y,
    csprng * R,
    int n )
```

**Parameters**

$x$	FF instance, on exit $x$ is a random number $< y$
$y$	FF instance, the modulus
$R$	an instance of a Cryptographically Secure Random Number Generator
$n$	size of FF in BIGs

**8.15.3.27 FF\_2048\_rawoutput()**

```
void FF_2048_rawoutput (
    BIG_1024_58 * x,
    int n )
```

**Parameters**

$x$	FF instance to be printed
$n$	size of FF in BIGs



## 8.15.3.28 FF\_2048\_shl()

```
void FF_2048_shl (
    BIG_1024_58 * x,
    int n )
```

## Parameters

<i>x</i>	FF instance to be shifted left
<i>n</i>	size of FF in BIGs

## 8.15.3.29 FF\_2048\_shr()

```
void FF_2048_shr (
    BIG_1024_58 * x,
    int n )
```

## Parameters

<i>x</i>	FF instance to be shifted right
<i>n</i>	size of FF in BIGs

## 8.15.3.30 FF\_2048\_skpow()

```
void FF_2048_skpow (
    BIG_1024_58 * r,
    BIG_1024_58 * x,
    BIG_1024_58 * e,
    BIG_1024_58 * p,
    int n,
    int en )
```

## Parameters

<i>r</i>	FF instance, on exit = $x^e \text{ mod } p$
<i>x</i>	FF instance
<i>e</i>	FF exponent
<i>p</i>	FF modulus
<i>n</i>	size of FF in BIGs
<i>en</i>	size of the exponent in BIGs

## 8.15.3.31 FF\_2048\_skpow2()

```
void FF_2048_skpow2 (
    BIG_1024_58 * r,
    BIG_1024_58 * x,
    BIG_1024_58 * e,
    BIG_1024_58 * y,
    BIG_1024_58 * f,
    BIG_1024_58 * p,
    int n,
    int en )
```

## Parameters

<i>r</i>	FF instance, on exit = $x^e.y^f \bmod p$
<i>x</i>	FF instance
<i>e</i>	FF exponent
<i>y</i>	FF instance
<i>f</i>	FF exponent
<i>p</i>	FF modulus
<i>n</i>	size of FF in BIGs
<i>en</i>	size of the exponent in BIGs

## 8.15.3.32 FF\_2048\_skspow()

```
void FF_2048_skspow (
    BIG_1024_58 * r,
    BIG_1024_58 * x,
    BIG_1024_58 e,
    BIG_1024_58 * p,
    int n )
```

For short BIG exponent

## Parameters

<i>r</i>	FF instance, on exit = $x^e \bmod p$
<i>x</i>	FF instance
<i>e</i>	BIG exponent
<i>p</i>	FF modulus
<i>n</i>	size of FF in BIGs

## 8.15.3.33 FF\_2048\_sqr()

```
void FF_2048_sqr (
    BIG_1024_58 * x,
```

```

    BIG_1024_58 * y,
    int n )

```

Uses Karatsuba method internally

#### Parameters

<i>x</i>	FF instance, on exit = $y^2$
<i>y</i>	FF instance to be squared
<i>n</i>	size of FF in BIGs

#### 8.15.3.34 FF\_2048\_sub()

```

void FF_2048_sub (
    BIG_1024_58 * x,
    BIG_1024_58 * y,
    BIG_1024_58 * z,
    int n )

```

#### Parameters

<i>x</i>	FF instance, on exit = $y-z$
<i>y</i>	FF instance
<i>z</i>	FF instance
<i>n</i>	size of FF in BIGs

#### 8.15.3.35 FF\_2048\_toOctet()

```

void FF_2048_toOctet (
    octet * S,
    BIG_1024_58 * x,
    int n )

```

Converts an FF to big-endian base 256 form.

#### Parameters

<i>S</i>	output octet string
<i>x</i>	FF instance to be converted to an octet string
<i>n</i>	size of FF in BIGs

### 8.15.3.36 FF\_2048\_zero()

```
void FF_2048_zero (
    BIG_1024_58 * x,
    int n )
```

#### Parameters

<i>x</i>	FF instance to be set to zero
<i>n</i>	size of FF in BIGs

## 8.16 ff\_4096.h File Reference

FF Header File.

```
#include "big_512_60.h"
#include "config_ff_4096.h"
```

### Macros

- `#define HFLEN_4096 (FFLEN_4096/2)`
- `#define P_MBITS_4096 (MODBYTES_512_60*8)`
- `#define P_TBITS_4096 (P_MBITS_4096%BASEBITS_512_60)`
- `#define P_EXCESS_4096(a) (((a[NLEN_512_60-1])>>(P_TBITS_4096))+1)`
- `#define P_FEXCESS_4096 ((chunk)1<<<(BASEBITS_512_60*NLEN_512_60-P_MBITS_4096-1))`

### Functions

- void `FF_4096_copy` (`BIG_512_60 *x`, `BIG_512_60 *y`, int `n`)  
*Copy one FF element of given length to another.*
- void `FF_4096_init` (`BIG_512_60 *x`, `sign32 m`, int `n`)  
*Initialize an FF element of given length from a 32-bit integer m.*
- void `FF_4096_zero` (`BIG_512_60 *x`, int `n`)  
*Set FF element of given size to zero.*
- int `FF_4096_iszilch` (`BIG_512_60 *x`, int `n`)  
*Tests for FF element equal to zero.*
- int `FF_4096_parity` (`BIG_512_60 *x`)  
*return parity of an FF, that is the least significant bit*
- int `FF_4096_lastbits` (`BIG_512_60 *x`, int `m`)  
*return least significant m bits of an FF*
- void `FF_4096_one` (`BIG_512_60 *x`, int `n`)  
*Set FF element of given size to unity.*
- int `FF_4096_comp` (`BIG_512_60 *x`, `BIG_512_60 *y`, int `n`)  
*Compares two FF numbers. Inputs must be normalised externally.*
- void `FF_4096_add` (`BIG_512_60 *x`, `BIG_512_60 *y`, `BIG_512_60 *z`, int `n`)  
*addition of two FFs*
- void `FF_4096_sub` (`BIG_512_60 *x`, `BIG_512_60 *y`, `BIG_512_60 *z`, int `n`)

- subtraction of two FFs*

  - void `FF_4096_inc` (`BIG_512_60 *x`, `int m`, `int n`)  
*increment an FF by an integer, and normalise*
  - void `FF_4096_dec` (`BIG_512_60 *x`, `int m`, `int n`)  
*Decrement an FF by an integer, and normalise.*
  - void `FF_4096_norm` (`BIG_512_60 *x`, `int n`)  
*Normalises the components of an FF.*
  - void `FF_4096_shl` (`BIG_512_60 *x`, `int n`)  
*Shift left an FF by 1 bit.*
  - void `FF_4096_shr` (`BIG_512_60 *x`, `int n`)  
*Shift right an FF by 1 bit.*
  - void `FF_4096_output` (`BIG_512_60 *x`, `int n`)  
*Formats and outputs an FF to the console.*
  - void `FF_4096_rawoutput` (`BIG_512_60 *x`, `int n`)  
*Formats and outputs an FF to the console, in raw form.*
  - void `FF_4096_toOctet` (`octet *S`, `BIG_512_60 *x`, `int n`)  
*Formats and outputs an FF instance to an octet string.*
  - void `FF_4096_fromOctet` (`BIG_512_60 *x`, `octet *S`, `int n`)  
*Populates an FF instance from an octet string.*
  - void `FF_4096_mul` (`BIG_512_60 *x`, `BIG_512_60 *y`, `BIG_512_60 *z`, `int n`)  
*Multiplication of two FFs.*
  - void `FF_4096_mod` (`BIG_512_60 *x`, `BIG_512_60 *p`, `int n`)  
*Reduce FF mod a modulus.*
  - void `FF_4096_sqr` (`BIG_512_60 *x`, `BIG_512_60 *y`, `int n`)  
*Square an FF.*
  - void `FF_4096_dmod` (`BIG_512_60 *x`, `BIG_512_60 *y`, `BIG_512_60 *z`, `int n`)  
*Reduces a double-length FF with respect to a given modulus.*
  - void `FF_4096_invmodp` (`BIG_512_60 *x`, `BIG_512_60 *y`, `BIG_512_60 *z`, `int n`)  
*Invert an FF mod a prime modulus.*
  - void `FF_4096_invmod2m` (`BIG_512_60 U[]`, `BIG_512_60 a[]`, `int n`)  
*Invert an FF mod  $2^n$  ( $n \cdot \text{BIGBITS}$ )*
  - void `FF_4096_random` (`BIG_512_60 *x`, `csprng *R`, `int n`)  
*Create an FF from a random number generator.*
  - void `FF_4096_randomnum` (`BIG_512_60 *x`, `BIG_512_60 *y`, `csprng *R`, `int n`)  
*Create a random FF less than a given modulus from a random number generator.*
  - void `FF_4096_skpow` (`BIG_512_60 *r`, `BIG_512_60 *x`, `BIG_512_60 *e`, `BIG_512_60 *p`, `int n`, `int en`)  
*Calculate  $r = x^e \pmod p$ , side channel resistant.*
  - void `FF_4096_skspow` (`BIG_512_60 *r`, `BIG_512_60 *x`, `BIG_512_60 e`, `BIG_512_60 *p`, `int n`)  
*Calculate  $r = x^e \pmod p$ , side channel resistant.*
  - void `FF_4096_skpow2` (`BIG_512_60 *r`, `BIG_512_60 *x`, `BIG_512_60 *e`, `BIG_512_60 *y`, `BIG_512_60 *f`, `BIG_512_60 *p`, `int n`, `int en`)  
*Calculate  $r = x^e \cdot y^f \pmod p$  for big  $e$  and  $f$ , side channel resistant.*
  - void `FF_4096_power` (`BIG_512_60 *r`, `BIG_512_60 *x`, `int e`, `BIG_512_60 *p`, `int n`)  
*Calculate  $r = x^e \pmod p$ .*
  - void `FF_4096_pow` (`BIG_512_60 *r`, `BIG_512_60 *x`, `BIG_512_60 *e`, `BIG_512_60 *p`, `int n`)  
*Calculate  $r = x^e \pmod p$ .*
  - void `FF_4096_pow2` (`BIG_512_60 *r`, `BIG_512_60 *x`, `BIG_512_60 e`, `BIG_512_60 *y`, `BIG_512_60 f`, `BIG_512_60 *m`, `int n`)  
*Calculate  $r = x^e \cdot y^f \pmod m$ .*
  - int `FF_4096_cfactor` (`BIG_512_60 *x`, `sign32 s`, `int n`)  
*Test if an FF has factor in common with integer  $s$ .*

- int `FF_4096_prime` (`BIG_512_60 *x`, `csprng *R`, int `n`)  
*Test if an FF is prime.*
- void `FF_4096_crt` (`BIG_512_60 *r`, `BIG_512_60 *rp`, `BIG_512_60 *rq`, `BIG_512_60 *p`, `BIG_512_60 *q`, int `n`)  
*Combine `rp` and `rq` using the Chinese Remainder Theorem.*

### 8.16.1 Detailed Description

#### Author

Mike Scott

### 8.16.2 Macro Definition Documentation

#### 8.16.2.1 HFLEN\_4096

```
#define HFLEN_4096 (FFLEN_4096/2)
```

Useful for half-size RSA private key operations

#### 8.16.2.2 P\_EXCESS\_4096

```
#define P_EXCESS_4096(  
    a ) (((a[NLEN_512_60-1]) >> (P_TBITS_4096)) + 1)
```

TODO

#### 8.16.2.3 P\_FEXCESS\_4096

```
#define P_FEXCESS_4096 ((chunk) 1 <<< (BASEBITS_512_60 * NLEN_512_60 - P_MBITS_4096 - 1))
```

TODO

#### 8.16.2.4 P\_MBITS\_4096

```
#define P_MBITS_4096 (MODBYTES_512_60 * 8)
```

Number of bits in modulus

#### 8.16.2.5 P\_TBITS\_4096

```
#define P_TBITS_4096 (P_MBITS_4096 % BASEBITS_512_60)
```

TODO

### 8.16.3 Function Documentation

#### 8.16.3.1 FF\_4096\_add()

```
void FF_4096_add (
    BIG_512_60 * x,
    BIG_512_60 * y,
    BIG_512_60 * z,
    int n )
```

##### Parameters

<i>x</i>	FF instance, on exit = y+z
<i>y</i>	FF instance
<i>z</i>	FF instance
<i>n</i>	size of FF in BIGs

#### 8.16.3.2 FF\_4096\_cfactor()

```
int FF_4096_cfactor (
    BIG_512_60 * x,
    sign32 s,
    int n )
```

##### Parameters

<i>x</i>	FF instance to be tested
<i>s</i>	the supplied integer
<i>n</i>	size of FF in BIGs

##### Returns

1 if  $\text{gcd}(x,s) \neq 1$ , else return 0

#### 8.16.3.3 FF\_4096\_comp()

```
int FF_4096_comp (
    BIG_512_60 * x,
    BIG_512_60 * y,
    int n )
```

**Parameters**

<i>x</i>	first FF number to be compared
<i>y</i>	second FF number to be compared
<i>n</i>	size of FF in BIGs

**Returns**

-1 is  $x < y$ , 0 if  $x = y$ , 1 if  $x > y$

**8.16.3.4 FF\_4096\_copy()**

```
void FF_4096_copy (
    BIG_512_60 * x,
    BIG_512_60 * y,
    int n )
```

**Parameters**

<i>x</i>	FF instance to be copied to, on exit = y
<i>y</i>	FF instance to be copied from
<i>n</i>	size of FF in BIGs

**8.16.3.5 FF\_4096\_crt()**

```
void FF_4096_crt (
    BIG_512_60 * r,
    BIG_512_60 * rp,
    BIG_512_60 * rq,
    BIG_512_60 * p,
    BIG_512_60 * q,
    int n )
```

**Parameters**

<i>r</i>	FF instance, on exit the solution of the system
<i>rp</i>	FF instance, solution modulo p
<i>rq</i>	FF instance, solution modulo q
<i>p</i>	FF instance, MUST be coprime with q
<i>q</i>	FF instance, MUST be coprime with p
<i>n</i>	size of p and q in BIGs



## 8.16.3.6 FF\_4096\_dec()

```
void FF_4096_dec (
    BIG_512_60 * x,
    int m,
    int n )
```

## Parameters

<i>x</i>	FF instance, on exit = x-m
<i>m</i>	an integer to be subtracted from x
<i>n</i>	size of FF in BIGs

## 8.16.3.7 FF\_4096\_dmod()

```
void FF_4096_dmod (
    BIG_512_60 * x,
    BIG_512_60 * y,
    BIG_512_60 * z,
    int n )
```

This is slow

## Parameters

<i>x</i>	FF instance, on exit = y mod z
<i>y</i>	FF instance, of double length 2*n
<i>z</i>	FF modulus
<i>n</i>	size of FF in BIGs

## 8.16.3.8 FF\_4096\_fromOctet()

```
void FF_4096_fromOctet (
    BIG_512_60 * x,
    octet * S,
    int n )
```

Creates FF from big-endian base 256 form.

## Parameters

<i>x</i>	FF instance to be created from an octet string
<i>S</i>	input octet string
<i>n</i>	size of FF in BIGs

## 8.16.3.9 FF\_4096\_inc()

```
void FF_4096_inc (
    BIG_512_60 * x,
    int m,
    int n )
```

## Parameters

<i>x</i>	FF instance, on exit = x+m
<i>m</i>	an integer to be added to x
<i>n</i>	size of FF in BIGs

## 8.16.3.10 FF\_4096\_init()

```
void FF_4096_init (
    BIG_512_60 * x,
    sign32 m,
    int n )
```

## Parameters

<i>x</i>	FF instance to be copied to, on exit = m
<i>m</i>	integer
<i>n</i>	size of FF in BIGs

## 8.16.3.11 FF\_4096\_invmod2m()

```
void FF_4096_invmod2m (
    BIG_512_60 U[],
    BIG_512_60 a[],
    int n )
```

## Parameters

<i>U</i>	FF instance, on exit $1/a \pmod{2^{(n \cdot \text{BIGBITS})}}$
<i>a</i>	FF instance
<i>n</i>	size of FF in BIGs

## 8.16.3.12 FF\_4096\_invmodp()

```
void FF_4096_invmodp (
    BIG_512_60 * x,
    BIG_512_60 * y,
    BIG_512_60 * z,
    int n )
```

## Parameters

<i>x</i>	FF instance, on exit = 1/y mod z
<i>y</i>	FF instance
<i>z</i>	FF prime modulus
<i>n</i>	size of FF in BIGs

## 8.16.3.13 FF\_4096\_iszilch()

```
int FF_4096_iszilch (
    BIG_512_60 * x,
    int n )
```

## Parameters

<i>x</i>	FF number to be tested
<i>n</i>	size of FF in BIGs

## Returns

1 if zero, else returns 0

## 8.16.3.14 FF\_4096\_lastbits()

```
int FF_4096_lastbits (
    BIG_512_60 * x,
    int m )
```

## Parameters

<i>x</i>	FF number
<i>m</i>	number of bits to return. Assumed to be less than BASEBITS.

## Returns

least significant n bits as an integer

**8.16.3.15 FF\_4096\_mod()**

```
void FF_4096_mod (
    BIG_512_60 * x,
    BIG_512_60 * p,
    int n )
```

This is slow

**Parameters**

$x$	FF instance to be reduced mod $p$ - on exit = $x \bmod p$
$p$	FF modulus
$n$	size of FF in BIGs

**8.16.3.16 FF\_4096\_mul()**

```
void FF_4096_mul (
    BIG_512_60 * x,
    BIG_512_60 * y,
    BIG_512_60 * z,
    int n )
```

Uses Karatsuba method internally

**Parameters**

$x$	FF instance, on exit = $y*z$
$y$	FF instance
$z$	FF instance
$n$	size of FF in BIGs

**8.16.3.17 FF\_4096\_norm()**

```
void FF_4096_norm (
    BIG_512_60 * x,
    int n )
```

**Parameters**

$x$	FF instance to be normalised
$n$	size of FF in BIGs

### 8.16.3.18 FF\_4096\_one()

```
void FF_4096_one (
    BIG_512_60 * x,
    int n )
```

#### Parameters

<i>x</i>	FF instance to be set to unity
<i>n</i>	size of FF in BIGs

### 8.16.3.19 FF\_4096\_output()

```
void FF_4096_output (
    BIG_512_60 * x,
    int n )
```

#### Parameters

<i>x</i>	FF instance to be printed
<i>n</i>	size of FF in BIGs

### 8.16.3.20 FF\_4096\_parity()

```
int FF_4096_parity (
    BIG_512_60 * x )
```

#### Parameters

<i>x</i>	FF number
----------	-----------

#### Returns

0 or 1

### 8.16.3.21 FF\_4096\_pow()

```
void FF_4096_pow (
    BIG_512_60 * r,
    BIG_512_60 * x,
    BIG_512_60 * e,
    BIG_512_60 * p,
    int n )
```

## Parameters

<i>r</i>	FF instance, on exit = $x^e \bmod p$
<i>x</i>	FF instance
<i>e</i>	FF exponent
<i>p</i>	FF modulus
<i>n</i>	size of FF in BIGs

## 8.16.3.22 FF\_4096\_pow2()

```
void FF_4096_pow2 (
    BIG_512_60 * r,
    BIG_512_60 * x,
    BIG_512_60 e,
    BIG_512_60 * y,
    BIG_512_60 f,
    BIG_512_60 * m,
    int n )
```

## Parameters

<i>r</i>	FF instance, on exit = $x^e.y^f \bmod p$
<i>x</i>	FF instance
<i>e</i>	BIG exponent
<i>y</i>	FF instance
<i>f</i>	BIG exponent
<i>m</i>	FF modulus
<i>n</i>	size of FF in BIGs

## 8.16.3.23 FF\_4096\_power()

```
void FF_4096_power (
    BIG_512_60 * r,
    BIG_512_60 * x,
    int e,
    BIG_512_60 * p,
    int n )
```

For very short integer exponent

## Parameters

<i>r</i>	FF instance, on exit = $x^e \bmod p$
<i>x</i>	FF instance
<i>e</i>	integer exponent
<i>p</i>	FF modulus
<i>n</i>	size of FF in BIGs

## 8.16.3.24 FF\_4096\_prime()

```
int FF_4096_prime (
    BIG_512_60 * x,
    csprng * R,
    int n )
```

Uses Miller-Rabin Method

## Parameters

<i>x</i>	FF instance to be tested
<i>R</i>	an instance of a Cryptographically Secure Random Number Generator
<i>n</i>	size of FF in BIGs

## Returns

1 if *x* is (almost certainly) prime, else return 0

## 8.16.3.25 FF\_4096\_random()

```
void FF_4096_random (
    BIG_512_60 * x,
    csprng * R,
    int n )
```

## Parameters

<i>x</i>	FF instance, on exit <i>x</i> is a random number of length <i>n</i> BIGs with most significant bit a 1
<i>R</i>	an instance of a Cryptographically Secure Random Number Generator
<i>n</i>	size of FF in BIGs

## 8.16.3.26 FF\_4096\_randomnum()

```
void FF_4096_randomnum (
    BIG_512_60 * x,
    BIG_512_60 * y,
    csprng * R,
    int n )
```

## Parameters

<i>x</i>	FF instance, on exit <i>x</i> is a random number < <i>y</i>
----------	---

## Parameters

<i>y</i>	FF instance, the modulus
<i>R</i>	an instance of a Cryptographically Secure Random Number Generator
<i>n</i>	size of FF in BIGs

## 8.16.3.27 FF\_4096\_rawoutput()

```
void FF_4096_rawoutput (
    BIG_512_60 * x,
    int n )
```

## Parameters

<i>x</i>	FF instance to be printed
<i>n</i>	size of FF in BIGs

## 8.16.3.28 FF\_4096\_shl()

```
void FF_4096_shl (
    BIG_512_60 * x,
    int n )
```

## Parameters

<i>x</i>	FF instance to be shifted left
<i>n</i>	size of FF in BIGs

## 8.16.3.29 FF\_4096\_shr()

```
void FF_4096_shr (
    BIG_512_60 * x,
    int n )
```

## Parameters

<i>x</i>	FF instance to be shifted right
<i>n</i>	size of FF in BIGs



## 8.16.3.30 FF\_4096\_skpow()

```
void FF_4096_skpow (
    BIG_512_60 * r,
    BIG_512_60 * x,
    BIG_512_60 * e,
    BIG_512_60 * p,
    int n,
    int en )
```

## Parameters

<i>r</i>	FF instance, on exit = $x^e \bmod p$
<i>x</i>	FF instance
<i>e</i>	FF exponent
<i>p</i>	FF modulus
<i>n</i>	size of FF in BIGs
<i>en</i>	size of the exponent in BIGs

## 8.16.3.31 FF\_4096\_skpow2()

```
void FF_4096_skpow2 (
    BIG_512_60 * r,
    BIG_512_60 * x,
    BIG_512_60 * e,
    BIG_512_60 * y,
    BIG_512_60 * f,
    BIG_512_60 * p,
    int n,
    int en )
```

## Parameters

<i>r</i>	FF instance, on exit = $x^e.y^f \bmod p$
<i>x</i>	FF instance
<i>e</i>	FF exponent
<i>y</i>	FF instance
<i>f</i>	FF exponent
<i>p</i>	FF modulus
<i>n</i>	size of FF in BIGs
<i>en</i>	size of the exponent in BIGs

## 8.16.3.32 FF\_4096\_skspow()

```
void FF_4096_skspow (
    BIG_512_60 * r,
```

```

    BIG_512_60 * x,
    BIG_512_60 e,
    BIG_512_60 * p,
    int n )

```

For short BIG exponent

#### Parameters

<i>r</i>	FF instance, on exit = $x^e \bmod p$
<i>x</i>	FF instance
<i>e</i>	BIG exponent
<i>p</i>	FF modulus
<i>n</i>	size of FF in BIGs

#### 8.16.3.33 FF\_4096\_sqr()

```

void FF_4096_sqr (
    BIG_512_60 * x,
    BIG_512_60 * y,
    int n )

```

Uses Karatsuba method internally

#### Parameters

<i>x</i>	FF instance, on exit = $y^2$
<i>y</i>	FF instance to be squared
<i>n</i>	size of FF in BIGs

#### 8.16.3.34 FF\_4096\_sub()

```

void FF_4096_sub (
    BIG_512_60 * x,
    BIG_512_60 * y,
    BIG_512_60 * z,
    int n )

```

#### Parameters

<i>x</i>	FF instance, on exit = $y-z$
<i>y</i>	FF instance
<i>z</i>	FF instance
<i>n</i>	size of FF in BIGs

**8.16.3.35 FF\_4096\_toOctet()**

```
void FF_4096_toOctet (
    octet * S,
    BIG_512_60 * x,
    int n )
```

Converts an FF to big-endian base 256 form.

**Parameters**

<i>S</i>	output octet string
<i>x</i>	FF instance to be converted to an octet string
<i>n</i>	size of FF in BIGs

**8.16.3.36 FF\_4096\_zero()**

```
void FF_4096_zero (
    BIG_512_60 * x,
    int n )
```

**Parameters**

<i>x</i>	FF instance to be set to zero
<i>n</i>	size of FF in BIGs

**8.17 fp12\_BLS381.h File Reference**

FP12 Header File.

```
#include "fp4_BLS381.h"
```

**Data Structures**

- struct [FP12\\_BLS381](#)

*FP12 Structure - towered over three FP4.*

## Functions

- int `FP12_BLS381_iszilch` (`FP12_BLS381 *x`)  
*Tests for FP12 equal to zero.*
- int `FP12_BLS381_isunity` (`FP12_BLS381 *x`)  
*Tests for FP12 equal to unity.*
- void `FP12_BLS381_copy` (`FP12_BLS381 *x`, `FP12_BLS381 *y`)  
*Copy FP12 to another FP12.*
- void `FP12_BLS381_one` (`FP12_BLS381 *x`)  
*Set FP12 to unity.*
- void `FP12_BLS381_zero` (`FP12_BLS381 *x`)  
*Set FP12 to zero.*
- int `FP12_BLS381_equals` (`FP12_BLS381 *x`, `FP12_BLS381 *y`)  
*Tests for equality of two FP12s.*
- void `FP12_BLS381_conj` (`FP12_BLS381 *x`, `FP12_BLS381 *y`)  
*Conjugation of FP12.*
- void `FP12_BLS381_from_FP4` (`FP12_BLS381 *x`, `FP4_BLS381 *a`)  
*Initialise FP12 from single FP4.*
- void `FP12_BLS381_from_FP4s` (`FP12_BLS381 *x`, `FP4_BLS381 *a`, `FP4_BLS381 *b`, `FP4_BLS381 *c`)  
*Initialise FP12 from three FP4s.*
- void `FP12_BLS381_usqr` (`FP12_BLS381 *x`, `FP12_BLS381 *y`)  
*Fast Squaring of an FP12 in "unitary" form.*
- void `FP12_BLS381_sqr` (`FP12_BLS381 *x`, `FP12_BLS381 *y`)  
*Squaring an FP12.*
- void `FP12_BLS381_smul` (`FP12_BLS381 *x`, `FP12_BLS381 *y`)  
*Fast multiplication of two sparse FP12s that arises from ATE pairing line functions.*
- void `FP12_BLS381_ssmul` (`FP12_BLS381 *x`, `FP12_BLS381 *y`)  
*Fast multiplication of what may be sparse multiplicands.*
- void `FP12_BLS381_mul` (`FP12_BLS381 *x`, `FP12_BLS381 *y`)  
*Full unconditional Multiplication of two FP12s.*
- void `FP12_BLS381_inv` (`FP12_BLS381 *x`, `FP12_BLS381 *y`)  
*Inverting an FP12.*
- void `FP12_BLS381_pow` (`FP12_BLS381 *r`, `FP12_BLS381 *x`, `BIG_384_58 b`)  
*Raises an FP12 to the power of a BIG.*
- void `FP12_BLS381_pinpow` (`FP12_BLS381 *x`, int `i`, int `b`)  
*Raises an FP12 instance x to a small integer power, side-channel resistant.*
- void `FP12_BLS381_compow` (`FP4_BLS381 *c`, `FP12_BLS381 *x`, `BIG_384_58 e`, `BIG_384_58 r`)  
*Raises an FP12 instance x to a BIG power, compressed to FP4.*
- void `FP12_BLS381_pow4` (`FP12_BLS381 *r`, `FP12_BLS381 *x`, `BIG_384_58 *b`)  
*Calculate  $x[0]^b[0].x[1]^b[1].x[2]^b[2].x[3]^b[3]$ , side-channel resistant.*
- void `FP12_BLS381_frob` (`FP12_BLS381 *x`, `FP2_BLS381 *f`)  
*Raises an FP12 to the power of the internal modulus p, using the Frobenius.*
- void `FP12_BLS381_reduce` (`FP12_BLS381 *x`)  
*Reduces all components of possibly unreduced FP12 mod Modulus.*
- void `FP12_BLS381_norm` (`FP12_BLS381 *x`)  
*Normalises the components of an FP12.*
- void `FP12_BLS381_output` (`FP12_BLS381 *x`)  
*Formats and outputs an FP12 to the console.*
- void `FP12_BLS381_toOctet` (`octet *S`, `FP12_BLS381 *x`)  
*Formats and outputs an FP12 instance to an octet string.*
- void `FP12_BLS381_fromOctet` (`FP12_BLS381 *x`, `octet *S`)

*Creates an FP12 instance from an octet string.*

- void `FP12_BLS381_trace` (`FP4_BLS381 *t`, `FP12_BLS381 *x`)

*Calculate the trace of an FP12.*

- void `FP12_BLS381_cmove` (`FP12_BLS381 *x`, `FP12_BLS381 *y`, int `s`)

*Conditional copy of FP12 number.*

## Variables

- const `BIG_384_58 Fra_BLS381`
- const `BIG_384_58 Frb_BLS381`

## 8.17.1 Detailed Description

### Author

Mike Scott

## 8.17.2 Function Documentation

### 8.17.2.1 FP12\_BLS381\_cmove()

```
void FP12_BLS381_cmove (
    FP12_BLS381 * x,
    FP12_BLS381 * y,
    int s )
```

Conditionally copies second parameter to the first (without branching)

#### Parameters

<code>x</code>	FP12 instance, set to <code>y</code> if <code>s!=0</code>
<code>y</code>	another FP12 instance
<code>s</code>	copy only takes place if not equal to 0

### 8.17.2.2 FP12\_BLS381\_compow()

```
void FP12_BLS381_compow (
    FP4_BLS381 * c,
    FP12_BLS381 * x,
    BIG_384_58 e,
    BIG_384_58 r )
```

**Parameters**

<i>c</i>	FP4 instance, on exit = $x^{(e \bmod r)}$ as FP4
<i>x</i>	FP12 input
<i>e</i>	BIG exponent
<i>r</i>	BIG group order

**8.17.2.3 FP12\_BLS381\_conj()**

```
void FP12_BLS381_conj (
    FP12_BLS381 * x,
    FP12_BLS381 * y )
```

If  $y=(a,b,c)$  (where  $a,b,c$  are its three FP4 components) on exit  $x=(\text{conj}(a),-\text{conj}(b),\text{conj}(c))$

**Parameters**

<i>x</i>	FP12 instance, on exit = $\text{conj}(y)$
<i>y</i>	FP12 instance

**8.17.2.4 FP12\_BLS381\_copy()**

```
void FP12_BLS381_copy (
    FP12_BLS381 * x,
    FP12_BLS381 * y )
```

**Parameters**

<i>x</i>	FP12 instance, on exit = <i>y</i>
<i>y</i>	FP12 instance to be copied

**8.17.2.5 FP12\_BLS381\_equals()**

```
int FP12_BLS381_equals (
    FP12_BLS381 * x,
    FP12_BLS381 * y )
```

**Parameters**

<i>x</i>	FP12 instance to be compared
<i>y</i>	FP12 instance to be compared

**Returns**

1 if  $x=y$ , else returns 0

**8.17.2.6 FP12\_BLS381\_frob()**

```
void FP12_BLS381_frob (
    FP12_BLS381 * x,
    FP2_BLS381 * f )
```

**Parameters**

<i>x</i>	FP12 instance, on exit = $x^p$
<i>f</i>	FP2 precalculated Frobenius constant

**8.17.2.7 FP12\_BLS381\_from\_FP4()**

```
void FP12_BLS381_from_FP4 (
    FP12_BLS381 * x,
    FP4_BLS381 * a )
```

Sets first FP4 component of an FP12, other components set to zero

**Parameters**

<i>x</i>	FP12 instance to be initialised
<i>a</i>	FP4 to form first part of FP4

**8.17.2.8 FP12\_BLS381\_from\_FP4s()**

```
void FP12_BLS381_from_FP4s (
    FP12_BLS381 * x,
    FP4_BLS381 * a,
    FP4_BLS381 * b,
    FP4_BLS381 * c )
```

**Parameters**

<i>x</i>	FP12 instance to be initialised
<i>a</i>	FP4 to form first part of FP12
<i>b</i>	FP4 to form second part of FP12
<i>c</i>	FP4 to form third part of FP12

### 8.17.2.9 FP12\_BLS381\_fromOctet()

```
void FP12_BLS381_fromOctet (
    FP12_BLS381 * x,
    octet * S )
```

De-serializes the components of an FP12 to create an FP12 from big-endian base 256 components.

#### Parameters

x	FP12 instance to be created from an octet string
S	input octet string

### 8.17.2.10 FP12\_BLS381\_inv()

```
void FP12_BLS381_inv (
    FP12_BLS381 * x,
    FP12_BLS381 * y )
```

#### Parameters

x	FP12 instance, on exit = 1/y
y	FP12 instance

### 8.17.2.11 FP12\_BLS381\_isunity()

```
int FP12_BLS381_isunity (
    FP12_BLS381 * x )
```

#### Parameters

x	FP12 number to be tested
---	--------------------------

#### Returns

1 if unity, else returns 0



## 8.17.2.12 FP12\_BLS381\_iszilch()

```
int FP12_BLS381_iszilch (
    FP12_BLS381 * x )
```

**Parameters**

<i>x</i>	FP12 number to be tested
----------	--------------------------

**Returns**

1 if zero, else returns 0

**8.17.2.13 FP12\_BLS381\_mul()**

```
void FP12_BLS381_mul (
    FP12_BLS381 * x,
    FP12_BLS381 * y )
```

**Parameters**

<i>x</i>	FP12 instance, on exit = x*y
<i>y</i>	FP12 instance, the multiplier

**8.17.2.14 FP12\_BLS381\_norm()**

```
void FP12_BLS381_norm (
    FP12_BLS381 * x )
```

**Parameters**

<i>x</i>	FP12 instance to be normalised
----------	--------------------------------

**8.17.2.15 FP12\_BLS381\_one()**

```
void FP12_BLS381_one (
    FP12_BLS381 * x )
```

**Parameters**

<i>x</i>	FP12 instance to be set to one
----------	--------------------------------

## 8.17.2.16 FP12\_BLS381\_output()

```
void FP12_BLS381_output (
    FP12_BLS381 * x )
```

## Parameters

<i>x</i>	FP12 instance to be printed
----------	-----------------------------

## 8.17.2.17 FP12\_BLS381\_pinpow()

```
void FP12_BLS381_pinpow (
    FP12_BLS381 * x,
    int i,
    int b )
```

## Parameters

<i>x</i>	FP12 instance, on exit = $x^i$
<i>i</i>	small integer exponent
<i>b</i>	maximum number of bits in exponent

## 8.17.2.18 FP12\_BLS381\_pow()

```
void FP12_BLS381_pow (
    FP12_BLS381 * r,
    FP12_BLS381 * x,
    BIG_384_58 b )
```

## Parameters

<i>r</i>	FP12 instance, on exit = $y^b$
<i>x</i>	FP12 instance
<i>b</i>	BIG number

## 8.17.2.19 FP12\_BLS381\_pow4()

```
void FP12_BLS381_pow4 (
    FP12_BLS381 * r,
    FP12_BLS381 * x,
    BIG_384_58 * b )
```

## Parameters

<i>r</i>	FP12 instance, on exit = $x[0]^b[0].x[1]^b[1].x[2]^b[2].x[3]^b[3]$
<i>x</i>	FP12 array with 4 FP12s
<i>b</i>	BIG array of 4 exponents

## 8.17.2.20 FP12\_BLS381\_reduce()

```
void FP12_BLS381_reduce (
    FP12_BLS381 * x )
```

## Parameters

<i>x</i>	FP12 instance, on exit reduced mod Modulus
----------	--

## 8.17.2.21 FP12\_BLS381\_smul()

```
void FP12_BLS381_smul (
    FP12_BLS381 * x,
    FP12_BLS381 * y )
```

## Parameters

<i>x</i>	FP12 instance, on exit = $x*y$
<i>y</i>	FP12 instance, of special form

## 8.17.2.22 FP12\_BLS381\_sqr()

```
void FP12_BLS381_sqr (
    FP12_BLS381 * x,
    FP12_BLS381 * y )
```

## Parameters

<i>x</i>	FP12 instance, on exit = $y^2$
<i>y</i>	FP12 instance

## 8.17.2.23 FP12\_BLS381\_ssmul()

```
void FP12_BLS381_ssmul (
    FP12_BLS381 * x,
    FP12_BLS381 * y )
```

## Parameters

<i>x</i>	FP12 instance, on exit = $x*y$
<i>y</i>	FP12 instance, of special form

## 8.17.2.24 FP12\_BLS381\_toOctet()

```
void FP12_BLS381_toOctet (
    octet * S,
    FP12_BLS381 * x )
```

Serializes the components of an FP12 to big-endian base 256 form.

## Parameters

<i>S</i>	output octet string
<i>x</i>	FP12 instance to be converted to an octet string

## 8.17.2.25 FP12\_BLS381\_trace()

```
void FP12_BLS381_trace (
    FP4_BLS381 * t,
    FP12_BLS381 * x )
```

## Parameters

<i>t</i>	FP4 trace of $x$ , on exit = $tr(x)$
<i>x</i>	FP12 instance

## 8.17.2.26 FP12\_BLS381\_usqr()

```
void FP12_BLS381_usqr (
    FP12_BLS381 * x,
    FP12_BLS381 * y )
```

**Parameters**

<i>x</i>	FP12 instance, on exit = $y^2$
<i>y</i>	FP4 instance, must be unitary

**8.17.2.27 FP12\_BLS381\_zero()**

```
void FP12_BLS381_zero (
    FP12_BLS381 * x )
```

**Parameters**

<i>x</i>	FP12 instance to be set to zero
----------	---------------------------------

**8.17.3 Variable Documentation****8.17.3.1 Fra\_BLS381**

```
const BIG_384_58 Fra_BLS381
```

real part of BN curve Frobenius Constant

**8.17.3.2 Frb\_BLS381**

```
const BIG_384_58 Frb_BLS381
```

imaginary part of BN curve Frobenius Constant

**8.18 fp2\_BLS381.h File Reference**

FP2 Header File.

```
#include "fp_BLS381.h"
```

**Data Structures**

- struct [FP2\\_BLS381](#)  
*FP2 Structure - quadratic extension field.*

## Functions

- int `FP2_BLS381_iszilch` (`FP2_BLS381 *x`)  
*Tests for FP2 equal to zero.*
- void `FP2_BLS381_cmove` (`FP2_BLS381 *x`, `FP2_BLS381 *y`, int `s`)  
*Conditional copy of FP2 number.*
- int `FP2_BLS381_isunity` (`FP2_BLS381 *x`)  
*Tests for FP2 equal to one.*
- int `FP2_BLS381_equals` (`FP2_BLS381 *x`, `FP2_BLS381 *y`)  
*Tests for equality of two FP2s.*
- void `FP2_BLS381_from_FPs` (`FP2_BLS381 *x`, `FP_BLS381 *a`, `FP_BLS381 *b`)  
*Initialise FP2 from two FP numbers.*
- void `FP2_BLS381_from_BIGs` (`FP2_BLS381 *x`, `BIG_384_58 a`, `BIG_384_58 b`)  
*Initialise FP2 from two BIG integers.*
- void `FP2_BLS381_from_FP` (`FP2_BLS381 *x`, `FP_BLS381 *a`)  
*Initialise FP2 from single FP.*
- void `FP2_BLS381_from_BIG` (`FP2_BLS381 *x`, `BIG_384_58 a`)  
*Initialise FP2 from single BIG.*
- void `FP2_BLS381_copy` (`FP2_BLS381 *x`, `FP2_BLS381 *y`)  
*Copy FP2 to another FP2.*
- void `FP2_BLS381_zero` (`FP2_BLS381 *x`)  
*Set FP2 to zero.*
- void `FP2_BLS381_one` (`FP2_BLS381 *x`)  
*Set FP2 to unity.*
- void `FP2_BLS381_neg` (`FP2_BLS381 *x`, `FP2_BLS381 *y`)  
*Negation of FP2.*
- void `FP2_BLS381_conj` (`FP2_BLS381 *x`, `FP2_BLS381 *y`)  
*Conjugation of FP2.*
- void `FP2_BLS381_add` (`FP2_BLS381 *x`, `FP2_BLS381 *y`, `FP2_BLS381 *z`)  
*addition of two FP2s*
- void `FP2_BLS381_sub` (`FP2_BLS381 *x`, `FP2_BLS381 *y`, `FP2_BLS381 *z`)  
*subtraction of two FP2s*
- void `FP2_BLS381_pmul` (`FP2_BLS381 *x`, `FP2_BLS381 *y`, `FP_BLS381 *b`)  
*Multiplication of an FP2 by an FP.*
- void `FP2_BLS381_imul` (`FP2_BLS381 *x`, `FP2_BLS381 *y`, int `i`)  
*Multiplication of an FP2 by a small integer.*
- void `FP2_BLS381_sqr` (`FP2_BLS381 *x`, `FP2_BLS381 *y`)  
*Squaring an FP2.*
- void `FP2_BLS381_mul` (`FP2_BLS381 *x`, `FP2_BLS381 *y`, `FP2_BLS381 *z`)  
*Multiplication of two FP2s.*
- void `FP2_BLS381_output` (`FP2_BLS381 *x`)  
*Formats and outputs an FP2 to the console.*
- void `FP2_BLS381_rawoutput` (`FP2_BLS381 *x`)  
*Formats and outputs an FP2 to the console in raw form (for debugging)*
- void `FP2_BLS381_inv` (`FP2_BLS381 *x`, `FP2_BLS381 *y`)  
*Inverting an FP2.*
- void `FP2_BLS381_div2` (`FP2_BLS381 *x`, `FP2_BLS381 *y`)  
*Divide an FP2 by 2.*
- void `FP2_BLS381_mul_ip` (`FP2_BLS381 *x`)  
*Multiply an FP2 by (1+sqrt(-1))*
- void `FP2_BLS381_div_ip2` (`FP2_BLS381 *x`)

- Divide an FP2 by  $(1+\sqrt{-1})/2$  -.*

  - void `FP2_BLS381_div_ip` (`FP2_BLS381 *x`)
- Divide an FP2 by  $(1+\sqrt{-1})$*

  - void `FP2_BLS381_norm` (`FP2_BLS381 *x`)
- Normalises the components of an FP2.*

  - void `FP2_BLS381_reduce` (`FP2_BLS381 *x`)
- Reduces all components of possibly unreduced FP2 mod Modulus.*

  - void `FP2_BLS381_pow` (`FP2_BLS381 *x`, `FP2_BLS381 *y`, `BIG_384_58 b`)
- Raises an FP2 to the power of a BIG.*

  - int `FP2_BLS381_sqrt` (`FP2_BLS381 *x`, `FP2_BLS381 *y`)
- Square root of an FP2.*

  - void `FP2_BLS381_times_i` (`FP2_BLS381 *x`)
- Multiply an FP2 by  $\sqrt{-1}$*

## 8.18.1 Detailed Description

### Author

Mike Scott

## 8.18.2 Function Documentation

### 8.18.2.1 `FP2_BLS381_add()`

```
void FP2_BLS381_add (
    FP2_BLS381 * x,
    FP2_BLS381 * y,
    FP2_BLS381 * z )
```

#### Parameters

<code>x</code>	FP2 instance, on exit = <code>y+z</code>
<code>y</code>	FP2 instance
<code>z</code>	FP2 instance

### 8.18.2.2 `FP2_BLS381_cmove()`

```
void FP2_BLS381_cmove (
    FP2_BLS381 * x,
    FP2_BLS381 * y,
    int s )
```

Conditionally copies second parameter to the first (without branching)



## Parameters

<i>x</i>	FP2 instance, set to <i>y</i> if <i>s</i> !=0
<i>y</i>	another FP2 instance
<i>s</i>	copy only takes place if not equal to 0

## 8.18.2.3 FP2\_BLS381\_conj()

```
void FP2_BLS381_conj (
    FP2_BLS381 * x,
    FP2_BLS381 * y )
```

If *y*=(*a*,*b*) on exit *x*=(*a*,-*b*)

## Parameters

<i>x</i>	FP2 instance, on exit = conj( <i>y</i> )
<i>y</i>	FP2 instance

## 8.18.2.4 FP2\_BLS381\_copy()

```
void FP2_BLS381_copy (
    FP2_BLS381 * x,
    FP2_BLS381 * y )
```

## Parameters

<i>x</i>	FP2 instance, on exit = <i>y</i>
<i>y</i>	FP2 instance to be copied

## 8.18.2.5 FP2\_BLS381\_div2()

```
void FP2_BLS381_div2 (
    FP2_BLS381 * x,
    FP2_BLS381 * y )
```

## Parameters

<i>x</i>	FP2 instance, on exit = <i>y</i> /2
<i>y</i>	FP2 instance

**8.18.2.6 FP2\_BLS381\_div\_ip()**

```
void FP2_BLS381_div_ip (
    FP2_BLS381 * x )
```

Note that  $(1+\sqrt{-1})$  is irreducible for FP4

**Parameters**

<i>x</i>	FP2 instance, on exit = $x/(1+\sqrt{-1})$
----------	---

**8.18.2.7 FP2\_BLS381\_div\_ip2()**

```
void FP2_BLS381_div_ip2 (
    FP2_BLS381 * x )
```

Note that  $(1+\sqrt{-1})$  is irreducible for FP4

**Parameters**

<i>x</i>	FP2 instance, on exit = $2x/(1+\sqrt{-1})$
----------	--

**8.18.2.8 FP2\_BLS381\_equals()**

```
int FP2_BLS381_equals (
    FP2_BLS381 * x,
    FP2_BLS381 * y )
```

**Parameters**

<i>x</i>	FP2 instance to be compared
<i>y</i>	FP2 instance to be compared

**Returns**

1 if  $x=y$ , else returns 0

### 8.18.2.9 FP2\_BLS381\_from\_BIG()

```
void FP2_BLS381_from_BIG (
    FP2_BLS381 * x,
    BIG_384_58 a )
```

Imaginary part is set to zero

#### Parameters

<i>x</i>	FP2 instance to be initialised
<i>a</i>	BIG to form real part of FP2

### 8.18.2.10 FP2\_BLS381\_from\_BIGs()

```
void FP2_BLS381_from_BIGs (
    FP2_BLS381 * x,
    BIG_384_58 a,
    BIG_384_58 b )
```

#### Parameters

<i>x</i>	FP2 instance to be initialised
<i>a</i>	BIG to form real part of FP2
<i>b</i>	BIG to form imaginary part of FP2

### 8.18.2.11 FP2\_BLS381\_from\_FP()

```
void FP2_BLS381_from_FP (
    FP2_BLS381 * x,
    FP_BLS381 * a )
```

Imaginary part is set to zero

#### Parameters

<i>x</i>	FP2 instance to be initialised
<i>a</i>	FP to form real part of FP2

### 8.18.2.12 FP2\_BLS381\_from\_FPs()

```
void FP2_BLS381_from_FPs (
    FP2_BLS381 * x,
```

```

FP_BLS381 * a,
FP_BLS381 * b )

```

#### Parameters

<i>x</i>	FP2 instance to be initialised
<i>a</i>	FP to form real part of FP2
<i>b</i>	FP to form imaginary part of FP2

#### 8.18.2.13 FP2\_BLS381\_imul()

```

void FP2_BLS381_imul (
    FP2_BLS381 * x,
    FP2_BLS381 * y,
    int i )

```

#### Parameters

<i>x</i>	FP2 instance, on exit = $y*i$
<i>y</i>	FP2 instance
<i>i</i>	an integer

#### 8.18.2.14 FP2\_BLS381\_inv()

```

void FP2_BLS381_inv (
    FP2_BLS381 * x,
    FP2_BLS381 * y )

```

#### Parameters

<i>x</i>	FP2 instance, on exit = $1/y$
<i>y</i>	FP2 instance

#### 8.18.2.15 FP2\_BLS381\_isunity()

```

int FP2_BLS381_isunity (
    FP2_BLS381 * x )

```

#### Parameters

<i>x</i>	FP2 instance to be tested
----------	---------------------------

**Returns**

1 if  $x=1$ , else returns 0

**8.18.2.16 FP2\_BLS381\_iszilch()**

```
int FP2_BLS381_iszilch (
    FP2_BLS381 * x )
```

**Parameters**

$x$	FP2 number to be tested
-----	-------------------------

**Returns**

1 if zero, else returns 0

**8.18.2.17 FP2\_BLS381\_mul()**

```
void FP2_BLS381_mul (
    FP2_BLS381 * x,
    FP2_BLS381 * y,
    FP2_BLS381 * z )
```

**Parameters**

$x$	FP2 instance, on exit = $y*z$
$y$	FP2 instance
$z$	FP2 instance

**8.18.2.18 FP2\_BLS381\_mul\_ip()**

```
void FP2_BLS381_mul_ip (
    FP2_BLS381 * x )
```

Note that  $(1+\sqrt{-1})$  is irreducible for FP4

**Parameters**

$x$	FP2 instance, on exit = $x*(1+\sqrt{-1})$
-----	---

### 8.18.2.19 FP2\_BLS381\_neg()

```
void FP2_BLS381_neg (
    FP2_BLS381 * x,
    FP2_BLS381 * y )
```

#### Parameters

x	FP2 instance, on exit = -y
y	FP2 instance

### 8.18.2.20 FP2\_BLS381\_norm()

```
void FP2_BLS381_norm (
    FP2_BLS381 * x )
```

#### Parameters

x	FP2 instance to be normalised
---	-------------------------------

### 8.18.2.21 FP2\_BLS381\_one()

```
void FP2_BLS381_one (
    FP2_BLS381 * x )
```

#### Parameters

x	FP2 instance to be set to one
---	-------------------------------

### 8.18.2.22 FP2\_BLS381\_output()

```
void FP2_BLS381_output (
    FP2_BLS381 * x )
```

#### Parameters

x	FP2 instance
---	--------------

### 8.18.2.23 FP2\_BLS381\_pmul()

```
void FP2_BLS381_pmul (
    FP2_BLS381 * x,
    FP2_BLS381 * y,
    FP_BLS381 * b )
```

#### Parameters

<i>x</i>	FP2 instance, on exit = $y*b$
<i>y</i>	FP2 instance
<i>b</i>	FP residue

### 8.18.2.24 FP2\_BLS381\_pow()

```
void FP2_BLS381_pow (
    FP2_BLS381 * x,
    FP2_BLS381 * y,
    BIG_384_58 b )
```

#### Parameters

<i>x</i>	FP2 instance, on exit = $y^b$
<i>y</i>	FP2 instance
<i>b</i>	BIG number

### 8.18.2.25 FP2\_BLS381\_rawoutput()

```
void FP2_BLS381_rawoutput (
    FP2_BLS381 * x )
```

#### Parameters

<i>x</i>	FP2 instance
----------	--------------

### 8.18.2.26 FP2\_BLS381\_reduce()

```
void FP2_BLS381_reduce (
    FP2_BLS381 * x )
```

## Parameters

$x$	FP2 instance, on exit reduced mod Modulus
-----	---

## 8.18.2.27 FP2\_BLS381\_sqr()

```
void FP2_BLS381_sqr (
    FP2_BLS381 * x,
    FP2_BLS381 * y )
```

## Parameters

$x$	FP2 instance, on exit = $y^2$
$y$	FP2 instance

## 8.18.2.28 FP2\_BLS381\_sqrt()

```
int FP2_BLS381_sqrt (
    FP2_BLS381 * x,
    FP2_BLS381 * y )
```

## Parameters

$x$	FP2 instance, on exit = $\text{sqrt}(y)$
$y$	FP2 instance

## 8.18.2.29 FP2\_BLS381\_sub()

```
void FP2_BLS381_sub (
    FP2_BLS381 * x,
    FP2_BLS381 * y,
    FP2_BLS381 * z )
```

## Parameters

$x$	FP2 instance, on exit = $y-z$
$y$	FP2 instance
$z$	FP2 instance



## 8.18.2.30 FP2\_BLS381\_times\_i()

```
void FP2_BLS381_times_i (
    FP2_BLS381 * x )
```

Note that -1 is QNR

## Parameters

x	FP2 instance, on exit = $x \cdot \sqrt{-1}$
---	---

## 8.18.2.31 FP2\_BLS381\_zero()

```
void FP2_BLS381_zero (
    FP2_BLS381 * x )
```

## Parameters

x	FP2 instance to be set to zero
---	--------------------------------

## 8.19 fp4\_BLS381.h File Reference

FP4 Header File.

```
#include "fp2_BLS381.h"
#include "config_curve_BLS381.h"
```

## Data Structures

- struct [FP4\\_BLS381](#)  
*FP4 Structure - towered over two FP2.*

## Functions

- int [FP4\\_BLS381\\_iszilch](#) ([FP4\\_BLS381](#) \*x)  
*Tests for FP4 equal to zero.*
- int [FP4\\_BLS381\\_isunity](#) ([FP4\\_BLS381](#) \*x)  
*Tests for FP4 equal to unity.*
- int [FP4\\_BLS381\\_equals](#) ([FP4\\_BLS381](#) \*x, [FP4\\_BLS381](#) \*y)  
*Tests for equality of two FP4s.*
- int [FP4\\_BLS381\\_isreal](#) ([FP4\\_BLS381](#) \*x)  
*Tests for FP4 having only a real part and no imaginary part.*
- void [FP4\\_BLS381\\_from\\_FP2s](#) ([FP4\\_BLS381](#) \*x, [FP2\\_BLS381](#) \*a, [FP2\\_BLS381](#) \*b)

- Initialise FP4 from two FP2s.*

  - void `FP4_BLS381_from_FP2` (`FP4_BLS381 *x`, `FP2_BLS381 *a`)
- Initialise FP4 from single FP2.*

  - void `FP4_BLS381_from_FP2H` (`FP4_BLS381 *x`, `FP2_BLS381 *a`)
- Initialise FP4 from single FP2.*

  - void `FP4_BLS381_copy` (`FP4_BLS381 *x`, `FP4_BLS381 *y`)
- Copy FP4 to another FP4.*

  - void `FP4_BLS381_zero` (`FP4_BLS381 *x`)
- Set FP4 to zero.*

  - void `FP4_BLS381_one` (`FP4_BLS381 *x`)
- Set FP4 to unity.*

  - void `FP4_BLS381_neg` (`FP4_BLS381 *x`, `FP4_BLS381 *y`)
- Negation of FP4.*

  - void `FP4_BLS381_conj` (`FP4_BLS381 *x`, `FP4_BLS381 *y`)
- Conjugation of FP4.*

  - void `FP4_BLS381_nconj` (`FP4_BLS381 *x`, `FP4_BLS381 *y`)
- Negative conjugation of FP4.*

  - void `FP4_BLS381_add` (`FP4_BLS381 *x`, `FP4_BLS381 *y`, `FP4_BLS381 *z`)
- addition of two FP4s*

  - void `FP4_BLS381_sub` (`FP4_BLS381 *x`, `FP4_BLS381 *y`, `FP4_BLS381 *z`)
- subtraction of two FP4s*

  - void `FP4_BLS381_pmul` (`FP4_BLS381 *x`, `FP4_BLS381 *y`, `FP2_BLS381 *a`)
- Multiplication of an FP4 by an FP2.*

  - void `FP4_BLS381_qmul` (`FP4_BLS381 *x`, `FP4_BLS381 *y`, `FP_BLS381 *a`)
- Multiplication of an FP4 by an FP.*

  - void `FP4_BLS381_imul` (`FP4_BLS381 *x`, `FP4_BLS381 *y`, `int i`)
- Multiplication of an FP4 by a small integer.*

  - void `FP4_BLS381_sqr` (`FP4_BLS381 *x`, `FP4_BLS381 *y`)
- Squaring an FP4.*

  - void `FP4_BLS381_mul` (`FP4_BLS381 *x`, `FP4_BLS381 *y`, `FP4_BLS381 *z`)
- Multiplication of two FP4s.*

  - void `FP4_BLS381_inv` (`FP4_BLS381 *x`, `FP4_BLS381 *y`)
- Inverting an FP4.*

  - void `FP4_BLS381_output` (`FP4_BLS381 *x`)
- Formats and outputs an FP4 to the console.*

  - void `FP4_BLS381_rawoutput` (`FP4_BLS381 *x`)
- Formats and outputs an FP4 to the console in raw form (for debugging)*

  - void `FP4_BLS381_times_i` (`FP4_BLS381 *x`)
- multiplies an FP4 instance by irreducible polynomial  $\text{sqrt}(1+\text{sqrt}(-1))$*

  - void `FP4_BLS381_norm` (`FP4_BLS381 *x`)
- Normalises the components of an FP4.*

  - void `FP4_BLS381_reduce` (`FP4_BLS381 *x`)
- Reduces all components of possibly unreduced FP4 mod Modulus.*

  - void `FP4_BLS381_pow` (`FP4_BLS381 *x`, `FP4_BLS381 *y`, `BIG_384_58 b`)
- Raises an FP4 to the power of a BIG.*

  - void `FP4_BLS381_frob` (`FP4_BLS381 *x`, `FP2_BLS381 *f`)
- Raises an FP4 to the power of the internal modulus p, using the Frobenius.*

  - void `FP4_BLS381_xtr_A` (`FP4_BLS381 *r`, `FP4_BLS381 *w`, `FP4_BLS381 *x`, `FP4_BLS381 *y`, `FP4_BLS381 *z`)
- Calculates the XTR addition function  $r=w*x-\text{conj}(x)*y+z$ .*

  - void `FP4_BLS381_xtr_D` (`FP4_BLS381 *r`, `FP4_BLS381 *x`)

- Calculates the XTR doubling function  $r=x^{2-2*\text{conj}(x)}$*

  - void `FP4_BLS381_xtr_pow` (`FP4_BLS381 *r`, `FP4_BLS381 *x`, `BIG_384_58 b`)

*Calculates FP4 trace of an FP12 raised to the power of a BIG number.*

  - void `FP4_BLS381_xtr_pow2` (`FP4_BLS381 *r`, `FP4_BLS381 *c`, `FP4_BLS381 *d`, `FP4_BLS381 *e`, `FP4_BLS381 *f`, `BIG_384_58 a`, `BIG_384_58 b`)

*Calculates FP4 trace of  $c^a.d^b$ , where  $c$  and  $d$  are derived from FP4 traces of FP12s.*

  - void `FP4_BLS381_cmove` (`FP4_BLS381 *x`, `FP4_BLS381 *y`, int  $s$ )

*Conditional copy of FP4 number.*

  - int `FP4_BLS381_sqrt` (`FP4_BLS381 *r`, `FP4_BLS381 *x`)

*Calculate square root of an FP4.*

  - void `FP4_BLS381_div_i` (`FP4_BLS381 *x`)

*Divide FP4 number by QNR.*

  - void `FP4_BLS381_div_2i` (`FP4_BLS381 *x`)

*Divide an FP4 by QNR/2.*

  - void `FP4_BLS381_div2` (`FP4_BLS381 *x`, `FP4_BLS381 *y`)

*Divide an FP4 by 2.*

## 8.19.1 Detailed Description

### Author

Mike Scott

## 8.19.2 Function Documentation

### 8.19.2.1 FP4\_BLS381\_add()

```
void FP4_BLS381_add (
    FP4_BLS381 * x,
    FP4_BLS381 * y,
    FP4_BLS381 * z )
```

#### Parameters

<code>x</code>	FP4 instance, on exit = $y+z$
<code>y</code>	FP4 instance
<code>z</code>	FP4 instance

### 8.19.2.2 FP4\_BLS381\_cmove()

```
void FP4_BLS381_cmove (
    FP4_BLS381 * x,
```

```
FP4_BLS381 * y,  
int s )
```

Conditionally copies second parameter to the first (without branching)

## Parameters

<i>x</i>	FP4 instance, set to <i>y</i> if <i>s</i> !=0
<i>y</i>	another FP4 instance
<i>s</i>	copy only takes place if not equal to 0

## 8.19.2.3 FP4\_BLS381\_conj()

```
void FP4_BLS381_conj (
    FP4_BLS381 * x,
    FP4_BLS381 * y )
```

If *y*=(*a*,*b*) on exit *x*=(*a*,*-b*)

## Parameters

<i>x</i>	FP4 instance, on exit = conj( <i>y</i> )
<i>y</i>	FP4 instance

## 8.19.2.4 FP4\_BLS381\_copy()

```
void FP4_BLS381_copy (
    FP4_BLS381 * x,
    FP4_BLS381 * y )
```

## Parameters

<i>x</i>	FP4 instance, on exit = <i>y</i>
<i>y</i>	FP4 instance to be copied

## 8.19.2.5 FP4\_BLS381\_div2()

```
void FP4_BLS381_div2 (
    FP4_BLS381 * x,
    FP4_BLS381 * y )
```

## Parameters

<i>x</i>	FP4 instance, on exit = <i>y</i> /2
<i>y</i>	FP4 instance

### 8.19.2.6 FP4\_BLS381\_div\_2i()

```
void FP4_BLS381_div_2i (
    FP4_BLS381 * x )
```

Divide FP4 by the QNR/2

#### Parameters

<i>x</i>	FP4 instance
----------	--------------

### 8.19.2.7 FP4\_BLS381\_div\_i()

```
void FP4_BLS381_div_i (
    FP4_BLS381 * x )
```

Divide FP4 by the QNR

#### Parameters

<i>x</i>	FP4 instance
----------	--------------

### 8.19.2.8 FP4\_BLS381\_equals()

```
int FP4_BLS381_equals (
    FP4_BLS381 * x,
    FP4_BLS381 * y )
```

#### Parameters

<i>x</i>	FP4 instance to be compared
<i>y</i>	FP4 instance to be compared

#### Returns

1 if x=y, else returns 0

## 8.19.2.9 FP4\_BLS381\_frob()

```
void FP4_BLS381_frob (
    FP4_BLS381 * x,
    FP2_BLS381 * f )
```

## Parameters

<i>x</i>	FP4 instance, on exit = $x^p$
<i>f</i>	FP2 precalculated Frobenius constant

## 8.19.2.10 FP4\_BLS381\_from\_FP2()

```
void FP4_BLS381_from_FP2 (
    FP4_BLS381 * x,
    FP2_BLS381 * a )
```

Imaginary part is set to zero

## Parameters

<i>x</i>	FP4 instance to be initialised
<i>a</i>	FP2 to form real part of FP4

## 8.19.2.11 FP4\_BLS381\_from\_FP2H()

```
void FP4_BLS381_from_FP2H (
    FP4_BLS381 * x,
    FP2_BLS381 * a )
```

real part is set to zero

## Parameters

<i>x</i>	FP4 instance to be initialised
<i>a</i>	FP2 to form imaginary part of FP4

## 8.19.2.12 FP4\_BLS381\_from\_FP2s()

```
void FP4_BLS381_from_FP2s (
    FP4_BLS381 * x,
```

```
FP2_BLS381 * a,  
FP2_BLS381 * b )
```



## Parameters

<i>x</i>	FP4 instance to be initialised
<i>a</i>	FP2 to form real part of FP4
<i>b</i>	FP2 to form imaginary part of FP4

## 8.19.2.13 FP4\_BLS381\_imul()

```
void FP4_BLS381_imul (
    FP4_BLS381 * x,
    FP4_BLS381 * y,
    int i )
```

## Parameters

<i>x</i>	FP4 instance, on exit = $y*i$
<i>y</i>	FP4 instance
<i>i</i>	an integer

## 8.19.2.14 FP4\_BLS381\_inv()

```
void FP4_BLS381_inv (
    FP4_BLS381 * x,
    FP4_BLS381 * y )
```

## Parameters

<i>x</i>	FP4 instance, on exit = $1/y$
<i>y</i>	FP4 instance

## 8.19.2.15 FP4\_BLS381\_isreal()

```
int FP4_BLS381_isreal (
    FP4_BLS381 * x )
```

## Parameters

<i>x</i>	FP4 number to be tested
----------	-------------------------

**Returns**

1 if real, else returns 0

**8.19.2.16 FP4\_BLS381\_unity()**

```
int FP4_BLS381_unity (
    FP4_BLS381 * x )
```

**Parameters**

<i>x</i>	FP4 number to be tested
----------	-------------------------

**Returns**

1 if unity, else returns 0

**8.19.2.17 FP4\_BLS381\_iszilch()**

```
int FP4_BLS381_iszilch (
    FP4_BLS381 * x )
```

**Parameters**

<i>x</i>	FP4 number to be tested
----------	-------------------------

**Returns**

1 if zero, else returns 0

**8.19.2.18 FP4\_BLS381\_mul()**

```
void FP4_BLS381_mul (
    FP4_BLS381 * x,
    FP4_BLS381 * y,
    FP4_BLS381 * z )
```

**Parameters**

<i>x</i>	FP4 instance, on exit = y*z
<i>y</i>	FP4 instance
<i>z</i>	FP4 instance

### 8.19.2.19 FP4\_BLS381\_nconj()

```
void FP4_BLS381_nconj (
    FP4_BLS381 * x,
    FP4_BLS381 * y )
```

If  $y=(a,b)$  on exit  $x=(-a,b)$

#### Parameters

<i>x</i>	FP4 instance, on exit = -conj(y)
<i>y</i>	FP4 instance

### 8.19.2.20 FP4\_BLS381\_neg()

```
void FP4_BLS381_neg (
    FP4_BLS381 * x,
    FP4_BLS381 * y )
```

#### Parameters

<i>x</i>	FP4 instance, on exit = -y
<i>y</i>	FP4 instance

### 8.19.2.21 FP4\_BLS381\_norm()

```
void FP4_BLS381_norm (
    FP4_BLS381 * x )
```

#### Parameters

<i>x</i>	FP4 instance to be normalised
----------	-------------------------------

### 8.19.2.22 FP4\_BLS381\_one()

```
void FP4_BLS381_one (
    FP4_BLS381 * x )
```

**Parameters**

<i>x</i>	FP4 instance to be set to one
----------	-------------------------------

**8.19.2.23 FP4\_BLS381\_output()**

```
void FP4_BLS381_output (
    FP4_BLS381 * x )
```

**Parameters**

<i>x</i>	FP4 instance to be printed
----------	----------------------------

**8.19.2.24 FP4\_BLS381\_pmul()**

```
void FP4_BLS381_pmul (
    FP4_BLS381 * x,
    FP4_BLS381 * y,
    FP2_BLS381 * a )
```

**Parameters**

<i>x</i>	FP4 instance, on exit = $y*a$
<i>y</i>	FP4 instance
<i>a</i>	FP2 multiplier

**8.19.2.25 FP4\_BLS381\_pow()**

```
void FP4_BLS381_pow (
    FP4_BLS381 * x,
    FP4_BLS381 * y,
    BIG_384_58 b )
```

**Parameters**

<i>x</i>	FP4 instance, on exit = $y^b$
<i>y</i>	FP4 instance
<i>b</i>	BIG number

### 8.19.2.26 FP4\_BLS381\_qmul()

```
void FP4_BLS381_qmul (
    FP4_BLS381 * x,
    FP4_BLS381 * y,
    FP_BLS381 * a )
```

#### Parameters

<i>x</i>	FP4 instance, on exit = $y*a$
<i>y</i>	FP4 instance
<i>a</i>	FP multiplier

### 8.19.2.27 FP4\_BLS381\_rawoutput()

```
void FP4_BLS381_rawoutput (
    FP4_BLS381 * x )
```

#### Parameters

<i>x</i>	FP4 instance to be printed
----------	----------------------------

### 8.19.2.28 FP4\_BLS381\_reduce()

```
void FP4_BLS381_reduce (
    FP4_BLS381 * x )
```

#### Parameters

<i>x</i>	FP4 instance, on exit reduced mod Modulus
----------	---

### 8.19.2.29 FP4\_BLS381\_sqr()

```
void FP4_BLS381_sqr (
    FP4_BLS381 * x,
    FP4_BLS381 * y )
```

#### Parameters

<i>x</i>	FP4 instance, on exit = $y^2$
<i>y</i>	FP4 instance

**8.19.2.30 FP4\_BLS381\_sqrt()**

```
int FP4_BLS381_sqrt (
    FP4_BLS381 * r,
    FP4_BLS381 * x )
```

Square root

**Parameters**

<i>r</i>	FP4 instance, on exit = sqrt(x)
<i>x</i>	FP4 instance

**Returns**

1 x is a QR, otherwise 0

**8.19.2.31 FP4\_BLS381\_sub()**

```
void FP4_BLS381_sub (
    FP4_BLS381 * x,
    FP4_BLS381 * y,
    FP4_BLS381 * z )
```

**Parameters**

<i>x</i>	FP4 instance, on exit = y-z
<i>y</i>	FP4 instance
<i>z</i>	FP4 instance

**8.19.2.32 FP4\_BLS381\_times\_i()**

```
void FP4_BLS381_times_i (
    FP4_BLS381 * x )
```

**Parameters**

<i>x</i>	FP4 instance, on exit = sqrt(1+sqrt(-1))*x
----------	--

## 8.19.2.33 FP4\_BLS381\_xtr\_A()

```
void FP4_BLS381_xtr_A (
    FP4_BLS381 * r,
    FP4_BLS381 * w,
    FP4_BLS381 * x,
    FP4_BLS381 * y,
    FP4_BLS381 * z )
```

## Parameters

<i>r</i>	FP4 instance, on exit = $w*x-\text{conj}(x)*y+z$
<i>w</i>	FP4 instance
<i>x</i>	FP4 instance
<i>y</i>	FP4 instance
<i>z</i>	FP4 instance

## 8.19.2.34 FP4\_BLS381\_xtr\_D()

```
void FP4_BLS381_xtr_D (
    FP4_BLS381 * r,
    FP4_BLS381 * x )
```

## Parameters

<i>r</i>	FP4 instance, on exit = $x^2-2*\text{conj}(x)$
<i>x</i>	FP4 instance

## 8.19.2.35 FP4\_BLS381\_xtr\_pow()

```
void FP4_BLS381_xtr_pow (
    FP4_BLS381 * r,
    FP4_BLS381 * x,
    BIG_384_58 b )
```

XTR single exponentiation

## Parameters

<i>r</i>	FP4 instance, on exit = $\text{trace}(w^b)$
<i>x</i>	FP4 instance, trace of an FP12 <i>w</i>
<i>b</i>	BIG number

### 8.19.2.36 FP4\_BLS381\_xtr\_pow2()

```
void FP4_BLS381_xtr_pow2 (
    FP4_BLS381 * r,
    FP4_BLS381 * c,
    FP4_BLS381 * d,
    FP4_BLS381 * e,
    FP4_BLS381 * f,
    BIG_384_58 a,
    BIG_384_58 b )
```

XTR double exponentiation Assumes  $c=\text{tr}(x^m)$ ,  $d=\text{tr}(x^n)$ ,  $e=\text{tr}(x^{(m-n)})$ ,  $f=\text{tr}(x^{(m-2n)})$

#### Parameters

<i>r</i>	FP4 instance, on exit = $\text{trace}(c^a.d^b)$
<i>c</i>	FP4 instance, trace of an FP12
<i>d</i>	FP4 instance, trace of an FP12
<i>e</i>	FP4 instance, trace of an FP12
<i>f</i>	FP4 instance, trace of an FP12
<i>a</i>	BIG number
<i>b</i>	BIG number

### 8.19.2.37 FP4\_BLS381\_zero()

```
void FP4_BLS381_zero (
    FP4_BLS381 * x )
```

#### Parameters

<i>x</i>	FP4 instance to be set to zero
----------	--------------------------------

## 8.20 fp\_BLS381.h File Reference

FP Header File.

```
#include "big_384_58.h"
#include "config_field_BLS381.h"
```

### Data Structures

- struct [FP\\_BLS381](#)

*FP Structure - quadratic extension field.*



## Macros

- #define `MODBITS_BLS381` `MBITS_BLS381`
- #define `TBITS_BLS381` `(MBITS_BLS381%BASEBITS_384_58)`
- #define `TMASK_BLS381` `((chunk)1<<TBITS_BLS381)-1)`
- #define `FEXCESS_BLS381` `((sign32)1<<MAXXES_BLS381)-1)`
- #define `OMASK_BLS381` `(-((chunk)(1)<<TBITS_BLS381))`

## Functions

- int `FP_BLS381_iszilch` (`FP_BLS381 *x`)  
*Tests for FP equal to zero mod Modulus.*
- void `FP_BLS381_zero` (`FP_BLS381 *x`)  
*Set FP to zero.*
- void `FP_BLS381_copy` (`FP_BLS381 *y, FP_BLS381 *x`)  
*Copy an FP.*
- void `FP_BLS381_rcopy` (`FP_BLS381 *y, const BIG_384_58 x`)  
*Copy from ROM to an FP.*
- int `FP_BLS381_equals` (`FP_BLS381 *x, FP_BLS381 *y`)  
*Compares two FPs.*
- void `FP_BLS381_cswap` (`FP_BLS381 *x, FP_BLS381 *y, int s`)  
*Conditional constant time swap of two FP numbers.*
- void `FP_BLS381_cmove` (`FP_BLS381 *x, FP_BLS381 *y, int s`)  
*Conditional copy of FP number.*
- void `FP_BLS381_nres` (`FP_BLS381 *y, BIG_384_58 x`)  
*Converts from BIG integer to residue form mod Modulus.*
- void `FP_BLS381_redc` (`BIG_384_58 x, FP_BLS381 *y`)  
*Converts from residue form back to BIG integer form.*
- void `FP_BLS381_one` (`FP_BLS381 *x`)  
*Sets FP to representation of unity in residue form.*
- void `FP_BLS381_mod` (`BIG_384_58 r, DBIG_384_58 d`)  
*Reduces DBIG to BIG exploiting special form of the modulus.*
- void `FP_BLS381_mul` (`FP_BLS381 *x, FP_BLS381 *y, FP_BLS381 *z`)  
*Fast Modular multiplication of two FPs, mod Modulus.*
- void `FP_BLS381_imul` (`FP_BLS381 *x, FP_BLS381 *y, int i`)  
*Fast Modular multiplication of an FP, by a small integer, mod Modulus.*
- void `FP_BLS381_sqr` (`FP_BLS381 *x, FP_BLS381 *y`)  
*Fast Modular squaring of an FP, mod Modulus.*
- void `FP_BLS381_add` (`FP_BLS381 *x, FP_BLS381 *y, FP_BLS381 *z`)  
*Modular addition of two FPs, mod Modulus.*
- void `FP_BLS381_sub` (`FP_BLS381 *x, FP_BLS381 *y, FP_BLS381 *z`)  
*Modular subtraction of two FPs, mod Modulus.*
- void `FP_BLS381_div2` (`FP_BLS381 *x, FP_BLS381 *y`)  
*Modular division by 2 of an FP, mod Modulus.*
- void `FP_BLS381_pow` (`FP_BLS381 *x, FP_BLS381 *y, BIG_384_58 z`)  
*Fast Modular exponentiation of an FP, to the power of a BIG, mod Modulus.*
- void `FP_BLS381_sqrt` (`FP_BLS381 *x, FP_BLS381 *y`)  
*Fast Modular square root of a an FP, mod Modulus.*
- void `FP_BLS381_neg` (`FP_BLS381 *x, FP_BLS381 *y`)  
*Modular negation of a an FP, mod Modulus.*
- void `FP_BLS381_output` (`FP_BLS381 *x`)

- Outputs an FP number to the console.*

  - void `FP_BLS381_rawoutput` (`FP_BLS381 *x`)

*Outputs an FP number to the console, in raw form.*
- void `FP_BLS381_reduce` (`FP_BLS381 *x`)

*Reduces possibly unreduced FP mod Modulus.*
- void `FP_BLS381_norm` (`FP_BLS381 *x`)

*normalizes FP*
- int `FP_BLS381_qr` (`FP_BLS381 *x`)

*Tests for FP a quadratic residue mod Modulus.*
- void `FP_BLS381_inv` (`FP_BLS381 *x`, `FP_BLS381 *y`)

*Modular inverse of a an FP, mod Modulus.*

## Variables

- const `BIG_384_58 Modulus_BLS381`
- const `BIG_384_58 R2modp_BLS381`
- const `chunk MConst_BLS381`

## 8.20.1 Detailed Description

### Author

Mike Scott

## 8.20.2 Macro Definition Documentation

### 8.20.2.1 FEXCESS\_BLS381

```
#define FEXCESS_BLS381 (((sign32)1)<<MAXXES_BLS381)-1)
```

$2^{(BASEBITS*NLEN-MODBITS)}-1$  - normalised BIG can be multiplied by less than this before reduction

### 8.20.2.2 MODBITS\_BLS381

```
#define MODBITS_BLS381 MBITS_BLS381
```

Number of bits in Modulus for selected curve

### 8.20.2.3 OMASK\_BLS381

```
#define OMASK_BLS381 (-((chunk)(1)<<TBITS_BLS381))
```

for masking out overflow bits

### 8.20.2.4 TBITS\_BLS381

```
#define TBITS_BLS381 (MBITS_BLS381%BASEBITS_384_58)
```

Number of active bits in top word

### 8.20.2.5 TMASK\_BLS381

```
#define TMASK_BLS381 (((chunk)1<<TBITS_BLS381)-1)
```

Mask for active bits in top word

## 8.20.3 Function Documentation

### 8.20.3.1 FP\_BLS381\_add()

```
void FP_BLS381_add (
    FP_BLS381 * x,
    FP_BLS381 * y,
    FP_BLS381 * z )
```

#### Parameters

<i>x</i>	FP number, on exit the modular sum = $y+z \bmod \text{Modulus}$
<i>y</i>	FP number
<i>z</i>	FP number

### 8.20.3.2 FP\_BLS381\_cmove()

```
void FP_BLS381_cmove (
    FP_BLS381 * x,
    FP_BLS381 * y,
    int s )
```

Conditionally copies second parameter to the first (without branching)

#### Parameters

<i>x</i>	an FP number
<i>y</i>	another FP number
<i>s</i>	copy takes place if not equal to 0

### 8.20.3.3 FP\_BLS381\_copy()

```
void FP_BLS381_copy (
    FP_BLS381 * y,
    FP_BLS381 * x )
```

#### Parameters

<i>y</i>	FP number to be copied to
<i>x</i>	FP to be copied from

### 8.20.3.4 FP\_BLS381\_cswap()

```
void FP_BLS381_cswap (
    FP_BLS381 * x,
    FP_BLS381 * y,
    int s )
```

Conditionally swaps parameters in constant time (without branching)

#### Parameters

<i>x</i>	an FP number
<i>y</i>	another FP number
<i>s</i>	swap takes place if not equal to 0

### 8.20.3.5 FP\_BLS381\_div2()

```
void FP_BLS381_div2 (
    FP_BLS381 * x,
    FP_BLS381 * y )
```

#### Parameters

<i>x</i>	FP number, on exit $=y/2 \bmod \text{Modulus}$
<i>y</i>	FP number

### 8.20.3.6 FP\_BLS381\_equals()

```
int FP_BLS381_equals (
    FP_BLS381 * x,
    FP_BLS381 * y )
```

## Parameters

$x$	FP number
$y$	FP number

## Returns

1 if equal, else returns 0

## 8.20.3.7 FP\_BLS381\_imul()

```
void FP_BLS381_imul (
    FP_BLS381 * x,
    FP_BLS381 * y,
    int i )
```

## Parameters

$x$	FP number, on exit the modular product = $y*i \bmod \text{Modulus}$
$y$	FP number, the multiplicand
$i$	a small number, the multiplier

## 8.20.3.8 FP\_BLS381\_inv()

```
void FP_BLS381_inv (
    FP_BLS381 * x,
    FP_BLS381 * y )
```

## Parameters

$x$	FP number, on exit = $1/y \bmod \text{Modulus}$
$y$	FP number

## 8.20.3.9 FP\_BLS381\_iszilch()

```
int FP_BLS381_iszilch (
    FP_BLS381 * x )
```

## Parameters

$x$	BIG number to be tested
-----	-------------------------

**Returns**

1 if zero, else returns 0

**8.20.3.10 FP\_BLS381\_mod()**

```
void FP_BLS381_mod (
    BIG_384_58 r,
    DBIG_384_58 d )
```

This function comes in different flavours depending on the form of Modulus that is currently in use.

**Parameters**

<i>r</i>	BIG number, on exit = $d \bmod \text{Modulus}$
<i>d</i>	DBIG number to be reduced

**8.20.3.11 FP\_BLS381\_mul()**

```
void FP_BLS381_mul (
    FP_BLS381 * x,
    FP_BLS381 * y,
    FP_BLS381 * z )
```

Uses appropriate fast modular reduction method

**Parameters**

<i>x</i>	FP number, on exit the modular product = $y*z \bmod \text{Modulus}$
<i>y</i>	FP number, the multiplicand
<i>z</i>	FP number, the multiplier

**8.20.3.12 FP\_BLS381\_neg()**

```
void FP_BLS381_neg (
    FP_BLS381 * x,
    FP_BLS381 * y )
```

**Parameters**

<i>x</i>	FP number, on exit = $-y \bmod \text{Modulus}$
<i>y</i>	FP number

### 8.20.3.13 FP\_BLS381\_norm()

```
void FP_BLS381_norm (
    FP_BLS381 * x )
```

#### Parameters

<i>x</i>	FP number, on exit normalized
----------	-------------------------------

### 8.20.3.14 FP\_BLS381\_nres()

```
void FP_BLS381_nres (
    FP_BLS381 * y,
    BIG_384_58 x )
```

#### Parameters

<i>x</i>	BIG number to be converted
<i>y</i>	FP result

### 8.20.3.15 FP\_BLS381\_one()

```
void FP_BLS381_one (
    FP_BLS381 * x )
```

#### Parameters

<i>x</i>	FP number to be set equal to unity.
----------	-------------------------------------

### 8.20.3.16 FP\_BLS381\_output()

```
void FP_BLS381_output (
    FP_BLS381 * x )
```

Converts from residue form before output

#### Parameters

<i>x</i>	an FP number
----------	--------------

**8.20.3.17 FP\_BLS381\_pow()**

```
void FP_BLS381_pow (
    FP_BLS381 * x,
    FP_BLS381 * y,
    BIG_384_58 z )
```

**Parameters**

<i>x</i>	FP number, on exit = $y^z \bmod \text{Modulus}$
<i>y</i>	FP number
<i>z</i>	BIG number exponent

**8.20.3.18 FP\_BLS381\_qr()**

```
int FP_BLS381_qr (
    FP_BLS381 * x )
```

**Parameters**

<i>x</i>	FP number to be tested
----------	------------------------

**Returns**

1 if quadratic residue, else returns 0 if quadratic non-residue

**8.20.3.19 FP\_BLS381\_rawoutput()**

```
void FP_BLS381_rawoutput (
    FP_BLS381 * x )
```

**Parameters**

<i>x</i>	a BIG number
----------	--------------

**8.20.3.20 FP\_BLS381\_rcopy()**

```
void FP_BLS381_rcopy (
```



```

    FP_BLS381 * y,
    const BIG_384_58 x )

```

**Parameters**

<i>y</i>	FP number to be copied to
<i>x</i>	BIG to be copied from ROM

**8.20.3.21 FP\_BLS381\_redc()**

```

void FP_BLS381_redc (
    BIG_384_58 x,
    FP_BLS381 * y )

```

**Parameters**

<i>y</i>	FP number to be converted to BIG
<i>x</i>	BIG result

**8.20.3.22 FP\_BLS381\_reduce()**

```

void FP_BLS381_reduce (
    FP_BLS381 * x )

```

**Parameters**

<i>x</i>	FP number, on exit reduced mod Modulus
----------	--

**8.20.3.23 FP\_BLS381\_sqr()**

```

void FP_BLS381_sqr (
    FP_BLS381 * x,
    FP_BLS381 * y )

```

Uses appropriate fast modular reduction method

**Parameters**

<i>x</i>	FP number, on exit the modular product = $y^2 \bmod \text{Modulus}$
<i>y</i>	FP number, the number to be squared

### 8.20.3.24 FP\_BLS381\_sqrt()

```
void FP_BLS381_sqrt (
    FP_BLS381 * x,
    FP_BLS381 * y )
```

#### Parameters

<i>x</i>	FP number, on exit = sqrt(y) mod Modulus
<i>y</i>	FP number, the number whose square root is calculated

### 8.20.3.25 FP\_BLS381\_sub()

```
void FP_BLS381_sub (
    FP_BLS381 * x,
    FP_BLS381 * y,
    FP_BLS381 * z )
```

#### Parameters

<i>x</i>	FP number, on exit the modular difference = y-z mod Modulus
<i>y</i>	FP number
<i>z</i>	FP number

### 8.20.3.26 FP\_BLS381\_zero()

```
void FP_BLS381_zero (
    FP_BLS381 * x )
```

#### Parameters

<i>x</i>	FP number to be set to 0
----------	--------------------------

## 8.20.4 Variable Documentation

### 8.20.4.1 MConst\_BLS381

```
const chunk MConst_BLS381
```

Constant associated with Modulus - for Montgomery =  $1/p \text{ mod } 2^{\wedge} \text{BASEBITS}$

## 8.20.4.2 Modulus\_BLS381

```
const BIG_384_58 Modulus_BLS381
```

Actual Modulus set in romf\_yyy.c

## 8.20.4.3 R2modp\_BLS381

```
const BIG_384_58 R2modp_BLS381
```

Montgomery constant

## 8.21 mpin\_BLS381.h File Reference

M-Pin Header file.

```
#include "pair_BLS381.h"
#include "pbc_support.h"
```

## Macros

- #define PGS\_BLS381 MODBYTES\_384\_58
- #define PFS\_BLS381 MODBYTES\_384\_58
- #define MPIN\_OK 0
- #define MPIN\_INVALID\_POINT -14
- #define MPIN\_BAD\_PIN -19
- #define MPIN\_PAS 16
- #define MAXPIN 10000
- #define PLEN 14
- #define MESSAGE\_SIZE 256
- #define M\_SIZE\_BLS381 (MESSAGE\_SIZE+2\*PFS\_BLS381+1)

## Functions

- void MPIN\_BLS381\_GET\_Y (int h, int t, `octet` \*O, `octet` \*Y)  
*Generate  $Y=H(s,O)$ , where  $s$  is epoch time,  $O$  is an octet, and  $H(.)$  is a hash function.*
- int MPIN\_BLS381\_EXTRACT\_FACTOR (int h, `octet` \*ID, int factor, int facbits, `octet` \*CS)  
*Extract a PIN number from a client secret.*
- int MPIN\_BLS381\_RESTORE\_FACTOR (int h, `octet` \*ID, int factor, int facbits, `octet` \*CS)  
*Extract a PIN number from a client secret.*
- int MPIN\_BLS381\_EXTRACT\_PIN (int h, `octet` \*ID, int pin, `octet` \*CS)  
*Extract a PIN number from a client secret.*
- int MPIN\_BLS381\_CLIENT (int h, int d, `octet` \*ID, `csprng` \*R, `octet` \*x, int pin, `octet` \*T, `octet` \*V, `octet` \*U, `octet` \*UT, `octet` \*TP, `octet` \*MESSAGE, int t, `octet` \*y)  
*Perform client side of the one-pass version of the M-Pin protocol.*
- int MPIN\_BLS381\_CLIENT\_1 (int h, int d, `octet` \*ID, `csprng` \*R, `octet` \*x, int pin, `octet` \*T, `octet` \*S, `octet` \*U, `octet` \*UT, `octet` \*TP)

- Perform first pass of the client side of the 3-pass version of the M-Pin protocol.*

  - int `MPIN_BLS381_RANDOM_GENERATE` (csprng \*R, octet \*S)

*Generate a random group element.*
- int `MPIN_BLS381_CLIENT_2` (octet \*x, octet \*y, octet \*V)

*Perform second pass of the client side of the 3-pass version of the M-Pin protocol.*
- int `MPIN_BLS381_SERVER` (int h, int d, octet \*HID, octet \*HTID, octet \*y, octet \*SS, octet \*U, octet \*UT, octet \*V, octet \*E, octet \*F, octet \*ID, octet \*MESSAGE, int t, octet \*Pa)

*Perform server side of the one-pass version of the M-Pin protocol.*
- void `MPIN_BLS381_SERVER_1` (int h, int d, octet \*ID, octet \*HID, octet \*HTID)

*Perform first pass of the server side of the 3-pass version of the M-Pin protocol.*
- int `MPIN_BLS381_SERVER_2` (int d, octet \*HID, octet \*HTID, octet \*y, octet \*SS, octet \*U, octet \*UT, octet \*V, octet \*E, octet \*F, octet \*Pa)

*Perform third pass on the server side of the 3-pass version of the M-Pin protocol.*
- int `MPIN_BLS381_RECOMBINE_G1` (octet \*Q1, octet \*Q2, octet \*Q)

*Add two members from the group G1.*
- int `MPIN_BLS381_RECOMBINE_G2` (octet \*P1, octet \*P2, octet \*P)

*Add two members from the group G2.*
- int `MPIN_BLS381_KANGAROO` (octet \*E, octet \*F)

*Use Kangaroos to find PIN error.*
- int `MPIN_BLS381_ENCODING` (csprng \*R, octet \*TP)

*Encoding of a Time Permit to make it indistinguishable from a random string.*
- int `MPIN_BLS381_DECODING` (octet \*TP)

*Encoding of an obfuscated Time Permit.*
- int `MPIN_BLS381_GET_G1_MULTIPLE` (csprng \*R, int type, octet \*x, octet \*G, octet \*W)

*Find a random multiple of a point in G1.*
- int `MPIN_BLS381_GET_G2_MULTIPLE` (csprng \*R, int type, octet \*x, octet \*G, octet \*W)

*Find a random multiple of a point in G1.*
- int `MPIN_BLS381_GET_CLIENT_SECRET` (octet \*S, octet \*ID, octet \*CS)

*Create a client secret in G1 from a master secret and the client ID.*
- int `MPIN_BLS381_GET_CLIENT_PERMIT` (int h, int d, octet \*S, octet \*ID, octet \*TP)

*Create a Time Permit in G1 from a master secret and the client ID.*
- int `MPIN_BLS381_GET_SERVER_SECRET` (octet \*S, octet \*SS)

*Create a server secret in G2 from a master secret.*
- int `MPIN_BLS381_PRECOMPUTE` (octet \*T, octet \*ID, octet \*CP, octet \*g1, octet \*g2)

*Precompute values for use by the client side of M-Pin Full.*
- int `MPIN_BLS381_SERVER_KEY` (int h, octet \*Z, octet \*SS, octet \*w, octet \*p, octet \*l, octet \*U, octet \*UT, octet \*K)

*Calculate Key on Server side for M-Pin Full.*
- int `MPIN_BLS381_CLIENT_KEY` (int h, octet \*g1, octet \*g2, int pin, octet \*r, octet \*x, octet \*p, octet \*T, octet \*K)

*Calculate Key on Client side for M-Pin Full.*
- int `MPIN_BLS381_GET_DVS_KEYPAIR` (csprng \*R, octet \*Z, octet \*Pa)

*Generates a random public key for the client z.Q.*

### 8.21.1 Detailed Description

#### Author

Mike Scott

## 8.21.2 Macro Definition Documentation

### 8.21.2.1 M\_SIZE\_BLS381

```
#define M_SIZE_BLS381 (MESSAGE_SIZE+2*PFS_BLS381+1)
```

Signature message size and G1 size

### 8.21.2.2 MAXPIN

```
#define MAXPIN 10000
```

max PIN

### 8.21.2.3 MESSAGE\_SIZE

```
#define MESSAGE_SIZE 256
```

Signature message size

### 8.21.2.4 MPIN\_BAD\_PIN

```
#define MPIN_BAD_PIN -19
```

Bad PIN number entered

### 8.21.2.5 MPIN\_INVALID\_POINT

```
#define MPIN_INVALID_POINT -14
```

Point is NOT on the curve

### 8.21.2.6 MPIN\_OK

```
#define MPIN_OK 0
```

Function completed without error

### 8.21.2.7 MPIN\_PAS

```
#define MPIN_PAS 16
```

MPIN Symmetric Key Size

### 8.21.2.8 PBLLEN

```
#define PBLLEN 14
```

max length of PIN in bits

### 8.21.2.9 PFS\_BLS381

```
#define PFS_BLS381 MODBYTES_384_58
```

MPIN Field Size

### 8.21.2.10 PGS\_BLS381

```
#define PGS_BLS381 MODBYTES_384_58
```

MPIN Group Size

## 8.21.3 Function Documentation

### 8.21.3.1 MPIN\_BLS381\_CLIENT()

```
int MPIN_BLS381_CLIENT (
    int h,
    int d,
    octet * ID,
    csprng * R,
    octet * x,
    int pin,
    octet * T,
    octet * V,
    octet * U,
    octet * UT,
    octet * TP,
    octet * MESSAGE,
    int t,
    octet * y )
```

If Time Permits are disabled, set  $d = 0$ , and  $UT$  is not generated and can be set to `NULL`. If Time Permits are enabled, and PIN error detection is OFF,  $U$  is not generated and can be set to `NULL`. If Time Permits are enabled, and PIN error detection is ON,  $U$  and  $UT$  are both generated.

#### Parameters

$h$	is the hash type
$d$	is input date, in days since the epoch. Set to 0 if Time permits disabled
$ID$	is the input client identity
$R$	is a pointer to a cryptographically secure random number generator

## Parameters

<i>x</i>	an output internally randomly generated if R!=NULL, otherwise must be provided as an input
<i>pin</i>	is the input PIN number
<i>T</i>	is the input M-Pin token (the client secret with PIN portion removed)
<i>V</i>	is output = $-(x+y)(CS+TP)$ , where CS is the reconstructed client secret, and TP is the time permit
<i>U</i>	is output = $x.H(ID)$
<i>UT</i>	is output = $x.(H(ID)+H(d H(ID)))$
<i>TP</i>	is the input time permit
<i>MESSAGE</i>	is the message to be signed
<i>t</i>	is input epoch time in seconds - a timestamp
<i>y</i>	is output $H(t U)$ or $H(t UT)$ if Time Permits enabled

## Returns

0 or an error code

## 8.21.3.2 MPIN\_BLS381\_CLIENT\_1()

```
int MPIN_BLS381_CLIENT_1 (
    int h,
    int d,
    octet * ID,
    csprng * R,
    octet * x,
    int pin,
    octet * T,
    octet * S,
    octet * U,
    octet * UT,
    octet * TP )
```

If Time Permits are disabled, set  $d = 0$ , and  $UT$  is not generated and can be set to NULL. If Time Permits are enabled, and PIN error detection is OFF,  $U$  is not generated and can be set to NULL. If Time Permits are enabled, and PIN error detection is ON,  $U$  and  $UT$  are both generated.

## Parameters

<i>h</i>	is the hash type
<i>d</i>	is input date, in days since the epoch. Set to 0 if Time permits disabled
<i>ID</i>	is the input client identity
<i>R</i>	is a pointer to a cryptographically secure random number generator
<i>x</i>	an output internally randomly generated if R!=NULL, otherwise must be provided as an input
<i>pin</i>	is the input PIN number
<i>T</i>	is the input M-Pin token (the client secret with PIN portion removed)
<i>S</i>	is output = $CS+TP$ , where CS= is the reconstructed client secret, and TP is the time permit
<i>U</i>	is output = $x.H(ID)$
<i>UT</i>	is output = $x.(H(ID)+H(d H(ID)))$
<i>TP</i>	is the input time permit

**Returns**

0 or an error code

**8.21.3.3 MPIN\_BLS381\_CLIENT\_2()**

```
int MPIN_BLS381_CLIENT_2 (
    octet * x,
    octet * y,
    octet * V )
```

**Parameters**

<i>x</i>	an input, a locally generated random number
<i>y</i>	an input random challenge from the server
<i>V</i>	on output = $-(x+y).V$

**Returns**

0 or an error code

**8.21.3.4 MPIN\_BLS381\_CLIENT\_KEY()**

```
int MPIN_BLS381_CLIENT_KEY (
    int h,
    octet * g1,
    octet * g2,
    int pin,
    octet * r,
    octet * x,
    octet * p,
    octet * T,
    octet * K )
```

**Parameters**

<i>h</i>	is the hash type
<i>g1</i>	precomputed input
<i>g2</i>	precomputed input
<i>pin</i>	is the input PIN number
<i>r</i>	is an input, a locally generated random number
<i>x</i>	is an input, a locally generated random number
<i>p</i>	is an input, hash of the protocol transcript
<i>T</i>	is the input Server-side Diffie-Hellman component
<i>K</i>	is the output calculated shared key



**Returns**

0 or an error code

**8.21.3.5 MPIN\_BLS381\_DECODING()**

```
int MPIN_BLS381_DECODING (  
    octet * TP )
```

**Parameters**

<i>TP</i>	is the input obfuscated time permit, restored on output
-----------	---

**Returns**

0 or an error code

**8.21.3.6 MPIN\_BLS381\_ENCODING()**

```
int MPIN_BLS381_ENCODING (  
    csprng * R,  
    octet * TP )
```

**Parameters**

<i>R</i>	is a pointer to a cryptographically secure random number generator
<i>TP</i>	is the input time permit, obfuscated on output

**Returns**

0 or an error code

**8.21.3.7 MPIN\_BLS381\_EXTRACT\_FACTOR()**

```
int MPIN_BLS381_EXTRACT_FACTOR (  
    int h,  
    octet * ID,  
    int factor,  
    int facbits,  
    octet * CS )
```

## Parameters

<i>h</i>	is the hash type
<i>ID</i>	is the input client identity
<i>factor</i>	is an input factor
<i>facbits</i>	is the number of bits in the factor
<i>CS</i>	is the client secret from which the factor is to be extracted

## Returns

0 or an error code

## 8.21.3.8 MPIN\_BLS381\_EXTRACT\_PIN()

```
int MPIN_BLS381_EXTRACT_PIN (
    int h,
    octet * ID,
    int pin,
    octet * CS )
```

## Parameters

<i>h</i>	is the hash type
<i>ID</i>	is the input client identity
<i>pin</i>	is an input PIN number
<i>CS</i>	is the client secret from which the PIN is to be extracted

## Returns

0 or an error code

## 8.21.3.9 MPIN\_BLS381\_GET\_CLIENT\_PERMIT()

```
int MPIN_BLS381_GET_CLIENT_PERMIT (
    int h,
    int d,
    octet * S,
    octet * ID,
    octet * TP )
```

## Parameters

<i>h</i>	is the hash type
<i>d</i>	is input date, in days since the epoch.
<i>S</i>	is an input master secret
<i>ID</i>	is the input client identity
<i>TP</i>	is a Time Permit for the given date = s.H(d H(ID))

**Returns**

0 or an error code

**8.21.3.10 MPIN\_BLS381\_GET\_CLIENT\_SECRET()**

```
int MPIN_BLS381_GET_CLIENT_SECRET (
    octet * S,
    octet * ID,
    octet * CS )
```

**Parameters**

<i>S</i>	is an input master secret
<i>ID</i>	is the input client identity
<i>CS</i>	is the full client secret = s.H(ID)

**Returns**

0 or an error code

**8.21.3.11 MPIN\_BLS381\_GET\_DVS\_KEYPAIR()**

```
int MPIN_BLS381_GET_DVS_KEYPAIR (
    csprng * R,
    octet * Z,
    octet * Pa )
```

**Parameters**

<i>R</i>	is a pointer to a cryptographically secure random number generator
<i>Z</i>	an output internally randomly generated if R!=NULL, otherwise it must be provided as an input
<i>Pa</i>	the output public key for the client

**8.21.3.12 MPIN\_BLS381\_GET\_G1\_MULTIPLE()**

```
int MPIN_BLS381_GET_G1_MULTIPLE (
    csprng * R,
    int type,
    octet * x,
    octet * G,
    octet * W )
```

## Parameters

<i>R</i>	is a pointer to a cryptographically secure random number generator
<i>type</i>	determines type of action to be taken
<i>x</i>	an output internally randomly generated if $R \neq \text{NULL}$ , otherwise must be provided as an input
<i>G</i>	if $\text{type}=0$ a point in $G_1$ , else an octet to be mapped to $G_1$
<i>W</i>	the output $=x.G$ or $x.M(G)$ , where $M(.)$ is a mapping

## Returns

0 or an error code

## 8.21.3.13 MPIN\_BLS381\_GET\_G2\_MULTIPLE()

```
int MPIN_BLS381_GET_G2_MULTIPLE (
    csprng * R,
    int type,
    octet * x,
    octet * G,
    octet * W )
```

## Parameters

<i>R</i>	is a pointer to a cryptographically secure random number generator
<i>type</i>	determines type of action to be taken
<i>x</i>	an output internally randomly generated if $R \neq \text{NULL}$ , otherwise must be provided as an input
<i>G</i>	a point in $G_2$
<i>W</i>	the output $=x.G$ or $(1/x).G$

## Returns

0 or an error code

## 8.21.3.14 MPIN\_BLS381\_GET\_SERVER\_SECRET()

```
int MPIN_BLS381_GET_SERVER_SECRET (
    octet * S,
    octet * SS )
```

## Parameters

<i>S</i>	is an input master secret
<i>SS</i>	is the server secret $= s.Q$ where $Q$ is a fixed generator of $G_2$

**Returns**

0 or an error code

**8.21.3.15 MPIN\_BLS381\_GET\_Y()**

```
void MPIN_BLS381_GET_Y (
    int h,
    int t,
    octet * O,
    octet * Y )
```

**Parameters**

<i>h</i>	is the hash type
<i>t</i>	is epoch time in seconds
<i>O</i>	is an input octet
<i>Y</i>	is the output octet

**8.21.3.16 MPIN\_BLS381\_KANGAROO()**

```
int MPIN_BLS381_KANGAROO (
    octet * E,
    octet * F )
```

**Parameters**

<i>E</i>	a member of the group GT
<i>F</i>	a member of the group $GT = E^e$

**Returns**

0 if Kangaroos failed, or the PIN error e

**8.21.3.17 MPIN\_BLS381\_PRECOMPUTE()**

```
int MPIN_BLS381_PRECOMPUTE (
    octet * T,
    octet * ID,
    octet * CP,
    octet * g1,
    octet * g2 )
```

**Parameters**

<i>T</i>	is the input M-Pin token (the client secret with PIN portion removed)
<i>ID</i>	is the input client identity
<i>CP</i>	is Public Key (or NULL)
<i>g1</i>	precomputed output
<i>g2</i>	precomputed output

**Returns**

0 or an error code

**8.21.3.18 MPIN\_BLS381\_RANDOM\_GENERATE()**

```
int MPIN_BLS381_RANDOM_GENERATE (
    csprng * R,
    octet * S )
```

**Parameters**

<i>R</i>	is a pointer to a cryptographically secure random number generator
<i>S</i>	is the output random octet

**Returns**

0 or an error code

**8.21.3.19 MPIN\_BLS381\_RECOMBINE\_G1()**

```
int MPIN_BLS381_RECOMBINE_G1 (
    octet * Q1,
    octet * Q2,
    octet * Q )
```

**Parameters**

<i>Q1</i>	an input member of G1
<i>Q2</i>	an input member of G1
<i>Q</i>	an output member of G1 = Q1+Q2

**Returns**

0 or an error code

## 8.21.3.20 MPIN\_BLS381\_RECOMBINE\_G2()

```
int MPIN_BLS381_RECOMBINE_G2 (
    octet * P1,
    octet * P2,
    octet * P )
```

## Parameters

<i>P1</i>	an input member of G2
<i>P2</i>	an input member of G2
<i>P</i>	an output member of G2 = P1+P2

## Returns

0 or an error code

## 8.21.3.21 MPIN\_BLS381\_RESTORE\_FACTOR()

```
int MPIN_BLS381_RESTORE_FACTOR (
    int h,
    octet * ID,
    int factor,
    int facbits,
    octet * CS )
```

## Parameters

<i>h</i>	is the hash type
<i>ID</i>	is the input client identity
<i>factor</i>	is an input factor
<i>facbits</i>	is the number of bits in the factor
<i>CS</i>	is the client secret to which the factor is to be added

## Returns

0 or an error code

## 8.21.3.22 MPIN\_BLS381\_SERVER()

```
int MPIN_BLS381_SERVER (
    int h,
    int d,
    octet * HID,
    octet * HTID,
```

```

    octet * y,
    octet * SS,
    octet * U,
    octet * UT,
    octet * V,
    octet * E,
    octet * F,
    octet * ID,
    octet * MESSAGE,
    int t,
    octet * Pa )

```

If Time Permits are disabled, set  $d = 0$ , and UT and HTID are not generated and can be set to NULL. If Time Permits are enabled, and PIN error detection is OFF, U and HID are not needed and can be set to NULL. If Time Permits are enabled, and PIN error detection is ON, U, UT, HID and HTID are all required.

#### Parameters

<i>h</i>	is the hash type
<i>d</i>	is input date, in days since the epoch. Set to 0 if Time permits disabled
<i>HID</i>	is output H(ID), a hash of the client ID
<i>HTID</i>	is output H(ID)+H(d H(ID))
<i>y</i>	is output H(t U) or H(t UT) if Time Permits enabled
<i>SS</i>	is the input server secret
<i>U</i>	is input from the client = x.H(ID)
<i>UT</i>	is input from the client= x.(H(ID)+H(d H(ID)))
<i>V</i>	is an input from the client
<i>E</i>	is an output to help the Kangaroos to find the PIN error, or NULL if not required
<i>F</i>	is an output to help the Kangaroos to find the PIN error, or NULL if not required
<i>ID</i>	is the input claimed client identity
<i>MESSAGE</i>	is the message to be signed
<i>t</i>	is input epoch time in seconds - a timestamp
<i>Pa</i>	is input from the client z.Q or NULL if the key-escrow less scheme is not used

#### Returns

0 or an error code

#### 8.21.3.23 MPIN\_BLS381\_SERVER\_1()

```

void MPIN_BLS381_SERVER_1 (
    int h,
    int d,
    octet * ID,
    octet * HID,
    octet * HTID )

```



## Parameters

<i>h</i>	is the hash type
<i>d</i>	is input date, in days since the epoch. Set to 0 if Time permits disabled
<i>ID</i>	is the input claimed client identity
<i>HID</i>	is output H(ID), a hash of the client ID
<i>HTID</i>	is output H(ID)+H(d H(ID))

## Returns

0 or an error code

## 8.21.3.24 MPIN\_BLS381\_SERVER\_2()

```
int MPIN_BLS381_SERVER_2 (
    int d,
    octet * HID,
    octet * HTID,
    octet * y,
    octet * SS,
    octet * U,
    octet * UT,
    octet * V,
    octet * E,
    octet * F,
    octet * Pa )
```

If Time Permits are disabled, set  $d = 0$ , and  $UT$  and  $HTID$  are not needed and can be set to  $NULL$ . If Time Permits are enabled, and PIN error detection is OFF,  $U$  and  $HID$  are not needed and can be set to  $NULL$ . If Time Permits are enabled, and PIN error detection is ON,  $U$ ,  $UT$ ,  $HID$  and  $HTID$  are all required.

## Parameters

<i>d</i>	is input date, in days since the epoch. Set to 0 if Time permits disabled
<i>HID</i>	is input H(ID), a hash of the client ID
<i>HTID</i>	is input H(ID)+H(d H(ID))
<i>y</i>	is the input server's randomly generated challenge
<i>SS</i>	is the input server secret
<i>U</i>	is input from the client = $x.H(ID)$
<i>UT</i>	is input from the client = $x.(H(ID)+H(d H(ID)))$
<i>V</i>	is an input from the client
<i>E</i>	is an output to help the Kangaroos to find the PIN error, or $NULL$ if not required
<i>F</i>	is an output to help the Kangaroos to find the PIN error, or $NULL$ if not required
<i>Pa</i>	is the input public key from the client, $z.Q$ or $NULL$ if the client uses regular mpin

**Returns**

0 or an error code

**8.21.3.25 MPIN\_BLS381\_SERVER\_KEY()**

```
int MPIN_BLS381_SERVER_KEY (
    int h,
    octet * Z,
    octet * SS,
    octet * w,
    octet * p,
    octet * I,
    octet * U,
    octet * UT,
    octet * K )
```

Uses UT internally for the key calculation, unless not available in which case U is used

**Parameters**

<i>h</i>	is the hash type
<i>Z</i>	is the input Client-side Diffie-Hellman component
<i>SS</i>	is the input server secret
<i>w</i>	is an input random number generated by the server
<i>p</i>	is an input, hash of the protocol transcript
<i>I</i>	is the hashed input client ID = H(ID)
<i>U</i>	is input from the client = x.H(ID)
<i>UT</i>	is input from the client = x.(H(ID)+H(d H(ID)))
<i>K</i>	is the output calculated shared key

**Returns**

0 or an error code

**8.22 paillier.h File Reference**

Paillier declarations.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "ff_4096.h"
#include "ff_2048.h"
```

## Data Structures

- struct [PAILLIER\\_public\\_key](#)  
*Paillier Public Key.*
- struct [PAILLIER\\_private\\_key](#)  
*Paillier Private Key.*

## Macros

- #define [FS\\_4096](#) `MODBYTES_512_60*FFLEN_4096`
- #define [FS\\_2048](#) `MODBYTES_1024_58*FFLEN_2048`
- #define [HFS\\_4096](#) `MODBYTES_512_60*HFLEN_4096`
- #define [HFS\\_2048](#) `MODBYTES_1024_58*HFLEN_2048`

## Functions

- void [PAILLIER\\_KEY\\_PAIR](#) (`csprng *RNG, octet *P, octet *Q, PAILLIER_public_key *PUB, PAILLIER_private_key *PRIV`)  
*Generate the key pair.*
- void [PAILLIER\\_PRIVATE\\_KEY\\_KILL](#) (`PAILLIER_private_key *PRIV`)  
*Clear private key.*
- void [PAILLIER\\_ENCRYPT](#) (`csprng *RNG, PAILLIER_public_key *PUB, octet *PT, octet *CT, octet *R`)  
*Encrypt a plaintext.*
- void [PAILLIER\\_DECRYPT](#) (`PAILLIER_private_key *PRIV, octet *CT, octet *PT`)  
*Decrypt ciphertext.*
- void [PAILLIER\\_ADD](#) (`PAILLIER_public_key *PUB, octet *CT1, octet *CT2, octet *CT`)  
*Homomorphic addition of plaintexts.*
- void [PAILLIER\\_MULT](#) (`PAILLIER_public_key *PUB, octet *CT1, octet *PT, octet *CT`)  
*Homomorphic multiplication of plaintexts.*
- void [PAILLIER\\_PK\\_fromOctet](#) (`PAILLIER_public_key *PUB, octet *PK`)  
*Read a public key from its octet representation.*
- void [PAILLIER\\_PK\\_toOctet](#) (`octet *PK, PAILLIER_public_key *PUB`)  
*Write a public key to an octet.*

### 8.22.1 Macro Definition Documentation

#### 8.22.1.1 FS\_2048

```
#define FS_2048 MODBYTES_1024_58*FFLEN_2048
```

2048 field size in bytes

#### 8.22.1.2 FS\_4096

```
#define FS_4096 MODBYTES_512_60*FFLEN_4096
```

4096 field size in bytes

### 8.22.1.3 HFS\_2048

```
#define HFS_2048 MODBYTES_1024_58*HFLEN_2048
```

Half 2048 field size in bytes

### 8.22.1.4 HFS\_4096

```
#define HFS_4096 MODBYTES_512_60*HFLEN_4096
```

Half 4096 field size in bytes

## 8.22.2 Function Documentation

### 8.22.2.1 PAILLIER\_ADD()

```
void PAILLIER_ADD (
    PAILLIER_public_key * PUB,
    octet * CT1,
    octet * CT2,
    octet * CT )
```

$$E(m1 + m2) = E(m1) * E(m2)$$

1.  $ct = ct1 * ct2 \pmod{n^2}$

#### Parameters

<i>PUB</i>	Public key
<i>CT1</i>	Ciphertext one
<i>CT2</i>	Ciphertext two
<i>CT</i>	Ciphertext

#### Returns

Returns 0 or else error code

### 8.22.2.2 PAILLIER\_DECRYPT()

```
void PAILLIER_DECRYPT (
    PAILLIER_private_key * PRIV,
```

```

    octet * CT,
    octet * PT )

```

These are the decryption steps modulo  $n$ . The computations are carried out modulo  $p$  and  $q$  and combined using the CRT.

1.  $ctl = ct^l \pmod{n^2} - 1$
2.  $ctl_n = ctl/n$
3.  $pt = ctl_n * m \pmod{n}$

#### Parameters

$P_{\leftarrow}$ $RIV$	Private key
$CT$	Ciphertext
$PT$	Plaintext

#### 8.22.2.3 PAILLIER\_ENCRYPT()

```

void PAILLIER_ENCRYPT (
    csprng * RNG,
    PAILLIER_public_key * PUB,
    octet * PT,
    octet * CT,
    octet * R )

```

These are the encryption steps.

1.  $m < n$
2.  $r < n$
3.  $c = g^m . r^n \pmod{n^2}$

#### Parameters

$RNG$	Pointer to a cryptographically secure random number generator
$PUB$	Public key
$PT$	Plaintext
$CT$	Ciphertext
$R$	R value for testing. If $RNG$ is NULL then this value is read.

#### 8.22.2.4 PAILLIER\_KEY\_PAIR()

```

void PAILLIER_KEY_PAIR (

```

```

    csprng * RNG,
    octet * P,
    octet * Q,
    PAILLIER_public_key * PUB,
    PAILLIER_private_key * PRIV )

```

Pick large prime numbers of the same size  $p$  and  $q$

1.  $n = pq$
2.  $g = n + 1$
3.  $l = (p - 1)(q - 1)$
4.  $m = l^{-1} \pmod{n}$

#### Parameters

<i>RNG</i>	Pointer to a cryptographically secure random number generator
<i>P</i>	Prime number. If RNG is NULL then this value is read
<i>Q</i>	Prime number. If RNG is NULL then this value is read
<i>PUB</i>	Public key
<i>P<sub>←</sub></i> <i>RIV</i>	Private key

#### 8.22.2.5 PAILLIER\_MULT()

```

void PAILLIER_MULT (
    PAILLIER_public_key * PUB,
    octet * CT1,
    octet * PT,
    octet * CT )

```

$$E(m1 * m2) = E(m1)^{m2}$$

1.  $ct = ct1^{m2} \pmod{n^2}$

#### Parameters

<i>PUB</i>	Public key
<i>CT1</i>	Ciphertext one
<i>PT</i>	Plaintext constant
<i>CT</i>	Ciphertext

## 8.22.2.6 PAILLIER\_PK\_fromOctet()

```
void PAILLIER_PK_fromOctet (
    PAILLIER_public_key * PUB,
    octet * PK )
```

## Parameters

<i>PUB</i>	Public key
<i>PK</i>	Octet representation of the public key

## 8.22.2.7 PAILLIER\_PK\_toOctet()

```
void PAILLIER_PK_toOctet (
    octet * PK,
    PAILLIER_public_key * PUB )
```

## Parameters

<i>PK</i>	Destination octet
<i>PUB</i>	Public key

## 8.22.2.8 PAILLIER\_PRIVATE\_KEY\_KILL()

```
void PAILLIER_PRIVATE_KEY_KILL (
    PAILLIER_private_key * PRIV )
```

## Parameters

<i>P<sub>←</sub></i> <i>RIV</i>	Private key to clean
------------------------------------	----------------------

## 8.23 pair\_BLS381.h File Reference

PAIR Header File.

```
#include "fp12_BLS381.h"
#include "ecp2_BLS381.h"
#include "ecp_BLS381.h"
```

## Functions

- void `PAIR_BLS381_another` (`FP12_BLS381 r[]`, `ECP2_BLS381 *PV`, `ECP_BLS381 *QV`)  
*Precompute line functions for n-pairing.*
- void `PAIR_BLS381_ate` (`FP12_BLS381 *r`, `ECP2_BLS381 *P`, `ECP_BLS381 *Q`)  
*Calculate Miller loop for Optimal ATE pairing  $e(P,Q)$*
- void `PAIR_BLS381_double_ate` (`FP12_BLS381 *r`, `ECP2_BLS381 *P`, `ECP_BLS381 *Q`, `ECP2_BLS381 *R`, `ECP_BLS381 *S`)  
*Calculate Miller loop for Optimal ATE double-pairing  $e(P,Q).e(R,S)$*
- void `PAIR_BLS381_fexp` (`FP12_BLS381 *x`)  
*Final exponentiation of pairing, converts output of Miller loop to element in GT.*
- void `PAIR_BLS381_G1mul` (`ECP_BLS381 *Q`, `BIG_384_58 b`)  
*Fast point multiplication of a member of the group G1 by a BIG number.*
- void `PAIR_BLS381_G2mul` (`ECP2_BLS381 *P`, `BIG_384_58 b`)  
*Fast point multiplication of a member of the group G2 by a BIG number.*
- void `PAIR_BLS381_GTpow` (`FP12_BLS381 *x`, `BIG_384_58 b`)  
*Fast raising of a member of GT to a BIG power.*
- int `PAIR_BLS381_GTmember` (`FP12_BLS381 *x`)  
*Tests FP12 for membership of GT.*
- int `PAIR_BLS381_nbits` (`BIG_384_58 n3`, `BIG_384_58 n`)  
*Prepare Ate parameter.*
- void `PAIR_BLS381_initmp` (`FP12_BLS381 r[]`)  
*Initialise structure for multi-pairing.*
- void `PAIR_BLS381_miller` (`FP12_BLS381 *res`, `FP12_BLS381 r[]`)  
*Miller loop.*

## Variables

- const `BIG_384_58 CURVE_Bnx_BLS381`
- const `BIG_384_58 CURVE_Cru_BLS381`
- const `BIG_384_58 CURVE_W_BLS381 [2]`
- const `BIG_384_58 CURVE_SB_BLS381 [2][2]`
- const `BIG_384_58 CURVE_WB_BLS381 [4]`
- const `BIG_384_58 CURVE_BB_BLS381 [4][4]`

### 8.23.1 Detailed Description

#### Author

Mike Scott

### 8.23.2 Function Documentation

#### 8.23.2.1 PAIR\_BLS381\_another()

```
void PAIR_BLS381_another (
    FP12_BLS381 r[],
    ECP2_BLS381 * PV,
    ECP_BLS381 * QV )
```



## Parameters

<i>r</i>	array of precomputed FP12 products of line functions
<i>PV</i>	ECP2 instance, an element of G2
<i>QV</i>	ECP instance, an element of G1

## 8.23.2.2 PAIR\_BLS381\_ate()

```
void PAIR_BLS381_ate (
    FP12_BLS381 * r,
    ECP2_BLS381 * P,
    ECP_BLS381 * Q )
```

## Parameters

<i>r</i>	FP12 result of the pairing calculation $e(P,Q)$
<i>P</i>	ECP2 instance, an element of G2
<i>Q</i>	ECP instance, an element of G1

## 8.23.2.3 PAIR\_BLS381\_double\_ate()

```
void PAIR_BLS381_double_ate (
    FP12_BLS381 * r,
    ECP2_BLS381 * P,
    ECP_BLS381 * Q,
    ECP2_BLS381 * R,
    ECP_BLS381 * S )
```

Faster than calculating two separate pairings

## Parameters

<i>r</i>	FP12 result of the pairing calculation $e(P,Q).e(R,S)$ , an element of GT
<i>P</i>	ECP2 instance, an element of G2
<i>Q</i>	ECP instance, an element of G1
<i>R</i>	ECP2 instance, an element of G2
<i>S</i>	ECP instance, an element of G1

## 8.23.2.4 PAIR\_BLS381\_fexp()

```
void PAIR_BLS381_fexp (
    FP12_BLS381 * x )
```

Here  $p$  is the internal modulus, and  $r$  is the group order

## Parameters

$x$	FP12, on exit = $x^{((p^{12}-1)/r)}$
-----	--------------------------------------

## 8.23.2.5 PAIR\_BLS381\_G1mul()

```
void PAIR_BLS381_G1mul (
    ECP_BLS381 * Q,
    BIG_384_58 b )
```

May exploit endomorphism for speed.

## Parameters

$Q$	ECP member of G1.
$b$	BIG multiplier

## 8.23.2.6 PAIR\_BLS381\_G2mul()

```
void PAIR_BLS381_G2mul (
    ECP2_BLS381 * P,
    BIG_384_58 b )
```

May exploit endomorphism for speed.

## Parameters

$P$	ECP2 member of G1.
$b$	BIG multiplier

## 8.23.2.7 PAIR\_BLS381\_GTmember()

```
int PAIR_BLS381_GTmember (
    FP12_BLS381 * x )
```

## Parameters

$x$	FP12 instance
-----	---------------

**Returns**

1 if  $x$  is in GT, else return 0

**8.23.2.8 PAIR\_BLS381\_GTpow()**

```
void PAIR_BLS381_GTpow (
    FP12_BLS381 * x,
    BIG_384_58 b )
```

May exploit endomorphism for speed.

**Parameters**

$x$	FP12 member of GT.
$b$	BIG exponent

**8.23.2.9 PAIR\_BLS381\_initmp()**

```
void PAIR_BLS381_initmp (
    FP12_BLS381 r[] )
```

**Parameters**

$r$	FP12 array, to be initialised to 1
-----	------------------------------------

**8.23.2.10 PAIR\_BLS381\_miller()**

```
void PAIR_BLS381_miller (
    FP12_BLS381 * res,
    FP12_BLS381 r[] )
```

**Parameters**

$res$	FP12 result
$r$	FP12 precomputed array of accumulated line functions

**8.23.2.11 PAIR\_BLS381\_nbits()**

```
int PAIR_BLS381_nbits (
```

```

    BIG_384_58 n3,
    BIG_384_58 n )

```

**Parameters**

$n$	BIG parameter
$n3$	BIG paramter = $3*n$

**Returns**

number of nits in  $n3$

**8.23.3 Variable Documentation****8.23.3.1 CURVE\_BB\_BLS381**

```
const BIG_384_58 CURVE_BB_BLS381[4][4]
```

BN curve constant for GS decomposition

**8.23.3.2 CURVE\_Bnx\_BLS381**

```
const BIG_384_58 CURVE_Bnx_BLS381
```

BN curve x parameter

**8.23.3.3 CURVE\_Cru\_BLS381**

```
const BIG_384_58 CURVE_Cru_BLS381
```

BN curve Cube Root of Unity

**8.23.3.4 CURVE\_SB\_BLS381**

```
const BIG_384_58 CURVE_SB_BLS381[2][2]
```

BN curve constant for GLV decomposition

**8.23.3.5 CURVE\_W\_BLS381**

```
const BIG_384_58 CURVE_W_BLS381[2]
```

BN curve constant for GLV decomposition

### 8.23.3.6 CURVE\_WB\_BLS381

```
const BIG_384_58 CURVE_WB_BLS381[4]
```

BN curve constant for GS decomposition

## 8.24 pbc\_support.h File Reference

Auxiliary functions for Pairing-based protocols.

```
#include "amcl.h"
```

### Macros

- #define `TIME_SLOT_MINUTES` 1440

### Functions

- void `mhashit` (int sha, int n, `octet *x`, `octet *w`)  
*general purpose hash function  $w=hash(n|x)$*
- `unsign32 today` (void)  
*Supply today's date as days from the epoch.*
- void `HASH_ALL` (int h, `octet *I`, `octet *U`, `octet *CU`, `octet *Y`, `octet *V`, `octet *R`, `octet *W`, `octet *H`)  
*Hash the session transcript.*
- void `HASH_ID` (int h, `octet *ID`, `octet *HID`)  
*Hash an M-Pin Identity to an octet string.*
- `unsign32 GET_TIME` (void)  
*Get epoch time as unsigned integer.*
- void `AES_GCM_ENCRYPT` (`octet *K`, `octet *IV`, `octet *H`, `octet *P`, `octet *C`, `octet *T`)  
*AES-GCM Encryption.*
- void `AES_GCM_DECRYPT` (`octet *K`, `octet *IV`, `octet *H`, `octet *C`, `octet *P`, `octet *T`)  
*AES-GCM Decryption.*

### 8.24.1 Detailed Description

#### Author

Mike Scott

### 8.24.2 Macro Definition Documentation

### 8.24.2.1 TIME\_SLOT\_MINUTES

```
#define TIME_SLOT_MINUTES 1440
```

Time Slot = 1 day

## 8.24.3 Function Documentation

### 8.24.3.1 AES\_GCM\_DECRYPT()

```
void AES_GCM_DECRYPT (  
    octet * K,  
    octet * IV,  
    octet * H,  
    octet * C,  
    octet * P,  
    octet * T )
```

#### Parameters

<i>K</i>	AES key
<i>IV</i>	Initialization vector
<i>H</i>	Header
<i>P</i>	Plaintext
<i>C</i>	Ciphertext
<i>T</i>	Checksum

### 8.24.3.2 AES\_GCM\_ENCRYPT()

```
void AES_GCM_ENCRYPT (  
    octet * K,  
    octet * IV,  
    octet * H,  
    octet * P,  
    octet * C,  
    octet * T )
```

#### Parameters

<i>K</i>	AES key
<i>IV</i>	Initialization vector
<i>H</i>	Header
<i>P</i>	Plaintext
<i>C</i>	Ciphertext
<i>T</i>	Checksum

### 8.24.3.3 GET\_TIME()

```
unsign32 GET_TIME (
    void )
```

#### Returns

current epoch time in seconds

### 8.24.3.4 HASH\_ALL()

```
void HASH_ALL (
    int h,
    octet * I,
    octet * U,
    octet * CU,
    octet * Y,
    octet * V,
    octet * R,
    octet * W,
    octet * H )
```

#### Parameters

<i>h</i>	is the hash type
<i>I</i>	is the hashed input client ID = H(ID)
<i>U</i>	is the client output = x.H(ID)
<i>CU</i>	is the client output = x.(H(ID)+H(T H(ID)))
<i>Y</i>	is the server challenge
<i>V</i>	is the client part response
<i>R</i>	is the client part response
<i>W</i>	is the server part response
<i>H</i>	the output is the hash of all of the above that apply

### 8.24.3.5 HASH\_ID()

```
void HASH_ID (
    int h,
    octet * ID,
    octet * HID )
```



## Parameters

<i>h</i>	is the hash type
<i>ID</i>	an octet containing the identity
<i>HID</i>	an octet containing the hashed identity

## 8.24.3.6 mhashit()

```
void mhashit (
    int sha,
    int n,
    octet * x,
    octet * w )
```

## Parameters

<i>sha</i>	is the hash type
<i>n</i>	integer involved in the hash
<i>x</i>	octet involved in the hash
<i>w</i>	output

## 8.24.3.7 today()

```
unsigned32 today (
    void )
```

## Returns

today's date, as number of days elapsed since the epoch

## 8.25 randapi.h File Reference

PRNG API File.

```
#include "amcl.h"
```

## Functions

- void [CREATE\\_CSPRNG](#) (csp rng \*R, octet \*S)  
*Initialise a random number generator.*
- void [KILL\\_CSPRNG](#) (csp rng \*R)  
*Kill a random number generator.*

## 8.25.1 Detailed Description

### Author

Mike Scott

## 8.25.2 Function Documentation

### 8.25.2.1 CREATE\_CSPRNG()

```
void CREATE_CSPRNG (
    csprng * R,
    octet * S )
```

#### Parameters

<i>R</i>	is a pointer to a cryptographically secure random number generator
<i>S</i>	is an input truly random seed value

### 8.25.2.2 KILL\_CSPRNG()

```
void KILL_CSPRNG (
    csprng * R )
```

Deletes all internal state

#### Parameters

<i>R</i>	is a pointer to a cryptographically secure random number generator
----------	--

## 8.26 rsa\_2048.h File Reference

RSA Header file for implementation of RSA protocol.

```
#include "ff_2048.h"
#include "rsa_support.h"
```

### Data Structures

- struct [rsa\\_public\\_key\\_2048](#)  
*Integer Factorisation Public Key.*
- struct [rsa\\_private\\_key\\_2048](#)  
*Integer Factorisation Private Key.*

## Macros

- #define `HASH_TYPE_RSA_2048` SHA256
- #define `RFS_2048` `MODBYTES_1024_58*FFLEN_2048`

## Functions

- void `RSA_2048_KEY_PAIR` (`csprng *R`, `sign32 e`, `rsa_private_key_2048 *PRIV`, `rsa_public_key_2048 *PUB`, `octet *P`, `octet *Q`)  
*RSA Key Pair Generator.*
- void `RSA_2048_ENCRYPT` (`rsa_public_key_2048 *PUB`, `octet *F`, `octet *G`)  
*RSA encryption of suitably padded plaintext.*
- void `RSA_2048_DECRYPT` (`rsa_private_key_2048 *PRIV`, `octet *G`, `octet *F`)  
*RSA decryption of ciphertext.*
- void `RSA_2048_PRIVATE_KEY_KILL` (`rsa_private_key_2048 *PRIV`)  
*Destroy an RSA private Key.*
- void `RSA_2048_fromOctet` (`BIG_1024_58 *x`, `octet *S`)  
*Populates an RSA public key from an octet string.*

### 8.26.1 Detailed Description

#### Author

Mike Scott declares functions

### 8.26.2 Macro Definition Documentation

#### 8.26.2.1 HASH\_TYPE\_RSA\_2048

```
#define HASH_TYPE_RSA_2048 SHA256
```

Chosen Hash algorithm

#### 8.26.2.2 RFS\_2048

```
#define RFS_2048 MODBYTES_1024_58*FFLEN_2048
```

RSA Public Key Size in bytes

### 8.26.3 Function Documentation

#### 8.26.3.1 RSA\_2048\_DECRYPT()

```
void RSA_2048_DECRYPT (
    rsa_private_key_2048 * PRIV,
    octet * G,
    octet * F )
```

## Parameters

$P_{\leftarrow}$ <i>RIV</i>	the input RSA private key
<i>G</i>	is the input ciphertext
<i>F</i>	is output plaintext (requires unpadding)

## 8.26.3.2 RSA\_2048\_ENCRYPT()

```
void RSA_2048_ENCRYPT (
    rsa_public_key_2048 * PUB,
    octet * F,
    octet * G )
```

## Parameters

<i>PUB</i>	the input RSA public key
<i>F</i>	is input padded message
<i>G</i>	is the output ciphertext

## 8.26.3.3 RSA\_2048\_fromOctet()

```
void RSA_2048_fromOctet (
    BIG_1024_58 * x,
    octet * S )
```

Creates RSA public key from big-endian base 256 form.

## Parameters

<i>x</i>	FF instance to be created from an octet string
<i>S</i>	input octet string

## 8.26.3.4 RSA\_2048\_KEY\_PAIR()

```
void RSA_2048_KEY_PAIR (
    csprng * R,
    sign32 e,
    rsa_private_key_2048 * PRIV,
    rsa_public_key_2048 * PUB,
    octet * P,
    octet * Q )
```

## Parameters

<i>R</i>	is a pointer to a cryptographically secure random number generator
<i>e</i>	the encryption exponent
<i>P</i> ↔ <i>RIV</i>	the output RSA private key
<i>PUB</i>	the output RSA public key
<i>P</i>	Input prime number. Used when <i>R</i> is equal to NULL for testing
<i>Q</i>	Input prime number. Used when <i>R</i> is equal to NULL for testing

## 8.26.3.5 RSA\_2048\_PRIVATE\_KEY\_KILL()

```
void RSA_2048_PRIVATE_KEY_KILL (
    rsa_private_key_2048 * PRIV )
```

## Parameters

<i>P</i> ↔ <i>RIV</i>	the input RSA private key. Destroyed on output.
--------------------------	---

## 8.27 rsa\_support.h File Reference

RSA Support Header File.

```
#include "amcl.h"
```

## Macros

- #define `MAX_RSA_BYTES` 512

## Functions

- int `PKCS15` (int *h*, `octet` \**M*, `octet` \**W*)  
*PKCS V1.5 padding of a message prior to RSA signature.*
- int `OAEP_ENCODE` (int *h*, `octet` \**M*, `csprng` \**R*, `octet` \**P*, `octet` \**F*)  
*OAEP padding of a message prior to RSA encryption.*
- int `OAEP_DECODE` (int *h*, `octet` \**P*, `octet` \**F*)  
*OAEP unpadding of a message after RSA decryption.*

## 8.27.1 Detailed Description

## Author

Mike Scott

## 8.27.2 Macro Definition Documentation

### 8.27.2.1 MAX\_RSA\_BYTES

```
#define MAX_RSA_BYTES 512
```

Maximum of 4096

## 8.27.3 Function Documentation

### 8.27.3.1 OAEP\_DECODE()

```
int OAEP_DECODE (
    int h,
    octet * P,
    octet * F )
```

Unpadding is done in-place

#### Parameters

<i>h</i>	is the hash type
<i>P</i>	are input encoding parameter string (could be NULL)
<i>F</i>	is input padded message, unpadded on output

#### Returns

0 if OK, else 1

### 8.27.3.2 OAEP\_ENCODE()

```
int OAEP_ENCODE (
    int h,
    octet * M,
    csprng * R,
    octet * P,
    octet * F )
```

#### Parameters

<i>h</i>	is the hash type
----------	------------------

## Parameters

<i>M</i>	is the input message
<i>R</i>	is a pointer to a cryptographically secure random number generator
<i>P</i>	are input encoding parameter string (could be NULL)
<i>F</i>	is the output encoding, ready for RSA encryption

## Returns

0 if OK, else 1

## 8.27.3.3 PKCS15()

```
int PKCS15 (
    int h,
    octet * M,
    octet * W )
```

## Parameters

<i>h</i>	is the hash type
<i>M</i>	is the input message
<i>W</i>	is the output encoding, ready for RSA signature

## Returns

1 if OK, else 0

## 8.28 utils.c File Reference

AMCL Support functions for M-Pin servers.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "amcl.h"
#include "utils.h"
```

## Functions

- void [amcl\\_hex2bin](#) (const char \*src, char \*dst, int src\_len)  
*Decode hex value.*
- void [amcl\\_bin2hex](#) (char \*src, char \*dst, int src\_len)  
*Encode binary string.*

- void `amcl_print_hex` (char \*src, int src\_len)  
*Print encoded binary string in hex.*
- int `generateOTP` (csprng \*RNG)  
*Generate a random six digit one time password.*
- void `generateRandom` (csprng \*RNG, octet \*randomValue)  
*Generate a random Octet.*

### 8.28.1 Detailed Description

#### Author

Mike Scott  
Kealan McCusker

#### Date

28th July 2016 LICENSE

Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

### 8.28.2 Function Documentation

#### 8.28.2.1 `amcl_bin2hex()`

```
void amcl_bin2hex (
    char * src,
    char * dst,
    int src_len )
```

Encode binary string.

#### Parameters

<code>src</code>	Binary string
<code>dst</code>	Hex encoded string
<code>src_len</code>	length binary string



### 8.28.2.2 amcl\_hex2bin()

```
void amcl_hex2bin (
    const char * src,
    char * dst,
    int src_len )
```

Decode hex value.

#### Parameters

<i>src</i>	Hex encoded string
<i>dst</i>	Binary string
<i>src_len</i>	length Hex encoded string

### 8.28.2.3 amcl\_print\_hex()

```
void amcl_print_hex (
    char * src,
    int src_len )
```

Print encoded binary string in hex.

#### Parameters

<i>src</i>	Binary string
<i>src_len</i>	length binary string

### 8.28.2.4 generateOTP()

```
int generateOTP (
    csprng * RNG )
```

Generates a random six digit one time password.

#### Parameters

<i>RNG</i>	random number generator
------------	-------------------------

#### Returns

OTP One Time Password

### 8.28.2.5 generateRandom()

```
void generateRandom (
    csprng * RNG,
    octet * randomValue )
```

Generate a random Octet.

#### Parameters

<i>RNG</i>	random number generator
<i>randomValue</i>	random Octet

## 8.29 utils.h File Reference

Utility functions Header File.

```
#include "amcl.h"
```

### Functions

- void [amcl\\_hex2bin](#) (const char \*src, char \*dst, int src\_len)  
*Decode hex value.*
- void [amcl\\_bin2hex](#) (char \*src, char \*dst, int src\_len)  
*Encode binary string.*
- void [amcl\\_print\\_hex](#) (char \*src, int src\_len)  
*Print encoded binary string in hex.*
- void [generateRandom](#) (csprng \*RNG, octet \*randomValue)  
*Generate a random Octet.*
- int [generateOTP](#) (csprng \*RNG)  
*Generate a random six digit one time password.*

### 8.29.1 Detailed Description

#### Author

Kealan McCusker

### 8.29.2 Function Documentation

#### 8.29.2.1 amcl\_bin2hex()

```
void amcl_bin2hex (
    char * src,
    char * dst,
    int src_len )
```

Encode binary string.

## Parameters

<i>src</i>	Binary string
<i>dst</i>	Hex encoded string
<i>src_len</i>	length binary string

## 8.29.2.2 amcl\_hex2bin()

```
void amcl_hex2bin (
    const char * src,
    char * dst,
    int src_len )
```

Decode hex value.

## Parameters

<i>src</i>	Hex encoded string
<i>dst</i>	Binary string
<i>src_len</i>	length Hex encoded string

## 8.29.2.3 amcl\_print\_hex()

```
void amcl_print_hex (
    char * src,
    int src_len )
```

Print encoded binary string in hex.

## Parameters

<i>src</i>	Binary string
<i>src_len</i>	length binary string

## 8.29.2.4 generateOTP()

```
int generateOTP (
    csprng * RNG )
```

Generates a random six digit one time password.

**Parameters**

<i>RNG</i>	random number generator
------------	-------------------------

**Returns**

OTP One Time Password

**8.29.2.5 generateRandom()**

```
void generateRandom (
    csprng * RNG,
    octet * randomValue )
```

Generate a random Octet.

**Parameters**

<i>RNG</i>	random number generator
<i>randomValue</i>	random Octet

**8.30 version.c File Reference**

AMCL version support function.

```
#include "version.h"
```

**Functions**

- void [amcl\\_version](#) (void)  
*Print version number and information about the build.*

**8.30.1 Detailed Description****Author**

Mike Scott  
Kealan McCusker

**Date**

28th April 2016 LICENSE

Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

## 8.30.2 Function Documentation

### 8.30.2.1 amcl\_version()

```
void amcl_version (  
    void )
```

Print version number and information about the build.

## 8.31 wcc\_BLS381.h File Reference

WCC Header File.

```
#include "pair_BLS381.h"  
#include "pbc_support.h"
```

### Macros

- #define [WCC\\_PGS\\_BLS381](#) MODBYTES\_384\_58
- #define [WCC\\_PFS\\_BLS381](#) MODBYTES\_384\_58
- #define [WCC\\_OK](#) 0
- #define [WCC\\_INVALID\\_POINT](#) -51
- #define [TIME\\_SLOT\\_MINUTES](#) 1440
- #define [PIV](#) 12
- #define [PTAG](#) 16

## Functions

- int [WCC\\_BLS381\\_RANDOM\\_GENERATE](#) (csprng \*RNG, octet \*S)  
*Generate a random integer.*
- void [WCC\\_BLS381\\_Hq](#) (int sha, octet \*A, octet \*B, octet \*C, octet \*D, octet \*h)  
*Hash EC Points and Id to an integer.*
- int [WCC\\_BLS381\\_GET\\_G2\\_MULTIPLE](#) (octet \*S, octet \*HID, octet \*VG2)  
*Calculate value in G2 multiplied by an integer.*
- int [WCC\\_BLS381\\_GET\\_G1\\_MULTIPLE](#) (octet \*S, octet \*HID, octet \*VG1)  
*Calculate value in G1 multiplied by an integer.*
- int [WCC\\_BLS381\\_SENDER\\_KEY](#) (int sha, octet \*xOct, octet \*piaOct, octet \*pibOct, octet \*PbG2Oct, octet \*PgG1Oct, octet \*AKeyG1Oct, octet \*IdBOct, octet \*AESKeyOct)  
*Calculate the sender AES key.*
- int [WCC\\_BLS381\\_RECEIVER\\_KEY](#) (int sha, octet \*yOct, octet \*wOct, octet \*piaOct, octet \*pibOct, octet \*PaG1Oct, octet \*PgG1Oct, octet \*BKeyG2Oct, octet \*IdAOct, octet \*AESKeyOct)  
*Calculate the receiver AES key.*
- int [WCC\\_BLS381\\_RECOMBINE\\_G1](#) (octet \*R1, octet \*R2, octet \*R)  
*Add two members from the group G1.*
- int [WCC\\_BLS381\\_RECOMBINE\\_G2](#) (octet \*W1, octet \*W2, octet \*W)  
*Add two members from the group G2.*

### 8.31.1 Detailed Description

#### Author

Mike Scott  
Kealan McCusker

### 8.31.2 Macro Definition Documentation

#### 8.31.2.1 PIV

```
#define PIV 12
```

AES-GCM Initialization Vector Size

#### 8.31.2.2 PTAG

```
#define PTAG 16
```

AES-GCM MAC Size

#### 8.31.2.3 TIME\_SLOT\_MINUTES

```
#define TIME_SLOT_MINUTES 1440
```

Time Slot = 1 day

#### 8.31.2.4 WCC\_INVALID\_POINT

```
#define WCC_INVALID_POINT -51
```

Point is NOT on the curve

#### 8.31.2.5 WCC\_OK

```
#define WCC_OK 0
```

Function completed without error

#### 8.31.2.6 WCC\_PFS\_BLS381

```
#define WCC_PFS_BLS381 MODBYTES_384_58
```

WCC Field Size

#### 8.31.2.7 WCC\_PGS\_BLS381

```
#define WCC_PGS_BLS381 MODBYTES_384_58
```

WCC Group Size

### 8.31.3 Function Documentation

#### 8.31.3.1 WCC\_BLS381\_GET\_G1\_MULTIPLE()

```
int WCC_BLS381_GET_G1_MULTIPLE (
    octet * S,
    octet * HID,
    octet * VG1 )
```

Calculate a value in G1.  $VG1 = s * H1(ID)$  where ID is the identity.

1.  $VG1 = s * H1(ID)$

#### Parameters

<i>S</i>	integer modulus curve order
<i>HID</i>	Hash of ID padded with zeros to the field size
<i>VG1</i>	EC point $VG1 = s * H1(ID)$

**Returns**

rtm Returns 0 if successful or else an error code

**8.31.3.2 WCC\_BLS381\_GET\_G2\_MULTIPLE()**

```
int WCC_BLS381_GET_G2_MULTIPLE (
    octet * S,
    octet * HID,
    octet * VG2 )
```

Calculate a value in G2.  $VG2 = s * H2(ID)$  where ID is the identity.

1.  $VG2 = s * H2(ID)$

**Parameters**

<i>S</i>	integer modulus curve order
<i>HID</i>	Hash of ID padded with zeros to the field size
<i>VG2</i>	EC Point $VG2 = s * H2(ID)$

**Returns**

rtm Returns 0 if successful or else an error code

**8.31.3.3 WCC\_BLS381\_Hq()**

```
void WCC_BLS381_Hq (
    int sha,
    octet * A,
    octet * B,
    octet * C,
    octet * D,
    octet * h )
```

Perform sha256 of EC Points and Id. Map to an integer modulo the curve order.

1.  $x = \text{toInteger}(\text{sha256}(A,B,C,D))$
2.  $h = x \% q$  where  $q$  is the curve order

**Parameters**

<i>sha</i>	Hash type
<i>A</i>	EC Point
<i>B</i>	EC Point
<i>C</i>	EC Point
<i>D</i>	Identity
<i>h</i>	Integer result



## 8.31.3.4 WCC\_BLS381\_RANDOM\_GENERATE()

```
int WCC_BLS381_RANDOM_GENERATE (
    csprng * RNG,
    octet * S )
```

Generate a random number modulus the group order.

## Parameters

<i>RNG</i>	cryptographically secure random number generator
<i>S</i>	Returned random integer modulus the group order

## 8.31.3.5 WCC\_BLS381\_RECEIVER\_KEY()

```
int WCC_BLS381_RECEIVER_KEY (
    int sha,
    octet * yOct,
    octet * wOct,
    octet * piaOct,
    octet * pibOct,
    octet * PaG1Oct,
    octet * PgG1Oct,
    octet * BKeyG2Oct,
    octet * IdAOct,
    octet * AESKeyOct )
```

Calculate the receiver AES key.

1.  $j = e(\text{pia} \cdot \text{AG1} + \text{PaG1}, (y + \text{pib}) \cdot \text{BKeyG2})$
2.  $K = H(j, w \cdot \text{PaG1})$

## Parameters

<i>sha</i>	Hash type
<i>yOct</i>	Random $y < q$ where $q$ is the curve order
<i>wOct</i>	Random $w < q$ where $q$ is the curve order
<i>piaOct</i>	$H_q(\text{PaG1}, \text{PbG2}, \text{PgG1})$
<i>pibOct</i>	$H_q(\text{PbG2}, \text{PaG1}, \text{PgG1})$
<i>PaG1Oct</i>	$x \cdot \text{AG1}$ where $x < q$
<i>PgG1Oct</i>	$w \cdot \text{AG1}$ where $w < q$
<i>BKeyG2Oct</i>	Receiver key
<i>IdAOct</i>	Sender identity
<i>AESKeyOct</i>	AES key returned

**Returns**

rtn Returns 0 if successful or else an error code

**8.31.3.6 WCC\_BLS381\_RECOMBINE\_G1()**

```
int WCC_BLS381_RECOMBINE_G1 (
    octet * R1,
    octet * R2,
    octet * R )
```

Add two members from the group G1.

**Parameters**

<i>R1</i>	member of G1
<i>R2</i>	member of G1
<i>R</i>	returns member of $G1 = R1+R2$

**Returns**

Returns 0 if successful or else an error code

**8.31.3.7 WCC\_BLS381\_RECOMBINE\_G2()**

```
int WCC_BLS381_RECOMBINE_G2 (
    octet * W1,
    octet * W2,
    octet * W )
```

Add two members from the group G2.

**Parameters**

<i>W1</i>	member of G2
<i>W2</i>	member of G2
<i>W</i>	returns member of $G2 = W1+W2$

**Returns**

Returns 0 if successful or else an error code

## 8.31.3.8 WCC\_BLS381\_SENDER\_KEY()

```
int WCC_BLS381_SENDER_KEY (
    int sha,
    octet * xOct,
    octet * piaOct,
    octet * pibOct,
    octet * PbG2Oct,
    octet * PgG1Oct,
    octet * AKeyG1Oct,
    octet * IdBOct,
    octet * AESKeyOct )
```

Calculate the sender AES Key.

1.  $j = e((x + pia).AKeyG1, pib.BG2 + PbG2)$
2.  $K = H(j, x.PgG1)$

## Parameters

<i>sha</i>	Hash type
<i>xOct</i>	Random $x < q$ where $q$ is the curve order
<i>piaOct</i>	$Hq(PaG1, PbG2, PgG1)$
<i>pibOct</i>	$Hq(PbG2, PaG1, PgG1)$
<i>PbG2Oct</i>	$y.BG2$ where $y < q$
<i>PgG1Oct</i>	$w.AG1$ where $w < q$
<i>AKeyG1Oct</i>	Sender key
<i>IdBOct</i>	Receiver identity
<i>AESKeyOct</i>	Returned AES key

## Returns

rtn Returns 0 if successful or else an error code

## 8.32 x509.h File Reference

X509 function Header File.

## Data Structures

- struct [pktype](#)  
*Public key type.*

## Functions

- `pktype X509_extract_cert_sig (octet *c, octet *s)`  
*Extract certificate signature.*
- `int X509_extract_cert (octet *sc, octet *c)`
- `pktype X509_extract_public_key (octet *c, octet *k)`
- `int X509_find_issuer (octet *c)`
- `int X509_find_validity (octet *c)`
- `int X509_find_subject (octet *c)`
- `int X509_find_entity_property (octet *c, octet *S, int s, int *f)`
- `int X509_find_start_date (octet *c, int s)`
- `int X509_find_expiry_date (octet *c, int s)`

### 8.32.1 Detailed Description

#### Author

Mike Scott

### 8.32.2 Function Documentation

#### 8.32.2.1 X509\_extract\_cert()

```
int X509_extract_cert (  
    octet * sc,  
    octet * c )
```

#### Parameters

<code>sc</code>	a signed certificate
<code>c</code>	the extracted certificate

#### Returns

0 on failure

#### 8.32.2.2 X509\_extract\_cert\_sig()

```
pktype X509_extract_cert_sig (  
    octet * c,  
    octet * s )
```

**Parameters**

<i>c</i>	an X.509 certificate
<i>s</i>	the extracted signature

**Returns**

0 on failure, or indicator of signature type (ECC or RSA)

**8.32.2.3 X509\_extract\_public\_key()**

```
pktype X509_extract_public_key (  
    octet * c,  
    octet * k )
```

**Parameters**

<i>c</i>	an X.509 certificate
<i>k</i>	the extracted key

**Returns**

0 on failure, or indicator of public key type (ECC or RSA)

**8.32.2.4 X509\_find\_entity\_property()**

```
int X509_find_entity_property (  
    octet * c,  
    octet * S,  
    int s,  
    int * f )
```

**Parameters**

<i>c</i>	an X.509 certificate
<i>S</i>	is OID of property we are looking for
<i>s</i>	is a pointer to the section of interest in the cert
<i>f</i>	is pointer to the length of the property

**Returns**

0 on failure, or pointer to the property

### 8.32.2.5 X509\_find\_expiry\_date()

```
int X509_find_expiry_date (
    octet * c,
    int s )
```

#### Parameters

<i>c</i>	an X.509 certificate
<i>s</i>	is a pointer to the start of the validity field

#### Returns

0 on failure, or pointer to the expiry date

### 8.32.2.6 X509\_find\_issuer()

```
int X509_find_issuer (
    octet * c )
```

#### Parameters

<i>c</i>	an X.509 certificate
----------	----------------------

#### Returns

0 on failure, or pointer to issuer field in cert

### 8.32.2.7 X509\_find\_start\_date()

```
int X509_find_start_date (
    octet * c,
    int s )
```

#### Parameters

<i>c</i>	an X.509 certificate
<i>s</i>	is a pointer to the start of the validity field

#### Returns

0 on failure, or pointer to the start date

### 8.32.2.8 X509\_find\_subject()

```
int X509_find_subject (  
    octet * c )
```

#### Parameters

<code>c</code>	an X.509 certificate
----------------	----------------------

#### Returns

0 on failure, or pointer to subject field in cert

### 8.32.2.9 X509\_find\_validity()

```
int X509_find_validity (  
    octet * c )
```

#### Parameters

<code>c</code>	an X.509 certificate
----------------	----------------------

#### Returns

0 on failure, or pointer to validity field in cert





# Index

## a

- FP12\_BLS381, [17](#)
- FP2\_BLS381, [18](#)
- FP4\_BLS381, [19](#)
- gcm, [20](#)
- AES\_CBC\_IV0\_DECRYPT
  - ecdh\_support.h, [131](#)
- AES\_CBC\_IV0\_ENCRYPT
  - ecdh\_support.h, [131](#)
- AES\_GCM\_DECRYPT
  - pbk\_support.h, [267](#)
- AES\_GCM\_ENCRYPT
  - pbk\_support.h, [267](#)
- amcl\_aes, [13](#)
  - f, [13](#)
  - fkey, [13](#)
  - mode, [13](#)
  - Nk, [14](#)
  - Nr, [14](#)
  - rkey, [14](#)
- amcl\_bin2hex
  - utils.c, [276](#)
  - utils.h, [278](#)
- amcl\_hex2bin
  - utils.c, [276](#)
  - utils.h, [279](#)
- amcl\_print\_hex
  - utils.c, [277](#)
  - utils.h, [279](#)
- amcl\_version
  - version.c, [281](#)
- arch.h, [31](#)
  - byte, [31](#)
  - CHUNK, [32](#)
  - chunk, [32](#)
  - sign16, [32](#)
  - sign32, [32](#)
  - sign64, [32](#)
  - sign8, [32](#)
  - uchar, [32](#)
  - unsign32, [33](#)
  - unsign64, [33](#)

## b

- config\_big\_384\_58.h, [123](#)
- BASEBITS\_512\_60
  - config\_big\_512\_60.h, [124](#)
- BFS\_BLS381
  - bls\_BLS381.h, [117](#)
- BGS\_BLS381
  - bls\_BLS381.h, [117](#)
- BIG\_1024\_58
  - big\_1024\_58.h, [38](#)
- BIG\_1024\_58\_add
  - big\_1024\_58.h, [38](#)
- BIG\_1024\_58\_bit
  - big\_1024\_58.h, [38](#)
- BIG\_1024\_58\_cmove
  - big\_1024\_58.h, [39](#)
- BIG\_1024\_58\_comp
  - big\_1024\_58.h, [39](#)
- BIG\_1024\_58\_copy
  - big\_1024\_58.h, [39](#)
- BIG\_1024\_58\_cswap
  - big\_1024\_58.h, [40](#)
- BIG\_1024\_58\_dadd
  - big\_1024\_58.h, [40](#)
- BIG\_1024\_58\_dcmove
  - big\_1024\_58.h, [40](#)
- BIG\_1024\_58\_dcomp
  - big\_1024\_58.h, [41](#)
- BIG\_1024\_58\_dcopy
  - big\_1024\_58.h, [41](#)
- BIG\_1024\_58\_ddiv
  - big\_1024\_58.h, [41](#)
- BIG\_1024\_58\_dec
  - big\_1024\_58.h, [42](#)
- BIG\_1024\_58\_dfromBytesLen
  - big\_1024\_58.h, [42](#)
- BIG\_1024\_58\_diszilch
  - big\_1024\_58.h, [42](#)
- BIG\_1024\_58\_div3
  - big\_1024\_58.h, [42](#)
- BIG\_1024\_58\_dmod
  - big\_1024\_58.h, [44](#)
- BIG\_1024\_58\_dmod2m
  - big\_1024\_58.h, [44](#)
- BIG\_1024\_58\_dnbits
  - big\_1024\_58.h, [44](#)
- BIG\_1024\_58\_dnorm
  - big\_1024\_58.h, [45](#)
- BIG\_1024\_58\_doutput
  - big\_1024\_58.h, [45](#)

BIG\_1024\_58\_drawoutput  
big\_1024\_58.h, [45](#)

BIG\_1024\_58\_dscopy  
big\_1024\_58.h, [45](#)

BIG\_1024\_58\_dshl  
big\_1024\_58.h, [46](#)

BIG\_1024\_58\_dshr  
big\_1024\_58.h, [46](#)

BIG\_1024\_58\_dsub  
big\_1024\_58.h, [46](#)

BIG\_1024\_58\_dsucopy  
big\_1024\_58.h, [47](#)

BIG\_1024\_58\_dzero  
big\_1024\_58.h, [47](#)

BIG\_1024\_58\_fromBytes  
big\_1024\_58.h, [47](#)

BIG\_1024\_58\_fromBytesLen  
big\_1024\_58.h, [47](#)

BIG\_1024\_58\_fshl  
big\_1024\_58.h, [48](#)

BIG\_1024\_58\_fshr  
big\_1024\_58.h, [48](#)

BIG\_1024\_58\_imul  
big\_1024\_58.h, [49](#)

BIG\_1024\_58\_inc  
big\_1024\_58.h, [49](#)

BIG\_1024\_58\_invmod2m  
big\_1024\_58.h, [49](#)

BIG\_1024\_58\_invmodp  
big\_1024\_58.h, [49](#)

BIG\_1024\_58\_isunity  
big\_1024\_58.h, [50](#)

BIG\_1024\_58\_iszilch  
big\_1024\_58.h, [50](#)

BIG\_1024\_58\_jacobi  
big\_1024\_58.h, [50](#)

BIG\_1024\_58\_lastbits  
big\_1024\_58.h, [51](#)

BIG\_1024\_58\_mod  
big\_1024\_58.h, [51](#)

BIG\_1024\_58\_mod2m  
big\_1024\_58.h, [51](#)

BIG\_1024\_58\_moddiv  
big\_1024\_58.h, [52](#)

BIG\_1024\_58\_modmul  
big\_1024\_58.h, [52](#)

BIG\_1024\_58\_modneg  
big\_1024\_58.h, [53](#)

BIG\_1024\_58\_modsqr  
big\_1024\_58.h, [53](#)

BIG\_1024\_58\_monty  
big\_1024\_58.h, [53](#)

BIG\_1024\_58\_mul  
big\_1024\_58.h, [54](#)

BIG\_1024\_58\_nbits  
big\_1024\_58.h, [54](#)

BIG\_1024\_58\_norm  
big\_1024\_58.h, [54](#)

BIG\_1024\_58\_one  
big\_1024\_58.h, [54](#)

BIG\_1024\_58\_or  
big\_1024\_58.h, [55](#)

BIG\_1024\_58\_output  
big\_1024\_58.h, [55](#)

BIG\_1024\_58\_parity  
big\_1024\_58.h, [55](#)

BIG\_1024\_58\_pmul  
big\_1024\_58.h, [56](#)

BIG\_1024\_58\_pxmuls  
big\_1024\_58.h, [56](#)

BIG\_1024\_58\_random  
big\_1024\_58.h, [56](#)

BIG\_1024\_58\_randomnum  
big\_1024\_58.h, [57](#)

BIG\_1024\_58\_rawoutput  
big\_1024\_58.h, [57](#)

BIG\_1024\_58\_rcopy  
big\_1024\_58.h, [57](#)

BIG\_1024\_58\_sdcopy  
big\_1024\_58.h, [57](#)

BIG\_1024\_58\_sdiv  
big\_1024\_58.h, [58](#)

BIG\_1024\_58\_sducopy  
big\_1024\_58.h, [58](#)

BIG\_1024\_58\_shl  
big\_1024\_58.h, [58](#)

BIG\_1024\_58\_shr  
big\_1024\_58.h, [59](#)

BIG\_1024\_58\_smul  
big\_1024\_58.h, [59](#)

BIG\_1024\_58\_split  
big\_1024\_58.h, [59](#)

BIG\_1024\_58\_sqr  
big\_1024\_58.h, [60](#)

BIG\_1024\_58\_ssn  
big\_1024\_58.h, [60](#)

BIG\_1024\_58\_sub  
big\_1024\_58.h, [60](#)

BIG\_1024\_58\_toBytes  
big\_1024\_58.h, [61](#)

BIG\_1024\_58\_zero  
big\_1024\_58.h, [61](#)

BIG\_384\_58  
big\_384\_58.h, [66](#)

BIG\_384\_58\_add  
big\_384\_58.h, [66](#)

BIG\_384\_58\_bit  
big\_384\_58.h, [67](#)

BIG\_384\_58\_cmove  
big\_384\_58.h, [67](#)

BIG\_384\_58\_comp  
big\_384\_58.h, [67](#)

BIG\_384\_58\_copy  
big\_384\_58.h, [68](#)

BIG\_384\_58\_cswap  
big\_384\_58.h, [68](#)

BIG\_384\_58\_dadd  
big\_384\_58.h, 68

BIG\_384\_58\_dcmove  
big\_384\_58.h, 69

BIG\_384\_58\_dcomp  
big\_384\_58.h, 69

BIG\_384\_58\_dcopy  
big\_384\_58.h, 69

BIG\_384\_58\_ddiv  
big\_384\_58.h, 70

BIG\_384\_58\_dec  
big\_384\_58.h, 70

BIG\_384\_58\_dfromBytesLen  
big\_384\_58.h, 70

BIG\_384\_58\_diszilch  
big\_384\_58.h, 70

BIG\_384\_58\_div3  
big\_384\_58.h, 71

BIG\_384\_58\_dmod  
big\_384\_58.h, 71

BIG\_384\_58\_dmod2m  
big\_384\_58.h, 71

BIG\_384\_58\_dnbits  
big\_384\_58.h, 72

BIG\_384\_58\_dnorm  
big\_384\_58.h, 72

BIG\_384\_58\_doutput  
big\_384\_58.h, 72

BIG\_384\_58\_drawoutput  
big\_384\_58.h, 73

BIG\_384\_58\_dscopy  
big\_384\_58.h, 73

BIG\_384\_58\_dshl  
big\_384\_58.h, 73

BIG\_384\_58\_dshr  
big\_384\_58.h, 73

BIG\_384\_58\_dsub  
big\_384\_58.h, 74

BIG\_384\_58\_dscopy  
big\_384\_58.h, 74

BIG\_384\_58\_dzero  
big\_384\_58.h, 74

BIG\_384\_58\_fromBytes  
big\_384\_58.h, 75

BIG\_384\_58\_fromBytesLen  
big\_384\_58.h, 75

BIG\_384\_58\_fshl  
big\_384\_58.h, 75

BIG\_384\_58\_fshr  
big\_384\_58.h, 75

BIG\_384\_58\_imul  
big\_384\_58.h, 76

BIG\_384\_58\_inc  
big\_384\_58.h, 76

BIG\_384\_58\_invmod2m  
big\_384\_58.h, 76

BIG\_384\_58\_invmodp  
big\_384\_58.h, 77

BIG\_384\_58\_isunity  
big\_384\_58.h, 77

BIG\_384\_58\_iszilch  
big\_384\_58.h, 77

BIG\_384\_58\_jacobi  
big\_384\_58.h, 78

BIG\_384\_58\_lastbits  
big\_384\_58.h, 78

BIG\_384\_58\_mod  
big\_384\_58.h, 78

BIG\_384\_58\_mod2m  
big\_384\_58.h, 79

BIG\_384\_58\_moddiv  
big\_384\_58.h, 79

BIG\_384\_58\_modmul  
big\_384\_58.h, 79

BIG\_384\_58\_modneg  
big\_384\_58.h, 80

BIG\_384\_58\_modsqr  
big\_384\_58.h, 80

BIG\_384\_58\_monty  
big\_384\_58.h, 81

BIG\_384\_58\_mul  
big\_384\_58.h, 81

BIG\_384\_58\_nbits  
big\_384\_58.h, 81

BIG\_384\_58\_norm  
big\_384\_58.h, 81

BIG\_384\_58\_one  
big\_384\_58.h, 82

BIG\_384\_58\_or  
big\_384\_58.h, 82

BIG\_384\_58\_output  
big\_384\_58.h, 82

BIG\_384\_58\_parity  
big\_384\_58.h, 83

BIG\_384\_58\_pmul  
big\_384\_58.h, 83

BIG\_384\_58\_pxmuls  
big\_384\_58.h, 83

BIG\_384\_58\_random  
big\_384\_58.h, 84

BIG\_384\_58\_randomnum  
big\_384\_58.h, 84

BIG\_384\_58\_rawoutput  
big\_384\_58.h, 84

BIG\_384\_58\_rcopy  
big\_384\_58.h, 84

BIG\_384\_58\_sdcopy  
big\_384\_58.h, 85

BIG\_384\_58\_sdiv  
big\_384\_58.h, 85

BIG\_384\_58\_sdcopy  
big\_384\_58.h, 85

BIG\_384\_58\_shl  
big\_384\_58.h, 86

BIG\_384\_58\_shr  
big\_384\_58.h, 86

BIG\_384\_58\_smul  
big\_384\_58.h, 86

BIG\_384\_58\_split  
big\_384\_58.h, 86

BIG\_384\_58\_sqr  
big\_384\_58.h, 87

BIG\_384\_58\_ssn  
big\_384\_58.h, 87

BIG\_384\_58\_sub  
big\_384\_58.h, 88

BIG\_384\_58\_toBytes  
big\_384\_58.h, 88

BIG\_384\_58\_zero  
big\_384\_58.h, 88

BIG\_512\_60  
big\_512\_60.h, 93

BIG\_512\_60\_add  
big\_512\_60.h, 93

BIG\_512\_60\_bit  
big\_512\_60.h, 94

BIG\_512\_60\_cmove  
big\_512\_60.h, 94

BIG\_512\_60\_comp  
big\_512\_60.h, 94

BIG\_512\_60\_copy  
big\_512\_60.h, 95

BIG\_512\_60\_cswap  
big\_512\_60.h, 95

BIG\_512\_60\_dadd  
big\_512\_60.h, 95

BIG\_512\_60\_dcmove  
big\_512\_60.h, 96

BIG\_512\_60\_dcomp  
big\_512\_60.h, 96

BIG\_512\_60\_dcopy  
big\_512\_60.h, 96

BIG\_512\_60\_ddiv  
big\_512\_60.h, 97

BIG\_512\_60\_dec  
big\_512\_60.h, 97

BIG\_512\_60\_dfromBytesLen  
big\_512\_60.h, 97

BIG\_512\_60\_diszilch  
big\_512\_60.h, 98

BIG\_512\_60\_div3  
big\_512\_60.h, 98

BIG\_512\_60\_dmod  
big\_512\_60.h, 98

BIG\_512\_60\_dmod2m  
big\_512\_60.h, 99

BIG\_512\_60\_dnbits  
big\_512\_60.h, 99

BIG\_512\_60\_dnorm  
big\_512\_60.h, 99

BIG\_512\_60\_doutput  
big\_512\_60.h, 99

BIG\_512\_60\_drawoutput  
big\_512\_60.h, 100

BIG\_512\_60\_dscopy  
big\_512\_60.h, 100

BIG\_512\_60\_dshl  
big\_512\_60.h, 100

BIG\_512\_60\_dshr  
big\_512\_60.h, 100

BIG\_512\_60\_dsub  
big\_512\_60.h, 101

BIG\_512\_60\_dsucopy  
big\_512\_60.h, 101

BIG\_512\_60\_dzero  
big\_512\_60.h, 101

BIG\_512\_60\_fromBytes  
big\_512\_60.h, 102

BIG\_512\_60\_fromBytesLen  
big\_512\_60.h, 102

BIG\_512\_60\_fshl  
big\_512\_60.h, 102

BIG\_512\_60\_fshr  
big\_512\_60.h, 103

BIG\_512\_60\_imul  
big\_512\_60.h, 103

BIG\_512\_60\_inc  
big\_512\_60.h, 103

BIG\_512\_60\_invmod2m  
big\_512\_60.h, 104

BIG\_512\_60\_invmodp  
big\_512\_60.h, 104

BIG\_512\_60\_isunity  
big\_512\_60.h, 104

BIG\_512\_60\_izsilch  
big\_512\_60.h, 104

BIG\_512\_60\_jacobi  
big\_512\_60.h, 106

BIG\_512\_60\_lastbits  
big\_512\_60.h, 106

BIG\_512\_60\_mod  
big\_512\_60.h, 106

BIG\_512\_60\_mod2m  
big\_512\_60.h, 107

BIG\_512\_60\_moddiv  
big\_512\_60.h, 107

BIG\_512\_60\_modmul  
big\_512\_60.h, 107

BIG\_512\_60\_modneg  
big\_512\_60.h, 108

BIG\_512\_60\_modsqr  
big\_512\_60.h, 108

BIG\_512\_60\_monty  
big\_512\_60.h, 109

BIG\_512\_60\_mul  
big\_512\_60.h, 109

BIG\_512\_60\_nbits  
big\_512\_60.h, 109

BIG\_512\_60\_norm  
big\_512\_60.h, 109

BIG\_512\_60\_one  
big\_512\_60.h, 110

BIG\_512\_60\_or  
  big\_512\_60.h, 110

BIG\_512\_60\_output  
  big\_512\_60.h, 110

BIG\_512\_60\_parity  
  big\_512\_60.h, 111

BIG\_512\_60\_pmul  
  big\_512\_60.h, 111

BIG\_512\_60\_pxmuls  
  big\_512\_60.h, 111

BIG\_512\_60\_random  
  big\_512\_60.h, 112

BIG\_512\_60\_randomnum  
  big\_512\_60.h, 112

BIG\_512\_60\_rawoutput  
  big\_512\_60.h, 112

BIG\_512\_60\_rcopy  
  big\_512\_60.h, 112

BIG\_512\_60\_sdcopy  
  big\_512\_60.h, 113

BIG\_512\_60\_sdiv  
  big\_512\_60.h, 113

BIG\_512\_60\_sdcopy  
  big\_512\_60.h, 113

BIG\_512\_60\_shl  
  big\_512\_60.h, 114

BIG\_512\_60\_shr  
  big\_512\_60.h, 114

BIG\_512\_60\_smul  
  big\_512\_60.h, 114

BIG\_512\_60\_split  
  big\_512\_60.h, 114

BIG\_512\_60\_sqr  
  big\_512\_60.h, 115

BIG\_512\_60\_ssn  
  big\_512\_60.h, 115

BIG\_512\_60\_sub  
  big\_512\_60.h, 116

BIG\_512\_60\_toBytes  
  big\_512\_60.h, 116

BIG\_512\_60\_zero  
  big\_512\_60.h, 116

BIGBITS\_1024\_58  
  big\_1024\_58.h, 37

BIGBITS\_384\_58  
  big\_384\_58.h, 65

BIGBITS\_512\_60  
  big\_512\_60.h, 92

BLS\_BLS381\_ADD\_G1  
  bls\_BLS381.h, 118

BLS\_BLS381\_ADD\_G2  
  bls\_BLS381.h, 119

BLS\_BLS381\_KEY\_PAIR\_GENERATE  
  bls\_BLS381.h, 119

BLS\_BLS381\_MAKE\_SHARES  
  bls\_BLS381.h, 119

BLS\_BLS381\_RECOVER\_SECRET  
  bls\_BLS381.h, 120

BLS\_BLS381\_RECOVER\_SIGNATURE  
  bls\_BLS381.h, 120

BLS\_BLS381\_SIGN  
  bls\_BLS381.h, 121

BLS\_BLS381\_VERIFY  
  bls\_BLS381.h, 121

BLS\_FAIL  
  bls\_BLS381.h, 118

BLS\_INVALID\_G1  
  bls\_BLS381.h, 118

BLS\_INVALID\_G2  
  bls\_BLS381.h, 118

BLS\_OK  
  bls\_BLS381.h, 118

BMASK\_1024\_58  
  big\_1024\_58.h, 37

BMASK\_384\_58  
  big\_384\_58.h, 65

BMASK\_512\_60  
  big\_512\_60.h, 92

big\_1024\_58.h, 33

  BIG\_1024\_58, 38

  BIG\_1024\_58\_add, 38

  BIG\_1024\_58\_bit, 38

  BIG\_1024\_58\_cmove, 39

  BIG\_1024\_58\_comp, 39

  BIG\_1024\_58\_copy, 39

  BIG\_1024\_58\_cswap, 40

  BIG\_1024\_58\_dadd, 40

  BIG\_1024\_58\_dcmove, 40

  BIG\_1024\_58\_dcomp, 41

  BIG\_1024\_58\_dcopy, 41

  BIG\_1024\_58\_ddiv, 41

  BIG\_1024\_58\_dec, 42

  BIG\_1024\_58\_dfromBytesLen, 42

  BIG\_1024\_58\_diszilch, 42

  BIG\_1024\_58\_div3, 42

  BIG\_1024\_58\_dmod, 44

  BIG\_1024\_58\_dmod2m, 44

  BIG\_1024\_58\_dnbits, 44

  BIG\_1024\_58\_dnorm, 45

  BIG\_1024\_58\_doutput, 45

  BIG\_1024\_58\_drawoutput, 45

  BIG\_1024\_58\_dscopy, 45

  BIG\_1024\_58\_dshl, 46

  BIG\_1024\_58\_dshr, 46

  BIG\_1024\_58\_dsub, 46

  BIG\_1024\_58\_dsucopy, 47

  BIG\_1024\_58\_dzero, 47

  BIG\_1024\_58\_fromBytes, 47

  BIG\_1024\_58\_fromBytesLen, 47

  BIG\_1024\_58\_fshl, 48

  BIG\_1024\_58\_fshr, 48

  BIG\_1024\_58\_imul, 49

  BIG\_1024\_58\_inc, 49

  BIG\_1024\_58\_invmod2m, 49

  BIG\_1024\_58\_invmodp, 49

  BIG\_1024\_58\_isunity, 50

- BIG\_1024\_58\_iszilch, [50](#)
- BIG\_1024\_58\_jacobi, [50](#)
- BIG\_1024\_58\_lastbits, [51](#)
- BIG\_1024\_58\_mod, [51](#)
- BIG\_1024\_58\_mod2m, [51](#)
- BIG\_1024\_58\_moddiv, [52](#)
- BIG\_1024\_58\_modmul, [52](#)
- BIG\_1024\_58\_modneg, [53](#)
- BIG\_1024\_58\_modsqr, [53](#)
- BIG\_1024\_58\_monty, [53](#)
- BIG\_1024\_58\_mul, [54](#)
- BIG\_1024\_58\_nbits, [54](#)
- BIG\_1024\_58\_norm, [54](#)
- BIG\_1024\_58\_one, [54](#)
- BIG\_1024\_58\_or, [55](#)
- BIG\_1024\_58\_output, [55](#)
- BIG\_1024\_58\_parity, [55](#)
- BIG\_1024\_58\_pmul, [56](#)
- BIG\_1024\_58\_pxm, [56](#)
- BIG\_1024\_58\_random, [56](#)
- BIG\_1024\_58\_randomnum, [57](#)
- BIG\_1024\_58\_rawoutput, [57](#)
- BIG\_1024\_58\_rcopy, [57](#)
- BIG\_1024\_58\_sdcopy, [57](#)
- BIG\_1024\_58\_sdiv, [58](#)
- BIG\_1024\_58\_sdcopy, [58](#)
- BIG\_1024\_58\_shl, [58](#)
- BIG\_1024\_58\_shr, [59](#)
- BIG\_1024\_58\_smul, [59](#)
- BIG\_1024\_58\_split, [59](#)
- BIG\_1024\_58\_sqr, [60](#)
- BIG\_1024\_58\_ssn, [60](#)
- BIG\_1024\_58\_sub, [60](#)
- BIG\_1024\_58\_toBytes, [61](#)
- BIG\_1024\_58\_zero, [61](#)
- BIGBITS\_1024\_58, [37](#)
- BMASK\_1024\_58, [37](#)
- DBIG\_1024\_58, [38](#)
- DNLEN\_1024\_58, [37](#)
- HBITS\_1024\_58, [37](#)
- HMASK\_1024\_58, [37](#)
- NEXCESS\_1024\_58, [37](#)
- NLEN\_1024\_58, [37](#)
- big\_384\_58.h, [61](#)
- BIG\_384\_58, [66](#)
- BIG\_384\_58\_add, [66](#)
- BIG\_384\_58\_bit, [67](#)
- BIG\_384\_58\_cmove, [67](#)
- BIG\_384\_58\_comp, [67](#)
- BIG\_384\_58\_copy, [68](#)
- BIG\_384\_58\_cswap, [68](#)
- BIG\_384\_58\_dadd, [68](#)
- BIG\_384\_58\_dcmove, [69](#)
- BIG\_384\_58\_dcomp, [69](#)
- BIG\_384\_58\_dcopy, [69](#)
- BIG\_384\_58\_ddiv, [70](#)
- BIG\_384\_58\_dec, [70](#)
- BIG\_384\_58\_dfromBytesLen, [70](#)
- BIG\_384\_58\_diszilch, [70](#)
- BIG\_384\_58\_div3, [71](#)
- BIG\_384\_58\_dmod, [71](#)
- BIG\_384\_58\_dmod2m, [71](#)
- BIG\_384\_58\_dnbits, [72](#)
- BIG\_384\_58\_dnorm, [72](#)
- BIG\_384\_58\_doutput, [72](#)
- BIG\_384\_58\_drawoutput, [73](#)
- BIG\_384\_58\_dscopy, [73](#)
- BIG\_384\_58\_dshl, [73](#)
- BIG\_384\_58\_dshr, [73](#)
- BIG\_384\_58\_dsub, [74](#)
- BIG\_384\_58\_dsucopy, [74](#)
- BIG\_384\_58\_dzero, [74](#)
- BIG\_384\_58\_fromBytes, [75](#)
- BIG\_384\_58\_fromBytesLen, [75](#)
- BIG\_384\_58\_fshl, [75](#)
- BIG\_384\_58\_fshr, [75](#)
- BIG\_384\_58\_imul, [76](#)
- BIG\_384\_58\_inc, [76](#)
- BIG\_384\_58\_invmod2m, [76](#)
- BIG\_384\_58\_invmodp, [77](#)
- BIG\_384\_58\_isunity, [77](#)
- BIG\_384\_58\_iszilch, [77](#)
- BIG\_384\_58\_jacobi, [78](#)
- BIG\_384\_58\_lastbits, [78](#)
- BIG\_384\_58\_mod, [78](#)
- BIG\_384\_58\_mod2m, [79](#)
- BIG\_384\_58\_moddiv, [79](#)
- BIG\_384\_58\_modmul, [79](#)
- BIG\_384\_58\_modneg, [80](#)
- BIG\_384\_58\_modsqr, [80](#)
- BIG\_384\_58\_monty, [81](#)
- BIG\_384\_58\_mul, [81](#)
- BIG\_384\_58\_nbits, [81](#)
- BIG\_384\_58\_norm, [81](#)
- BIG\_384\_58\_one, [82](#)
- BIG\_384\_58\_or, [82](#)
- BIG\_384\_58\_output, [82](#)
- BIG\_384\_58\_parity, [83](#)
- BIG\_384\_58\_pmul, [83](#)
- BIG\_384\_58\_pxm, [83](#)
- BIG\_384\_58\_random, [84](#)
- BIG\_384\_58\_randomnum, [84](#)
- BIG\_384\_58\_rawoutput, [84](#)
- BIG\_384\_58\_rcopy, [84](#)
- BIG\_384\_58\_sdcopy, [85](#)
- BIG\_384\_58\_sdiv, [85](#)
- BIG\_384\_58\_sdcopy, [85](#)
- BIG\_384\_58\_shl, [86](#)
- BIG\_384\_58\_shr, [86](#)
- BIG\_384\_58\_smul, [86](#)
- BIG\_384\_58\_split, [86](#)
- BIG\_384\_58\_sqr, [87](#)
- BIG\_384\_58\_ssn, [87](#)
- BIG\_384\_58\_sub, [88](#)
- BIG\_384\_58\_toBytes, [88](#)
- BIG\_384\_58\_zero, [88](#)

- BIGBITS\_384\_58, 65
- BMASK\_384\_58, 65
- DBIG\_384\_58, 66
- DNLEN\_384\_58, 65
- HBITS\_384\_58, 65
- HMASK\_384\_58, 65
- NEXCESS\_384\_58, 65
- NLEN\_384\_58, 66
- big\_512\_60.h, 88
  - BIG\_512\_60, 93
  - BIG\_512\_60\_add, 93
  - BIG\_512\_60\_bit, 94
  - BIG\_512\_60\_cmove, 94
  - BIG\_512\_60\_comp, 94
  - BIG\_512\_60\_copy, 95
  - BIG\_512\_60\_cswap, 95
  - BIG\_512\_60\_dadd, 95
  - BIG\_512\_60\_dcmove, 96
  - BIG\_512\_60\_dcomp, 96
  - BIG\_512\_60\_dcopy, 96
  - BIG\_512\_60\_ddiv, 97
  - BIG\_512\_60\_dec, 97
  - BIG\_512\_60\_dfromBytesLen, 97
  - BIG\_512\_60\_diszilch, 98
  - BIG\_512\_60\_div3, 98
  - BIG\_512\_60\_dmod, 98
  - BIG\_512\_60\_dmod2m, 99
  - BIG\_512\_60\_dnbits, 99
  - BIG\_512\_60\_dnorm, 99
  - BIG\_512\_60\_doutput, 99
  - BIG\_512\_60\_drawoutput, 100
  - BIG\_512\_60\_dscopy, 100
  - BIG\_512\_60\_dshl, 100
  - BIG\_512\_60\_dshr, 100
  - BIG\_512\_60\_dsub, 101
  - BIG\_512\_60\_dscopy, 101
  - BIG\_512\_60\_dzero, 101
  - BIG\_512\_60\_fromBytes, 102
  - BIG\_512\_60\_fromBytesLen, 102
  - BIG\_512\_60\_fshl, 102
  - BIG\_512\_60\_fshr, 103
  - BIG\_512\_60\_imul, 103
  - BIG\_512\_60\_inc, 103
  - BIG\_512\_60\_invmod2m, 104
  - BIG\_512\_60\_invmodp, 104
  - BIG\_512\_60\_isunity, 104
  - BIG\_512\_60\_iszilch, 104
  - BIG\_512\_60\_jacobi, 106
  - BIG\_512\_60\_lastbits, 106
  - BIG\_512\_60\_mod, 106
  - BIG\_512\_60\_mod2m, 107
  - BIG\_512\_60\_moddiv, 107
  - BIG\_512\_60\_modmul, 107
  - BIG\_512\_60\_modneg, 108
  - BIG\_512\_60\_modsqr, 108
  - BIG\_512\_60\_monty, 109
  - BIG\_512\_60\_mul, 109
  - BIG\_512\_60\_nbits, 109
  - BIG\_512\_60\_norm, 109
  - BIG\_512\_60\_one, 110
  - BIG\_512\_60\_or, 110
  - BIG\_512\_60\_output, 110
  - BIG\_512\_60\_parity, 111
  - BIG\_512\_60\_pmul, 111
  - BIG\_512\_60\_pxm, 111
  - BIG\_512\_60\_random, 112
  - BIG\_512\_60\_randomnum, 112
  - BIG\_512\_60\_rawoutput, 112
  - BIG\_512\_60\_rcopy, 112
  - BIG\_512\_60\_sdcopy, 113
  - BIG\_512\_60\_sdiv, 113
  - BIG\_512\_60\_sducopy, 113
  - BIG\_512\_60\_shl, 114
  - BIG\_512\_60\_shr, 114
  - BIG\_512\_60\_smul, 114
  - BIG\_512\_60\_split, 114
  - BIG\_512\_60\_sqr, 115
  - BIG\_512\_60\_ssn, 115
  - BIG\_512\_60\_sub, 116
  - BIG\_512\_60\_toBytes, 116
  - BIG\_512\_60\_zero, 116
  - BIGBITS\_512\_60, 92
  - BMASK\_512\_60, 92
  - DBIG\_512\_60, 93
  - DNLEN\_512\_60, 92
  - HBITS\_512\_60, 92
  - HMASK\_512\_60, 93
  - NEXCESS\_512\_60, 93
  - NLEN\_512\_60, 93
- bls\_BLS381.h, 116
  - BFS\_BLS381, 117
  - BGS\_BLS381, 117
  - BLS\_BLS381\_ADD\_G1, 118
  - BLS\_BLS381\_ADD\_G2, 119
  - BLS\_BLS381\_KEY\_PAIR\_GENERATE, 119
  - BLS\_BLS381\_MAKE\_SHARES, 119
  - BLS\_BLS381\_RECOVER\_SECRET, 120
  - BLS\_BLS381\_RECOVER\_SIGNATURE, 120
  - BLS\_BLS381\_SIGN, 121
  - BLS\_BLS381\_VERIFY, 121
  - BLS\_FAIL, 118
  - BLS\_INVALID\_G1, 118
  - BLS\_INVALID\_G2, 118
  - BLS\_OK, 118
- borrow
  - csprng, 14
- byte
  - arch.h, 31
- c
  - FP12\_BLS381, 18
  - rsa\_private\_key\_2048, 28
- CHUNK
  - arch.h, 32
- CREATE\_CSPRNG
  - randapi.h, 270
- CURVE\_A\_BLS381

- ecp2\_BLS381.h, [142](#)
- ecp\_BLS381.h, [155](#)
- CURVE\_B\_BLS381
  - ecp2\_BLS381.h, [142](#)
  - ecp\_BLS381.h, [155](#)
- CURVE\_B\_I\_BLS381
  - ecp2\_BLS381.h, [142](#)
  - ecp\_BLS381.h, [155](#)
- CURVE\_BB\_BLS381
  - ecp\_BLS381.h, [155](#)
  - pair\_BLS381.h, [265](#)
- CURVE\_Bnx\_BLS381
  - ecp2\_BLS381.h, [143](#)
  - ecp\_BLS381.h, [155](#)
  - pair\_BLS381.h, [265](#)
- CURVE\_Cof\_BLS381
  - ecp2\_BLS381.h, [143](#)
  - ecp\_BLS381.h, [155](#)
- CURVE\_Cof\_I\_BLS381
  - ecp\_BLS381.h, [155](#)
- CURVE\_Cru\_BLS381
  - ecp\_BLS381.h, [156](#)
  - pair\_BLS381.h, [265](#)
- CURVE\_Gx\_BLS381
  - ecp2\_BLS381.h, [143](#)
  - ecp\_BLS381.h, [156](#)
- CURVE\_Gy\_BLS381
  - ecp2\_BLS381.h, [143](#)
  - ecp\_BLS381.h, [156](#)
- CURVE\_Order\_BLS381
  - ecp2\_BLS381.h, [143](#)
  - ecp\_BLS381.h, [156](#)
- CURVE\_Pxa\_BLS381
  - ecp2\_BLS381.h, [143](#)
  - ecp\_BLS381.h, [156](#)
- CURVE\_Pxaa\_BLS381
  - ecp\_BLS381.h, [156](#)
- CURVE\_Pxaaa\_BLS381
  - ecp\_BLS381.h, [156](#)
- CURVE\_Pxaab\_BLS381
  - ecp\_BLS381.h, [156](#)
- CURVE\_Pxab\_BLS381
  - ecp\_BLS381.h, [157](#)
- CURVE\_Pxaba\_BLS381
  - ecp\_BLS381.h, [157](#)
- CURVE\_Pxabbb\_BLS381
  - ecp\_BLS381.h, [157](#)
- CURVE\_Pxb\_BLS381
  - ecp2\_BLS381.h, [143](#)
  - ecp\_BLS381.h, [157](#)
- CURVE\_Pxba\_BLS381
  - ecp\_BLS381.h, [157](#)
- CURVE\_Pxbaa\_BLS381
  - ecp\_BLS381.h, [157](#)
- CURVE\_Pxbab\_BLS381
  - ecp\_BLS381.h, [157](#)
- CURVE\_Pxbb\_BLS381
  - ecp\_BLS381.h, [157](#)
- CURVE\_Pxbba\_BLS381
  - ecp\_BLS381.h, [158](#)
- CURVE\_Pxbbb\_BLS381
  - ecp\_BLS381.h, [158](#)
- CURVE\_Pya\_BLS381
  - ecp2\_BLS381.h, [143](#)
  - ecp\_BLS381.h, [158](#)
- CURVE\_Pyaa\_BLS381
  - ecp\_BLS381.h, [158](#)
- CURVE\_Pyaaa\_BLS381
  - ecp\_BLS381.h, [158](#)
- CURVE\_Pyaab\_BLS381
  - ecp\_BLS381.h, [158](#)
- CURVE\_Pyab\_BLS381
  - ecp\_BLS381.h, [158](#)
- CURVE\_Pyaba\_BLS381
  - ecp\_BLS381.h, [158](#)
- CURVE\_Pyabb\_BLS381
  - ecp\_BLS381.h, [159](#)
- CURVE\_Pyb\_BLS381
  - ecp2\_BLS381.h, [144](#)
  - ecp\_BLS381.h, [159](#)
- CURVE\_Pyba\_BLS381
  - ecp\_BLS381.h, [159](#)
- CURVE\_Pybaa\_BLS381
  - ecp\_BLS381.h, [159](#)
- CURVE\_Pybab\_BLS381
  - ecp\_BLS381.h, [159](#)
- CURVE\_Pybb\_BLS381
  - ecp\_BLS381.h, [159](#)
- CURVE\_Pybbba\_BLS381
  - ecp\_BLS381.h, [159](#)
- CURVE\_Pybbb\_BLS381
  - ecp\_BLS381.h, [159](#)
- CURVE\_SB\_BLS381
  - ecp\_BLS381.h, [160](#)
  - pair\_BLS381.h, [265](#)
- CURVE\_W\_BLS381
  - ecp\_BLS381.h, [160](#)
  - pair\_BLS381.h, [265](#)
- CURVE\_WB\_BLS381
  - ecp\_BLS381.h, [160](#)
  - pair\_BLS381.h, [265](#)
- chunk
  - arch.h, [32](#)
- config\_big\_1024\_58.h, [122](#)
  - BASEBITS\_1024\_58, [122](#)
  - MODBYTES\_1024\_58, [122](#)
- config\_big\_384\_58.h, [122](#)
  - BASEBITS\_384\_58, [123](#)
  - MODBYTES\_384\_58, [123](#)
- config\_big\_512\_60.h, [123](#)
  - BASEBITS\_512\_60, [124](#)
  - MODBYTES\_512\_60, [124](#)
- config\_ff\_2048.h, [124](#)
  - FFLEN\_2048, [124](#)
- config\_ff\_4096.h, [125](#)
  - FFLEN\_4096, [125](#)



- csprng, [14](#)
  - borrow, [14](#)
  - ira, [15](#)
  - pool, [15](#)
  - pool\_ptr, [15](#)
  - rndptr, [15](#)
- curve
  - pktype, [27](#)
- DBIG\_1024\_58
  - big\_1024\_58.h, [38](#)
- DBIG\_384\_58
  - big\_384\_58.h, [66](#)
- DBIG\_512\_60
  - big\_512\_60.h, [93](#)
- DNLEN\_1024\_58
  - big\_1024\_58.h, [37](#)
- DNLEN\_384\_58
  - big\_384\_58.h, [65](#)
- DNLEN\_512\_60
  - big\_512\_60.h, [92](#)
- dp
  - rsa\_private\_key\_2048, [28](#)
- dq
  - rsa\_private\_key\_2048, [29](#)
- e
  - rsa\_public\_key\_2048, [29](#)
- ECDH\_ERROR
  - ecdh\_BLS381.h, [126](#)
- ECDH\_INVALID\_PUBLIC\_KEY
  - ecdh\_BLS381.h, [126](#)
- ECDH\_INVALID
  - ecdh\_BLS381.h, [126](#)
- ECDH\_OK
  - ecdh\_BLS381.h, [126](#)
- ECP2\_BLS381, [15](#)
  - x, [16](#)
  - y, [16](#)
  - z, [16](#)
- ECP2\_BLS381\_add
  - ecp2\_BLS381.h, [136](#)
- ECP2\_BLS381\_affine
  - ecp2\_BLS381.h, [136](#)
- ECP2\_BLS381\_copy
  - ecp2\_BLS381.h, [136](#)
- ECP2\_BLS381\_dbl
  - ecp2\_BLS381.h, [136](#)
- ECP2\_BLS381\_equals
  - ecp2\_BLS381.h, [137](#)
- ECP2\_BLS381\_frob
  - ecp2\_BLS381.h, [137](#)
- ECP2\_BLS381\_fromOctet
  - ecp2\_BLS381.h, [137](#)
- ECP2\_BLS381\_generator
  - ecp2\_BLS381.h, [138](#)
- ECP2\_BLS381\_get
  - ecp2\_BLS381.h, [138](#)
- ECP2\_BLS381\_inf
  - ecp2\_BLS381.h, [138](#)
- ECP2\_BLS381\_isinf
  - ecp2\_BLS381.h, [139](#)
- ECP2\_BLS381\_mapit
  - ecp2\_BLS381.h, [139](#)
- ECP2\_BLS381\_mul
  - ecp2\_BLS381.h, [139](#)
- ECP2\_BLS381\_mul4
  - ecp2\_BLS381.h, [140](#)
- ECP2\_BLS381\_neg
  - ecp2\_BLS381.h, [140](#)
- ECP2\_BLS381\_output
  - ecp2\_BLS381.h, [140](#)
- ECP2\_BLS381\_outputxyz
  - ecp2\_BLS381.h, [140](#)
- ECP2\_BLS381\_rhs
  - ecp2\_BLS381.h, [141](#)
- ECP2\_BLS381\_set
  - ecp2\_BLS381.h, [141](#)
- ECP2\_BLS381\_setx
  - ecp2\_BLS381.h, [141](#)
- ECP2\_BLS381\_sub
  - ecp2\_BLS381.h, [142](#)
- ECP2\_BLS381\_toOctet
  - ecp2\_BLS381.h, [142](#)
- ECP\_BLS381, [16](#)
  - x, [16](#)
  - y, [17](#)
  - z, [17](#)
- ECP\_BLS381\_ECIES\_DECRYPT
  - ecdh\_BLS381.h, [127](#)
- ECP\_BLS381\_ECIES\_ENCRYPT
  - ecdh\_BLS381.h, [128](#)
- ECP\_BLS381\_KEY\_PAIR\_GENERATE
  - ecdh\_BLS381.h, [128](#)
- ECP\_BLS381\_PUBLIC\_KEY\_VALIDATE
  - ecdh\_BLS381.h, [129](#)
- ECP\_BLS381\_SP\_DSA
  - ecdh\_BLS381.h, [129](#)
- ECP\_BLS381\_SVDP\_DH
  - ecdh\_BLS381.h, [129](#)
- ECP\_BLS381\_VP\_DSA
  - ecdh\_BLS381.h, [130](#)
- ECP\_BLS381\_add
  - ecp\_BLS381.h, [147](#)
- ECP\_BLS381\_affine
  - ecp\_BLS381.h, [147](#)
- ECP\_BLS381\_cfp
  - ecp\_BLS381.h, [147](#)
- ECP\_BLS381\_copy
  - ecp\_BLS381.h, [147](#)
- ECP\_BLS381\_dbl
  - ecp\_BLS381.h, [148](#)
- ECP\_BLS381\_equals
  - ecp\_BLS381.h, [148](#)
- ECP\_BLS381\_fromOctet
  - ecp\_BLS381.h, [148](#)
- ECP\_BLS381\_generator

ecp\_BLS381.h, 149  
 ECP\_BLS381\_get  
   ecp\_BLS381.h, 149  
 ECP\_BLS381\_inf  
   ecp\_BLS381.h, 149  
 ECP\_BLS381\_isinf  
   ecp\_BLS381.h, 149  
 ECP\_BLS381\_mapit  
   ecp\_BLS381.h, 150  
 ECP\_BLS381\_mul  
   ecp\_BLS381.h, 150  
 ECP\_BLS381\_mul2  
   ecp\_BLS381.h, 150  
 ECP\_BLS381\_neg  
   ecp\_BLS381.h, 151  
 ECP\_BLS381\_output  
   ecp\_BLS381.h, 151  
 ECP\_BLS381\_outputxyz  
   ecp\_BLS381.h, 151  
 ECP\_BLS381\_pinmul  
   ecp\_BLS381.h, 151  
 ECP\_BLS381\_rawoutput  
   ecp\_BLS381.h, 152  
 ECP\_BLS381\_rhs  
   ecp\_BLS381.h, 152  
 ECP\_BLS381\_set  
   ecp\_BLS381.h, 152  
 ECP\_BLS381\_setx  
   ecp\_BLS381.h, 153  
 ECP\_BLS381\_sub  
   ecp\_BLS381.h, 153  
 ECP\_BLS381\_toOctet  
   ecp\_BLS381.h, 153  
 EFS\_BLS381  
   ecdh\_BLS381.h, 127  
 EGS\_BLS381  
   ecdh\_BLS381.h, 127  
 ecdh\_BLS381.h, 125  
   ECDH\_ERROR, 126  
   ECDH\_INVALID\_PUBLIC\_KEY, 126  
   ECDH\_INVALID, 126  
   ECDH\_OK, 126  
   ECP\_BLS381\_ECIES\_DECRYPT, 127  
   ECP\_BLS381\_ECIES\_ENCRYPT, 128  
   ECP\_BLS381\_KEY\_PAIR\_GENERATE, 128  
   ECP\_BLS381\_PUBLIC\_KEY\_VALIDATE, 129  
   ECP\_BLS381\_SP\_DSA, 129  
   ECP\_BLS381\_SVDP\_DH, 129  
   ECP\_BLS381\_VP\_DSA, 130  
   EFS\_BLS381, 127  
   EGS\_BLS381, 127  
 ecdh\_support.h, 130  
   AES\_CBC\_IV0\_DECRYPT, 131  
   AES\_CBC\_IV0\_ENCRYPT, 131  
   ehashit, 132  
   HASH, 132  
   HMAC, 133  
   KDF2, 133  
   PBKDF2, 133  
 ecp2\_BLS381.h, 134  
   CURVE\_A\_BLS381, 142  
   CURVE\_B\_BLS381, 142  
   CURVE\_B\_I\_BLS381, 142  
   CURVE\_Bnx\_BLS381, 143  
   CURVE\_Cof\_BLS381, 143  
   CURVE\_Gx\_BLS381, 143  
   CURVE\_Gy\_BLS381, 143  
   CURVE\_Order\_BLS381, 143  
   CURVE\_Pxa\_BLS381, 143  
   CURVE\_Pxb\_BLS381, 143  
   CURVE\_Pya\_BLS381, 143  
   CURVE\_Pyb\_BLS381, 144  
   ECP2\_BLS381\_add, 136  
   ECP2\_BLS381\_affine, 136  
   ECP2\_BLS381\_copy, 136  
   ECP2\_BLS381\_dbl, 136  
   ECP2\_BLS381\_equals, 137  
   ECP2\_BLS381\_frob, 137  
   ECP2\_BLS381\_fromOctet, 137  
   ECP2\_BLS381\_generator, 138  
   ECP2\_BLS381\_get, 138  
   ECP2\_BLS381\_inf, 138  
   ECP2\_BLS381\_isinf, 139  
   ECP2\_BLS381\_mapit, 139  
   ECP2\_BLS381\_mul, 139  
   ECP2\_BLS381\_mul4, 140  
   ECP2\_BLS381\_neg, 140  
   ECP2\_BLS381\_output, 140  
   ECP2\_BLS381\_outputxyz, 140  
   ECP2\_BLS381\_rhs, 141  
   ECP2\_BLS381\_set, 141  
   ECP2\_BLS381\_setx, 141  
   ECP2\_BLS381\_sub, 142  
   ECP2\_BLS381\_toOctet, 142  
   Fra\_BLS381, 144  
   Frb\_BLS381, 144  
 ecp\_BLS381.h, 144  
   CURVE\_A\_BLS381, 155  
   CURVE\_B\_BLS381, 155  
   CURVE\_B\_I\_BLS381, 155  
   CURVE\_BB\_BLS381, 155  
   CURVE\_Bnx\_BLS381, 155  
   CURVE\_Cof\_BLS381, 155  
   CURVE\_Cof\_I\_BLS381, 155  
   CURVE\_Cru\_BLS381, 156  
   CURVE\_Gx\_BLS381, 156  
   CURVE\_Gy\_BLS381, 156  
   CURVE\_Order\_BLS381, 156  
   CURVE\_Pxa\_BLS381, 156  
   CURVE\_Pxaa\_BLS381, 156  
   CURVE\_Pxaaa\_BLS381, 156  
   CURVE\_Pxaab\_BLS381, 156  
   CURVE\_Pxab\_BLS381, 157  
   CURVE\_Pxaba\_BLS381, 157  
   CURVE\_Pxabb\_BLS381, 157  
   CURVE\_Pxb\_BLS381, 157

- CURVE\_Pxba\_BLS381, [157](#)
- CURVE\_Pxbaa\_BLS381, [157](#)
- CURVE\_Pxbab\_BLS381, [157](#)
- CURVE\_Pxbb\_BLS381, [157](#)
- CURVE\_Pxbba\_BLS381, [158](#)
- CURVE\_Pxbbb\_BLS381, [158](#)
- CURVE\_Pya\_BLS381, [158](#)
- CURVE\_Pyaa\_BLS381, [158](#)
- CURVE\_Pyaaa\_BLS381, [158](#)
- CURVE\_Pyaab\_BLS381, [158](#)
- CURVE\_Pyab\_BLS381, [158](#)
- CURVE\_Pyaba\_BLS381, [158](#)
- CURVE\_Pyabb\_BLS381, [159](#)
- CURVE\_Pyb\_BLS381, [159](#)
- CURVE\_Pyba\_BLS381, [159](#)
- CURVE\_Pybaa\_BLS381, [159](#)
- CURVE\_Pybab\_BLS381, [159](#)
- CURVE\_Pybb\_BLS381, [159](#)
- CURVE\_Pybba\_BLS381, [159](#)
- CURVE\_Pybbb\_BLS381, [159](#)
- CURVE\_SB\_BLS381, [160](#)
- CURVE\_W\_BLS381, [160](#)
- CURVE\_WB\_BLS381, [160](#)
- ECP\_BLS381\_add, [147](#)
- ECP\_BLS381\_affine, [147](#)
- ECP\_BLS381\_cfp, [147](#)
- ECP\_BLS381\_copy, [147](#)
- ECP\_BLS381\_dbl, [148](#)
- ECP\_BLS381\_equals, [148](#)
- ECP\_BLS381\_fromOctet, [148](#)
- ECP\_BLS381\_generator, [149](#)
- ECP\_BLS381\_get, [149](#)
- ECP\_BLS381\_inf, [149](#)
- ECP\_BLS381\_isinf, [149](#)
- ECP\_BLS381\_mapit, [150](#)
- ECP\_BLS381\_mul, [150](#)
- ECP\_BLS381\_mul2, [150](#)
- ECP\_BLS381\_neg, [151](#)
- ECP\_BLS381\_output, [151](#)
- ECP\_BLS381\_outputxyz, [151](#)
- ECP\_BLS381\_pinmul, [151](#)
- ECP\_BLS381\_rawoutput, [152](#)
- ECP\_BLS381\_rhs, [152](#)
- ECP\_BLS381\_set, [152](#)
- ECP\_BLS381\_setx, [153](#)
- ECP\_BLS381\_sub, [153](#)
- ECP\_BLS381\_toOctet, [153](#)
- Fra\_BLS381, [160](#)
- Frb\_BLS381, [160](#)
- ehashit
  - ecdh\_support.h, [132](#)
- f
  - amcl\_aes, [13](#)
- FEXCESS\_BLS381
  - fp\_BLS381.h, [230](#)
- FF\_2048\_add
  - ff\_2048.h, [163](#)
- FF\_2048\_cfactor
  - ff\_2048.h, [163](#)
- FF\_2048\_comp
  - ff\_2048.h, [164](#)
- FF\_2048\_copy
  - ff\_2048.h, [164](#)
- FF\_2048\_crt
  - ff\_2048.h, [165](#)
- FF\_2048\_dec
  - ff\_2048.h, [165](#)
- FF\_2048\_dmod
  - ff\_2048.h, [165](#)
- FF\_2048\_fromOctet
  - ff\_2048.h, [166](#)
- FF\_2048\_inc
  - ff\_2048.h, [166](#)
- FF\_2048\_init
  - ff\_2048.h, [166](#)
- FF\_2048\_invmod2m
  - ff\_2048.h, [167](#)
- FF\_2048\_invmodp
  - ff\_2048.h, [167](#)
- FF\_2048\_iszilch
  - ff\_2048.h, [167](#)
- FF\_2048\_lastbits
  - ff\_2048.h, [168](#)
- FF\_2048\_mod
  - ff\_2048.h, [168](#)
- FF\_2048\_mul
  - ff\_2048.h, [168](#)
- FF\_2048\_norm
  - ff\_2048.h, [169](#)
- FF\_2048\_one
  - ff\_2048.h, [169](#)
- FF\_2048\_output
  - ff\_2048.h, [169](#)
- FF\_2048\_parity
  - ff\_2048.h, [170](#)
- FF\_2048\_pow
  - ff\_2048.h, [170](#)
- FF\_2048\_pow2
  - ff\_2048.h, [170](#)
- FF\_2048\_power
  - ff\_2048.h, [171](#)
- FF\_2048\_prime
  - ff\_2048.h, [171](#)
- FF\_2048\_random
  - ff\_2048.h, [172](#)
- FF\_2048\_randomnum
  - ff\_2048.h, [172](#)
- FF\_2048\_rawoutput
  - ff\_2048.h, [172](#)
- FF\_2048\_shl
  - ff\_2048.h, [173](#)
- FF\_2048\_shr
  - ff\_2048.h, [173](#)
- FF\_2048\_skpow
  - ff\_2048.h, [173](#)
- FF\_2048\_skpow2
  - ff\_2048.h, [173](#)

ff\_2048.h, [173](#)  
FF\_2048\_skspow  
ff\_2048.h, [174](#)  
FF\_2048\_sqr  
ff\_2048.h, [174](#)  
FF\_2048\_sub  
ff\_2048.h, [175](#)  
FF\_2048\_toOctet  
ff\_2048.h, [175](#)  
FF\_2048\_zero  
ff\_2048.h, [175](#)  
FF\_4096\_add  
ff\_4096.h, [179](#)  
FF\_4096\_cfactor  
ff\_4096.h, [179](#)  
FF\_4096\_comp  
ff\_4096.h, [179](#)  
FF\_4096\_copy  
ff\_4096.h, [180](#)  
FF\_4096\_crt  
ff\_4096.h, [180](#)  
FF\_4096\_dec  
ff\_4096.h, [180](#)  
FF\_4096\_dmod  
ff\_4096.h, [181](#)  
FF\_4096\_fromOctet  
ff\_4096.h, [181](#)  
FF\_4096\_inc  
ff\_4096.h, [182](#)  
FF\_4096\_init  
ff\_4096.h, [182](#)  
FF\_4096\_invmod2m  
ff\_4096.h, [182](#)  
FF\_4096\_invmodp  
ff\_4096.h, [182](#)  
FF\_4096\_iszilch  
ff\_4096.h, [183](#)  
FF\_4096\_lastbits  
ff\_4096.h, [183](#)  
FF\_4096\_mod  
ff\_4096.h, [183](#)  
FF\_4096\_mul  
ff\_4096.h, [184](#)  
FF\_4096\_norm  
ff\_4096.h, [184](#)  
FF\_4096\_one  
ff\_4096.h, [184](#)  
FF\_4096\_output  
ff\_4096.h, [185](#)  
FF\_4096\_parity  
ff\_4096.h, [185](#)  
FF\_4096\_pow  
ff\_4096.h, [185](#)  
FF\_4096\_pow2  
ff\_4096.h, [186](#)  
FF\_4096\_power  
ff\_4096.h, [186](#)  
FF\_4096\_prime  
ff\_4096.h, [187](#)  
FF\_4096\_random  
ff\_4096.h, [187](#)  
FF\_4096\_randomnum  
ff\_4096.h, [187](#)  
FF\_4096\_rawoutput  
ff\_4096.h, [188](#)  
FF\_4096\_shl  
ff\_4096.h, [188](#)  
FF\_4096\_shr  
ff\_4096.h, [188](#)  
FF\_4096\_skpow  
ff\_4096.h, [188](#)  
FF\_4096\_skpow2  
ff\_4096.h, [189](#)  
FF\_4096\_skspow  
ff\_4096.h, [189](#)  
FF\_4096\_sqr  
ff\_4096.h, [190](#)  
FF\_4096\_sub  
ff\_4096.h, [190](#)  
FF\_4096\_toOctet  
ff\_4096.h, [191](#)  
FF\_4096\_zero  
ff\_4096.h, [191](#)  
FFLEN\_2048  
config\_ff\_2048.h, [124](#)  
FFLEN\_4096  
config\_ff\_4096.h, [125](#)  
FP12\_BLS381, [17](#)  
a, [17](#)  
b, [17](#)  
c, [18](#)  
type, [18](#)  
FP12\_BLS381\_cmove  
fp12\_BLS381.h, [193](#)  
FP12\_BLS381\_compow  
fp12\_BLS381.h, [193](#)  
FP12\_BLS381\_conj  
fp12\_BLS381.h, [194](#)  
FP12\_BLS381\_copy  
fp12\_BLS381.h, [194](#)  
FP12\_BLS381\_equals  
fp12\_BLS381.h, [194](#)  
FP12\_BLS381\_frob  
fp12\_BLS381.h, [195](#)  
FP12\_BLS381\_from\_FP4  
fp12\_BLS381.h, [195](#)  
FP12\_BLS381\_from\_FP4s  
fp12\_BLS381.h, [195](#)  
FP12\_BLS381\_fromOctet  
fp12\_BLS381.h, [196](#)  
FP12\_BLS381\_inv  
fp12\_BLS381.h, [196](#)  
FP12\_BLS381\_ismunity  
fp12\_BLS381.h, [196](#)  
FP12\_BLS381\_iszilch  
fp12\_BLS381.h, [196](#)

FP12\_BLS381\_mul  
  fp12\_BLS381.h, 198

FP12\_BLS381\_norm  
  fp12\_BLS381.h, 198

FP12\_BLS381\_one  
  fp12\_BLS381.h, 198

FP12\_BLS381\_output  
  fp12\_BLS381.h, 198

FP12\_BLS381\_pinpow  
  fp12\_BLS381.h, 199

FP12\_BLS381\_pow  
  fp12\_BLS381.h, 199

FP12\_BLS381\_pow4  
  fp12\_BLS381.h, 199

FP12\_BLS381\_reduce  
  fp12\_BLS381.h, 200

FP12\_BLS381\_smul  
  fp12\_BLS381.h, 200

FP12\_BLS381\_sqr  
  fp12\_BLS381.h, 200

FP12\_BLS381\_ssmul  
  fp12\_BLS381.h, 200

FP12\_BLS381\_toOctet  
  fp12\_BLS381.h, 201

FP12\_BLS381\_trace  
  fp12\_BLS381.h, 201

FP12\_BLS381\_usqr  
  fp12\_BLS381.h, 201

FP12\_BLS381\_zero  
  fp12\_BLS381.h, 202

FP2\_BLS381, 18  
  a, 18  
  b, 18

FP2\_BLS381\_add  
  fp2\_BLS381.h, 204

FP2\_BLS381\_cmove  
  fp2\_BLS381.h, 204

FP2\_BLS381\_conj  
  fp2\_BLS381.h, 205

FP2\_BLS381\_copy  
  fp2\_BLS381.h, 205

FP2\_BLS381\_div2  
  fp2\_BLS381.h, 205

FP2\_BLS381\_div\_ip  
  fp2\_BLS381.h, 206

FP2\_BLS381\_div\_ip2  
  fp2\_BLS381.h, 206

FP2\_BLS381\_equals  
  fp2\_BLS381.h, 206

FP2\_BLS381\_from\_BIGs  
  fp2\_BLS381.h, 207

FP2\_BLS381\_from\_BIG  
  fp2\_BLS381.h, 206

FP2\_BLS381\_from\_FPs  
  fp2\_BLS381.h, 207

FP2\_BLS381\_from\_FP  
  fp2\_BLS381.h, 207

FP2\_BLS381\_imul  
  fp2\_BLS381.h, 208

FP2\_BLS381\_inv  
  fp2\_BLS381.h, 208

FP2\_BLS381\_isunity  
  fp2\_BLS381.h, 208

FP2\_BLS381\_iszilch  
  fp2\_BLS381.h, 209

FP2\_BLS381\_mul  
  fp2\_BLS381.h, 209

FP2\_BLS381\_mul\_ip  
  fp2\_BLS381.h, 209

FP2\_BLS381\_neg  
  fp2\_BLS381.h, 209

FP2\_BLS381\_norm  
  fp2\_BLS381.h, 210

FP2\_BLS381\_one  
  fp2\_BLS381.h, 210

FP2\_BLS381\_output  
  fp2\_BLS381.h, 210

FP2\_BLS381\_pmul  
  fp2\_BLS381.h, 210

FP2\_BLS381\_pow  
  fp2\_BLS381.h, 211

FP2\_BLS381\_rawoutput  
  fp2\_BLS381.h, 211

FP2\_BLS381\_reduce  
  fp2\_BLS381.h, 211

FP2\_BLS381\_sqr  
  fp2\_BLS381.h, 212

FP2\_BLS381\_sqrt  
  fp2\_BLS381.h, 212

FP2\_BLS381\_sub  
  fp2\_BLS381.h, 212

FP2\_BLS381\_times\_i  
  fp2\_BLS381.h, 212

FP2\_BLS381\_zero  
  fp2\_BLS381.h, 213

FP4\_BLS381, 19  
  a, 19  
  b, 19

FP4\_BLS381\_add  
  fp4\_BLS381.h, 215

FP4\_BLS381\_cmove  
  fp4\_BLS381.h, 215

FP4\_BLS381\_conj  
  fp4\_BLS381.h, 217

FP4\_BLS381\_copy  
  fp4\_BLS381.h, 217

FP4\_BLS381\_div2  
  fp4\_BLS381.h, 217

FP4\_BLS381\_div\_2i  
  fp4\_BLS381.h, 218

FP4\_BLS381\_div\_i  
  fp4\_BLS381.h, 218

FP4\_BLS381\_equals  
  fp4\_BLS381.h, 218

FP4\_BLS381\_frob  
  fp4\_BLS381.h, 218

FP4\_BLS381\_from\_FP2  
fp4\_BLS381.h, 219

FP4\_BLS381\_from\_FP2H  
fp4\_BLS381.h, 219

FP4\_BLS381\_from\_FP2s  
fp4\_BLS381.h, 219

FP4\_BLS381\_imul  
fp4\_BLS381.h, 221

FP4\_BLS381\_inv  
fp4\_BLS381.h, 221

FP4\_BLS381\_isreal  
fp4\_BLS381.h, 221

FP4\_BLS381\_isunity  
fp4\_BLS381.h, 222

FP4\_BLS381\_iszilch  
fp4\_BLS381.h, 222

FP4\_BLS381\_mul  
fp4\_BLS381.h, 222

FP4\_BLS381\_nconj  
fp4\_BLS381.h, 223

FP4\_BLS381\_neg  
fp4\_BLS381.h, 223

FP4\_BLS381\_norm  
fp4\_BLS381.h, 223

FP4\_BLS381\_one  
fp4\_BLS381.h, 223

FP4\_BLS381\_output  
fp4\_BLS381.h, 224

FP4\_BLS381\_pmul  
fp4\_BLS381.h, 224

FP4\_BLS381\_pow  
fp4\_BLS381.h, 224

FP4\_BLS381\_qmul  
fp4\_BLS381.h, 224

FP4\_BLS381\_rawoutput  
fp4\_BLS381.h, 225

FP4\_BLS381\_reduce  
fp4\_BLS381.h, 225

FP4\_BLS381\_sqr  
fp4\_BLS381.h, 225

FP4\_BLS381\_sqrt  
fp4\_BLS381.h, 226

FP4\_BLS381\_sub  
fp4\_BLS381.h, 226

FP4\_BLS381\_times\_i  
fp4\_BLS381.h, 226

FP4\_BLS381\_xtr\_A  
fp4\_BLS381.h, 226

FP4\_BLS381\_xtr\_D  
fp4\_BLS381.h, 227

FP4\_BLS381\_xtr\_pow  
fp4\_BLS381.h, 227

FP4\_BLS381\_xtr\_pow2  
fp4\_BLS381.h, 227

FP4\_BLS381\_zero  
fp4\_BLS381.h, 228

FP\_BLS381, 19  
g, 20

XES, 20

FP\_BLS381\_add  
fp\_BLS381.h, 231

FP\_BLS381\_cmove  
fp\_BLS381.h, 231

FP\_BLS381\_copy  
fp\_BLS381.h, 231

FP\_BLS381\_cswap  
fp\_BLS381.h, 232

FP\_BLS381\_div2  
fp\_BLS381.h, 232

FP\_BLS381\_equals  
fp\_BLS381.h, 232

FP\_BLS381\_imul  
fp\_BLS381.h, 233

FP\_BLS381\_inv  
fp\_BLS381.h, 233

FP\_BLS381\_iszilch  
fp\_BLS381.h, 233

FP\_BLS381\_mod  
fp\_BLS381.h, 234

FP\_BLS381\_mul  
fp\_BLS381.h, 234

FP\_BLS381\_neg  
fp\_BLS381.h, 234

FP\_BLS381\_norm  
fp\_BLS381.h, 235

FP\_BLS381\_nres  
fp\_BLS381.h, 235

FP\_BLS381\_one  
fp\_BLS381.h, 235

FP\_BLS381\_output  
fp\_BLS381.h, 235

FP\_BLS381\_pow  
fp\_BLS381.h, 236

FP\_BLS381\_qr  
fp\_BLS381.h, 236

FP\_BLS381\_rawoutput  
fp\_BLS381.h, 236

FP\_BLS381\_rcopy  
fp\_BLS381.h, 236

FP\_BLS381\_redc  
fp\_BLS381.h, 237

FP\_BLS381\_reduce  
fp\_BLS381.h, 237

FP\_BLS381\_sqr  
fp\_BLS381.h, 237

FP\_BLS381\_sqrt  
fp\_BLS381.h, 238

FP\_BLS381\_sub  
fp\_BLS381.h, 238

FP\_BLS381\_zero  
fp\_BLS381.h, 238

FS\_2048  
paillier.h, 255

FS\_4096  
paillier.h, 255

ff\_2048.h, 160

- FF\_2048\_add, 163
- FF\_2048\_cfactor, 163
- FF\_2048\_comp, 164
- FF\_2048\_copy, 164
- FF\_2048\_crt, 165
- FF\_2048\_dec, 165
- FF\_2048\_dmod, 165
- FF\_2048\_fromOctet, 166
- FF\_2048\_inc, 166
- FF\_2048\_init, 166
- FF\_2048\_invmod2m, 167
- FF\_2048\_invmodp, 167
- FF\_2048\_iszilch, 167
- FF\_2048\_lastbits, 168
- FF\_2048\_mod, 168
- FF\_2048\_mul, 168
- FF\_2048\_norm, 169
- FF\_2048\_one, 169
- FF\_2048\_output, 169
- FF\_2048\_parity, 170
- FF\_2048\_pow, 170
- FF\_2048\_pow2, 170
- FF\_2048\_power, 171
- FF\_2048\_prime, 171
- FF\_2048\_random, 172
- FF\_2048\_randomnum, 172
- FF\_2048\_rawoutput, 172
- FF\_2048\_shl, 173
- FF\_2048\_shr, 173
- FF\_2048\_skpow, 173
- FF\_2048\_skpow2, 173
- FF\_2048\_skspow, 174
- FF\_2048\_sqr, 174
- FF\_2048\_sub, 175
- FF\_2048\_toOctet, 175
- FF\_2048\_zero, 175
- HFLen\_2048, 162
- P\_EXCESS\_2048, 162
- P\_FEXCESS\_2048, 163
- P\_MBITS\_2048, 163
- P\_TBITS\_2048, 163
- ff\_4096.h, 176
  - FF\_4096\_add, 179
  - FF\_4096\_cfactor, 179
  - FF\_4096\_comp, 179
  - FF\_4096\_copy, 180
  - FF\_4096\_crt, 180
  - FF\_4096\_dec, 180
  - FF\_4096\_dmod, 181
  - FF\_4096\_fromOctet, 181
  - FF\_4096\_inc, 182
  - FF\_4096\_init, 182
  - FF\_4096\_invmod2m, 182
  - FF\_4096\_invmodp, 182
  - FF\_4096\_iszilch, 183
  - FF\_4096\_lastbits, 183
  - FF\_4096\_mod, 183
  - FF\_4096\_mul, 184
  - FF\_4096\_norm, 184
  - FF\_4096\_one, 184
  - FF\_4096\_output, 185
  - FF\_4096\_parity, 185
  - FF\_4096\_pow, 185
  - FF\_4096\_pow2, 186
  - FF\_4096\_power, 186
  - FF\_4096\_prime, 187
  - FF\_4096\_random, 187
  - FF\_4096\_randomnum, 187
  - FF\_4096\_rawoutput, 188
  - FF\_4096\_shl, 188
  - FF\_4096\_shr, 188
  - FF\_4096\_skpow, 188
  - FF\_4096\_skpow2, 189
  - FF\_4096\_skspow, 189
  - FF\_4096\_sqr, 190
  - FF\_4096\_sub, 190
  - FF\_4096\_toOctet, 191
  - FF\_4096\_zero, 191
  - HFLen\_4096, 178
  - P\_EXCESS\_4096, 178
  - P\_FEXCESS\_4096, 178
  - P\_MBITS\_4096, 178
  - P\_TBITS\_4096, 178
- fkey
  - amcl\_aes, 13
- fp12\_BLS381.h, 191
  - FP12\_BLS381\_cmove, 193
  - FP12\_BLS381\_compow, 193
  - FP12\_BLS381\_conj, 194
  - FP12\_BLS381\_copy, 194
  - FP12\_BLS381\_equals, 194
  - FP12\_BLS381\_frob, 195
  - FP12\_BLS381\_from\_FP4, 195
  - FP12\_BLS381\_from\_FP4s, 195
  - FP12\_BLS381\_fromOctet, 196
  - FP12\_BLS381\_inv, 196
  - FP12\_BLS381\_isunity, 196
  - FP12\_BLS381\_iszilch, 196
  - FP12\_BLS381\_mul, 198
  - FP12\_BLS381\_norm, 198
  - FP12\_BLS381\_one, 198
  - FP12\_BLS381\_output, 198
  - FP12\_BLS381\_pinpow, 199
  - FP12\_BLS381\_pow, 199
  - FP12\_BLS381\_pow4, 199
  - FP12\_BLS381\_reduce, 200
  - FP12\_BLS381\_smul, 200
  - FP12\_BLS381\_sqr, 200
  - FP12\_BLS381\_ssmul, 200
  - FP12\_BLS381\_toOctet, 201
  - FP12\_BLS381\_trace, 201
  - FP12\_BLS381\_usqr, 201
  - FP12\_BLS381\_zero, 202
  - Fra\_BLS381, 202
  - Frb\_BLS381, 202
- fp2\_BLS381.h, 202

- FP2\_BLS381\_add, [204](#)
- FP2\_BLS381\_cmove, [204](#)
- FP2\_BLS381\_conj, [205](#)
- FP2\_BLS381\_copy, [205](#)
- FP2\_BLS381\_div2, [205](#)
- FP2\_BLS381\_div\_ip, [206](#)
- FP2\_BLS381\_div\_ip2, [206](#)
- FP2\_BLS381\_equals, [206](#)
- FP2\_BLS381\_from\_BIGs, [207](#)
- FP2\_BLS381\_from\_BIG, [206](#)
- FP2\_BLS381\_from\_FPs, [207](#)
- FP2\_BLS381\_from\_FP, [207](#)
- FP2\_BLS381\_imul, [208](#)
- FP2\_BLS381\_inv, [208](#)
- FP2\_BLS381\_isunity, [208](#)
- FP2\_BLS381\_iszilch, [209](#)
- FP2\_BLS381\_mul, [209](#)
- FP2\_BLS381\_mul\_ip, [209](#)
- FP2\_BLS381\_neg, [209](#)
- FP2\_BLS381\_norm, [210](#)
- FP2\_BLS381\_one, [210](#)
- FP2\_BLS381\_output, [210](#)
- FP2\_BLS381\_pmul, [210](#)
- FP2\_BLS381\_pow, [211](#)
- FP2\_BLS381\_rawoutput, [211](#)
- FP2\_BLS381\_reduce, [211](#)
- FP2\_BLS381\_sqr, [212](#)
- FP2\_BLS381\_sqrt, [212](#)
- FP2\_BLS381\_sub, [212](#)
- FP2\_BLS381\_times\_i, [212](#)
- FP2\_BLS381\_zero, [213](#)
- fp4\_BLS381.h, [213](#)
- FP4\_BLS381\_add, [215](#)
- FP4\_BLS381\_cmove, [215](#)
- FP4\_BLS381\_conj, [217](#)
- FP4\_BLS381\_copy, [217](#)
- FP4\_BLS381\_div2, [217](#)
- FP4\_BLS381\_div\_2i, [218](#)
- FP4\_BLS381\_div\_i, [218](#)
- FP4\_BLS381\_equals, [218](#)
- FP4\_BLS381\_frob, [218](#)
- FP4\_BLS381\_from\_FP2, [219](#)
- FP4\_BLS381\_from\_FP2H, [219](#)
- FP4\_BLS381\_from\_FP2s, [219](#)
- FP4\_BLS381\_imul, [221](#)
- FP4\_BLS381\_inv, [221](#)
- FP4\_BLS381\_isreal, [221](#)
- FP4\_BLS381\_isunity, [222](#)
- FP4\_BLS381\_iszilch, [222](#)
- FP4\_BLS381\_mul, [222](#)
- FP4\_BLS381\_nconj, [223](#)
- FP4\_BLS381\_neg, [223](#)
- FP4\_BLS381\_norm, [223](#)
- FP4\_BLS381\_one, [223](#)
- FP4\_BLS381\_output, [224](#)
- FP4\_BLS381\_pmul, [224](#)
- FP4\_BLS381\_pow, [224](#)
- FP4\_BLS381\_qmul, [224](#)
- FP4\_BLS381\_rawoutput, [225](#)
- FP4\_BLS381\_reduce, [225](#)
- FP4\_BLS381\_sqr, [225](#)
- FP4\_BLS381\_sqrt, [226](#)
- FP4\_BLS381\_sub, [226](#)
- FP4\_BLS381\_times\_i, [226](#)
- FP4\_BLS381\_xtr\_A, [226](#)
- FP4\_BLS381\_xtr\_D, [227](#)
- FP4\_BLS381\_xtr\_pow, [227](#)
- FP4\_BLS381\_xtr\_pow2, [227](#)
- FP4\_BLS381\_zero, [228](#)
- fp\_BLS381.h, [228](#)
- FEXCESS\_BLS381, [230](#)
- FP\_BLS381\_add, [231](#)
- FP\_BLS381\_cmove, [231](#)
- FP\_BLS381\_copy, [231](#)
- FP\_BLS381\_cswap, [232](#)
- FP\_BLS381\_div2, [232](#)
- FP\_BLS381\_equals, [232](#)
- FP\_BLS381\_imul, [233](#)
- FP\_BLS381\_inv, [233](#)
- FP\_BLS381\_iszilch, [233](#)
- FP\_BLS381\_mod, [234](#)
- FP\_BLS381\_mul, [234](#)
- FP\_BLS381\_neg, [234](#)
- FP\_BLS381\_norm, [235](#)
- FP\_BLS381\_nres, [235](#)
- FP\_BLS381\_one, [235](#)
- FP\_BLS381\_output, [235](#)
- FP\_BLS381\_pow, [236](#)
- FP\_BLS381\_qr, [236](#)
- FP\_BLS381\_rawoutput, [236](#)
- FP\_BLS381\_rcopy, [236](#)
- FP\_BLS381\_redc, [237](#)
- FP\_BLS381\_reduce, [237](#)
- FP\_BLS381\_sqr, [237](#)
- FP\_BLS381\_sqrt, [238](#)
- FP\_BLS381\_sub, [238](#)
- FP\_BLS381\_zero, [238](#)
- MConst\_BLS381, [238](#)
- MODBITS\_BLS381, [230](#)
- Modulus\_BLS381, [238](#)
- OMASK\_BLS381, [230](#)
- R2modp\_BLS381, [239](#)
- TBITS\_BLS381, [230](#)
- TMASK\_BLS381, [231](#)
- Fra\_BLS381
  - ecp2\_BLS381.h, [144](#)
  - ecp\_BLS381.h, [160](#)
  - fp12\_BLS381.h, [202](#)
- Frb\_BLS381
  - ecp2\_BLS381.h, [144](#)
  - ecp\_BLS381.h, [160](#)
  - fp12\_BLS381.h, [202](#)
- g
  - FP\_BLS381, [20](#)
  - PAILLIER\_public\_key, [27](#)
- GET\_TIME



- gcm, [20](#)
    - a, [20](#)
    - lenA, [21](#)
    - lenC, [21](#)
    - stateX, [21](#)
    - status, [21](#)
    - table, [21](#)
    - Y\_0, [21](#)
  - generateOTP
    - utils.c, [277](#)
    - utils.h, [279](#)
  - generateRandom
    - utils.c, [277](#)
    - utils.h, [280](#)
- h
  - hash256, [22](#)
  - hash512, [23](#)
- HASH\_ALL
  - pbcsupport.h, [268](#)
- HASH\_ID
  - pbcsupport.h, [268](#)
- HASH\_TYPE\_RSA\_2048
  - rsa\_2048.h, [271](#)
- HASH
  - ecdh\_support.h, [132](#)
- HBITS\_1024\_58
  - big\_1024\_58.h, [37](#)
- HBITS\_384\_58
  - big\_384\_58.h, [65](#)
- HBITS\_512\_60
  - big\_512\_60.h, [92](#)
- HFLLEN\_2048
  - ff\_2048.h, [162](#)
- HFLLEN\_4096
  - ff\_4096.h, [178](#)
- HFS\_2048
  - paillier.h, [255](#)
- HFS\_4096
  - paillier.h, [256](#)
- HMASK\_1024\_58
  - big\_1024\_58.h, [37](#)
- HMASK\_384\_58
  - big\_384\_58.h, [65](#)
- HMASK\_512\_60
  - big\_512\_60.h, [93](#)
- HMAC
  - ecdh\_support.h, [133](#)
- hash
  - pktype, [28](#)
- hash256, [22](#)
  - h, [22](#)
  - hlen, [22](#)
  - length, [22](#)
  - w, [22](#)
- hash512, [23](#)
  - h, [23](#)
  - hlen, [23](#)
  - length, [23](#)
  - w, [23](#)
- hlen
  - hash256, [22](#)
  - hash512, [23](#)
- invp
  - PAILLIER\_private\_key, [25](#)
- invq
  - PAILLIER\_private\_key, [25](#)
- ira
  - csprng, [15](#)
- KDF2
  - ecdh\_support.h, [133](#)
- KILL\_CSPRNG
  - randapi.h, [270](#)
- len
  - octet, [24](#)
  - sha3, [30](#)
- lenA
  - gcm, [21](#)
- lenC
  - gcm, [21](#)
- length
  - hash256, [22](#)
  - hash512, [23](#)
  - sha3, [30](#)
- lp
  - PAILLIER\_private\_key, [25](#)
- lq
  - PAILLIER\_private\_key, [25](#)
- M\_SIZE\_BLS381
  - mpin\_BLS381.h, [241](#)
- MAX\_RSA\_BYTES
  - rsa\_support.h, [274](#)
- MAXPIN
  - mpin\_BLS381.h, [241](#)
- MConst\_BLS381
  - fp\_BLS381.h, [238](#)
- MESSAGE\_SIZE
  - mpin\_BLS381.h, [241](#)
- MODBITS\_BLS381
  - fp\_BLS381.h, [230](#)
- MODBYTES\_1024\_58
  - config\_big\_1024\_58.h, [122](#)
- MODBYTES\_384\_58
  - config\_big\_384\_58.h, [123](#)
- MODBYTES\_512\_60
  - config\_big\_512\_60.h, [124](#)
- MPIN\_BAD\_PIN
  - mpin\_BLS381.h, [241](#)
- MPIN\_BLS381\_CLIENT\_1
  - mpin\_BLS381.h, [243](#)
- MPIN\_BLS381\_CLIENT\_2
  - mpin\_BLS381.h, [244](#)
- MPIN\_BLS381\_CLIENT\_KEY

- mpin\_BLS381.h, [244](#)
- MPIN\_BLS381\_CLIENT
  - mpin\_BLS381.h, [242](#)
- MPIN\_BLS381\_DECODING
  - mpin\_BLS381.h, [245](#)
- MPIN\_BLS381\_ENCODING
  - mpin\_BLS381.h, [245](#)
- MPIN\_BLS381\_EXTRACT\_FACTOR
  - mpin\_BLS381.h, [245](#)
- MPIN\_BLS381\_EXTRACT\_PIN
  - mpin\_BLS381.h, [246](#)
- MPIN\_BLS381\_GET\_CLIENT\_PERMIT
  - mpin\_BLS381.h, [246](#)
- MPIN\_BLS381\_GET\_CLIENT\_SECRET
  - mpin\_BLS381.h, [247](#)
- MPIN\_BLS381\_GET\_DVS\_KEYPAIR
  - mpin\_BLS381.h, [247](#)
- MPIN\_BLS381\_GET\_G1\_MULTIPLE
  - mpin\_BLS381.h, [247](#)
- MPIN\_BLS381\_GET\_G2\_MULTIPLE
  - mpin\_BLS381.h, [248](#)
- MPIN\_BLS381\_GET\_SERVER\_SECRET
  - mpin\_BLS381.h, [248](#)
- MPIN\_BLS381\_GET\_Y
  - mpin\_BLS381.h, [249](#)
- MPIN\_BLS381\_KANGAROO
  - mpin\_BLS381.h, [249](#)
- MPIN\_BLS381\_PRECOMPUTE
  - mpin\_BLS381.h, [249](#)
- MPIN\_BLS381\_RANDOM\_GENERATE
  - mpin\_BLS381.h, [250](#)
- MPIN\_BLS381\_RECOMBINE\_G1
  - mpin\_BLS381.h, [250](#)
- MPIN\_BLS381\_RECOMBINE\_G2
  - mpin\_BLS381.h, [250](#)
- MPIN\_BLS381\_RESTORE\_FACTOR
  - mpin\_BLS381.h, [251](#)
- MPIN\_BLS381\_SERVER\_1
  - mpin\_BLS381.h, [252](#)
- MPIN\_BLS381\_SERVER\_2
  - mpin\_BLS381.h, [253](#)
- MPIN\_BLS381\_SERVER\_KEY
  - mpin\_BLS381.h, [254](#)
- MPIN\_BLS381\_SERVER
  - mpin\_BLS381.h, [251](#)
- MPIN\_INVALID\_POINT
  - mpin\_BLS381.h, [241](#)
- MPIN\_OK
  - mpin\_BLS381.h, [241](#)
- MPIN\_PAS
  - mpin\_BLS381.h, [241](#)
- max
  - octet, [24](#)
- mhashit
  - pbk\_support.h, [269](#)
- mode
  - amcl\_aes, [13](#)
- Modulus\_BLS381
  - fp\_BLS381.h, [238](#)
- mp
  - PAILLIER\_private\_key, [25](#)
- mpin\_BLS381.h, [239](#)
  - M\_SIZE\_BLS381, [241](#)
  - MAXPIN, [241](#)
  - MESSAGE\_SIZE, [241](#)
  - MPIN\_BAD\_PIN, [241](#)
  - MPIN\_BLS381\_CLIENT\_1, [243](#)
  - MPIN\_BLS381\_CLIENT\_2, [244](#)
  - MPIN\_BLS381\_CLIENT\_KEY, [244](#)
  - MPIN\_BLS381\_CLIENT, [242](#)
  - MPIN\_BLS381\_DECODING, [245](#)
  - MPIN\_BLS381\_ENCODING, [245](#)
  - MPIN\_BLS381\_EXTRACT\_FACTOR, [245](#)
  - MPIN\_BLS381\_EXTRACT\_PIN, [246](#)
  - MPIN\_BLS381\_GET\_CLIENT\_PERMIT, [246](#)
  - MPIN\_BLS381\_GET\_CLIENT\_SECRET, [247](#)
  - MPIN\_BLS381\_GET\_DVS\_KEYPAIR, [247](#)
  - MPIN\_BLS381\_GET\_G1\_MULTIPLE, [247](#)
  - MPIN\_BLS381\_GET\_G2\_MULTIPLE, [248](#)
  - MPIN\_BLS381\_GET\_SERVER\_SECRET, [248](#)
  - MPIN\_BLS381\_GET\_Y, [249](#)
  - MPIN\_BLS381\_KANGAROO, [249](#)
  - MPIN\_BLS381\_PRECOMPUTE, [249](#)
  - MPIN\_BLS381\_RANDOM\_GENERATE, [250](#)
  - MPIN\_BLS381\_RECOMBINE\_G1, [250](#)
  - MPIN\_BLS381\_RECOMBINE\_G2, [250](#)
  - MPIN\_BLS381\_RESTORE\_FACTOR, [251](#)
  - MPIN\_BLS381\_SERVER\_1, [252](#)
  - MPIN\_BLS381\_SERVER\_2, [253](#)
  - MPIN\_BLS381\_SERVER\_KEY, [254](#)
  - MPIN\_BLS381\_SERVER, [251](#)
  - MPIN\_INVALID\_POINT, [241](#)
  - MPIN\_OK, [241](#)
  - MPIN\_PAS, [241](#)
  - PBLEN, [241](#)
  - PFS\_BLS381, [242](#)
  - PGS\_BLS381, [242](#)
- mq
  - PAILLIER\_private\_key, [25](#)
- n
  - PAILLIER\_public\_key, [27](#)
  - rsa\_public\_key\_2048, [30](#)
- n2
  - PAILLIER\_public\_key, [27](#)
- NEXCESS\_1024\_58
  - big\_1024\_58.h, [37](#)
- NEXCESS\_384\_58
  - big\_384\_58.h, [65](#)
- NEXCESS\_512\_60
  - big\_512\_60.h, [93](#)
- NLEN\_1024\_58
  - big\_1024\_58.h, [37](#)
- NLEN\_384\_58
  - big\_384\_58.h, [66](#)
- NLEN\_512\_60
  - big\_512\_60.h, [93](#)

- Nk
  - amcl\_aes, 14
- Nr
  - amcl\_aes, 14
- OAEP\_DECODE
  - rsa\_support.h, 274
- OAEP\_ENCODE
  - rsa\_support.h, 274
- OMASK\_BLS381
  - fp\_BLS381.h, 230
- octet, 24
  - len, 24
  - max, 24
  - val, 24
- p
  - PAILLIER\_private\_key, 26
  - rsa\_private\_key\_2048, 29
- p2
  - PAILLIER\_private\_key, 26
- P\_EXCESS\_2048
  - ff\_2048.h, 162
- P\_EXCESS\_4096
  - ff\_4096.h, 178
- P\_FEXCESS\_2048
  - ff\_2048.h, 163
- P\_FEXCESS\_4096
  - ff\_4096.h, 178
- P\_MBITS\_2048
  - ff\_2048.h, 163
- P\_MBITS\_4096
  - ff\_4096.h, 178
- P\_TBITS\_2048
  - ff\_2048.h, 163
- P\_TBITS\_4096
  - ff\_4096.h, 178
- PAILLIER\_ADD
  - paillier.h, 256
- PAILLIER\_DECRYPT
  - paillier.h, 256
- PAILLIER\_ENCRYPT
  - paillier.h, 257
- PAILLIER\_KEY\_PAIR
  - paillier.h, 257
- PAILLIER\_MULT
  - paillier.h, 258
- PAILLIER\_PK\_fromOctet
  - paillier.h, 258
- PAILLIER\_PK\_toOctet
  - paillier.h, 259
- PAILLIER\_PRIVATE\_KEY\_KILL
  - paillier.h, 259
- PAILLIER\_private\_key, 24
  - invp, 25
  - invq, 25
  - lp, 25
  - lq, 25
  - mp, 25
  - mq, 25
  - p, 26
  - p2, 26
  - q, 26
  - q2, 26
- PAILLIER\_public\_key, 26
  - g, 27
  - n, 27
  - n2, 27
- PAIR\_BLS381\_G1mul
  - pair\_BLS381.h, 263
- PAIR\_BLS381\_G2mul
  - pair\_BLS381.h, 263
- PAIR\_BLS381\_GTmember
  - pair\_BLS381.h, 263
- PAIR\_BLS381\_GTpow
  - pair\_BLS381.h, 264
- PAIR\_BLS381\_another
  - pair\_BLS381.h, 260
- PAIR\_BLS381\_ate
  - pair\_BLS381.h, 261
- PAIR\_BLS381\_double\_ate
  - pair\_BLS381.h, 261
- PAIR\_BLS381\_fexp
  - pair\_BLS381.h, 261
- PAIR\_BLS381\_initmp
  - pair\_BLS381.h, 264
- PAIR\_BLS381\_miller
  - pair\_BLS381.h, 264
- PAIR\_BLS381\_nbits
  - pair\_BLS381.h, 264
- PBKDF2
  - ecdh\_support.h, 133
- PBLEN
  - mpin\_BLS381.h, 241
- PFS\_BLS381
  - mpin\_BLS381.h, 242
- PGS\_BLS381
  - mpin\_BLS381.h, 242
- PIV
  - wcc\_BLS381.h, 282
- PKCS15
  - rsa\_support.h, 275
- PTAG
  - wcc\_BLS381.h, 282
- paillier.h, 254
  - FS\_2048, 255
  - FS\_4096, 255
  - HFS\_2048, 255
  - HFS\_4096, 256
  - PAILLIER\_ADD, 256
  - PAILLIER\_DECRYPT, 256
  - PAILLIER\_ENCRYPT, 257
  - PAILLIER\_KEY\_PAIR, 257
  - PAILLIER\_MULT, 258
  - PAILLIER\_PK\_fromOctet, 258
  - PAILLIER\_PK\_toOctet, 259
  - PAILLIER\_PRIVATE\_KEY\_KILL, 259

- pair\_BLS381.h, 259
  - CURVE\_BB\_BLS381, 265
  - CURVE\_Bnx\_BLS381, 265
  - CURVE\_Cru\_BLS381, 265
  - CURVE\_SB\_BLS381, 265
  - CURVE\_W\_BLS381, 265
  - CURVE\_WB\_BLS381, 265
  - PAIR\_BLS381\_G1mul, 263
  - PAIR\_BLS381\_G2mul, 263
  - PAIR\_BLS381\_GTmember, 263
  - PAIR\_BLS381\_GTpow, 264
  - PAIR\_BLS381\_another, 260
  - PAIR\_BLS381\_ate, 261
  - PAIR\_BLS381\_double\_ate, 261
  - PAIR\_BLS381\_fexp, 261
  - PAIR\_BLS381\_initmp, 264
  - PAIR\_BLS381\_miller, 264
  - PAIR\_BLS381\_nbits, 264
- pbcsupport.h, 266
  - AES\_GCM\_DECRYPT, 267
  - AES\_GCM\_ENCRYPT, 267
  - GET\_TIME, 268
  - HASH\_ALL, 268
  - HASH\_ID, 268
  - mhashit, 269
  - TIME\_SLOT\_MINUTES, 266
  - today, 269
- pktype, 27
  - curve, 27
  - hash, 28
  - type, 28
- pool
  - csprng, 15
- pool\_ptr
  - csprng, 15
- q
  - PAILLIER\_private\_key, 26
  - rsa\_private\_key\_2048, 29
- q2
  - PAILLIER\_private\_key, 26
- R2modp\_BLS381
  - fp\_BLS381.h, 239
- RFS\_2048
  - rsa\_2048.h, 271
- RSA\_2048\_DECRYPT
  - rsa\_2048.h, 271
- RSA\_2048\_ENCRYPT
  - rsa\_2048.h, 272
- RSA\_2048\_KEY\_PAIR
  - rsa\_2048.h, 272
- RSA\_2048\_PRIVATE\_KEY\_KILL
  - rsa\_2048.h, 273
- RSA\_2048\_fromOctet
  - rsa\_2048.h, 272
- randapi.h, 269
  - CREATE\_CSPRNG, 270
  - KILL\_CSPRNG, 270
- rate
  - sha3, 30
- rkey
  - amcl\_aes, 14
- rndptr
  - csprng, 15
- rsa\_2048.h, 270
  - HASH\_TYPE\_RSA\_2048, 271
  - RFS\_2048, 271
  - RSA\_2048\_DECRYPT, 271
  - RSA\_2048\_ENCRYPT, 272
  - RSA\_2048\_KEY\_PAIR, 272
  - RSA\_2048\_PRIVATE\_KEY\_KILL, 273
  - RSA\_2048\_fromOctet, 272
- rsa\_private\_key\_2048, 28
  - c, 28
  - dp, 28
  - dq, 29
  - p, 29
  - q, 29
- rsa\_public\_key\_2048, 29
  - e, 29
  - n, 30
- rsa\_support.h, 273
  - MAX\_RSA\_BYTES, 274
  - OAEP\_DECODE, 274
  - OAEP\_ENCODE, 274
  - PKCS15, 275
- S
  - sha3, 30
- sha3, 30
  - len, 30
  - length, 30
  - rate, 30
  - S, 30
- sign16
  - arch.h, 32
- sign32
  - arch.h, 32
- sign64
  - arch.h, 32
- sign8
  - arch.h, 32
- stateX
  - gcm, 21
- status
  - gcm, 21
- TBITS\_BLS381
  - fp\_BLS381.h, 230
- TIME\_SLOT\_MINUTES
  - pbcsupport.h, 266
  - wcc\_BLS381.h, 282
- TMASK\_BLS381
  - fp\_BLS381.h, 231
- table
  - gcm, 21
- today

