

AMCL

Generated by Doxygen 1.8.13



# Contents

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

<b>7 Data Structure Documentation</b>	<b>15</b>
7.1 amcl_aes Struct Reference	15
7.1.1 Field Documentation	15
7.1.1.1 f	15
7.1.1.2 fkey	15
7.1.1.3 mode	16
7.1.1.4 Nk	16
7.1.1.5 Nr	16
7.1.1.6 rkey	16
7.2 csprng Struct Reference	16
7.2.1 Field Documentation	16
7.2.1.1 borrow	17
7.2.1.2 ira	17
7.2.1.3 pool	17
7.2.1.4 pool_ptr	17
7.2.1.5 rndptr	17
7.3 ECP2_BLS381 Struct Reference	17
7.3.1 Field Documentation	18
7.3.1.1 x	18
7.3.1.2 y	18
7.3.1.3 z	18
7.4 ECP_BLS381 Struct Reference	18
7.4.1 Field Documentation	18
7.4.1.1 x	19
7.4.1.2 y	19
7.4.1.3 z	19
7.5 ECP_ED25519 Struct Reference	19
7.5.1 Field Documentation	19
7.5.1.1 x	19
7.5.1.2 y	20

7.5.1.3	z	20
7.6	ECP_GOLDILOCKS Struct Reference	20
7.6.1	Field Documentation	20
7.6.1.1	x	20
7.6.1.2	y	20
7.6.1.3	z	21
7.7	ECP_NIST256 Struct Reference	21
7.7.1	Field Documentation	21
7.7.1.1	x	21
7.7.1.2	y	21
7.7.1.3	z	21
7.8	FP12_BLS381 Struct Reference	22
7.8.1	Field Documentation	22
7.8.1.1	a	22
7.8.1.2	b	22
7.8.1.3	c	22
7.8.1.4	type	22
7.9	FP2_BLS381 Struct Reference	23
7.9.1	Field Documentation	23
7.9.1.1	a	23
7.9.1.2	b	23
7.10	FP4_BLS381 Struct Reference	23
7.10.1	Field Documentation	23
7.10.1.1	a	24
7.10.1.2	b	24
7.11	FP_25519 Struct Reference	24
7.11.1	Field Documentation	24
7.11.1.1	g	24
7.11.1.2	XES	24
7.12	FP_BLS381 Struct Reference	25

7.12.1	Field Documentation	25
7.12.1.1	g	25
7.12.1.2	XES	25
7.13	FP_GOLDILOCKS Struct Reference	25
7.13.1	Field Documentation	25
7.13.1.1	g	26
7.13.1.2	XES	26
7.14	FP_NIST256 Struct Reference	26
7.14.1	Field Documentation	26
7.14.1.1	g	26
7.14.1.2	XES	26
7.15	gcm Struct Reference	27
7.15.1	Field Documentation	27
7.15.1.1	a	27
7.15.1.2	lenA	27
7.15.1.3	lenC	27
7.15.1.4	stateX	27
7.15.1.5	status	28
7.15.1.6	table	28
7.15.1.7	Y_0	28
7.16	hash256 Struct Reference	28
7.16.1	Field Documentation	28
7.16.1.1	h	28
7.16.1.2	hlen	29
7.16.1.3	length	29
7.16.1.4	w	29
7.17	hash512 Struct Reference	29
7.17.1	Field Documentation	29
7.17.1.1	h	29
7.17.1.2	hlen	30

7.17.1.3	length	30
7.17.1.4	w	30
7.18	octet Struct Reference	30
7.18.1	Field Documentation	30
7.18.1.1	len	30
7.18.1.2	max	31
7.18.1.3	val	31
7.19	pktype Struct Reference	31
7.19.1	Field Documentation	31
7.19.1.1	curve	31
7.19.1.2	hash	31
7.19.1.3	type	32
7.20	rsa_private_key_2048 Struct Reference	32
7.20.1	Field Documentation	32
7.20.1.1	c	32
7.20.1.2	dp	32
7.20.1.3	dq	32
7.20.1.4	p	33
7.20.1.5	q	33
7.21	rsa_private_key_3072 Struct Reference	33
7.21.1	Field Documentation	33
7.21.1.1	c	33
7.21.1.2	dp	33
7.21.1.3	dq	34
7.21.1.4	p	34
7.21.1.5	q	34
7.22	rsa_public_key_2048 Struct Reference	34
7.22.1	Field Documentation	34
7.22.1.1	e	34
7.22.1.2	n	35
7.23	rsa_public_key_3072 Struct Reference	35
7.23.1	Field Documentation	35
7.23.1.1	e	35
7.23.1.2	n	35
7.24	sha3 Struct Reference	35
7.24.1	Field Documentation	36
7.24.1.1	len	36
7.24.1.2	length	36
7.24.1.3	rate	36
7.24.1.4	S	36

<b>8 File Documentation</b>	<b>37</b>
8.1 arch.h File Reference	37
8.1.1 Detailed Description	37
8.1.2 Macro Definition Documentation	37
8.1.2.1 byte	38
8.1.2.2 CHUNK	38
8.1.2.3 chunk	38
8.1.2.4 sign16	38
8.1.2.5 sign32	38
8.1.2.6 sign64	38
8.1.2.7 sign8	38
8.1.2.8 uchar	39
8.1.2.9 unsign32	39
8.1.2.10 unsign64	39
8.2 big_1024_58.h File Reference	39
8.2.1 Detailed Description	42
8.2.2 Macro Definition Documentation	43
8.2.2.1 BIGBITS_1024_58	43
8.2.2.2 BMASK_1024_58	43
8.2.2.3 DNLEN_1024_58	43
8.2.2.4 HBITS_1024_58	43
8.2.2.5 HMASK_1024_58	43
8.2.2.6 NEXCESS_1024_58	43
8.2.2.7 NLEN_1024_58	43
8.2.3 Typedef Documentation	44
8.2.3.1 BIG_1024_58	44
8.2.3.2 DBIG_1024_58	44
8.2.4 Function Documentation	44
8.2.4.1 BIG_1024_58_add()	44
8.2.4.2 BIG_1024_58_bit()	44



8.2.4.3	BIG_1024_58_cmove()	45
8.2.4.4	BIG_1024_58_comp()	45
8.2.4.5	BIG_1024_58_copy()	45
8.2.4.6	BIG_1024_58_cswap()	46
8.2.4.7	BIG_1024_58_dadd()	46
8.2.4.8	BIG_1024_58_dcmove()	46
8.2.4.9	BIG_1024_58_dcomp()	47
8.2.4.10	BIG_1024_58_dcopy()	47
8.2.4.11	BIG_1024_58_ddiv()	47
8.2.4.12	BIG_1024_58_dec()	48
8.2.4.13	BIG_1024_58_dfromBytesLen()	48
8.2.4.14	BIG_1024_58_diszilch()	48
8.2.4.15	BIG_1024_58_div3()	49
8.2.4.16	BIG_1024_58_dmod()	50
8.2.4.17	BIG_1024_58_dmod2m()	50
8.2.4.18	BIG_1024_58_dnbits()	50
8.2.4.19	BIG_1024_58_dnorm()	51
8.2.4.20	BIG_1024_58_doutput()	51
8.2.4.21	BIG_1024_58_drawoutput()	51
8.2.4.22	BIG_1024_58_dscopy()	52
8.2.4.23	BIG_1024_58_dshl()	52
8.2.4.24	BIG_1024_58_dshr()	52
8.2.4.25	BIG_1024_58_dsub()	52
8.2.4.26	BIG_1024_58_dscopy()	53
8.2.4.27	BIG_1024_58_dzero()	53
8.2.4.28	BIG_1024_58_fromBytes()	53
8.2.4.29	BIG_1024_58_fromBytesLen()	54
8.2.4.30	BIG_1024_58_fshl()	54
8.2.4.31	BIG_1024_58_fshr()	54
8.2.4.32	BIG_1024_58_imul()	55

8.2.4.33	BIG_1024_58_inc()	55
8.2.4.34	BIG_1024_58_invmod2m()	55
8.2.4.35	BIG_1024_58_invmodp()	55
8.2.4.36	BIG_1024_58_isunity()	56
8.2.4.37	BIG_1024_58_iszilch()	56
8.2.4.38	BIG_1024_58_jacobi()	56
8.2.4.39	BIG_1024_58_lastbits()	57
8.2.4.40	BIG_1024_58_mod()	57
8.2.4.41	BIG_1024_58_mod2m()	57
8.2.4.42	BIG_1024_58_moddiv()	58
8.2.4.43	BIG_1024_58_modmul()	58
8.2.4.44	BIG_1024_58_modneg()	59
8.2.4.45	BIG_1024_58_modsqr()	59
8.2.4.46	BIG_1024_58_monty()	59
8.2.4.47	BIG_1024_58_mul()	60
8.2.4.48	BIG_1024_58_nbits()	60
8.2.4.49	BIG_1024_58_norm()	60
8.2.4.50	BIG_1024_58_one()	61
8.2.4.51	BIG_1024_58_or()	61
8.2.4.52	BIG_1024_58_output()	61
8.2.4.53	BIG_1024_58_parity()	61
8.2.4.54	BIG_1024_58_pmul()	62
8.2.4.55	BIG_1024_58_pxmud()	62
8.2.4.56	BIG_1024_58_random()	62
8.2.4.57	BIG_1024_58_randomnum()	63
8.2.4.58	BIG_1024_58_rawoutput()	63
8.2.4.59	BIG_1024_58_rcopy()	63
8.2.4.60	BIG_1024_58_sdcopy()	64
8.2.4.61	BIG_1024_58_sdiv()	64
8.2.4.62	BIG_1024_58_sducopy()	64

8.2.4.63	BIG_1024_58_shl()	64
8.2.4.64	BIG_1024_58_shr()	65
8.2.4.65	BIG_1024_58_smul()	65
8.2.4.66	BIG_1024_58_split()	65
8.2.4.67	BIG_1024_58_sqr()	66
8.2.4.68	BIG_1024_58_ssn()	66
8.2.4.69	BIG_1024_58_sub()	66
8.2.4.70	BIG_1024_58_toBytes()	67
8.2.4.71	BIG_1024_58_zero()	67
8.3	big_256_56.h File Reference	67
8.3.1	Detailed Description	71
8.3.2	Macro Definition Documentation	71
8.3.2.1	BIGBITS_256_56	71
8.3.2.2	BMASK_256_56	71
8.3.2.3	DNLEN_256_56	71
8.3.2.4	HBITS_256_56	71
8.3.2.5	HMASK_256_56	71
8.3.2.6	NEXCESS_256_56	72
8.3.2.7	NLEN_256_56	72
8.3.3	Typedef Documentation	72
8.3.3.1	BIG_256_56	72
8.3.3.2	DBIG_256_56	72
8.3.4	Function Documentation	72
8.3.4.1	BIG_256_56_add()	72
8.3.4.2	BIG_256_56_bit()	73
8.3.4.3	BIG_256_56_cmove()	73
8.3.4.4	BIG_256_56_comp()	73
8.3.4.5	BIG_256_56_copy()	74
8.3.4.6	BIG_256_56_cswap()	74
8.3.4.7	BIG_256_56_dadd()	74

8.3.4.8	BIG_256_56_dcmove()	75
8.3.4.9	BIG_256_56_dcomp()	75
8.3.4.10	BIG_256_56_dcopy()	75
8.3.4.11	BIG_256_56_ddiv()	76
8.3.4.12	BIG_256_56_dec()	76
8.3.4.13	BIG_256_56_dfromBytesLen()	76
8.3.4.14	BIG_256_56_diszilch()	77
8.3.4.15	BIG_256_56_div3()	77
8.3.4.16	BIG_256_56_dmod()	77
8.3.4.17	BIG_256_56_dmod2m()	78
8.3.4.18	BIG_256_56_dnbits()	78
8.3.4.19	BIG_256_56_dnorm()	78
8.3.4.20	BIG_256_56_doutput()	78
8.3.4.21	BIG_256_56_drawoutput()	79
8.3.4.22	BIG_256_56_dscopy()	79
8.3.4.23	BIG_256_56_dshl()	79
8.3.4.24	BIG_256_56_dshr()	79
8.3.4.25	BIG_256_56_dsub()	80
8.3.4.26	BIG_256_56_dsucopy()	80
8.3.4.27	BIG_256_56_dzero()	80
8.3.4.28	BIG_256_56_fromBytes()	81
8.3.4.29	BIG_256_56_fromBytesLen()	81
8.3.4.30	BIG_256_56_fshl()	81
8.3.4.31	BIG_256_56_fshr()	82
8.3.4.32	BIG_256_56_imul()	82
8.3.4.33	BIG_256_56_inc()	82
8.3.4.34	BIG_256_56_invmod2m()	83
8.3.4.35	BIG_256_56_invmodp()	83
8.3.4.36	BIG_256_56_isunity()	83
8.3.4.37	BIG_256_56_iszilch()	83

8.3.4.38	BIG_256_56_jacobi()	84
8.3.4.39	BIG_256_56_lastbits()	84
8.3.4.40	BIG_256_56_mod()	84
8.3.4.41	BIG_256_56_mod2m()	85
8.3.4.42	BIG_256_56_moddiv()	85
8.3.4.43	BIG_256_56_modmul()	85
8.3.4.44	BIG_256_56_modneg()	86
8.3.4.45	BIG_256_56_modsqr()	86
8.3.4.46	BIG_256_56_monty()	87
8.3.4.47	BIG_256_56_mul()	87
8.3.4.48	BIG_256_56_nbits()	87
8.3.4.49	BIG_256_56_norm()	88
8.3.4.50	BIG_256_56_one()	88
8.3.4.51	BIG_256_56_or()	88
8.3.4.52	BIG_256_56_output()	88
8.3.4.53	BIG_256_56_parity()	89
8.3.4.54	BIG_256_56_pmul()	89
8.3.4.55	BIG_256_56_pxmuls()	89
8.3.4.56	BIG_256_56_random()	90
8.3.4.57	BIG_256_56_randomnum()	90
8.3.4.58	BIG_256_56_rawoutput()	90
8.3.4.59	BIG_256_56_rcopy()	90
8.3.4.60	BIG_256_56_sdcopy()	91
8.3.4.61	BIG_256_56_sdiv()	91
8.3.4.62	BIG_256_56_sducopys()	91
8.3.4.63	BIG_256_56_shl()	92
8.3.4.64	BIG_256_56_shr()	92
8.3.4.65	BIG_256_56_smuls()	92
8.3.4.66	BIG_256_56_split()	92
8.3.4.67	BIG_256_56_sqr()	93

8.3.4.68	BIG_256_56_ssn()	93
8.3.4.69	BIG_256_56_sub()	94
8.3.4.70	BIG_256_56_toBytes()	94
8.3.4.71	BIG_256_56_zero()	94
8.4	big_384_56.h File Reference	94
8.4.1	Detailed Description	98
8.4.2	Macro Definition Documentation	98
8.4.2.1	BIGBITS_384_56	98
8.4.2.2	BMASK_384_56	98
8.4.2.3	DNLEN_384_56	98
8.4.2.4	HBITS_384_56	99
8.4.2.5	HMASK_384_56	99
8.4.2.6	NEXCESS_384_56	99
8.4.2.7	NLEN_384_56	99
8.4.3	Typedef Documentation	99
8.4.3.1	BIG_384_56	99
8.4.3.2	DBIG_384_56	99
8.4.4	Function Documentation	99
8.4.4.1	BIG_384_56_add()	99
8.4.4.2	BIG_384_56_bit()	100
8.4.4.3	BIG_384_56_cmove()	100
8.4.4.4	BIG_384_56_comp()	100
8.4.4.5	BIG_384_56_copy()	101
8.4.4.6	BIG_384_56_cswap()	101
8.4.4.7	BIG_384_56_dadd()	101
8.4.4.8	BIG_384_56_dcmove()	102
8.4.4.9	BIG_384_56_dcomp()	102
8.4.4.10	BIG_384_56_dcopy()	102
8.4.4.11	BIG_384_56_ddiv()	103
8.4.4.12	BIG_384_56_dec()	103

8.4.4.13	BIG_384_56_dfromBytesLen()	103
8.4.4.14	BIG_384_56_diszilch()	104
8.4.4.15	BIG_384_56_div3()	104
8.4.4.16	BIG_384_56_dmod()	104
8.4.4.17	BIG_384_56_dmod2m()	105
8.4.4.18	BIG_384_56_dnbits()	105
8.4.4.19	BIG_384_56_dnorm()	105
8.4.4.20	BIG_384_56_doutput()	105
8.4.4.21	BIG_384_56_drawoutput()	106
8.4.4.22	BIG_384_56_dscopy()	106
8.4.4.23	BIG_384_56_dshl()	106
8.4.4.24	BIG_384_56_dshr()	107
8.4.4.25	BIG_384_56_dsub()	107
8.4.4.26	BIG_384_56_dsucopy()	107
8.4.4.27	BIG_384_56_dzero()	107
8.4.4.28	BIG_384_56_fromBytes()	108
8.4.4.29	BIG_384_56_fromBytesLen()	108
8.4.4.30	BIG_384_56_fshl()	108
8.4.4.31	BIG_384_56_fshr()	109
8.4.4.32	BIG_384_56_imul()	109
8.4.4.33	BIG_384_56_inc()	109
8.4.4.34	BIG_384_56_invmod2m()	110
8.4.4.35	BIG_384_56_invmodp()	110
8.4.4.36	BIG_384_56_isunity()	110
8.4.4.37	BIG_384_56_iszilch()	111
8.4.4.38	BIG_384_56_jacobi()	112
8.4.4.39	BIG_384_56_lastbits()	112
8.4.4.40	BIG_384_56_mod()	112
8.4.4.41	BIG_384_56_mod2m()	113
8.4.4.42	BIG_384_56_moddiv()	113

8.4.4.43	BIG_384_56_modmul()	113
8.4.4.44	BIG_384_56_modneg()	114
8.4.4.45	BIG_384_56_modsqr()	114
8.4.4.46	BIG_384_56_monty()	115
8.4.4.47	BIG_384_56_mul()	115
8.4.4.48	BIG_384_56_nbits()	115
8.4.4.49	BIG_384_56_norm()	116
8.4.4.50	BIG_384_56_one()	116
8.4.4.51	BIG_384_56_or()	116
8.4.4.52	BIG_384_56_output()	116
8.4.4.53	BIG_384_56_parity()	117
8.4.4.54	BIG_384_56_pmul()	117
8.4.4.55	BIG_384_56_pxmud()	117
8.4.4.56	BIG_384_56_random()	118
8.4.4.57	BIG_384_56_randomnum()	118
8.4.4.58	BIG_384_56_rawoutput()	118
8.4.4.59	BIG_384_56_rcopy()	118
8.4.4.60	BIG_384_56_sdcopy()	119
8.4.4.61	BIG_384_56_sdiv()	119
8.4.4.62	BIG_384_56_sducopu()	119
8.4.4.63	BIG_384_56_shl()	120
8.4.4.64	BIG_384_56_shr()	120
8.4.4.65	BIG_384_56_smul()	120
8.4.4.66	BIG_384_56_split()	120
8.4.4.67	BIG_384_56_sqr()	121
8.4.4.68	BIG_384_56_ssn()	121
8.4.4.69	BIG_384_56_sub()	122
8.4.4.70	BIG_384_56_toBytes()	122
8.4.4.71	BIG_384_56_zero()	122
8.5	big_384_58.h File Reference	122



8.5.1	Detailed Description	126
8.5.2	Macro Definition Documentation	126
8.5.2.1	BIGBITS_384_58	126
8.5.2.2	BMASK_384_58	126
8.5.2.3	DNLEN_384_58	126
8.5.2.4	HBITS_384_58	127
8.5.2.5	HMASK_384_58	127
8.5.2.6	NEXCESS_384_58	127
8.5.2.7	NLEN_384_58	127
8.5.3	Typedef Documentation	127
8.5.3.1	BIG_384_58	127
8.5.3.2	DBIG_384_58	127
8.5.4	Function Documentation	127
8.5.4.1	BIG_384_58_add()	127
8.5.4.2	BIG_384_58_bit()	128
8.5.4.3	BIG_384_58_cmove()	128
8.5.4.4	BIG_384_58_comp()	128
8.5.4.5	BIG_384_58_copy()	129
8.5.4.6	BIG_384_58_cswap()	129
8.5.4.7	BIG_384_58_dadd()	129
8.5.4.8	BIG_384_58_dcmove()	130
8.5.4.9	BIG_384_58_dcomp()	130
8.5.4.10	BIG_384_58_dcopy()	130
8.5.4.11	BIG_384_58_ddiv()	131
8.5.4.12	BIG_384_58_dec()	131
8.5.4.13	BIG_384_58_dfromBytesLen()	131
8.5.4.14	BIG_384_58_diszilch()	132
8.5.4.15	BIG_384_58_div3()	132
8.5.4.16	BIG_384_58_dmod()	132
8.5.4.17	BIG_384_58_dmod2m()	133

8.5.4.18	BIG_384_58_dnbits()	133
8.5.4.19	BIG_384_58_dnorm()	133
8.5.4.20	BIG_384_58_doutput()	133
8.5.4.21	BIG_384_58_drawoutput()	134
8.5.4.22	BIG_384_58_dscopy()	134
8.5.4.23	BIG_384_58_dshl()	134
8.5.4.24	BIG_384_58_dshr()	135
8.5.4.25	BIG_384_58_dsub()	135
8.5.4.26	BIG_384_58_dsucopy()	135
8.5.4.27	BIG_384_58_dzero()	135
8.5.4.28	BIG_384_58_fromBytes()	136
8.5.4.29	BIG_384_58_fromBytesLen()	136
8.5.4.30	BIG_384_58_fshl()	136
8.5.4.31	BIG_384_58_fshr()	137
8.5.4.32	BIG_384_58_imul()	137
8.5.4.33	BIG_384_58_inc()	137
8.5.4.34	BIG_384_58_invmod2m()	138
8.5.4.35	BIG_384_58_invmodp()	138
8.5.4.36	BIG_384_58_isunity()	138
8.5.4.37	BIG_384_58_iszilch()	139
8.5.4.38	BIG_384_58_jacobi()	140
8.5.4.39	BIG_384_58_lastbits()	140
8.5.4.40	BIG_384_58_mod()	140
8.5.4.41	BIG_384_58_mod2m()	141
8.5.4.42	BIG_384_58_moddiv()	141
8.5.4.43	BIG_384_58_modmul()	141
8.5.4.44	BIG_384_58_modneg()	142
8.5.4.45	BIG_384_58_modsqr()	142
8.5.4.46	BIG_384_58_monty()	143
8.5.4.47	BIG_384_58_mul()	143

8.5.4.48	BIG_384_58_nbits()	143
8.5.4.49	BIG_384_58_norm()	144
8.5.4.50	BIG_384_58_one()	144
8.5.4.51	BIG_384_58_or()	144
8.5.4.52	BIG_384_58_output()	144
8.5.4.53	BIG_384_58_parity()	145
8.5.4.54	BIG_384_58_pmul()	145
8.5.4.55	BIG_384_58_pxmulo()	145
8.5.4.56	BIG_384_58_random()	146
8.5.4.57	BIG_384_58_randomnum()	146
8.5.4.58	BIG_384_58_rawoutput()	146
8.5.4.59	BIG_384_58_rcopy()	146
8.5.4.60	BIG_384_58_sdcopy()	147
8.5.4.61	BIG_384_58_sdiv()	147
8.5.4.62	BIG_384_58_sducopy()	147
8.5.4.63	BIG_384_58_shl()	148
8.5.4.64	BIG_384_58_shr()	148
8.5.4.65	BIG_384_58_smulo()	148
8.5.4.66	BIG_384_58_split()	148
8.5.4.67	BIG_384_58_sqr()	149
8.5.4.68	BIG_384_58_ssn()	149
8.5.4.69	BIG_384_58_sub()	150
8.5.4.70	BIG_384_58_toBytes()	150
8.5.4.71	BIG_384_58_zero()	150
8.6	big_448_58.h File Reference	150
8.6.1	Detailed Description	154
8.6.2	Macro Definition Documentation	154
8.6.2.1	BIGBITS_448_58	154
8.6.2.2	BMASK_448_58	154
8.6.2.3	DNLEN_448_58	154

8.6.2.4	HBITS_448_58 . . . . .	155
8.6.2.5	HMASK_448_58 . . . . .	155
8.6.2.6	NEXCESS_448_58 . . . . .	155
8.6.2.7	NLEN_448_58 . . . . .	155
8.6.3	Typedef Documentation . . . . .	155
8.6.3.1	BIG_448_58 . . . . .	155
8.6.3.2	DBIG_448_58 . . . . .	155
8.6.4	Function Documentation . . . . .	155
8.6.4.1	BIG_448_58_add() . . . . .	155
8.6.4.2	BIG_448_58_bit() . . . . .	156
8.6.4.3	BIG_448_58_cmove() . . . . .	156
8.6.4.4	BIG_448_58_comp() . . . . .	156
8.6.4.5	BIG_448_58_copy() . . . . .	157
8.6.4.6	BIG_448_58_cswap() . . . . .	157
8.6.4.7	BIG_448_58_dadd() . . . . .	157
8.6.4.8	BIG_448_58_dcmove() . . . . .	158
8.6.4.9	BIG_448_58_dcomp() . . . . .	158
8.6.4.10	BIG_448_58_dcopy() . . . . .	158
8.6.4.11	BIG_448_58_ddiv() . . . . .	159
8.6.4.12	BIG_448_58_dec() . . . . .	159
8.6.4.13	BIG_448_58_dfromBytesLen() . . . . .	159
8.6.4.14	BIG_448_58_diszilch() . . . . .	160
8.6.4.15	BIG_448_58_div3() . . . . .	160
8.6.4.16	BIG_448_58_dmod() . . . . .	160
8.6.4.17	BIG_448_58_dmod2m() . . . . .	161
8.6.4.18	BIG_448_58_dnbits() . . . . .	161
8.6.4.19	BIG_448_58_dnorm() . . . . .	161
8.6.4.20	BIG_448_58_doutput() . . . . .	161
8.6.4.21	BIG_448_58_drawoutput() . . . . .	162
8.6.4.22	BIG_448_58_dscopy() . . . . .	162

8.6.4.23	BIG_448_58_dshl()	162
8.6.4.24	BIG_448_58_dshr()	163
8.6.4.25	BIG_448_58_dsub()	163
8.6.4.26	BIG_448_58_dscopy()	163
8.6.4.27	BIG_448_58_dzero()	163
8.6.4.28	BIG_448_58_fromBytes()	164
8.6.4.29	BIG_448_58_fromBytesLen()	164
8.6.4.30	BIG_448_58_fshl()	164
8.6.4.31	BIG_448_58_fshr()	165
8.6.4.32	BIG_448_58_imul()	165
8.6.4.33	BIG_448_58_inc()	165
8.6.4.34	BIG_448_58_invmod2m()	166
8.6.4.35	BIG_448_58_invmodp()	166
8.6.4.36	BIG_448_58_isunity()	166
8.6.4.37	BIG_448_58_iszilch()	167
8.6.4.38	BIG_448_58_jacobi()	168
8.6.4.39	BIG_448_58_lastbits()	168
8.6.4.40	BIG_448_58_mod()	168
8.6.4.41	BIG_448_58_mod2m()	169
8.6.4.42	BIG_448_58_moddiv()	169
8.6.4.43	BIG_448_58_modmul()	169
8.6.4.44	BIG_448_58_modneg()	170
8.6.4.45	BIG_448_58_modsqr()	170
8.6.4.46	BIG_448_58_monty()	171
8.6.4.47	BIG_448_58_mul()	171
8.6.4.48	BIG_448_58_nbits()	171
8.6.4.49	BIG_448_58_norm()	172
8.6.4.50	BIG_448_58_one()	172
8.6.4.51	BIG_448_58_or()	172
8.6.4.52	BIG_448_58_output()	172

8.6.4.53	BIG_448_58_parity()	173
8.6.4.54	BIG_448_58_pmul()	173
8.6.4.55	BIG_448_58_pxmulo()	173
8.6.4.56	BIG_448_58_random()	174
8.6.4.57	BIG_448_58_randomnum()	174
8.6.4.58	BIG_448_58_rawoutput()	174
8.6.4.59	BIG_448_58_rcopy()	174
8.6.4.60	BIG_448_58_sdcopy()	175
8.6.4.61	BIG_448_58_sdiv()	175
8.6.4.62	BIG_448_58_sducopy()	175
8.6.4.63	BIG_448_58_shl()	176
8.6.4.64	BIG_448_58_shr()	176
8.6.4.65	BIG_448_58_smulo()	176
8.6.4.66	BIG_448_58_split()	176
8.6.4.67	BIG_448_58_sqr()	177
8.6.4.68	BIG_448_58_ssn()	177
8.6.4.69	BIG_448_58_sub()	178
8.6.4.70	BIG_448_58_toBytes()	178
8.6.4.71	BIG_448_58_zero()	178
8.7	bls_BLS381.h File Reference	178
8.7.1	Detailed Description	179
8.7.2	Macro Definition Documentation	179
8.7.2.1	BFS_BLS381	179
8.7.2.2	BGS_BLS381	180
8.7.2.3	BLS_FAIL	180
8.7.2.4	BLS_INVALID_G1	180
8.7.2.5	BLS_INVALID_G2	180
8.7.2.6	BLS_OK	180
8.7.3	Function Documentation	180
8.7.3.1	BLS_BLS381_ADD_G1()	180

8.7.3.2	BLS_BLS381_ADD_G2()	181
8.7.3.3	BLS_BLS381_KEY_PAIR_GENERATE()	181
8.7.3.4	BLS_BLS381_SIGN()	181
8.7.3.5	BLS_BLS381_VERIFY()	182
8.8	config_big_1024_58.h File Reference	182
8.8.1	Detailed Description	182
8.8.2	Macro Definition Documentation	183
8.8.2.1	BASEBITS_1024_58	183
8.8.2.2	MODBYTES_1024_58	183
8.9	config_big_256_56.h File Reference	183
8.9.1	Detailed Description	183
8.9.2	Macro Definition Documentation	183
8.9.2.1	BASEBITS_256_56	183
8.9.2.2	MODBYTES_256_56	184
8.10	config_big_384_56.h File Reference	184
8.10.1	Detailed Description	184
8.10.2	Macro Definition Documentation	184
8.10.2.1	BASEBITS_384_56	184
8.10.2.2	MODBYTES_384_56	184
8.11	config_big_384_58.h File Reference	184
8.11.1	Detailed Description	185
8.11.2	Macro Definition Documentation	185
8.11.2.1	BASEBITS_384_58	185
8.11.2.2	MODBYTES_384_58	185
8.12	config_big_448_58.h File Reference	185
8.12.1	Detailed Description	185
8.12.2	Macro Definition Documentation	186
8.12.2.1	BASEBITS_448_58	186
8.12.2.2	MODBYTES_448_58	186
8.13	config_ff_2048.h File Reference	186

8.13.1 Detailed Description . . . . .	186
8.13.2 Macro Definition Documentation . . . . .	186
8.13.2.1 FFLEN_2048 . . . . .	186
8.14 config_ff_3072.h File Reference . . . . .	187
8.14.1 Detailed Description . . . . .	187
8.14.2 Macro Definition Documentation . . . . .	187
8.14.2.1 FFLEN_3072 . . . . .	187
8.15 ecdh_BLS381.h File Reference . . . . .	187
8.15.1 Detailed Description . . . . .	188
8.15.2 Macro Definition Documentation . . . . .	188
8.15.2.1 ECDH_ERROR . . . . .	188
8.15.2.2 ECDH_INVALID . . . . .	188
8.15.2.3 ECDH_INVALID_PUBLIC_KEY . . . . .	188
8.15.2.4 ECDH_OK . . . . .	189
8.15.2.5 EFS_BLS381 . . . . .	189
8.15.2.6 EGS_BLS381 . . . . .	189
8.15.3 Function Documentation . . . . .	189
8.15.3.1 ECP_BLS381_ECIES_DECRYPT() . . . . .	189
8.15.3.2 ECP_BLS381_ECIES_ENCRYPT() . . . . .	190
8.15.3.3 ECP_BLS381_KEY_PAIR_GENERATE() . . . . .	190
8.15.3.4 ECP_BLS381_PUBLIC_KEY_VALIDATE() . . . . .	191
8.15.3.5 ECP_BLS381_SP_DSA() . . . . .	191
8.15.3.6 ECP_BLS381_SVDP_DH() . . . . .	191
8.15.3.7 ECP_BLS381_VP_DSA() . . . . .	192
8.16 ecdh_ED25519.h File Reference . . . . .	192
8.16.1 Detailed Description . . . . .	193
8.16.2 Macro Definition Documentation . . . . .	193
8.16.2.1 ECDH_ERROR . . . . .	193
8.16.2.2 ECDH_INVALID . . . . .	193
8.16.2.3 ECDH_INVALID_PUBLIC_KEY . . . . .	194



8.16.2.4	ECDH_OK	194
8.16.2.5	EFS_ED25519	194
8.16.2.6	EGS_ED25519	194
8.16.3	Function Documentation	194
8.16.3.1	ECP_ED25519_ECIES_DECRYPT()	194
8.16.3.2	ECP_ED25519_ECIES_ENCRYPT()	195
8.16.3.3	ECP_ED25519_KEY_PAIR_GENERATE()	195
8.16.3.4	ECP_ED25519_PUBLIC_KEY_VALIDATE()	196
8.16.3.5	ECP_ED25519_SP_DSA()	196
8.16.3.6	ECP_ED25519_SVDP_DH()	196
8.16.3.7	ECP_ED25519_VP_DSA()	197
8.17	ecdh_GOLDILOCKS.h File Reference	197
8.17.1	Detailed Description	198
8.17.2	Macro Definition Documentation	198
8.17.2.1	ECDH_ERROR	198
8.17.2.2	ECDH_INVALID	198
8.17.2.3	ECDH_INVALID_PUBLIC_KEY	199
8.17.2.4	ECDH_OK	199
8.17.2.5	EFS_GOLDILOCKS	199
8.17.2.6	EGS_GOLDILOCKS	199
8.17.3	Function Documentation	199
8.17.3.1	ECP_GOLDILOCKS_ECIES_DECRYPT()	199
8.17.3.2	ECP_GOLDILOCKS_ECIES_ENCRYPT()	200
8.17.3.3	ECP_GOLDILOCKS_KEY_PAIR_GENERATE()	200
8.17.3.4	ECP_GOLDILOCKS_PUBLIC_KEY_VALIDATE()	201
8.17.3.5	ECP_GOLDILOCKS_SP_DSA()	201
8.17.3.6	ECP_GOLDILOCKS_SVDP_DH()	201
8.17.3.7	ECP_GOLDILOCKS_VP_DSA()	202
8.18	ecdh_NIST256.h File Reference	202
8.18.1	Detailed Description	203

8.18.2	Macro Definition Documentation	203
8.18.2.1	ECDH_ERROR	203
8.18.2.2	ECDH_INVALID	203
8.18.2.3	ECDH_INVALID_PUBLIC_KEY	204
8.18.2.4	ECDH_OK	204
8.18.2.5	EFS_NIST256	204
8.18.2.6	EGS_NIST256	204
8.18.3	Function Documentation	204
8.18.3.1	ECP_NIST256_ECIES_DECRYPT()	204
8.18.3.2	ECP_NIST256_ECIES_ENCRYPT()	205
8.18.3.3	ECP_NIST256_KEY_PAIR_GENERATE()	205
8.18.3.4	ECP_NIST256_PUBLIC_KEY_VALIDATE()	206
8.18.3.5	ECP_NIST256_SP_DSA()	206
8.18.3.6	ECP_NIST256_SVDP_DH()	206
8.18.3.7	ECP_NIST256_VP_DSA()	207
8.19	ecdh_support.h File Reference	207
8.19.1	Detailed Description	208
8.19.2	Function Documentation	208
8.19.2.1	AES_CBC_IV0_DECRYPT()	208
8.19.2.2	AES_CBC_IV0_ENCRYPT()	209
8.19.2.3	ehashit()	209
8.19.2.4	HASH()	209
8.19.2.5	HMAC()	210
8.19.2.6	KDF2()	210
8.19.2.7	PBKDF2()	211
8.20	ecp2_BLS381.h File Reference	211
8.20.1	Detailed Description	213
8.20.2	Function Documentation	213
8.20.2.1	ECP2_BLS381_add()	213
8.20.2.2	ECP2_BLS381_affine()	213

8.20.2.3	<a href="#">ECP2_BLS381_copy()</a>	213
8.20.2.4	<a href="#">ECP2_BLS381_dbl()</a>	214
8.20.2.5	<a href="#">ECP2_BLS381_equals()</a>	214
8.20.2.6	<a href="#">ECP2_BLS381_frob()</a>	214
8.20.2.7	<a href="#">ECP2_BLS381_fromOctet()</a>	214
8.20.2.8	<a href="#">ECP2_BLS381_generator()</a>	215
8.20.2.9	<a href="#">ECP2_BLS381_get()</a>	215
8.20.2.10	<a href="#">ECP2_BLS381_inf()</a>	215
8.20.2.11	<a href="#">ECP2_BLS381_isinf()</a>	216
8.20.2.12	<a href="#">ECP2_BLS381_mapit()</a>	216
8.20.2.13	<a href="#">ECP2_BLS381_mul()</a>	216
8.20.2.14	<a href="#">ECP2_BLS381_mul4()</a>	217
8.20.2.15	<a href="#">ECP2_BLS381_neg()</a>	217
8.20.2.16	<a href="#">ECP2_BLS381_output()</a>	217
8.20.2.17	<a href="#">ECP2_BLS381_outputxyz()</a>	217
8.20.2.18	<a href="#">ECP2_BLS381_rhs()</a>	218
8.20.2.19	<a href="#">ECP2_BLS381_set()</a>	218
8.20.2.20	<a href="#">ECP2_BLS381_setx()</a>	218
8.20.2.21	<a href="#">ECP2_BLS381_sub()</a>	219
8.20.2.22	<a href="#">ECP2_BLS381_toOctet()</a>	219
8.20.3	<a href="#">Variable Documentation</a>	219
8.20.3.1	<a href="#">CURVE_A_BLS381</a>	219
8.20.3.2	<a href="#">CURVE_B_BLS381</a>	219
8.20.3.3	<a href="#">CURVE_B_I_BLS381</a>	220
8.20.3.4	<a href="#">CURVE_Bnx_BLS381</a>	220
8.20.3.5	<a href="#">CURVE_Cof_BLS381</a>	220
8.20.3.6	<a href="#">CURVE_Gx_BLS381</a>	220
8.20.3.7	<a href="#">CURVE_Gy_BLS381</a>	220
8.20.3.8	<a href="#">CURVE_Order_BLS381</a>	220
8.20.3.9	<a href="#">CURVE_Pxa_BLS381</a>	220

8.20.3.10 CURVE_Pxb_BLS381 . . . . .	220
8.20.3.11 CURVE_Pya_BLS381 . . . . .	221
8.20.3.12 CURVE_Pyb_BLS381 . . . . .	221
8.20.3.13 Fra_BLS381 . . . . .	221
8.20.3.14 Frb_BLS381 . . . . .	221
8.21 ecp_BLS381.h File Reference . . . . .	221
8.21.1 Detailed Description . . . . .	223
8.21.2 Function Documentation . . . . .	224
8.21.2.1 ECP_BLS381_add() . . . . .	224
8.21.2.2 ECP_BLS381_affine() . . . . .	224
8.21.2.3 ECP_BLS381_cfp() . . . . .	224
8.21.2.4 ECP_BLS381_copy() . . . . .	224
8.21.2.5 ECP_BLS381_dbl() . . . . .	225
8.21.2.6 ECP_BLS381_equals() . . . . .	225
8.21.2.7 ECP_BLS381_fromOctet() . . . . .	225
8.21.2.8 ECP_BLS381_generator() . . . . .	226
8.21.2.9 ECP_BLS381_get() . . . . .	226
8.21.2.10 ECP_BLS381_inf() . . . . .	226
8.21.2.11 ECP_BLS381_isinf() . . . . .	227
8.21.2.12 ECP_BLS381_mapit() . . . . .	227
8.21.2.13 ECP_BLS381_mul() . . . . .	227
8.21.2.14 ECP_BLS381_mul2() . . . . .	227
8.21.2.15 ECP_BLS381_neg() . . . . .	228
8.21.2.16 ECP_BLS381_output() . . . . .	228
8.21.2.17 ECP_BLS381_outputxyz() . . . . .	228
8.21.2.18 ECP_BLS381_pinmul() . . . . .	229
8.21.2.19 ECP_BLS381_rawoutput() . . . . .	229
8.21.2.20 ECP_BLS381_rhs() . . . . .	229
8.21.2.21 ECP_BLS381_set() . . . . .	229
8.21.2.22 ECP_BLS381_setx() . . . . .	230

8.21.2.23 ECP_BLS381_sub()	230
8.21.2.24 ECP_BLS381_toOctet()	230
8.21.3 Variable Documentation	232
8.21.3.1 CURVE_A_BLS381	232
8.21.3.2 CURVE_B_BLS381	232
8.21.3.3 CURVE_B_I_BLS381	232
8.21.3.4 CURVE_BB_BLS381	232
8.21.3.5 CURVE_Bnx_BLS381	232
8.21.3.6 CURVE_Cof_BLS381	232
8.21.3.7 CURVE_Cof_I_BLS381	233
8.21.3.8 CURVE_Cru_BLS381	233
8.21.3.9 CURVE_Gx_BLS381	233
8.21.3.10 CURVE_Gy_BLS381	233
8.21.3.11 CURVE_Order_BLS381	233
8.21.3.12 CURVE_Pxa_BLS381	233
8.21.3.13 CURVE_Pxaa_BLS381	233
8.21.3.14 CURVE_Pxaaa_BLS381	233
8.21.3.15 CURVE_Pxaab_BLS381	234
8.21.3.16 CURVE_Pxab_BLS381	234
8.21.3.17 CURVE_Pxaba_BLS381	234
8.21.3.18 CURVE_Pxab_b_BLS381	234
8.21.3.19 CURVE_Pxb_BLS381	234
8.21.3.20 CURVE_Pxba_BLS381	234
8.21.3.21 CURVE_Pxbaa_BLS381	234
8.21.3.22 CURVE_Pxbab_BLS381	234
8.21.3.23 CURVE_Pxbb_BLS381	235
8.21.3.24 CURVE_Pxbba_BLS381	235
8.21.3.25 CURVE_Pxbbb_BLS381	235
8.21.3.26 CURVE_Pya_BLS381	235
8.21.3.27 CURVE_Pyaa_BLS381	235

8.21.3.28 CURVE_Pyaaa_BLS381 . . . . .	235
8.21.3.29 CURVE_Pyaab_BLS381 . . . . .	235
8.21.3.30 CURVE_Pyab_BLS381 . . . . .	235
8.21.3.31 CURVE_Pyaba_BLS381 . . . . .	236
8.21.3.32 CURVE_Pyabb_BLS381 . . . . .	236
8.21.3.33 CURVE_Pyb_BLS381 . . . . .	236
8.21.3.34 CURVE_Pyba_BLS381 . . . . .	236
8.21.3.35 CURVE_Pybaa_BLS381 . . . . .	236
8.21.3.36 CURVE_Pybab_BLS381 . . . . .	236
8.21.3.37 CURVE_Pybb_BLS381 . . . . .	236
8.21.3.38 CURVE_Pybba_BLS381 . . . . .	236
8.21.3.39 CURVE_Pybbb_BLS381 . . . . .	237
8.21.3.40 CURVE_SB_BLS381 . . . . .	237
8.21.3.41 CURVE_W_BLS381 . . . . .	237
8.21.3.42 CURVE_WB_BLS381 . . . . .	237
8.21.3.43 Fra_BLS381 . . . . .	237
8.21.3.44 Frb_BLS381 . . . . .	237
8.22 ecp_ED25519.h File Reference . . . . .	237
8.22.1 Detailed Description . . . . .	240
8.22.2 Function Documentation . . . . .	240
8.22.2.1 ECP_ED25519_add() . . . . .	240
8.22.2.2 ECP_ED25519_affine() . . . . .	240
8.22.2.3 ECP_ED25519_cfp() . . . . .	240
8.22.2.4 ECP_ED25519_copy() . . . . .	241
8.22.2.5 ECP_ED25519_dbl() . . . . .	241
8.22.2.6 ECP_ED25519_equals() . . . . .	241
8.22.2.7 ECP_ED25519_fromOctet() . . . . .	241
8.22.2.8 ECP_ED25519_generator() . . . . .	242
8.22.2.9 ECP_ED25519_get() . . . . .	242
8.22.2.10 ECP_ED25519_inf() . . . . .	242

8.22.2.11 ECP_ED25519_isinf()	243
8.22.2.12 ECP_ED25519_mapit()	243
8.22.2.13 ECP_ED25519_mul()	243
8.22.2.14 ECP_ED25519_mul2()	244
8.22.2.15 ECP_ED25519_neg()	244
8.22.2.16 ECP_ED25519_output()	244
8.22.2.17 ECP_ED25519_outputxyz()	244
8.22.2.18 ECP_ED25519_pinmul()	245
8.22.2.19 ECP_ED25519_rawoutput()	245
8.22.2.20 ECP_ED25519_rhs()	245
8.22.2.21 ECP_ED25519_set()	245
8.22.2.22 ECP_ED25519_setx()	246
8.22.2.23 ECP_ED25519_sub()	246
8.22.2.24 ECP_ED25519_toOctet()	247
8.22.3 Variable Documentation	247
8.22.3.1 CURVE_A_ED25519	247
8.22.3.2 CURVE_B_ED25519	247
8.22.3.3 CURVE_B_I_ED25519	247
8.22.3.4 CURVE_BB_ED25519	247
8.22.3.5 CURVE_Bnx_ED25519	247
8.22.3.6 CURVE_Cof_ED25519	248
8.22.3.7 CURVE_Cof_I_ED25519	248
8.22.3.8 CURVE_Cru_ED25519	248
8.22.3.9 CURVE_Gx_ED25519	248
8.22.3.10 CURVE_Gy_ED25519	248
8.22.3.11 CURVE_Order_ED25519	248
8.22.3.12 CURVE_Pxa_ED25519	248
8.22.3.13 CURVE_Pxaa_ED25519	248
8.22.3.14 CURVE_Pxaaa_ED25519	249
8.22.3.15 CURVE_Pxaab_ED25519	249

8.22.3.16 CURVE_Pxab_ED25519 . . . . .	249
8.22.3.17 CURVE_Pxaba_ED25519 . . . . .	249
8.22.3.18 CURVE_Pxabb_ED25519 . . . . .	249
8.22.3.19 CURVE_Pxb_ED25519 . . . . .	249
8.22.3.20 CURVE_Pxba_ED25519 . . . . .	249
8.22.3.21 CURVE_Pxbaa_ED25519 . . . . .	249
8.22.3.22 CURVE_Pxbab_ED25519 . . . . .	250
8.22.3.23 CURVE_Pxbb_ED25519 . . . . .	250
8.22.3.24 CURVE_Pxbba_ED25519 . . . . .	250
8.22.3.25 CURVE_Pxbbb_ED25519 . . . . .	250
8.22.3.26 CURVE_Pya_ED25519 . . . . .	250
8.22.3.27 CURVE_Pyaa_ED25519 . . . . .	250
8.22.3.28 CURVE_Pyaaa_ED25519 . . . . .	250
8.22.3.29 CURVE_Pyaab_ED25519 . . . . .	250
8.22.3.30 CURVE_Pyab_ED25519 . . . . .	251
8.22.3.31 CURVE_Pyaba_ED25519 . . . . .	251
8.22.3.32 CURVE_Pyabb_ED25519 . . . . .	251
8.22.3.33 CURVE_Pyb_ED25519 . . . . .	251
8.22.3.34 CURVE_Pyba_ED25519 . . . . .	251
8.22.3.35 CURVE_Pybaa_ED25519 . . . . .	251
8.22.3.36 CURVE_Pybab_ED25519 . . . . .	251
8.22.3.37 CURVE_Pybb_ED25519 . . . . .	251
8.22.3.38 CURVE_Pybba_ED25519 . . . . .	252
8.22.3.39 CURVE_Pybbb_ED25519 . . . . .	252
8.22.3.40 CURVE_SB_ED25519 . . . . .	252
8.22.3.41 CURVE_W_ED25519 . . . . .	252
8.22.3.42 CURVE_WB_ED25519 . . . . .	252
8.22.3.43 Fra_25519 . . . . .	252
8.22.3.44 Frb_25519 . . . . .	252
8.23 ecp_GOLDILOCKS.h File Reference . . . . .	253



8.23.1 Detailed Description . . . . .	255
8.23.2 Function Documentation . . . . .	255
8.23.2.1 ECP_GOLDILOCKS_add() . . . . .	255
8.23.2.2 ECP_GOLDILOCKS_affine() . . . . .	255
8.23.2.3 ECP_GOLDILOCKS_cfp() . . . . .	256
8.23.2.4 ECP_GOLDILOCKS_copy() . . . . .	256
8.23.2.5 ECP_GOLDILOCKS_dbl() . . . . .	256
8.23.2.6 ECP_GOLDILOCKS_equals() . . . . .	256
8.23.2.7 ECP_GOLDILOCKS_fromOctet() . . . . .	257
8.23.2.8 ECP_GOLDILOCKS_generator() . . . . .	257
8.23.2.9 ECP_GOLDILOCKS_get() . . . . .	257
8.23.2.10 ECP_GOLDILOCKS_inf() . . . . .	258
8.23.2.11 ECP_GOLDILOCKS_isinf() . . . . .	258
8.23.2.12 ECP_GOLDILOCKS_mapit() . . . . .	258
8.23.2.13 ECP_GOLDILOCKS_mul() . . . . .	258
8.23.2.14 ECP_GOLDILOCKS_mul2() . . . . .	259
8.23.2.15 ECP_GOLDILOCKS_neg() . . . . .	259
8.23.2.16 ECP_GOLDILOCKS_output() . . . . .	259
8.23.2.17 ECP_GOLDILOCKS_outputxyz() . . . . .	260
8.23.2.18 ECP_GOLDILOCKS_pinmul() . . . . .	260
8.23.2.19 ECP_GOLDILOCKS_rawoutput() . . . . .	260
8.23.2.20 ECP_GOLDILOCKS_rhs() . . . . .	260
8.23.2.21 ECP_GOLDILOCKS_set() . . . . .	261
8.23.2.22 ECP_GOLDILOCKS_setx() . . . . .	261
8.23.2.23 ECP_GOLDILOCKS_sub() . . . . .	262
8.23.2.24 ECP_GOLDILOCKS_toOctet() . . . . .	262
8.23.3 Variable Documentation . . . . .	262
8.23.3.1 CURVE_A_GOLDILOCKS . . . . .	262
8.23.3.2 CURVE_B_GOLDILOCKS . . . . .	262
8.23.3.3 CURVE_B_I_GOLDILOCKS . . . . .	262

8.23.3.4	CURVE_BB_GOLDILOCKS	263
8.23.3.5	CURVE_Bnx_GOLDILOCKS	263
8.23.3.6	CURVE_Cof_GOLDILOCKS	263
8.23.3.7	CURVE_Cof_I_GOLDILOCKS	263
8.23.3.8	CURVE_Cru_GOLDILOCKS	263
8.23.3.9	CURVE_Gx_GOLDILOCKS	263
8.23.3.10	CURVE_Gy_GOLDILOCKS	263
8.23.3.11	CURVE_Order_GOLDILOCKS	263
8.23.3.12	CURVE_Pxa_GOLDILOCKS	264
8.23.3.13	CURVE_Pxaa_GOLDILOCKS	264
8.23.3.14	CURVE_Pxaaa_GOLDILOCKS	264
8.23.3.15	CURVE_Pxaab_GOLDILOCKS	264
8.23.3.16	CURVE_Pxab_GOLDILOCKS	264
8.23.3.17	CURVE_Pxaba_GOLDILOCKS	264
8.23.3.18	CURVE_Pxab_b_GOLDILOCKS	264
8.23.3.19	CURVE_Pxb_GOLDILOCKS	264
8.23.3.20	CURVE_Pxba_GOLDILOCKS	265
8.23.3.21	CURVE_Pxbaa_GOLDILOCKS	265
8.23.3.22	CURVE_Pxbab_GOLDILOCKS	265
8.23.3.23	CURVE_Pxbb_GOLDILOCKS	265
8.23.3.24	CURVE_Pxbba_GOLDILOCKS	265
8.23.3.25	CURVE_Pxbbb_GOLDILOCKS	265
8.23.3.26	CURVE_Pya_GOLDILOCKS	265
8.23.3.27	CURVE_Pyaa_GOLDILOCKS	265
8.23.3.28	CURVE_Pyaaa_GOLDILOCKS	266
8.23.3.29	CURVE_Pyaab_GOLDILOCKS	266
8.23.3.30	CURVE_Pyab_GOLDILOCKS	266
8.23.3.31	CURVE_Pyaba_GOLDILOCKS	266
8.23.3.32	CURVE_Pyabb_GOLDILOCKS	266
8.23.3.33	CURVE_Pyb_GOLDILOCKS	266

8.23.3.34 CURVE_Pyba_GOLDILOCKS . . . . .	266
8.23.3.35 CURVE_Pybaa_GOLDILOCKS . . . . .	266
8.23.3.36 CURVE_Pybab_GOLDILOCKS . . . . .	267
8.23.3.37 CURVE_Pybb_GOLDILOCKS . . . . .	267
8.23.3.38 CURVE_Pybba_GOLDILOCKS . . . . .	267
8.23.3.39 CURVE_Pybbb_GOLDILOCKS . . . . .	267
8.23.3.40 CURVE_SB_GOLDILOCKS . . . . .	267
8.23.3.41 CURVE_W_GOLDILOCKS . . . . .	267
8.23.3.42 CURVE_WB_GOLDILOCKS . . . . .	267
8.23.3.43 Fra_GOLDILOCKS . . . . .	267
8.23.3.44 Frb_GOLDILOCKS . . . . .	268
8.24 ecp_NIST256.h File Reference . . . . .	268
8.24.1 Detailed Description . . . . .	270
8.24.2 Function Documentation . . . . .	270
8.24.2.1 ECP_NIST256_add() . . . . .	270
8.24.2.2 ECP_NIST256_affine() . . . . .	270
8.24.2.3 ECP_NIST256_cfp() . . . . .	271
8.24.2.4 ECP_NIST256_copy() . . . . .	271
8.24.2.5 ECP_NIST256_dbl() . . . . .	271
8.24.2.6 ECP_NIST256_equals() . . . . .	271
8.24.2.7 ECP_NIST256_fromOctet() . . . . .	272
8.24.2.8 ECP_NIST256_generator() . . . . .	272
8.24.2.9 ECP_NIST256_get() . . . . .	272
8.24.2.10 ECP_NIST256_inf() . . . . .	273
8.24.2.11 ECP_NIST256_isinf() . . . . .	273
8.24.2.12 ECP_NIST256_mapit() . . . . .	273
8.24.2.13 ECP_NIST256_mul() . . . . .	274
8.24.2.14 ECP_NIST256_mul2() . . . . .	274
8.24.2.15 ECP_NIST256_neg() . . . . .	274
8.24.2.16 ECP_NIST256_output() . . . . .	274

8.24.2.17 ECP_NIST256_outputxyz()	275
8.24.2.18 ECP_NIST256_pinmul()	275
8.24.2.19 ECP_NIST256_rawoutput()	275
8.24.2.20 ECP_NIST256_rhs()	276
8.24.2.21 ECP_NIST256_set()	276
8.24.2.22 ECP_NIST256_setx()	276
8.24.2.23 ECP_NIST256_sub()	277
8.24.2.24 ECP_NIST256_toOctet()	277
8.24.3 Variable Documentation	277
8.24.3.1 CURVE_A_NIST256	277
8.24.3.2 CURVE_B_I_NIST256	277
8.24.3.3 CURVE_B_NIST256	278
8.24.3.4 CURVE_BB_NIST256	278
8.24.3.5 CURVE_Bnx_NIST256	278
8.24.3.6 CURVE_Cof_I_NIST256	278
8.24.3.7 CURVE_Cof_NIST256	278
8.24.3.8 CURVE_Cru_NIST256	278
8.24.3.9 CURVE_Gx_NIST256	278
8.24.3.10 CURVE_Gy_NIST256	278
8.24.3.11 CURVE_Order_NIST256	279
8.24.3.12 CURVE_Pxa_NIST256	279
8.24.3.13 CURVE_Pxaa_NIST256	279
8.24.3.14 CURVE_Pxaaa_NIST256	279
8.24.3.15 CURVE_Pxaab_NIST256	279
8.24.3.16 CURVE_Pxab_NIST256	279
8.24.3.17 CURVE_Pxaba_NIST256	279
8.24.3.18 CURVE_Pxabb_NIST256	279
8.24.3.19 CURVE_Pxb_NIST256	280
8.24.3.20 CURVE_Pxba_NIST256	280
8.24.3.21 CURVE_Pxbaa_NIST256	280

8.24.3.22 CURVE_Pxbab_NIST256 . . . . .	280
8.24.3.23 CURVE_Pxbb_NIST256 . . . . .	280
8.24.3.24 CURVE_Pxbba_NIST256 . . . . .	280
8.24.3.25 CURVE_Pxbbb_NIST256 . . . . .	280
8.24.3.26 CURVE_Pya_NIST256 . . . . .	280
8.24.3.27 CURVE_Pyaa_NIST256 . . . . .	281
8.24.3.28 CURVE_Pyaaa_NIST256 . . . . .	281
8.24.3.29 CURVE_Pyaab_NIST256 . . . . .	281
8.24.3.30 CURVE_Pyab_NIST256 . . . . .	281
8.24.3.31 CURVE_Pyaba_NIST256 . . . . .	281
8.24.3.32 CURVE_Pyabb_NIST256 . . . . .	281
8.24.3.33 CURVE_Pyb_NIST256 . . . . .	281
8.24.3.34 CURVE_Pyba_NIST256 . . . . .	281
8.24.3.35 CURVE_Pybaa_NIST256 . . . . .	282
8.24.3.36 CURVE_Pybab_NIST256 . . . . .	282
8.24.3.37 CURVE_Pybb_NIST256 . . . . .	282
8.24.3.38 CURVE_Pybba_NIST256 . . . . .	282
8.24.3.39 CURVE_Pybbb_NIST256 . . . . .	282
8.24.3.40 CURVE_SB_NIST256 . . . . .	282
8.24.3.41 CURVE_W_NIST256 . . . . .	282
8.24.3.42 CURVE_WB_NIST256 . . . . .	282
8.24.3.43 Fra_NIST256 . . . . .	283
8.24.3.44 Frb_NIST256 . . . . .	283
8.25 ff_2048.h File Reference . . . . .	283
8.25.1 Detailed Description . . . . .	285
8.25.2 Macro Definition Documentation . . . . .	285
8.25.2.1 HFLEN_2048 . . . . .	285
8.25.2.2 P_EXCESS_2048 . . . . .	285
8.25.2.3 P_FEXCESS_2048 . . . . .	285
8.25.2.4 P_MBITS_2048 . . . . .	285

8.25.2.5	P_TBITS_2048	285
8.25.3	Function Documentation	285
8.25.3.1	FF_2048_add()	285
8.25.3.2	FF_2048_cfactor()	286
8.25.3.3	FF_2048_comp()	286
8.25.3.4	FF_2048_copy()	287
8.25.3.5	FF_2048_dec()	287
8.25.3.6	FF_2048_dmod()	287
8.25.3.7	FF_2048_fromOctet()	288
8.25.3.8	FF_2048_inc()	288
8.25.3.9	FF_2048_init()	288
8.25.3.10	FF_2048_invmodp()	289
8.25.3.11	FF_2048_iszilch()	289
8.25.3.12	FF_2048_lastbits()	289
8.25.3.13	FF_2048_mod()	290
8.25.3.14	FF_2048_mul()	290
8.25.3.15	FF_2048_norm()	290
8.25.3.16	FF_2048_one()	291
8.25.3.17	FF_2048_output()	291
8.25.3.18	FF_2048_parity()	291
8.25.3.19	FF_2048_pow()	291
8.25.3.20	FF_2048_pow2()	292
8.25.3.21	FF_2048_power()	292
8.25.3.22	FF_2048_prime()	293
8.25.3.23	FF_2048_random()	293
8.25.3.24	FF_2048_randomnum()	293
8.25.3.25	FF_2048_rawoutput()	294
8.25.3.26	FF_2048_shl()	294
8.25.3.27	FF_2048_shr()	294
8.25.3.28	FF_2048_skpow()	295

8.25.3.29	FF_2048_skspow()	295
8.25.3.30	FF_2048_sqr()	295
8.25.3.31	FF_2048_sub()	296
8.25.3.32	FF_2048_toOctet()	296
8.25.3.33	FF_2048_zero()	296
8.26	ff_3072.h File Reference	297
8.26.1	Detailed Description	298
8.26.2	Macro Definition Documentation	299
8.26.2.1	HFLen_3072	299
8.26.2.2	P_EXCESS_3072	299
8.26.2.3	P_FEXCESS_3072	299
8.26.2.4	P_MBITS_3072	299
8.26.2.5	P_TBITS_3072	299
8.26.3	Function Documentation	299
8.26.3.1	FF_3072_add()	299
8.26.3.2	FF_3072_cfactor()	300
8.26.3.3	FF_3072_comp()	300
8.26.3.4	FF_3072_copy()	301
8.26.3.5	FF_3072_dec()	301
8.26.3.6	FF_3072_dmod()	301
8.26.3.7	FF_3072_fromOctet()	302
8.26.3.8	FF_3072_inc()	302
8.26.3.9	FF_3072_init()	302
8.26.3.10	FF_3072_invmodp()	303
8.26.3.11	FF_3072_iszilch()	303
8.26.3.12	FF_3072_lastbits()	303
8.26.3.13	FF_3072_mod()	304
8.26.3.14	FF_3072_mul()	304
8.26.3.15	FF_3072_norm()	304
8.26.3.16	FF_3072_one()	305

8.26.3.17	FF_3072_output()	305
8.26.3.18	FF_3072_parity()	305
8.26.3.19	FF_3072_pow()	305
8.26.3.20	FF_3072_pow2()	306
8.26.3.21	FF_3072_power()	306
8.26.3.22	FF_3072_prime()	307
8.26.3.23	FF_3072_random()	307
8.26.3.24	FF_3072_randomnum()	307
8.26.3.25	FF_3072_rawoutput()	308
8.26.3.26	FF_3072_shl()	308
8.26.3.27	FF_3072_shr()	308
8.26.3.28	FF_3072_skpow()	309
8.26.3.29	FF_3072_skspow()	309
8.26.3.30	FF_3072_sqr()	309
8.26.3.31	FF_3072_sub()	310
8.26.3.32	FF_3072_toOctet()	310
8.26.3.33	FF_3072_zero()	310
8.27	fp12_BLS381.h File Reference	311
8.27.1	Detailed Description	312
8.27.2	Function Documentation	312
8.27.2.1	FP12_BLS381_cmove()	312
8.27.2.2	FP12_BLS381_compow()	313
8.27.2.3	FP12_BLS381_conj()	313
8.27.2.4	FP12_BLS381_copy()	313
8.27.2.5	FP12_BLS381_equals()	314
8.27.2.6	FP12_BLS381_frob()	314
8.27.2.7	FP12_BLS381_from_FP4()	314
8.27.2.8	FP12_BLS381_from_FP4s()	315
8.27.2.9	FP12_BLS381_fromOctet()	315
8.27.2.10	FP12_BLS381_inv()	315



8.27.2.11 FP12_BLS381_isunity()	316
8.27.2.12 FP12_BLS381_iszilch()	316
8.27.2.13 FP12_BLS381_mul()	316
8.27.2.14 FP12_BLS381_norm()	317
8.27.2.15 FP12_BLS381_one()	317
8.27.2.16 FP12_BLS381_output()	317
8.27.2.17 FP12_BLS381_pinpow()	317
8.27.2.18 FP12_BLS381_pow()	318
8.27.2.19 FP12_BLS381_pow4()	318
8.27.2.20 FP12_BLS381_reduce()	318
8.27.2.21 FP12_BLS381_smul()	318
8.27.2.22 FP12_BLS381_sqr()	319
8.27.2.23 FP12_BLS381_ssmul()	319
8.27.2.24 FP12_BLS381_toOctet()	319
8.27.2.25 FP12_BLS381_trace()	320
8.27.2.26 FP12_BLS381_usqr()	320
8.27.2.27 FP12_BLS381_zero()	320
8.27.3 Variable Documentation	320
8.27.3.1 Fra_BLS381	320
8.27.3.2 Frb_BLS381	321
8.28 fp2_BLS381.h File Reference	321
8.28.1 Detailed Description	322
8.28.2 Function Documentation	322
8.28.2.1 FP2_BLS381_add()	322
8.28.2.2 FP2_BLS381_cmove()	323
8.28.2.3 FP2_BLS381_conj()	323
8.28.2.4 FP2_BLS381_copy()	323
8.28.2.5 FP2_BLS381_div2()	324
8.28.2.6 FP2_BLS381_div_ip()	324
8.28.2.7 FP2_BLS381_div_ip2()	324

8.28.2.8	FP2_BLS381_equals()	325
8.28.2.9	FP2_BLS381_from_BIG()	325
8.28.2.10	FP2_BLS381_from_BIGs()	325
8.28.2.11	FP2_BLS381_from_FP()	326
8.28.2.12	FP2_BLS381_from_FPs()	326
8.28.2.13	FP2_BLS381_imul()	326
8.28.2.14	FP2_BLS381_inv()	326
8.28.2.15	FP2_BLS381_isunity()	327
8.28.2.16	FP2_BLS381_iszilch()	327
8.28.2.17	FP2_BLS381_mul()	327
8.28.2.18	FP2_BLS381_mul_ip()	328
8.28.2.19	FP2_BLS381_neg()	328
8.28.2.20	FP2_BLS381_norm()	328
8.28.2.21	FP2_BLS381_one()	329
8.28.2.22	FP2_BLS381_output()	329
8.28.2.23	FP2_BLS381_pmul()	329
8.28.2.24	FP2_BLS381_pow()	329
8.28.2.25	FP2_BLS381_rawoutput()	330
8.28.2.26	FP2_BLS381_reduce()	330
8.28.2.27	FP2_BLS381_sqr()	330
8.28.2.28	FP2_BLS381_sqrt()	330
8.28.2.29	FP2_BLS381_sub()	331
8.28.2.30	FP2_BLS381_times_i()	331
8.28.2.31	FP2_BLS381_zero()	331
8.29	fp4_BLS381.h File Reference	331
8.29.1	Detailed Description	333
8.29.2	Function Documentation	333
8.29.2.1	FP4_BLS381_add()	333
8.29.2.2	FP4_BLS381_cmove()	334
8.29.2.3	FP4_BLS381_conj()	334

8.29.2.4	FP4_BLS381_copy()	334
8.29.2.5	FP4_BLS381_div2()	335
8.29.2.6	FP4_BLS381_div_2i()	335
8.29.2.7	FP4_BLS381_div_i()	335
8.29.2.8	FP4_BLS381_equals()	336
8.29.2.9	FP4_BLS381_frob()	336
8.29.2.10	FP4_BLS381_from_FP2()	336
8.29.2.11	FP4_BLS381_from_FP2H()	336
8.29.2.12	FP4_BLS381_from_FP2s()	337
8.29.2.13	FP4_BLS381_imul()	337
8.29.2.14	FP4_BLS381_inv()	337
8.29.2.15	FP4_BLS381_isreal()	338
8.29.2.16	FP4_BLS381_isunity()	338
8.29.2.17	FP4_BLS381_iszilch()	338
8.29.2.18	FP4_BLS381_mul()	339
8.29.2.19	FP4_BLS381_nconj()	339
8.29.2.20	FP4_BLS381_neg()	339
8.29.2.21	FP4_BLS381_norm()	340
8.29.2.22	FP4_BLS381_one()	340
8.29.2.23	FP4_BLS381_output()	340
8.29.2.24	FP4_BLS381_pmul()	340
8.29.2.25	FP4_BLS381_pow()	341
8.29.2.26	FP4_BLS381_qmul()	341
8.29.2.27	FP4_BLS381_rawoutput()	341
8.29.2.28	FP4_BLS381_reduce()	341
8.29.2.29	FP4_BLS381_sqr()	342
8.29.2.30	FP4_BLS381_sqrt()	342
8.29.2.31	FP4_BLS381_sub()	342
8.29.2.32	FP4_BLS381_times_i()	343
8.29.2.33	FP4_BLS381_xtr_A()	343

8.29.2.34	FP4_BLS381_xtr_D()	343
8.29.2.35	FP4_BLS381_xtr_pow()	343
8.29.2.36	FP4_BLS381_xtr_pow2()	344
8.29.2.37	FP4_BLS381_zero()	344
8.30	fp_25519.h File Reference	345
8.30.1	Detailed Description	346
8.30.2	Macro Definition Documentation	346
8.30.2.1	FEXCESS_25519	346
8.30.2.2	MODBITS_25519	347
8.30.2.3	OMASK_25519	347
8.30.2.4	TBITS_25519	347
8.30.2.5	TMASK_25519	347
8.30.3	Function Documentation	347
8.30.3.1	FP_25519_add()	347
8.30.3.2	FP_25519_cmove()	347
8.30.3.3	FP_25519_copy()	348
8.30.3.4	FP_25519_cswap()	348
8.30.3.5	FP_25519_div2()	348
8.30.3.6	FP_25519_equals()	349
8.30.3.7	FP_25519_imul()	349
8.30.3.8	FP_25519_inv()	349
8.30.3.9	FP_25519_iszilch()	350
8.30.3.10	FP_25519_mod()	350
8.30.3.11	FP_25519_mul()	350
8.30.3.12	FP_25519_neg()	351
8.30.3.13	FP_25519_norm()	351
8.30.3.14	FP_25519_nres()	351
8.30.3.15	FP_25519_one()	352
8.30.3.16	FP_25519_output()	352
8.30.3.17	FP_25519_pow()	352

8.30.3.18	FP_25519_qr()	352
8.30.3.19	FP_25519_rawoutput()	353
8.30.3.20	FP_25519_rcopy()	353
8.30.3.21	FP_25519_redc()	353
8.30.3.22	FP_25519_reduce()	353
8.30.3.23	FP_25519_sqr()	354
8.30.3.24	FP_25519_sqrt()	354
8.30.3.25	FP_25519_sub()	354
8.30.3.26	FP_25519_zero()	355
8.30.4	Variable Documentation	355
8.30.4.1	MConst_25519	355
8.30.4.2	Modulus_25519	355
8.30.4.3	R2modp_25519	355
8.31	fp_BLS381.h File Reference	355
8.31.1	Detailed Description	357
8.31.2	Macro Definition Documentation	357
8.31.2.1	FEXCESS_BLS381	357
8.31.2.2	MODBITS_BLS381	357
8.31.2.3	OMASK_BLS381	357
8.31.2.4	TBITS_BLS381	358
8.31.2.5	TMASK_BLS381	358
8.31.3	Function Documentation	358
8.31.3.1	FP_BLS381_add()	358
8.31.3.2	FP_BLS381_cmove()	358
8.31.3.3	FP_BLS381_copy()	359
8.31.3.4	FP_BLS381_cswap()	359
8.31.3.5	FP_BLS381_div2()	359
8.31.3.6	FP_BLS381_equals()	359
8.31.3.7	FP_BLS381_imul()	360
8.31.3.8	FP_BLS381_inv()	360

8.31.3.9	FP_BLS381_iszilch()	360
8.31.3.10	FP_BLS381_mod()	361
8.31.3.11	FP_BLS381_mul()	361
8.31.3.12	FP_BLS381_neg()	361
8.31.3.13	FP_BLS381_norm()	362
8.31.3.14	FP_BLS381_nres()	362
8.31.3.15	FP_BLS381_one()	362
8.31.3.16	FP_BLS381_output()	362
8.31.3.17	FP_BLS381_pow()	363
8.31.3.18	FP_BLS381_qr()	363
8.31.3.19	FP_BLS381_rawoutput()	363
8.31.3.20	FP_BLS381_rcopy()	363
8.31.3.21	FP_BLS381_redc()	364
8.31.3.22	FP_BLS381_reduce()	364
8.31.3.23	FP_BLS381_sqr()	364
8.31.3.24	FP_BLS381_sqrt()	365
8.31.3.25	FP_BLS381_sub()	365
8.31.3.26	FP_BLS381_zero()	365
8.31.4	Variable Documentation	365
8.31.4.1	MConst_BLS381	365
8.31.4.2	Modulus_BLS381	366
8.31.4.3	R2modp_BLS381	366
8.32	fp_GOLDILOCKS.h File Reference	366
8.32.1	Detailed Description	367
8.32.2	Macro Definition Documentation	368
8.32.2.1	FEXCESS_GOLDILOCKS	368
8.32.2.2	MODBITS_GOLDILOCKS	368
8.32.2.3	OMASK_GOLDILOCKS	368
8.32.2.4	TBITS_GOLDILOCKS	368
8.32.2.5	TMASK_GOLDILOCKS	368

8.32.3	Function Documentation	368
8.32.3.1	FP_GOLDILOCKS_add()	368
8.32.3.2	FP_GOLDILOCKS_cmove()	369
8.32.3.3	FP_GOLDILOCKS_copy()	369
8.32.3.4	FP_GOLDILOCKS_cswap()	369
8.32.3.5	FP_GOLDILOCKS_div2()	370
8.32.3.6	FP_GOLDILOCKS_equals()	370
8.32.3.7	FP_GOLDILOCKS_imul()	370
8.32.3.8	FP_GOLDILOCKS_inv()	371
8.32.3.9	FP_GOLDILOCKS_iszilch()	371
8.32.3.10	FP_GOLDILOCKS_mod()	371
8.32.3.11	FP_GOLDILOCKS_mul()	372
8.32.3.12	FP_GOLDILOCKS_neg()	372
8.32.3.13	FP_GOLDILOCKS_norm()	372
8.32.3.14	FP_GOLDILOCKS_nres()	372
8.32.3.15	FP_GOLDILOCKS_one()	373
8.32.3.16	FP_GOLDILOCKS_output()	373
8.32.3.17	FP_GOLDILOCKS_pow()	373
8.32.3.18	FP_GOLDILOCKS_qr()	374
8.32.3.19	FP_GOLDILOCKS_rawoutput()	374
8.32.3.20	FP_GOLDILOCKS_rcopy()	374
8.32.3.21	FP_GOLDILOCKS_redc()	374
8.32.3.22	FP_GOLDILOCKS_reduce()	375
8.32.3.23	FP_GOLDILOCKS_sqr()	375
8.32.3.24	FP_GOLDILOCKS_sqrt()	375
8.32.3.25	FP_GOLDILOCKS_sub()	375
8.32.3.26	FP_GOLDILOCKS_zero()	376
8.32.4	Variable Documentation	376
8.32.4.1	MConst_GOLDILOCKS	376
8.32.4.2	Modulus_GOLDILOCKS	376

8.32.4.3 R2modp_GOLDILOCKS . . . . .	376
8.33 fp_NIST256.h File Reference . . . . .	376
8.33.1 Detailed Description . . . . .	378
8.33.2 Macro Definition Documentation . . . . .	378
8.33.2.1 FEXCESS_NIST256 . . . . .	378
8.33.2.2 MODBITS_NIST256 . . . . .	378
8.33.2.3 OMASK_NIST256 . . . . .	379
8.33.2.4 TBITS_NIST256 . . . . .	379
8.33.2.5 TMASK_NIST256 . . . . .	379
8.33.3 Function Documentation . . . . .	379
8.33.3.1 FP_NIST256_add() . . . . .	379
8.33.3.2 FP_NIST256_cmove() . . . . .	379
8.33.3.3 FP_NIST256_copy() . . . . .	380
8.33.3.4 FP_NIST256_cswap() . . . . .	380
8.33.3.5 FP_NIST256_div2() . . . . .	380
8.33.3.6 FP_NIST256_equals() . . . . .	381
8.33.3.7 FP_NIST256_imul() . . . . .	381
8.33.3.8 FP_NIST256_inv() . . . . .	381
8.33.3.9 FP_NIST256_iszilch() . . . . .	382
8.33.3.10 FP_NIST256_mod() . . . . .	382
8.33.3.11 FP_NIST256_mul() . . . . .	382
8.33.3.12 FP_NIST256_neg() . . . . .	383
8.33.3.13 FP_NIST256_norm() . . . . .	383
8.33.3.14 FP_NIST256_nres() . . . . .	383
8.33.3.15 FP_NIST256_one() . . . . .	383
8.33.3.16 FP_NIST256_output() . . . . .	384
8.33.3.17 FP_NIST256_pow() . . . . .	384
8.33.3.18 FP_NIST256_qr() . . . . .	384
8.33.3.19 FP_NIST256_rawoutput() . . . . .	384
8.33.3.20 FP_NIST256_rcopy() . . . . .	385



8.33.3.21	FP_NIST256_redc()	385
8.33.3.22	FP_NIST256_reduce()	385
8.33.3.23	FP_NIST256_sqr()	385
8.33.3.24	FP_NIST256_sqrt()	386
8.33.3.25	FP_NIST256_sub()	386
8.33.3.26	FP_NIST256_zero()	386
8.33.4	Variable Documentation	387
8.33.4.1	MConst_NIST256	387
8.33.4.2	Modulus_NIST256	387
8.33.4.3	R2modp_NIST256	387
8.34	mpin_BLS381.h File Reference	387
8.34.1	Detailed Description	389
8.34.2	Macro Definition Documentation	389
8.34.2.1	M_SIZE_BLS381	389
8.34.2.2	MAXPIN	389
8.34.2.3	MESSAGE_SIZE	389
8.34.2.4	MPIN_BAD_PIN	389
8.34.2.5	MPIN_INVALID_POINT	389
8.34.2.6	MPIN_OK	390
8.34.2.7	MPIN_PAS	390
8.34.2.8	PBLEN	390
8.34.2.9	PFS_BLS381	390
8.34.2.10	PGS_BLS381	390
8.34.3	Function Documentation	390
8.34.3.1	MPIN_BLS381_CLIENT()	390
8.34.3.2	MPIN_BLS381_CLIENT_1()	391
8.34.3.3	MPIN_BLS381_CLIENT_2()	392
8.34.3.4	MPIN_BLS381_CLIENT_KEY()	392
8.34.3.5	MPIN_BLS381_DECODING()	393
8.34.3.6	MPIN_BLS381_ENCODING()	393

8.34.3.7	MPIN_BLS381_EXTRACT_FACTOR()	394
8.34.3.8	MPIN_BLS381_EXTRACT_PIN()	394
8.34.3.9	MPIN_BLS381_GET_CLIENT_PERMIT()	394
8.34.3.10	MPIN_BLS381_GET_CLIENT_SECRET()	396
8.34.3.11	MPIN_BLS381_GET_DVS_KEYPAIR()	396
8.34.3.12	MPIN_BLS381_GET_G1_MULTIPLE()	397
8.34.3.13	MPIN_BLS381_GET_G2_MULTIPLE()	397
8.34.3.14	MPIN_BLS381_GET_SERVER_SECRET()	397
8.34.3.15	MPIN_BLS381_GET_Y()	398
8.34.3.16	MPIN_BLS381_KANGAROO()	398
8.34.3.17	MPIN_BLS381_PRECOMPUTE()	398
8.34.3.18	MPIN_BLS381_RANDOM_GENERATE()	399
8.34.3.19	MPIN_BLS381_RECOMBINE_G1()	399
8.34.3.20	MPIN_BLS381_RECOMBINE_G2()	400
8.34.3.21	MPIN_BLS381_RESTORE_FACTOR()	400
8.34.3.22	MPIN_BLS381_SERVER()	401
8.34.3.23	MPIN_BLS381_SERVER_1()	401
8.34.3.24	MPIN_BLS381_SERVER_2()	402
8.34.3.25	MPIN_BLS381_SERVER_KEY()	403
8.35	pair_BLS381.h File Reference	403
8.35.1	Detailed Description	404
8.35.2	Function Documentation	404
8.35.2.1	PAIR_BLS381_another()	404
8.35.2.2	PAIR_BLS381_ate()	405
8.35.2.3	PAIR_BLS381_double_ate()	405
8.35.2.4	PAIR_BLS381_fexp()	405
8.35.2.5	PAIR_BLS381_G1mul()	407
8.35.2.6	PAIR_BLS381_G2mul()	407
8.35.2.7	PAIR_BLS381_GTmember()	407
8.35.2.8	PAIR_BLS381_GTpow()	408

8.35.2.9 PAIR_BLS381_initmp()	408
8.35.2.10 PAIR_BLS381_miller()	408
8.35.2.11 PAIR_BLS381_nbits()	408
8.35.3 Variable Documentation	409
8.35.3.1 CURVE_BB_BLS381	409
8.35.3.2 CURVE_Bnx_BLS381	409
8.35.3.3 CURVE_Cru_BLS381	409
8.35.3.4 CURVE_SB_BLS381	409
8.35.3.5 CURVE_W_BLS381	409
8.35.3.6 CURVE_WB_BLS381	410
8.36 pbc_support.h File Reference	410
8.36.1 Detailed Description	410
8.36.2 Macro Definition Documentation	410
8.36.2.1 TIME_SLOT_MINUTES	411
8.36.3 Function Documentation	411
8.36.3.1 AES_GCM_DECRYPT()	411
8.36.3.2 AES_GCM_ENCRYPT()	411
8.36.3.3 GET_TIME()	412
8.36.3.4 HASH_ALL()	412
8.36.3.5 HASH_ID()	412
8.36.3.6 mhashit()	413
8.36.3.7 today()	413
8.37 randapi.h File Reference	413
8.37.1 Detailed Description	414
8.37.2 Function Documentation	414
8.37.2.1 CREATE_CSPRNG()	414
8.37.2.2 KILL_CSPRNG()	414
8.38 rsa_2048.h File Reference	414
8.38.1 Detailed Description	415
8.38.2 Macro Definition Documentation	415

8.38.2.1	<a href="#">HASH_TYPE_RSA_2048</a>	415
8.38.2.2	<a href="#">RFS_2048</a>	415
8.38.3	<a href="#">Function Documentation</a>	415
8.38.3.1	<a href="#">RSA_2048_DECRYPT()</a>	415
8.38.3.2	<a href="#">RSA_2048_ENCRYPT()</a>	416
8.38.3.3	<a href="#">RSA_2048_fromOctet()</a>	416
8.38.3.4	<a href="#">RSA_2048_KEY_PAIR()</a>	416
8.38.3.5	<a href="#">RSA_2048_PRIVATE_KEY_KILL()</a>	417
8.39	<a href="#">rsa_3072.h File Reference</a>	417
8.39.1	<a href="#">Detailed Description</a>	418
8.39.2	<a href="#">Macro Definition Documentation</a>	418
8.39.2.1	<a href="#">HASH_TYPE_RSA_3072</a>	418
8.39.2.2	<a href="#">RFS_3072</a>	418
8.39.3	<a href="#">Function Documentation</a>	418
8.39.3.1	<a href="#">RSA_3072_DECRYPT()</a>	418
8.39.3.2	<a href="#">RSA_3072_ENCRYPT()</a>	419
8.39.3.3	<a href="#">RSA_3072_fromOctet()</a>	419
8.39.3.4	<a href="#">RSA_3072_KEY_PAIR()</a>	419
8.39.3.5	<a href="#">RSA_3072_PRIVATE_KEY_KILL()</a>	420
8.40	<a href="#">rsa_support.h File Reference</a>	420
8.40.1	<a href="#">Detailed Description</a>	420
8.40.2	<a href="#">Macro Definition Documentation</a>	421
8.40.2.1	<a href="#">MAX_RSA_BYTES</a>	421
8.40.3	<a href="#">Function Documentation</a>	421
8.40.3.1	<a href="#">OAEP_DECODE()</a>	421
8.40.3.2	<a href="#">OAEP_ENCODE()</a>	421
8.40.3.3	<a href="#">PKCS15()</a>	422
8.41	<a href="#">utils.c File Reference</a>	422
8.41.1	<a href="#">Detailed Description</a>	423
8.41.2	<a href="#">Function Documentation</a>	423

8.41.2.1	<a href="#">amcl_bin2hex()</a>	423
8.41.2.2	<a href="#">amcl_hex2bin()</a>	424
8.41.2.3	<a href="#">amcl_print_hex()</a>	424
8.41.2.4	<a href="#">generateOTP()</a>	424
8.41.2.5	<a href="#">generateRandom()</a>	425
8.42	<a href="#">utils.h File Reference</a>	425
8.42.1	<a href="#">Detailed Description</a>	425
8.42.2	<a href="#">Function Documentation</a>	425
8.42.2.1	<a href="#">amcl_bin2hex()</a>	425
8.42.2.2	<a href="#">amcl_hex2bin()</a>	426
8.42.2.3	<a href="#">amcl_print_hex()</a>	426
8.42.2.4	<a href="#">generateOTP()</a>	426
8.42.2.5	<a href="#">generateRandom()</a>	427
8.43	<a href="#">version.c File Reference</a>	427
8.43.1	<a href="#">Detailed Description</a>	427
8.43.2	<a href="#">Function Documentation</a>	428
8.43.2.1	<a href="#">amcl_version()</a>	428
8.44	<a href="#">wcc_BLS381.h File Reference</a>	428
8.44.1	<a href="#">Detailed Description</a>	429
8.44.2	<a href="#">Macro Definition Documentation</a>	429
8.44.2.1	<a href="#">PIV</a>	429
8.44.2.2	<a href="#">PTAG</a>	429
8.44.2.3	<a href="#">TIME_SLOT_MINUTES</a>	429
8.44.2.4	<a href="#">WCC_INVALID_POINT</a>	430
8.44.2.5	<a href="#">WCC_OK</a>	430
8.44.2.6	<a href="#">WCC_PFS_BLS381</a>	430
8.44.2.7	<a href="#">WCC_PGS_BLS381</a>	430
8.44.3	<a href="#">Function Documentation</a>	430
8.44.3.1	<a href="#">WCC_BLS381_GET_G1_MULTIPLE()</a>	430
8.44.3.2	<a href="#">WCC_BLS381_GET_G2_MULTIPLE()</a>	431

8.44.3.3	WCC_BLS381_Hq()	431
8.44.3.4	WCC_BLS381_RANDOM_GENERATE()	432
8.44.3.5	WCC_BLS381_RECEIVER_KEY()	432
8.44.3.6	WCC_BLS381_RECOMBINE_G1()	433
8.44.3.7	WCC_BLS381_RECOMBINE_G2()	433
8.44.3.8	WCC_BLS381_SENDER_KEY()	434
8.45	x509.h File Reference	434
8.45.1	Detailed Description	435
8.45.2	Function Documentation	435
8.45.2.1	X509_extract_cert()	435
8.45.2.2	X509_extract_cert_sig()	435
8.45.2.3	X509_extract_public_key()	436
8.45.2.4	X509_find_entity_property()	436
8.45.2.5	X509_find_expiry_date()	437
8.45.2.6	X509_find_issuer()	437
8.45.2.7	X509_find_start_date()	437
8.45.2.8	X509_find_subject()	438
8.45.2.9	X509_find_validity()	438
<b>Index</b>		<b>439</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" ../..
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" -DWORD\_LENGTH=32 ../..

## 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:

<a href="#">amcl_aes</a>	AES instance . . . . .	15
<a href="#">csprng</a>	Cryptographically secure pseudo-random number generator instance . . . . .	16
<a href="#">ECP2_BLS381</a>	ECP2 Structure - Elliptic Curve Point over quadratic extension field . . . . .	17
<a href="#">ECP_BLS381</a>	ECP structure - Elliptic Curve Point over base field . . . . .	18
<a href="#">ECP_ED25519</a>	ECP structure - Elliptic Curve Point over base field . . . . .	19
<a href="#">ECP_GOLDILOCKS</a>	ECP structure - Elliptic Curve Point over base field . . . . .	20
<a href="#">ECP_NIST256</a>	ECP structure - Elliptic Curve Point over base field . . . . .	21
<a href="#">FP12_BLS381</a>	FP12 Structure - towered over three FP4 . . . . .	22
<a href="#">FP2_BLS381</a>	FP2 Structure - quadratic extension field . . . . .	23
<a href="#">FP4_BLS381</a>	FP4 Structure - towered over two FP2 . . . . .	23
<a href="#">FP_25519</a>	FP Structure - quadratic extension field . . . . .	24
<a href="#">FP_BLS381</a>	FP Structure - quadratic extension field . . . . .	25
<a href="#">FP_GOLDILOCKS</a>	FP Structure - quadratic extension field . . . . .	25
<a href="#">FP_NIST256</a>	FP Structure - quadratic extension field . . . . .	26
<a href="#">gcm</a>	GCM mode instance, using AES internally . . . . .	27
<a href="#">hash256</a>	SHA256 hash function instance . . . . .	28
<a href="#">hash512</a>	SHA384-512 hash function instance . . . . .	29
<a href="#">octet</a>	Portable representation of a big positive number . . . . .	30

<a href="#">pktype</a>	Public key type . . . . .	31
<a href="#">rsa_private_key_2048</a>	Integer Factorisation Private Key . . . . .	32
<a href="#">rsa_private_key_3072</a>	Integer Factorisation Private Key . . . . .	33
<a href="#">rsa_public_key_2048</a>	Integer Factorisation Public Key . . . . .	34
<a href="#">rsa_public_key_3072</a>	Integer Factorisation Public Key . . . . .	35
<a href="#">sha3</a>	SHA3 hash function instance . . . . .	35



## Chapter 6

# File Index

### 6.1 File List

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

<b>amcl.h</b>	..	??
<a href="#">arch.h</a>	Architecture Header File	37
<a href="#">big_1024_58.h</a>	BIG Header File	39
<a href="#">big_256_56.h</a>	BIG Header File	67
<a href="#">big_384_56.h</a>	BIG Header File	94
<a href="#">big_384_58.h</a>	BIG Header File	122
<a href="#">big_448_58.h</a>	BIG Header File	150
<a href="#">bls_BLS381.h</a>	BLS Header file	178
<a href="#">config_big_1024_58.h</a>	Config BIG Header File	182
<a href="#">config_big_256_56.h</a>	Config BIG Header File	183
<a href="#">config_big_384_56.h</a>	Config BIG Header File	184
<a href="#">config_big_384_58.h</a>	Config BIG Header File	184
<a href="#">config_big_448_58.h</a>	Config BIG Header File	185
<b>config_curve_BLS381.h</b>		??
<b>config_curve_ED25519.h</b>		??
<b>config_curve_GOLDILOCKS.h</b>		??
<b>config_curve_NIST256.h</b>		??
<a href="#">config_ff_2048.h</a>	COnfig FF Header File	186
<a href="#">config_ff_3072.h</a>	COnfig FF Header File	187
<b>config_field_25519.h</b>		??
<b>config_field_BLS381.h</b>		??

<b>config_field_GOLDILOCKS.h</b>	??
<b>config_field_NIST256.h</b>	??
<b>config_test.h</b>	??
<a href="#">ecdh_BLS381.h</a>	
ECDH Header file for implementation of standard EC protocols	187
<a href="#">ecdh_ED25519.h</a>	
ECDH Header file for implementation of standard EC protocols	192
<a href="#">ecdh_GOLDILOCKS.h</a>	
ECDH Header file for implementation of standard EC protocols	197
<a href="#">ecdh_NIST256.h</a>	
ECDH Header file for implementation of standard EC protocols	202
<a href="#">ecdh_support.h</a>	
ECDH Support Header File	207
<a href="#">ecp2_BLS381.h</a>	
ECP2 Header File	211
<a href="#">ecp_BLS381.h</a>	
ECP Header File	221
<a href="#">ecp_ED25519.h</a>	
ECP Header File	237
<a href="#">ecp_GOLDILOCKS.h</a>	
ECP Header File	253
<a href="#">ecp_NIST256.h</a>	
ECP Header File	268
<a href="#">ff_2048.h</a>	
FF Header File	283
<a href="#">ff_3072.h</a>	
FF Header File	297
<a href="#">fp12_BLS381.h</a>	
FP12 Header File	311
<a href="#">fp2_BLS381.h</a>	
FP2 Header File	321
<a href="#">fp4_BLS381.h</a>	
FP4 Header File	331
<a href="#">fp_25519.h</a>	
FP Header File	345
<a href="#">fp_BLS381.h</a>	
FP Header File	355
<a href="#">fp_GOLDILOCKS.h</a>	
FP Header File	366
<a href="#">fp_NIST256.h</a>	
FP Header File	376
<a href="#">mpin_BLS381.h</a>	
M-Pin Header file	387
<a href="#">pair_BLS381.h</a>	
PAIR Header File	403
<a href="#">pbc_support.h</a>	
Auxiliary functions for Pairing-based protocols	410
<a href="#">randapi.h</a>	
PRNG API File	413
<a href="#">rsa_2048.h</a>	
RSA Header file for implementation of RSA protocol	414
<a href="#">rsa_3072.h</a>	
RSA Header file for implementation of RSA protocol	417
<a href="#">rsa_support.h</a>	
RSA Support Header File	420
<a href="#">utils.c</a>	
AMCL Support functions for M-Pin servers	422

<a href="#">utils.h</a>	Utility functions Header File . . . . .	425
<a href="#">version.c</a>	AMCL version support function . . . . .	427
<b>version.h</b>	. . . . .	??
<a href="#">wcc_BLS381.h</a>	WCC Header File . . . . .	428
<a href="#">x509.h</a>	X509 function Header File . . . . .	434



## 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

```
unsigned 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

- [unsigned ira](#) [NK]
- [int rndptr](#)
- [unsigned 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 ECP\_ED25519 Struct Reference

ECP structure - Elliptic Curve Point over base field.

```
#include <ecp_ED25519.h>
```

### Data Fields

- [FP\\_25519](#) x
- [FP\\_25519](#) y
- [FP\\_25519](#) z

### 7.5.1 Field Documentation

#### 7.5.1.1 x

[FP\\_25519](#) ECP\_ED25519::x

x-coordinate of point

### 7.5.1.2 y

[FP\\_25519](#) `ECP_ED25519::y`

y-coordinate of point. Not needed for Montgomery representation

### 7.5.1.3 z

[FP\\_25519](#) `ECP_ED25519::z`

z-coordinate of point

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

- [ecp\\_ED25519.h](#)

## 7.6 ECP\_GOLDILOCKS Struct Reference

ECP structure - Elliptic Curve Point over base field.

```
#include <ecp_GOLDILOCKS.h>
```

### Data Fields

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

### 7.6.1 Field Documentation

#### 7.6.1.1 x

[FP\\_GOLDILOCKS](#) `ECP_GOLDILOCKS::x`

x-coordinate of point

#### 7.6.1.2 y

[FP\\_GOLDILOCKS](#) `ECP_GOLDILOCKS::y`

y-coordinate of point. Not needed for Montgomery representation

### 7.6.1.3 z

`FP_GOLDILOCKS ECP_GOLDILOCKS::z`

z-coordinate of point

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

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

## 7.7 ECP\_NIST256 Struct Reference

ECP structure - Elliptic Curve Point over base field.

```
#include <ecp_NIST256.h>
```

### Data Fields

- [FP\\_NIST256 x](#)
- [FP\\_NIST256 y](#)
- [FP\\_NIST256 z](#)

### 7.7.1 Field Documentation

#### 7.7.1.1 x

`FP_NIST256 ECP_NIST256::x`

x-coordinate of point

#### 7.7.1.2 y

`FP_NIST256 ECP_NIST256::y`

y-coordinate of point. Not needed for Montgomery representation

#### 7.7.1.3 z

`FP_NIST256 ECP_NIST256::z`

z-coordinate of point

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

- [ecp\\_NIST256.h](#)

## 7.8 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](#)

### 7.8.1 Field Documentation

#### 7.8.1.1 a

```
FP4\_BLS381 FP12_BLS381::a
```

first part of FP12

#### 7.8.1.2 b

```
FP4\_BLS381 FP12_BLS381::b
```

second part of FP12

#### 7.8.1.3 c

```
FP4\_BLS381 FP12_BLS381::c
```

third part of FP12

#### 7.8.1.4 type

```
int FP12_BLS381::type
```

Type

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

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

## 7.9 FP2\_BLS381 Struct Reference

FP2 Structure - quadratic extension field.

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

### Data Fields

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

### 7.9.1 Field Documentation

#### 7.9.1.1 a

```
FP_BLS381 FP2_BLS381::a
```

real part of FP2

#### 7.9.1.2 b

```
FP_BLS381 FP2_BLS381::b
```

imaginary part of FP2

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

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

## 7.10 FP4\_BLS381 Struct Reference

FP4 Structure - towered over two FP2.

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

### Data Fields

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

### 7.10.1 Field Documentation

#### 7.10.1.1 a

[FP2\\_BLS381](#) FP4\_BLS381::a

real part of FP4

#### 7.10.1.2 b

[FP2\\_BLS381](#) FP4\_BLS381::b

imaginary part of FP4

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

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

## 7.11 FP\_25519 Struct Reference

FP Structure - quadratic extension field.

```
#include <fp_25519.h>
```

### Data Fields

- [BIG\\_256\\_56](#) g
- [sign32](#) XES

#### 7.11.1 Field Documentation

##### 7.11.1.1 g

[BIG\\_256\\_56](#) FP\_25519::g

Big representation of field element

##### 7.11.1.2 XES

[sign32](#) FP\_25519::XES

Excess

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

- [fp\\_25519.h](#)

## 7.12 FP\_BLS381 Struct Reference

FP Structure - quadratic extension field.

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

### Data Fields

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

### 7.12.1 Field Documentation

#### 7.12.1.1 g

```
BIG_384_58 FP_BLS381::g
```

Big representation of field element

#### 7.12.1.2 XES

```
sign32 FP_BLS381::XES
```

Excess

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

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

## 7.13 FP\_GOLDILOCKS Struct Reference

FP Structure - quadratic extension field.

```
#include <fp_GOLDILOCKS.h>
```

### Data Fields

- [BIG\\_448\\_58 g](#)
- [sign32 XES](#)

### 7.13.1 Field Documentation

#### 7.13.1.1 g

[BIG\\_448\\_58](#) FP\_GOLDILOCKS::g

Big representation of field element

#### 7.13.1.2 XES

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

Excess

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

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

### 7.14 FP\_NIST256 Struct Reference

FP Structure - quadratic extension field.

```
#include <fp_NIST256.h>
```

#### Data Fields

- [BIG\\_256\\_56](#) g
- [sign32](#) XES

#### 7.14.1 Field Documentation

##### 7.14.1.1 g

[BIG\\_256\\_56](#) FP\_NIST256::g

Big representation of field element

##### 7.14.1.2 XES

[sign32](#) FP\_NIST256::XES

Excess

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

- [fp\\_NIST256.h](#)



## 7.15 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`

### 7.15.1 Field Documentation

#### 7.15.1.1 `a`

```
amcl_aes gcm::a
```

Internal Instance of AMCL\_AES cipher

#### 7.15.1.2 `lenA`

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

GCM 64-bit length of header

#### 7.15.1.3 `lenC`

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

GCM 64-bit length of ciphertext

#### 7.15.1.4 `stateX`

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

GCM Internal State

#### 7.15.1.5 status

```
int gcm::status
```

GCM Status

#### 7.15.1.6 table

```
unsigned32 gcm::table[128][4]
```

2k byte table

#### 7.15.1.7 Y\_0

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

GCM Internal State

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

- amcl.h

## 7.16 hash256 Struct Reference

SHA256 hash function instance.

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

### Data Fields

- [unsigned32 length](#) [2]
- [unsigned32 h](#) [8]
- [unsigned32 w](#) [80]
- [int hlen](#)

### 7.16.1 Field Documentation

#### 7.16.1.1 h

```
unsigned32 hash256::h[8]
```

Internal state

#### 7.16.1.2 hlen

```
int hash256::hlen
```

Hash length in bytes

#### 7.16.1.3 length

```
unsigned32 hash256::length[2]
```

64-bit input length

#### 7.16.1.4 w

```
unsigned32 hash256::w[80]
```

Internal state

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

- amcl.h

## 7.17 hash512 Struct Reference

SHA384-512 hash function instance.

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

### Data Fields

- [unsigned64 length](#) [2]
- [unsigned64 h](#) [8]
- [unsigned64 w](#) [80]
- [int hlen](#)

### 7.17.1 Field Documentation

#### 7.17.1.1 h

```
unsigned64 hash512::h[8]
```

Internal state

#### 7.17.1.2 hlen

```
int hash512::hlen
```

Hash length in bytes

#### 7.17.1.3 length

```
unsigned64 hash512::length[2]
```

64-bit input length

#### 7.17.1.4 w

```
unsigned64 hash512::w[80]
```

Internal state

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

- amcl.h

## 7.18 octet Struct Reference

Portable representation of a big positive number.

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

### Data Fields

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

### 7.18.1 Field Documentation

#### 7.18.1.1 len

```
int octet::len
```

length in bytes

### 7.18.1.2 max

```
int  octet::max
```

max length allowed - enforce truncation

### 7.18.1.3 val

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

byte array

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

- amcl.h

## 7.19 pktype Struct Reference

Public key type.

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

### Data Fields

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

### 7.19.1 Field Documentation

#### 7.19.1.1 curve

```
int  pktype::curve
```

elliptic curve used or RSA key length in bits

#### 7.19.1.2 hash

```
int  pktype::hash
```

hash type

### 7.19.1.3 type

```
int pkttype::type
```

signature type (ECC or RSA)

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

- [x509.h](#)

## 7.20 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](#)]

### 7.20.1 Field Documentation

#### 7.20.1.1 c

```
BIG\_1024\_58 rsa_private_key_2048::c [FFLEN\_2048/2]
```

$1/p \bmod q$

#### 7.20.1.2 dp

```
BIG\_1024\_58 rsa_private_key_2048::dp [FFLEN\_2048/2]
```

decrypting exponent mod (p-1)

#### 7.20.1.3 dq

```
BIG\_1024\_58 rsa_private_key_2048::dq [FFLEN\_2048/2]
```

decrypting exponent mod (q-1)

#### 7.20.1.4 `p`

[BIG\\_1024\\_58](#) `rsa_private_key_2048::p` [[FFLEN\\_2048](#)/2]

secret prime `p`

#### 7.20.1.5 `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](#)

## 7.21 `rsa_private_key_3072` Struct Reference

Integer Factorisation Private Key.

```
#include <rsa_3072.h>
```

### Data Fields

- [BIG\\_384\\_56](#) `p` [[FFLEN\\_3072](#)/2]
- [BIG\\_384\\_56](#) `q` [[FFLEN\\_3072](#)/2]
- [BIG\\_384\\_56](#) `dp` [[FFLEN\\_3072](#)/2]
- [BIG\\_384\\_56](#) `dq` [[FFLEN\\_3072](#)/2]
- [BIG\\_384\\_56](#) `c` [[FFLEN\\_3072](#)/2]

### 7.21.1 Field Documentation

#### 7.21.1.1 `c`

[BIG\\_384\\_56](#) `rsa_private_key_3072::c` [[FFLEN\\_3072](#)/2]

$1/p \bmod q$

#### 7.21.1.2 `dp`

[BIG\\_384\\_56](#) `rsa_private_key_3072::dp` [[FFLEN\\_3072](#)/2]

decrypting exponent mod  $(p-1)$

## 7.21.1.3 dq

[BIG\\_384\\_56](#) rsa\_private\_key\_3072::dq[[FFLEN\\_3072](#)/2]

decrypting exponent mod (q-1)

## 7.21.1.4 p

[BIG\\_384\\_56](#) rsa\_private\_key\_3072::p[[FFLEN\\_3072](#)/2]

secret prime p

## 7.21.1.5 q

[BIG\\_384\\_56](#) rsa\_private\_key\_3072::q[[FFLEN\\_3072](#)/2]

secret prime q

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

- [rsa\\_3072.h](#)

## 7.22 rsa\_public\_key\_2048 Struct Reference

Integer Factorisation Public Key.

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

### Data Fields

- [sign32](#) e
- [BIG\\_1024\\_58](#) n [[FFLEN\\_2048](#)]

### 7.22.1 Field Documentation

## 7.22.1.1 e

[sign32](#) rsa\_public\_key\_2048::e

RSA exponent (typically 65537)



7.22.1.2 `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](#)

## 7.23 `rsa_public_key_3072` Struct Reference

Integer Factorisation Public Key.

```
#include <rsa_3072.h>
```

### Data Fields

- [sign32 e](#)
- [BIG\\_384\\_56 n](#) [FFLEN\_3072]

### 7.23.1 Field Documentation

7.23.1.1 `e`

```
sign32 rsa_public_key_3072::e
```

RSA exponent (typically 65537)

7.23.1.2 `n`

```
BIG_384_56 rsa_public_key_3072::n[FFLEN_3072]
```

An array of BIGs to store public key

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

- [rsa\\_3072.h](#)

## 7.24 `sha3` Struct Reference

SHA3 hash function instance.

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

## Data Fields

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

### 7.24.1 Field Documentation

#### 7.24.1.1 len

```
int sha3::len
```

Hash length in bytes

#### 7.24.1.2 length

```
unsign64 sha3::length
```

64-bit input length

#### 7.24.1.3 rate

```
int sha3::rate
```

TODO

#### 7.24.1.4 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\\_sdcopy](#) ([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\\_modsqr](#) (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**

<i>x</i>	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**

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

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

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

**Truncation****Parameters**

<i>x</i>	DBIG number, on reduced mod $2^m$
<i>m</i>	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,
    DBIG_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 = $1/y \bmod n$
$y$	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

## 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

<i>x</i>	BIG number, on exit a random number
<i>r</i>	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

<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

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

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

## Parameters

<i>x</i>	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

<i>x</i>	BIG number
<i>y</i>	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\_256\_56.h File Reference

BIG Header File.

```
#include <stdio.h>  
#include <stdlib.h>  
#include <inttypes.h>  
#include "arch.h"  
#include "amcl.h"  
#include "config_big_256_56.h"
```

## Macros

- `#define BIGBITS_256_56 (8*MODBYTES_256_56)`
- `#define NLEN_256_56 (1+((8*MODBYTES_256_56-1)/BASEBITS_256_56))`
- `#define DNLEN_256_56 2*NLEN_256_56`
- `#define BMASK_256_56 (((chunk)1<<BASEBITS_256_56)-1)`
- `#define NEXCESS_256_56 (1<<(CHUNK-BASEBITS_256_56-1))`
- `#define HBITS_256_56 (BASEBITS_256_56/2)`
- `#define HMASK_256_56 (((chunk)1<<HBITS_256_56)-1)`

## Typedefs

- `typedef chunk BIG_256_56[NLEN_256_56]`
- `typedef chunk DBIG_256_56[DNLEN_256_56]`

## Functions

- `int BIG_256_56_iszilch (BIG_256_56 x)`  
*Tests for BIG equal to zero.*
- `int BIG_256_56_isunity (BIG_256_56 x)`  
*Tests for BIG equal to one.*
- `int BIG_256_56_diszilch (DBIG_256_56 x)`  
*Tests for DBIG equal to zero.*
- `void BIG_256_56_output (BIG_256_56 x)`  
*Outputs a BIG number to the console.*
- `void BIG_256_56_rawoutput (BIG_256_56 x)`  
*Outputs a BIG number to the console in raw form (for debugging)*
- `void BIG_256_56_cswap (BIG_256_56 x, BIG_256_56 y, int s)`  
*Conditional constant time swap of two BIG numbers.*
- `void BIG_256_56_cmove (BIG_256_56 x, BIG_256_56 y, int s)`  
*Conditional copy of BIG number.*
- `void BIG_256_56_dcmove (BIG_256_56 x, BIG_256_56 y, int s)`  
*Conditional copy of DBIG number.*
- `void BIG_256_56_toBytes (char *a, BIG_256_56 x)`  
*Convert from BIG number to byte array.*
- `void BIG_256_56_fromBytes (BIG_256_56 x, char *a)`  
*Convert to BIG number from byte array.*
- `void BIG_256_56_fromBytesLen (BIG_256_56 x, char *a, int s)`  
*Convert to BIG number from byte array of given length.*
- `void BIG_256_56_dfromBytesLen (DBIG_256_56 x, char *a, int s)`  
*Convert to DBIG number from byte array of given length.*
- `void BIG_256_56_doutput (DBIG_256_56 x)`  
*Outputs a DBIG number to the console.*
- `void BIG_256_56_drawoutput (DBIG_256_56 x)`  
*Outputs a DBIG number to the console.*
- `void BIG_256_56_rcopy (BIG_256_56 x, const BIG_256_56 y)`  
*Copy BIG from Read-Only Memory to a BIG.*
- `void BIG_256_56_copy (BIG_256_56 x, BIG_256_56 y)`  
*Copy BIG to another BIG.*
- `void BIG_256_56_dcopy (DBIG_256_56 x, DBIG_256_56 y)`

- Copy DBIG to another DBIG.*

  - void [BIG\\_256\\_56\\_dsucopy](#) (DBIG\_256\_56 x, BIG\_256\_56 y)
- Copy BIG to upper half of DBIG.*

  - void [BIG\\_256\\_56\\_dscopy](#) (DBIG\_256\_56 x, BIG\_256\_56 y)
- Copy BIG to lower half of DBIG.*

  - void [BIG\\_256\\_56\\_sdcopy](#) (BIG\_256\_56 x, DBIG\_256\_56 y)
- Copy lower half of DBIG to a BIG.*

  - void [BIG\\_256\\_56\\_sducopy](#) (BIG\_256\_56 x, DBIG\_256\_56 y)
- Copy upper half of DBIG to a BIG.*

  - void [BIG\\_256\\_56\\_zero](#) (BIG\_256\_56 x)
- Set BIG to zero.*

  - void [BIG\\_256\\_56\\_dzero](#) (DBIG\_256\_56 x)
- Set DBIG to zero.*

  - void [BIG\\_256\\_56\\_one](#) (BIG\_256\_56 x)
- Set BIG to one (unity)*

  - void [BIG\\_256\\_56\\_invmod2m](#) (BIG\_256\_56 x)
- Set BIG to inverse mod  $2^{256}$ .*

  - void [BIG\\_256\\_56\\_add](#) (BIG\_256\_56 x, BIG\_256\_56 y, BIG\_256\_56 z)
- Set BIG to sum of two BIGs - output not normalised.*

  - void [BIG\\_256\\_56\\_or](#) (BIG\_256\_56 x, BIG\_256\_56 y, BIG\_256\_56 z)
- Set BIG to logical or of two BIGs - output normalised.*

  - void [BIG\\_256\\_56\\_inc](#) (BIG\_256\_56 x, int i)
- Increment BIG by a small integer - output not normalised.*

  - void [BIG\\_256\\_56\\_sub](#) (BIG\_256\_56 x, BIG\_256\_56 y, BIG\_256\_56 z)
- Set BIG to difference of two BIGs.*

  - void [BIG\\_256\\_56\\_dec](#) (BIG\_256\_56 x, int i)
- Decrement BIG by a small integer - output not normalised.*

  - void [BIG\\_256\\_56\\_dadd](#) (DBIG\_256\_56 x, DBIG\_256\_56 y, DBIG\_256\_56 z)
- Set DBIG to sum of two DBIGs.*

  - void [BIG\\_256\\_56\\_dsub](#) (DBIG\_256\_56 x, DBIG\_256\_56 y, DBIG\_256\_56 z)
- Set DBIG to difference of two DBIGs.*

  - void [BIG\\_256\\_56\\_imul](#) (BIG\_256\_56 x, BIG\_256\_56 y, int i)
- Multiply BIG by a small integer - output not normalised.*

  - [chunk](#) [BIG\\_256\\_56\\_pmul](#) (BIG\_256\_56 x, BIG\_256\_56 y, int i)
- Multiply BIG by not-so-small small integer - output normalised.*

  - int [BIG\\_256\\_56\\_div3](#) (BIG\_256\_56 x)
- Divide BIG by 3 - output normalised.*

  - void [BIG\\_256\\_56\\_pxmud](#) (DBIG\_256\_56 x, BIG\_256\_56 y, int i)
- Multiply BIG by even bigger small integer resulting in a DBIG - output normalised.*

  - void [BIG\\_256\\_56\\_mul](#) (DBIG\_256\_56 x, BIG\_256\_56 y, BIG\_256\_56 z)
- Multiply BIG by another BIG resulting in DBIG - inputs normalised and output normalised.*

  - void [BIG\\_256\\_56\\_smul](#) (BIG\_256\_56 x, BIG\_256\_56 y, BIG\_256\_56 z)
- Multiply BIG by another BIG resulting in another BIG - inputs normalised and output normalised.*

  - void [BIG\\_256\\_56\\_sqr](#) (DBIG\_256\_56 x, BIG\_256\_56 y)
- Square BIG resulting in a DBIG - input normalised and output normalised.*

  - void [BIG\\_256\\_56\\_monty](#) (BIG\_256\_56 a, BIG\_256\_56 md, [chunk](#) MC, DBIG\_256\_56 d)
- Montgomery reduction of a DBIG to a BIG - input normalised and output normalised.*

  - void [BIG\\_256\\_56\\_shl](#) (BIG\_256\_56 x, int s)
- Shifts a BIG left by any number of bits - input must be normalised, output normalised.*

  - int [BIG\\_256\\_56\\_fshl](#) (BIG\_256\_56 x, int s)
- Fast shifts a BIG left by a small number of bits - input must be normalised, output will be normalised.*

- void `BIG_256_56_dshl` (`DBIG_256_56` x, int s)  
*Shifts a DBIG left by any number of bits - input must be normalised, output normalised.*
- void `BIG_256_56_shr` (`BIG_256_56` x, int s)  
*Shifts a BIG right by any number of bits - input must be normalised, output normalised.*
- int `BIG_256_56_ssn` (`BIG_256_56` r, `BIG_256_56` a, `BIG_256_56` m)  
*Fast time-critical combined shift by 1 bit, subtract and normalise.*
- int `BIG_256_56_fshr` (`BIG_256_56` x, int s)  
*Fast shifts a BIG right by a small number of bits - input must be normalised, output will be normalised.*
- void `BIG_256_56_dshr` (`DBIG_256_56` x, int s)  
*Shifts a DBIG right by any number of bits - input must be normalised, output normalised.*
- `chunk` `BIG_256_56_split` (`BIG_256_56` x, `BIG_256_56` y, `DBIG_256_56` z, int s)  
*Splits a DBIG into two BIGs - input must be normalised, outputs normalised.*
- `chunk` `BIG_256_56_norm` (`BIG_256_56` x)  
*Normalizes a BIG number - output normalised.*
- void `BIG_256_56_dnorm` (`DBIG_256_56` x)  
*Normalizes a DBIG number - output normalised.*
- int `BIG_256_56_comp` (`BIG_256_56` x, `BIG_256_56` y)  
*Compares two BIG numbers. Inputs must be normalised externally.*
- int `BIG_256_56_dcomp` (`DBIG_256_56` x, `DBIG_256_56` y)  
*Compares two DBIG numbers. Inputs must be normalised externally.*
- int `BIG_256_56_nbits` (`BIG_256_56` x)  
*Calculate number of bits in a BIG - output normalised.*
- int `BIG_256_56_dnbits` (`DBIG_256_56` x)  
*Calculate number of bits in a DBIG - output normalised.*
- void `BIG_256_56_mod` (`BIG_256_56` x, `BIG_256_56` n)  
*Reduce x mod n - input and output normalised.*
- void `BIG_256_56_sdiv` (`BIG_256_56` x, `BIG_256_56` n)  
*Divide x by n - output normalised.*
- void `BIG_256_56_dmod` (`BIG_256_56` x, `DBIG_256_56` y, `BIG_256_56` n)  
 *$x=y \bmod n$  - output normalised*
- void `BIG_256_56_ddiv` (`BIG_256_56` x, `DBIG_256_56` y, `BIG_256_56` n)  
 *$x=y/n$  - output normalised*
- int `BIG_256_56_parity` (`BIG_256_56` x)  
*return parity of BIG, that is the least significant bit*
- int `BIG_256_56_bit` (`BIG_256_56` x, int i)  
*return i-th of BIG*
- int `BIG_256_56_lastbits` (`BIG_256_56` x, int n)  
*return least significant bits of a BIG*
- void `BIG_256_56_random` (`BIG_256_56` x, `csprng` \*r)  
*Create a random BIG from a random number generator.*
- void `BIG_256_56_randomnum` (`BIG_256_56` x, `BIG_256_56` n, `csprng` \*r)  
*Create an unbiased random BIG from a random number generator, reduced with respect to a modulus.*
- void `BIG_256_56_modmul` (`BIG_256_56` x, `BIG_256_56` y, `BIG_256_56` z, `BIG_256_56` n)  
*Calculate  $x=y*z \bmod n$ .*
- void `BIG_256_56_moddiv` (`BIG_256_56` x, `BIG_256_56` y, `BIG_256_56` z, `BIG_256_56` n)  
*Calculate  $x=y/z \bmod n$ .*
- void `BIG_256_56_modsq` (`BIG_256_56` x, `BIG_256_56` y, `BIG_256_56` n)  
*Calculate  $x=y^2 \bmod n$ .*
- void `BIG_256_56_modneg` (`BIG_256_56` x, `BIG_256_56` y, `BIG_256_56` n)  
*Calculate  $x=-y \bmod n$ .*
- int `BIG_256_56_jacobi` (`BIG_256_56` x, `BIG_256_56` y)

*Calculate jacobi Symbol (x/y)*

- void [BIG\\_256\\_56\\_invmodp](#) ([BIG\\_256\\_56](#) x, [BIG\\_256\\_56](#) y, [BIG\\_256\\_56](#) n)

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

- void [BIG\\_256\\_56\\_mod2m](#) ([BIG\\_256\\_56](#) x, int m)

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

- void [BIG\\_256\\_56\\_dmod2m](#) ([BIG\\_256\\_56](#) 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\_256\_56

```
#define BIGBITS_256_56 (8*MODBYTES_256_56)
```

Length in bits

#### 8.3.2.2 BMASK\_256\_56

```
#define BMASK_256_56 (((chunk)1<<BASEBITS_256_56)-1)
```

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

#### 8.3.2.3 DNLEN\_256\_56

```
#define DNLEN_256_56 2*NLEN_256_56
```

Double length in bytes

#### 8.3.2.4 HBITS\_256\_56

```
#define HBITS_256_56 (BASEBITS_256_56/2)
```

Number of bits in number base divided by 2

#### 8.3.2.5 HMASK\_256\_56

```
#define HMASK_256_56 (((chunk)1<<HBITS_256_56)-1)
```

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

### 8.3.2.6 NEXCESS\_256\_56

```
#define NEXCESS_256_56 (1<<(CHUNK-BASEBITS_256_56-1))
```

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

### 8.3.2.7 NLEN\_256\_56

```
#define NLEN_256_56 (1+((8*MODBYTES_256_56-1)/BASEBITS_256_56))
```

length in bytes

## 8.3.3 Typedef Documentation

### 8.3.3.1 BIG\_256\_56

```
typedef chunk BIG_256_56[NLEN_256_56]
```

Define type BIG as array of chunks

### 8.3.3.2 DBIG\_256\_56

```
typedef chunk DBIG_256_56[DNLEN_256_56]
```

Define type DBIG as array of chunks

## 8.3.4 Function Documentation

### 8.3.4.1 BIG\_256\_56\_add()

```
void BIG_256_56_add (
    BIG_256_56 x,
    BIG_256_56 y,
    BIG_256_56 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\_256\_56\_bit()

```
int BIG_256_56_bit (
    BIG_256_56 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\_256\_56\_cmove()

```
void BIG_256_56_cmove (
    BIG_256_56 x,
    BIG_256_56 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\_256\_56\_comp()

```
int BIG_256_56_comp (
    BIG_256_56 x,
    BIG_256_56 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  $x > y$

**8.3.4.5 BIG\_256\_56\_copy()**

```
void BIG_256_56_copy (
    BIG_256_56 x,
    BIG_256_56 y )
```

**Parameters**

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

**8.3.4.6 BIG\_256\_56\_cswap()**

```
void BIG_256_56_cswap (
    BIG_256_56 x,
    BIG_256_56 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\_256\_56\_dadd()**

```
void BIG_256_56_dadd (
    DBIG_256_56 x,
    DBIG_256_56 y,
    DBIG_256_56 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\_256\_56\_dcmove()

```
void BIG_256_56_dcmove (
    BIG_256_56 x,
    BIG_256_56 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\_256\_56\_dcomp()

```
int BIG_256_56_dcomp (
    DBIG_256_56 x,
    DBIG_256_56 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\_256\_56\_dcopy()

```
void BIG_256_56_dcopy (
    DBIG_256_56 x,
    DBIG_256_56 y )
```

##### Parameters

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

#### 8.3.4.11 BIG\_256\_56\_ddiv()

```
void BIG_256_56_ddiv (
    BIG_256_56 x,
    DBIG_256_56 y,
    BIG_256_56 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

#### 8.3.4.12 BIG\_256\_56\_dec()

```
void BIG_256_56_dec (
    BIG_256_56 x,
    int i )
```

##### Parameters

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

#### 8.3.4.13 BIG\_256\_56\_dfromBytesLen()

```
void BIG_256_56_dfromBytesLen (
    DBIG_256_56 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\_256\_56\_diszilch()

```
int BIG_256_56_diszilch (
    DBIG_256_56 x )
```

##### Parameters

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

##### Returns

1 if zero, else returns 0

#### 8.3.4.15 BIG\_256\_56\_div3()

```
int BIG_256_56_div3 (
    BIG_256_56 x )
```

##### Parameters

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

##### Returns

Remainder

#### 8.3.4.16 BIG\_256\_56\_dmod()

```
void BIG_256_56_dmod (
    BIG_256_56 x,
    DBIG_256_56 y,
    BIG_256_56 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\_256\_56\_dmod2m()**

```
void BIG_256_56_dmod2m (
    DBIG_256_56 x,
    int m )
```

Truncation

Parameters

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

**8.3.4.18 BIG\_256\_56\_dnbits()**

```
int BIG_256_56_dnbits (
    DBIG_256_56 x )
```

Parameters

$x$	DBIG number
-----	-------------

Returns

Number of bits in  $x$

**8.3.4.19 BIG\_256\_56\_dnorm()**

```
void BIG_256_56_dnorm (
    DBIG_256_56 x )
```

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

Parameters

$x$	DBIG number to be normalised
-----	------------------------------

**8.3.4.20 BIG\_256\_56\_doutput()**

```
void BIG_256_56_doutput (
    DBIG_256_56 x )
```

## Parameters

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

## 8.3.4.21 BIG\_256\_56\_drawoutput()

```
void BIG_256_56_drawoutput (
    DBIG_256_56 x )
```

## Parameters

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

## 8.3.4.22 BIG\_256\_56\_dscopy()

```
void BIG_256_56_dscopy (
    DBIG_256_56 x,
    BIG_256_56 y )
```

## Parameters

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

## 8.3.4.23 BIG\_256\_56\_dshl()

```
void BIG_256_56_dshl (
    DBIG_256_56 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\_256\_56\_dshr()

```
void BIG_256_56_dshr (
```

```

    DBIG_256_56 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\_256\_56\_dsub()

```

void BIG_256_56_dsub (
    DBIG_256_56 x,
    DBIG_256_56 y,
    DBIG_256_56 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\_256\_56\_dsucopy()

```

void BIG_256_56_dsucopy (
    DBIG_256_56 x,
    BIG_256_56 y )

```

#### Parameters

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

#### 8.3.4.27 BIG\_256\_56\_dzero()

```

void BIG_256_56_dzero (
    DBIG_256_56 x )

```

#### Parameters

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

#### 8.3.4.28 BIG\_256\_56\_fromBytes()

```
void BIG_256_56_fromBytes (
    BIG_256_56 x,
    char * a )
```

##### Parameters

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

#### 8.3.4.29 BIG\_256\_56\_fromBytesLen()

```
void BIG_256_56_fromBytesLen (
    BIG_256_56 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\_256\_56\_fshl()

```
int BIG_256_56_fshl (
    BIG_256_56 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\_256\_56\_fshr()

```
int BIG_256_56_fshr (
    BIG_256_56 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\_256\_56\_imul()

```
void BIG_256_56_imul (
    BIG_256_56 x,
    BIG_256_56 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\_256\_56\_inc()

```
void BIG_256_56_inc (
    BIG_256_56 x,
    int i )
```

##### Parameters

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



#### 8.3.4.34 BIG\_256\_56\_invmod2m()

```
void BIG_256_56_invmod2m (
    BIG_256_56 x )
```

##### Parameters

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

#### 8.3.4.35 BIG\_256\_56\_invmodp()

```
void BIG_256_56_invmodp (
    BIG_256_56 x,
    BIG_256_56 y,
    BIG_256_56 n )
```

Modular Inversion - This is slow. Uses binary method.

##### Parameters

<i>x</i>	BIG number, on exit = 1/y mod n
<i>y</i>	BIG number
<i>n</i>	The BIG Modulus

#### 8.3.4.36 BIG\_256\_56\_isunity()

```
int BIG_256_56_isunity (
    BIG_256_56 x )
```

##### Parameters

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

##### Returns

1 if one, else returns 0

#### 8.3.4.37 BIG\_256\_56\_iszilch()

```
int BIG_256_56_iszilch (
    BIG_256_56 x )
```

**Parameters**

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

**Returns**

1 if zero, else returns 0

**8.3.4.38 BIG\_256\_56\_jacobi()**

```
int BIG_256_56_jacobi (
    BIG_256_56 x,
    BIG_256_56 y )
```

**Parameters**

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

**Returns**

Jacobi symbol, -1,0 or 1

**8.3.4.39 BIG\_256\_56\_lastbits()**

```
int BIG_256_56_lastbits (
    BIG_256_56 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\_256\_56\_mod()**

```
void BIG_256_56_mod (
    BIG_256_56 x,
    BIG_256_56 n )
```

Slow but rarely used

#### Parameters

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

#### 8.3.4.41 BIG\_256\_56\_mod2m()

```
void BIG_256_56_mod2m (
    BIG_256_56 x,
    int m )
```

Truncation

#### Parameters

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

#### 8.3.4.42 BIG\_256\_56\_moddiv()

```
void BIG_256_56_moddiv (
    BIG_256_56 x,
    BIG_256_56 y,
    BIG_256_56 z,
    BIG_256_56 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\_256\_56\_modmul()

```
void BIG_256_56_modmul (
    BIG_256_56 x,
    BIG_256_56 y,
```

```
BIG_256_56 z,
BIG_256_56 n )
```

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

Given  $x$  and  $3x$  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\_256\_56\_modneg()

```
void BIG_256_56_modneg (
    BIG_256_56 x,
    BIG_256_56 y,
    BIG_256_56 n )
```

Modular negation

#### Parameters

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

#### 8.3.4.45 BIG\_256\_56\_modsqr()

```
void BIG_256_56_modsqr (
    BIG_256_56 x,
    BIG_256_56 y,
    BIG_256_56 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\_256\_56\_monty()

```
void BIG_256_56_monty (
    BIG_256_56 a,
    BIG_256_56 md,
    chunk MC,
    DBIG_256_56 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\_256\_56\_mul()

```
void BIG_256_56_mul (
    DBIG_256_56 x,
    BIG_256_56 y,
    BIG_256_56 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\_256\_56\_nbits()

```
int BIG_256_56_nbits (
    BIG_256_56 x )
```

##### Parameters

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

##### Returns

Number of bits in x

**8.3.4.49 BIG\_256\_56\_norm()**

```
chunk BIG_256_56_norm (
    BIG_256_56 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\_256\_56\_one()**

```
void BIG_256_56_one (
    BIG_256_56 x )
```

**Parameters**

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

**8.3.4.51 BIG\_256\_56\_or()**

```
void BIG_256_56_or (
    BIG_256_56 x,
    BIG_256_56 y,
    BIG_256_56 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\_256\_56\_output()**

```
void BIG_256_56_output (
    BIG_256_56 x )
```

**Parameters**

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

#### 8.3.4.53 BIG\_256\_56\_parity()

```
int BIG_256_56_parity (
    BIG_256_56 x )
```

##### Parameters

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

##### Returns

0 or 1

#### 8.3.4.54 BIG\_256\_56\_pmul()

```
chunk BIG_256_56_pmul (
    BIG_256_56 x,
    BIG_256_56 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\_256\_56\_pxmuls()

```
void BIG_256_56_pxmuls (
    DBIG_256_56 x,
    BIG_256_56 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\_256\_56\_random()**

```
void BIG_256_56_random (
    BIG_256_56 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\_256\_56\_randomnum()**

```
void BIG_256_56_randomnum (
    BIG_256_56 x,
    BIG_256_56 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\_256\_56\_rawoutput()**

```
void BIG_256_56_rawoutput (
    BIG_256_56 x )
```

**Parameters**

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

**8.3.4.59 BIG\_256\_56\_rcopy()**

```
void BIG_256_56_rcopy (
```



```
BIG_256_56 x,  
const BIG_256_56 y )
```

**Parameters**

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

**8.3.4.60 BIG\_256\_56\_sdcopy()**

```
void BIG_256_56_sdcopy (  
    BIG_256_56 x,  
    DBIG_256_56 y )
```

**Parameters**

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

**8.3.4.61 BIG\_256\_56\_sdiv()**

```
void BIG_256_56_sdiv (  
    BIG_256_56 x,  
    BIG_256_56 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\_256\_56\_sducopu()**

```
void BIG_256_56_sducopu (  
    BIG_256_56 x,  
    DBIG_256_56 y )
```

**Parameters**

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

#### 8.3.4.63 BIG\_256\_56\_shl()

```
void BIG_256_56_shl (
    BIG_256_56 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\_256\_56\_shr()

```
void BIG_256_56_shr (
    BIG_256_56 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\_256\_56\_smul()

```
void BIG_256_56_smul (
    BIG_256_56 x,
    BIG_256_56 y,
    BIG_256_56 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\_256\_56\_split()

```
chunk BIG_256_56_split (
```

```

BIG_256_56 x,
BIG_256_56 y,
DBIG_256_56 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\_256\_56\_sqr()

```

void BIG_256_56_sqr (
    DBIG_256_56 x,
    BIG_256_56 y )

```

#### Parameters

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

#### 8.3.4.68 BIG\_256\_56\_ssn()

```

int BIG_256_56_ssn (
    BIG_256_56 r,
    BIG_256_56 a,
    BIG_256_56 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\_256\_56\_sub()**

```
void BIG_256_56_sub (
    BIG_256_56 x,
    BIG_256_56 y,
    BIG_256_56 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\_256\_56\_toBytes()**

```
void BIG_256_56_toBytes (
    char * a,
    BIG_256_56 x )
```

**Parameters**

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

**8.3.4.71 BIG\_256\_56\_zero()**

```
void BIG_256_56_zero (
    BIG_256_56 x )
```

**Parameters**

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

**8.4 big\_384\_56.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_56.h"
```

## Macros

- #define [BIGBITS\\_384\\_56](#) (8\*MODBYTES\_384\_56)
- #define [NLEN\\_384\\_56](#) (1+((8\*MODBYTES\_384\_56-1)/BASEBITS\_384\_56))
- #define [DNLEN\\_384\\_56](#) 2\*NLEN\_384\_56
- #define [BMASK\\_384\\_56](#) (((chunk)1<<BASEBITS\_384\_56)-1)
- #define [NEXCESS\\_384\\_56](#) (1<<(CHUNK-BASEBITS\_384\_56-1))
- #define [HBITS\\_384\\_56](#) (BASEBITS\_384\_56/2)
- #define [HMASK\\_384\\_56](#) (((chunk)1<<HBITS\_384\_56)-1)

## Typedefs

- typedef [chunk](#) [BIG\\_384\\_56](#)[[NLEN\\_384\\_56](#)]
- typedef [chunk](#) [DBIG\\_384\\_56](#)[[DNLEN\\_384\\_56](#)]

## Functions

- int [BIG\\_384\\_56\\_iszilch](#) ([BIG\\_384\\_56](#) x)  
*Tests for BIG equal to zero.*
- int [BIG\\_384\\_56\\_isunity](#) ([BIG\\_384\\_56](#) x)  
*Tests for BIG equal to one.*
- int [BIG\\_384\\_56\\_diszilch](#) ([DBIG\\_384\\_56](#) x)  
*Tests for DBIG equal to zero.*
- void [BIG\\_384\\_56\\_output](#) ([BIG\\_384\\_56](#) x)  
*Outputs a BIG number to the console.*
- void [BIG\\_384\\_56\\_rawoutput](#) ([BIG\\_384\\_56](#) x)  
*Outputs a BIG number to the console in raw form (for debugging)*
- void [BIG\\_384\\_56\\_cswap](#) ([BIG\\_384\\_56](#) x, [BIG\\_384\\_56](#) y, int s)  
*Conditional constant time swap of two BIG numbers.*
- void [BIG\\_384\\_56\\_cmove](#) ([BIG\\_384\\_56](#) x, [BIG\\_384\\_56](#) y, int s)  
*Conditional copy of BIG number.*
- void [BIG\\_384\\_56\\_dcmove](#) ([BIG\\_384\\_56](#) x, [BIG\\_384\\_56](#) y, int s)  
*Conditional copy of DBIG number.*
- void [BIG\\_384\\_56\\_toBytes](#) (char \*a, [BIG\\_384\\_56](#) x)  
*Convert from BIG number to byte array.*
- void [BIG\\_384\\_56\\_fromBytes](#) ([BIG\\_384\\_56](#) x, char \*a)  
*Convert to BIG number from byte array.*
- void [BIG\\_384\\_56\\_fromBytesLen](#) ([BIG\\_384\\_56](#) x, char \*a, int s)  
*Convert to BIG number from byte array of given length.*
- void [BIG\\_384\\_56\\_dfromBytesLen](#) ([DBIG\\_384\\_56](#) x, char \*a, int s)  
*Convert to DBIG number from byte array of given length.*
- void [BIG\\_384\\_56\\_doutput](#) ([DBIG\\_384\\_56](#) x)

- Outputs a DBIG number to the console.*

  - void `BIG_384_56_drawoutput` (`DBIG_384_56 x`)
- Outputs a DBIG number to the console.*

  - void `BIG_384_56_rcopy` (`BIG_384_56 x`, `const BIG_384_56 y`)
- Copy BIG from Read-Only Memory to a BIG.*

  - void `BIG_384_56_copy` (`BIG_384_56 x`, `BIG_384_56 y`)
- Copy BIG to another BIG.*

  - void `BIG_384_56_dcopy` (`DBIG_384_56 x`, `DBIG_384_56 y`)
- Copy DBIG to another DBIG.*

  - void `BIG_384_56_dsucopy` (`DBIG_384_56 x`, `BIG_384_56 y`)
- Copy BIG to upper half of DBIG.*

  - void `BIG_384_56_dscopy` (`DBIG_384_56 x`, `BIG_384_56 y`)
- Copy BIG to lower half of DBIG.*

  - void `BIG_384_56_sdcopy` (`BIG_384_56 x`, `DBIG_384_56 y`)
- Copy lower half of DBIG to a BIG.*

  - void `BIG_384_56_sducopy` (`BIG_384_56 x`, `DBIG_384_56 y`)
- Copy upper half of DBIG to a BIG.*

  - void `BIG_384_56_zero` (`BIG_384_56 x`)
- Set BIG to zero.*

  - void `BIG_384_56_dzero` (`DBIG_384_56 x`)
- Set DBIG to zero.*

  - void `BIG_384_56_one` (`BIG_384_56 x`)
- Set BIG to one (unity)*

  - void `BIG_384_56_invmod2m` (`BIG_384_56 x`)
- Set BIG to inverse mod  $2^{256}$ .*

  - void `BIG_384_56_add` (`BIG_384_56 x`, `BIG_384_56 y`, `BIG_384_56 z`)
- Set BIG to sum of two BIGs - output not normalised.*

  - void `BIG_384_56_or` (`BIG_384_56 x`, `BIG_384_56 y`, `BIG_384_56 z`)
- Set BIG to logical or of two BIGs - output normalised.*

  - void `BIG_384_56_inc` (`BIG_384_56 x`, `int i`)
- Increment BIG by a small integer - output not normalised.*

  - void `BIG_384_56_sub` (`BIG_384_56 x`, `BIG_384_56 y`, `BIG_384_56 z`)
- Set BIG to difference of two BIGs.*

  - void `BIG_384_56_dec` (`BIG_384_56 x`, `int i`)
- Decrement BIG by a small integer - output not normalised.*

  - void `BIG_384_56_dadd` (`DBIG_384_56 x`, `DBIG_384_56 y`, `DBIG_384_56 z`)
- Set DBIG to sum of two DBIGs.*

  - void `BIG_384_56_dsub` (`DBIG_384_56 x`, `DBIG_384_56 y`, `DBIG_384_56 z`)
- Set DBIG to difference of two DBIGs.*

  - void `BIG_384_56_imul` (`BIG_384_56 x`, `BIG_384_56 y`, `int i`)
- Multiply BIG by a small integer - output not normalised.*

  - `chunk` `BIG_384_56_pmul` (`BIG_384_56 x`, `BIG_384_56 y`, `int i`)
- Multiply BIG by not-so-small small integer - output normalised.*

  - `int` `BIG_384_56_div3` (`BIG_384_56 x`)
- Divide BIG by 3 - output normalised.*

  - void `BIG_384_56_pmul` (`DBIG_384_56 x`, `BIG_384_56 y`, `int i`)
- Multiply BIG by even bigger small integer resulting in a DBIG - output normalised.*

  - void `BIG_384_56_mul` (`DBIG_384_56 x`, `BIG_384_56 y`, `BIG_384_56 z`)
- Multiply BIG by another BIG resulting in DBIG - inputs normalised and output normalised.*

  - void `BIG_384_56_smul` (`BIG_384_56 x`, `BIG_384_56 y`, `BIG_384_56 z`)
- Multiply BIG by another BIG resulting in another BIG - inputs normalised and output normalised.*

- void [BIG\\_384\\_56\\_sqr](#) (DBIG\_384\_56 x, BIG\_384\_56 y)  
*Square BIG resulting in a DBIG - input normalised and output normalised.*
- void [BIG\\_384\\_56\\_monty](#) (BIG\_384\_56 a, BIG\_384\_56 md, chunk MC, DBIG\_384\_56 d)  
*Montgomery reduction of a DBIG to a BIG - input normalised and output normalised.*
- void [BIG\\_384\\_56\\_shl](#) (BIG\_384\_56 x, int s)  
*Shifts a BIG left by any number of bits - input must be normalised, output normalised.*
- int [BIG\\_384\\_56\\_fshl](#) (BIG\_384\_56 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\\_56\\_dshl](#) (DBIG\_384\_56 x, int s)  
*Shifts a DBIG left by any number of bits - input must be normalised, output normalised.*
- void [BIG\\_384\\_56\\_shr](#) (BIG\_384\_56 x, int s)  
*Shifts a BIG right by any number of bits - input must be normalised, output normalised.*
- int [BIG\\_384\\_56\\_ssn](#) (BIG\_384\_56 r, BIG\_384\_56 a, BIG\_384\_56 m)  
*Fast time-critical combined shift by 1 bit, subtract and normalise.*
- int [BIG\\_384\\_56\\_fshr](#) (BIG\_384\_56 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\\_56\\_dshr](#) (DBIG\_384\_56 x, int s)  
*Shifts a DBIG right by any number of bits - input must be normalised, output normalised.*
- chunk [BIG\\_384\\_56\\_split](#) (BIG\_384\_56 x, BIG\_384\_56 y, DBIG\_384\_56 z, int s)  
*Splits a DBIG into two BIGs - input must be normalised, outputs normalised.*
- chunk [BIG\\_384\\_56\\_norm](#) (BIG\_384\_56 x)  
*Normalizes a BIG number - output normalised.*
- void [BIG\\_384\\_56\\_dnorm](#) (DBIG\_384\_56 x)  
*Normalizes a DBIG number - output normalised.*
- int [BIG\\_384\\_56\\_comp](#) (BIG\_384\_56 x, BIG\_384\_56 y)  
*Compares two BIG numbers. Inputs must be normalised externally.*
- int [BIG\\_384\\_56\\_dcomp](#) (DBIG\_384\_56 x, DBIG\_384\_56 y)  
*Compares two DBIG numbers. Inputs must be normalised externally.*
- int [BIG\\_384\\_56\\_nbits](#) (BIG\_384\_56 x)  
*Calculate number of bits in a BIG - output normalised.*
- int [BIG\\_384\\_56\\_dnbits](#) (DBIG\_384\_56 x)  
*Calculate number of bits in a DBIG - output normalised.*
- void [BIG\\_384\\_56\\_mod](#) (BIG\_384\_56 x, BIG\_384\_56 n)  
*Reduce x mod n - input and output normalised.*
- void [BIG\\_384\\_56\\_sdiv](#) (BIG\_384\_56 x, BIG\_384\_56 n)  
*Divide x by n - output normalised.*
- void [BIG\\_384\\_56\\_dmod](#) (BIG\_384\_56 x, DBIG\_384\_56 y, BIG\_384\_56 n)  
 *$x=y \bmod n$  - output normalised*
- void [BIG\\_384\\_56\\_ddiv](#) (BIG\_384\_56 x, DBIG\_384\_56 y, BIG\_384\_56 n)  
 *$x=y/n$  - output normalised*
- int [BIG\\_384\\_56\\_parity](#) (BIG\_384\_56 x)  
*return parity of BIG, that is the least significant bit*
- int [BIG\\_384\\_56\\_bit](#) (BIG\_384\_56 x, int i)  
*return i-th of BIG*
- int [BIG\\_384\\_56\\_lastbits](#) (BIG\_384\_56 x, int n)  
*return least significant bits of a BIG*
- void [BIG\\_384\\_56\\_random](#) (BIG\_384\_56 x, csprng \*r)  
*Create a random BIG from a random number generator.*
- void [BIG\\_384\\_56\\_randomnum](#) (BIG\_384\_56 x, BIG\_384\_56 n, csprng \*r)  
*Create an unbiased random BIG from a random number generator, reduced with respect to a modulus.*
- void [BIG\\_384\\_56\\_modmul](#) (BIG\_384\_56 x, BIG\_384\_56 y, BIG\_384\_56 z, BIG\_384\_56 n)

- Calculate  $x=y*z \bmod n$ .*
  - void [BIG\\_384\\_56\\_moddiv](#) ([BIG\\_384\\_56](#) x, [BIG\\_384\\_56](#) y, [BIG\\_384\\_56](#) z, [BIG\\_384\\_56](#) n)
- Calculate  $x=y/z \bmod n$ .*
  - void [BIG\\_384\\_56\\_modsqr](#) ([BIG\\_384\\_56](#) x, [BIG\\_384\\_56](#) y, [BIG\\_384\\_56](#) n)
- Calculate  $x=y^2 \bmod n$ .*
  - void [BIG\\_384\\_56\\_modneg](#) ([BIG\\_384\\_56](#) x, [BIG\\_384\\_56](#) y, [BIG\\_384\\_56](#) n)
- Calculate  $x=-y \bmod n$ .*
  - int [BIG\\_384\\_56\\_jacobi](#) ([BIG\\_384\\_56](#) x, [BIG\\_384\\_56](#) y)
- Calculate jacobi Symbol ( $x/y$ )*
  - void [BIG\\_384\\_56\\_invmodp](#) ([BIG\\_384\\_56](#) x, [BIG\\_384\\_56](#) y, [BIG\\_384\\_56](#) n)
- Calculate  $x=1/y \bmod n$ .*
  - void [BIG\\_384\\_56\\_mod2m](#) ([BIG\\_384\\_56](#) x, int m)
- Calculate  $x=x \bmod 2^m$ .*
  - void [BIG\\_384\\_56\\_dmod2m](#) ([DBIG\\_384\\_56](#) 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\_384\_56

```
#define BIGBITS_384_56 (8*MODBYTES_384_56)
```

Length in bits

#### 8.4.2.2 BMASK\_384\_56

```
#define BMASK_384_56 (((chunk)1<<BASEBITS_384_56)-1)
```

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

#### 8.4.2.3 DNLEN\_384\_56

```
#define DNLEN_384_56 2*NLEN_384_56
```

Double length in bytes



#### 8.4.2.4 HBITS\_384\_56

```
#define HBITS_384_56 (BASEBITS_384_56/2)
```

Number of bits in number base divided by 2

#### 8.4.2.5 HMASK\_384\_56

```
#define HMASK_384_56 (((chunk)1<<HBITS_384_56)-1)
```

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

#### 8.4.2.6 NEXCESS\_384\_56

```
#define NEXCESS_384_56 (1<<(CHUNK-BASEBITS_384_56-1))
```

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

#### 8.4.2.7 NLEN\_384\_56

```
#define NLEN_384_56 (1+((8*MODBYTES_384_56-1)/BASEBITS_384_56))
```

length in bytes

### 8.4.3 Typedef Documentation

#### 8.4.3.1 BIG\_384\_56

```
typedef chunk BIG_384_56[NLEN_384_56]
```

Define type BIG as array of chunks

#### 8.4.3.2 DBIG\_384\_56

```
typedef chunk DBIG_384_56[DNLEN_384_56]
```

Define type DBIG as array of chunks

### 8.4.4 Function Documentation

#### 8.4.4.1 BIG\_384\_56\_add()

```
void BIG_384_56_add (
    BIG_384_56 x,
    BIG_384_56 y,
    BIG_384_56 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\_384\_56\_bit()**

```
int BIG_384_56_bit (
    BIG_384_56 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\_384\_56\_cmove()**

```
void BIG_384_56_cmove (
    BIG_384_56 x,
    BIG_384_56 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\_384\_56\_comp()**

```
int BIG_384_56_comp (
    BIG_384_56 x,
    BIG_384_56 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\_384\_56\_copy()**

```
void BIG_384_56_copy (
    BIG_384_56 x,
    BIG_384_56 y )
```

**Parameters**

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

**8.4.4.6 BIG\_384\_56\_cswap()**

```
void BIG_384_56_cswap (
    BIG_384_56 x,
    BIG_384_56 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\_384\_56\_dadd()**

```
void BIG_384_56_dadd (
    DBIG_384_56 x,
    DBIG_384_56 y,
    DBIG_384_56 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\_384\_56\_dcmove()**

```
void BIG_384_56_dcmove (
    BIG_384_56 x,
    BIG_384_56 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\_384\_56\_dcomp()**

```
int BIG_384_56_dcomp (
    DBIG_384_56 x,
    DBIG_384_56 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\_384\_56\_dcopy()**

```
void BIG_384_56_dcopy (
    DBIG_384_56 x,
    DBIG_384_56 y )
```

## Parameters

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

## 8.4.4.11 BIG\_384\_56\_ddiv()

```
void BIG_384_56_ddiv (
    BIG_384_56 x,
    DBIG_384_56 y,
    BIG_384_56 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\_384\_56\_dec()

```
void BIG_384_56_dec (
    BIG_384_56 x,
    int i )
```

## Parameters

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

## 8.4.4.13 BIG\_384\_56\_dfromBytesLen()

```
void BIG_384_56_dfromBytesLen (
    DBIG_384_56 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\_384\_56\_diszilch()

```
int BIG_384_56_diszilch (
    DBIG_384_56 x )
```

##### Parameters

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

##### Returns

1 if zero, else returns 0

#### 8.4.4.15 BIG\_384\_56\_div3()

```
int BIG_384_56_div3 (
    BIG_384_56 x )
```

##### Parameters

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

##### Returns

Remainder

#### 8.4.4.16 BIG\_384\_56\_dmod()

```
void BIG_384_56_dmod (
    BIG_384_56 x,
    DBIG_384_56 y,
    BIG_384_56 n )
```

Slow but rarely used. y is destroyed.

##### Parameters

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

#### 8.4.4.17 BIG\_384\_56\_dmod2m()

```
void BIG_384_56_dmod2m (
    DBIG_384_56 x,
    int m )
```

Truncation

Parameters

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

#### 8.4.4.18 BIG\_384\_56\_dnbits()

```
int BIG_384_56_dnbits (
    DBIG_384_56 x )
```

Parameters

$x$	DBIG number
-----	-------------

Returns

Number of bits in  $x$

#### 8.4.4.19 BIG\_384\_56\_dnorm()

```
void BIG_384_56_dnorm (
    DBIG_384_56 x )
```

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

Parameters

$x$	DBIG number to be normalised
-----	------------------------------

#### 8.4.4.20 BIG\_384\_56\_doutput()

```
void BIG_384_56_doutput (
```

```
DBIG_384_56 x )
```

**Parameters**

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

**8.4.4.21 BIG\_384\_56\_drawoutput()**

```
void BIG_384_56_drawoutput (
    DBIG_384_56 x )
```

**Parameters**

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

**8.4.4.22 BIG\_384\_56\_dscopy()**

```
void BIG_384_56_dscopy (
    DBIG_384_56 x,
    BIG_384_56 y )
```

**Parameters**

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

**8.4.4.23 BIG\_384\_56\_dshl()**

```
void BIG_384_56_dshl (
    DBIG_384_56 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\_384\_56\_dshr()

```
void BIG_384_56_dshr (
    DBIG_384_56 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\_384\_56\_dsub()

```
void BIG_384_56_dsub (
    DBIG_384_56 x,
    DBIG_384_56 y,
    DBIG_384_56 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\_384\_56\_dsucopy()

```
void BIG_384_56_dsucopy (
    DBIG_384_56 x,
    DBIG_384_56 y )
```

##### Parameters

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

#### 8.4.4.27 BIG\_384\_56\_dzero()

```
void BIG_384_56_dzero (
    DBIG_384_56 x )
```

**Parameters**

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

**8.4.4.28 BIG\_384\_56\_fromBytes()**

```
void BIG_384_56_fromBytes (
    BIG_384_56 x,
    char * a )
```

**Parameters**

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

**8.4.4.29 BIG\_384\_56\_fromBytesLen()**

```
void BIG_384_56_fromBytesLen (
    BIG_384_56 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\_384\_56\_fshl()**

```
int BIG_384_56_fshl (
    BIG_384_56 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\_384\_56\_fshr()**

```
int BIG_384_56_fshr (
    BIG_384_56 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\_384\_56\_imul()**

```
void BIG_384_56_imul (
    BIG_384_56 x,
    BIG_384_56 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\_384\_56\_inc()**

```
void BIG_384_56_inc (
    BIG_384_56 x,
    int i )
```

**Parameters**

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

**8.4.4.34 BIG\_384\_56\_invmod2m()**

```
void BIG_384_56_invmod2m (
    BIG_384_56 x )
```

**Parameters**

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

**8.4.4.35 BIG\_384\_56\_invmodp()**

```
void BIG_384_56_invmodp (
    BIG_384_56 x,
    BIG_384_56 y,
    BIG_384_56 n )
```

Modular Inversion - This is slow. Uses binary method.

**Parameters**

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

**8.4.4.36 BIG\_384\_56\_isunity()**

```
int BIG_384_56_isunity (
    BIG_384_56 x )
```

**Parameters**

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

**Returns**

1 if one, else returns 0

#### 8.4.4.37 BIG\_384\_56\_iszilch()

```
int BIG_384_56_iszilch (  
    BIG_384_56 x )
```

**Parameters**

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

**Returns**

1 if zero, else returns 0

**8.4.4.38 BIG\_384\_56\_jacobi()**

```
int BIG_384_56_jacobi (
    BIG_384_56 x,
    BIG_384_56 y )
```

**Parameters**

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

**Returns**

Jacobi symbol, -1,0 or 1

**8.4.4.39 BIG\_384\_56\_lastbits()**

```
int BIG_384_56_lastbits (
    BIG_384_56 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\_384\_56\_mod()**

```
void BIG_384_56_mod (
    BIG_384_56 x,
    BIG_384_56 n )
```

Slow but rarely used

#### Parameters

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

#### 8.4.4.41 BIG\_384\_56\_mod2m()

```
void BIG_384_56_mod2m (
    BIG_384_56 x,
    int m )
```

Truncation

#### Parameters

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

#### 8.4.4.42 BIG\_384\_56\_moddiv()

```
void BIG_384_56_moddiv (
    BIG_384_56 x,
    BIG_384_56 y,
    BIG_384_56 z,
    BIG_384_56 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\_384\_56\_modmul()

```
void BIG_384_56_modmul (
    BIG_384_56 x,
    BIG_384_56 y,
```

```
BIG_384_56 z,
BIG_384_56 n )
```

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

Given  $x$  and  $3 \times x$  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 = $y \cdot z \bmod n$
$y$	BIG number
$z$	BIG number
$n$	The BIG Modulus

#### 8.4.4.44 BIG\_384\_56\_modneg()

```
void BIG_384_56_modneg (
    BIG_384_56 x,
    BIG_384_56 y,
    BIG_384_56 n )
```

Modular negation

#### Parameters

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

#### 8.4.4.45 BIG\_384\_56\_modsqr()

```
void BIG_384_56_modsqr (
    BIG_384_56 x,
    BIG_384_56 y,
    BIG_384_56 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\_384\_56\_monty()

```
void BIG_384_56_monty (
    BIG_384_56 a,
    BIG_384_56 md,
    chunk MC,
    DBIG_384_56 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\_384\_56\_mul()

```
void BIG_384_56_mul (
    DBIG_384_56 x,
    BIG_384_56 y,
    BIG_384_56 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\_384\_56\_nbits()

```
int BIG_384_56_nbits (
    BIG_384_56 x )
```

##### Parameters

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

##### Returns

Number of bits in x

**8.4.4.49 BIG\_384\_56\_norm()**

```
chunk BIG_384_56_norm (
    BIG_384_56 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\_384\_56\_one()**

```
void BIG_384_56_one (
    BIG_384_56 x )
```

**Parameters**

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

**8.4.4.51 BIG\_384\_56\_or()**

```
void BIG_384_56_or (
    BIG_384_56 x,
    BIG_384_56 y,
    BIG_384_56 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\_384\_56\_output()**

```
void BIG_384_56_output (
    BIG_384_56 x )
```

**Parameters**

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

#### 8.4.4.53 BIG\_384\_56\_parity()

```
int BIG_384_56_parity (
    BIG_384_56 x )
```

##### Parameters

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

##### Returns

0 or 1

#### 8.4.4.54 BIG\_384\_56\_pmul()

```
chunk BIG_384_56_pmul (
    BIG_384_56 x,
    BIG_384_56 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\_384\_56\_pxmuls()

```
void BIG_384_56_pxmuls (
    DBIG_384_56 x,
    BIG_384_56 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\_384\_56\_random()**

```
void BIG_384_56_random (
    BIG_384_56 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\_384\_56\_randomnum()**

```
void BIG_384_56_randomnum (
    BIG_384_56 x,
    BIG_384_56 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\_384\_56\_rawoutput()**

```
void BIG_384_56_rawoutput (
    BIG_384_56 x )
```

**Parameters**

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

**8.4.4.59 BIG\_384\_56\_rcopy()**

```
void BIG_384_56_rcopy (
```

```
BIG_384_56 x,  
const BIG_384_56 y )
```

**Parameters**

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

**8.4.4.60 BIG\_384\_56\_sdcopy()**

```
void BIG_384_56_sdcopy (  
    BIG_384_56 x,  
    DBIG_384_56 y )
```

**Parameters**

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

**8.4.4.61 BIG\_384\_56\_sdiv()**

```
void BIG_384_56_sdiv (  
    BIG_384_56 x,  
    BIG_384_56 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\_384\_56\_sducopy()**

```
void BIG_384_56_sducopy (  
    BIG_384_56 x,  
    DBIG_384_56 y )
```

**Parameters**

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

#### 8.4.4.63 BIG\_384\_56\_shl()

```
void BIG_384_56_shl (
    BIG_384_56 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\_384\_56\_shr()

```
void BIG_384_56_shr (
    BIG_384_56 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\_384\_56\_smul()

```
void BIG_384_56_smul (
    BIG_384_56 x,
    BIG_384_56 y,
    BIG_384_56 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\_384\_56\_split()

```
chunk BIG_384_56_split (
```

```

BIG_384_56 x,
BIG_384_56 y,
DBIG_384_56 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\_384\_56\_sqr()

```

void BIG_384_56_sqr (
    DBIG_384_56 x,
    BIG_384_56 y )

```

#### Parameters

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

#### 8.4.4.68 BIG\_384\_56\_ssn()

```

int BIG_384_56_ssn (
    BIG_384_56 r,
    BIG_384_56 a,
    BIG_384_56 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\_384\_56\_sub()**

```
void BIG_384_56_sub (
    BIG_384_56 x,
    BIG_384_56 y,
    BIG_384_56 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\_384\_56\_toBytes()**

```
void BIG_384_56_toBytes (
    char * a,
    BIG_384_56 x )
```

**Parameters**

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

**8.4.4.71 BIG\_384\_56\_zero()**

```
void BIG_384_56_zero (
    BIG_384_56 x )
```

**Parameters**

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

**8.5 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\*MODBYTES\_384\_58)
- #define [NLEN\\_384\\_58](#) (1+((8\*MODBYTES\_384\_58-1)/BASEBITS\_384\_58))
- #define [DNLEN\\_384\\_58](#) 2\*NLEN\_384\_58
- #define [BMASK\\_384\\_58](#) (((chunk)1<<BASEBITS\_384\_58)-1)
- #define [NEXCESS\\_384\\_58](#) (1<<(CHUNK-BASEBITS\_384\_58-1))
- #define [HBITS\\_384\\_58](#) (BASEBITS\_384\_58/2)
- #define [HMASK\\_384\\_58](#) (((chunk)1<<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\\_isunity](#) ([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_pmul` (`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)  
 *$x=y \bmod n$  - 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  $x=y*z \bmod n$ .*
  - void [BIG\\_384\\_58\\_moddiv](#) ([BIG\\_384\\_58](#) x, [BIG\\_384\\_58](#) y, [BIG\\_384\\_58](#) z, [BIG\\_384\\_58](#) n)
- Calculate  $x=y/z \bmod n$ .*
  - void [BIG\\_384\\_58\\_modsqr](#) ([BIG\\_384\\_58](#) x, [BIG\\_384\\_58](#) y, [BIG\\_384\\_58](#) n)
- Calculate  $x=y^2 \bmod n$ .*
  - void [BIG\\_384\\_58\\_modneg](#) ([BIG\\_384\\_58](#) x, [BIG\\_384\\_58](#) y, [BIG\\_384\\_58](#) n)
- Calculate  $x=-y \bmod n$ .*
  - 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  $x=x \bmod 2^m$ .*

### 8.5.1 Detailed Description

#### Author

Mike Scott

### 8.5.2 Macro Definition Documentation

#### 8.5.2.1 BIGBITS\_384\_58

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

Length in bits

#### 8.5.2.2 BMASK\_384\_58

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

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

#### 8.5.2.3 DNLEN\_384\_58

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

Double length in bytes

#### 8.5.2.4 HBITS\_384\_58

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

Number of bits in number base divided by 2

#### 8.5.2.5 HMASK\_384\_58

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

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

#### 8.5.2.6 NEXCESS\_384\_58

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

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

#### 8.5.2.7 NLEN\_384\_58

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

length in bytes

### 8.5.3 Typedef Documentation

#### 8.5.3.1 BIG\_384\_58

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

Define type BIG as array of chunks

#### 8.5.3.2 DBIG\_384\_58

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

Define type DBIG as array of chunks

### 8.5.4 Function Documentation

#### 8.5.4.1 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

**8.5.4.2 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

**8.5.4.3 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

**8.5.4.4 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 is  $x < y$ , 0 if  $x = y$ , 1 if  $x > y$

**8.5.4.5 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

**8.5.4.6 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

**8.5.4.7 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

**8.5.4.8 BIG\_384\_58\_dcmove()**

```
void BIG_384_58_dcmove (
    DBIG_384_58 x,
    DBIG_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

**8.5.4.9 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  $x > y$

**8.5.4.10 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

## 8.5.4.11 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</i> / <i>n</i>
<i>y</i>	DBIG number
<i>n</i>	Modulus

## 8.5.4.12 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

## 8.5.4.13 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

#### 8.5.4.14 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

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

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

##### Parameters

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

##### Returns

Remainder

#### 8.5.4.16 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 mod n
<i>y</i>	DBIG number
<i>n</i>	Modulus

#### 8.5.4.17 BIG\_384\_58\_dmod2m()

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

Truncation

Parameters

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

#### 8.5.4.18 BIG\_384\_58\_dnbits()

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

Parameters

$x$	DBIG number
-----	-------------

Returns

Number of bits in  $x$

#### 8.5.4.19 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

$x$	DBIG number to be normalised
-----	------------------------------

#### 8.5.4.20 BIG\_384\_58\_doutput()

```
void BIG_384_58_doutput (
```

```
DBIG_384_58 x )
```

**Parameters**

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

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

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

**Parameters**

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

**8.5.4.22 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

**8.5.4.23 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

#### 8.5.4.24 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

#### 8.5.4.25 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

#### 8.5.4.26 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

#### 8.5.4.27 BIG\_384\_58\_dzero()

```
void BIG_384_58_dzero (
    DBIG_384_58 x )
```

**Parameters**

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

**8.5.4.28 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

**8.5.4.29 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

**8.5.4.30 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

**8.5.4.31 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

**8.5.4.32 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

**8.5.4.33 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

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

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

##### Parameters

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

#### 8.5.4.35 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 = $1/y \bmod n$
$y$	BIG number
$n$	The BIG Modulus

#### 8.5.4.36 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



#### 8.5.4.37 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

**8.5.4.38 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

**8.5.4.39 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

**8.5.4.40 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

#### 8.5.4.41 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

#### 8.5.4.42 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

#### 8.5.4.43 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

#### 8.5.4.44 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

#### 8.5.4.45 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

#### 8.5.4.46 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

#### 8.5.4.47 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

#### 8.5.4.48 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

#### 8.5.4.49 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
----------	-----------------------------

#### 8.5.4.50 BIG\_384\_58\_one()

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

##### Parameters

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

#### 8.5.4.51 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

#### 8.5.4.52 BIG\_384\_58\_output()

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

##### Parameters

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

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

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

##### Parameters

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

##### Returns

0 or 1

#### 8.5.4.54 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

#### 8.5.4.55 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

**8.5.4.56 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

**8.5.4.57 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

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

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

**Parameters**

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

**8.5.4.59 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

**8.5.4.60 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

**8.5.4.61 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

**8.5.4.62 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

#### 8.5.4.63 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

#### 8.5.4.64 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

#### 8.5.4.65 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

#### 8.5.4.66 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

#### 8.5.4.67 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

#### 8.5.4.68 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

**8.5.4.69 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

**8.5.4.70 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

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

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

**Parameters**

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

**8.6 big\_448\_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_448_58.h"
```

## Macros

- #define [BIGBITS\\_448\\_58](#) (8\*MODBYTES\_448\_58)
- #define [NLEN\\_448\\_58](#) (1+((8\*MODBYTES\_448\_58-1)/BASEBITS\_448\_58))
- #define [DNLEN\\_448\\_58](#) 2\*NLEN\_448\_58
- #define [BMASK\\_448\\_58](#) (((chunk)1<<BASEBITS\_448\_58)-1)
- #define [NEXCESS\\_448\\_58](#) (1<<(CHUNK-BASEBITS\_448\_58-1))
- #define [HBITS\\_448\\_58](#) (BASEBITS\_448\_58/2)
- #define [HMASK\\_448\\_58](#) (((chunk)1<<HBITS\_448\_58)-1)

## Typedefs

- typedef [chunk](#) [BIG\\_448\\_58](#)[[NLEN\\_448\\_58](#)]
- typedef [chunk](#) [DBIG\\_448\\_58](#)[[DNLEN\\_448\\_58](#)]

## Functions

- int [BIG\\_448\\_58\\_iszilch](#) ([BIG\\_448\\_58](#) x)  
*Tests for BIG equal to zero.*
- int [BIG\\_448\\_58\\_isunity](#) ([BIG\\_448\\_58](#) x)  
*Tests for BIG equal to one.*
- int [BIG\\_448\\_58\\_diszilch](#) ([DBIG\\_448\\_58](#) x)  
*Tests for DBIG equal to zero.*
- void [BIG\\_448\\_58\\_output](#) ([BIG\\_448\\_58](#) x)  
*Outputs a BIG number to the console.*
- void [BIG\\_448\\_58\\_rawoutput](#) ([BIG\\_448\\_58](#) x)  
*Outputs a BIG number to the console in raw form (for debugging)*
- void [BIG\\_448\\_58\\_cswap](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y, int s)  
*Conditional constant time swap of two BIG numbers.*
- void [BIG\\_448\\_58\\_cmove](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y, int s)  
*Conditional copy of BIG number.*
- void [BIG\\_448\\_58\\_dcmove](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y, int s)  
*Conditional copy of DBIG number.*
- void [BIG\\_448\\_58\\_toBytes](#) (char \*a, [BIG\\_448\\_58](#) x)  
*Convert from BIG number to byte array.*
- void [BIG\\_448\\_58\\_fromBytes](#) ([BIG\\_448\\_58](#) x, char \*a)  
*Convert to BIG number from byte array.*
- void [BIG\\_448\\_58\\_fromBytesLen](#) ([BIG\\_448\\_58](#) x, char \*a, int s)  
*Convert to BIG number from byte array of given length.*
- void [BIG\\_448\\_58\\_dfromBytesLen](#) ([DBIG\\_448\\_58](#) x, char \*a, int s)  
*Convert to DBIG number from byte array of given length.*
- void [BIG\\_448\\_58\\_doutput](#) ([DBIG\\_448\\_58](#) x)

- Outputs a DBIG number to the console.*

  - void [BIG\\_448\\_58\\_drawoutput](#) ([DBIG\\_448\\_58](#) x)
- Outputs a DBIG number to the console.*

  - void [BIG\\_448\\_58\\_rcopy](#) ([BIG\\_448\\_58](#) x, const [BIG\\_448\\_58](#) y)
- Copy BIG from Read-Only Memory to a BIG.*

  - void [BIG\\_448\\_58\\_copy](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y)
- Copy BIG to another BIG.*

  - void [BIG\\_448\\_58\\_dcopy](#) ([DBIG\\_448\\_58](#) x, [DBIG\\_448\\_58](#) y)
- Copy DBIG to another DBIG.*

  - void [BIG\\_448\\_58\\_dsucopy](#) ([DBIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y)
- Copy BIG to upper half of DBIG.*

  - void [BIG\\_448\\_58\\_dscopy](#) ([DBIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y)
- Copy BIG to lower half of DBIG.*

  - void [BIG\\_448\\_58\\_sdcopy](#) ([BIG\\_448\\_58](#) x, [DBIG\\_448\\_58](#) y)
- Copy lower half of DBIG to a BIG.*

  - void [BIG\\_448\\_58\\_sducopy](#) ([BIG\\_448\\_58](#) x, [DBIG\\_448\\_58](#) y)
- Copy upper half of DBIG to a BIG.*

  - void [BIG\\_448\\_58\\_zero](#) ([BIG\\_448\\_58](#) x)
- Set BIG to zero.*

  - void [BIG\\_448\\_58\\_dzero](#) ([DBIG\\_448\\_58](#) x)
- Set DBIG to zero.*

  - void [BIG\\_448\\_58\\_one](#) ([BIG\\_448\\_58](#) x)
- Set BIG to one (unity)*

  - void [BIG\\_448\\_58\\_invmod2m](#) ([BIG\\_448\\_58](#) x)
- Set BIG to inverse mod  $2^{256}$ .*

  - void [BIG\\_448\\_58\\_add](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y, [BIG\\_448\\_58](#) z)
- Set BIG to sum of two BIGs - output not normalised.*

  - void [BIG\\_448\\_58\\_or](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y, [BIG\\_448\\_58](#) z)
- Set BIG to logical or of two BIGs - output normalised.*

  - void [BIG\\_448\\_58\\_inc](#) ([BIG\\_448\\_58](#) x, int i)
- Increment BIG by a small integer - output not normalised.*

  - void [BIG\\_448\\_58\\_sub](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y, [BIG\\_448\\_58](#) z)
- Set BIG to difference of two BIGs.*

  - void [BIG\\_448\\_58\\_dec](#) ([BIG\\_448\\_58](#) x, int i)
- Decrement BIG by a small integer - output not normalised.*

  - void [BIG\\_448\\_58\\_dadd](#) ([DBIG\\_448\\_58](#) x, [DBIG\\_448\\_58](#) y, [DBIG\\_448\\_58](#) z)
- Set DBIG to sum of two DBIGs.*

  - void [BIG\\_448\\_58\\_dsub](#) ([DBIG\\_448\\_58](#) x, [DBIG\\_448\\_58](#) y, [DBIG\\_448\\_58](#) z)
- Set DBIG to difference of two DBIGs.*

  - void [BIG\\_448\\_58\\_imul](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y, int i)
- Multiply BIG by a small integer - output not normalised.*

  - [chunk](#) [BIG\\_448\\_58\\_pmul](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y, int i)
- Multiply BIG by not-so-small small integer - output normalised.*

  - int [BIG\\_448\\_58\\_div3](#) ([BIG\\_448\\_58](#) x)
- Divide BIG by 3 - output normalised.*

  - void [BIG\\_448\\_58\\_pmul](#) ([DBIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y, int i)
- Multiply BIG by even bigger small integer resulting in a DBIG - output normalised.*

  - void [BIG\\_448\\_58\\_mul](#) ([DBIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y, [BIG\\_448\\_58](#) z)
- Multiply BIG by another BIG resulting in DBIG - inputs normalised and output normalised.*

  - void [BIG\\_448\\_58\\_smul](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y, [BIG\\_448\\_58](#) z)
- Multiply BIG by another BIG resulting in another BIG - inputs normalised and output normalised.*

- void [BIG\\_448\\_58\\_sqr](#) ([DBIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y)  
*Square BIG resulting in a DBIG - input normalised and output normalised.*
- void [BIG\\_448\\_58\\_monty](#) ([BIG\\_448\\_58](#) a, [BIG\\_448\\_58](#) md, [chunk](#) MC, [DBIG\\_448\\_58](#) d)  
*Montgomery reduction of a DBIG to a BIG - input normalised and output normalised.*
- void [BIG\\_448\\_58\\_shl](#) ([BIG\\_448\\_58](#) x, int s)  
*Shifts a BIG left by any number of bits - input must be normalised, output normalised.*
- int [BIG\\_448\\_58\\_fshl](#) ([BIG\\_448\\_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\\_448\\_58\\_dshl](#) ([DBIG\\_448\\_58](#) x, int s)  
*Shifts a DBIG left by any number of bits - input must be normalised, output normalised.*
- void [BIG\\_448\\_58\\_shr](#) ([BIG\\_448\\_58](#) x, int s)  
*Shifts a BIG right by any number of bits - input must be normalised, output normalised.*
- int [BIG\\_448\\_58\\_ssn](#) ([BIG\\_448\\_58](#) r, [BIG\\_448\\_58](#) a, [BIG\\_448\\_58](#) m)  
*Fast time-critical combined shift by 1 bit, subtract and normalise.*
- int [BIG\\_448\\_58\\_fshr](#) ([BIG\\_448\\_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\\_448\\_58\\_dshr](#) ([DBIG\\_448\\_58](#) x, int s)  
*Shifts a DBIG right by any number of bits - input must be normalised, output normalised.*
- [chunk](#) [BIG\\_448\\_58\\_split](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y, [DBIG\\_448\\_58](#) z, int s)  
*Splits a DBIG into two BIGs - input must be normalised, outputs normalised.*
- [chunk](#) [BIG\\_448\\_58\\_norm](#) ([BIG\\_448\\_58](#) x)  
*Normalizes a BIG number - output normalised.*
- void [BIG\\_448\\_58\\_dnorm](#) ([DBIG\\_448\\_58](#) x)  
*Normalizes a DBIG number - output normalised.*
- int [BIG\\_448\\_58\\_comp](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y)  
*Compares two BIG numbers. Inputs must be normalised externally.*
- int [BIG\\_448\\_58\\_dcomp](#) ([DBIG\\_448\\_58](#) x, [DBIG\\_448\\_58](#) y)  
*Compares two DBIG numbers. Inputs must be normalised externally.*
- int [BIG\\_448\\_58\\_nbits](#) ([BIG\\_448\\_58](#) x)  
*Calculate number of bits in a BIG - output normalised.*
- int [BIG\\_448\\_58\\_dnbits](#) ([DBIG\\_448\\_58](#) x)  
*Calculate number of bits in a DBIG - output normalised.*
- void [BIG\\_448\\_58\\_mod](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) n)  
*Reduce x mod n - input and output normalised.*
- void [BIG\\_448\\_58\\_sdiv](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) n)  
*Divide x by n - output normalised.*
- void [BIG\\_448\\_58\\_dmod](#) ([BIG\\_448\\_58](#) x, [DBIG\\_448\\_58](#) y, [BIG\\_448\\_58](#) n)  
 *$x=y \bmod n$  - output normalised*
- void [BIG\\_448\\_58\\_ddiv](#) ([BIG\\_448\\_58](#) x, [DBIG\\_448\\_58](#) y, [BIG\\_448\\_58](#) n)  
 *$x=y/n$  - output normalised*
- int [BIG\\_448\\_58\\_parity](#) ([BIG\\_448\\_58](#) x)  
*return parity of BIG, that is the least significant bit*
- int [BIG\\_448\\_58\\_bit](#) ([BIG\\_448\\_58](#) x, int i)  
*return i-th of BIG*
- int [BIG\\_448\\_58\\_lastbits](#) ([BIG\\_448\\_58](#) x, int n)  
*return least significant bits of a BIG*
- void [BIG\\_448\\_58\\_random](#) ([BIG\\_448\\_58](#) x, [csprng](#) \*r)  
*Create a random BIG from a random number generator.*
- void [BIG\\_448\\_58\\_randomnum](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) n, [csprng](#) \*r)  
*Create an unbiased random BIG from a random number generator, reduced with respect to a modulus.*
- void [BIG\\_448\\_58\\_modmul](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y, [BIG\\_448\\_58](#) z, [BIG\\_448\\_58](#) n)

- Calculate  $x=y*z \bmod n$ .*
  - void [BIG\\_448\\_58\\_moddiv](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y, [BIG\\_448\\_58](#) z, [BIG\\_448\\_58](#) n)
- Calculate  $x=y/z \bmod n$ .*
  - void [BIG\\_448\\_58\\_modsqr](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y, [BIG\\_448\\_58](#) n)
- Calculate  $x=y^2 \bmod n$ .*
  - void [BIG\\_448\\_58\\_modneg](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y, [BIG\\_448\\_58](#) n)
- Calculate  $x=-y \bmod n$ .*
  - int [BIG\\_448\\_58\\_jacobi](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y)
- Calculate jacobi Symbol ( $x/y$ )*
  - void [BIG\\_448\\_58\\_invmodp](#) ([BIG\\_448\\_58](#) x, [BIG\\_448\\_58](#) y, [BIG\\_448\\_58](#) n)
- Calculate  $x=1/y \bmod n$ .*
  - void [BIG\\_448\\_58\\_mod2m](#) ([BIG\\_448\\_58](#) x, int m)
- Calculate  $x=x \bmod 2^m$ .*
  - void [BIG\\_448\\_58\\_dmod2m](#) ([DBIG\\_448\\_58](#) x, int m)
- Calculate  $x=x \bmod 2^m$ .*

### 8.6.1 Detailed Description

#### Author

Mike Scott

### 8.6.2 Macro Definition Documentation

#### 8.6.2.1 BIGBITS\_448\_58

```
#define BIGBITS_448_58 (8*MODBYTES_448_58)
```

Length in bits

#### 8.6.2.2 BMASK\_448\_58

```
#define BMASK_448_58 (((chunk)1<<BASEBITS_448_58)-1)
```

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

#### 8.6.2.3 DNLEN\_448\_58

```
#define DNLEN_448_58 2*NLEN_448_58
```

Double length in bytes



#### 8.6.2.4 HBITS\_448\_58

```
#define HBITS_448_58 (BASEBITS_448_58/2)
```

Number of bits in number base divided by 2

#### 8.6.2.5 HMASK\_448\_58

```
#define HMASK_448_58 (((chunk)1<<HBITS_448_58)-1)
```

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

#### 8.6.2.6 NEXCESS\_448\_58

```
#define NEXCESS_448_58 (1<<(CHUNK-BASEBITS_448_58-1))
```

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

#### 8.6.2.7 NLEN\_448\_58

```
#define NLEN_448_58 (1+((8*MODBYTES_448_58-1)/BASEBITS_448_58))
```

length in bytes

### 8.6.3 Typedef Documentation

#### 8.6.3.1 BIG\_448\_58

```
typedef chunk BIG_448_58[NLEN_448_58]
```

Define type BIG as array of chunks

#### 8.6.3.2 DBIG\_448\_58

```
typedef chunk DBIG_448_58[DNLEN_448_58]
```

Define type DBIG as array of chunks

### 8.6.4 Function Documentation

#### 8.6.4.1 BIG\_448\_58\_add()

```
void BIG_448_58_add (
    BIG_448_58 x,
    BIG_448_58 y,
    BIG_448_58 z )
```

**Parameters**

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

**8.6.4.2 BIG\_448\_58\_bit()**

```
int BIG_448_58_bit (
    BIG_448_58 x,
    int i )
```

**Parameters**

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

**Returns**

0 or 1

**8.6.4.3 BIG\_448\_58\_cmove()**

```
void BIG_448_58_cmove (
    BIG_448_58 x,
    BIG_448_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

**8.6.4.4 BIG\_448\_58\_comp()**

```
int BIG_448_58_comp (
    BIG_448_58 x,
    BIG_448_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  $x > y$

**8.6.4.5 BIG\_448\_58\_copy()**

```
void BIG_448_58_copy (
    BIG_448_58 x,
    BIG_448_58 y )
```

**Parameters**

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

**8.6.4.6 BIG\_448\_58\_cswap()**

```
void BIG_448_58_cswap (
    BIG_448_58 x,
    BIG_448_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

**8.6.4.7 BIG\_448\_58\_dadd()**

```
void BIG_448_58_dadd (
    DBIG_448_58 x,
    DBIG_448_58 y,
    DBIG_448_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

**8.6.4.8 BIG\_448\_58\_dcmove()**

```
void BIG_448_58_dcmove (
    BIG_448_58 x,
    BIG_448_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

**8.6.4.9 BIG\_448\_58\_dcomp()**

```
int BIG_448_58_dcomp (
    DBIG_448_58 x,
    DBIG_448_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  $x > y$

**8.6.4.10 BIG\_448\_58\_dcopy()**

```
void BIG_448_58_dcopy (
    DBIG_448_58 x,
    DBIG_448_58 y )
```

## Parameters

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

## 8.6.4.11 BIG\_448\_58\_ddiv()

```
void BIG_448_58_ddiv (
    BIG_448_58 x,
    DBIG_448_58 y,
    BIG_448_58 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

## 8.6.4.12 BIG\_448\_58\_dec()

```
void BIG_448_58_dec (
    BIG_448_58 x,
    int i )
```

## Parameters

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

## 8.6.4.13 BIG\_448\_58\_dfromBytesLen()

```
void BIG_448_58_dfromBytesLen (
    DBIG_448_58 x,
    char * a,
    int s )
```

## Parameters

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

#### 8.6.4.14 BIG\_448\_58\_diszilch()

```
int BIG_448_58_diszilch (
    DBIG_448_58 x )
```

##### Parameters

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

##### Returns

1 if zero, else returns 0

#### 8.6.4.15 BIG\_448\_58\_div3()

```
int BIG_448_58_div3 (
    BIG_448_58 x )
```

##### Parameters

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

##### Returns

Remainder

#### 8.6.4.16 BIG\_448\_58\_dmod()

```
void BIG_448_58_dmod (
    BIG_448_58 x,
    DBIG_448_58 y,
    BIG_448_58 n )
```

Slow but rarely used. y is destroyed.

##### Parameters

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

#### 8.6.4.17 BIG\_448\_58\_dmod2m()

```
void BIG_448_58_dmod2m (
    DBIG_448_58 x,
    int m )
```

Truncation

Parameters

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

#### 8.6.4.18 BIG\_448\_58\_dnbits()

```
int BIG_448_58_dnbits (
    DBIG_448_58 x )
```

Parameters

$x$	DBIG number
-----	-------------

Returns

Number of bits in  $x$

#### 8.6.4.19 BIG\_448\_58\_dnorm()

```
void BIG_448_58_dnorm (
    DBIG_448_58 x )
```

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

Parameters

$x$	DBIG number to be normalised
-----	------------------------------

#### 8.6.4.20 BIG\_448\_58\_doutput()

```
void BIG_448_58_doutput (
```

```
DBIG_448_58 x )
```

**Parameters**

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

**8.6.4.21 BIG\_448\_58\_drawoutput()**

```
void BIG_448_58_drawoutput (  
    DBIG_448_58 x )
```

**Parameters**

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

**8.6.4.22 BIG\_448\_58\_dscopy()**

```
void BIG_448_58_dscopy (  
    DBIG_448_58 x,  
    BIG_448_58 y )
```

**Parameters**

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

**8.6.4.23 BIG\_448\_58\_dshl()**

```
void BIG_448_58_dshl (  
    DBIG_448_58 x,  
    int s )
```

**Parameters**

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



#### 8.6.4.24 BIG\_448\_58\_dshr()

```
void BIG_448_58_dshr (
    DBIG_448_58 x,
    int s )
```

##### Parameters

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

#### 8.6.4.25 BIG\_448\_58\_dsub()

```
void BIG_448_58_dsub (
    DBIG_448_58 x,
    DBIG_448_58 y,
    DBIG_448_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

#### 8.6.4.26 BIG\_448\_58\_dsucopy()

```
void BIG_448_58_dsucopy (
    DBIG_448_58 x,
    DBIG_448_58 y )
```

##### Parameters

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

#### 8.6.4.27 BIG\_448\_58\_dzero()

```
void BIG_448_58_dzero (
    DBIG_448_58 x )
```

**Parameters**

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

**8.6.4.28 BIG\_448\_58\_fromBytes()**

```
void BIG_448_58_fromBytes (
    BIG_448_58 x,
    char * a )
```

**Parameters**

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

**8.6.4.29 BIG\_448\_58\_fromBytesLen()**

```
void BIG_448_58_fromBytesLen (
    BIG_448_58 x,
    char * a,
    int s )
```

**Parameters**

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

**8.6.4.30 BIG\_448\_58\_fshl()**

```
int BIG_448_58_fshl (
    BIG_448_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

**8.6.4.31 BIG\_448\_58\_fshr()**

```
int BIG_448_58_fshr (
    BIG_448_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

**8.6.4.32 BIG\_448\_58\_imul()**

```
void BIG_448_58_imul (
    BIG_448_58 x,
    BIG_448_58 y,
    int i )
```

**Parameters**

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

**8.6.4.33 BIG\_448\_58\_inc()**

```
void BIG_448_58_inc (
    BIG_448_58 x,
    int i )
```

**Parameters**

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

#### 8.6.4.34 BIG\_448\_58\_invmod2m()

```
void BIG_448_58_invmod2m (
    BIG_448_58 x )
```

##### Parameters

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

#### 8.6.4.35 BIG\_448\_58\_invmodp()

```
void BIG_448_58_invmodp (
    BIG_448_58 x,
    BIG_448_58 y,
    BIG_448_58 n )
```

Modular Inversion - This is slow. Uses binary method.

##### Parameters

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

#### 8.6.4.36 BIG\_448\_58\_isunity()

```
int BIG_448_58_isunity (
    BIG_448_58 x )
```

##### Parameters

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

##### Returns

1 if one, else returns 0

#### 8.6.4.37 BIG\_448\_58\_iszilch()

```
int BIG_448_58_iszilch (  
    BIG_448_58 x )
```

**Parameters**

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

**Returns**

1 if zero, else returns 0

**8.6.4.38 BIG\_448\_58\_jacobi()**

```
int BIG_448_58_jacobi (
    BIG_448_58 x,
    BIG_448_58 y )
```

**Parameters**

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

**Returns**

Jacobi symbol, -1,0 or 1

**8.6.4.39 BIG\_448\_58\_lastbits()**

```
int BIG_448_58_lastbits (
    BIG_448_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

**8.6.4.40 BIG\_448\_58\_mod()**

```
void BIG_448_58_mod (
    BIG_448_58 x,
    BIG_448_58 n )
```

Slow but rarely used

#### Parameters

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

#### 8.6.4.41 BIG\_448\_58\_mod2m()

```
void BIG_448_58_mod2m (  
    BIG_448_58  $x$ ,  
    int  $m$  )
```

Truncation

#### Parameters

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

#### 8.6.4.42 BIG\_448\_58\_moddiv()

```
void BIG_448_58_moddiv (  
    BIG_448_58  $x$ ,  
    BIG_448_58  $y$ ,  
    BIG_448_58  $z$ ,  
    BIG_448_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

#### 8.6.4.43 BIG\_448\_58\_modmul()

```
void BIG_448_58_modmul (  
    BIG_448_58  $x$ ,  
    BIG_448_58  $y$ ,
```

```
BIG_448_58 z,
BIG_448_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

#### 8.6.4.44 BIG\_448\_58\_modneg()

```
void BIG_448_58_modneg (
    BIG_448_58 x,
    BIG_448_58 y,
    BIG_448_58 n )
```

Modular negation

#### Parameters

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

#### 8.6.4.45 BIG\_448\_58\_modsqr()

```
void BIG_448_58_modsqr (
    BIG_448_58 x,
    BIG_448_58 y,
    BIG_448_58 n )
```

Slow method for modular squaring

#### Parameters

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



#### 8.6.4.46 BIG\_448\_58\_monty()

```
void BIG_448_58_monty (
    BIG_448_58 a,
    BIG_448_58 md,
    chunk MC,
    DBIG_448_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

#### 8.6.4.47 BIG\_448\_58\_mul()

```
void BIG_448_58_mul (
    DBIG_448_58 x,
    BIG_448_58 y,
    BIG_448_58 z )
```

##### Parameters

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

#### 8.6.4.48 BIG\_448\_58\_nbits()

```
int BIG_448_58_nbits (
    BIG_448_58 x )
```

##### Parameters

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

##### Returns

Number of bits in x

#### 8.6.4.49 BIG\_448\_58\_norm()

```
chunk BIG_448_58_norm (
    BIG_448_58 x )
```

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

##### Parameters

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

#### 8.6.4.50 BIG\_448\_58\_one()

```
void BIG_448_58_one (
    BIG_448_58 x )
```

##### Parameters

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

#### 8.6.4.51 BIG\_448\_58\_or()

```
void BIG_448_58_or (
    BIG_448_58 x,
    BIG_448_58 y,
    BIG_448_58 z )
```

##### Parameters

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

#### 8.6.4.52 BIG\_448\_58\_output()

```
void BIG_448_58_output (
    BIG_448_58 x )
```

##### Parameters

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

#### 8.6.4.53 BIG\_448\_58\_parity()

```
int BIG_448_58_parity (
    BIG_448_58 x )
```

##### Parameters

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

##### Returns

0 or 1

#### 8.6.4.54 BIG\_448\_58\_pmul()

```
chunk BIG_448_58_pmul (
    BIG_448_58 x,
    BIG_448_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

#### 8.6.4.55 BIG\_448\_58\_pxmuls()

```
void BIG_448_58_pxmuls (
    DBIG_448_58 x,
    BIG_448_58 y,
    int i )
```

##### Parameters

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

**8.6.4.56 BIG\_448\_58\_random()**

```
void BIG_448_58_random (
    BIG_448_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

**8.6.4.57 BIG\_448\_58\_randomnum()**

```
void BIG_448_58_randomnum (
    BIG_448_58 x,
    BIG_448_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

**8.6.4.58 BIG\_448\_58\_rawoutput()**

```
void BIG_448_58_rawoutput (
    BIG_448_58 x )
```

**Parameters**

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

**8.6.4.59 BIG\_448\_58\_rcopy()**

```
void BIG_448_58_rcopy (
```

```
BIG_448_58 x,  
const BIG_448_58 y )
```

**Parameters**

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

**8.6.4.60 BIG\_448\_58\_sdcopy()**

```
void BIG_448_58_sdcopy (  
    BIG_448_58 x,  
    DBIG_448_58 y )
```

**Parameters**

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

**8.6.4.61 BIG\_448\_58\_sdiv()**

```
void BIG_448_58_sdiv (  
    BIG_448_58 x,  
    BIG_448_58 n )
```

Slow but rarely used

**Parameters**

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

**8.6.4.62 BIG\_448\_58\_sducoppy()**

```
void BIG_448_58_sducoppy (  
    BIG_448_58 x,  
    DBIG_448_58 y )
```

**Parameters**

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

**8.6.4.63 BIG\_448\_58\_shl()**

```
void BIG_448_58_shl (
    BIG_448_58 x,
    int s )
```

**Parameters**

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

**8.6.4.64 BIG\_448\_58\_shr()**

```
void BIG_448_58_shr (
    BIG_448_58 x,
    int s )
```

**Parameters**

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

**8.6.4.65 BIG\_448\_58\_smul()**

```
void BIG_448_58_smul (
    BIG_448_58 x,
    BIG_448_58 y,
    BIG_448_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

**8.6.4.66 BIG\_448\_58\_split()**

```
chunk BIG_448_58_split (
```

```

    BIG_448_58 x,
    BIG_448_58 y,
    DBIG_448_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

#### 8.6.4.67 BIG\_448\_58\_sqr()

```

void BIG_448_58_sqr (
    DBIG_448_58 x,
    BIG_448_58 y )

```

#### Parameters

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

#### 8.6.4.68 BIG\_448\_58\_ssn()

```

int BIG_448_58_ssn (
    BIG_448_58 r,
    BIG_448_58 a,
    BIG_448_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

**8.6.4.69 BIG\_448\_58\_sub()**

```
void BIG_448_58_sub (
    BIG_448_58 x,
    BIG_448_58 y,
    BIG_448_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

**8.6.4.70 BIG\_448\_58\_toBytes()**

```
void BIG_448_58_toBytes (
    char * a,
    BIG_448_58 x )
```

**Parameters**

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

**8.6.4.71 BIG\_448\_58\_zero()**

```
void BIG_448_58_zero (
    BIG_448_58 x )
```

**Parameters**

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

**8.7 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, char \*m, [octet](#) \*S)  
*Calculate a signature.*
- int [BLS\\_BLS381\\_VERIFY](#) ([octet](#) \*SIG, char \*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.*

### 8.7.1 Detailed Description

#### Author

Mike Scott

#### Date

28th Novemebr 2018 Allows some user configuration defines structures declares functions

### 8.7.2 Macro Definition Documentation

#### 8.7.2.1 BFS\_BLS381

```
#define BFS_BLS381_MODBYTES_384_58
```

#### BLS Field Size

### 8.7.2.2 BGS\_BLS381

```
#define BGS_BLS381 MODBYTES_384_58
```

BLS Group Size

### 8.7.2.3 BLS\_FAIL

```
#define BLS_FAIL 41
```

Invalid signature

### 8.7.2.4 BLS\_INVALID\_G1

```
#define BLS_INVALID_G1 42
```

Not a valid G1 point on the curve

### 8.7.2.5 BLS\_INVALID\_G2

```
#define BLS_INVALID_G2 43
```

Not a valid G2 point on the curve

### 8.7.2.6 BLS\_OK

```
#define BLS_OK 0
```

Function completed without error

## 8.7.3 Function Documentation

### 8.7.3.1 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

**8.7.3.2 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

**8.7.3.3 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
<i>W</i>	Public Key. $W = S * G$ , where G is fixed generator

**Returns**

Zero for success or else an error code

**8.7.3.4 BLS\_BLS381\_SIGN()**

```
int BLS_BLS381_SIGN (
    octet * SIG,
    char * 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

**8.7.3.5 BLS\_BLS381\_VERIFY()**

```
int BLS_BLS381_VERIFY (
    octet * SIG,
    char * 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

**8.8 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

**8.8.1 Detailed Description****Author**

Mike Scott

## 8.8.2 Macro Definition Documentation

### 8.8.2.1 BASEBITS\_1024\_58

```
#define BASEBITS_1024_58 58
```

Numbers represented to base  $2^{\text{BASEBITS}}$

### 8.8.2.2 MODBYTES\_1024\_58

```
#define MODBYTES_1024_58 128
```

Number of bytes in Modulus

## 8.9 config\_big\_256\_56.h File Reference

Config BIG Header File.

```
#include "amcl.h"
```

### Macros

- `#define MODBYTES_256_56 32`
- `#define BASEBITS_256_56 56`

### 8.9.1 Detailed Description

Author

Mike Scott

## 8.9.2 Macro Definition Documentation

### 8.9.2.1 BASEBITS\_256\_56

```
#define BASEBITS_256_56 56
```

Numbers represented to base  $2^{\text{BASEBITS}}$

### 8.9.2.2 MODBYTES\_256\_56

```
#define MODBYTES_256_56 32
```

Number of bytes in Modulus

## 8.10 config\_big\_384\_56.h File Reference

Config BIG Header File.

```
#include "amcl.h"
```

### Macros

- `#define MODBYTES_384_56 48`
- `#define BASEBITS_384_56 56`

### 8.10.1 Detailed Description

Author

Mike Scott

### 8.10.2 Macro Definition Documentation

#### 8.10.2.1 BASEBITS\_384\_56

```
#define BASEBITS_384_56 56
```

Numbers represented to base  $2 \times \text{BASEBITS}$

#### 8.10.2.2 MODBYTES\_384\_56

```
#define MODBYTES_384_56 48
```

Number of bytes in Modulus

## 8.11 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

### 8.11.1 Detailed Description

#### Author

Mike Scott

### 8.11.2 Macro Definition Documentation

#### 8.11.2.1 BASEBITS\_384\_58

```
#define BASEBITS_384_58 58
```

Numbers represented to base  $2^{\text{BASEBITS}}$

#### 8.11.2.2 MODBYTES\_384\_58

```
#define MODBYTES_384_58 48
```

Number of bytes in Modulus

## 8.12 config\_big\_448\_58.h File Reference

Config BIG Header File.

```
#include "amcl.h"
```

## Macros

- `#define MODBYTES_448_58` 56
- `#define BASEBITS_448_58` 58

### 8.12.1 Detailed Description

#### Author

Mike Scott

## 8.12.2 Macro Definition Documentation

### 8.12.2.1 BASEBITS\_448\_58

```
#define BASEBITS_448_58 58
```

Numbers represented to base  $2^{\text{BASEBITS}}$

### 8.12.2.2 MODBYTES\_448\_58

```
#define MODBYTES_448_58 56
```

Number of bytes in Modulus

## 8.13 config\_ff\_2048.h File Reference

COnfig FF Header File.

```
#include "amcl.h"
#include "config_big_1024_58.h"
```

### Macros

- `#define FFLEN_2048 2`

### 8.13.1 Detailed Description

#### Author

Mike Scott

## 8.13.2 Macro Definition Documentation

### 8.13.2.1 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



## 8.14 config\_ff\_3072.h File Reference

COnfig FF Header File.

```
#include "amcl.h"
#include "config_big_384_56.h"
```

### Macros

- `#define FFLEN_3072 8`

### 8.14.1 Detailed Description

#### Author

Mike Scott

### 8.14.2 Macro Definition Documentation

#### 8.14.2.1 FFLEN\_3072

```
#define FFLEN_3072 8
```

$2^n$  multiplier of BIGBITS to specify supported Finite Field size, e.g  $2048=256*2^3$  where BIGBITS=256

## 8.15 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.*

### 8.15.1 Detailed Description

#### Author

Mike Scott

### 8.15.2 Macro Definition Documentation

#### 8.15.2.1 ECDH\_ERROR

```
#define ECDH_ERROR -3
```

ECDH Internal Error

#### 8.15.2.2 ECDH\_INVALID

```
#define ECDH_INVALID -4
```

ECDH Internal Error

#### 8.15.2.3 ECDH\_INVALID\_PUBLIC\_KEY

```
#define ECDH_INVALID_PUBLIC_KEY -2
```

Public Key is Invalid

#### 8.15.2.4 ECDH\_OK

```
#define ECDH_OK 0
```

Function completed without error

#### 8.15.2.5 EFS\_BLS381

```
#define EFS_BLS381 MODBYTES_384_58
```

ECC Field Size in bytes

#### 8.15.2.6 EGS\_BLS381

```
#define EGS_BLS381 MODBYTES_384_58
```

ECC Group Size in bytes

### 8.15.3 Function Documentation

#### 8.15.3.1 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

**8.15.3.2 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 recieving 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

**8.15.3.3 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

**8.15.3.4 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

**8.15.3.5 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

**8.15.3.6 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

#### 8.15.3.7 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

## 8.16 ecdh\_ED25519.h File Reference

ECDH Header file for implementation of standard EC protocols.

```

#include "ecp_ED25519.h"
#include "ecdh_support.h"

```

## Macros

- #define [EGS\\_ED25519\\_MODBYTES\\_256\\_56](#)
- #define [EFS\\_ED25519\\_MODBYTES\\_256\\_56](#)
- #define [ECDH\\_OK](#) 0
- #define [ECDH\\_INVALID\\_PUBLIC\\_KEY](#) -2
- #define [ECDH\\_ERROR](#) -3
- #define [ECDH\\_INVALID](#) -4

## Functions

- int [ECP\\_ED25519\\_KEY\\_PAIR\\_GENERATE](#) (csprng \*R, octet \*s, octet \*W)  
*Generate an ECC public/private key pair.*
- int [ECP\\_ED25519\\_PUBLIC\\_KEY\\_VALIDATE](#) (octet \*W)  
*Validate an ECC public key.*
- int [ECP\\_ED25519\\_SVDP\\_DH](#) (octet \*s, octet \*W, octet \*K)  
*Generate Diffie-Hellman shared key.*
- void [ECP\\_ED25519\\_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\\_ED25519\\_ECIES\\_DECRYPT](#) (int h, octet \*P1, octet \*P2, octet \*V, octet \*C, octet \*T, octet \*U, octet \*M)  
*ECIES Decryption.*
- int [ECP\\_ED25519\\_SP\\_DSA](#) (int h, csprng \*R, octet \*k, octet \*s, octet \*M, octet \*c, octet \*d)  
*ECDSA Signature.*
- int [ECP\\_ED25519\\_VP\\_DSA](#) (int h, octet \*W, octet \*M, octet \*c, octet \*d)  
*ECDSA Signature Verification.*

### 8.16.1 Detailed Description

#### Author

Mike Scott

### 8.16.2 Macro Definition Documentation

#### 8.16.2.1 ECDH\_ERROR

```
#define ECDH_ERROR -3
```

ECDH Internal Error

#### 8.16.2.2 ECDH\_INVALID

```
#define ECDH_INVALID -4
```

ECDH Internal Error

### 8.16.2.3 ECDH\_INVALID\_PUBLIC\_KEY

```
#define ECDH_INVALID_PUBLIC_KEY -2
```

Public Key is Invalid

### 8.16.2.4 ECDH\_OK

```
#define ECDH_OK 0
```

Function completed without error

### 8.16.2.5 EFS\_ED25519

```
#define EFS_ED25519 MODBYTES_256_56
```

ECC Field Size in bytes

### 8.16.2.6 EGS\_ED25519

```
#define EGS_ED25519 MODBYTES_256_56
```

ECC Group Size in bytes

## 8.16.3 Function Documentation

### 8.16.3.1 ECP\_ED25519\_ECIES\_DECRYPT()

```
int ECP_ED25519_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

**8.16.3.2 ECP\_ED25519\_ECIES\_ENCRYPT()**

```
void ECP_ED25519_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 recieving 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

**8.16.3.3 ECP\_ED25519\_KEY\_PAIR\_GENERATE()**

```
int ECP_ED25519_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

**8.16.3.4 ECP\_ED25519\_PUBLIC\_KEY\_VALIDATE()**

```
int ECP_ED25519_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

**8.16.3.5 ECP\_ED25519\_SP\_DSA()**

```
int ECP_ED25519_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

**8.16.3.6 ECP\_ED25519\_SVDP\_DH()**

```
int ECP_ED25519_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

#### 8.16.3.7 ECP\_ED25519\_VP\_DSA()

```

int ECP_ED25519_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

## 8.17 ecdh\_GOLDILOCKS.h File Reference

ECDH Header file for implementation of standard EC protocols.

```

#include "ecp_GOLDILOCKS.h"
#include "ecdh_support.h"

```

## Macros

- `#define EGS_GOLDILOCKS MODBYTES_448_58`
- `#define EFS_GOLDILOCKS MODBYTES_448_58`
- `#define ECDH_OK 0`
- `#define ECDH_INVALID_PUBLIC_KEY -2`
- `#define ECDH_ERROR -3`
- `#define ECDH_INVALID -4`

## Functions

- `int ECP_GOLDILOCKS_KEY_PAIR_GENERATE (csprng *R, octet *s, octet *W)`  
*Generate an ECC public/private key pair.*
- `int ECP_GOLDILOCKS_PUBLIC_KEY_VALIDATE (octet *W)`  
*Validate an ECC public key.*
- `int ECP_GOLDILOCKS_SVDP_DH (octet *s, octet *W, octet *K)`  
*Generate Diffie-Hellman shared key.*
- `void ECP_GOLDILOCKS_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_GOLDILOCKS_ECIES_DECRYPT (int h, octet *P1, octet *P2, octet *V, octet *C, octet *T, octet *U, octet *M)`  
*ECIES Decryption.*
- `int ECP_GOLDILOCKS_SP_DSA (int h, csprng *R, octet *k, octet *s, octet *M, octet *c, octet *d)`  
*ECDSA Signature.*
- `int ECP_GOLDILOCKS_VP_DSA (int h, octet *W, octet *M, octet *c, octet *d)`  
*ECDSA Signature Verification.*

### 8.17.1 Detailed Description

#### Author

Mike Scott

### 8.17.2 Macro Definition Documentation

#### 8.17.2.1 ECDH\_ERROR

```
#define ECDH_ERROR -3
```

ECDH Internal Error

#### 8.17.2.2 ECDH\_INVALID

```
#define ECDH_INVALID -4
```

ECDH Internal Error

### 8.17.2.3 ECDH\_INVALID\_PUBLIC\_KEY

```
#define ECDH_INVALID_PUBLIC_KEY -2
```

Public Key is Invalid

### 8.17.2.4 ECDH\_OK

```
#define ECDH_OK 0
```

Function completed without error

### 8.17.2.5 EFS\_GOLDILOCKS

```
#define EFS_GOLDILOCKS MODBYTES_448_58
```

ECC Field Size in bytes

### 8.17.2.6 EGS\_GOLDILOCKS

```
#define EGS_GOLDILOCKS MODBYTES_448_58
```

ECC Group Size in bytes

## 8.17.3 Function Documentation

### 8.17.3.1 ECP\_GOLDILOCKS\_ECIES\_DECRYPT()

```
int ECP_GOLDILOCKS_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

**8.17.3.2 ECP\_GOLDILOCKS\_ECIES\_ENCRYPT()**

```
void ECP_GOLDILOCKS_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 recieving 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

**8.17.3.3 ECP\_GOLDILOCKS\_KEY\_PAIR\_GENERATE()**

```
int ECP_GOLDILOCKS_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

**8.17.3.4 ECP\_GOLDILOCKS\_PUBLIC\_KEY\_VALIDATE()**

```
int ECP_GOLDILOCKS_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

**8.17.3.5 ECP\_GOLDILOCKS\_SP\_DSA()**

```
int ECP_GOLDILOCKS_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

**8.17.3.6 ECP\_GOLDILOCKS\_SVDP\_DH()**

```
int ECP_GOLDILOCKS_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

### 8.17.3.7 ECP\_GOLDILOCKS\_VP\_DSA()

```

int ECP_GOLDILOCKS_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

## 8.18 ecdh\_NIST256.h File Reference

ECDH Header file for implementation of standard EC protocols.

```

#include "ecp_NIST256.h"
#include "ecdh_support.h"

```



## Macros

- `#define EGS_NIST256 MODBYTES_256_56`
- `#define EFS_NIST256 MODBYTES_256_56`
- `#define ECDH_OK 0`
- `#define ECDH_INVALID_PUBLIC_KEY -2`
- `#define ECDH_ERROR -3`
- `#define ECDH_INVALID -4`

## Functions

- `int ECP_NIST256_KEY_PAIR_GENERATE (csprng *R, octet *S, octet *W)`  
*Generate an ECC public/private key pair.*
- `int ECP_NIST256_PUBLIC_KEY_VALIDATE (octet *W)`  
*Validate an ECC public key.*
- `int ECP_NIST256_SVDP_DH (octet *S, octet *W, octet *K)`  
*Generate Diffie-Hellman shared key.*
- `void ECP_NIST256_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_NIST256_ECIES_DECRYPT (int h, octet *P1, octet *P2, octet *V, octet *C, octet *T, octet *U, octet *M)`  
*ECIES Decryption.*
- `int ECP_NIST256_SP_DSA (int h, csprng *R, octet *k, octet *S, octet *M, octet *C, octet *d)`  
*ECDSA Signature.*
- `int ECP_NIST256_VP_DSA (int h, octet *W, octet *M, octet *C, octet *d)`  
*ECDSA Signature Verification.*

### 8.18.1 Detailed Description

#### Author

Mike Scott

### 8.18.2 Macro Definition Documentation

#### 8.18.2.1 ECDH\_ERROR

```
#define ECDH_ERROR -3
```

ECDH Internal Error

#### 8.18.2.2 ECDH\_INVALID

```
#define ECDH_INVALID -4
```

ECDH Internal Error

### 8.18.2.3 ECDH\_INVALID\_PUBLIC\_KEY

```
#define ECDH_INVALID_PUBLIC_KEY -2
```

Public Key is Invalid

### 8.18.2.4 ECDH\_OK

```
#define ECDH_OK 0
```

Function completed without error

### 8.18.2.5 EFS\_NIST256

```
#define EFS_NIST256 MODBYTES_256_56
```

ECC Field Size in bytes

### 8.18.2.6 EGS\_NIST256

```
#define EGS_NIST256 MODBYTES_256_56
```

ECC Group Size in bytes

## 8.18.3 Function Documentation

### 8.18.3.1 ECP\_NIST256\_ECIES\_DECRYPT()

```
int ECP_NIST256_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

## 8.18.3.2 ECP\_NIST256\_ECIES\_ENCRYPT()

```
void ECP_NIST256_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 recieving 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

## 8.18.3.3 ECP\_NIST256\_KEY\_PAIR\_GENERATE()

```
int ECP_NIST256_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

**8.18.3.4 ECP\_NIST256\_PUBLIC\_KEY\_VALIDATE()**

```
int ECP_NIST256_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

**8.18.3.5 ECP\_NIST256\_SP\_DSA()**

```
int ECP_NIST256_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

**8.18.3.6 ECP\_NIST256\_SVDP\_DH()**

```
int ECP_NIST256_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

#### 8.18.3.7 ECP\_NIST256\_VP\_DSA()

```

int ECP_NIST256_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

## 8.19 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.*

### 8.19.1 Detailed Description

#### Author

Mike Scott

### 8.19.2 Function Documentation

#### 8.19.2.1 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

## 8.19.2.2 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

## 8.19.2.3 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

## 8.19.2.4 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 -
	$H(I)$

### 8.19.2.5 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

### 8.19.2.6 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



## 8.19.2.7 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

## 8.20 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 ECP2 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](#)

## 8.20.1 Detailed Description

### Author

Mike Scott

## 8.20.2 Function Documentation

### 8.20.2.1 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

### 8.20.2.2 ECP2\_BLS381\_affine()

```
void ECP2_BLS381_affine (
    ECP2_BLS381 * P )
```

#### Parameters

<i>P</i>	ECP2 instance to be converted to affine form
----------	--

### 8.20.2.3 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

#### 8.20.2.4 ECP2\_BLS381\_dbl()

```
int ECP2_BLS381_dbl (
    ECP2_BLS381 * P )
```

##### Parameters

<i>P</i>	ECP2 instance, on exit =2*P
----------	-----------------------------

#### 8.20.2.5 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

#### 8.20.2.6 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

#### 8.20.2.7 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

**8.20.2.8 ECP2\_BLS381\_generator()**

```
void ECP2_BLS381_generator (  
    ECP2_BLS381 *  $G$  )
```

**Parameters**

$G$	ECP2 instance
-----	---------------

**8.20.2.9 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

**8.20.2.10 ECP2\_BLS381\_inf()**

```
void ECP2_BLS381_inf (  
    ECP2_BLS381 *  $P$  )
```

## Parameters

<i>P</i>	ECP2 instance to be set to infinity
----------	-------------------------------------

## 8.20.2.11 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

## 8.20.2.12 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

## 8.20.2.13 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

#### 8.20.2.14 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

#### 8.20.2.15 ECP2\_BLS381\_neg()

```
void ECP2_BLS381_neg (
    ECP2_BLS381 * P )
```

##### Parameters

<i>P</i>	ECP2 instance, on exit = -P
----------	-----------------------------

#### 8.20.2.16 ECP2\_BLS381\_output()

```
void ECP2_BLS381_output (
    ECP2_BLS381 * P )
```

##### Parameters

<i>P</i>	ECP2 instance to be printed
----------	-----------------------------

#### 8.20.2.17 ECP2\_BLS381\_outputxyz()

```
void ECP2_BLS381_outputxyz (
    ECP2_BLS381 * P )
```

##### Parameters

<i>P</i>	ECP2 instance to be printed
----------	-----------------------------

### 8.20.2.18 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

### 8.20.2.19 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

### 8.20.2.20 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>	BitG x coordinate of point



**Returns**

1 if point exists, else 0

**8.20.2.21 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

**8.20.2.22 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

**8.20.3 Variable Documentation****8.20.3.1 CURVE\_A\_BLS381**

```
const int CURVE_A_BLS381
```

Elliptic curve A parameter

**8.20.3.2 CURVE\_B\_BLS381**

```
const BIG_384_58 CURVE_B_BLS381
```

Elliptic curve B parameter

### 8.20.3.3 CURVE\_B\_I\_BLS381

```
const int CURVE_B_I_BLS381
```

Elliptic curve B parameter

### 8.20.3.4 CURVE\_Bnx\_BLS381

```
const BIG_384_58 CURVE_Bnx_BLS381
```

Elliptic curve parameter

### 8.20.3.5 CURVE\_Cof\_BLS381

```
const BIG_384_58 CURVE_Cof_BLS381
```

Elliptic curve cofactor

### 8.20.3.6 CURVE\_Gx\_BLS381

```
const BIG_384_58 CURVE_Gx_BLS381
```

x-coordinate of generator point in group G1

### 8.20.3.7 CURVE\_Gy\_BLS381

```
const BIG_384_58 CURVE_Gy_BLS381
```

y-coordinate of generator point in group G1

### 8.20.3.8 CURVE\_Order\_BLS381

```
const BIG_384_58 CURVE_Order_BLS381
```

Elliptic curve group order

### 8.20.3.9 CURVE\_Pxa\_BLS381

```
const BIG_384_58 CURVE_Pxa_BLS381
```

real part of x-coordinate of generator point in group G2

### 8.20.3.10 CURVE\_Pxb\_BLS381

```
const BIG_384_58 CURVE_Pxb_BLS381
```

imaginary part of x-coordinate of generator point in group G2

### 8.20.3.11 CURVE\_Pya\_BLS381

```
const BIG_384_58 CURVE_Pya_BLS381
```

real part of y-coordinate of generator point in group G2

### 8.20.3.12 CURVE\_Pyb\_BLS381

```
const BIG_384_58 CURVE_Pyb_BLS381
```

imaginary part of y-coordinate of generator point in group G2

### 8.20.3.13 Fra\_BLS381

```
const BIG_384_58 Fra_BLS381
```

real part of BN curve Frobenius Constant

### 8.20.3.14 Frb\_BLS381

```
const BIG_384_58 Frb_BLS381
```

imaginary part of BN curve Frobenius Constant

## 8.21 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_jsinf` (`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\_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]

### 8.21.1 Detailed Description

#### Author

Mike Scott

## 8.21.2 Function Documentation

### 8.21.2.1 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

### 8.21.2.2 ECP\_BLS381\_affine()

```
void ECP_BLS381_affine (
    ECP_BLS381 * P )
```

#### Parameters

<i>P</i>	ECP instance to be converted to affine form
----------	---

### 8.21.2.3 ECP\_BLS381\_cfp()

```
void ECP_BLS381_cfp (
    ECP_BLS381 * Q )
```

#### Parameters

<i>Q</i>	ECP instance
----------	--------------

### 8.21.2.4 ECP\_BLS381\_copy()

```
void ECP_BLS381_copy (
    ECP_BLS381 * P,
    ECP_BLS381 * Q )
```

## Parameters

<i>P</i>	ECP instance, on exit = Q
<i>Q</i>	ECP instance to be copied

## 8.21.2.5 ECP\_BLS381\_dbl()

```
void ECP_BLS381_dbl (
    ECP_BLS381 * P )
```

## Parameters

<i>P</i>	ECP instance, on exit =2*P
----------	----------------------------

## 8.21.2.6 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

## 8.21.2.7 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

#### 8.21.2.8 ECP\_BLS381\_generator()

```
void ECP_BLS381_generator (
    ECP_BLS381 * G )
```

##### Parameters

<i>G</i>	ECP instance
----------	--------------

#### 8.21.2.9 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

#### 8.21.2.10 ECP\_BLS381\_inf()

```
void ECP_BLS381_inf (
    ECP_BLS381 * P )
```

##### Parameters

<i>P</i>	ECP instance to be set to infinity
----------	------------------------------------



#### 8.21.2.11 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

#### 8.21.2.12 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

#### 8.21.2.13 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

#### 8.21.2.14 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

**8.21.2.15 ECP\_BLS381\_neg()**

```
void ECP_BLS381_neg (  
    ECP_BLS381 * P )
```

**Parameters**

<i>P</i>	ECP instance, on exit = $-P$
----------	------------------------------

**8.21.2.16 ECP\_BLS381\_output()**

```
void ECP_BLS381_output (  
    ECP_BLS381 * P )
```

**Parameters**

<i>P</i>	ECP instance to be printed
----------	----------------------------

**8.21.2.17 ECP\_BLS381\_outputxyz()**

```
void ECP_BLS381_outputxyz (  
    ECP_BLS381 * P )
```

**Parameters**

<i>P</i>	ECP instance to be printed
----------	----------------------------

### 8.21.2.18 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

### 8.21.2.19 ECP\_BLS381\_rawoutput()

```
void ECP_BLS381_rawoutput (
    ECP_BLS381 * P )
```

#### Parameters

<i>P</i>	ECP instance to be printed
----------	----------------------------

### 8.21.2.20 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

### 8.21.2.21 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

**8.21.2.22 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.

**8.21.2.23 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

**8.21.2.24 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

### 8.21.3 Variable Documentation

#### 8.21.3.1 CURVE\_A\_BLS381

```
const int CURVE_A_BLS381
```

Elliptic curve A parameter

#### 8.21.3.2 CURVE\_B\_BLS381

```
const BIG_384_58 CURVE_B_BLS381
```

Elliptic curve B parameter

#### 8.21.3.3 CURVE\_B\_I\_BLS381

```
const int CURVE_B_I_BLS381
```

Elliptic curve B\_i parameter

#### 8.21.3.4 CURVE\_BB\_BLS381

```
const BIG_384_58 CURVE_BB_BLS381[4][4]
```

BN curve constant for GS decomposition

#### 8.21.3.5 CURVE\_Bnx\_BLS381

```
const BIG_384_58 CURVE_Bnx_BLS381
```

BN curve x parameter

#### 8.21.3.6 CURVE\_Cof\_BLS381

```
const BIG_384_58 CURVE_Cof_BLS381
```

Elliptic curve cofactor

#### 8.21.3.7 CURVE\_Cof\_I\_BLS381

```
const int CURVE_Cof_I_BLS381
```

Elliptic curve cofactor

#### 8.21.3.8 CURVE\_Cru\_BLS381

```
const BIG_384_58 CURVE_Cru_BLS381
```

BN curve Cube Root of Unity

#### 8.21.3.9 CURVE\_Gx\_BLS381

```
const BIG_384_58 CURVE_Gx_BLS381
```

x-coordinate of generator point in group G1

#### 8.21.3.10 CURVE\_Gy\_BLS381

```
const BIG_384_58 CURVE_Gy_BLS381
```

y-coordinate of generator point in group G1

#### 8.21.3.11 CURVE\_Order\_BLS381

```
const BIG_384_58 CURVE_Order_BLS381
```

Elliptic curve group order

#### 8.21.3.12 CURVE\_Pxa\_BLS381

```
const BIG_384_58 CURVE_Pxa_BLS381
```

real part of x-coordinate of generator point in group G2

#### 8.21.3.13 CURVE\_Pxaa\_BLS381

```
const BIG_384_58 CURVE_Pxaa_BLS381
```

real part of x-coordinate of generator point in group G2

#### 8.21.3.14 CURVE\_Pxaaa\_BLS381

```
const BIG_384_58 CURVE_Pxaaa_BLS381
```

real part of x-coordinate of generator point in group G2

**8.21.3.15 CURVE\_Pxaab\_BLS381**

```
const BIG_384_58 CURVE_Pxaab_BLS381
```

imaginary part of x-coordinate of generator point in group G2

**8.21.3.16 CURVE\_Pxab\_BLS381**

```
const BIG_384_58 CURVE_Pxab_BLS381
```

imaginary part of x-coordinate of generator point in group G2

**8.21.3.17 CURVE\_Pxaba\_BLS381**

```
const BIG_384_58 CURVE_Pxaba_BLS381
```

real part of x-coordinate of generator point in group G2

**8.21.3.18 CURVE\_Pxabb\_BLS381**

```
const BIG_384_58 CURVE_Pxabb_BLS381
```

imaginary part of x-coordinate of generator point in group G2

**8.21.3.19 CURVE\_Pxb\_BLS381**

```
const BIG_384_58 CURVE_Pxb_BLS381
```

imaginary part of x-coordinate of generator point in group G2

**8.21.3.20 CURVE\_Pxba\_BLS381**

```
const BIG_384_58 CURVE_Pxba_BLS381
```

real part of x-coordinate of generator point in group G2

**8.21.3.21 CURVE\_Pxbaa\_BLS381**

```
const BIG_384_58 CURVE_Pxbaa_BLS381
```

real part of x-coordinate of generator point in group G2

**8.21.3.22 CURVE\_Pxbab\_BLS381**

```
const BIG_384_58 CURVE_Pxbab_BLS381
```

imaginary part of x-coordinate of generator point in group G2



**8.21.3.23 CURVE\_Pxbb\_BLS381**

```
const BIG_384_58 CURVE_Pxbb_BLS381
```

imaginary part of x-coordinate of generator point in group G2

**8.21.3.24 CURVE\_Pxbba\_BLS381**

```
const BIG_384_58 CURVE_Pxbba_BLS381
```

real part of x-coordinate of generator point in group G2

**8.21.3.25 CURVE\_Pxbbb\_BLS381**

```
const BIG_384_58 CURVE_Pxbbb_BLS381
```

imaginary part of x-coordinate of generator point in group G2

**8.21.3.26 CURVE\_Pya\_BLS381**

```
const BIG_384_58 CURVE_Pya_BLS381
```

real part of y-coordinate of generator point in group G2

**8.21.3.27 CURVE\_Pyaa\_BLS381**

```
const BIG_384_58 CURVE_Pyaa_BLS381
```

real part of y-coordinate of generator point in group G2

**8.21.3.28 CURVE\_Pyaaa\_BLS381**

```
const BIG_384_58 CURVE_Pyaaa_BLS381
```

real part of y-coordinate of generator point in group G2

**8.21.3.29 CURVE\_Pyaab\_BLS381**

```
const BIG_384_58 CURVE_Pyaab_BLS381
```

imaginary part of y-coordinate of generator point in group G2

**8.21.3.30 CURVE\_Pyab\_BLS381**

```
const BIG_384_58 CURVE_Pyab_BLS381
```

imaginary part of y-coordinate of generator point in group G2

**8.21.3.31 CURVE\_Pyaba\_BLS381**

```
const BIG_384_58 CURVE_Pyaba_BLS381
```

real part of y-coordinate of generator point in group G2

**8.21.3.32 CURVE\_Pyabb\_BLS381**

```
const BIG_384_58 CURVE_Pyabb_BLS381
```

imaginary part of y-coordinate of generator point in group G2

**8.21.3.33 CURVE\_Pyb\_BLS381**

```
const BIG_384_58 CURVE_Pyb_BLS381
```

imaginary part of y-coordinate of generator point in group G2

**8.21.3.34 CURVE\_Pyba\_BLS381**

```
const BIG_384_58 CURVE_Pyba_BLS381
```

real part of y-coordinate of generator point in group G2

**8.21.3.35 CURVE\_Pybaa\_BLS381**

```
const BIG_384_58 CURVE_Pybaa_BLS381
```

real part of y-coordinate of generator point in group G2

**8.21.3.36 CURVE\_Pybab\_BLS381**

```
const BIG_384_58 CURVE_Pybab_BLS381
```

imaginary part of y-coordinate of generator point in group G2

**8.21.3.37 CURVE\_Pybb\_BLS381**

```
const BIG_384_58 CURVE_Pybb_BLS381
```

imaginary part of y-coordinate of generator point in group G2

**8.21.3.38 CURVE\_Pybba\_BLS381**

```
const BIG_384_58 CURVE_Pybba_BLS381
```

real part of y-coordinate of generator point in group G2

#### 8.21.3.39 CURVE\_Pybbb\_BLS381

```
const BIG_384_58 CURVE_Pybbb_BLS381
```

imaginary part of y-coordinate of generator point in group G2

#### 8.21.3.40 CURVE\_SB\_BLS381

```
const BIG_384_58 CURVE_SB_BLS381[2][2]
```

BN curve constant for GLV decomposition

#### 8.21.3.41 CURVE\_W\_BLS381

```
const BIG_384_58 CURVE_W_BLS381[2]
```

BN curve constant for GLV decomposition

#### 8.21.3.42 CURVE\_WB\_BLS381

```
const BIG_384_58 CURVE_WB_BLS381[4]
```

BN curve constant for GS decomposition

#### 8.21.3.43 Fra\_BLS381

```
const BIG_384_58 Fra_BLS381
```

real part of BN curve Frobenius Constant

#### 8.21.3.44 Frb\_BLS381

```
const BIG_384_58 Frb_BLS381
```

imaginary part of BN curve Frobenius Constant

## 8.22 ecp\_ED25519.h File Reference

ECP Header File.

```
#include "fp_25519.h"
#include "config_curve_ED25519.h"
```

## Data Structures

- struct [ECP\\_ED25519](#)  
*ECP structure - Elliptic Curve Point over base field.*

## Functions

- int [ECP\\_ED25519\\_isinf](#) ([ECP\\_ED25519](#) \*P)  
*Tests for ECP point equal to infinity.*
- int [ECP\\_ED25519\\_equals](#) ([ECP\\_ED25519](#) \*P, [ECP\\_ED25519](#) \*Q)  
*Tests for equality of two ECPs.*
- void [ECP\\_ED25519\\_copy](#) ([ECP\\_ED25519](#) \*P, [ECP\\_ED25519](#) \*Q)  
*Copy ECP point to another ECP point.*
- void [ECP\\_ED25519\\_neg](#) ([ECP\\_ED25519](#) \*P)  
*Negation of an ECP point.*
- void [ECP\\_ED25519\\_inf](#) ([ECP\\_ED25519](#) \*P)  
*Set ECP to point-at-infinity.*
- void [ECP\\_ED25519\\_rhs](#) ([FP\\_25519](#) \*r, [FP\\_25519](#) \*x)  
*Calculate Right Hand Side of curve equation  $y^2=f(x)$*
- int [ECP\\_ED25519\\_set](#) ([ECP\\_ED25519](#) \*P, [BIG\\_256\\_56](#) x, [BIG\\_256\\_56](#) y)  
*Set ECP to point(x,y) given x and y.*
- int [ECP\\_ED25519\\_get](#) ([BIG\\_256\\_56](#) x, [BIG\\_256\\_56](#) y, [ECP\\_ED25519](#) \*P)  
*Extract x and y coordinates of an ECP point P*
- void [ECP\\_ED25519\\_add](#) ([ECP\\_ED25519](#) \*P, [ECP\\_ED25519](#) \*Q)  
*Adds ECP instance Q to ECP instance P.*
- void [ECP\\_ED25519\\_sub](#) ([ECP\\_ED25519](#) \*P, [ECP\\_ED25519](#) \*Q)  
*Subtracts ECP instance Q from ECP instance P.*
- int [ECP\\_ED25519\\_setx](#) ([ECP\\_ED25519](#) \*P, [BIG\\_256\\_56](#) x, int s)  
*Set ECP to point(x,y) given just x and sign of y.*
- void [ECP\\_ED25519\\_cfp](#) ([ECP\\_ED25519](#) \*Q)  
*Multiplies Point by curve co-factor.*
- void [ECP\\_ED25519\\_mapit](#) ([ECP\\_ED25519](#) \*Q, [octet](#) \*w)  
*Maps random BIG to curve point of correct order.*
- void [ECP\\_ED25519\\_affine](#) ([ECP\\_ED25519](#) \*P)  
*Converts an ECP point from Projective (x,y,z) coordinates to affine (x,y) coordinates.*
- void [ECP\\_ED25519\\_outputxyz](#) ([ECP\\_ED25519](#) \*P)  
*Formats and outputs an ECP point to the console, in projective coordinates.*
- void [ECP\\_ED25519\\_output](#) ([ECP\\_ED25519](#) \*P)  
*Formats and outputs an ECP point to the console, converted to affine coordinates.*
- void [ECP\\_ED25519\\_rawoutput](#) ([ECP\\_ED25519](#) \*P)  
*Formats and outputs an ECP point to the console.*
- void [ECP\\_ED25519\\_toOctet](#) ([octet](#) \*S, [ECP\\_ED25519](#) \*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\\_ED25519\\_fromOctet](#) ([ECP\\_ED25519](#) \*P, [octet](#) \*S)  
*Creates an ECP point from an octet string.*
- void [ECP\\_ED25519\\_dbl](#) ([ECP\\_ED25519](#) \*P)  
*Doubles an ECP instance P.*
- void [ECP\\_ED25519\\_pinmul](#) ([ECP\\_ED25519](#) \*P, int i, int b)  
*Multiplies an ECP instance P by a small integer, side-channel resistant.*

- void [ECP\\_ED25519\\_mul](#) (ECP\_ED25519 \*P, BIG\_256\_56 b)  
*Multiplies an ECP instance P by a BIG, side-channel resistant.*
- void [ECP\\_ED25519\\_mul2](#) (ECP\_ED25519 \*P, ECP\_ED25519 \*Q, BIG\_256\_56 e, BIG\_256\_56 f)  
*Calculates double multiplication  $P=e*P+f*Q$ , side-channel resistant.*
- void [ECP\\_ED25519\\_generator](#) (ECP\_ED25519 \*G)  
*Get Group Generator from ROM.*

## Variables

- const int [CURVE\\_A\\_ED25519](#)
- const int [CURVE\\_Cof\\_I\\_ED25519](#)
- const int [CURVE\\_B\\_I\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_B\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Order\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Cof\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Gx\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Gy\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pxa\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pxb\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pya\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pyb\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pxaa\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pxab\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pxba\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pxbb\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pyaa\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pyab\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pyba\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pybb\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pxaaa\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pxaab\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pxaba\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pxab\\_b\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pxbaa\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pxbab\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pxbba\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pxbbb\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pyaaa\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pyaab\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pyaba\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pyabb\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pybaa\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pybab\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pybba\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Pybbb\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Bnx\\_ED25519](#)
- const [BIG\\_256\\_56 CURVE\\_Cru\\_ED25519](#)
- const [BIG\\_256\\_56 Fra\\_25519](#)
- const [BIG\\_256\\_56 Frb\\_25519](#)
- const [BIG\\_256\\_56 CURVE\\_W\\_ED25519](#) [2]
- const [BIG\\_256\\_56 CURVE\\_SB\\_ED25519](#) [2][2]
- const [BIG\\_256\\_56 CURVE\\_WB\\_ED25519](#) [4]
- const [BIG\\_256\\_56 CURVE\\_BB\\_ED25519](#) [4][4]

## 8.22.1 Detailed Description

### Author

Mike Scott

## 8.22.2 Function Documentation

### 8.22.2.1 ECP\_ED25519\_add()

```
void ECP_ED25519_add (
    ECP_ED25519 * P,
    ECP_ED25519 * Q )
```

#### Parameters

<i>P</i>	ECP instance, on exit =P+Q
<i>Q</i>	ECP instance to be added to P

### 8.22.2.2 ECP\_ED25519\_affine()

```
void ECP_ED25519_affine (
    ECP_ED25519 * P )
```

#### Parameters

<i>P</i>	ECP instance to be converted to affine form
----------	---

### 8.22.2.3 ECP\_ED25519\_cfp()

```
void ECP_ED25519_cfp (
    ECP_ED25519 * Q )
```

#### Parameters

<i>Q</i>	ECP instance
----------	--------------

#### 8.22.2.4 ECP\_ED25519\_copy()

```
void ECP_ED25519_copy (
    ECP_ED25519 * P,
    ECP_ED25519 * Q )
```

##### Parameters

<i>P</i>	ECP instance, on exit = Q
<i>Q</i>	ECP instance to be copied

#### 8.22.2.5 ECP\_ED25519\_dbl()

```
void ECP_ED25519_dbl (
    ECP_ED25519 * P )
```

##### Parameters

<i>P</i>	ECP instance, on exit =2*P
----------	----------------------------

#### 8.22.2.6 ECP\_ED25519\_equals()

```
int ECP_ED25519_equals (
    ECP_ED25519 * P,
    ECP_ED25519 * 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

#### 8.22.2.7 ECP\_ED25519\_fromOctet()

```
int ECP_ED25519_fromOctet (
    ECP_ED25519 * 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

## 8.22.2.8 ECP\_ED25519\_generator()

```
void ECP_ED25519_generator (
    ECP_ED25519 * G )
```

## Parameters

<i>G</i>	ECP instance
----------	--------------

## 8.22.2.9 ECP\_ED25519\_get()

```
int ECP_ED25519_get (
    BIG_256_56 x,
    BIG_256_56 y,
    ECP_ED25519 * 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

## 8.22.2.10 ECP\_ED25519\_inf()

```
void ECP_ED25519_inf (
    ECP_ED25519 * P )
```

## Parameters

<i>P</i>	ECP instance to be set to infinity
----------	------------------------------------



### 8.22.2.11 ECP\_ED25519\_isinf()

```
int ECP_ED25519_isinf (
    ECP_ED25519 * P )
```

#### Parameters

<i>P</i>	ECP point to be tested
----------	------------------------

#### Returns

1 if infinity, else returns 0

### 8.22.2.12 ECP\_ED25519\_mapit()

```
void ECP_ED25519_mapit (
    ECP_ED25519 * Q,
    octet * w )
```

#### Parameters

<i>Q</i>	ECP instance of correct order
<i>w</i>	OCTET byte array to be mapped

### 8.22.2.13 ECP\_ED25519\_mul()

```
void ECP_ED25519_mul (
    ECP_ED25519 * P,
    BIG_256_56 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

**8.22.2.14 ECP\_ED25519\_mul2()**

```
void ECP_ED25519_mul2 (
    ECP_ED25519 * P,
    ECP_ED25519 * Q,
    BIG_256_56 e,
    BIG_256_56 f )
```

**Parameters**

<i>P</i>	ECP instance, on exit = $e \cdot P + f \cdot Q$
<i>Q</i>	ECP instance
<i>e</i>	BIG number multiplier
<i>f</i>	BIG number multiplier

**8.22.2.15 ECP\_ED25519\_neg()**

```
void ECP_ED25519_neg (
    ECP_ED25519 * P )
```

**Parameters**

<i>P</i>	ECP instance, on exit = $-P$
----------	------------------------------

**8.22.2.16 ECP\_ED25519\_output()**

```
void ECP_ED25519_output (
    ECP_ED25519 * P )
```

**Parameters**

<i>P</i>	ECP instance to be printed
----------	----------------------------

**8.22.2.17 ECP\_ED25519\_outputxyz()**

```
void ECP_ED25519_outputxyz (
    ECP_ED25519 * P )
```

**Parameters**

<i>P</i>	ECP instance to be printed
----------	----------------------------

### 8.22.2.18 ECP\_ED25519\_pinmul()

```
void ECP_ED25519_pinmul (
    ECP_ED25519 * P,
    int i,
    int b )
```

#### Parameters

<i>P</i>	ECP instance, on exit $=i \cdot P$
<i>i</i>	small integer multiplier
<i>b</i>	maximum number of bits in multiplier

### 8.22.2.19 ECP\_ED25519\_rawoutput()

```
void ECP_ED25519_rawoutput (
    ECP_ED25519 * P )
```

#### Parameters

<i>P</i>	ECP instance to be printed
----------	----------------------------

### 8.22.2.20 ECP\_ED25519\_rhs()

```
void ECP_ED25519_rhs (
    FP_25519 * r,
    FP_25519 * 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 $x$

### 8.22.2.21 ECP\_ED25519\_set()

```
int ECP_ED25519_set (
    ECP_ED25519 * P,
```

```

    BIG_256_56 x,
    BIG_256_56 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

#### 8.22.2.22 ECP\_ED25519\_setx()

```

int ECP_ED25519_setx (
    ECP_ED25519 * P,
    BIG_256_56 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.

#### 8.22.2.23 ECP\_ED25519\_sub()

```

void ECP_ED25519_sub (
    ECP_ED25519 * P,
    ECP_ED25519 * Q )

```

#### Parameters

<i>P</i>	ECP instance, on exit =P-Q
<i>Q</i>	ECP instance to be subtracted from P

#### 8.22.2.24 ECP\_ED25519\_toOctet()

```
void ECP_ED25519_toOctet (
    octet * S,
    ECP_ED25519 * 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

### 8.22.3 Variable Documentation

#### 8.22.3.1 CURVE\_A\_ED25519

```
const int CURVE_A_ED25519
```

Elliptic curve A parameter

#### 8.22.3.2 CURVE\_B\_ED25519

```
const BIG_256_56 CURVE_B_ED25519
```

Elliptic curve B parameter

#### 8.22.3.3 CURVE\_B\_I\_ED25519

```
const int CURVE_B_I_ED25519
```

Elliptic curve B\_i parameter

#### 8.22.3.4 CURVE\_BB\_ED25519

```
const BIG_256_56 CURVE_BB_ED25519[4][4]
```

BN curve constant for GS decomposition

#### 8.22.3.5 CURVE\_Bnx\_ED25519

```
const BIG_256_56 CURVE_Bnx_ED25519
```

BN curve x parameter

#### 8.22.3.6 CURVE\_Cof\_ED25519

```
const BIG_256_56 CURVE_Cof_ED25519
```

Elliptic curve cofactor

#### 8.22.3.7 CURVE\_Cof\_I\_ED25519

```
const int CURVE_Cof_I_ED25519
```

Elliptic curve cofactor

#### 8.22.3.8 CURVE\_Cru\_ED25519

```
const BIG_256_56 CURVE_Cru_ED25519
```

BN curve Cube Root of Unity

#### 8.22.3.9 CURVE\_Gx\_ED25519

```
const BIG_256_56 CURVE_Gx_ED25519
```

x-coordinate of generator point in group G1

#### 8.22.3.10 CURVE\_Gy\_ED25519

```
const BIG_256_56 CURVE_Gy_ED25519
```

y-coordinate of generator point in group G1

#### 8.22.3.11 CURVE\_Order\_ED25519

```
const BIG_256_56 CURVE_Order_ED25519
```

Elliptic curve group order

#### 8.22.3.12 CURVE\_Pxa\_ED25519

```
const BIG_256_56 CURVE_Pxa_ED25519
```

real part of x-coordinate of generator point in group G2

#### 8.22.3.13 CURVE\_Pxaa\_ED25519

```
const BIG_256_56 CURVE_Pxaa_ED25519
```

real part of x-coordinate of generator point in group G2

**8.22.3.14 CURVE\_Pxaaa\_ED25519**

```
const BIG_256_56 CURVE_Pxaaa_ED25519
```

real part of x-coordinate of generator point in group G2

**8.22.3.15 CURVE\_Pxaab\_ED25519**

```
const BIG_256_56 CURVE_Pxaab_ED25519
```

imaginary part of x-coordinate of generator point in group G2

**8.22.3.16 CURVE\_Pxab\_ED25519**

```
const BIG_256_56 CURVE_Pxab_ED25519
```

imaginary part of x-coordinate of generator point in group G2

**8.22.3.17 CURVE\_Pxaba\_ED25519**

```
const BIG_256_56 CURVE_Pxaba_ED25519
```

real part of x-coordinate of generator point in group G2

**8.22.3.18 CURVE\_Pxabbb\_ED25519**

```
const BIG_256_56 CURVE_Pxabbb_ED25519
```

imaginary part of x-coordinate of generator point in group G2

**8.22.3.19 CURVE\_Pxb\_ED25519**

```
const BIG_256_56 CURVE_Pxb_ED25519
```

imaginary part of x-coordinate of generator point in group G2

**8.22.3.20 CURVE\_Pxba\_ED25519**

```
const BIG_256_56 CURVE_Pxba_ED25519
```

real part of x-coordinate of generator point in group G2

**8.22.3.21 CURVE\_Pxbaa\_ED25519**

```
const BIG_256_56 CURVE_Pxbaa_ED25519
```

real part of x-coordinate of generator point in group G2

**8.22.3.22 CURVE\_Pxbab\_ED25519**

```
const BIG_256_56 CURVE_Pxbab_ED25519
```

imaginary part of x-coordinate of generator point in group G2

**8.22.3.23 CURVE\_Pxbb\_ED25519**

```
const BIG_256_56 CURVE_Pxbb_ED25519
```

imaginary part of x-coordinate of generator point in group G2

**8.22.3.24 CURVE\_Pxbba\_ED25519**

```
const BIG_256_56 CURVE_Pxbba_ED25519
```

real part of x-coordinate of generator point in group G2

**8.22.3.25 CURVE\_Pxbbb\_ED25519**

```
const BIG_256_56 CURVE_Pxbbb_ED25519
```

imaginary part of x-coordinate of generator point in group G2

**8.22.3.26 CURVE\_Pya\_ED25519**

```
const BIG_256_56 CURVE_Pya_ED25519
```

real part of y-coordinate of generator point in group G2

**8.22.3.27 CURVE\_Pyaa\_ED25519**

```
const BIG_256_56 CURVE_Pyaa_ED25519
```

real part of y-coordinate of generator point in group G2

**8.22.3.28 CURVE\_Pyaaa\_ED25519**

```
const BIG_256_56 CURVE_Pyaaa_ED25519
```

real part of y-coordinate of generator point in group G2

**8.22.3.29 CURVE\_Pyaab\_ED25519**

```
const BIG_256_56 CURVE_Pyaab_ED25519
```

imaginary part of y-coordinate of generator point in group G2



**8.22.3.30 CURVE\_Pyab\_ED25519**

const [BIG\\_256\\_56](#) CURVE\_Pyab\_ED25519

imaginary part of y-coordinate of generator point in group G2

**8.22.3.31 CURVE\_Pyaba\_ED25519**

const [BIG\\_256\\_56](#) CURVE\_Pyaba\_ED25519

real part of y-coordinate of generator point in group G2

**8.22.3.32 CURVE\_Pyabb\_ED25519**

const [BIG\\_256\\_56](#) CURVE\_Pyabb\_ED25519

imaginary part of y-coordinate of generator point in group G2

**8.22.3.33 CURVE\_Pyb\_ED25519**

const [BIG\\_256\\_56](#) CURVE\_Pyb\_ED25519

imaginary part of y-coordinate of generator point in group G2

**8.22.3.34 CURVE\_Pyba\_ED25519**

const [BIG\\_256\\_56](#) CURVE\_Pyba\_ED25519

real part of y-coordinate of generator point in group G2

**8.22.3.35 CURVE\_Pybaa\_ED25519**

const [BIG\\_256\\_56](#) CURVE\_Pybaa\_ED25519

real part of y-coordinate of generator point in group G2

**8.22.3.36 CURVE\_Pybab\_ED25519**

const [BIG\\_256\\_56](#) CURVE\_Pybab\_ED25519

imaginary part of y-coordinate of generator point in group G2

**8.22.3.37 CURVE\_Pybb\_ED25519**

const [BIG\\_256\\_56](#) CURVE\_Pybb\_ED25519

imaginary part of y-coordinate of generator point in group G2

#### 8.22.3.38 CURVE\_Pybba\_ED25519

```
const BIG_256_56 CURVE_Pybba_ED25519
```

real part of y-coordinate of generator point in group G2

#### 8.22.3.39 CURVE\_Pybbb\_ED25519

```
const BIG_256_56 CURVE_Pybbb_ED25519
```

imaginary part of y-coordinate of generator point in group G2

#### 8.22.3.40 CURVE\_SB\_ED25519

```
const BIG_256_56 CURVE_SB_ED25519[2][2]
```

BN curve constant for GLV decomposition

#### 8.22.3.41 CURVE\_W\_ED25519

```
const BIG_256_56 CURVE_W_ED25519[2]
```

BN curve constant for GLV decomposition

#### 8.22.3.42 CURVE\_WB\_ED25519

```
const BIG_256_56 CURVE_WB_ED25519[4]
```

BN curve constant for GS decomposition

#### 8.22.3.43 Fra\_25519

```
const BIG_256_56 Fra_25519
```

real part of BN curve Frobenius Constant

#### 8.22.3.44 Frb\_25519

```
const BIG_256_56 Frb_25519
```

imaginary part of BN curve Frobenius Constant

## 8.23 ecp\_GOLDILOCKS.h File Reference

ECP Header File.

```
#include "fp_GOLDILOCKS.h"
#include "config_curve_GOLDILOCKS.h"
```

### Data Structures

- struct [ECP\\_GOLDILOCKS](#)  
*ECP structure - Elliptic Curve Point over base field.*

### Functions

- int [ECP\\_GOLDILOCKS\\_isinf](#) (ECP\_GOLDILOCKS \*P)  
*Tests for ECP point equal to infinity.*
- int [ECP\\_GOLDILOCKS\\_equals](#) (ECP\_GOLDILOCKS \*P, ECP\_GOLDILOCKS \*Q)  
*Tests for equality of two ECPs.*
- void [ECP\\_GOLDILOCKS\\_copy](#) (ECP\_GOLDILOCKS \*P, ECP\_GOLDILOCKS \*Q)  
*Copy ECP point to another ECP point.*
- void [ECP\\_GOLDILOCKS\\_neg](#) (ECP\_GOLDILOCKS \*P)  
*Negation of an ECP point.*
- void [ECP\\_GOLDILOCKS\\_inf](#) (ECP\_GOLDILOCKS \*P)  
*Set ECP to point-at-infinity.*
- void [ECP\\_GOLDILOCKS\\_rhs](#) (FP\_GOLDILOCKS \*r, FP\_GOLDILOCKS \*x)  
*Calculate Right Hand Side of curve equation  $y^2=f(x)$*
- int [ECP\\_GOLDILOCKS\\_set](#) (ECP\_GOLDILOCKS \*P, BIG\_448\_58 x, BIG\_448\_58 y)  
*Set ECP to point(x,y) given x and y.*
- int [ECP\\_GOLDILOCKS\\_get](#) (BIG\_448\_58 x, BIG\_448\_58 y, ECP\_GOLDILOCKS \*P)  
*Extract x and y coordinates of an ECP point P.*
- void [ECP\\_GOLDILOCKS\\_add](#) (ECP\_GOLDILOCKS \*P, ECP\_GOLDILOCKS \*Q)  
*Adds ECP instance Q to ECP instance P.*
- void [ECP\\_GOLDILOCKS\\_sub](#) (ECP\_GOLDILOCKS \*P, ECP\_GOLDILOCKS \*Q)  
*Subtracts ECP instance Q from ECP instance P.*
- int [ECP\\_GOLDILOCKS\\_setx](#) (ECP\_GOLDILOCKS \*P, BIG\_448\_58 x, int s)  
*Set ECP to point(x,y) given just x and sign of y.*
- void [ECP\\_GOLDILOCKS\\_cfp](#) (ECP\_GOLDILOCKS \*Q)  
*Multiplies Point by curve co-factor.*
- void [ECP\\_GOLDILOCKS\\_mapit](#) (ECP\_GOLDILOCKS \*Q, octet \*w)  
*Maps random BIG to curve point of correct order.*
- void [ECP\\_GOLDILOCKS\\_affine](#) (ECP\_GOLDILOCKS \*P)  
*Converts an ECP point from Projective (x,y,z) coordinates to affine (x,y) coordinates.*
- void [ECP\\_GOLDILOCKS\\_outputxyz](#) (ECP\_GOLDILOCKS \*P)  
*Formats and outputs an ECP point to the console, in projective coordinates.*
- void [ECP\\_GOLDILOCKS\\_output](#) (ECP\_GOLDILOCKS \*P)  
*Formats and outputs an ECP point to the console, converted to affine coordinates.*
- void [ECP\\_GOLDILOCKS\\_rawoutput](#) (ECP\_GOLDILOCKS \*P)  
*Formats and outputs an ECP point to the console.*

- void `ECP_GOLDILOCKS_toOctet` (octet \*S, ECP\_GOLDILOCKS \*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_GOLDILOCKS_fromOctet` (ECP\_GOLDILOCKS \*P, octet \*S)  
*Creates an ECP point from an octet string.*
- void `ECP_GOLDILOCKS_dbl` (ECP\_GOLDILOCKS \*P)  
*Doubles an ECP instance P.*
- void `ECP_GOLDILOCKS_pinmul` (ECP\_GOLDILOCKS \*P, int i, int b)  
*Multiplies an ECP instance P by a small integer, side-channel resistant.*
- void `ECP_GOLDILOCKS_mul` (ECP\_GOLDILOCKS \*P, BIG\_448\_58 b)  
*Multiplies an ECP instance P by a BIG, side-channel resistant.*
- void `ECP_GOLDILOCKS_mul2` (ECP\_GOLDILOCKS \*P, ECP\_GOLDILOCKS \*Q, BIG\_448\_58 e, BIG\_448\_58 f)  
*Calculates double multiplication  $P=e*P+f*Q$ , side-channel resistant.*
- void `ECP_GOLDILOCKS_generator` (ECP\_GOLDILOCKS \*G)  
*Get Group Generator from ROM.*

## Variables

- const int `CURVE_A_GOLDILOCKS`
- const int `CURVE_Cof_I_GOLDILOCKS`
- const int `CURVE_B_I_GOLDILOCKS`
- const `BIG_448_58 CURVE_B_GOLDILOCKS`
- const `BIG_448_58 CURVE_Order_GOLDILOCKS`
- const `BIG_448_58 CURVE_Cof_GOLDILOCKS`
- const `BIG_448_58 CURVE_Gx_GOLDILOCKS`
- const `BIG_448_58 CURVE_Gy_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pxa_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pxb_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pya_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pyb_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pxaa_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pxab_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pxba_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pxbb_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pyaa_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pyab_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pyba_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pybb_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pxaaa_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pxaab_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pxaba_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pxabbb_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pxbaa_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pxbab_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pxbba_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pxbbb_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pyaaa_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pyaab_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pyaba_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pyabb_GOLDILOCKS`
- const `BIG_448_58 CURVE_Pybaa_GOLDILOCKS`

- const [BIG\\_448\\_58 CURVE\\_Pybab\\_GOLDILOCKS](#)
- const [BIG\\_448\\_58 CURVE\\_Pybba\\_GOLDILOCKS](#)
- const [BIG\\_448\\_58 CURVE\\_Pybbb\\_GOLDILOCKS](#)
- const [BIG\\_448\\_58 CURVE\\_Bnx\\_GOLDILOCKS](#)
- const [BIG\\_448\\_58 CURVE\\_Cru\\_GOLDILOCKS](#)
- const [BIG\\_448\\_58 Fra\\_GOLDILOCKS](#)
- const [BIG\\_448\\_58 Frb\\_GOLDILOCKS](#)
- const [BIG\\_448\\_58 CURVE\\_W\\_GOLDILOCKS](#) [2]
- const [BIG\\_448\\_58 CURVE\\_SB\\_GOLDILOCKS](#) [2][2]
- const [BIG\\_448\\_58 CURVE\\_WB\\_GOLDILOCKS](#) [4]
- const [BIG\\_448\\_58 CURVE\\_BB\\_GOLDILOCKS](#) [4][4]

### 8.23.1 Detailed Description

#### Author

Mike Scott

### 8.23.2 Function Documentation

#### 8.23.2.1 ECP\_GOLDILOCKS\_add()

```
void ECP_GOLDILOCKS_add (
    ECP\_GOLDILOCKS * P,
    ECP\_GOLDILOCKS * Q )
```

##### Parameters

<i>P</i>	ECP instance, on exit =P+Q
<i>Q</i>	ECP instance to be added to P

#### 8.23.2.2 ECP\_GOLDILOCKS\_affine()

```
void ECP_GOLDILOCKS_affine (
    ECP\_GOLDILOCKS * P )
```

##### Parameters

<i>P</i>	ECP instance to be converted to affine form
----------	---

### 8.23.2.3 ECP\_GOLDILOCKS\_cfp()

```
void ECP_GOLDILOCKS_cfp (
    ECP_GOLDILOCKS * Q )
```

#### Parameters

<i>Q</i>	ECP instance
----------	--------------

### 8.23.2.4 ECP\_GOLDILOCKS\_copy()

```
void ECP_GOLDILOCKS_copy (
    ECP_GOLDILOCKS * P,
    ECP_GOLDILOCKS * Q )
```

#### Parameters

<i>P</i>	ECP instance, on exit = Q
<i>Q</i>	ECP instance to be copied

### 8.23.2.5 ECP\_GOLDILOCKS\_dbl()

```
void ECP_GOLDILOCKS_dbl (
    ECP_GOLDILOCKS * P )
```

#### Parameters

<i>P</i>	ECP instance, on exit =2*P
----------	----------------------------

### 8.23.2.6 ECP\_GOLDILOCKS\_equals()

```
int ECP_GOLDILOCKS_equals (
    ECP_GOLDILOCKS * P,
    ECP_GOLDILOCKS * 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

**8.23.2.7 ECP\_GOLDILOCKS\_fromOctet()**

```
int ECP_GOLDILOCKS_fromOctet (
    ECP_GOLDILOCKS * 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

**8.23.2.8 ECP\_GOLDILOCKS\_generator()**

```
void ECP_GOLDILOCKS_generator (
    ECP_GOLDILOCKS * G )
```

**Parameters**

<i>G</i>	ECP instance
----------	--------------

**8.23.2.9 ECP\_GOLDILOCKS\_get()**

```
int ECP_GOLDILOCKS_get (
    BIG_448_58 x,
    BIG_448_58 y,
    ECP_GOLDILOCKS * 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

**8.23.2.10 ECP\_GOLDILOCKS\_inf()**

```
void ECP_GOLDILOCKS_inf (
    ECP_GOLDILOCKS *  $P$  )
```

**Parameters**

$P$	ECP instance to be set to infinity
-----	------------------------------------

**8.23.2.11 ECP\_GOLDILOCKS\_isinf()**

```
int ECP_GOLDILOCKS_isinf (
    ECP_GOLDILOCKS *  $P$  )
```

**Parameters**

$P$	ECP point to be tested
-----	------------------------

**Returns**

1 if infinity, else returns 0

**8.23.2.12 ECP\_GOLDILOCKS\_mapit()**

```
void ECP_GOLDILOCKS_mapit (
    ECP_GOLDILOCKS *  $Q$ ,
    octet *  $w$  )
```

**Parameters**

$Q$	ECP instance of correct order
$w$	OCTET byte array to be mapped

**8.23.2.13 ECP\_GOLDILOCKS\_mul()**

```
void ECP_GOLDILOCKS_mul (
```



```

ECP_GOLDILOCKS * P,
BIG_448_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

#### 8.23.2.14 ECP\_GOLDILOCKS\_mul2()

```

void ECP_GOLDILOCKS_mul2 (
    ECP_GOLDILOCKS * P,
    ECP_GOLDILOCKS * Q,
    BIG_448_58 e,
    BIG_448_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

#### 8.23.2.15 ECP\_GOLDILOCKS\_neg()

```

void ECP_GOLDILOCKS_neg (
    ECP_GOLDILOCKS * P )

```

#### Parameters

<i>P</i>	ECP instance, on exit = -P
----------	----------------------------

#### 8.23.2.16 ECP\_GOLDILOCKS\_output()

```

void ECP_GOLDILOCKS_output (
    ECP_GOLDILOCKS * P )

```

#### Parameters

<i>P</i>	ECP instance to be printed
----------	----------------------------

### 8.23.2.17 ECP\_GOLDILOCKS\_outputxyz()

```
void ECP_GOLDILOCKS_outputxyz (
    ECP_GOLDILOCKS * P )
```

#### Parameters

<i>P</i>	ECP instance to be printed
----------	----------------------------

### 8.23.2.18 ECP\_GOLDILOCKS\_pinmul()

```
void ECP_GOLDILOCKS_pinmul (
    ECP_GOLDILOCKS * 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

### 8.23.2.19 ECP\_GOLDILOCKS\_rawoutput()

```
void ECP_GOLDILOCKS_rawoutput (
    ECP_GOLDILOCKS * P )
```

#### Parameters

<i>P</i>	ECP instance to be printed
----------	----------------------------

### 8.23.2.20 ECP\_GOLDILOCKS\_rhs()

```
void ECP_GOLDILOCKS_rhs (
    FP_GOLDILOCKS * r,
    FP_GOLDILOCKS * 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 $x$

## 8.23.2.21 ECP\_GOLDILOCKS\_set()

```
int ECP_GOLDILOCKS_set (
    ECP_GOLDILOCKS * P,
    BIG_448_58 x,
    BIG_448_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

## 8.23.2.22 ECP\_GOLDILOCKS\_setx()

```
int ECP_GOLDILOCKS_setx (
    ECP_GOLDILOCKS * P,
    BIG_448_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.

### 8.23.2.23 ECP\_GOLDILOCKS\_sub()

```
void ECP_GOLDILOCKS_sub (
    ECP_GOLDILOCKS * P,
    ECP_GOLDILOCKS * Q )
```

#### Parameters

<i>P</i>	ECP instance, on exit =P-Q
<i>Q</i>	ECP instance to be subtracted from P

### 8.23.2.24 ECP\_GOLDILOCKS\_toOctet()

```
void ECP_GOLDILOCKS_toOctet (
    octet * S,
    ECP_GOLDILOCKS * 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

## 8.23.3 Variable Documentation

### 8.23.3.1 CURVE\_A\_GOLDILOCKS

```
const int CURVE_A_GOLDILOCKS
```

Elliptic curve A parameter

### 8.23.3.2 CURVE\_B\_GOLDILOCKS

```
const BIG_448_58 CURVE_B_GOLDILOCKS
```

Elliptic curve B parameter

### 8.23.3.3 CURVE\_B\_I\_GOLDILOCKS

```
const int CURVE_B_I_GOLDILOCKS
```

Elliptic curve B\_i parameter

#### 8.23.3.4 CURVE\_BB\_GOLDILOCKS

```
const BIG_448_58 CURVE_BB_GOLDILOCKS[4][4]
```

BN curve constant for GS decomposition

#### 8.23.3.5 CURVE\_Bnx\_GOLDILOCKS

```
const BIG_448_58 CURVE_Bnx_GOLDILOCKS
```

BN curve x parameter

#### 8.23.3.6 CURVE\_Cof\_GOLDILOCKS

```
const BIG_448_58 CURVE_Cof_GOLDILOCKS
```

Elliptic curve cofactor

#### 8.23.3.7 CURVE\_Cof\_I\_GOLDILOCKS

```
const int CURVE_Cof_I_GOLDILOCKS
```

Elliptic curve cofactor

#### 8.23.3.8 CURVE\_Cru\_GOLDILOCKS

```
const BIG_448_58 CURVE_Cru_GOLDILOCKS
```

BN curve Cube Root of Unity

#### 8.23.3.9 CURVE\_Gx\_GOLDILOCKS

```
const BIG_448_58 CURVE_Gx_GOLDILOCKS
```

x-coordinate of generator point in group G1

#### 8.23.3.10 CURVE\_Gy\_GOLDILOCKS

```
const BIG_448_58 CURVE_Gy_GOLDILOCKS
```

y-coordinate of generator point in group G1

#### 8.23.3.11 CURVE\_Order\_GOLDILOCKS

```
const BIG_448_58 CURVE_Order_GOLDILOCKS
```

Elliptic curve group order

**8.23.3.12 CURVE\_Pxa\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pxa_GOLDILOCKS
```

real part of x-coordinate of generator point in group G2

**8.23.3.13 CURVE\_Pxaa\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pxaa_GOLDILOCKS
```

real part of x-coordinate of generator point in group G2

**8.23.3.14 CURVE\_Pxaaa\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pxaaa_GOLDILOCKS
```

real part of x-coordinate of generator point in group G2

**8.23.3.15 CURVE\_Pxaab\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pxaab_GOLDILOCKS
```

imaginary part of x-coordinate of generator point in group G2

**8.23.3.16 CURVE\_Pxab\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pxab_GOLDILOCKS
```

imaginary part of x-coordinate of generator point in group G2

**8.23.3.17 CURVE\_Pxaba\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pxaba_GOLDILOCKS
```

real part of x-coordinate of generator point in group G2

**8.23.3.18 CURVE\_Pxab\_b\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pxab_b_GOLDILOCKS
```

imaginary part of x-coordinate of generator point in group G2

**8.23.3.19 CURVE\_Pxb\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pxb_GOLDILOCKS
```

imaginary part of x-coordinate of generator point in group G2

**8.23.3.20 CURVE\_Pxba\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pxba_GOLDILOCKS
```

real part of x-coordinate of generator point in group G2

**8.23.3.21 CURVE\_Pxbaa\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pxbaa_GOLDILOCKS
```

real part of x-coordinate of generator point in group G2

**8.23.3.22 CURVE\_Pxbab\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pxbab_GOLDILOCKS
```

imaginary part of x-coordinate of generator point in group G2

**8.23.3.23 CURVE\_Pxbb\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pxbb_GOLDILOCKS
```

imaginary part of x-coordinate of generator point in group G2

**8.23.3.24 CURVE\_Pxbba\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pxbba_GOLDILOCKS
```

real part of x-coordinate of generator point in group G2

**8.23.3.25 CURVE\_Pxbbb\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pxbbb_GOLDILOCKS
```

imaginary part of x-coordinate of generator point in group G2

**8.23.3.26 CURVE\_Pya\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pya_GOLDILOCKS
```

real part of y-coordinate of generator point in group G2

**8.23.3.27 CURVE\_Pyaa\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pyaa_GOLDILOCKS
```

real part of y-coordinate of generator point in group G2

**8.23.3.28 CURVE\_Pyaaa\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pyaaa_GOLDILOCKS
```

real part of y-coordinate of generator point in group G2

**8.23.3.29 CURVE\_Pyaab\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pyaab_GOLDILOCKS
```

imaginary part of y-coordinate of generator point in group G2

**8.23.3.30 CURVE\_Pyab\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pyab_GOLDILOCKS
```

imaginary part of y-coordinate of generator point in group G2

**8.23.3.31 CURVE\_Pyaba\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pyaba_GOLDILOCKS
```

real part of y-coordinate of generator point in group G2

**8.23.3.32 CURVE\_Pyabb\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pyabb_GOLDILOCKS
```

imaginary part of y-coordinate of generator point in group G2

**8.23.3.33 CURVE\_Pyb\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pyb_GOLDILOCKS
```

imaginary part of y-coordinate of generator point in group G2

**8.23.3.34 CURVE\_Pyba\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pyba_GOLDILOCKS
```

real part of y-coordinate of generator point in group G2

**8.23.3.35 CURVE\_Pybaa\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pybaa_GOLDILOCKS
```

real part of y-coordinate of generator point in group G2



**8.23.3.36 CURVE\_Pybab\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pybab_GOLDILOCKS
```

imaginary part of y-coordinate of generator point in group G2

**8.23.3.37 CURVE\_Pybb\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pybb_GOLDILOCKS
```

imaginary part of y-coordinate of generator point in group G2

**8.23.3.38 CURVE\_Pybba\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pybba_GOLDILOCKS
```

real part of y-coordinate of generator point in group G2

**8.23.3.39 CURVE\_Pybbb\_GOLDILOCKS**

```
const BIG_448_58 CURVE_Pybbb_GOLDILOCKS
```

imaginary part of y-coordinate of generator point in group G2

**8.23.3.40 CURVE\_SB\_GOLDILOCKS**

```
const BIG_448_58 CURVE_SB_GOLDILOCKS[2][2]
```

BN curve constant for GLV decomposition

**8.23.3.41 CURVE\_W\_GOLDILOCKS**

```
const BIG_448_58 CURVE_W_GOLDILOCKS[2]
```

BN curve constant for GLV decomposition

**8.23.3.42 CURVE\_WB\_GOLDILOCKS**

```
const BIG_448_58 CURVE_WB_GOLDILOCKS[4]
```

BN curve constant for GS decomposition

**8.23.3.43 Fra\_GOLDILOCKS**

```
const BIG_448_58 Fra_GOLDILOCKS
```

real part of BN curve Frobenius Constant

### 8.23.3.44 Frb\_GOLDILOCKS

```
const BIG_448_58 Frb_GOLDILOCKS
```

imaginary part of BN curve Frobenius Constant

## 8.24 ecp\_NIST256.h File Reference

ECP Header File.

```
#include "fp_NIST256.h"
#include "config_curve_NIST256.h"
```

### Data Structures

- struct [ECP\\_NIST256](#)  
*ECP structure - Elliptic Curve Point over base field.*

### Functions

- int [ECP\\_NIST256\\_isinf](#) ([ECP\\_NIST256](#) \*P)  
*Tests for ECP point equal to infinity.*
- int [ECP\\_NIST256\\_equals](#) ([ECP\\_NIST256](#) \*P, [ECP\\_NIST256](#) \*Q)  
*Tests for equality of two ECPs.*
- void [ECP\\_NIST256\\_copy](#) ([ECP\\_NIST256](#) \*P, [ECP\\_NIST256](#) \*Q)  
*Copy ECP point to another ECP point.*
- void [ECP\\_NIST256\\_neg](#) ([ECP\\_NIST256](#) \*P)  
*Negation of an ECP point.*
- void [ECP\\_NIST256\\_inf](#) ([ECP\\_NIST256](#) \*P)  
*Set ECP to point-at-infinity.*
- void [ECP\\_NIST256\\_rhs](#) ([FP\\_NIST256](#) \*r, [FP\\_NIST256](#) \*x)  
*Calculate Right Hand Side of curve equation  $y^2=f(x)$*
- int [ECP\\_NIST256\\_set](#) ([ECP\\_NIST256](#) \*P, [BIG\\_256\\_56](#) x, [BIG\\_256\\_56](#) y)  
*Set ECP to point(x,y) given x and y.*
- int [ECP\\_NIST256\\_get](#) ([BIG\\_256\\_56](#) x, [BIG\\_256\\_56](#) y, [ECP\\_NIST256](#) \*P)  
*Extract x and y coordinates of an ECP point P.*
- void [ECP\\_NIST256\\_add](#) ([ECP\\_NIST256](#) \*P, [ECP\\_NIST256](#) \*Q)  
*Adds ECP instance Q to ECP instance P.*
- void [ECP\\_NIST256\\_sub](#) ([ECP\\_NIST256](#) \*P, [ECP\\_NIST256](#) \*Q)  
*Subtracts ECP instance Q from ECP instance P.*
- int [ECP\\_NIST256\\_setx](#) ([ECP\\_NIST256](#) \*P, [BIG\\_256\\_56](#) x, int s)  
*Set ECP to point(x,y) given just x and sign of y.*
- void [ECP\\_NIST256\\_cfp](#) ([ECP\\_NIST256](#) \*Q)  
*Multiplies Point by curve co-factor.*
- void [ECP\\_NIST256\\_mapit](#) ([ECP\\_NIST256](#) \*Q, [octet](#) \*w)  
*Maps random BIG to curve point of correct order.*
- void [ECP\\_NIST256\\_affine](#) ([ECP\\_NIST256](#) \*P)

- Converts an ECP point from Projective (x,y,z) coordinates to affine (x,y) coordinates.*

  - void [ECP\\_NIST256\\_outputxyz](#) ([ECP\\_NIST256](#) \*P)

*Formats and outputs an ECP point to the console, in projective coordinates.*

  - void [ECP\\_NIST256\\_output](#) ([ECP\\_NIST256](#) \*P)

*Formats and outputs an ECP point to the console, converted to affine coordinates.*

  - void [ECP\\_NIST256\\_rawoutput](#) ([ECP\\_NIST256](#) \*P)

*Formats and outputs an ECP point to the console.*

  - void [ECP\\_NIST256\\_toOctet](#) (octet \*S, [ECP\\_NIST256](#) \*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\\_NIST256\\_fromOctet](#) ([ECP\\_NIST256](#) \*P, octet \*S)

*Creates an ECP point from an octet string.*

  - void [ECP\\_NIST256\\_dbl](#) ([ECP\\_NIST256](#) \*P)

*Doubles an ECP instance P.*

  - void [ECP\\_NIST256\\_pinmul](#) ([ECP\\_NIST256](#) \*P, int i, int b)

*Multiplies an ECP instance P by a small integer, side-channel resistant.*

  - void [ECP\\_NIST256\\_mul](#) ([ECP\\_NIST256](#) \*P, [BIG\\_256\\_56](#) b)

*Multiplies an ECP instance P by a BIG, side-channel resistant.*

  - void [ECP\\_NIST256\\_mul2](#) ([ECP\\_NIST256](#) \*P, [ECP\\_NIST256](#) \*Q, [BIG\\_256\\_56](#) e, [BIG\\_256\\_56](#) f)

*Calculates double multiplication  $P=e*P+f*Q$ , side-channel resistant.*

  - void [ECP\\_NIST256\\_generator](#) ([ECP\\_NIST256](#) \*G)

*Get Group Generator from ROM.*

## Variables

- const int [CURVE\\_A\\_NIST256](#)
- const int [CURVE\\_Cof\\_I\\_NIST256](#)
- const int [CURVE\\_B\\_I\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_B\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Order\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Cof\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Gx\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Gy\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Pxa\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Pxb\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Pya\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Pyb\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Pxaa\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Pxab\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Pxba\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Pxbb\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Pyaa\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Pyab\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Pyba\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Pybb\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Pxaaa\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Pxaab\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Pxaba\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Pxabb\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Pxbaa\\_NIST256](#)
- const [BIG\\_256\\_56](#) [CURVE\\_Pxbab\\_NIST256](#)

- const [BIG\\_256\\_56\\_CURVE\\_Pxbba\\_NIST256](#)
- const [BIG\\_256\\_56\\_CURVE\\_Pxbbb\\_NIST256](#)
- const [BIG\\_256\\_56\\_CURVE\\_Pyaaa\\_NIST256](#)
- const [BIG\\_256\\_56\\_CURVE\\_Pyaab\\_NIST256](#)
- const [BIG\\_256\\_56\\_CURVE\\_Pyaba\\_NIST256](#)
- const [BIG\\_256\\_56\\_CURVE\\_Pyabb\\_NIST256](#)
- const [BIG\\_256\\_56\\_CURVE\\_Pybaa\\_NIST256](#)
- const [BIG\\_256\\_56\\_CURVE\\_Pybab\\_NIST256](#)
- const [BIG\\_256\\_56\\_CURVE\\_Pybba\\_NIST256](#)
- const [BIG\\_256\\_56\\_CURVE\\_Pybbb\\_NIST256](#)
- const [BIG\\_256\\_56\\_CURVE\\_Bnx\\_NIST256](#)
- const [BIG\\_256\\_56\\_CURVE\\_Cru\\_NIST256](#)
- const [BIG\\_256\\_56\\_Fra\\_NIST256](#)
- const [BIG\\_256\\_56\\_Frb\\_NIST256](#)
- const [BIG\\_256\\_56\\_CURVE\\_W\\_NIST256](#) [2]
- const [BIG\\_256\\_56\\_CURVE\\_SB\\_NIST256](#) [2][2]
- const [BIG\\_256\\_56\\_CURVE\\_WB\\_NIST256](#) [4]
- const [BIG\\_256\\_56\\_CURVE\\_BB\\_NIST256](#) [4][4]

### 8.24.1 Detailed Description

#### Author

Mike Scott

### 8.24.2 Function Documentation

#### 8.24.2.1 ECP\_NIST256\_add()

```
void ECP_NIST256_add (
    ECP\_NIST256 * P,
    ECP\_NIST256 * Q )
```

#### Parameters

<i>P</i>	ECP instance, on exit =P+Q
<i>Q</i>	ECP instance to be added to P

#### 8.24.2.2 ECP\_NIST256\_affine()

```
void ECP_NIST256_affine (
    ECP\_NIST256 * P )
```

## Parameters

<i>P</i>	ECP instance to be converted to affine form
----------	---

## 8.24.2.3 ECP\_NIST256\_cfp()

```
void ECP_NIST256_cfp (
    ECP_NIST256 * Q )
```

## Parameters

<i>Q</i>	ECP instance
----------	--------------

## 8.24.2.4 ECP\_NIST256\_copy()

```
void ECP_NIST256_copy (
    ECP_NIST256 * P,
    ECP_NIST256 * Q )
```

## Parameters

<i>P</i>	ECP instance, on exit = <i>Q</i>
<i>Q</i>	ECP instance to be copied

## 8.24.2.5 ECP\_NIST256\_dbl()

```
void ECP_NIST256_dbl (
    ECP_NIST256 * P )
```

## Parameters

<i>P</i>	ECP instance, on exit =2* <i>P</i>
----------	------------------------------------

## 8.24.2.6 ECP\_NIST256\_equals()

```
int ECP_NIST256_equals (
    ECP_NIST256 * P,
    ECP_NIST256 * 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

**8.24.2.7 ECP\_NIST256\_fromOctet()**

```
int ECP_NIST256_fromOctet (
    ECP_NIST256 * 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

**8.24.2.8 ECP\_NIST256\_generator()**

```
void ECP_NIST256_generator (
    ECP_NIST256 * G )
```

**Parameters**

<i>G</i>	ECP instance
----------	--------------

**8.24.2.9 ECP\_NIST256\_get()**

```
int ECP_NIST256_get (
    BIG_256_56 x,
    BIG_256_56 y,
    ECP_NIST256 * P )
```

If x=y, returns only x

## Parameters

$x$	BIG on exit = x coordinate of point
$y$	BIG on exit = y coordinate of point (unless $x=y$ )
$P$	ECP instance (x,y)

## Returns

sign of y, or -1 if P is point-at-infinity

## 8.24.2.10 ECP\_NIST256\_inf()

```
void ECP_NIST256_inf (
    ECP_NIST256 * P )
```

## Parameters

$P$	ECP instance to be set to infinity
-----	------------------------------------

## 8.24.2.11 ECP\_NIST256\_isinf()

```
int ECP_NIST256_isinf (
    ECP_NIST256 * P )
```

## Parameters

$P$	ECP point to be tested
-----	------------------------

## Returns

1 if infinity, else returns 0

## 8.24.2.12 ECP\_NIST256\_mapit()

```
void ECP_NIST256_mapit (
    ECP_NIST256 * Q,
    octet * w )
```

## Parameters

$Q$	ECP instance of correct order
$w$	OCTET byte array to be mapped

### 8.24.2.13 ECP\_NIST256\_mul()

```
void ECP_NIST256_mul (
    ECP_NIST256 * P,
    BIG_256_56 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

### 8.24.2.14 ECP\_NIST256\_mul2()

```
void ECP_NIST256_mul2 (
    ECP_NIST256 * P,
    ECP_NIST256 * Q,
    BIG_256_56 e,
    BIG_256_56 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

### 8.24.2.15 ECP\_NIST256\_neg()

```
void ECP_NIST256_neg (
    ECP_NIST256 * P )
```

#### Parameters

<i>P</i>	ECP instance, on exit = -P
----------	----------------------------

### 8.24.2.16 ECP\_NIST256\_output()

```
void ECP_NIST256_output (
```



```
ECP_NIST256 * P )
```

**Parameters**

<i>P</i>	ECP instance to be printed
----------	----------------------------

**8.24.2.17 ECP\_NIST256\_outputxyz()**

```
void ECP_NIST256_outputxyz (
    ECP_NIST256 * P )
```

**Parameters**

<i>P</i>	ECP instance to be printed
----------	----------------------------

**8.24.2.18 ECP\_NIST256\_pinmul()**

```
void ECP_NIST256_pinmul (
    ECP_NIST256 * 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

**8.24.2.19 ECP\_NIST256\_rawoutput()**

```
void ECP_NIST256_rawoutput (
    ECP_NIST256 * P )
```

**Parameters**

<i>P</i>	ECP instance to be printed
----------	----------------------------

**8.24.2.20 ECP\_NIST256\_rhs()**

```
void ECP_NIST256_rhs (
    FP_NIST256 * r,
    FP_NIST256 * 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

**8.24.2.21 ECP\_NIST256\_set()**

```
int ECP_NIST256_set (
    ECP_NIST256 * P,
    BIG_256_56 x,
    BIG_256_56 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

**8.24.2.22 ECP\_NIST256\_setx()**

```
int ECP_NIST256_setx (
    ECP_NIST256 * P,
    BIG_256_56 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.

#### 8.24.2.23 ECP\_NIST256\_sub()

```
void ECP_NIST256_sub (
    ECP_NIST256 * P,
    ECP_NIST256 * Q )
```

##### Parameters

<i>P</i>	ECP instance, on exit =P-Q
<i>Q</i>	ECP instance to be subtracted from P

#### 8.24.2.24 ECP\_NIST256\_toOctet()

```
void ECP_NIST256_toOctet (
    octet * S,
    ECP_NIST256 * 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

### 8.24.3 Variable Documentation

#### 8.24.3.1 CURVE\_A\_NIST256

```
const int CURVE_A_NIST256
```

Elliptic curve A parameter

#### 8.24.3.2 CURVE\_B\_I\_NIST256

```
const int CURVE_B_I_NIST256
```

Elliptic curve B<sub>i</sub> parameter

#### 8.24.3.3 CURVE\_B\_NIST256

```
const BIG_256_56 CURVE_B_NIST256
```

Elliptic curve B parameter

#### 8.24.3.4 CURVE\_BB\_NIST256

```
const BIG_256_56 CURVE_BB_NIST256[4][4]
```

BN curve constant for GS decomposition

#### 8.24.3.5 CURVE\_Bnx\_NIST256

```
const BIG_256_56 CURVE_Bnx_NIST256
```

BN curve x parameter

#### 8.24.3.6 CURVE\_Cof\_I\_NIST256

```
const int CURVE_Cof_I_NIST256
```

Elliptic curve cofactor

#### 8.24.3.7 CURVE\_Cof\_NIST256

```
const BIG_256_56 CURVE_Cof_NIST256
```

Elliptic curve cofactor

#### 8.24.3.8 CURVE\_Cru\_NIST256

```
const BIG_256_56 CURVE_Cru_NIST256
```

BN curve Cube Root of Unity

#### 8.24.3.9 CURVE\_Gx\_NIST256

```
const BIG_256_56 CURVE_Gx_NIST256
```

x-coordinate of generator point in group G1

#### 8.24.3.10 CURVE\_Gy\_NIST256

```
const BIG_256_56 CURVE_Gy_NIST256
```

y-coordinate of generator point in group G1

#### 8.24.3.11 CURVE\_Order\_NIST256

```
const BIG_256_56 CURVE_Order_NIST256
```

Elliptic curve group order

#### 8.24.3.12 CURVE\_Pxa\_NIST256

```
const BIG_256_56 CURVE_Pxa_NIST256
```

real part of x-coordinate of generator point in group G2

#### 8.24.3.13 CURVE\_Pxaa\_NIST256

```
const BIG_256_56 CURVE_Pxaa_NIST256
```

real part of x-coordinate of generator point in group G2

#### 8.24.3.14 CURVE\_Pxaaa\_NIST256

```
const BIG_256_56 CURVE_Pxaaa_NIST256
```

real part of x-coordinate of generator point in group G2

#### 8.24.3.15 CURVE\_Pxaab\_NIST256

```
const BIG_256_56 CURVE_Pxaab_NIST256
```

imaginary part of x-coordinate of generator point in group G2

#### 8.24.3.16 CURVE\_Pxab\_NIST256

```
const BIG_256_56 CURVE_Pxab_NIST256
```

imaginary part of x-coordinate of generator point in group G2

#### 8.24.3.17 CURVE\_Pxaba\_NIST256

```
const BIG_256_56 CURVE_Pxaba_NIST256
```

real part of x-coordinate of generator point in group G2

#### 8.24.3.18 CURVE\_Pxabb\_NIST256

```
const BIG_256_56 CURVE_Pxabb_NIST256
```

imaginary part of x-coordinate of generator point in group G2

**8.24.3.19 CURVE\_Pxb\_NIST256**

const [BIG\\_256\\_56](#) CURVE\_Pxb\_NIST256

imaginary part of x-coordinate of generator point in group G2

**8.24.3.20 CURVE\_Pxba\_NIST256**

const [BIG\\_256\\_56](#) CURVE\_Pxba\_NIST256

real part of x-coordinate of generator point in group G2

**8.24.3.21 CURVE\_Pxaa\_NIST256**

const [BIG\\_256\\_56](#) CURVE\_Pxaa\_NIST256

real part of x-coordinate of generator point in group G2

**8.24.3.22 CURVE\_Pxbab\_NIST256**

const [BIG\\_256\\_56](#) CURVE\_Pxbab\_NIST256

imaginary part of x-coordinate of generator point in group G2

**8.24.3.23 CURVE\_Pxbb\_NIST256**

const [BIG\\_256\\_56](#) CURVE\_Pxbb\_NIST256

imaginary part of x-coordinate of generator point in group G2

**8.24.3.24 CURVE\_Pxbba\_NIST256**

const [BIG\\_256\\_56](#) CURVE\_Pxbba\_NIST256

real part of x-coordinate of generator point in group G2

**8.24.3.25 CURVE\_Pxbbb\_NIST256**

const [BIG\\_256\\_56](#) CURVE\_Pxbbb\_NIST256

imaginary part of x-coordinate of generator point in group G2

**8.24.3.26 CURVE\_Pya\_NIST256**

const [BIG\\_256\\_56](#) CURVE\_Pya\_NIST256

real part of y-coordinate of generator point in group G2

**8.24.3.27 CURVE\_Pyaa\_NIST256**

```
const BIG_256_56 CURVE_Pyaa_NIST256
```

real part of y-coordinate of generator point in group G2

**8.24.3.28 CURVE\_Pyaaa\_NIST256**

```
const BIG_256_56 CURVE_Pyaaa_NIST256
```

real part of y-coordinate of generator point in group G2

**8.24.3.29 CURVE\_Pyaab\_NIST256**

```
const BIG_256_56 CURVE_Pyaab_NIST256
```

imaginary part of y-coordinate of generator point in group G2

**8.24.3.30 CURVE\_Pyab\_NIST256**

```
const BIG_256_56 CURVE_Pyab_NIST256
```

imaginary part of y-coordinate of generator point in group G2

**8.24.3.31 CURVE\_Pyaba\_NIST256**

```
const BIG_256_56 CURVE_Pyaba_NIST256
```

real part of y-coordinate of generator point in group G2

**8.24.3.32 CURVE\_Pyabb\_NIST256**

```
const BIG_256_56 CURVE_Pyabb_NIST256
```

imaginary part of y-coordinate of generator point in group G2

**8.24.3.33 CURVE\_Pyb\_NIST256**

```
const BIG_256_56 CURVE_Pyb_NIST256
```

imaginary part of y-coordinate of generator point in group G2

**8.24.3.34 CURVE\_Pyba\_NIST256**

```
const BIG_256_56 CURVE_Pyba_NIST256
```

real part of y-coordinate of generator point in group G2

**8.24.3.35 CURVE\_Pybaa\_NIST256**

```
const BIG_256_56 CURVE_Pybaa_NIST256
```

real part of y-coordinate of generator point in group G2

**8.24.3.36 CURVE\_Pybab\_NIST256**

```
const BIG_256_56 CURVE_Pybab_NIST256
```

imaginary part of y-coordinate of generator point in group G2

**8.24.3.37 CURVE\_Pybb\_NIST256**

```
const BIG_256_56 CURVE_Pybb_NIST256
```

imaginary part of y-coordinate of generator point in group G2

**8.24.3.38 CURVE\_Pybba\_NIST256**

```
const BIG_256_56 CURVE_Pybba_NIST256
```

real part of y-coordinate of generator point in group G2

**8.24.3.39 CURVE\_Pybbb\_NIST256**

```
const BIG_256_56 CURVE_Pybbb_NIST256
```

imaginary part of y-coordinate of generator point in group G2

**8.24.3.40 CURVE\_SB\_NIST256**

```
const BIG_256_56 CURVE_SB_NIST256[2][2]
```

BN curve constant for GLV decomposition

**8.24.3.41 CURVE\_W\_NIST256**

```
const BIG_256_56 CURVE_W_NIST256[2]
```

BN curve constant for GLV decomposition

**8.24.3.42 CURVE\_WB\_NIST256**

```
const BIG_256_56 CURVE_WB_NIST256[4]
```

BN curve constant for GS decomposition



## 8.24.3.43 Fra\_NIST256

```
const BIG_256_56 Fra_NIST256
```

real part of BN curve Frobenius Constant

## 8.24.3.44 Frb\_NIST256

```
const BIG_256_56 Frb_NIST256
```

imaginary part of BN curve Frobenius Constant

## 8.25 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 *m`, 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_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 *m`, int `n`)  
*Calculate  $r = x^e \bmod m$ , side channel resistant.*
- void `FF_2048_skspow` (`BIG_1024_58 *r`, `BIG_1024_58 *x`, `BIG_1024_58 e`, `BIG_1024_58 *m`, int `n`)  
*Calculate  $r = x^e \bmod m$ , side channel resistant.*
- void `FF_2048_power` (`BIG_1024_58 *r`, `BIG_1024_58 *x`, int `e`, `BIG_1024_58 *m`, int `n`)  
*Calculate  $r = x^e \bmod m$ .*
- void `FF_2048_pow` (`BIG_1024_58 *r`, `BIG_1024_58 *x`, `BIG_1024_58 *e`, `BIG_1024_58 *m`, int `n`)  
*Calculate  $r = x^e \bmod m$ .*
- 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_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  $r = x^e \cdot y^f \bmod m$ .*

## 8.25.1 Detailed Description

### Author

Mike Scott

## 8.25.2 Macro Definition Documentation

### 8.25.2.1 HFLEN\_2048

```
#define HFLEN_2048 (FFLEN_2048/2)
```

Useful for half-size RSA private key operations

### 8.25.2.2 P\_EXCESS\_2048

```
#define P_EXCESS_2048(  
    a ) (((a[NLEN_1024_58-1])>>(P_TBITS_2048))+1)
```

TODO

### 8.25.2.3 P\_FEXCESS\_2048

```
#define P_FEXCESS_2048 ((chunk)1<<(BASEBITS_1024_58*NLEN_1024_58-P_MBITS_2048-1))
```

TODO

### 8.25.2.4 P\_MBITS\_2048

```
#define P_MBITS_2048 (MODBYTES_1024_58*8)
```

Number of bits in modulus

### 8.25.2.5 P\_TBITS\_2048

```
#define P_TBITS_2048 (P_MBITS_2048%BASEBITS_1024_58)
```

TODO

## 8.25.3 Function Documentation

### 8.25.3.1 FF\_2048\_add()

```
void FF_2048_add (  
    BIG_1024_58 * x,  
    BIG_1024_58 * y,  
    BIG_1024_58 * z,  
    int n )
```

**Parameters**

$x$	FF instance, on exit = $y+z$
$y$	FF instance
$z$	FF instance
$n$	size of FF in BIGs

**8.25.3.2 FF\_2048\_cfactor()**

```
int FF_2048_cfactor (
    BIG_1024_58 * x,
    sign32 s,
    int n )
```

**Parameters**

$x$	FF instance to be tested
$s$	the supplied integer
$n$	size of FF in BIGs

**Returns**

1 if  $\gcd(x,s) \neq 1$ , else return 0

**8.25.3.3 FF\_2048\_comp()**

```
int FF_2048_comp (
    BIG_1024_58 * x,
    BIG_1024_58 * y,
    int n )
```

**Parameters**

$x$	first FF number to be compared
$y$	second FF number to be compared
$n$	size of FF in BIGs

**Returns**

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

## 8.25.3.4 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

## 8.25.3.5 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

## 8.25.3.6 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 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

### 8.25.3.7 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

### 8.25.3.8 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

### 8.25.3.9 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

### 8.25.3.10 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 mod z
<i>y</i>	FF instance
<i>z</i>	FF prime modulus
<i>n</i>	size of FF in BIGs

### 8.25.3.11 FF\_2048\_iszilch()

```
int FF_2048_iszilch (
    BIG_1024_58 * 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

### 8.25.3.12 FF\_2048\_lastbits()

```
int FF_2048_lastbits (
    BIG_1024_58 * 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

## 8.25.3.13 FF\_2048\_mod()

```
void FF_2048_mod (
    BIG_1024_58 * x,
    BIG_1024_58 * m,
    int n )
```

This is slow

## Parameters

$x$	FF instance to be reduced mod $m$ - on exit = $x \bmod m$
$m$	FF modulus
$n$	size of FF in BIGs

## 8.25.3.14 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

$x$	FF instance, on exit = $y*z$
$y$	FF instance
$z$	FF instance
$n$	size of FF in BIGs

## 8.25.3.15 FF\_2048\_norm()

```
void FF_2048_norm (
    BIG_1024_58 * x,
    int n )
```

## Parameters

$x$	FF instance to be normalised
$n$	size of FF in BIGs



### 8.25.3.16 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

### 8.25.3.17 FF\_2048\_output()

```
void FF_2048_output (
    BIG_1024_58 * x,
    int n )
```

#### Parameters

<i>x</i>	FF instance to be printed
<i>n</i>	size of FF in BIGs

### 8.25.3.18 FF\_2048\_parity()

```
int FF_2048_parity (
    BIG_1024_58 * x )
```

#### Parameters

<i>x</i>	FF number
----------	-----------

#### Returns

0 or 1

### 8.25.3.19 FF\_2048\_pow()

```
void FF_2048_pow (
    BIG_1024_58 * r,
    BIG_1024_58 * x,
    BIG_1024_58 * e,
    BIG_1024_58 * m,
    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>m</i>	FF modulus
<i>n</i>	size of FF in BIGs

## 8.25.3.20 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

## 8.25.3.21 FF\_2048\_power()

```
void FF_2048_power (
    BIG_1024_58 * r,
    BIG_1024_58 * x,
    int e,
    BIG_1024_58 * m,
    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>m</i>	FF modulus
<i>n</i>	size of FF in BIGs

### 8.25.3.22 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

### 8.25.3.23 FF\_2048\_random()

```
void FF_2048_random (
    BIG_1024_58 * 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

### 8.25.3.24 FF\_2048\_randomnum()

```
void FF_2048_randomnum (
    BIG_1024_58 * x,
    BIG_1024_58 * y,
    csprng * R,
    int n )
```

#### Parameters

<i>x</i>	FF instance, on exit <i>x</i> is a random number < <i>y</i>
----------	---

## Parameters

$y$	FF instance, the modulus
$R$	an instance of a Cryptographically Secure Random Number Generator
$n$	size of FF in BIGs

## 8.25.3.25 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

## 8.25.3.26 FF\_2048\_shl()

```
void FF_2048_shl (
    BIG_1024_58 * x,
    int n )
```

## Parameters

$x$	FF instance to be shifted left
$n$	size of FF in BIGs

## 8.25.3.27 FF\_2048\_shr()

```
void FF_2048_shr (
    BIG_1024_58 * x,
    int n )
```

## Parameters

$x$	FF instance to be shifted right
$n$	size of FF in BIGs

### 8.25.3.28 FF\_2048\_skpow()

```
void FF_2048_skpow (
    BIG_1024_58 * r,
    BIG_1024_58 * x,
    BIG_1024_58 * e,
    BIG_1024_58 * m,
    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>m</i>	FF modulus
<i>n</i>	size of FF in BIGs

### 8.25.3.29 FF\_2048\_skspow()

```
void FF_2048_skspow (
    BIG_1024_58 * r,
    BIG_1024_58 * x,
    BIG_1024_58 e,
    BIG_1024_58 * m,
    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>m</i>	FF modulus
<i>n</i>	size of FF in BIGs

### 8.25.3.30 FF\_2048\_sqr()

```
void FF_2048_sqr (
    BIG_1024_58 * x,
    BIG_1024_58 * y,
    int n )
```

Uses Karatsuba method internally

**Parameters**

$x$	FF instance, on exit = $y^2$
$y$	FF instance to be squared
$n$	size of FF in BIGs

**8.25.3.31 FF\_2048\_sub()**

```
void FF_2048_sub (
    BIG_1024_58 * x,
    BIG_1024_58 * y,
    BIG_1024_58 * z,
    int n )
```

**Parameters**

$x$	FF instance, on exit = $y-z$
$y$	FF instance
$z$	FF instance
$n$	size of FF in BIGs

**8.25.3.32 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**

$S$	output octet string
$x$	FF instance to be converted to an octet string
$n$	size of FF in BIGs

**8.25.3.33 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

## 8.26 ff\_3072.h File Reference

FF Header File.

```
#include "big_384_56.h"
#include "config_ff_3072.h"
```

## Macros

- `#define HFLEN_3072 (FFLEN_3072/2)`
- `#define P_MBITS_3072 (MODBYTES_384_56*8)`
- `#define P_TBITS_3072 (P_MBITS_3072%BASEBITS_384_56)`
- `#define P_EXCESS_3072(a) (((a[NLEN_384_56-1])>>(P_TBITS_3072))+1)`
- `#define P_FEXCESS_3072 ((chunk)1<<((BASEBITS_384_56*NLEN_384_56-P_MBITS_3072-1)))`

## Functions

- void `FF_3072_copy` (`BIG_384_56 *x`, `BIG_384_56 *y`, int `n`)  
*Copy one FF element of given length to another.*
- void `FF_3072_init` (`BIG_384_56 *x`, `sign32 m`, int `n`)  
*Initialize an FF element of given length from a 32-bit integer m.*
- void `FF_3072_zero` (`BIG_384_56 *x`, int `n`)  
*Set FF element of given size to zero.*
- int `FF_3072_iszilch` (`BIG_384_56 *x`, int `n`)  
*Tests for FF element equal to zero.*
- int `FF_3072_parity` (`BIG_384_56 *x`)  
*return parity of an FF, that is the least significant bit*
- int `FF_3072_lastbits` (`BIG_384_56 *x`, int `m`)  
*return least significant m bits of an FF*
- void `FF_3072_one` (`BIG_384_56 *x`, int `n`)  
*Set FF element of given size to unity.*
- int `FF_3072_comp` (`BIG_384_56 *x`, `BIG_384_56 *y`, int `n`)  
*Compares two FF numbers. Inputs must be normalised externally.*
- void `FF_3072_add` (`BIG_384_56 *x`, `BIG_384_56 *y`, `BIG_384_56 *z`, int `n`)  
*addition of two FFs*
- void `FF_3072_sub` (`BIG_384_56 *x`, `BIG_384_56 *y`, `BIG_384_56 *z`, int `n`)  
*subtraction of two FFs*
- void `FF_3072_inc` (`BIG_384_56 *x`, int `m`, int `n`)  
*increment an FF by an integer, and normalise*
- void `FF_3072_dec` (`BIG_384_56 *x`, int `m`, int `n`)  
*Decrement an FF by an integer, and normalise.*
- void `FF_3072_norm` (`BIG_384_56 *x`, int `n`)

- Normalises the components of an FF.*

  - void `FF_3072_shl` (`BIG_384_56 *x`, int `n`)

*Shift left an FF by 1 bit.*
- void `FF_3072_shr` (`BIG_384_56 *x`, int `n`)

*Shift right an FF by 1 bit.*
- void `FF_3072_output` (`BIG_384_56 *x`, int `n`)

*Formats and outputs an FF to the console.*
- void `FF_3072_rawoutput` (`BIG_384_56 *x`, int `n`)

*Formats and outputs an FF to the console, in raw form.*
- void `FF_3072_toOctet` (octet `*S`, `BIG_384_56 *x`, int `n`)

*Formats and outputs an FF instance to an octet string.*
- void `FF_3072_fromOctet` (`BIG_384_56 *x`, octet `*S`, int `n`)

*Populates an FF instance from an octet string.*
- void `FF_3072_mul` (`BIG_384_56 *x`, `BIG_384_56 *y`, `BIG_384_56 *z`, int `n`)

*Multiplication of two FFs.*
- void `FF_3072_mod` (`BIG_384_56 *x`, `BIG_384_56 *m`, int `n`)

*Reduce FF mod a modulus.*
- void `FF_3072_sqr` (`BIG_384_56 *x`, `BIG_384_56 *y`, int `n`)

*Square an FF.*
- void `FF_3072_dmod` (`BIG_384_56 *x`, `BIG_384_56 *y`, `BIG_384_56 *z`, int `n`)

*Reduces a double-length FF with respect to a given modulus.*
- void `FF_3072_invmodp` (`BIG_384_56 *x`, `BIG_384_56 *y`, `BIG_384_56 *z`, int `n`)

*Invert an FF mod a prime modulus.*
- void `FF_3072_random` (`BIG_384_56 *x`, csprng `*R`, int `n`)

*Create an FF from a random number generator.*
- void `FF_3072_randomnum` (`BIG_384_56 *x`, `BIG_384_56 *y`, csprng `*R`, int `n`)

*Create a random FF less than a given modulus from a random number generator.*
- void `FF_3072_skpow` (`BIG_384_56 *r`, `BIG_384_56 *x`, `BIG_384_56 *e`, `BIG_384_56 *m`, int `n`)

*Calculate  $r=x^e \bmod m$ , side channel resistant.*
- void `FF_3072_skspow` (`BIG_384_56 *r`, `BIG_384_56 *x`, `BIG_384_56 e`, `BIG_384_56 *m`, int `n`)

*Calculate  $r=x^e \bmod m$ , side channel resistant.*
- void `FF_3072_power` (`BIG_384_56 *r`, `BIG_384_56 *x`, int `e`, `BIG_384_56 *m`, int `n`)

*Calculate  $r=x^e \bmod m$ .*
- void `FF_3072_pow` (`BIG_384_56 *r`, `BIG_384_56 *x`, `BIG_384_56 *e`, `BIG_384_56 *m`, int `n`)

*Calculate  $r=x^e \bmod m$ .*
- int `FF_3072_cfactor` (`BIG_384_56 *x`, sign32 `s`, int `n`)

*Test if an FF has factor in common with integer `s`.*
- int `FF_3072_prime` (`BIG_384_56 *x`, csprng `*R`, int `n`)

*Test if an FF is prime.*
- void `FF_3072_pow2` (`BIG_384_56 *r`, `BIG_384_56 *x`, `BIG_384_56 e`, `BIG_384_56 *y`, `BIG_384_56 f`, `BIG_384_56 *m`, int `n`)

*Calculate  $r=x^e.y^f \bmod m$ .*

### 8.26.1 Detailed Description

Author

Mike Scott



## 8.26.2 Macro Definition Documentation

### 8.26.2.1 HFLEN\_3072

```
#define HFLEN_3072 (FFLEN_3072/2)
```

Useful for half-size RSA private key operations

### 8.26.2.2 P\_EXCESS\_3072

```
#define P_EXCESS_3072(  
    a ) (((a[NLEN_384_56-1])>>(P_TBITS_3072))+1)
```

TODO

### 8.26.2.3 P\_FEXCESS\_3072

```
#define P_FEXCESS_3072 ((chunk)1<<(BASEBITS_384_56*NLEN_384_56-P_MBITS_3072-1))
```

TODO

### 8.26.2.4 P\_MBITS\_3072

```
#define P_MBITS_3072 (MODBYTES_384_56*8)
```

Number of bits in modulus

### 8.26.2.5 P\_TBITS\_3072

```
#define P_TBITS_3072 (P_MBITS_3072%BASEBITS_384_56)
```

TODO

## 8.26.3 Function Documentation

### 8.26.3.1 FF\_3072\_add()

```
void FF_3072_add (  
    BIG_384_56 * x,  
    BIG_384_56 * y,  
    BIG_384_56 * z,  
    int n )
```

**Parameters**

$x$	FF instance, on exit = $y+z$
$y$	FF instance
$z$	FF instance
$n$	size of FF in BIGs

**8.26.3.2 FF\_3072\_cfactor()**

```
int FF_3072_cfactor (
    BIG_384_56 * x,
    sign32 s,
    int n )
```

**Parameters**

$x$	FF instance to be tested
$s$	the supplied integer
$n$	size of FF in BIGs

**Returns**

1 if  $\gcd(x,s) \neq 1$ , else return 0

**8.26.3.3 FF\_3072\_comp()**

```
int FF_3072_comp (
    BIG_384_56 * x,
    BIG_384_56 * y,
    int n )
```

**Parameters**

$x$	first FF number to be compared
$y$	second FF number to be compared
$n$	size of FF in BIGs

**Returns**

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

## 8.26.3.4 FF\_3072\_copy()

```
void FF_3072_copy (
    BIG_384_56 * x,
    BIG_384_56 * 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

## 8.26.3.5 FF\_3072\_dec()

```
void FF_3072_dec (
    BIG_384_56 * 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

## 8.26.3.6 FF\_3072\_dmod()

```
void FF_3072_dmod (
    BIG_384_56 * x,
    BIG_384_56 * y,
    BIG_384_56 * 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

### 8.26.3.7 FF\_3072\_fromOctet()

```
void FF_3072_fromOctet (
    BIG_384_56 * 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

### 8.26.3.8 FF\_3072\_inc()

```
void FF_3072_inc (
    BIG_384_56 * 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

### 8.26.3.9 FF\_3072\_init()

```
void FF_3072_init (
    BIG_384_56 * 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

### 8.26.3.10 FF\_3072\_invmodp()

```
void FF_3072_invmodp (
    BIG_384_56 * x,
    BIG_384_56 * y,
    BIG_384_56 * 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

### 8.26.3.11 FF\_3072\_iszilch()

```
int FF_3072_iszilch (
    BIG_384_56 * 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

### 8.26.3.12 FF\_3072\_lastbits()

```
int FF_3072_lastbits (
    BIG_384_56 * 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

### 8.26.3.13 FF\_3072\_mod()

```
void FF_3072_mod (
    BIG_384_56 * x,
    BIG_384_56 * m,
    int n )
```

This is slow

#### Parameters

$x$	FF instance to be reduced mod $m$ - on exit = $x \bmod m$
$m$	FF modulus
$n$	size of FF in BIGs

### 8.26.3.14 FF\_3072\_mul()

```
void FF_3072_mul (
    BIG_384_56 * x,
    BIG_384_56 * y,
    BIG_384_56 * 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

### 8.26.3.15 FF\_3072\_norm()

```
void FF_3072_norm (
    BIG_384_56 * x,
    int n )
```

#### Parameters

$x$	FF instance to be normalised
$n$	size of FF in BIGs

### 8.26.3.16 FF\_3072\_one()

```
void FF_3072_one (
    BIG_384_56 * x,
    int n )
```

#### Parameters

<i>x</i>	FF instance to be set to unity
<i>n</i>	size of FF in BIGs

### 8.26.3.17 FF\_3072\_output()

```
void FF_3072_output (
    BIG_384_56 * x,
    int n )
```

#### Parameters

<i>x</i>	FF instance to be printed
<i>n</i>	size of FF in BIGs

### 8.26.3.18 FF\_3072\_parity()

```
int FF_3072_parity (
    BIG_384_56 * x )
```

#### Parameters

<i>x</i>	FF number
----------	-----------

#### Returns

0 or 1

### 8.26.3.19 FF\_3072\_pow()

```
void FF_3072_pow (
    BIG_384_56 * r,
    BIG_384_56 * x,
    BIG_384_56 * e,
    BIG_384_56 * m,
    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>m</i>	FF modulus
<i>n</i>	size of FF in BIGs

## 8.26.3.20 FF\_3072\_pow2()

```
void FF_3072_pow2 (
    BIG_384_56 * r,
    BIG_384_56 * x,
    BIG_384_56 e,
    BIG_384_56 * y,
    BIG_384_56 f,
    BIG_384_56 * 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

## 8.26.3.21 FF\_3072\_power()

```
void FF_3072_power (
    BIG_384_56 * r,
    BIG_384_56 * x,
    int e,
    BIG_384_56 * m,
    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>m</i>	FF modulus
<i>n</i>	size of FF in BIGs



### 8.26.3.22 FF\_3072\_prime()

```
int FF_3072_prime (
    BIG_384_56 * 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

### 8.26.3.23 FF\_3072\_random()

```
void FF_3072_random (
    BIG_384_56 * 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

### 8.26.3.24 FF\_3072\_randomnum()

```
void FF_3072_randomnum (
    BIG_384_56 * x,
    BIG_384_56 * y,
    csprng * R,
    int n )
```

#### Parameters

<i>x</i>	FF instance, on exit <i>x</i> is a random number < <i>y</i>
----------	---

## Parameters

$y$	FF instance, the modulus
$R$	an instance of a Cryptographically Secure Random Number Generator
$n$	size of FF in BIGs

## 8.26.3.25 FF\_3072\_rawoutput()

```
void FF_3072_rawoutput (
    BIG_384_56 * x,
    int n )
```

## Parameters

$x$	FF instance to be printed
$n$	size of FF in BIGs

## 8.26.3.26 FF\_3072\_shl()

```
void FF_3072_shl (
    BIG_384_56 * x,
    int n )
```

## Parameters

$x$	FF instance to be shifted left
$n$	size of FF in BIGs

## 8.26.3.27 FF\_3072\_shr()

```
void FF_3072_shr (
    BIG_384_56 * x,
    int n )
```

## Parameters

$x$	FF instance to be shifted right
$n$	size of FF in BIGs

## 8.26.3.28 FF\_3072\_skpow()

```
void FF_3072_skpow (
    BIG_384_56 * r,
    BIG_384_56 * x,
    BIG_384_56 * e,
    BIG_384_56 * m,
    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>m</i>	FF modulus
<i>n</i>	size of FF in BIGs

## 8.26.3.29 FF\_3072\_skspow()

```
void FF_3072_skspow (
    BIG_384_56 * r,
    BIG_384_56 * x,
    BIG_384_56 e,
    BIG_384_56 * m,
    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>m</i>	FF modulus
<i>n</i>	size of FF in BIGs

## 8.26.3.30 FF\_3072\_sqr()

```
void FF_3072_sqr (
    BIG_384_56 * x,
    BIG_384_56 * 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

**8.26.3.31 FF\_3072\_sub()**

```
void FF_3072_sub (
    BIG_384_56 * x,
    BIG_384_56 * y,
    BIG_384_56 * 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

**8.26.3.32 FF\_3072\_toOctet()**

```
void FF_3072_toOctet (
    octet * S,
    BIG_384_56 * 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

**8.26.3.33 FF\_3072\_zero()**

```
void FF_3072_zero (
    BIG_384_56 * x,
    int n )
```

## Parameters

$x$	FF instance to be set to zero
$n$	size of FF in BIGs

## 8.27 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`

## 8.27.1 Detailed Description

### Author

Mike Scott

## 8.27.2 Function Documentation

### 8.27.2.1 `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

<i>x</i>	FP12 instance, set to y if s!=0
<i>y</i>	another FP12 instance
<i>s</i>	copy only takes place if not equal to 0

## 8.27.2.2 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 = $x^e \pmod r$ as FP4
<i>x</i>	FP12 input
<i>e</i>	BIG exponent
<i>r</i>	BIG group order

## 8.27.2.3 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 = conj(y)
<i>y</i>	FP12 instance

## 8.27.2.4 FP12\_BLS381\_copy()

```
void FP12_BLS381_copy (
    FP12_BLS381 * x,
    FP12_BLS381 * y )
```

**Parameters**

$x$	FP12 instance, on exit = $y$
$y$	FP12 instance to be copied

**8.27.2.5 FP12\_BLS381\_equals()**

```
int FP12_BLS381_equals (
    FP12_BLS381 *  $x$ ,
    FP12_BLS381 *  $y$  )
```

**Parameters**

$x$	FP12 instance to be compared
$y$	FP12 instance to be compared

**Returns**

1 if  $x=y$ , else returns 0

**8.27.2.6 FP12\_BLS381\_frob()**

```
void FP12_BLS381_frob (
    FP12_BLS381 *  $x$ ,
    FP2_BLS381 *  $f$  )
```

**Parameters**

$x$	FP12 instance, on exit = $x^p$
$f$	FP2 precalculated Frobenius constant

**8.27.2.7 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

## 8.27.2.8 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

## 8.27.2.9 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

<i>x</i>	FP12 instance to be created from an octet string
<i>S</i>	input octet string

## 8.27.2.10 FP12\_BLS381\_inv()

```
void FP12_BLS381_inv (
    FP12_BLS381 * x,
    FP12_BLS381 * y )
```

## Parameters

<i>x</i>	FP12 instance, on exit = 1/y
<i>y</i>	FP12 instance

### 8.27.2.11 FP12\_BLS381\_isunity()

```
int FP12_BLS381_isunity (
    FP12_BLS381 * x )
```

#### Parameters

<i>x</i>	FP12 number to be tested
----------	--------------------------

#### Returns

1 if unity, else returns 0

### 8.27.2.12 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

### 8.27.2.13 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

## 8.27.2.14 FP12\_BLS381\_norm()

```
void FP12_BLS381_norm (
    FP12_BLS381 * x )
```

## Parameters

<i>x</i>	FP12 instance to be normalised
----------	--------------------------------

## 8.27.2.15 FP12\_BLS381\_one()

```
void FP12_BLS381_one (
    FP12_BLS381 * x )
```

## Parameters

<i>x</i>	FP12 instance to be set to one
----------	--------------------------------

## 8.27.2.16 FP12\_BLS381\_output()

```
void FP12_BLS381_output (
    FP12_BLS381 * x )
```

## Parameters

<i>x</i>	FP12 instance to be printed
----------	-----------------------------

## 8.27.2.17 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

## 8.27.2.18 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

## 8.27.2.19 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

## 8.27.2.20 FP12\_BLS381\_reduce()

```
void FP12_BLS381_reduce (
    FP12_BLS381 * x )
```

## Parameters

<i>x</i>	FP12 instance, on exit reduced mod Modulus
----------	--

## 8.27.2.21 FP12\_BLS381\_smul()

```
void FP12_BLS381_smul (
    FP12_BLS381 * x,
    FP12_BLS381 * y )
```

## Parameters

$x$	FP12 instance, on exit = $x*y$
$y$	FP12 instance, of special form

## 8.27.2.22 FP12\_BLS381\_sqr()

```
void FP12_BLS381_sqr (
    FP12_BLS381 * x,
    FP12_BLS381 * y )
```

## Parameters

$x$	FP12 instance, on exit = $y^2$
$y$	FP12 instance

## 8.27.2.23 FP12\_BLS381\_ssmul()

```
void FP12_BLS381_ssmul (
    FP12_BLS381 * x,
    FP12_BLS381 * y )
```

## Parameters

$x$	FP12 instance, on exit = $x*y$
$y$	FP12 instance, of special form

## 8.27.2.24 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

$S$	output octet string
$x$	FP12 instance to be converted to an octet string

#### 8.27.2.25 FP12\_BLS381\_trace()

```
void FP12_BLS381_trace (
    FP4_BLS381 * t,
    FP12_BLS381 * x )
```

##### Parameters

$t$	FP4 trace of $x$ , on exit = $\text{tr}(x)$
$x$	FP12 instance

#### 8.27.2.26 FP12\_BLS381\_usqr()

```
void FP12_BLS381_usqr (
    FP12_BLS381 * x,
    FP12_BLS381 * y )
```

##### Parameters

$x$	FP12 instance, on exit = $y^2$
$y$	FP4 instance, must be unitary

#### 8.27.2.27 FP12\_BLS381\_zero()

```
void FP12_BLS381_zero (
    FP12_BLS381 * x )
```

##### Parameters

$x$	FP12 instance to be set to zero
-----	---------------------------------

### 8.27.3 Variable Documentation

#### 8.27.3.1 Fra\_BLS381

```
const BIG_384_58 Fra_BLS381
```

real part of BN curve Frobenius Constant

## 8.27.3.2 Frb\_BLS381

```
const BIG_384_58 Frb_BLS381
```

imaginary part of BN curve Frobenius Constant

## 8.28 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  $\sqrt{-1}$*

## 8.28.1 Detailed Description

Author

Mike Scott

## 8.28.2 Function Documentation

### 8.28.2.1 `FP2_BLS381_add()`

```
void FP2_BLS381_add (
    FP2_BLS381 * x,
    FP2_BLS381 * y,
    FP2_BLS381 * z )
```



## Parameters

<i>x</i>	FP2 instance, on exit = $y+z$
<i>y</i>	FP2 instance
<i>z</i>	FP2 instance

## 8.28.2.2 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 $s \neq 0$
<i>y</i>	another FP2 instance
<i>s</i>	copy only takes place if not equal to 0

## 8.28.2.3 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

## 8.28.2.4 FP2\_BLS381\_copy()

```
void FP2_BLS381_copy (
    FP2_BLS381 * x,
    FP2_BLS381 * y )
```

## Parameters

$x$	FP2 instance, on exit = $y$
$y$	FP2 instance to be copied

## 8.28.2.5 FP2\_BLS381\_div2()

```
void FP2_BLS381_div2 (
    FP2_BLS381 * x,
    FP2_BLS381 * y )
```

## Parameters

$x$	FP2 instance, on exit = $y/2$
$y$	FP2 instance

## 8.28.2.6 FP2\_BLS381\_div\_ip()

```
void FP2_BLS381_div_ip (
    FP2_BLS381 * x )
```

Note that  $(1+\sqrt{-1})$  is irreducible for FP4

## Parameters

$x$	FP2 instance, on exit = $x/(1+\sqrt{-1})$
-----	---

## 8.28.2.7 FP2\_BLS381\_div\_ip2()

```
void FP2_BLS381_div_ip2 (
    FP2_BLS381 * x )
```

Note that  $(1+\sqrt{-1})$  is irreducible for FP4

## Parameters

$x$	FP2 instance, on exit = $2x/(1+\sqrt{-1})$
-----	--

### 8.28.2.8 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

### 8.28.2.9 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

### 8.28.2.10 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

**8.28.2.11 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

**8.28.2.12 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

**8.28.2.13 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

**8.28.2.14 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

**8.28.2.15 FP2\_BLS381\_issunity()**

```
int FP2_BLS381_issunity (  
    FP2_BLS381 * x )
```

**Parameters**

<i>x</i>	FP2 instance to be tested
----------	---------------------------

**Returns**

1 if x=1, else returns 0

**8.28.2.16 FP2\_BLS381\_iszilch()**

```
int FP2_BLS381_iszilch (  
    FP2_BLS381 * x )
```

**Parameters**

<i>x</i>	FP2 number to be tested
----------	-------------------------

**Returns**

1 if zero, else returns 0

**8.28.2.17 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

**8.28.2.18 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 = $x*(1+\sqrt{-1})$
-----	---

**8.28.2.19 FP2\_BLS381\_neg()**

```
void FP2_BLS381_neg (
    FP2_BLS381 * x,
    FP2_BLS381 * y )
```

**Parameters**

$x$	FP2 instance, on exit = $-y$
$y$	FP2 instance

**8.28.2.20 FP2\_BLS381\_norm()**

```
void FP2_BLS381_norm (
    FP2_BLS381 * x )
```

**Parameters**

$x$	FP2 instance to be normalised
-----	-------------------------------

#### 8.28.2.21 FP2\_BLS381\_one()

```
void FP2_BLS381_one (
    FP2_BLS381 * x )
```

##### Parameters

<i>x</i>	FP2 instance to be set to one
----------	-------------------------------

#### 8.28.2.22 FP2\_BLS381\_output()

```
void FP2_BLS381_output (
    FP2_BLS381 * x )
```

##### Parameters

<i>x</i>	FP2 instance
----------	--------------

#### 8.28.2.23 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

#### 8.28.2.24 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

#### 8.28.2.25 FP2\_BLS381\_rawoutput()

```
void FP2_BLS381_rawoutput (
    FP2_BLS381 * x )
```

##### Parameters

$x$	FP2 instance
-----	--------------

#### 8.28.2.26 FP2\_BLS381\_reduce()

```
void FP2_BLS381_reduce (
    FP2_BLS381 * x )
```

##### Parameters

$x$	FP2 instance, on exit reduced mod Modulus
-----	---

#### 8.28.2.27 FP2\_BLS381\_sqr()

```
void FP2_BLS381_sqr (
    FP2_BLS381 * x,
    FP2_BLS381 * y )
```

##### Parameters

$x$	FP2 instance, on exit = $y^2$
$y$	FP2 instance

#### 8.28.2.28 FP2\_BLS381\_sqrt()

```
int FP2_BLS381_sqrt (
    FP2_BLS381 * x,
    FP2_BLS381 * y )
```

##### Parameters

$x$	FP2 instance, on exit = sqrt( $y$ )
$y$	FP2 instance



**8.28.2.29 FP2\_BLS381\_sub()**

```
void FP2_BLS381_sub (
    FP2_BLS381 * x,
    FP2_BLS381 * y,
    FP2_BLS381 * z )
```

**Parameters**

<i>x</i>	FP2 instance, on exit = y-z
<i>y</i>	FP2 instance
<i>z</i>	FP2 instance

**8.28.2.30 FP2\_BLS381\_times\_i()**

```
void FP2_BLS381_times_i (
    FP2_BLS381 * x )
```

Note that -1 is QNR

**Parameters**

<i>x</i>	FP2 instance, on exit = $x \cdot \sqrt{-1}$
----------	---

**8.28.2.31 FP2\_BLS381\_zero()**

```
void FP2_BLS381_zero (
    FP2_BLS381 * x )
```

**Parameters**

<i>x</i>	FP2 instance to be set to zero
----------	--------------------------------

**8.29 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.*

## 8.29.1 Detailed Description

### Author

Mike Scott

## 8.29.2 Function Documentation

### 8.29.2.1 FP4\_BLS381\_add()

```
void FP4_BLS381_add (
    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

**8.29.2.2 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 $s \neq 0$
<i>y</i>	another FP4 instance
<i>s</i>	copy only takes place if not equal to 0

**8.29.2.3 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

**8.29.2.4 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

## 8.29.2.5 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

## 8.29.2.6 FP4\_BLS381\_div\_2i()

```
void FP4_BLS381_div_2i (
    FP4_BLS381 * x )
```

Divide FP4 by the QNR/2

## Parameters

<i>x</i>	FP4 instance
----------	--------------

## 8.29.2.7 FP4\_BLS381\_div\_i()

```
void FP4_BLS381_div_i (
    FP4_BLS381 * x )
```

Divide FP4 by the QNR

## Parameters

<i>x</i>	FP4 instance
----------	--------------

### 8.29.2.8 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

### 8.29.2.9 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

### 8.29.2.10 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

### 8.29.2.11 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

#### 8.29.2.12 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

#### 8.29.2.13 FP4\_BLS381\_imul()

```

void FP4_BLS381_imul (
    FP4_BLS381 * x,
    FP4_BLS381 * y,
    int i )

```

#### Parameters

<i>x</i>	FP4 instance, on exit = $y \cdot i$
<i>y</i>	FP4 instance
<i>i</i>	an integer

#### 8.29.2.14 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

**8.29.2.15 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

**8.29.2.16 FP4\_BLS381\_isunity()**

```
int FP4_BLS381_isunity (
    FP4_BLS381 * x )
```

**Parameters**

<i>x</i>	FP4 number to be tested
----------	-------------------------

**Returns**

1 if unity, else returns 0

**8.29.2.17 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

**8.29.2.18 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

**8.29.2.19 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 = $-\text{conj}(y)$
<i>y</i>	FP4 instance

**8.29.2.20 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

#### 8.29.2.21 FP4\_BLS381\_norm()

```
void FP4_BLS381_norm (
    FP4_BLS381 * x )
```

##### Parameters

<i>x</i>	FP4 instance to be normalised
----------	-------------------------------

#### 8.29.2.22 FP4\_BLS381\_one()

```
void FP4_BLS381_one (
    FP4_BLS381 * x )
```

##### Parameters

<i>x</i>	FP4 instance to be set to one
----------	-------------------------------

#### 8.29.2.23 FP4\_BLS381\_output()

```
void FP4_BLS381_output (
    FP4_BLS381 * x )
```

##### Parameters

<i>x</i>	FP4 instance to be printed
----------	----------------------------

#### 8.29.2.24 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

#### 8.29.2.25 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

#### 8.29.2.26 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

#### 8.29.2.27 FP4\_BLS381\_rawoutput()

```
void FP4_BLS381_rawoutput (
    FP4_BLS381 * x )
```

##### Parameters

<i>x</i>	FP4 instance to be printed
----------	----------------------------

#### 8.29.2.28 FP4\_BLS381\_reduce()

```
void FP4_BLS381_reduce (
    FP4_BLS381 * x )
```

## Parameters

$x$	FP4 instance, on exit reduced mod Modulus
-----	---

## 8.29.2.29 FP4\_BLS381\_sqr()

```
void FP4_BLS381_sqr (
    FP4_BLS381 * x,
    FP4_BLS381 * y )
```

## Parameters

$x$	FP4 instance, on exit = $y^2$
$y$	FP4 instance

## 8.29.2.30 FP4\_BLS381\_sqrt()

```
int FP4_BLS381_sqrt (
    FP4_BLS381 * r,
    FP4_BLS381 * x )
```

Square root

## Parameters

$r$	FP4 instance, on exit = sqrt( $x$ )
$x$	FP4 instance

## Returns

1 if  $x$  is a QR, otherwise 0

## 8.29.2.31 FP4\_BLS381\_sub()

```
void FP4_BLS381_sub (
    FP4_BLS381 * x,
    FP4_BLS381 * y,
    FP4_BLS381 * z )
```

## Parameters

$x$	FP4 instance, on exit = $y-z$
$y$	FP4 instance
$z$	FP4 instance

## 8.29.2.32 FP4\_BLS381\_times\_i()

```
void FP4_BLS381_times_i (
    FP4_BLS381 * x )
```

## Parameters

$x$	FP4 instance, on exit = $\sqrt{1+\sqrt{-1}}*x$
-----	--

## 8.29.2.33 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

$r$	FP4 instance, on exit = $w*x-\text{conj}(x)*y+z$
$w$	FP4 instance
$x$	FP4 instance
$y$	FP4 instance
$z$	FP4 instance

## 8.29.2.34 FP4\_BLS381\_xtr\_D()

```
void FP4_BLS381_xtr_D (
    FP4_BLS381 * r,
    FP4_BLS381 * x )
```

## Parameters

$r$	FP4 instance, on exit = $x^2-2*\text{conj}(x)$
$x$	FP4 instance

## 8.29.2.35 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

#### 8.29.2.36 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

#### 8.29.2.37 FP4\_BLS381\_zero()

```

void FP4_BLS381_zero (
    FP4_BLS381 * x )

```

#### Parameters

<i>x</i>	FP4 instance to be set to zero
----------	--------------------------------

## 8.30 fp\_25519.h File Reference

FP Header File.

```
#include "big_256_56.h"
#include "config_field_25519.h"
```

### Data Structures

- struct [FP\\_25519](#)  
*FP Structure - quadratic extension field.*

### Macros

- #define [MODBITS\\_25519](#) MBITS\_25519
- #define [TBITS\\_25519](#) (MBITS\_25519%BASEBITS\_256\_56)
- #define [TMASK\\_25519](#) ((([chunk](#))1<<TBITS\_25519)-1)
- #define [FEXCESS\\_25519](#) ((([sign32](#))1<<MAXXES\_25519)-1)
- #define [OMASK\\_25519](#) (-(([chunk](#))(1)<<TBITS\_25519))

### Functions

- int [FP\\_25519\\_iszilch](#) ([FP\\_25519](#) \*x)  
*Tests for FP equal to zero mod Modulus.*
- void [FP\\_25519\\_zero](#) ([FP\\_25519](#) \*x)  
*Set FP to zero.*
- void [FP\\_25519\\_copy](#) ([FP\\_25519](#) \*y, [FP\\_25519](#) \*x)  
*Copy an FP.*
- void [FP\\_25519\\_rcopy](#) ([FP\\_25519](#) \*y, const [BIG\\_256\\_56](#) x)  
*Copy from ROM to an FP.*
- int [FP\\_25519\\_equals](#) ([FP\\_25519](#) \*x, [FP\\_25519](#) \*y)  
*Compares two FPs.*
- void [FP\\_25519\\_cswap](#) ([FP\\_25519](#) \*x, [FP\\_25519](#) \*y, int s)  
*Conditional constant time swap of two FP numbers.*
- void [FP\\_25519\\_cmove](#) ([FP\\_25519](#) \*x, [FP\\_25519](#) \*y, int s)  
*Conditional copy of FP number.*
- void [FP\\_25519\\_nres](#) ([FP\\_25519](#) \*y, [BIG\\_256\\_56](#) x)  
*Converts from BIG integer to residue form mod Modulus.*
- void [FP\\_25519\\_redc](#) ([BIG\\_256\\_56](#) x, [FP\\_25519](#) \*y)  
*Converts from residue form back to BIG integer form.*
- void [FP\\_25519\\_one](#) ([FP\\_25519](#) \*x)  
*Sets FP to representation of unity in residue form.*
- void [FP\\_25519\\_mod](#) ([BIG\\_256\\_56](#) r, [DBIG\\_256\\_56](#) d)  
*Reduces DBIG to BIG exploiting special form of the modulus.*
- void [FP\\_25519\\_mul](#) ([FP\\_25519](#) \*x, [FP\\_25519](#) \*y, [FP\\_25519](#) \*z)  
*Fast Modular multiplication of two FPs, mod Modulus.*
- void [FP\\_25519\\_imul](#) ([FP\\_25519](#) \*x, [FP\\_25519](#) \*y, int i)  
*Fast Modular multiplication of an FP, by a small integer, mod Modulus.*

- void `FP_25519_sqr` (`FP_25519 *x`, `FP_25519 *y`)  
*Fast Modular squaring of an FP, mod Modulus.*
- void `FP_25519_add` (`FP_25519 *x`, `FP_25519 *y`, `FP_25519 *z`)  
*Modular addition of two FPs, mod Modulus.*
- void `FP_25519_sub` (`FP_25519 *x`, `FP_25519 *y`, `FP_25519 *z`)  
*Modular subtraction of two FPs, mod Modulus.*
- void `FP_25519_div2` (`FP_25519 *x`, `FP_25519 *y`)  
*Modular division by 2 of an FP, mod Modulus.*
- void `FP_25519_pow` (`FP_25519 *x`, `FP_25519 *y`, `BIG_256_56 z`)  
*Fast Modular exponentiation of an FP, to the power of a BIG, mod Modulus.*
- void `FP_25519_sqrt` (`FP_25519 *x`, `FP_25519 *y`)  
*Fast Modular square root of a an FP, mod Modulus.*
- void `FP_25519_neg` (`FP_25519 *x`, `FP_25519 *y`)  
*Modular negation of a an FP, mod Modulus.*
- void `FP_25519_output` (`FP_25519 *x`)  
*Outputs an FP number to the console.*
- void `FP_25519_rawoutput` (`FP_25519 *x`)  
*Outputs an FP number to the console, in raw form.*
- void `FP_25519_reduce` (`FP_25519 *x`)  
*Reduces possibly unreduced FP mod Modulus.*
- void `FP_25519_norm` (`FP_25519 *x`)  
*normalizes FP*
- int `FP_25519_qr` (`FP_25519 *x`)  
*Tests for FP a quadratic residue mod Modulus.*
- void `FP_25519_inv` (`FP_25519 *x`, `FP_25519 *y`)  
*Modular inverse of a an FP, mod Modulus.*

## Variables

- const `BIG_256_56 Modulus_25519`
- const `BIG_256_56 R2modp_25519`
- const `chunk MConst_25519`

### 8.30.1 Detailed Description

#### Author

Mike Scott

### 8.30.2 Macro Definition Documentation

#### 8.30.2.1 FEXCESS\_25519

```
#define FEXCESS_25519 (((sign32)1<<MAXXES_25519)-1)
```

$2^{(BASEBITS*NLEN-MODBITS)-1}$  - normalised BIG can be multiplied by less than this before reduction



### 8.30.2.2 MODBITS\_25519

```
#define MODBITS_25519 MBITS_25519
```

Number of bits in Modulus for selected curve

### 8.30.2.3 OMASK\_25519

```
#define OMASK_25519 ( - ( (chunk) (1) << TBITS_25519 ) )
```

for masking out overflow bits

### 8.30.2.4 TBITS\_25519

```
#define TBITS_25519 (MBITS_25519%BASEBITS_256_56)
```

Number of active bits in top word

### 8.30.2.5 TMASK\_25519

```
#define TMASK_25519 ( ( (chunk) 1 << TBITS_25519 ) - 1 )
```

Mask for active bits in top word

## 8.30.3 Function Documentation

### 8.30.3.1 FP\_25519\_add()

```
void FP_25519_add (
    FP_25519 * x,
    FP_25519 * y,
    FP_25519 * z )
```

#### Parameters

x	FP number, on exit the modular sum = y+z mod Modulus
y	FP number
z	FP number

### 8.30.3.2 FP\_25519\_cmove()

```
void FP_25519_cmove (
```

```

    FP_25519 * x,
    FP_25519 * 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

#### 8.30.3.3 FP\_25519\_copy()

```

void FP_25519_copy (
    FP_25519 * y,
    FP_25519 * x )

```

#### Parameters

<i>y</i>	FP number to be copied to
<i>x</i>	FP to be copied from

#### 8.30.3.4 FP\_25519\_cswap()

```

void FP_25519_cswap (
    FP_25519 * x,
    FP_25519 * 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

#### 8.30.3.5 FP\_25519\_div2()

```

void FP_25519_div2 (
    FP_25519 * x,
    FP_25519 * y )

```

## Parameters

<i>x</i>	FP number, on exit $=y/2 \bmod \text{Modulus}$
<i>y</i>	FP number

## 8.30.3.6 FP\_25519\_equals()

```
int FP_25519_equals (
    FP_25519 * x,
    FP_25519 * y )
```

## Parameters

<i>x</i>	FP number
<i>y</i>	FP number

## Returns

1 if equal, else returns 0

## 8.30.3.7 FP\_25519\_imul()

```
void FP_25519_imul (
    FP_25519 * x,
    FP_25519 * y,
    int i )
```

## Parameters

<i>x</i>	FP number, on exit the modular product $= y*i \bmod \text{Modulus}$
<i>y</i>	FP number, the multiplicand
<i>i</i>	a small number, the multiplier

## 8.30.3.8 FP\_25519\_inv()

```
void FP_25519_inv (
    FP_25519 * x,
    FP_25519 * y )
```

**Parameters**

<i>x</i>	FP number, on exit = 1/y mod Modulus
<i>y</i>	FP number

**8.30.3.9 FP\_25519\_iszilch()**

```
int FP_25519_iszilch (
    FP_25519 * x )
```

**Parameters**

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

**Returns**

1 if zero, else returns 0

**8.30.3.10 FP\_25519\_mod()**

```
void FP_25519_mod (
    BIG_256_56 r,
    DBIG_256_56 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 mod Modulus
<i>d</i>	DBIG number to be reduced

**8.30.3.11 FP\_25519\_mul()**

```
void FP_25519_mul (
    FP_25519 * x,
    FP_25519 * y,
    FP_25519 * 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

## 8.30.3.12 FP\_25519\_neg()

```
void FP_25519_neg (
    FP_25519 * x,
    FP_25519 * y )
```

## Parameters

<i>x</i>	FP number, on exit = $-y \bmod \text{Modulus}$
<i>y</i>	FP number

## 8.30.3.13 FP\_25519\_norm()

```
void FP_25519_norm (
    FP_25519 * x )
```

## Parameters

<i>x</i>	FP number, on exit normalized
----------	-------------------------------

## 8.30.3.14 FP\_25519\_nres()

```
void FP_25519_nres (
    FP_25519 * y,
    BIG_256_56 x )
```

## Parameters

<i>x</i>	BIG number to be converted
<i>y</i>	FP result

**8.30.3.15 FP\_25519\_one()**

```
void FP_25519_one (
    FP_25519 * x )
```

**Parameters**

<i>x</i>	FP number to be set equal to unity.
----------	-------------------------------------

**8.30.3.16 FP\_25519\_output()**

```
void FP_25519_output (
    FP_25519 * x )
```

Converts from residue form before output

**Parameters**

<i>x</i>	an FP number
----------	--------------

**8.30.3.17 FP\_25519\_pow()**

```
void FP_25519_pow (
    FP_25519 * x,
    FP_25519 * y,
    BIG_256_56 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

**8.30.3.18 FP\_25519\_qr()**

```
int FP_25519_qr (
    FP_25519 * x )
```

**Parameters**

<i>x</i>	FP number to be tested
----------	------------------------

**Returns**

1 if quadratic residue, else returns 0 if quadratic non-residue

**8.30.3.19 FP\_25519\_rawoutput()**

```
void FP_25519_rawoutput (
    FP_25519 * x )
```

**Parameters**

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

**8.30.3.20 FP\_25519\_rcopy()**

```
void FP_25519_rcopy (
    FP_25519 * y,
    const BIG_256_56 x )
```

**Parameters**

<i>y</i>	FP number to be copied to
<i>x</i>	BIG to be copied from ROM

**8.30.3.21 FP\_25519\_redc()**

```
void FP_25519_redc (
    BIG_256_56 x,
    FP_25519 * y )
```

**Parameters**

<i>y</i>	FP number to be converted to BIG
<i>x</i>	BIG result

**8.30.3.22 FP\_25519\_reduce()**

```
void FP_25519_reduce (
    FP_25519 * x )
```

**Parameters**

$x$	FP number, on exit reduced mod Modulus
-----	--

**8.30.3.23 FP\_25519\_sqr()**

```
void FP_25519_sqr (
    FP_25519 * x,
    FP_25519 * y )
```

Uses appropriate fast modular reduction method

**Parameters**

$x$	FP number, on exit the modular product = $y^2 \bmod \text{Modulus}$
$y$	FP number, the number to be squared

**8.30.3.24 FP\_25519\_sqrt()**

```
void FP_25519_sqrt (
    FP_25519 * x,
    FP_25519 * y )
```

**Parameters**

$x$	FP number, on exit = $\text{sqrt}(y) \bmod \text{Modulus}$
$y$	FP number, the number whose square root is calculated

**8.30.3.25 FP\_25519\_sub()**

```
void FP_25519_sub (
    FP_25519 * x,
    FP_25519 * y,
    FP_25519 * z )
```

**Parameters**

$x$	FP number, on exit the modular difference = $y - z \bmod \text{Modulus}$
$y$	FP number
$z$	FP number



## 8.30.3.26 FP\_25519\_zero()

```
void FP_25519_zero (
    FP_25519 * x )
```

## Parameters

<i>x</i>	FP number to be set to 0
----------	--------------------------

## 8.30.4 Variable Documentation

## 8.30.4.1 MConst\_25519

```
const chunk MConst_25519
```

Constant associated with Modulus - for Montgomery =  $1/p \bmod 2^{\text{BASEBITS}}$

## 8.30.4.2 Modulus\_25519

```
const BIG_256_56 Modulus_25519
```

Actual Modulus set in romf\_yyy.c

## 8.30.4.3 R2modp\_25519

```
const BIG_256_56 R2modp_25519
```

Montgomery constant

## 8.31 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`

## 8.31.1 Detailed Description

### Author

Mike Scott

## 8.31.2 Macro Definition Documentation

### 8.31.2.1 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

### 8.31.2.2 MODBITS\_BLS381

```
#define MODBITS_BLS381 MBITS_BLS381
```

Number of bits in Modulus for selected curve

### 8.31.2.3 OMASK\_BLS381

```
#define OMASK_BLS381 (~((chunk)(1)<<TBITS_BLS381))
```

for masking out overflow bits

#### 8.31.2.4 TBITS\_BLS381

```
#define TBITS_BLS381 (MBITS_BLS381%BASEBITS_384_58)
```

Number of active bits in top word

#### 8.31.2.5 TMASK\_BLS381

```
#define TMASK_BLS381 (((chunk)1<<TBITS_BLS381)-1)
```

Mask for active bits in top word

### 8.31.3 Function Documentation

#### 8.31.3.1 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 mod Modulus
<i>y</i>	FP number
<i>z</i>	FP number

#### 8.31.3.2 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

### 8.31.3.3 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

### 8.31.3.4 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

### 8.31.3.5 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

### 8.31.3.6 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

**8.31.3.7 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

**8.31.3.8 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

**8.31.3.9 FP\_BLS381\_iszilch()**

```
int FP_BLS381_iszilch (
    FP_BLS381 *  $x$  )
```

**Parameters**

$x$	BIG number to be tested
-----	-------------------------

**Returns**

1 if zero, else returns 0

**8.31.3.10 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

**8.31.3.11 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

**8.31.3.12 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

#### 8.31.3.13 FP\_BLS381\_norm()

```
void FP_BLS381_norm (
    FP_BLS381 * x )
```

##### Parameters

<i>x</i>	FP number, on exit normalized
----------	-------------------------------

#### 8.31.3.14 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

#### 8.31.3.15 FP\_BLS381\_one()

```
void FP_BLS381_one (
    FP_BLS381 * x )
```

##### Parameters

<i>x</i>	FP number to be set equal to unity.
----------	-------------------------------------

#### 8.31.3.16 FP\_BLS381\_output()

```
void FP_BLS381_output (
    FP_BLS381 * x )
```

Converts from residue form before output

##### Parameters

<i>x</i>	an FP number
----------	--------------



#### 8.31.3.17 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

#### 8.31.3.18 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

#### 8.31.3.19 FP\_BLS381\_rawoutput()

```
void FP_BLS381_rawoutput (
    FP_BLS381 * x )
```

##### Parameters

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

#### 8.31.3.20 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

**8.31.3.21 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

**8.31.3.22 FP\_BLS381\_reduce()**

```

void FP_BLS381_reduce (
    FP_BLS381 * x )

```

**Parameters**

<i>x</i>	FP number, on exit reduced mod Modulus
----------	--

**8.31.3.23 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

#### 8.31.3.24 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

#### 8.31.3.25 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

#### 8.31.3.26 FP\_BLS381\_zero()

```
void FP_BLS381_zero (
    FP_BLS381 * x )
```

##### Parameters

<i>x</i>	FP number to be set to 0
----------	--------------------------

### 8.31.4 Variable Documentation

#### 8.31.4.1 MConst\_BLS381

```
const chunk MConst_BLS381
```

Constant associated with Modulus - for Montgomery =  $1/p \bmod 2^{\text{BASEBITS}}$

#### 8.31.4.2 Modulus\_BLS381

```
const BIG_384_58 Modulus_BLS381
```

Actual Modulus set in romf\_yyy.c

#### 8.31.4.3 R2modp\_BLS381

```
const BIG_384_58 R2modp_BLS381
```

Montgomery constant

## 8.32 fp\_GOLDILOCKS.h File Reference

FP Header File.

```
#include "big_448_58.h"
#include "config_field_GOLDILOCKS.h"
```

### Data Structures

- struct [FP\\_GOLDILOCKS](#)  
*FP Structure - quadratic extension field.*

### Macros

- #define [MODBITS\\_GOLDILOCKS](#) MBITS\_GOLDILOCKS
- #define [TBITS\\_GOLDILOCKS](#) (MBITS\_GOLDILOCKS%BASEBITS\_448\_58)
- #define [TMASK\\_GOLDILOCKS](#) (((chunk)1<<TBITS\_GOLDILOCKS)-1)
- #define [FEXCESS\\_GOLDILOCKS](#) (((sign32)1<<MAXXES\_GOLDILOCKS)-1)
- #define [OMASK\\_GOLDILOCKS](#) (-((chunk)1)<<TBITS\_GOLDILOCKS))

### Functions

- int [FP\\_GOLDILOCKS\\_iszilch](#) (FP\_GOLDILOCKS \*x)  
*Tests for FP equal to zero mod Modulus.*
- void [FP\\_GOLDILOCKS\\_zero](#) (FP\_GOLDILOCKS \*x)  
*Set FP to zero.*
- void [FP\\_GOLDILOCKS\\_copy](#) (FP\_GOLDILOCKS \*y, FP\_GOLDILOCKS \*x)  
*Copy an FP.*
- void [FP\\_GOLDILOCKS\\_rcopy](#) (FP\_GOLDILOCKS \*y, const [BIG\\_448\\_58](#) x)  
*Copy from ROM to an FP.*
- int [FP\\_GOLDILOCKS\\_equals](#) (FP\_GOLDILOCKS \*x, FP\_GOLDILOCKS \*y)  
*Compares two FPs.*
- void [FP\\_GOLDILOCKS\\_cswap](#) (FP\_GOLDILOCKS \*x, FP\_GOLDILOCKS \*y, int s)  
*Conditional constant time swap of two FP numbers.*

- void `FP_GOLDILOCKS_cmove` (`FP_GOLDILOCKS *x`, `FP_GOLDILOCKS *y`, int `s`)  
*Conditional copy of FP number.*
- void `FP_GOLDILOCKS_nres` (`FP_GOLDILOCKS *y`, `BIG_448_58 x`)  
*Converts from BIG integer to residue form mod Modulus.*
- void `FP_GOLDILOCKS_redc` (`BIG_448_58 x`, `FP_GOLDILOCKS *y`)  
*Converts from residue form back to BIG integer form.*
- void `FP_GOLDILOCKS_one` (`FP_GOLDILOCKS *x`)  
*Sets FP to representation of unity in residue form.*
- void `FP_GOLDILOCKS_mod` (`BIG_448_58 r`, `DBIG_448_58 d`)  
*Reduces DBIG to BIG exploiting special form of the modulus.*
- void `FP_GOLDILOCKS_mul` (`FP_GOLDILOCKS *x`, `FP_GOLDILOCKS *y`, `FP_GOLDILOCKS *z`)  
*Fast Modular multiplication of two FPs, mod Modulus.*
- void `FP_GOLDILOCKS_imul` (`FP_GOLDILOCKS *x`, `FP_GOLDILOCKS *y`, int `i`)  
*Fast Modular multiplication of an FP, by a small integer, mod Modulus.*
- void `FP_GOLDILOCKS_sqr` (`FP_GOLDILOCKS *x`, `FP_GOLDILOCKS *y`)  
*Fast Modular squaring of an FP, mod Modulus.*
- void `FP_GOLDILOCKS_add` (`FP_GOLDILOCKS *x`, `FP_GOLDILOCKS *y`, `FP_GOLDILOCKS *z`)  
*Modular addition of two FPs, mod Modulus.*
- void `FP_GOLDILOCKS_sub` (`FP_GOLDILOCKS *x`, `FP_GOLDILOCKS *y`, `FP_GOLDILOCKS *z`)  
*Modular subtraction of two FPs, mod Modulus.*
- void `FP_GOLDILOCKS_div2` (`FP_GOLDILOCKS *x`, `FP_GOLDILOCKS *y`)  
*Modular division by 2 of an FP, mod Modulus.*
- void `FP_GOLDILOCKS_pow` (`FP_GOLDILOCKS *x`, `FP_GOLDILOCKS *y`, `BIG_448_58 z`)  
*Fast Modular exponentiation of an FP, to the power of a BIG, mod Modulus.*
- void `FP_GOLDILOCKS_sqrt` (`FP_GOLDILOCKS *x`, `FP_GOLDILOCKS *y`)  
*Fast Modular square root of a an FP, mod Modulus.*
- void `FP_GOLDILOCKS_neg` (`FP_GOLDILOCKS *x`, `FP_GOLDILOCKS *y`)  
*Modular negation of a an FP, mod Modulus.*
- void `FP_GOLDILOCKS_output` (`FP_GOLDILOCKS *x`)  
*Outputs an FP number to the console.*
- void `FP_GOLDILOCKS_rawoutput` (`FP_GOLDILOCKS *x`)  
*Outputs an FP number to the console, in raw form.*
- void `FP_GOLDILOCKS_reduce` (`FP_GOLDILOCKS *x`)  
*Reduces possibly unreduced FP mod Modulus.*
- void `FP_GOLDILOCKS_norm` (`FP_GOLDILOCKS *x`)  
*normalizes FP*
- int `FP_GOLDILOCKS_qr` (`FP_GOLDILOCKS *x`)  
*Tests for FP a quadratic residue mod Modulus.*
- void `FP_GOLDILOCKS_inv` (`FP_GOLDILOCKS *x`, `FP_GOLDILOCKS *y`)  
*Modular inverse of a an FP, mod Modulus.*

## Variables

- const `BIG_448_58 Modulus_GOLDILOCKS`
- const `BIG_448_58 R2modp_GOLDILOCKS`
- const `chunk MConst_GOLDILOCKS`

## 8.32.1 Detailed Description

### Author

Mike Scott

## 8.32.2 Macro Definition Documentation

### 8.32.2.1 FEXCESS\_GOLDILOCKS

```
#define FEXCESS_GOLDILOCKS (((sign32)1)<<MAXXES_GOLDILOCKS)-1)
```

$2^{(BASEBITS*NLEN-MODBITS)}-1$  - normalised BIG can be multiplied by less than this before reduction

### 8.32.2.2 MODBITS\_GOLDILOCKS

```
#define MODBITS_GOLDILOCKS MBITS_GOLDILOCKS
```

Number of bits in Modulus for selected curve

### 8.32.2.3 OMASK\_GOLDILOCKS

```
#define OMASK_GOLDILOCKS (-((chunk)1)<<TBITS_GOLDILOCKS))
```

for masking out overflow bits

### 8.32.2.4 TBITS\_GOLDILOCKS

```
#define TBITS_GOLDILOCKS (MBITS_GOLDILOCKS%BASEBITS\_448\_58)
```

Number of active bits in top word

### 8.32.2.5 TMASK\_GOLDILOCKS

```
#define TMASK_GOLDILOCKS (((chunk)1)<<TBITS_GOLDILOCKS)-1)
```

Mask for active bits in top word

## 8.32.3 Function Documentation

### 8.32.3.1 FP\_GOLDILOCKS\_add()

```
void FP_GOLDILOCKS_add (
    FP\_GOLDILOCKS * x,
    FP\_GOLDILOCKS * y,
    FP\_GOLDILOCKS * 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

## 8.32.3.2 FP\_GOLDILOCKS\_cmove()

```
void FP_GOLDILOCKS_cmove (
    FP_GOLDILOCKS * x,
    FP_GOLDILOCKS * 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

## 8.32.3.3 FP\_GOLDILOCKS\_copy()

```
void FP_GOLDILOCKS_copy (
    FP_GOLDILOCKS * y,
    FP_GOLDILOCKS * x )
```

## Parameters

<i>y</i>	FP number to be copied to
<i>x</i>	FP to be copied from

## 8.32.3.4 FP\_GOLDILOCKS\_cswap()

```
void FP_GOLDILOCKS_cswap (
    FP_GOLDILOCKS * x,
    FP_GOLDILOCKS * 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

## 8.32.3.5 FP\_GOLDILOCKS\_div2()

```
void FP_GOLDILOCKS_div2 (
    FP_GOLDILOCKS * x,
    FP_GOLDILOCKS * y )
```

## Parameters

<i>x</i>	FP number, on exit $=y/2 \bmod \text{Modulus}$
<i>y</i>	FP number

## 8.32.3.6 FP\_GOLDILOCKS\_equals()

```
int FP_GOLDILOCKS_equals (
    FP_GOLDILOCKS * x,
    FP_GOLDILOCKS * y )
```

## Parameters

<i>x</i>	FP number
<i>y</i>	FP number

## Returns

1 if equal, else returns 0

## 8.32.3.7 FP\_GOLDILOCKS\_imul()

```
void FP_GOLDILOCKS_imul (
    FP_GOLDILOCKS * x,
    FP_GOLDILOCKS * y,
    int i )
```

## Parameters

<i>x</i>	FP number, on exit the modular product $= y*i \bmod \text{Modulus}$
<i>y</i>	FP number, the multiplicand
<i>i</i>	a small number, the multiplier



### 8.32.3.8 FP\_GOLDILOCKS\_inv()

```
void FP_GOLDILOCKS_inv (
    FP_GOLDILOCKS * x,
    FP_GOLDILOCKS * y )
```

#### Parameters

<i>x</i>	FP number, on exit = 1/y mod Modulus
<i>y</i>	FP number

### 8.32.3.9 FP\_GOLDILOCKS\_iszilch()

```
int FP_GOLDILOCKS_iszilch (
    FP_GOLDILOCKS * x )
```

#### Parameters

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

#### Returns

1 if zero, else returns 0

### 8.32.3.10 FP\_GOLDILOCKS\_mod()

```
void FP_GOLDILOCKS_mod (
    BIG_448_58 r,
    DBIG_448_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 mod Modulus
<i>d</i>	DBIG number to be reduced

## 8.32.3.11 FP\_GOLDILOCKS\_mul()

```
void FP_GOLDILOCKS_mul (
    FP_GOLDILOCKS * x,
    FP_GOLDILOCKS * y,
    FP_GOLDILOCKS * z )
```

Uses appropriate fast modular reduction method

## Parameters

x	FP number, on exit the modular product = y*z mod Modulus
y	FP number, the multiplicand
z	FP number, the multiplier

## 8.32.3.12 FP\_GOLDILOCKS\_neg()

```
void FP_GOLDILOCKS_neg (
    FP_GOLDILOCKS * x,
    FP_GOLDILOCKS * y )
```

## Parameters

x	FP number, on exit = -y mod Modulus
y	FP number

## 8.32.3.13 FP\_GOLDILOCKS\_norm()

```
void FP_GOLDILOCKS_norm (
    FP_GOLDILOCKS * x )
```

## Parameters

x	FP number, on exit normalized
---	-------------------------------

## 8.32.3.14 FP\_GOLDILOCKS\_nres()

```
void FP_GOLDILOCKS_nres (
    FP_GOLDILOCKS * y,
    BIG_448_58 x )
```

## Parameters

<i>x</i>	BIG number to be converted
<i>y</i>	FP result

## 8.32.3.15 FP\_GOLDILOCKS\_one()

```
void FP_GOLDILOCKS_one (
    FP_GOLDILOCKS * x )
```

## Parameters

<i>x</i>	FP number to be set equal to unity.
----------	-------------------------------------

## 8.32.3.16 FP\_GOLDILOCKS\_output()

```
void FP_GOLDILOCKS_output (
    FP_GOLDILOCKS * x )
```

Converts from residue form before output

## Parameters

<i>x</i>	an FP number
----------	--------------

## 8.32.3.17 FP\_GOLDILOCKS\_pow()

```
void FP_GOLDILOCKS_pow (
    FP_GOLDILOCKS * x,
    FP_GOLDILOCKS * y,
    BIG_448_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

### 8.32.3.18 FP\_GOLDILOCKS\_qr()

```
int FP_GOLDILOCKS_qr (
    FP_GOLDILOCKS * x )
```

#### Parameters

<i>x</i>	FP number to be tested
----------	------------------------

#### Returns

1 if quadratic residue, else returns 0 if quadratic non-residue

### 8.32.3.19 FP\_GOLDILOCKS\_rawoutput()

```
void FP_GOLDILOCKS_rawoutput (
    FP_GOLDILOCKS * x )
```

#### Parameters

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

### 8.32.3.20 FP\_GOLDILOCKS\_rcopy()

```
void FP_GOLDILOCKS_rcopy (
    FP_GOLDILOCKS * y,
    const BIG_448_58 x )
```

#### Parameters

<i>y</i>	FP number to be copied to
<i>x</i>	BIG to be copied from ROM

### 8.32.3.21 FP\_GOLDILOCKS\_redc()

```
void FP_GOLDILOCKS_redc (
    BIG_448_58 x,
    FP_GOLDILOCKS * y )
```

#### Parameters

<i>y</i>	FP number to be converted to BIG
<i>x</i>	BIG result

### 8.32.3.22 FP\_GOLDILOCKS\_reduce()

```
void FP_GOLDILOCKS_reduce (
    FP_GOLDILOCKS * x )
```

#### Parameters

x	FP number, on exit reduced mod Modulus
---	--

### 8.32.3.23 FP\_GOLDILOCKS\_sqr()

```
void FP_GOLDILOCKS_sqr (
    FP_GOLDILOCKS * x,
    FP_GOLDILOCKS * y )
```

Uses appropriate fast modular reduction method

#### Parameters

x	FP number, on exit the modular product = $y^2 \bmod \text{Modulus}$
y	FP number, the number to be squared

### 8.32.3.24 FP\_GOLDILOCKS\_sqrt()

```
void FP_GOLDILOCKS_sqrt (
    FP_GOLDILOCKS * x,
    FP_GOLDILOCKS * y )
```

#### Parameters

x	FP number, on exit = $\text{sqrt}(y) \bmod \text{Modulus}$
y	FP number, the number whose square root is calculated

### 8.32.3.25 FP\_GOLDILOCKS\_sub()

```
void FP_GOLDILOCKS_sub (
    FP_GOLDILOCKS * x,
    FP_GOLDILOCKS * y,
    FP_GOLDILOCKS * z )
```

## Parameters

<i>x</i>	FP number, on exit the modular difference = $y - z \bmod \text{Modulus}$
<i>y</i>	FP number
<i>z</i>	FP number

## 8.32.3.26 FP\_GOLDILOCKS\_zero()

```
void FP_GOLDILOCKS_zero (
    FP_GOLDILOCKS * x )
```

## Parameters

<i>x</i>	FP number to be set to 0
----------	--------------------------

## 8.32.4 Variable Documentation

## 8.32.4.1 MConst\_GOLDILOCKS

```
const chunk MConst_GOLDILOCKS
```

Constant associated with Modulus - for Montgomery =  $1/p \bmod 2^{\text{BASEBITS}}$

## 8.32.4.2 Modulus\_GOLDILOCKS

```
const BIG_448_58 Modulus_GOLDILOCKS
```

Actual Modulus set in romf\_yyy.c

## 8.32.4.3 R2modp\_GOLDILOCKS

```
const BIG_448_58 R2modp_GOLDILOCKS
```

Montgomery constant

## 8.33 fp\_NIST256.h File Reference

FP Header File.

```
#include "big_256_56.h"
#include "config_field_NIST256.h"
```

## Data Structures

- struct [FP\\_NIST256](#)  
*FP Structure - quadratic extension field.*

## Macros

- #define [MODBITS\\_NIST256](#) MBITS\_NIST256
- #define [TBITS\\_NIST256](#) (MBITS\_NIST256%BASEBITS\_256\_56)
- #define [TMASK\\_NIST256](#) (((chunk)1<<TBITS\_NIST256)-1)
- #define [FEXCESS\\_NIST256](#) (((sign32)1<<MAXXES\_NIST256)-1)
- #define [OMASK\\_NIST256](#) (-((chunk)(1)<<TBITS\_NIST256))

## Functions

- int [FP\\_NIST256\\_iszilch](#) ([FP\\_NIST256](#) \*x)  
*Tests for FP equal to zero mod Modulus.*
- void [FP\\_NIST256\\_zero](#) ([FP\\_NIST256](#) \*x)  
*Set FP to zero.*
- void [FP\\_NIST256\\_copy](#) ([FP\\_NIST256](#) \*y, [FP\\_NIST256](#) \*x)  
*Copy an FP.*
- void [FP\\_NIST256\\_rcopy](#) ([FP\\_NIST256](#) \*y, const [BIG\\_256\\_56](#) x)  
*Copy from ROM to an FP.*
- int [FP\\_NIST256\\_equals](#) ([FP\\_NIST256](#) \*x, [FP\\_NIST256](#) \*y)  
*Compares two FPs.*
- void [FP\\_NIST256\\_cswap](#) ([FP\\_NIST256](#) \*x, [FP\\_NIST256](#) \*y, int s)  
*Conditional constant time swap of two FP numbers.*
- void [FP\\_NIST256\\_cmove](#) ([FP\\_NIST256](#) \*x, [FP\\_NIST256](#) \*y, int s)  
*Conditional copy of FP number.*
- void [FP\\_NIST256\\_nres](#) ([FP\\_NIST256](#) \*y, [BIG\\_256\\_56](#) x)  
*Converts from BIG integer to residue form mod Modulus.*
- void [FP\\_NIST256\\_redc](#) ([BIG\\_256\\_56](#) x, [FP\\_NIST256](#) \*y)  
*Converts from residue form back to BIG integer form.*
- void [FP\\_NIST256\\_one](#) ([FP\\_NIST256](#) \*x)  
*Sets FP to representation of unity in residue form.*
- void [FP\\_NIST256\\_mod](#) ([BIG\\_256\\_56](#) r, [DBIG\\_256\\_56](#) d)  
*Reduces DBIG to BIG exploiting special form of the modulus.*
- void [FP\\_NIST256\\_mul](#) ([FP\\_NIST256](#) \*x, [FP\\_NIST256](#) \*y, [FP\\_NIST256](#) \*z)  
*Fast Modular multiplication of two FPs, mod Modulus.*
- void [FP\\_NIST256\\_imul](#) ([FP\\_NIST256](#) \*x, [FP\\_NIST256](#) \*y, int i)  
*Fast Modular multiplication of an FP, by a small integer, mod Modulus.*
- void [FP\\_NIST256\\_sqr](#) ([FP\\_NIST256](#) \*x, [FP\\_NIST256](#) \*y)  
*Fast Modular squaring of an FP, mod Modulus.*
- void [FP\\_NIST256\\_add](#) ([FP\\_NIST256](#) \*x, [FP\\_NIST256](#) \*y, [FP\\_NIST256](#) \*z)  
*Modular addition of two FPs, mod Modulus.*
- void [FP\\_NIST256\\_sub](#) ([FP\\_NIST256](#) \*x, [FP\\_NIST256](#) \*y, [FP\\_NIST256](#) \*z)  
*Modular subtraction of two FPs, mod Modulus.*
- void [FP\\_NIST256\\_div2](#) ([FP\\_NIST256](#) \*x, [FP\\_NIST256](#) \*y)  
*Modular division by 2 of an FP, mod Modulus.*
- void [FP\\_NIST256\\_pow](#) ([FP\\_NIST256](#) \*x, [FP\\_NIST256](#) \*y, [BIG\\_256\\_56](#) z)

- *Fast Modular exponentiation of an FP, to the power of a BIG, mod Modulus.*  
 • void `FP_NIST256_sqrt` (`FP_NIST256 *x`, `FP_NIST256 *y`)
- *Fast Modular square root of a an FP, mod Modulus.*  
 • void `FP_NIST256_neg` (`FP_NIST256 *x`, `FP_NIST256 *y`)
- *Modular negation of a an FP, mod Modulus.*  
 • void `FP_NIST256_output` (`FP_NIST256 *x`)
- *Outputs an FP number to the console.*  
 • void `FP_NIST256_rawoutput` (`FP_NIST256 *x`)
- *Outputs an FP number to the console, in raw form.*  
 • void `FP_NIST256_reduce` (`FP_NIST256 *x`)
- *Reduces possibly unreduced FP mod Modulus.*  
 • void `FP_NIST256_norm` (`FP_NIST256 *x`)
- *normalizes FP*  
 • int `FP_NIST256_qr` (`FP_NIST256 *x`)
- *Tests for FP a quadratic residue mod Modulus.*  
 • void `FP_NIST256_inv` (`FP_NIST256 *x`, `FP_NIST256 *y`)
- *Modular inverse of a an FP, mod Modulus.*

## Variables

- const `BIG_256_56 Modulus_NIST256`
- const `BIG_256_56 R2modp_NIST256`
- const `chunk MConst_NIST256`

### 8.33.1 Detailed Description

#### Author

Mike Scott

### 8.33.2 Macro Definition Documentation

#### 8.33.2.1 FEXCESS\_NIST256

```
#define FEXCESS_NIST256 ((sign32)1<<MAXXES_NIST256)-1)
```

$2^{(BASEBITS*NLEN-MODBITS)}-1$  - normalised BIG can be multiplied by less than this before reduction

#### 8.33.2.2 MODBITS\_NIST256

```
#define MODBITS_NIST256 MBITS_NIST256
```

Number of bits in Modulus for selected curve



## 8.33.2.3 OMASK\_NIST256

```
#define OMASK_NIST256 ( - ( (chunk) (1) << TBITS_NIST256) )
```

for masking out overflow bits

## 8.33.2.4 TBITS\_NIST256

```
#define TBITS_NIST256 (MBITS_NIST256%BASEBITS_256_56)
```

Number of active bits in top word

## 8.33.2.5 TMASK\_NIST256

```
#define TMASK_NIST256 ( ( (chunk) 1 << TBITS_NIST256) - 1 )
```

Mask for active bits in top word

## 8.33.3 Function Documentation

## 8.33.3.1 FP\_NIST256\_add()

```
void FP_NIST256_add (
    FP_NIST256 * x,
    FP_NIST256 * y,
    FP_NIST256 * z )
```

## Parameters

<i>x</i>	FP number, on exit the modular sum = y+z mod Modulus
<i>y</i>	FP number
<i>z</i>	FP number

## 8.33.3.2 FP\_NIST256\_cmove()

```
void FP_NIST256_cmove (
    FP_NIST256 * x,
    FP_NIST256 * 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

**8.33.3.3 FP\_NIST256\_copy()**

```
void FP_NIST256_copy (
    FP_NIST256 * y,
    FP_NIST256 * x )
```

**Parameters**

<i>y</i>	FP number to be copied to
<i>x</i>	FP to be copied from

**8.33.3.4 FP\_NIST256\_cswap()**

```
void FP_NIST256_cswap (
    FP_NIST256 * x,
    FP_NIST256 * 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

**8.33.3.5 FP\_NIST256\_div2()**

```
void FP_NIST256_div2 (
    FP_NIST256 * x,
    FP_NIST256 * y )
```

**Parameters**

<i>x</i>	FP number, on exit =y/2 mod Modulus
<i>y</i>	FP number

### 8.33.3.6 FP\_NIST256\_equals()

```
int FP_NIST256_equals (
    FP_NIST256 * x,
    FP_NIST256 * y )
```

#### Parameters

<i>x</i>	FP number
<i>y</i>	FP number

#### Returns

1 if equal, else returns 0

### 8.33.3.7 FP\_NIST256\_imul()

```
void FP_NIST256_imul (
    FP_NIST256 * x,
    FP_NIST256 * y,
    int i )
```

#### Parameters

<i>x</i>	FP number, on exit the modular product = $y*i \bmod \text{Modulus}$
<i>y</i>	FP number, the multiplicand
<i>i</i>	a small number, the multiplier

### 8.33.3.8 FP\_NIST256\_inv()

```
void FP_NIST256_inv (
    FP_NIST256 * x,
    FP_NIST256 * y )
```

#### Parameters

<i>x</i>	FP number, on exit = $1/y \bmod \text{Modulus}$
<i>y</i>	FP number

### 8.33.3.9 FP\_NIST256\_iszilch()

```
int FP_NIST256_iszilch (
    FP_NIST256 * x )
```

#### Parameters

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

#### Returns

1 if zero, else returns 0

### 8.33.3.10 FP\_NIST256\_mod()

```
void FP_NIST256_mod (
    BIG_256_56 r,
    DBIG_256_56 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

### 8.33.3.11 FP\_NIST256\_mul()

```
void FP_NIST256_mul (
    FP_NIST256 * x,
    FP_NIST256 * y,
    FP_NIST256 * 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

#### 8.33.3.12 FP\_NIST256\_neg()

```
void FP_NIST256_neg (
    FP_NIST256 * x,
    FP_NIST256 * y )
```

##### Parameters

<i>x</i>	FP number, on exit = -y mod Modulus
<i>y</i>	FP number

#### 8.33.3.13 FP\_NIST256\_norm()

```
void FP_NIST256_norm (
    FP_NIST256 * x )
```

##### Parameters

<i>x</i>	FP number, on exit normalized
----------	-------------------------------

#### 8.33.3.14 FP\_NIST256\_nres()

```
void FP_NIST256_nres (
    FP_NIST256 * y,
    BIG_256_56 x )
```

##### Parameters

<i>x</i>	BIG number to be converted
<i>y</i>	FP result

#### 8.33.3.15 FP\_NIST256\_one()

```
void FP_NIST256_one (
    FP_NIST256 * x )
```

##### Parameters

<i>x</i>	FP number to be set equal to unity.
----------	-------------------------------------

**8.33.3.16 FP\_NIST256\_output()**

```
void FP_NIST256_output (
    FP_NIST256 * x )
```

Converts from residue form before output

**Parameters**

<i>x</i>	an FP number
----------	--------------

**8.33.3.17 FP\_NIST256\_pow()**

```
void FP_NIST256_pow (
    FP_NIST256 * x,
    FP_NIST256 * y,
    BIG_256_56 z )
```

**Parameters**

<i>x</i>	FP number, on exit = $y^x \bmod \text{Modulus}$
<i>y</i>	FP number
<i>z</i>	BIG number exponent

**8.33.3.18 FP\_NIST256\_qr()**

```
int FP_NIST256_qr (
    FP_NIST256 * x )
```

**Parameters**

<i>x</i>	FP number to be tested
----------	------------------------

**Returns**

1 if quadratic residue, else returns 0 if quadratic non-residue

**8.33.3.19 FP\_NIST256\_rawoutput()**

```
void FP_NIST256_rawoutput (
    FP_NIST256 * x )
```

## Parameters

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

## 8.33.3.20 FP\_NIST256\_rcopy()

```
void FP_NIST256_rcopy (
    FP_NIST256 * y,
    const BIG_256_56 x )
```

## Parameters

<i>y</i>	FP number to be copied to
<i>x</i>	BIG to be copied from ROM

## 8.33.3.21 FP\_NIST256\_redc()

```
void FP_NIST256_redc (
    BIG_256_56 x,
    FP_NIST256 * y )
```

## Parameters

<i>y</i>	FP number to be converted to BIG
<i>x</i>	BIG result

## 8.33.3.22 FP\_NIST256\_reduce()

```
void FP_NIST256_reduce (
    FP_NIST256 * x )
```

## Parameters

<i>x</i>	FP number, on exit reduced mod Modulus
----------	--

## 8.33.3.23 FP\_NIST256\_sqr()

```
void FP_NIST256_sqr (
```

```
FP_NIST256 * x,  
FP_NIST256 * 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

#### 8.33.3.24 FP\_NIST256\_sqrt()

```
void FP_NIST256_sqrt (  
    FP_NIST256 * x,  
    FP_NIST256 * y )
```

#### Parameters

<i>x</i>	FP number, on exit = $\text{sqrt}(y) \bmod \text{Modulus}$
<i>y</i>	FP number, the number whose square root is calculated

#### 8.33.3.25 FP\_NIST256\_sub()

```
void FP_NIST256_sub (  
    FP_NIST256 * x,  
    FP_NIST256 * y,  
    FP_NIST256 * z )
```

#### Parameters

<i>x</i>	FP number, on exit the modular difference = $y - z \bmod \text{Modulus}$
<i>y</i>	FP number
<i>z</i>	FP number

#### 8.33.3.26 FP\_NIST256\_zero()

```
void FP_NIST256_zero (  
    FP_NIST256 * x )
```

#### Parameters

<i>x</i>	FP number to be set to 0
----------	--------------------------



### 8.33.4 Variable Documentation

#### 8.33.4.1 MConst\_NIST256

```
const chunk MConst_NIST256
```

Constant associated with Modulus - for Montgomery =  $1/p \bmod 2^{\text{BASEBITS}}$

#### 8.33.4.2 Modulus\_NIST256

```
const BIG_256_56 Modulus_NIST256
```

Actual Modulus set in romf\_yyy.c

#### 8.33.4.3 R2modp\_NIST256

```
const BIG_256_56 R2modp_NIST256
```

Montgomery constant

## 8.34 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.*

### 8.34.1 Detailed Description

Author

Mike Scott

### 8.34.2 Macro Definition Documentation

#### 8.34.2.1 M\_SIZE\_BLS381

```
#define M_SIZE_BLS381 (MESSAGE_SIZE+2*PFS_BLS381+1)
```

Signature message size and G1 size

#### 8.34.2.2 MAXPIN

```
#define MAXPIN 10000
```

max PIN

#### 8.34.2.3 MESSAGE\_SIZE

```
#define MESSAGE_SIZE 256
```

Signature message size

#### 8.34.2.4 MPIN\_BAD\_PIN

```
#define MPIN_BAD_PIN -19
```

Bad PIN number entered

#### 8.34.2.5 MPIN\_INVALID\_POINT

```
#define MPIN_INVALID_POINT -14
```

Point is NOT on the curve

#### 8.34.2.6 MPIN\_OK

```
#define MPIN_OK 0
```

Function completed without error

#### 8.34.2.7 MPIN\_PAS

```
#define MPIN_PAS 16
```

MPIN Symmetric Key Size

#### 8.34.2.8 PBLLEN

```
#define PBLLEN 14
```

max length of PIN in bits

#### 8.34.2.9 PFS\_BLS381

```
#define PFS_BLS381 MODBYTES_384_58
```

MPIN Field Size

#### 8.34.2.10 PGS\_BLS381

```
#define PGS_BLS381 MODBYTES_384_58
```

MPIN Group Size

### 8.34.3 Function Documentation

#### 8.34.3.1 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

<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 $R \neq \text{NULL}$ , 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

## 8.34.3.2 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  $\text{NULL}$ . If Time Permits are enabled, and PIN error detection is OFF,  $U$  is not generated and can be set to  $\text{NULL}$ . 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 $R \neq \text{NULL}$ , 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)

**Parameters**

<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

**8.34.3.3 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

**8.34.3.4 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

## Parameters

<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

## 8.34.3.5 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

## 8.34.3.6 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

#### 8.34.3.7 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

#### 8.34.3.8 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

#### 8.34.3.9 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

**8.34.3.10 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

**8.34.3.11 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 $R \neq \text{NULL}$ , otherwise it must be provided as an input
<i>Pa</i>	the output public key for the client

## 8.34.3.12 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!=NULL, otherwise must be provided as an input
<i>G</i>	if type=0 a point in G1, else an octet to be mapped to G1
<i>W</i>	the output =x.G or x.M(G), where M(.) is a mapping

## Returns

0 or an error code

## 8.34.3.13 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!=NULL, otherwise must be provided as an input
<i>G</i>	a point in G2
<i>W</i>	the output =x.G or (1/x).G

## Returns

0 or an error code

## 8.34.3.14 MPIN\_BLS381\_GET\_SERVER\_SECRET()

```
int MPIN_BLS381_GET_SERVER_SECRET (
    octet * S,
    octet * SS )
```

## Parameters

$S$	is an input master secret
$SS$	is the server secret = $s.Q$ where $Q$ is a fixed generator of $G_2$

## Returns

0 or an error code

## 8.34.3.15 MPIN\_BLS381\_GET\_Y()

```
void MPIN_BLS381_GET_Y (
    int h,
    int t,
    octet * O,
    octet * Y )
```

## Parameters

$h$	is the hash type
$t$	is epoch time in seconds
$O$	is an input octet
$Y$	is the output octet

## 8.34.3.16 MPIN\_BLS381\_KANGAROO()

```
int MPIN_BLS381_KANGAROO (
    octet * E,
    octet * F )
```

## Parameters

$E$	a member of the group $GT$
$F$	a member of the group $GT = E^e$

## Returns

0 if Kangaroos failed, or the PIN error  $e$

## 8.34.3.17 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

**8.34.3.18 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

**8.34.3.19 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

**8.34.3.20 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

**8.34.3.21 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

## 8.34.3.22 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

$h$	is the hash type
$d$	is input date, in days since the epoch. Set to 0 if Time permits disabled
$HID$	is output $H(ID)$ , a hash of the client ID
$HTID$	is output $H(ID)+H(d H(ID))$
$y$	is output $H(t U)$ or $H(t UT)$ if Time Permits enabled
$SS$	is the input server secret
$U$	is input from the client = $x.H(ID)$
$UT$	is input from the client = $x.(H(ID)+H(d H(ID)))$
$V$	is an input from the client
$E$	is an output to help the Kangaroos to find the PIN error, or <code>NULL</code> if not required
$F$	is an output to help the Kangaroos to find the PIN error, or <code>NULL</code> if not required
$ID$	is the input claimed client identity
$MESSAGE$	is the message to be signed
$t$	is input epoch time in seconds - a timestamp
$Pa$	is input from the client $z.Q$ or <code>NULL</code> if the key-escrow less scheme is not used

## Returns

0 or an error code

## 8.34.3.23 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

#### 8.34.3.24 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

**8.34.3.25 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

**8.35 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]`

### 8.35.1 Detailed Description

#### Author

Mike Scott

### 8.35.2 Function Documentation

#### 8.35.2.1 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

## 8.35.2.2 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

## 8.35.2.3 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

## 8.35.2.4 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 = $x^{((p^{12}-1)/r)}$
-----	--------------------------------------

## 8.35.2.5 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

## 8.35.2.6 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

## 8.35.2.7 PAIR\_BLS381\_GTmember()

```
int PAIR_BLS381_GTmember (
    FP12_BLS381 *  $x$  )
```

## Parameters

$x$	FP12 instance
-----	---------------

**Returns**

1 if  $x$  is in GT, else return 0

**8.35.2.8 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

**8.35.2.9 PAIR\_BLS381\_initmp()**

```
void PAIR_BLS381_initmp (
    FP12_BLS381  $r[ ]$  )
```

**Parameters**

$r$	FP12 array, to be initialised to 1
-----	------------------------------------

**8.35.2.10 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

**8.35.2.11 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

**8.35.3 Variable Documentation****8.35.3.1 CURVE\_BB\_BLS381**

```
const BIG_384_58 CURVE_BB_BLS381[4][4]
```

BN curve constant for GS decomposition

**8.35.3.2 CURVE\_Bnx\_BLS381**

```
const BIG_384_58 CURVE_Bnx_BLS381
```

BN curve x parameter

**8.35.3.3 CURVE\_Cru\_BLS381**

```
const BIG_384_58 CURVE_Cru_BLS381
```

BN curve Cube Root of Unity

**8.35.3.4 CURVE\_SB\_BLS381**

```
const BIG_384_58 CURVE_SB_BLS381[2][2]
```

BN curve constant for GLV decomposition

**8.35.3.5 CURVE\_W\_BLS381**

```
const BIG_384_58 CURVE_W_BLS381[2]
```

BN curve constant for GLV decomposition

### 8.35.3.6 CURVE\_WB\_BLS381

```
const BIG_384_58 CURVE_WB_BLS381[4]
```

BN curve constant for GS decomposition

## 8.36 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  $w = \text{hash}(n|x)$*
- `unsign32 today` (void)  
*Supply today's date as days from the epoch.*
- void `HASH_ALL` (int h, `octet *l`, `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.*

### 8.36.1 Detailed Description

#### Author

Mike Scott

### 8.36.2 Macro Definition Documentation



### 8.36.2.1 TIME\_SLOT\_MINUTES

```
#define TIME_SLOT_MINUTES 1440
```

Time Slot = 1 day

## 8.36.3 Function Documentation

### 8.36.3.1 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

### 8.36.3.2 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

### 8.36.3.3 GET\_TIME()

```

unsign32 GET_TIME (
    void )

```

#### Returns

current epoch time in seconds

### 8.36.3.4 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

### 8.36.3.5 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

## 8.36.3.6 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>	octect involved in the h ash
<i>w</i>	output

## 8.36.3.7 today()

```
unsigned32 today (
    void )
```

## Returns

today's date, as number of days elapsed since the epoch

## 8.37 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.*

### 8.37.1 Detailed Description

#### Author

Mike Scott

### 8.37.2 Function Documentation

#### 8.37.2.1 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

#### 8.37.2.2 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
----------	--

## 8.38 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.*

### 8.38.1 Detailed Description

#### Author

Mike Scott declares functions

### 8.38.2 Macro Definition Documentation

#### 8.38.2.1 HASH\_TYPE\_RSA\_2048

```
#define HASH_TYPE_RSA_2048 SHA256
```

Chosen Hash algorithm

#### 8.38.2.2 RFS\_2048

```
#define RFS_2048 MODBYTES_1024_58*FFLEN_2048
```

RSA Public Key Size in bytes

### 8.38.3 Function Documentation

#### 8.38.3.1 RSA\_2048\_DECRYPT()

```
void RSA_2048_DECRYPT (
    rsa_private_key_2048 * PRIV,
    octet * G,
    octet * F )
```

## Parameters

$P \leftarrow RIV$	the input RSA private key
$G$	is the input ciphertext
$F$	is output plaintext (requires unpadding)

## 8.38.3.2 RSA\_2048\_ENCRYPT()

```
void RSA_2048_ENCRYPT (
    rsa_public_key_2048 * PUB,
    octet * F,
    octet * G )
```

## Parameters

$PUB$	the input RSA public key
$F$	is input padded message
$G$	is the output ciphertext

## 8.38.3.3 RSA\_2048\_fromOctet()

```
void RSA_2048_fromOctet (
    BIG_1024_58 * x,
    octet * S )
```

Creates RSA public key from big-endian base 256 form.

## Parameters

$x$	FF instance to be created from an octet string
$S$	input octet string

## 8.38.3.4 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

$R$	is a pointer to a cryptographically secure random number generator
$e$	the encryption exponent
$P \leftrightarrow RIV$	the output RSA private key
$PUB$	the output RSA public key
$P$	Input prime number. Used when R is equal to NULL for testing
$Q$	Input prime number. Used when R is equal to NULL for testing

## 8.38.3.5 RSA\_2048\_PRIVATE\_KEY\_KILL()

```
void RSA_2048_PRIVATE_KEY_KILL (
    rsa_private_key_2048 * PRIV )
```

## Parameters

$P \leftrightarrow RIV$	the input RSA private key. Destroyed on output.
-------------------------	---

## 8.39 rsa\_3072.h File Reference

RSA Header file for implementation of RSA protocol.

```
#include "ff_3072.h"
#include "rsa_support.h"
```

## Data Structures

- struct [rsa\\_public\\_key\\_3072](#)  
*Integer Factorisation Public Key.*
- struct [rsa\\_private\\_key\\_3072](#)  
*Integer Factorisation Private Key.*

## Macros

- #define [HASH\\_TYPE\\_RSA\\_3072](#) SHA256
- #define [RFS\\_3072](#) MODBYTES\_384\_56\*FFLEN\_3072

## Functions

- void [RSA\\_3072\\_KEY\\_PAIR](#) (csprng \*R, sign32 e, rsa\_private\_key\_3072 \*PRIV, rsa\_public\_key\_3072 \*PUB, [octet](#) \*P, [octet](#) \*Q)  
*RSA Key Pair Generator.*
- void [RSA\\_3072\\_ENCRYPT](#) (rsa\_public\_key\_3072 \*PUB, [octet](#) \*F, [octet](#) \*G)  
*RSA encryption of suitably padded plaintext.*
- void [RSA\\_3072\\_DECRYPT](#) (rsa\_private\_key\_3072 \*PRIV, [octet](#) \*G, [octet](#) \*F)  
*RSA decryption of ciphertext.*
- void [RSA\\_3072\\_PRIVATE\\_KEY\\_KILL](#) (rsa\_private\_key\_3072 \*PRIV)  
*Destroy an RSA private Key.*
- void [RSA\\_3072\\_fromOctet](#) ([BIG\\_384\\_56](#) \*x, [octet](#) \*S)  
*Populates an RSA public key from an octet string.*

### 8.39.1 Detailed Description

#### Author

Mike Scott declares functions

### 8.39.2 Macro Definition Documentation

#### 8.39.2.1 HASH\_TYPE\_RSA\_3072

```
#define HASH_TYPE_RSA_3072 SHA256
```

Chosen Hash algorithm

#### 8.39.2.2 RFS\_3072

```
#define RFS_3072 MODBYTES_384_56*FFLEN_3072
```

RSA Public Key Size in bytes

### 8.39.3 Function Documentation

#### 8.39.3.1 RSA\_3072\_DECRYPT()

```
void RSA_3072_DECRYPT (
    rsa_private_key_3072 * PRIV,
    octet * G,
    octet * F )
```



## Parameters

$P \leftarrow RIV$	the input RSA private key
$G$	is the input ciphertext
$F$	is output plaintext (requires unpadding)

## 8.39.3.2 RSA\_3072\_ENCRYPT()

```
void RSA_3072_ENCRYPT (
    rsa_public_key_3072 * PUB,
    octet * F,
    octet * G )
```

## Parameters

$PUB$	the input RSA public key
$F$	is input padded message
$G$	is the output ciphertext

## 8.39.3.3 RSA\_3072\_fromOctet()

```
void RSA_3072_fromOctet (
    BIG_384_56 * x,
    octet * S )
```

Creates RSA public key from big-endian base 256 form.

## Parameters

$x$	FF instance to be created from an octet string
$S$	input octet string

## 8.39.3.4 RSA\_3072\_KEY\_PAIR()

```
void RSA_3072_KEY_PAIR (
    csprng * R,
    sign32 e,
    rsa_private_key_3072 * PRIV,
    rsa_public_key_3072 * 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 R is equal to NULL for testing
<i>Q</i>	Input prime number. Used when R is equal to NULL for testing

## 8.39.3.5 RSA\_3072\_PRIVATE\_KEY\_KILL()

```
void RSA_3072_PRIVATE_KEY_KILL (
    rsa_private_key_3072 * PRIV )
```

## Parameters

<i>P</i> ↔ <i>RIV</i>	the input RSA private key. Destroyed on output.
--------------------------	---

## 8.40 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.*

## 8.40.1 Detailed Description

## Author

Mike Scott

## 8.40.2 Macro Definition Documentation

### 8.40.2.1 MAX\_RSA\_BYTES

```
#define MAX_RSA_BYTES 512
```

Maximum of 4096

## 8.40.3 Function Documentation

### 8.40.3.1 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

### 8.40.3.2 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

**8.40.3.3 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

**8.41 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.*

### 8.41.1 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.

### 8.41.2 Function Documentation

#### 8.41.2.1 `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

#### 8.41.2.2 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

#### 8.41.2.3 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

#### 8.41.2.4 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

## 8.41.2.5 generateRandom()

```
void generateRandom (
    csprng * RNG,
    octet * randomValue )
```

Generate a random Octet.

## Parameters

<i>RNG</i>	random number generator
<i>randomValue</i>	random Octet

## 8.42 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.*

## 8.42.1 Detailed Description

## Author

Kealan McCusker

## 8.42.2 Function Documentation

## 8.42.2.1 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

**8.42.2.2 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

**8.42.2.3 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

**8.42.2.4 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

## 8.42.2.5 generateRandom()

```
void generateRandom (
    csprng * RNG,
    octet * randomValue )
```

Generate a random Octet.

## Parameters

<i>RNG</i>	random number generator
<i>randomValue</i>	random Octet

## 8.43 version.c File Reference

AMCL version support function.

```
#include "version.h"
```

## Functions

- void `amcl_version` (void)  
*Print version number and information about the build.*

### 8.43.1 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.

## 8.43.2 Function Documentation

### 8.43.2.1 amcl\_version()

```
void amcl_version (
    void )
```

Print version number and information about the build.

## 8.44 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.*

### 8.44.1 Detailed Description

#### Author

Mike Scott  
Kealan McCusker

### 8.44.2 Macro Definition Documentation

#### 8.44.2.1 PIV

```
#define PIV 12
```

AES-GCM Initialization Vector Size

#### 8.44.2.2 PTAG

```
#define PTAG 16
```

AES-GCM MAC Size

#### 8.44.2.3 TIME\_SLOT\_MINUTES

```
#define TIME_SLOT_MINUTES 1440
```

Time Slot = 1 day

#### 8.44.2.4 WCC\_INVALID\_POINT

```
#define WCC_INVALID_POINT -51
```

Point is NOT on the curve

#### 8.44.2.5 WCC\_OK

```
#define WCC_OK 0
```

Function completed without error

#### 8.44.2.6 WCC\_PFS\_BLS381

```
#define WCC_PFS_BLS381 MODBYTES_384_58
```

WCC Field Size

#### 8.44.2.7 WCC\_PGS\_BLS381

```
#define WCC_PGS_BLS381 MODBYTES_384_58
```

WCC Group Size

### 8.44.3 Function Documentation

#### 8.44.3.1 WCC\_BLS381\_GET\_G1\_MULTIPLE()

```
int WCC_BLS381_GET_G1_MULTIPLE (
    octet * S,
    octet * HID,
    octet * VG1 )
```

Calculate a value in G1.  $VG1 = s \cdot H1(ID)$  where ID is the identity.

1.  $VG1 = s \cdot H1(ID)$

#### 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 $VG1 = s \cdot H1(ID)$

**Returns**

rtn Returns 0 if successful or else an error code

**8.44.3.2 WCC\_BLS381\_GET\_G2\_MULTIPLE()**

```
int WCC_BLS381_GET_G2_MULTIPLE (
    octet * S,
    octet * HID,
    octet * VG2 )
```

Calculate a value in G2.  $VG2 = s \cdot H2(ID)$  where ID is the identity.

1.  $VG2 = s \cdot H2(ID)$

**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 $VG2 = s \cdot H2(ID)$

**Returns**

rtn Returns 0 if successful or else an error code

**8.44.3.3 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

#### 8.44.3.4 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

#### 8.44.3.5 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.  $j = e(\text{pia}, \text{AG1} + \text{PaG1}, (y + \text{pib}).\text{BKeyG2})$
2.  $K = H(j, w, \text{PaG1})$

##### 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>	$x.\text{AG1}$ where $x < q$
<i>PgG1Oct</i>	$w.\text{AG1}$ 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

**8.44.3.6 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

**8.44.3.7 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

### 8.44.3.8 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

## 8.45 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)`

### 8.45.1 Detailed Description

#### Author

Mike Scott

### 8.45.2 Function Documentation

#### 8.45.2.1 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

#### 8.45.2.2 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)

**8.45.2.3 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)

**8.45.2.4 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

#### 8.45.2.5 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

#### 8.45.2.6 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

#### 8.45.2.7 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

#### 8.45.2.8 X509\_find\_subject()

```
int X509_find_subject (
    octet * c )
```

##### Parameters

<i>c</i>	an X.509 certificate
----------	----------------------

##### Returns

0 on failure, or pointer to subject field in cert

#### 8.45.2.9 X509\_find\_validity()

```
int X509_find_validity (
    octet * c )
```

##### Parameters

<i>c</i>	an X.509 certificate
----------	----------------------

##### Returns

0 on failure, or pointer to validity field in cert

# Index

## a

- FP12\_BLS381, [22](#)
- FP2\_BLS381, [23](#)
- FP4\_BLS381, [23](#)
- gcm, [27](#)
- AES\_CBC\_IV0\_DECRYPT
  - ecdh\_support.h, [208](#)
- AES\_CBC\_IV0\_ENCRYPT
  - ecdh\_support.h, [208](#)
- AES\_GCM\_DECRYPT
  - pbk\_support.h, [411](#)
- AES\_GCM\_ENCRYPT
  - pbk\_support.h, [411](#)
- amcl\_aes, [15](#)
  - f, [15](#)
  - fkey, [15](#)
  - mode, [15](#)
  - Nk, [16](#)
  - Nr, [16](#)
  - rkey, [16](#)
- amcl\_bin2hex
  - utils.c, [423](#)
  - utils.h, [425](#)
- amcl\_hex2bin
  - utils.c, [423](#)
  - utils.h, [426](#)
- amcl\_print\_hex
  - utils.c, [424](#)
  - utils.h, [426](#)
- amcl\_version
  - version.c, [428](#)
- arch.h, [37](#)
  - byte, [37](#)
  - CHUNK, [38](#)
  - chunk, [38](#)
  - sign16, [38](#)
  - sign32, [38](#)
  - sign64, [38](#)
  - sign8, [38](#)
  - uchar, [38](#)
  - unsign32, [39](#)
  - unsign64, [39](#)

## b

- FP12\_BLS381, [22](#)
- FP2\_BLS381, [23](#)
- FP4\_BLS381, [24](#)
- BASEBITS\_1024\_58
  - config\_big\_1024\_58.h, [183](#)
- BASEBITS\_256\_56

- config\_big\_256\_56.h, [183](#)
- BASEBITS\_384\_56
  - config\_big\_384\_56.h, [184](#)
- BASEBITS\_384\_58
  - config\_big\_384\_58.h, [185](#)
- BASEBITS\_448\_58
  - config\_big\_448\_58.h, [186](#)
- BFS\_BLS381
  - bls\_BLS381.h, [179](#)
- BGS\_BLS381
  - bls\_BLS381.h, [179](#)
- BIG\_1024\_58
  - big\_1024\_58.h, [44](#)
- BIG\_1024\_58\_add
  - big\_1024\_58.h, [44](#)
- BIG\_1024\_58\_bit
  - big\_1024\_58.h, [44](#)
- BIG\_1024\_58\_cmove
  - big\_1024\_58.h, [45](#)
- BIG\_1024\_58\_comp
  - big\_1024\_58.h, [45](#)
- BIG\_1024\_58\_copy
  - big\_1024\_58.h, [45](#)
- BIG\_1024\_58\_cswap
  - big\_1024\_58.h, [46](#)
- BIG\_1024\_58\_dadd
  - big\_1024\_58.h, [46](#)
- BIG\_1024\_58\_dcmove
  - big\_1024\_58.h, [46](#)
- BIG\_1024\_58\_dcomp
  - big\_1024\_58.h, [47](#)
- BIG\_1024\_58\_dcopy
  - big\_1024\_58.h, [47](#)
- BIG\_1024\_58\_ddiv
  - big\_1024\_58.h, [47](#)
- BIG\_1024\_58\_dec
  - big\_1024\_58.h, [48](#)
- BIG\_1024\_58\_dfromBytesLen
  - big\_1024\_58.h, [48](#)
- BIG\_1024\_58\_diszilch
  - big\_1024\_58.h, [48](#)
- BIG\_1024\_58\_div3
  - big\_1024\_58.h, [48](#)
- BIG\_1024\_58\_dmod
  - big\_1024\_58.h, [50](#)
- BIG\_1024\_58\_dmod2m
  - big\_1024\_58.h, [50](#)
- BIG\_1024\_58\_dnbits
  - big\_1024\_58.h, [50](#)

BIG\_1024\_58\_dnorm  
    big\_1024\_58.h, [51](#)  
BIG\_1024\_58\_doutput  
    big\_1024\_58.h, [51](#)  
BIG\_1024\_58\_drawoutput  
    big\_1024\_58.h, [51](#)  
BIG\_1024\_58\_dscopy  
    big\_1024\_58.h, [51](#)  
BIG\_1024\_58\_dshl  
    big\_1024\_58.h, [52](#)  
BIG\_1024\_58\_dshr  
    big\_1024\_58.h, [52](#)  
BIG\_1024\_58\_dsub  
    big\_1024\_58.h, [52](#)  
BIG\_1024\_58\_dsucopy  
    big\_1024\_58.h, [53](#)  
BIG\_1024\_58\_dzero  
    big\_1024\_58.h, [53](#)  
BIG\_1024\_58\_fromBytes  
    big\_1024\_58.h, [53](#)  
BIG\_1024\_58\_fromBytesLen  
    big\_1024\_58.h, [53](#)  
BIG\_1024\_58\_fshl  
    big\_1024\_58.h, [54](#)  
BIG\_1024\_58\_fshr  
    big\_1024\_58.h, [54](#)  
BIG\_1024\_58\_imul  
    big\_1024\_58.h, [55](#)  
BIG\_1024\_58\_inc  
    big\_1024\_58.h, [55](#)  
BIG\_1024\_58\_invmod2m  
    big\_1024\_58.h, [55](#)  
BIG\_1024\_58\_invmodp  
    big\_1024\_58.h, [55](#)  
BIG\_1024\_58\_isunity  
    big\_1024\_58.h, [56](#)  
BIG\_1024\_58\_iszilch  
    big\_1024\_58.h, [56](#)  
BIG\_1024\_58\_jacobi  
    big\_1024\_58.h, [56](#)  
BIG\_1024\_58\_lastbits  
    big\_1024\_58.h, [57](#)  
BIG\_1024\_58\_mod  
    big\_1024\_58.h, [57](#)  
BIG\_1024\_58\_mod2m  
    big\_1024\_58.h, [57](#)  
BIG\_1024\_58\_moddiv  
    big\_1024\_58.h, [58](#)  
BIG\_1024\_58\_modmul  
    big\_1024\_58.h, [58](#)  
BIG\_1024\_58\_modneg  
    big\_1024\_58.h, [59](#)  
BIG\_1024\_58\_modsqr  
    big\_1024\_58.h, [59](#)  
BIG\_1024\_58\_monty  
    big\_1024\_58.h, [59](#)  
BIG\_1024\_58\_mul  
    big\_1024\_58.h, [60](#)  
BIG\_1024\_58\_nbits  
    big\_1024\_58.h, [60](#)  
BIG\_1024\_58\_norm  
    big\_1024\_58.h, [60](#)  
BIG\_1024\_58\_one  
    big\_1024\_58.h, [60](#)  
BIG\_1024\_58\_or  
    big\_1024\_58.h, [61](#)  
BIG\_1024\_58\_output  
    big\_1024\_58.h, [61](#)  
BIG\_1024\_58\_parity  
    big\_1024\_58.h, [61](#)  
BIG\_1024\_58\_pmul  
    big\_1024\_58.h, [62](#)  
BIG\_1024\_58\_pxmuls  
    big\_1024\_58.h, [62](#)  
BIG\_1024\_58\_random  
    big\_1024\_58.h, [62](#)  
BIG\_1024\_58\_randomnum  
    big\_1024\_58.h, [63](#)  
BIG\_1024\_58\_rawoutput  
    big\_1024\_58.h, [63](#)  
BIG\_1024\_58\_rcopy  
    big\_1024\_58.h, [63](#)  
BIG\_1024\_58\_sdcopy  
    big\_1024\_58.h, [63](#)  
BIG\_1024\_58\_sdiv  
    big\_1024\_58.h, [64](#)  
BIG\_1024\_58\_sducopy  
    big\_1024\_58.h, [64](#)  
BIG\_1024\_58\_shl  
    big\_1024\_58.h, [64](#)  
BIG\_1024\_58\_shr  
    big\_1024\_58.h, [65](#)  
BIG\_1024\_58\_smul  
    big\_1024\_58.h, [65](#)  
BIG\_1024\_58\_split  
    big\_1024\_58.h, [65](#)  
BIG\_1024\_58\_sqr  
    big\_1024\_58.h, [66](#)  
BIG\_1024\_58\_ssn  
    big\_1024\_58.h, [66](#)  
BIG\_1024\_58\_sub  
    big\_1024\_58.h, [66](#)  
BIG\_1024\_58\_toBytes  
    big\_1024\_58.h, [67](#)  
BIG\_1024\_58\_zero  
    big\_1024\_58.h, [67](#)  
BIG\_256\_56  
    big\_256\_56.h, [72](#)  
BIG\_256\_56\_add  
    big\_256\_56.h, [72](#)  
BIG\_256\_56\_bit  
    big\_256\_56.h, [73](#)  
BIG\_256\_56\_cmove  
    big\_256\_56.h, [73](#)  
BIG\_256\_56\_comp  
    big\_256\_56.h, [73](#)

BIG\_256\_56\_copy  
big\_256\_56.h, [74](#)

BIG\_256\_56\_cswap  
big\_256\_56.h, [74](#)

BIG\_256\_56\_dadd  
big\_256\_56.h, [74](#)

BIG\_256\_56\_dcmove  
big\_256\_56.h, [75](#)

BIG\_256\_56\_dcomp  
big\_256\_56.h, [75](#)

BIG\_256\_56\_dcopy  
big\_256\_56.h, [75](#)

BIG\_256\_56\_ddiv  
big\_256\_56.h, [76](#)

BIG\_256\_56\_dec  
big\_256\_56.h, [76](#)

BIG\_256\_56\_dfromBytesLen  
big\_256\_56.h, [76](#)

BIG\_256\_56\_diszilch  
big\_256\_56.h, [76](#)

BIG\_256\_56\_div3  
big\_256\_56.h, [77](#)

BIG\_256\_56\_dmod  
big\_256\_56.h, [77](#)

BIG\_256\_56\_dmod2m  
big\_256\_56.h, [77](#)

BIG\_256\_56\_dnbits  
big\_256\_56.h, [78](#)

BIG\_256\_56\_dnorm  
big\_256\_56.h, [78](#)

BIG\_256\_56\_doutput  
big\_256\_56.h, [78](#)

BIG\_256\_56\_drawoutput  
big\_256\_56.h, [79](#)

BIG\_256\_56\_dscopy  
big\_256\_56.h, [79](#)

BIG\_256\_56\_dshl  
big\_256\_56.h, [79](#)

BIG\_256\_56\_dshr  
big\_256\_56.h, [79](#)

BIG\_256\_56\_dsub  
big\_256\_56.h, [80](#)

BIG\_256\_56\_dscopy  
big\_256\_56.h, [80](#)

BIG\_256\_56\_dzero  
big\_256\_56.h, [80](#)

BIG\_256\_56\_fromBytes  
big\_256\_56.h, [81](#)

BIG\_256\_56\_fromBytesLen  
big\_256\_56.h, [81](#)

BIG\_256\_56\_fshl  
big\_256\_56.h, [81](#)

BIG\_256\_56\_fshr  
big\_256\_56.h, [81](#)

BIG\_256\_56\_imul  
big\_256\_56.h, [82](#)

BIG\_256\_56\_inc  
big\_256\_56.h, [82](#)

BIG\_256\_56\_invmod2m  
big\_256\_56.h, [82](#)

BIG\_256\_56\_invmodp  
big\_256\_56.h, [83](#)

BIG\_256\_56\_isunity  
big\_256\_56.h, [83](#)

BIG\_256\_56\_iszilch  
big\_256\_56.h, [83](#)

BIG\_256\_56\_jacobi  
big\_256\_56.h, [84](#)

BIG\_256\_56\_lastbits  
big\_256\_56.h, [84](#)

BIG\_256\_56\_mod  
big\_256\_56.h, [84](#)

BIG\_256\_56\_mod2m  
big\_256\_56.h, [85](#)

BIG\_256\_56\_moddiv  
big\_256\_56.h, [85](#)

BIG\_256\_56\_modmul  
big\_256\_56.h, [85](#)

BIG\_256\_56\_modneg  
big\_256\_56.h, [86](#)

BIG\_256\_56\_modsqr  
big\_256\_56.h, [86](#)

BIG\_256\_56\_monty  
big\_256\_56.h, [87](#)

BIG\_256\_56\_mul  
big\_256\_56.h, [87](#)

BIG\_256\_56\_nbits  
big\_256\_56.h, [87](#)

BIG\_256\_56\_norm  
big\_256\_56.h, [87](#)

BIG\_256\_56\_one  
big\_256\_56.h, [88](#)

BIG\_256\_56\_or  
big\_256\_56.h, [88](#)

BIG\_256\_56\_output  
big\_256\_56.h, [88](#)

BIG\_256\_56\_parity  
big\_256\_56.h, [89](#)

BIG\_256\_56\_pmul  
big\_256\_56.h, [89](#)

BIG\_256\_56\_pxmuls  
big\_256\_56.h, [89](#)

BIG\_256\_56\_random  
big\_256\_56.h, [90](#)

BIG\_256\_56\_randomnum  
big\_256\_56.h, [90](#)

BIG\_256\_56\_rawoutput  
big\_256\_56.h, [90](#)

BIG\_256\_56\_rcopy  
big\_256\_56.h, [90](#)

BIG\_256\_56\_sdcopy  
big\_256\_56.h, [91](#)

BIG\_256\_56\_sdiv  
big\_256\_56.h, [91](#)

BIG\_256\_56\_sdcopy  
big\_256\_56.h, [91](#)

BIG\_256\_56\_shl  
big\_256\_56.h, [92](#)

BIG\_256\_56\_shr  
big\_256\_56.h, [92](#)

BIG\_256\_56\_smul  
big\_256\_56.h, [92](#)

BIG\_256\_56\_split  
big\_256\_56.h, [92](#)

BIG\_256\_56\_sqr  
big\_256\_56.h, [93](#)

BIG\_256\_56\_ssn  
big\_256\_56.h, [93](#)

BIG\_256\_56\_sub  
big\_256\_56.h, [94](#)

BIG\_256\_56\_toBytes  
big\_256\_56.h, [94](#)

BIG\_256\_56\_zero  
big\_256\_56.h, [94](#)

BIG\_384\_56  
big\_384\_56.h, [99](#)

BIG\_384\_56\_add  
big\_384\_56.h, [99](#)

BIG\_384\_56\_bit  
big\_384\_56.h, [100](#)

BIG\_384\_56\_cmove  
big\_384\_56.h, [100](#)

BIG\_384\_56\_comp  
big\_384\_56.h, [100](#)

BIG\_384\_56\_copy  
big\_384\_56.h, [101](#)

BIG\_384\_56\_cswap  
big\_384\_56.h, [101](#)

BIG\_384\_56\_dadd  
big\_384\_56.h, [101](#)

BIG\_384\_56\_dcmove  
big\_384\_56.h, [102](#)

BIG\_384\_56\_dcomp  
big\_384\_56.h, [102](#)

BIG\_384\_56\_dcopy  
big\_384\_56.h, [102](#)

BIG\_384\_56\_ddiv  
big\_384\_56.h, [103](#)

BIG\_384\_56\_dec  
big\_384\_56.h, [103](#)

BIG\_384\_56\_dfromBytesLen  
big\_384\_56.h, [103](#)

BIG\_384\_56\_diszilch  
big\_384\_56.h, [104](#)

BIG\_384\_56\_div3  
big\_384\_56.h, [104](#)

BIG\_384\_56\_dmod  
big\_384\_56.h, [104](#)

BIG\_384\_56\_dmod2m  
big\_384\_56.h, [105](#)

BIG\_384\_56\_dnbits  
big\_384\_56.h, [105](#)

BIG\_384\_56\_dnorm  
big\_384\_56.h, [105](#)

BIG\_384\_56\_doutput  
big\_384\_56.h, [105](#)

BIG\_384\_56\_drawoutput  
big\_384\_56.h, [106](#)

BIG\_384\_56\_dscopy  
big\_384\_56.h, [106](#)

BIG\_384\_56\_dshl  
big\_384\_56.h, [106](#)

BIG\_384\_56\_dshr  
big\_384\_56.h, [106](#)

BIG\_384\_56\_dsub  
big\_384\_56.h, [107](#)

BIG\_384\_56\_dsucopy  
big\_384\_56.h, [107](#)

BIG\_384\_56\_dzero  
big\_384\_56.h, [107](#)

BIG\_384\_56\_fromBytes  
big\_384\_56.h, [108](#)

BIG\_384\_56\_fromBytesLen  
big\_384\_56.h, [108](#)

BIG\_384\_56\_fshl  
big\_384\_56.h, [108](#)

BIG\_384\_56\_fshr  
big\_384\_56.h, [109](#)

BIG\_384\_56\_imul  
big\_384\_56.h, [109](#)

BIG\_384\_56\_inc  
big\_384\_56.h, [109](#)

BIG\_384\_56\_invmod2m  
big\_384\_56.h, [110](#)

BIG\_384\_56\_invmodp  
big\_384\_56.h, [110](#)

BIG\_384\_56\_isunity  
big\_384\_56.h, [110](#)

BIG\_384\_56\_iszilch  
big\_384\_56.h, [110](#)

BIG\_384\_56\_jacobi  
big\_384\_56.h, [112](#)

BIG\_384\_56\_lastbits  
big\_384\_56.h, [112](#)

BIG\_384\_56\_mod  
big\_384\_56.h, [112](#)

BIG\_384\_56\_mod2m  
big\_384\_56.h, [113](#)

BIG\_384\_56\_moddiv  
big\_384\_56.h, [113](#)

BIG\_384\_56\_modmul  
big\_384\_56.h, [113](#)

BIG\_384\_56\_modneg  
big\_384\_56.h, [114](#)

BIG\_384\_56\_modsqr  
big\_384\_56.h, [114](#)

BIG\_384\_56\_monty  
big\_384\_56.h, [115](#)

BIG\_384\_56\_mul  
big\_384\_56.h, [115](#)

BIG\_384\_56\_nbits  
big\_384\_56.h, [115](#)



BIG\_384\_56\_norm  
    big\_384\_56.h, [115](#)  
BIG\_384\_56\_one  
    big\_384\_56.h, [116](#)  
BIG\_384\_56\_or  
    big\_384\_56.h, [116](#)  
BIG\_384\_56\_output  
    big\_384\_56.h, [116](#)  
BIG\_384\_56\_parity  
    big\_384\_56.h, [117](#)  
BIG\_384\_56\_pmul  
    big\_384\_56.h, [117](#)  
BIG\_384\_56\_pxmulo  
    big\_384\_56.h, [117](#)  
BIG\_384\_56\_random  
    big\_384\_56.h, [118](#)  
BIG\_384\_56\_randomnum  
    big\_384\_56.h, [118](#)  
BIG\_384\_56\_rawoutput  
    big\_384\_56.h, [118](#)  
BIG\_384\_56\_rcopy  
    big\_384\_56.h, [118](#)  
BIG\_384\_56\_sdcopy  
    big\_384\_56.h, [119](#)  
BIG\_384\_56\_sdiv  
    big\_384\_56.h, [119](#)  
BIG\_384\_56\_sducopy  
    big\_384\_56.h, [119](#)  
BIG\_384\_56\_shl  
    big\_384\_56.h, [120](#)  
BIG\_384\_56\_shr  
    big\_384\_56.h, [120](#)  
BIG\_384\_56\_smulo  
    big\_384\_56.h, [120](#)  
BIG\_384\_56\_split  
    big\_384\_56.h, [120](#)  
BIG\_384\_56\_sqr  
    big\_384\_56.h, [121](#)  
BIG\_384\_56\_ssn  
    big\_384\_56.h, [121](#)  
BIG\_384\_56\_sub  
    big\_384\_56.h, [122](#)  
BIG\_384\_56\_toBytes  
    big\_384\_56.h, [122](#)  
BIG\_384\_56\_zero  
    big\_384\_56.h, [122](#)  
BIG\_384\_58  
    big\_384\_58.h, [127](#)  
BIG\_384\_58\_add  
    big\_384\_58.h, [127](#)  
BIG\_384\_58\_bit  
    big\_384\_58.h, [128](#)  
BIG\_384\_58\_cmove  
    big\_384\_58.h, [128](#)  
BIG\_384\_58\_comp  
    big\_384\_58.h, [128](#)  
BIG\_384\_58\_copy  
    big\_384\_58.h, [129](#)  
BIG\_384\_58\_cswap  
    big\_384\_58.h, [129](#)  
BIG\_384\_58\_dadd  
    big\_384\_58.h, [129](#)  
BIG\_384\_58\_dcmulo  
    big\_384\_58.h, [130](#)  
BIG\_384\_58\_dcomp  
    big\_384\_58.h, [130](#)  
BIG\_384\_58\_dcopy  
    big\_384\_58.h, [130](#)  
BIG\_384\_58\_ddiv  
    big\_384\_58.h, [131](#)  
BIG\_384\_58\_dec  
    big\_384\_58.h, [131](#)  
BIG\_384\_58\_dfromBytesLen  
    big\_384\_58.h, [131](#)  
BIG\_384\_58\_diszilo  
    big\_384\_58.h, [132](#)  
BIG\_384\_58\_div3  
    big\_384\_58.h, [132](#)  
BIG\_384\_58\_dmod  
    big\_384\_58.h, [132](#)  
BIG\_384\_58\_dmod2m  
    big\_384\_58.h, [133](#)  
BIG\_384\_58\_dnbits  
    big\_384\_58.h, [133](#)  
BIG\_384\_58\_dnorm  
    big\_384\_58.h, [133](#)  
BIG\_384\_58\_doutput  
    big\_384\_58.h, [133](#)  
BIG\_384\_58\_drawoutput  
    big\_384\_58.h, [134](#)  
BIG\_384\_58\_dscopy  
    big\_384\_58.h, [134](#)  
BIG\_384\_58\_dshl  
    big\_384\_58.h, [134](#)  
BIG\_384\_58\_dshr  
    big\_384\_58.h, [134](#)  
BIG\_384\_58\_dsub  
    big\_384\_58.h, [135](#)  
BIG\_384\_58\_dsucopy  
    big\_384\_58.h, [135](#)  
BIG\_384\_58\_dzero  
    big\_384\_58.h, [135](#)  
BIG\_384\_58\_fromBytes  
    big\_384\_58.h, [136](#)  
BIG\_384\_58\_fromBytesLen  
    big\_384\_58.h, [136](#)  
BIG\_384\_58\_fshl  
    big\_384\_58.h, [136](#)  
BIG\_384\_58\_fshr  
    big\_384\_58.h, [137](#)  
BIG\_384\_58\_imulo  
    big\_384\_58.h, [137](#)  
BIG\_384\_58\_inc  
    big\_384\_58.h, [137](#)  
BIG\_384\_58\_invmod2m  
    big\_384\_58.h, [138](#)

BIG\_384\_58\_invmodp  
big\_384\_58.h, 138

BIG\_384\_58\_isunity  
big\_384\_58.h, 138

BIG\_384\_58\_iszilch  
big\_384\_58.h, 138

BIG\_384\_58\_jacobi  
big\_384\_58.h, 140

BIG\_384\_58\_lastbits  
big\_384\_58.h, 140

BIG\_384\_58\_mod  
big\_384\_58.h, 140

BIG\_384\_58\_mod2m  
big\_384\_58.h, 141

BIG\_384\_58\_moddiv  
big\_384\_58.h, 141

BIG\_384\_58\_modmul  
big\_384\_58.h, 141

BIG\_384\_58\_modneg  
big\_384\_58.h, 142

BIG\_384\_58\_modsqr  
big\_384\_58.h, 142

BIG\_384\_58\_monty  
big\_384\_58.h, 143

BIG\_384\_58\_mul  
big\_384\_58.h, 143

BIG\_384\_58\_nbits  
big\_384\_58.h, 143

BIG\_384\_58\_norm  
big\_384\_58.h, 143

BIG\_384\_58\_one  
big\_384\_58.h, 144

BIG\_384\_58\_or  
big\_384\_58.h, 144

BIG\_384\_58\_output  
big\_384\_58.h, 144

BIG\_384\_58\_parity  
big\_384\_58.h, 145

BIG\_384\_58\_pmul  
big\_384\_58.h, 145

BIG\_384\_58\_pxmuls  
big\_384\_58.h, 145

BIG\_384\_58\_random  
big\_384\_58.h, 146

BIG\_384\_58\_randomnum  
big\_384\_58.h, 146

BIG\_384\_58\_rawoutput  
big\_384\_58.h, 146

BIG\_384\_58\_rcopy  
big\_384\_58.h, 146

BIG\_384\_58\_sdcopy  
big\_384\_58.h, 147

BIG\_384\_58\_sdiv  
big\_384\_58.h, 147

BIG\_384\_58\_sducopys  
big\_384\_58.h, 147

BIG\_384\_58\_shl  
big\_384\_58.h, 148

BIG\_384\_58\_shr  
big\_384\_58.h, 148

BIG\_384\_58\_smuls  
big\_384\_58.h, 148

BIG\_384\_58\_split  
big\_384\_58.h, 148

BIG\_384\_58\_sqr  
big\_384\_58.h, 149

BIG\_384\_58\_ssn  
big\_384\_58.h, 149

BIG\_384\_58\_sub  
big\_384\_58.h, 150

BIG\_384\_58\_toBytes  
big\_384\_58.h, 150

BIG\_384\_58\_zero  
big\_384\_58.h, 150

BIG\_448\_58  
big\_448\_58.h, 155

BIG\_448\_58\_add  
big\_448\_58.h, 155

BIG\_448\_58\_bit  
big\_448\_58.h, 156

BIG\_448\_58\_cmove  
big\_448\_58.h, 156

BIG\_448\_58\_comp  
big\_448\_58.h, 156

BIG\_448\_58\_copy  
big\_448\_58.h, 157

BIG\_448\_58\_cswap  
big\_448\_58.h, 157

BIG\_448\_58\_dadd  
big\_448\_58.h, 157

BIG\_448\_58\_dcmoves  
big\_448\_58.h, 158

BIG\_448\_58\_dcomp  
big\_448\_58.h, 158

BIG\_448\_58\_dcopy  
big\_448\_58.h, 158

BIG\_448\_58\_ddiv  
big\_448\_58.h, 159

BIG\_448\_58\_dec  
big\_448\_58.h, 159

BIG\_448\_58\_dfromBytesLen  
big\_448\_58.h, 159

BIG\_448\_58\_diszilch  
big\_448\_58.h, 160

BIG\_448\_58\_div3  
big\_448\_58.h, 160

BIG\_448\_58\_dmod  
big\_448\_58.h, 160

BIG\_448\_58\_dmod2m  
big\_448\_58.h, 161

BIG\_448\_58\_dnbits  
big\_448\_58.h, 161

BIG\_448\_58\_dnorm  
big\_448\_58.h, 161

BIG\_448\_58\_doutput  
big\_448\_58.h, 161

- BIG\_448\_58\_drawoutput
  - big\_448\_58.h, [162](#)
- BIG\_448\_58\_dscopy
  - big\_448\_58.h, [162](#)
- BIG\_448\_58\_dshl
  - big\_448\_58.h, [162](#)
- BIG\_448\_58\_dshr
  - big\_448\_58.h, [162](#)
- BIG\_448\_58\_dsub
  - big\_448\_58.h, [163](#)
- BIG\_448\_58\_dsucopy
  - big\_448\_58.h, [163](#)
- BIG\_448\_58\_dzero
  - big\_448\_58.h, [163](#)
- BIG\_448\_58\_fromBytes
  - big\_448\_58.h, [164](#)
- BIG\_448\_58\_fromBytesLen
  - big\_448\_58.h, [164](#)
- BIG\_448\_58\_fshl
  - big\_448\_58.h, [164](#)
- BIG\_448\_58\_fshr
  - big\_448\_58.h, [165](#)
- BIG\_448\_58\_imul
  - big\_448\_58.h, [165](#)
- BIG\_448\_58\_inc
  - big\_448\_58.h, [165](#)
- BIG\_448\_58\_invmod2m
  - big\_448\_58.h, [166](#)
- BIG\_448\_58\_invmodp
  - big\_448\_58.h, [166](#)
- BIG\_448\_58\_isunity
  - big\_448\_58.h, [166](#)
- BIG\_448\_58\_iszilch
  - big\_448\_58.h, [166](#)
- BIG\_448\_58\_jacobi
  - big\_448\_58.h, [168](#)
- BIG\_448\_58\_lastbits
  - big\_448\_58.h, [168](#)
- BIG\_448\_58\_mod
  - big\_448\_58.h, [168](#)
- BIG\_448\_58\_mod2m
  - big\_448\_58.h, [169](#)
- BIG\_448\_58\_moddiv
  - big\_448\_58.h, [169](#)
- BIG\_448\_58\_modmul
  - big\_448\_58.h, [169](#)
- BIG\_448\_58\_modneg
  - big\_448\_58.h, [170](#)
- BIG\_448\_58\_modsqr
  - big\_448\_58.h, [170](#)
- BIG\_448\_58\_monty
  - big\_448\_58.h, [171](#)
- BIG\_448\_58\_mul
  - big\_448\_58.h, [171](#)
- BIG\_448\_58\_nbits
  - big\_448\_58.h, [171](#)
- BIG\_448\_58\_norm
  - big\_448\_58.h, [171](#)
- BIG\_448\_58\_one
  - big\_448\_58.h, [172](#)
- BIG\_448\_58\_or
  - big\_448\_58.h, [172](#)
- BIG\_448\_58\_output
  - big\_448\_58.h, [172](#)
- BIG\_448\_58\_parity
  - big\_448\_58.h, [173](#)
- BIG\_448\_58\_pmul
  - big\_448\_58.h, [173](#)
- BIG\_448\_58\_pxmuls
  - big\_448\_58.h, [173](#)
- BIG\_448\_58\_random
  - big\_448\_58.h, [174](#)
- BIG\_448\_58\_randomnum
  - big\_448\_58.h, [174](#)
- BIG\_448\_58\_rawoutput
  - big\_448\_58.h, [174](#)
- BIG\_448\_58\_rcopy
  - big\_448\_58.h, [174](#)
- BIG\_448\_58\_sdcopy
  - big\_448\_58.h, [175](#)
- BIG\_448\_58\_sdiv
  - big\_448\_58.h, [175](#)
- BIG\_448\_58\_sducopy
  - big\_448\_58.h, [175](#)
- BIG\_448\_58\_shl
  - big\_448\_58.h, [176](#)
- BIG\_448\_58\_shr
  - big\_448\_58.h, [176](#)
- BIG\_448\_58\_smul
  - big\_448\_58.h, [176](#)
- BIG\_448\_58\_split
  - big\_448\_58.h, [176](#)
- BIG\_448\_58\_sqr
  - big\_448\_58.h, [177](#)
- BIG\_448\_58\_ssn
  - big\_448\_58.h, [177](#)
- BIG\_448\_58\_sub
  - big\_448\_58.h, [178](#)
- BIG\_448\_58\_toBytes
  - big\_448\_58.h, [178](#)
- BIG\_448\_58\_zero
  - big\_448\_58.h, [178](#)
- BIGBITS\_1024\_58
  - big\_1024\_58.h, [43](#)
- BIGBITS\_256\_56
  - big\_256\_56.h, [71](#)
- BIGBITS\_384\_56
  - big\_384\_56.h, [98](#)
- BIGBITS\_384\_58
  - big\_384\_58.h, [126](#)
- BIGBITS\_448\_58
  - big\_448\_58.h, [154](#)
- BLS\_BLS381\_ADD\_G1
  - bls\_BLS381.h, [180](#)
- BLS\_BLS381\_ADD\_G2
  - bls\_BLS381.h, [181](#)

BLS\_BLS381\_KEY\_PAIR\_GENERATE  
     bls\_BLS381.h, 181  
 BLS\_BLS381\_SIGN  
     bls\_BLS381.h, 181  
 BLS\_BLS381\_VERIFY  
     bls\_BLS381.h, 182  
 BLS\_FAIL  
     bls\_BLS381.h, 180  
 BLS\_INVALID\_G1  
     bls\_BLS381.h, 180  
 BLS\_INVALID\_G2  
     bls\_BLS381.h, 180  
 BLS\_OK  
     bls\_BLS381.h, 180  
 BMASK\_1024\_58  
     big\_1024\_58.h, 43  
 BMASK\_256\_56  
     big\_256\_56.h, 71  
 BMASK\_384\_56  
     big\_384\_56.h, 98  
 BMASK\_384\_58  
     big\_384\_58.h, 126  
 BMASK\_448\_58  
     big\_448\_58.h, 154  
 big\_1024\_58.h, 39  
     BIG\_1024\_58, 44  
     BIG\_1024\_58\_add, 44  
     BIG\_1024\_58\_bit, 44  
     BIG\_1024\_58\_cmove, 45  
     BIG\_1024\_58\_comp, 45  
     BIG\_1024\_58\_copy, 45  
     BIG\_1024\_58\_cswap, 46  
     BIG\_1024\_58\_dadd, 46  
     BIG\_1024\_58\_dcmove, 46  
     BIG\_1024\_58\_dcomp, 47  
     BIG\_1024\_58\_dcopy, 47  
     BIG\_1024\_58\_ddiv, 47  
     BIG\_1024\_58\_dec, 48  
     BIG\_1024\_58\_dfromBytesLen, 48  
     BIG\_1024\_58\_diszilch, 48  
     BIG\_1024\_58\_div3, 48  
     BIG\_1024\_58\_dmod, 50  
     BIG\_1024\_58\_dmod2m, 50  
     BIG\_1024\_58\_dnbits, 50  
     BIG\_1024\_58\_dnorm, 51  
     BIG\_1024\_58\_doutput, 51  
     BIG\_1024\_58\_drawoutput, 51  
     BIG\_1024\_58\_dscopy, 51  
     BIG\_1024\_58\_dshl, 52  
     BIG\_1024\_58\_dshr, 52  
     BIG\_1024\_58\_dsub, 52  
     BIG\_1024\_58\_dsucopy, 53  
     BIG\_1024\_58\_dzero, 53  
     BIG\_1024\_58\_fromBytes, 53  
     BIG\_1024\_58\_fromBytesLen, 53  
     BIG\_1024\_58\_fshl, 54  
     BIG\_1024\_58\_fshr, 54  
     BIG\_1024\_58\_imul, 55  
     BIG\_1024\_58\_inc, 55  
     BIG\_1024\_58\_invmod2m, 55  
     BIG\_1024\_58\_invmodp, 55  
     BIG\_1024\_58\_isunity, 56  
     BIG\_1024\_58\_iszilch, 56  
     BIG\_1024\_58\_jacobi, 56  
     BIG\_1024\_58\_lastbits, 57  
     BIG\_1024\_58\_mod, 57  
     BIG\_1024\_58\_mod2m, 57  
     BIG\_1024\_58\_moddiv, 58  
     BIG\_1024\_58\_modmul, 58  
     BIG\_1024\_58\_modneg, 59  
     BIG\_1024\_58\_modsqr, 59  
     BIG\_1024\_58\_monty, 59  
     BIG\_1024\_58\_mul, 60  
     BIG\_1024\_58\_nbits, 60  
     BIG\_1024\_58\_norm, 60  
     BIG\_1024\_58\_one, 60  
     BIG\_1024\_58\_or, 61  
     BIG\_1024\_58\_output, 61  
     BIG\_1024\_58\_parity, 61  
     BIG\_1024\_58\_pmul, 62  
     BIG\_1024\_58\_pxm, 62  
     BIG\_1024\_58\_random, 62  
     BIG\_1024\_58\_randomnum, 63  
     BIG\_1024\_58\_rawoutput, 63  
     BIG\_1024\_58\_rcopy, 63  
     BIG\_1024\_58\_sdcopy, 63  
     BIG\_1024\_58\_sdiv, 64  
     BIG\_1024\_58\_sducopy, 64  
     BIG\_1024\_58\_shl, 64  
     BIG\_1024\_58\_shr, 65  
     BIG\_1024\_58\_smul, 65  
     BIG\_1024\_58\_split, 65  
     BIG\_1024\_58\_sqr, 66  
     BIG\_1024\_58\_ssn, 66  
     BIG\_1024\_58\_sub, 66  
     BIG\_1024\_58\_toBytes, 67  
     BIG\_1024\_58\_zero, 67  
 BIGBITS\_1024\_58, 43  
 BMASK\_1024\_58, 43  
 DBIG\_1024\_58, 44  
 DNLEN\_1024\_58, 43  
 HBITS\_1024\_58, 43  
 HMASK\_1024\_58, 43  
 NEXCESS\_1024\_58, 43  
 NLEN\_1024\_58, 43  
 big\_256\_56.h, 67  
     BIG\_256\_56, 72  
     BIG\_256\_56\_add, 72  
     BIG\_256\_56\_bit, 73  
     BIG\_256\_56\_cmove, 73  
     BIG\_256\_56\_comp, 73  
     BIG\_256\_56\_copy, 74  
     BIG\_256\_56\_cswap, 74  
     BIG\_256\_56\_dadd, 74  
     BIG\_256\_56\_dcmove, 75  
     BIG\_256\_56\_dcomp, 75

BIG\_256\_56\_dcopy, 75  
BIG\_256\_56\_ddiv, 76  
BIG\_256\_56\_dec, 76  
BIG\_256\_56\_dfromBytesLen, 76  
BIG\_256\_56\_diszilch, 76  
BIG\_256\_56\_div3, 77  
BIG\_256\_56\_dmod, 77  
BIG\_256\_56\_dmod2m, 77  
BIG\_256\_56\_dnbits, 78  
BIG\_256\_56\_dnorm, 78  
BIG\_256\_56\_doutput, 78  
BIG\_256\_56\_drawoutput, 79  
BIG\_256\_56\_dscopy, 79  
BIG\_256\_56\_dshl, 79  
BIG\_256\_56\_dshr, 79  
BIG\_256\_56\_dsub, 80  
BIG\_256\_56\_dsucopy, 80  
BIG\_256\_56\_dzero, 80  
BIG\_256\_56\_fromBytes, 81  
BIG\_256\_56\_fromBytesLen, 81  
BIG\_256\_56\_fshl, 81  
BIG\_256\_56\_fshr, 81  
BIG\_256\_56\_imul, 82  
BIG\_256\_56\_inc, 82  
BIG\_256\_56\_invmod2m, 82  
BIG\_256\_56\_invmodp, 83  
BIG\_256\_56\_isunity, 83  
BIG\_256\_56\_iszilch, 83  
BIG\_256\_56\_jacobi, 84  
BIG\_256\_56\_lastbits, 84  
BIG\_256\_56\_mod, 84  
BIG\_256\_56\_mod2m, 85  
BIG\_256\_56\_moddiv, 85  
BIG\_256\_56\_modmul, 85  
BIG\_256\_56\_modneg, 86  
BIG\_256\_56\_modsq, 86  
BIG\_256\_56\_monty, 87  
BIG\_256\_56\_mul, 87  
BIG\_256\_56\_nbits, 87  
BIG\_256\_56\_norm, 87  
BIG\_256\_56\_one, 88  
BIG\_256\_56\_or, 88  
BIG\_256\_56\_output, 88  
BIG\_256\_56\_parity, 89  
BIG\_256\_56\_pmul, 89  
BIG\_256\_56\_pxm, 89  
BIG\_256\_56\_random, 90  
BIG\_256\_56\_randomnum, 90  
BIG\_256\_56\_rawoutput, 90  
BIG\_256\_56\_rcopy, 90  
BIG\_256\_56\_sdcopy, 91  
BIG\_256\_56\_sdiv, 91  
BIG\_256\_56\_sducopy, 91  
BIG\_256\_56\_shl, 92  
BIG\_256\_56\_shr, 92  
BIG\_256\_56\_smul, 92  
BIG\_256\_56\_split, 92  
BIG\_256\_56\_sqr, 93  
BIG\_256\_56\_ssn, 93  
BIG\_256\_56\_sub, 94  
BIG\_256\_56\_toBytes, 94  
BIG\_256\_56\_zero, 94  
BIGBITS\_256\_56, 71  
BMASK\_256\_56, 71  
DBIG\_256\_56, 72  
DNLEN\_256\_56, 71  
HBITS\_256\_56, 71  
HMASK\_256\_56, 71  
NEXCESS\_256\_56, 71  
NLEN\_256\_56, 72  
big\_384\_56.h, 94  
BIG\_384\_56, 99  
BIG\_384\_56\_add, 99  
BIG\_384\_56\_bit, 100  
BIG\_384\_56\_cmove, 100  
BIG\_384\_56\_comp, 100  
BIG\_384\_56\_copy, 101  
BIG\_384\_56\_cswap, 101  
BIG\_384\_56\_dadd, 101  
BIG\_384\_56\_dcmove, 102  
BIG\_384\_56\_dcomp, 102  
BIG\_384\_56\_dcopy, 102  
BIG\_384\_56\_ddiv, 103  
BIG\_384\_56\_dec, 103  
BIG\_384\_56\_dfromBytesLen, 103  
BIG\_384\_56\_diszilch, 104  
BIG\_384\_56\_div3, 104  
BIG\_384\_56\_dmod, 104  
BIG\_384\_56\_dmod2m, 105  
BIG\_384\_56\_dnbits, 105  
BIG\_384\_56\_dnorm, 105  
BIG\_384\_56\_doutput, 105  
BIG\_384\_56\_drawoutput, 106  
BIG\_384\_56\_dscopy, 106  
BIG\_384\_56\_dshl, 106  
BIG\_384\_56\_dshr, 106  
BIG\_384\_56\_dsub, 107  
BIG\_384\_56\_dsucopy, 107  
BIG\_384\_56\_dzero, 107  
BIG\_384\_56\_fromBytes, 108  
BIG\_384\_56\_fromBytesLen, 108  
BIG\_384\_56\_fshl, 108  
BIG\_384\_56\_fshr, 109  
BIG\_384\_56\_imul, 109  
BIG\_384\_56\_inc, 109  
BIG\_384\_56\_invmod2m, 110  
BIG\_384\_56\_invmodp, 110  
BIG\_384\_56\_isunity, 110  
BIG\_384\_56\_iszilch, 110  
BIG\_384\_56\_jacobi, 112  
BIG\_384\_56\_lastbits, 112  
BIG\_384\_56\_mod, 112  
BIG\_384\_56\_mod2m, 113  
BIG\_384\_56\_moddiv, 113  
BIG\_384\_56\_modmul, 113  
BIG\_384\_56\_modneg, 114

- BIG\_384\_56\_modsqr, 114
- BIG\_384\_56\_monty, 115
- BIG\_384\_56\_mul, 115
- BIG\_384\_56\_nbits, 115
- BIG\_384\_56\_norm, 115
- BIG\_384\_56\_one, 116
- BIG\_384\_56\_or, 116
- BIG\_384\_56\_output, 116
- BIG\_384\_56\_parity, 117
- BIG\_384\_56\_pmul, 117
- BIG\_384\_56\_pxm, 117
- BIG\_384\_56\_random, 118
- BIG\_384\_56\_randomnum, 118
- BIG\_384\_56\_rawoutput, 118
- BIG\_384\_56\_rcopy, 118
- BIG\_384\_56\_sdcopy, 119
- BIG\_384\_56\_sdiv, 119
- BIG\_384\_56\_sducopy, 119
- BIG\_384\_56\_shl, 120
- BIG\_384\_56\_shr, 120
- BIG\_384\_56\_smul, 120
- BIG\_384\_56\_split, 120
- BIG\_384\_56\_sqr, 121
- BIG\_384\_56\_ssn, 121
- BIG\_384\_56\_sub, 122
- BIG\_384\_56\_toBytes, 122
- BIG\_384\_56\_zero, 122
- BIGBITS\_384\_56, 98
- BMASK\_384\_56, 98
- DBIG\_384\_56, 99
- DNLEN\_384\_56, 98
- HBITS\_384\_56, 98
- HMASK\_384\_56, 99
- NEXCESS\_384\_56, 99
- NLEN\_384\_56, 99
- big\_384\_58.h, 122
  - BIG\_384\_58, 127
  - BIG\_384\_58\_add, 127
  - BIG\_384\_58\_bit, 128
  - BIG\_384\_58\_cmove, 128
  - BIG\_384\_58\_comp, 128
  - BIG\_384\_58\_copy, 129
  - BIG\_384\_58\_cswap, 129
  - BIG\_384\_58\_dadd, 129
  - BIG\_384\_58\_dcmove, 130
  - BIG\_384\_58\_dcomp, 130
  - BIG\_384\_58\_dcopy, 130
  - BIG\_384\_58\_ddiv, 131
  - BIG\_384\_58\_dec, 131
  - BIG\_384\_58\_dfromBytesLen, 131
  - BIG\_384\_58\_diszilch, 132
  - BIG\_384\_58\_div3, 132
  - BIG\_384\_58\_dmod, 132
  - BIG\_384\_58\_dmod2m, 133
  - BIG\_384\_58\_dnbits, 133
  - BIG\_384\_58\_dnorm, 133
  - BIG\_384\_58\_doutput, 133
  - BIG\_384\_58\_drawoutput, 134
  - BIG\_384\_58\_dscopy, 134
  - BIG\_384\_58\_dshl, 134
  - BIG\_384\_58\_dshr, 134
  - BIG\_384\_58\_dsub, 135
  - BIG\_384\_58\_dscopy, 135
  - BIG\_384\_58\_dzero, 135
  - BIG\_384\_58\_fromBytes, 136
  - BIG\_384\_58\_fromBytesLen, 136
  - BIG\_384\_58\_fshl, 136
  - BIG\_384\_58\_fshr, 137
  - BIG\_384\_58\_imul, 137
  - BIG\_384\_58\_inc, 137
  - BIG\_384\_58\_invmod2m, 138
  - BIG\_384\_58\_invmodp, 138
  - BIG\_384\_58\_isunity, 138
  - BIG\_384\_58\_iszilch, 138
  - BIG\_384\_58\_jacobi, 140
  - BIG\_384\_58\_lastbits, 140
  - BIG\_384\_58\_mod, 140
  - BIG\_384\_58\_mod2m, 141
  - BIG\_384\_58\_moddiv, 141
  - BIG\_384\_58\_modmul, 141
  - BIG\_384\_58\_modneg, 142
  - BIG\_384\_58\_modsqr, 142
  - BIG\_384\_58\_monty, 143
  - BIG\_384\_58\_mul, 143
  - BIG\_384\_58\_nbits, 143
  - BIG\_384\_58\_norm, 143
  - BIG\_384\_58\_one, 144
  - BIG\_384\_58\_or, 144
  - BIG\_384\_58\_output, 144
  - BIG\_384\_58\_parity, 145
  - BIG\_384\_58\_pmul, 145
  - BIG\_384\_58\_pxm, 145
  - BIG\_384\_58\_random, 146
  - BIG\_384\_58\_randomnum, 146
  - BIG\_384\_58\_rawoutput, 146
  - BIG\_384\_58\_rcopy, 146
  - BIG\_384\_58\_sdcopy, 147
  - BIG\_384\_58\_sdiv, 147
  - BIG\_384\_58\_sducopy, 147
  - BIG\_384\_58\_shl, 148
  - BIG\_384\_58\_shr, 148
  - BIG\_384\_58\_smul, 148
  - BIG\_384\_58\_split, 148
  - BIG\_384\_58\_sqr, 149
  - BIG\_384\_58\_ssn, 149
  - BIG\_384\_58\_sub, 150
  - BIG\_384\_58\_toBytes, 150
  - BIG\_384\_58\_zero, 150
  - BIGBITS\_384\_58, 126
  - BMASK\_384\_58, 126
  - DBIG\_384\_58, 127
  - DNLEN\_384\_58, 126
  - HBITS\_384\_58, 126
  - HMASK\_384\_58, 127
  - NEXCESS\_384\_58, 127
  - NLEN\_384\_58, 127

- big\_448\_58.h, [150](#)
  - BIG\_448\_58, [155](#)
  - BIG\_448\_58\_add, [155](#)
  - BIG\_448\_58\_bit, [156](#)
  - BIG\_448\_58\_cmove, [156](#)
  - BIG\_448\_58\_comp, [156](#)
  - BIG\_448\_58\_copy, [157](#)
  - BIG\_448\_58\_cswap, [157](#)
  - BIG\_448\_58\_dadd, [157](#)
  - BIG\_448\_58\_dcmove, [158](#)
  - BIG\_448\_58\_dcomp, [158](#)
  - BIG\_448\_58\_dcopy, [158](#)
  - BIG\_448\_58\_ddiv, [159](#)
  - BIG\_448\_58\_dec, [159](#)
  - BIG\_448\_58\_dfromBytesLen, [159](#)
  - BIG\_448\_58\_diszilch, [160](#)
  - BIG\_448\_58\_div3, [160](#)
  - BIG\_448\_58\_dmod, [160](#)
  - BIG\_448\_58\_dmod2m, [161](#)
  - BIG\_448\_58\_dnbits, [161](#)
  - BIG\_448\_58\_dnorm, [161](#)
  - BIG\_448\_58\_doutput, [161](#)
  - BIG\_448\_58\_drawoutput, [162](#)
  - BIG\_448\_58\_dscopy, [162](#)
  - BIG\_448\_58\_dshl, [162](#)
  - BIG\_448\_58\_dshr, [162](#)
  - BIG\_448\_58\_dsub, [163](#)
  - BIG\_448\_58\_dscopy, [163](#)
  - BIG\_448\_58\_dzero, [163](#)
  - BIG\_448\_58\_fromBytes, [164](#)
  - BIG\_448\_58\_fromBytesLen, [164](#)
  - BIG\_448\_58\_fshl, [164](#)
  - BIG\_448\_58\_fshr, [165](#)
  - BIG\_448\_58\_imul, [165](#)
  - BIG\_448\_58\_inc, [165](#)
  - BIG\_448\_58\_invmod2m, [166](#)
  - BIG\_448\_58\_invmodp, [166](#)
  - BIG\_448\_58\_isunity, [166](#)
  - BIG\_448\_58\_iszilch, [166](#)
  - BIG\_448\_58\_jacobi, [168](#)
  - BIG\_448\_58\_lastbits, [168](#)
  - BIG\_448\_58\_mod, [168](#)
  - BIG\_448\_58\_mod2m, [169](#)
  - BIG\_448\_58\_moddiv, [169](#)
  - BIG\_448\_58\_modmul, [169](#)
  - BIG\_448\_58\_modneg, [170](#)
  - BIG\_448\_58\_modsqr, [170](#)
  - BIG\_448\_58\_monty, [171](#)
  - BIG\_448\_58\_mul, [171](#)
  - BIG\_448\_58\_nbits, [171](#)
  - BIG\_448\_58\_norm, [171](#)
  - BIG\_448\_58\_one, [172](#)
  - BIG\_448\_58\_or, [172](#)
  - BIG\_448\_58\_output, [172](#)
  - BIG\_448\_58\_parity, [173](#)
  - BIG\_448\_58\_pmul, [173](#)
  - BIG\_448\_58\_pxm, [173](#)
  - BIG\_448\_58\_random, [174](#)
  - BIG\_448\_58\_randomnum, [174](#)
  - BIG\_448\_58\_rawoutput, [174](#)
  - BIG\_448\_58\_rcopy, [174](#)
  - BIG\_448\_58\_sdcopy, [175](#)
  - BIG\_448\_58\_sdiv, [175](#)
  - BIG\_448\_58\_sducop, [175](#)
  - BIG\_448\_58\_shl, [176](#)
  - BIG\_448\_58\_shr, [176](#)
  - BIG\_448\_58\_smul, [176](#)
  - BIG\_448\_58\_split, [176](#)
  - BIG\_448\_58\_sqr, [177](#)
  - BIG\_448\_58\_ssn, [177](#)
  - BIG\_448\_58\_sub, [178](#)
  - BIG\_448\_58\_toBytes, [178](#)
  - BIG\_448\_58\_zero, [178](#)
  - BIGBITS\_448\_58, [154](#)
  - BMASK\_448\_58, [154](#)
  - DBIG\_448\_58, [155](#)
  - DNLEN\_448\_58, [154](#)
  - HBITS\_448\_58, [154](#)
  - HMASK\_448\_58, [155](#)
  - NEXCESS\_448\_58, [155](#)
  - NLEN\_448\_58, [155](#)
- bls\_BLS381.h, [178](#)
  - BFS\_BLS381, [179](#)
  - BGS\_BLS381, [179](#)
  - BLS\_BLS381\_ADD\_G1, [180](#)
  - BLS\_BLS381\_ADD\_G2, [181](#)
  - BLS\_BLS381\_KEY\_PAIR\_GENERATE, [181](#)
  - BLS\_BLS381\_SIGN, [181](#)
  - BLS\_BLS381\_VERIFY, [182](#)
  - BLS\_FAIL, [180](#)
  - BLS\_INVALID\_G1, [180](#)
  - BLS\_INVALID\_G2, [180](#)
  - BLS\_OK, [180](#)
- borrow
  - csprng, [16](#)
- byte
  - arch.h, [37](#)
- c
  - FP12\_BLS381, [22](#)
  - rsa\_private\_key\_2048, [32](#)
  - rsa\_private\_key\_3072, [33](#)
- CHUNK
  - arch.h, [38](#)
- CREATE\_CSPRNG
  - randapi.h, [414](#)
- CURVE\_A\_BLS381
  - ecp2\_BLS381.h, [219](#)
  - ecp\_BLS381.h, [232](#)
- CURVE\_A\_ED25519
  - ecp\_ED25519.h, [247](#)
- CURVE\_A\_GOLDILOCKS
  - ecp\_GOLDILOCKS.h, [262](#)
- CURVE\_A\_NIST256
  - ecp\_NIST256.h, [277](#)
- CURVE\_B\_BLS381
  - ecp2\_BLS381.h, [219](#)



ecp\_BLS381.h, [232](#)  
 CURVE\_B\_ED25519  
 ecp\_ED25519.h, [247](#)  
 CURVE\_B\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [262](#)  
 CURVE\_B\_I\_BLS381  
 ecp2\_BLS381.h, [219](#)  
 ecp\_BLS381.h, [232](#)  
 CURVE\_B\_I\_ED25519  
 ecp\_ED25519.h, [247](#)  
 CURVE\_B\_I\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [262](#)  
 CURVE\_B\_I\_NIST256  
 ecp\_NIST256.h, [277](#)  
 CURVE\_B\_NIST256  
 ecp\_NIST256.h, [277](#)  
 CURVE\_BB\_BLS381  
 ecp\_BLS381.h, [232](#)  
 pair\_BLS381.h, [409](#)  
 CURVE\_BB\_ED25519  
 ecp\_ED25519.h, [247](#)  
 CURVE\_BB\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [262](#)  
 CURVE\_BB\_NIST256  
 ecp\_NIST256.h, [278](#)  
 CURVE\_Bnx\_BLS381  
 ecp2\_BLS381.h, [220](#)  
 ecp\_BLS381.h, [232](#)  
 pair\_BLS381.h, [409](#)  
 CURVE\_Bnx\_ED25519  
 ecp\_ED25519.h, [247](#)  
 CURVE\_Bnx\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [263](#)  
 CURVE\_Bnx\_NIST256  
 ecp\_NIST256.h, [278](#)  
 CURVE\_Cof\_BLS381  
 ecp2\_BLS381.h, [220](#)  
 ecp\_BLS381.h, [232](#)  
 CURVE\_Cof\_ED25519  
 ecp\_ED25519.h, [247](#)  
 CURVE\_Cof\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [263](#)  
 CURVE\_Cof\_I\_BLS381  
 ecp\_BLS381.h, [232](#)  
 CURVE\_Cof\_I\_ED25519  
 ecp\_ED25519.h, [248](#)  
 CURVE\_Cof\_I\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [263](#)  
 CURVE\_Cof\_I\_NIST256  
 ecp\_NIST256.h, [278](#)  
 CURVE\_Cof\_NIST256  
 ecp\_NIST256.h, [278](#)  
 CURVE\_Cru\_BLS381  
 ecp\_BLS381.h, [233](#)  
 pair\_BLS381.h, [409](#)  
 CURVE\_Cru\_ED25519  
 ecp\_ED25519.h, [248](#)  
 CURVE\_Cru\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [263](#)  
 CURVE\_Cru\_NIST256  
 ecp\_NIST256.h, [278](#)  
 CURVE\_Gx\_BLS381  
 ecp2\_BLS381.h, [220](#)  
 ecp\_BLS381.h, [233](#)  
 CURVE\_Gx\_ED25519  
 ecp\_ED25519.h, [248](#)  
 CURVE\_Gx\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [263](#)  
 CURVE\_Gx\_NIST256  
 ecp\_NIST256.h, [278](#)  
 CURVE\_Gy\_BLS381  
 ecp2\_BLS381.h, [220](#)  
 ecp\_BLS381.h, [233](#)  
 CURVE\_Gy\_ED25519  
 ecp\_ED25519.h, [248](#)  
 CURVE\_Gy\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [263](#)  
 CURVE\_Gy\_NIST256  
 ecp\_NIST256.h, [278](#)  
 CURVE\_Order\_BLS381  
 ecp2\_BLS381.h, [220](#)  
 ecp\_BLS381.h, [233](#)  
 CURVE\_Order\_ED25519  
 ecp\_ED25519.h, [248](#)  
 CURVE\_Order\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [263](#)  
 CURVE\_Order\_NIST256  
 ecp\_NIST256.h, [278](#)  
 CURVE\_Pxa\_BLS381  
 ecp2\_BLS381.h, [220](#)  
 ecp\_BLS381.h, [233](#)  
 CURVE\_Pxa\_ED25519  
 ecp\_ED25519.h, [248](#)  
 CURVE\_Pxa\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [263](#)  
 CURVE\_Pxa\_NIST256  
 ecp\_NIST256.h, [279](#)  
 CURVE\_Pxaa\_BLS381  
 ecp\_BLS381.h, [233](#)  
 CURVE\_Pxaa\_ED25519  
 ecp\_ED25519.h, [248](#)  
 CURVE\_Pxaa\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [264](#)  
 CURVE\_Pxaa\_NIST256  
 ecp\_NIST256.h, [279](#)  
 CURVE\_Pxaaa\_BLS381  
 ecp\_BLS381.h, [233](#)  
 CURVE\_Pxaaa\_ED25519  
 ecp\_ED25519.h, [248](#)  
 CURVE\_Pxaaa\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [264](#)  
 CURVE\_Pxaaa\_NIST256  
 ecp\_NIST256.h, [279](#)  
 CURVE\_Pxaab\_BLS381  
 ecp\_BLS381.h, [233](#)  
 CURVE\_Pxaab\_ED25519



ecp\_ED25519.h, [249](#)  
CURVE\_Pxaab\_GOLDILOCKS  
ecp\_GOLDILOCKS.h, [264](#)  
CURVE\_Pxaab\_NIST256  
ecp\_NIST256.h, [279](#)  
CURVE\_Pxab\_BLS381  
ecp\_BLS381.h, [234](#)  
CURVE\_Pxab\_ED25519  
ecp\_ED25519.h, [249](#)  
CURVE\_Pxab\_GOLDILOCKS  
ecp\_GOLDILOCKS.h, [264](#)  
CURVE\_Pxab\_NIST256  
ecp\_NIST256.h, [279](#)  
CURVE\_Pxaba\_BLS381  
ecp\_BLS381.h, [234](#)  
CURVE\_Pxaba\_ED25519  
ecp\_ED25519.h, [249](#)  
CURVE\_Pxaba\_GOLDILOCKS  
ecp\_GOLDILOCKS.h, [264](#)  
CURVE\_Pxaba\_NIST256  
ecp\_NIST256.h, [279](#)  
CURVE\_Pxabb\_BLS381  
ecp\_BLS381.h, [234](#)  
CURVE\_Pxabb\_ED25519  
ecp\_ED25519.h, [249](#)  
CURVE\_Pxabb\_GOLDILOCKS  
ecp\_GOLDILOCKS.h, [264](#)  
CURVE\_Pxabb\_NIST256  
ecp\_NIST256.h, [279](#)  
CURVE\_Pxb\_BLS381  
ecp2\_BLS381.h, [220](#)  
ecp\_BLS381.h, [234](#)  
CURVE\_Pxb\_ED25519  
ecp\_ED25519.h, [249](#)  
CURVE\_Pxb\_GOLDILOCKS  
ecp\_GOLDILOCKS.h, [264](#)  
CURVE\_Pxb\_NIST256  
ecp\_NIST256.h, [279](#)  
CURVE\_Pxba\_BLS381  
ecp\_BLS381.h, [234](#)  
CURVE\_Pxba\_ED25519  
ecp\_ED25519.h, [249](#)  
CURVE\_Pxba\_GOLDILOCKS  
ecp\_GOLDILOCKS.h, [264](#)  
CURVE\_Pxba\_NIST256  
ecp\_NIST256.h, [280](#)  
CURVE\_Pxbaa\_BLS381  
ecp\_BLS381.h, [234](#)  
CURVE\_Pxbaa\_ED25519  
ecp\_ED25519.h, [249](#)  
CURVE\_Pxbaa\_GOLDILOCKS  
ecp\_GOLDILOCKS.h, [265](#)  
CURVE\_Pxbaa\_NIST256  
ecp\_NIST256.h, [280](#)  
CURVE\_Pxbab\_BLS381  
ecp\_BLS381.h, [234](#)  
CURVE\_Pxbab\_ED25519  
ecp\_ED25519.h, [249](#)  
CURVE\_Pxbab\_GOLDILOCKS  
ecp\_GOLDILOCKS.h, [265](#)  
CURVE\_Pxbab\_NIST256  
ecp\_NIST256.h, [280](#)  
CURVE\_Pxbb\_BLS381  
ecp\_BLS381.h, [234](#)  
CURVE\_Pxbb\_ED25519  
ecp\_ED25519.h, [250](#)  
CURVE\_Pxbb\_GOLDILOCKS  
ecp\_GOLDILOCKS.h, [265](#)  
CURVE\_Pxbb\_NIST256  
ecp\_NIST256.h, [280](#)  
CURVE\_Pxbba\_BLS381  
ecp\_BLS381.h, [235](#)  
CURVE\_Pxbba\_ED25519  
ecp\_ED25519.h, [250](#)  
CURVE\_Pxbba\_GOLDILOCKS  
ecp\_GOLDILOCKS.h, [265](#)  
CURVE\_Pxbba\_NIST256  
ecp\_NIST256.h, [280](#)  
CURVE\_Pxbbb\_BLS381  
ecp\_BLS381.h, [235](#)  
CURVE\_Pxbbb\_ED25519  
ecp\_ED25519.h, [250](#)  
CURVE\_Pxbbb\_GOLDILOCKS  
ecp\_GOLDILOCKS.h, [265](#)  
CURVE\_Pxbbb\_NIST256  
ecp\_NIST256.h, [280](#)  
CURVE\_Pya\_BLS381  
ecp2\_BLS381.h, [220](#)  
ecp\_BLS381.h, [235](#)  
CURVE\_Pya\_ED25519  
ecp\_ED25519.h, [250](#)  
CURVE\_Pya\_GOLDILOCKS  
ecp\_GOLDILOCKS.h, [265](#)  
CURVE\_Pya\_NIST256  
ecp\_NIST256.h, [280](#)  
CURVE\_Pyaa\_BLS381  
ecp\_BLS381.h, [235](#)  
CURVE\_Pyaa\_ED25519  
ecp\_ED25519.h, [250](#)  
CURVE\_Pyaa\_GOLDILOCKS  
ecp\_GOLDILOCKS.h, [265](#)  
CURVE\_Pyaa\_NIST256  
ecp\_NIST256.h, [280](#)  
CURVE\_Pyaaa\_BLS381  
ecp\_BLS381.h, [235](#)  
CURVE\_Pyaaa\_ED25519  
ecp\_ED25519.h, [250](#)  
CURVE\_Pyaaa\_GOLDILOCKS  
ecp\_GOLDILOCKS.h, [265](#)  
CURVE\_Pyaaa\_NIST256  
ecp\_NIST256.h, [281](#)  
CURVE\_Pyaab\_BLS381  
ecp\_BLS381.h, [235](#)  
CURVE\_Pyaab\_ED25519  
ecp\_ED25519.h, [250](#)  
CURVE\_Pyaab\_GOLDILOCKS

ecp\_GOLDILOCKS.h, [266](#)  
 CURVE\_Pyaab\_NIST256  
 ecp\_NIST256.h, [281](#)  
 CURVE\_Pyab\_BLS381  
 ecp\_BLS381.h, [235](#)  
 CURVE\_Pyab\_ED25519  
 ecp\_ED25519.h, [250](#)  
 CURVE\_Pyab\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [266](#)  
 CURVE\_Pyab\_NIST256  
 ecp\_NIST256.h, [281](#)  
 CURVE\_Pyaba\_BLS381  
 ecp\_BLS381.h, [235](#)  
 CURVE\_Pyaba\_ED25519  
 ecp\_ED25519.h, [251](#)  
 CURVE\_Pyaba\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [266](#)  
 CURVE\_Pyaba\_NIST256  
 ecp\_NIST256.h, [281](#)  
 CURVE\_Pyabb\_BLS381  
 ecp\_BLS381.h, [236](#)  
 CURVE\_Pyabb\_ED25519  
 ecp\_ED25519.h, [251](#)  
 CURVE\_Pyabb\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [266](#)  
 CURVE\_Pyabb\_NIST256  
 ecp\_NIST256.h, [281](#)  
 CURVE\_Pyb\_BLS381  
 ecp2\_BLS381.h, [221](#)  
 ecp\_BLS381.h, [236](#)  
 CURVE\_Pyb\_ED25519  
 ecp\_ED25519.h, [251](#)  
 CURVE\_Pyb\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [266](#)  
 CURVE\_Pyb\_NIST256  
 ecp\_NIST256.h, [281](#)  
 CURVE\_Pyba\_BLS381  
 ecp\_BLS381.h, [236](#)  
 CURVE\_Pyba\_ED25519  
 ecp\_ED25519.h, [251](#)  
 CURVE\_Pyba\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [266](#)  
 CURVE\_Pyba\_NIST256  
 ecp\_NIST256.h, [281](#)  
 CURVE\_Pybaa\_BLS381  
 ecp\_BLS381.h, [236](#)  
 CURVE\_Pybaa\_ED25519  
 ecp\_ED25519.h, [251](#)  
 CURVE\_Pybaa\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [266](#)  
 CURVE\_Pybaa\_NIST256  
 ecp\_NIST256.h, [281](#)  
 CURVE\_Pybab\_BLS381  
 ecp\_BLS381.h, [236](#)  
 CURVE\_Pybab\_ED25519  
 ecp\_ED25519.h, [251](#)  
 CURVE\_Pybab\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [266](#)  
 CURVE\_Pybab\_NIST256  
 ecp\_NIST256.h, [282](#)  
 CURVE\_Pybb\_BLS381  
 ecp\_BLS381.h, [236](#)  
 CURVE\_Pybb\_ED25519  
 ecp\_ED25519.h, [251](#)  
 CURVE\_Pybb\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [267](#)  
 CURVE\_Pybb\_NIST256  
 ecp\_NIST256.h, [282](#)  
 CURVE\_Pybba\_BLS381  
 ecp\_BLS381.h, [236](#)  
 CURVE\_Pybba\_ED25519  
 ecp\_ED25519.h, [251](#)  
 CURVE\_Pybba\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [267](#)  
 CURVE\_Pybba\_NIST256  
 ecp\_NIST256.h, [282](#)  
 CURVE\_Pybbb\_BLS381  
 ecp\_BLS381.h, [236](#)  
 CURVE\_Pybbb\_ED25519  
 ecp\_ED25519.h, [252](#)  
 CURVE\_Pybbb\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [267](#)  
 CURVE\_Pybbb\_NIST256  
 ecp\_NIST256.h, [282](#)  
 CURVE\_SB\_BLS381  
 ecp\_BLS381.h, [237](#)  
 pair\_BLS381.h, [409](#)  
 CURVE\_SB\_ED25519  
 ecp\_ED25519.h, [252](#)  
 CURVE\_SB\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [267](#)  
 CURVE\_SB\_NIST256  
 ecp\_NIST256.h, [282](#)  
 CURVE\_W\_BLS381  
 ecp\_BLS381.h, [237](#)  
 pair\_BLS381.h, [409](#)  
 CURVE\_W\_ED25519  
 ecp\_ED25519.h, [252](#)  
 CURVE\_W\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [267](#)  
 CURVE\_W\_NIST256  
 ecp\_NIST256.h, [282](#)  
 CURVE\_WB\_BLS381  
 ecp\_BLS381.h, [237](#)  
 pair\_BLS381.h, [409](#)  
 CURVE\_WB\_ED25519  
 ecp\_ED25519.h, [252](#)  
 CURVE\_WB\_GOLDILOCKS  
 ecp\_GOLDILOCKS.h, [267](#)  
 CURVE\_WB\_NIST256  
 ecp\_NIST256.h, [282](#)  
 chunk  
 arch.h, [38](#)  
 config\_big\_1024\_58.h, [182](#)  
 BASEBITS\_1024\_58, [183](#)  
 MODBYTES\_1024\_58, [183](#)

- config\_big\_256\_56.h, [183](#)
  - BASEBITS\_256\_56, [183](#)
  - MODBYTES\_256\_56, [183](#)
- config\_big\_384\_56.h, [184](#)
  - BASEBITS\_384\_56, [184](#)
  - MODBYTES\_384\_56, [184](#)
- config\_big\_384\_58.h, [184](#)
  - BASEBITS\_384\_58, [185](#)
  - MODBYTES\_384\_58, [185](#)
- config\_big\_448\_58.h, [185](#)
  - BASEBITS\_448\_58, [186](#)
  - MODBYTES\_448\_58, [186](#)
- config\_ff\_2048.h, [186](#)
  - FFLEN\_2048, [186](#)
- config\_ff\_3072.h, [187](#)
  - FFLEN\_3072, [187](#)
- csprng, [16](#)
  - borrow, [16](#)
  - ira, [17](#)
  - pool, [17](#)
  - pool\_ptr, [17](#)
  - rndptr, [17](#)
- curve
  - pktype, [31](#)
- DBIG\_1024\_58
  - big\_1024\_58.h, [44](#)
- DBIG\_256\_56
  - big\_256\_56.h, [72](#)
- DBIG\_384\_56
  - big\_384\_56.h, [99](#)
- DBIG\_384\_58
  - big\_384\_58.h, [127](#)
- DBIG\_448\_58
  - big\_448\_58.h, [155](#)
- DNLEN\_1024\_58
  - big\_1024\_58.h, [43](#)
- DNLEN\_256\_56
  - big\_256\_56.h, [71](#)
- DNLEN\_384\_56
  - big\_384\_56.h, [98](#)
- DNLEN\_384\_58
  - big\_384\_58.h, [126](#)
- DNLEN\_448\_58
  - big\_448\_58.h, [154](#)
- dp
  - rsa\_private\_key\_2048, [32](#)
  - rsa\_private\_key\_3072, [33](#)
- dq
  - rsa\_private\_key\_2048, [32](#)
  - rsa\_private\_key\_3072, [33](#)
- e
  - rsa\_public\_key\_2048, [34](#)
  - rsa\_public\_key\_3072, [35](#)
- ECDH\_ERROR
  - ecdh\_BLS381.h, [188](#)
  - ecdh\_ED25519.h, [193](#)
  - ecdh\_GOLDILOCKS.h, [198](#)
- ecdh\_NIST256.h, [203](#)
- ECDH\_INVALID\_PUBLIC\_KEY
  - ecdh\_BLS381.h, [188](#)
  - ecdh\_ED25519.h, [193](#)
  - ecdh\_GOLDILOCKS.h, [198](#)
  - ecdh\_NIST256.h, [203](#)
- ECDH\_INVALID
  - ecdh\_BLS381.h, [188](#)
  - ecdh\_ED25519.h, [193](#)
  - ecdh\_GOLDILOCKS.h, [198](#)
  - ecdh\_NIST256.h, [203](#)
- ECDH\_OK
  - ecdh\_BLS381.h, [188](#)
  - ecdh\_ED25519.h, [194](#)
  - ecdh\_GOLDILOCKS.h, [199](#)
  - ecdh\_NIST256.h, [204](#)
- ECP2\_BLS381, [17](#)
  - x, [18](#)
  - y, [18](#)
  - z, [18](#)
- ECP2\_BLS381\_add
  - ecp2\_BLS381.h, [213](#)
- ECP2\_BLS381\_affine
  - ecp2\_BLS381.h, [213](#)
- ECP2\_BLS381\_copy
  - ecp2\_BLS381.h, [213](#)
- ECP2\_BLS381\_dbl
  - ecp2\_BLS381.h, [213](#)
- ECP2\_BLS381\_equals
  - ecp2\_BLS381.h, [214](#)
- ECP2\_BLS381\_frob
  - ecp2\_BLS381.h, [214](#)
- ECP2\_BLS381\_fromOctet
  - ecp2\_BLS381.h, [214](#)
- ECP2\_BLS381\_generator
  - ecp2\_BLS381.h, [215](#)
- ECP2\_BLS381\_get
  - ecp2\_BLS381.h, [215](#)
- ECP2\_BLS381\_inf
  - ecp2\_BLS381.h, [215](#)
- ECP2\_BLS381\_isinf
  - ecp2\_BLS381.h, [216](#)
- ECP2\_BLS381\_mapit
  - ecp2\_BLS381.h, [216](#)
- ECP2\_BLS381\_mul
  - ecp2\_BLS381.h, [216](#)
- ECP2\_BLS381\_mul4
  - ecp2\_BLS381.h, [217](#)
- ECP2\_BLS381\_neg
  - ecp2\_BLS381.h, [217](#)
- ECP2\_BLS381\_output
  - ecp2\_BLS381.h, [217](#)
- ECP2\_BLS381\_outputxyz
  - ecp2\_BLS381.h, [217](#)
- ECP2\_BLS381\_rhs
  - ecp2\_BLS381.h, [218](#)
- ECP2\_BLS381\_set
  - ecp2\_BLS381.h, [218](#)

ECP2\_BLS381\_setx  
     ecp2\_BLS381.h, [218](#)  
 ECP2\_BLS381\_sub  
     ecp2\_BLS381.h, [219](#)  
 ECP2\_BLS381\_toOctet  
     ecp2\_BLS381.h, [219](#)  
 ECP\_BLS381, [18](#)  
     x, [18](#)  
     y, [19](#)  
     z, [19](#)  
 ECP\_BLS381\_ECIES\_DECRYPT  
     ecdh\_BLS381.h, [189](#)  
 ECP\_BLS381\_ECIES\_ENCRYPT  
     ecdh\_BLS381.h, [190](#)  
 ECP\_BLS381\_KEY\_PAIR\_GENERATE  
     ecdh\_BLS381.h, [190](#)  
 ECP\_BLS381\_PUBLIC\_KEY\_VALIDATE  
     ecdh\_BLS381.h, [191](#)  
 ECP\_BLS381\_SP\_DSA  
     ecdh\_BLS381.h, [191](#)  
 ECP\_BLS381\_SVDP\_DH  
     ecdh\_BLS381.h, [191](#)  
 ECP\_BLS381\_VP\_DSA  
     ecdh\_BLS381.h, [192](#)  
 ECP\_BLS381\_add  
     ecp\_BLS381.h, [224](#)  
 ECP\_BLS381\_affine  
     ecp\_BLS381.h, [224](#)  
 ECP\_BLS381\_cfp  
     ecp\_BLS381.h, [224](#)  
 ECP\_BLS381\_copy  
     ecp\_BLS381.h, [224](#)  
 ECP\_BLS381\_dbl  
     ecp\_BLS381.h, [225](#)  
 ECP\_BLS381\_equals  
     ecp\_BLS381.h, [225](#)  
 ECP\_BLS381\_fromOctet  
     ecp\_BLS381.h, [225](#)  
 ECP\_BLS381\_generator  
     ecp\_BLS381.h, [226](#)  
 ECP\_BLS381\_get  
     ecp\_BLS381.h, [226](#)  
 ECP\_BLS381\_inf  
     ecp\_BLS381.h, [226](#)  
 ECP\_BLS381\_isinf  
     ecp\_BLS381.h, [226](#)  
 ECP\_BLS381\_mapit  
     ecp\_BLS381.h, [227](#)  
 ECP\_BLS381\_mul  
     ecp\_BLS381.h, [227](#)  
 ECP\_BLS381\_mul2  
     ecp\_BLS381.h, [227](#)  
 ECP\_BLS381\_neg  
     ecp\_BLS381.h, [228](#)  
 ECP\_BLS381\_output  
     ecp\_BLS381.h, [228](#)  
 ECP\_BLS381\_outputxyz  
     ecp\_BLS381.h, [228](#)  
 ECP\_BLS381\_pinmul  
     ecp\_BLS381.h, [228](#)  
 ECP\_BLS381\_rawoutput  
     ecp\_BLS381.h, [229](#)  
 ECP\_BLS381\_rhs  
     ecp\_BLS381.h, [229](#)  
 ECP\_BLS381\_set  
     ecp\_BLS381.h, [229](#)  
 ECP\_BLS381\_setx  
     ecp\_BLS381.h, [230](#)  
 ECP\_BLS381\_sub  
     ecp\_BLS381.h, [230](#)  
 ECP\_BLS381\_toOctet  
     ecp\_BLS381.h, [230](#)  
 ECP\_ED25519, [19](#)  
     x, [19](#)  
     y, [19](#)  
     z, [20](#)  
 ECP\_ED25519\_ECIES\_DECRYPT  
     ecdh\_ED25519.h, [194](#)  
 ECP\_ED25519\_ECIES\_ENCRYPT  
     ecdh\_ED25519.h, [195](#)  
 ECP\_ED25519\_KEY\_PAIR\_GENERATE  
     ecdh\_ED25519.h, [195](#)  
 ECP\_ED25519\_PUBLIC\_KEY\_VALIDATE  
     ecdh\_ED25519.h, [196](#)  
 ECP\_ED25519\_SP\_DSA  
     ecdh\_ED25519.h, [196](#)  
 ECP\_ED25519\_SVDP\_DH  
     ecdh\_ED25519.h, [196](#)  
 ECP\_ED25519\_VP\_DSA  
     ecdh\_ED25519.h, [197](#)  
 ECP\_ED25519\_add  
     ecp\_ED25519.h, [240](#)  
 ECP\_ED25519\_affine  
     ecp\_ED25519.h, [240](#)  
 ECP\_ED25519\_cfp  
     ecp\_ED25519.h, [240](#)  
 ECP\_ED25519\_copy  
     ecp\_ED25519.h, [240](#)  
 ECP\_ED25519\_dbl  
     ecp\_ED25519.h, [241](#)  
 ECP\_ED25519\_equals  
     ecp\_ED25519.h, [241](#)  
 ECP\_ED25519\_fromOctet  
     ecp\_ED25519.h, [241](#)  
 ECP\_ED25519\_generator  
     ecp\_ED25519.h, [242](#)  
 ECP\_ED25519\_get  
     ecp\_ED25519.h, [242](#)  
 ECP\_ED25519\_inf  
     ecp\_ED25519.h, [242](#)  
 ECP\_ED25519\_isinf  
     ecp\_ED25519.h, [243](#)  
 ECP\_ED25519\_mapit  
     ecp\_ED25519.h, [243](#)  
 ECP\_ED25519\_mul  
     ecp\_ED25519.h, [243](#)

- ECP\_ED25519\_mul2
  - ecp\_ED25519.h, [243](#)
- ECP\_ED25519\_neg
  - ecp\_ED25519.h, [244](#)
- ECP\_ED25519\_output
  - ecp\_ED25519.h, [244](#)
- ECP\_ED25519\_outputxyz
  - ecp\_ED25519.h, [244](#)
- ECP\_ED25519\_pinmul
  - ecp\_ED25519.h, [245](#)
- ECP\_ED25519\_rawoutput
  - ecp\_ED25519.h, [245](#)
- ECP\_ED25519\_rhs
  - ecp\_ED25519.h, [245](#)
- ECP\_ED25519\_set
  - ecp\_ED25519.h, [245](#)
- ECP\_ED25519\_setx
  - ecp\_ED25519.h, [246](#)
- ECP\_ED25519\_sub
  - ecp\_ED25519.h, [246](#)
- ECP\_ED25519\_toOctet
  - ecp\_ED25519.h, [246](#)
- ECP\_GOLDILOCKS\_ECIES\_DECRYPT
  - ecdh\_GOLDILOCKS.h, [199](#)
- ECP\_GOLDILOCKS\_ECIES\_ENCRYPT
  - ecdh\_GOLDILOCKS.h, [200](#)
- ECP\_GOLDILOCKS\_KEY\_PAIR\_GENERATE
  - ecdh\_GOLDILOCKS.h, [200](#)
- ECP\_GOLDILOCKS\_PUBLIC\_KEY\_VALIDATE
  - ecdh\_GOLDILOCKS.h, [201](#)
- ECP\_GOLDILOCKS\_SP\_DSA
  - ecdh\_GOLDILOCKS.h, [201](#)
- ECP\_GOLDILOCKS\_SVDP\_DH
  - ecdh\_GOLDILOCKS.h, [201](#)
- ECP\_GOLDILOCKS\_VP\_DSA
  - ecdh\_GOLDILOCKS.h, [202](#)
- ECP\_GOLDILOCKS\_add
  - ecp\_GOLDILOCKS.h, [255](#)
- ECP\_GOLDILOCKS\_affine
  - ecp\_GOLDILOCKS.h, [255](#)
- ECP\_GOLDILOCKS\_cfp
  - ecp\_GOLDILOCKS.h, [255](#)
- ECP\_GOLDILOCKS\_copy
  - ecp\_GOLDILOCKS.h, [256](#)
- ECP\_GOLDILOCKS\_dbl
  - ecp\_GOLDILOCKS.h, [256](#)
- ECP\_GOLDILOCKS\_equals
  - ecp\_GOLDILOCKS.h, [256](#)
- ECP\_GOLDILOCKS\_fromOctet
  - ecp\_GOLDILOCKS.h, [257](#)
- ECP\_GOLDILOCKS\_generator
  - ecp\_GOLDILOCKS.h, [257](#)
- ECP\_GOLDILOCKS\_get
  - ecp\_GOLDILOCKS.h, [257](#)
- ECP\_GOLDILOCKS\_inf
  - ecp\_GOLDILOCKS.h, [258](#)
- ECP\_GOLDILOCKS\_isinf
  - ecp\_GOLDILOCKS.h, [258](#)
- ECP\_GOLDILOCKS\_mapit
  - ecp\_GOLDILOCKS.h, [258](#)
- ECP\_GOLDILOCKS\_mul
  - ecp\_GOLDILOCKS.h, [258](#)
- ECP\_GOLDILOCKS\_mul2
  - ecp\_GOLDILOCKS.h, [259](#)
- ECP\_GOLDILOCKS\_neg
  - ecp\_GOLDILOCKS.h, [259](#)
- ECP\_GOLDILOCKS\_output
  - ecp\_GOLDILOCKS.h, [259](#)
- ECP\_GOLDILOCKS\_outputxyz
  - ecp\_GOLDILOCKS.h, [260](#)
- ECP\_GOLDILOCKS\_pinmul
  - ecp\_GOLDILOCKS.h, [260](#)
- ECP\_GOLDILOCKS\_rawoutput
  - ecp\_GOLDILOCKS.h, [260](#)
- ECP\_GOLDILOCKS\_rhs
  - ecp\_GOLDILOCKS.h, [260](#)
- ECP\_GOLDILOCKS\_set
  - ecp\_GOLDILOCKS.h, [261](#)
- ECP\_GOLDILOCKS\_setx
  - ecp\_GOLDILOCKS.h, [261](#)
- ECP\_GOLDILOCKS\_sub
  - ecp\_GOLDILOCKS.h, [261](#)
- ECP\_GOLDILOCKS\_toOctet
  - ecp\_GOLDILOCKS.h, [262](#)
- ECP\_GOLDILOCKS, [20](#)
  - x, [20](#)
  - y, [20](#)
  - z, [20](#)
- ECP\_NIST256, [21](#)
  - x, [21](#)
  - y, [21](#)
  - z, [21](#)
- ECP\_NIST256\_ECIES\_DECRYPT
  - ecdh\_NIST256.h, [204](#)
- ECP\_NIST256\_ECIES\_ENCRYPT
  - ecdh\_NIST256.h, [205](#)
- ECP\_NIST256\_KEY\_PAIR\_GENERATE
  - ecdh\_NIST256.h, [205](#)
- ECP\_NIST256\_PUBLIC\_KEY\_VALIDATE
  - ecdh\_NIST256.h, [206](#)
- ECP\_NIST256\_SP\_DSA
  - ecdh\_NIST256.h, [206](#)
- ECP\_NIST256\_SVDP\_DH
  - ecdh\_NIST256.h, [206](#)
- ECP\_NIST256\_VP\_DSA
  - ecdh\_NIST256.h, [207](#)
- ECP\_NIST256\_add
  - ecp\_NIST256.h, [270](#)
- ECP\_NIST256\_affine
  - ecp\_NIST256.h, [270](#)
- ECP\_NIST256\_cfp
  - ecp\_NIST256.h, [271](#)
- ECP\_NIST256\_copy
  - ecp\_NIST256.h, [271](#)
- ECP\_NIST256\_dbl
  - ecp\_NIST256.h, [271](#)

ECP\_NIST256\_equals  
     ecp\_NIST256.h, [271](#)  
 ECP\_NIST256\_fromOctet  
     ecp\_NIST256.h, [272](#)  
 ECP\_NIST256\_generator  
     ecp\_NIST256.h, [272](#)  
 ECP\_NIST256\_get  
     ecp\_NIST256.h, [272](#)  
 ECP\_NIST256\_inf  
     ecp\_NIST256.h, [273](#)  
 ECP\_NIST256\_isinf  
     ecp\_NIST256.h, [273](#)  
 ECP\_NIST256\_mapit  
     ecp\_NIST256.h, [273](#)  
 ECP\_NIST256\_mul  
     ecp\_NIST256.h, [274](#)  
 ECP\_NIST256\_mul2  
     ecp\_NIST256.h, [274](#)  
 ECP\_NIST256\_neg  
     ecp\_NIST256.h, [274](#)  
 ECP\_NIST256\_output  
     ecp\_NIST256.h, [274](#)  
 ECP\_NIST256\_outputxyz  
     ecp\_NIST256.h, [275](#)  
 ECP\_NIST256\_pinmul  
     ecp\_NIST256.h, [275](#)  
 ECP\_NIST256\_rawoutput  
     ecp\_NIST256.h, [275](#)  
 ECP\_NIST256\_rhs  
     ecp\_NIST256.h, [275](#)  
 ECP\_NIST256\_set  
     ecp\_NIST256.h, [276](#)  
 ECP\_NIST256\_setx  
     ecp\_NIST256.h, [276](#)  
 ECP\_NIST256\_sub  
     ecp\_NIST256.h, [277](#)  
 ECP\_NIST256\_toOctet  
     ecp\_NIST256.h, [277](#)  
 EFS\_BLS381  
     ecdh\_BLS381.h, [189](#)  
 EFS\_ED25519  
     ecdh\_ED25519.h, [194](#)  
 EFS\_GOLDILOCKS  
     ecdh\_GOLDILOCKS.h, [199](#)  
 EFS\_NIST256  
     ecdh\_NIST256.h, [204](#)  
 EGS\_BLS381  
     ecdh\_BLS381.h, [189](#)  
 EGS\_ED25519  
     ecdh\_ED25519.h, [194](#)  
 EGS\_GOLDILOCKS  
     ecdh\_GOLDILOCKS.h, [199](#)  
 EGS\_NIST256  
     ecdh\_NIST256.h, [204](#)  
 ecdh\_BLS381.h, [187](#)  
     ECDH\_ERROR, [188](#)  
     ECDH\_INVALID\_PUBLIC\_KEY, [188](#)  
     ECDH\_INVALID, [188](#)  
     ECDH\_OK, [188](#)  
     ECP\_BLS381\_ECIES\_DECRYPT, [189](#)  
     ECP\_BLS381\_ECIES\_ENCRYPT, [190](#)  
     ECP\_BLS381\_KEY\_PAIR\_GENERATE, [190](#)  
     ECP\_BLS381\_PUBLIC\_KEY\_VALIDATE, [191](#)  
     ECP\_BLS381\_SP\_DSA, [191](#)  
     ECP\_BLS381\_SVDP\_DH, [191](#)  
     ECP\_BLS381\_VP\_DSA, [192](#)  
     EFS\_BLS381, [189](#)  
     EGS\_BLS381, [189](#)  
 ecdh\_ED25519.h, [192](#)  
     ECDH\_ERROR, [193](#)  
     ECDH\_INVALID\_PUBLIC\_KEY, [193](#)  
     ECDH\_INVALID, [193](#)  
     ECDH\_OK, [194](#)  
     ECP\_ED25519\_ECIES\_DECRYPT, [194](#)  
     ECP\_ED25519\_ECIES\_ENCRYPT, [195](#)  
     ECP\_ED25519\_KEY\_PAIR\_GENERATE, [195](#)  
     ECP\_ED25519\_PUBLIC\_KEY\_VALIDATE, [196](#)  
     ECP\_ED25519\_SP\_DSA, [196](#)  
     ECP\_ED25519\_SVDP\_DH, [196](#)  
     ECP\_ED25519\_VP\_DSA, [197](#)  
     EFS\_ED25519, [194](#)  
     EGS\_ED25519, [194](#)  
 ecdh\_GOLDILOCKS.h, [197](#)  
     ECDH\_ERROR, [198](#)  
     ECDH\_INVALID\_PUBLIC\_KEY, [198](#)  
     ECDH\_INVALID, [198](#)  
     ECDH\_OK, [199](#)  
     ECP\_GOLDILOCKS\_ECIES\_DECRYPT, [199](#)  
     ECP\_GOLDILOCKS\_ECIES\_ENCRYPT, [200](#)  
     ECP\_GOLDILOCKS\_KEY\_PAIR\_GENERATE, [200](#)  
     ECP\_GOLDILOCKS\_PUBLIC\_KEY\_VALIDATE, [201](#)  
     ECP\_GOLDILOCKS\_SP\_DSA, [201](#)  
     ECP\_GOLDILOCKS\_SVDP\_DH, [201](#)  
     ECP\_GOLDILOCKS\_VP\_DSA, [202](#)  
     EFS\_GOLDILOCKS, [199](#)  
     EGS\_GOLDILOCKS, [199](#)  
 ecdh\_NIST256.h, [202](#)  
     ECDH\_ERROR, [203](#)  
     ECDH\_INVALID\_PUBLIC\_KEY, [203](#)  
     ECDH\_INVALID, [203](#)  
     ECDH\_OK, [204](#)  
     ECP\_NIST256\_ECIES\_DECRYPT, [204](#)  
     ECP\_NIST256\_ECIES\_ENCRYPT, [205](#)  
     ECP\_NIST256\_KEY\_PAIR\_GENERATE, [205](#)  
     ECP\_NIST256\_PUBLIC\_KEY\_VALIDATE, [206](#)  
     ECP\_NIST256\_SP\_DSA, [206](#)  
     ECP\_NIST256\_SVDP\_DH, [206](#)  
     ECP\_NIST256\_VP\_DSA, [207](#)  
     EFS\_NIST256, [204](#)  
     EGS\_NIST256, [204](#)  
 ecdh\_support.h, [207](#)  
     AES\_CBC\_IV0\_DECRYPT, [208](#)  
     AES\_CBC\_IV0\_ENCRYPT, [208](#)  
     ehashit, [209](#)



- HASH, [209](#)
- HMAC, [210](#)
- KDF2, [210](#)
- PBKDF2, [210](#)
- `ecp2_BLS381.h`, [211](#)
  - CURVE\_A\_BLS381, [219](#)
  - CURVE\_B\_BLS381, [219](#)
  - CURVE\_B\_I\_BLS381, [219](#)
  - CURVE\_Bnx\_BLS381, [220](#)
  - CURVE\_Cof\_BLS381, [220](#)
  - CURVE\_Gx\_BLS381, [220](#)
  - CURVE\_Gy\_BLS381, [220](#)
  - CURVE\_Order\_BLS381, [220](#)
  - CURVE\_Pxa\_BLS381, [220](#)
  - CURVE\_Pxb\_BLS381, [220](#)
  - CURVE\_Pya\_BLS381, [220](#)
  - CURVE\_Pyb\_BLS381, [221](#)
  - ECP2\_BLS381\_add, [213](#)
  - ECP2\_BLS381\_affine, [213](#)
  - ECP2\_BLS381\_copy, [213](#)
  - ECP2\_BLS381\_dbl, [213](#)
  - ECP2\_BLS381\_equals, [214](#)
  - ECP2\_BLS381\_frob, [214](#)
  - ECP2\_BLS381\_fromOctet, [214](#)
  - ECP2\_BLS381\_generator, [215](#)
  - ECP2\_BLS381\_get, [215](#)
  - ECP2\_BLS381\_inf, [215](#)
  - ECP2\_BLS381\_isinf, [216](#)
  - ECP2\_BLS381\_mapit, [216](#)
  - ECP2\_BLS381\_mul, [216](#)
  - ECP2\_BLS381\_mul4, [217](#)
  - ECP2\_BLS381\_neg, [217](#)
  - ECP2\_BLS381\_output, [217](#)
  - ECP2\_BLS381\_outputxyz, [217](#)
  - ECP2\_BLS381\_rhs, [218](#)
  - ECP2\_BLS381\_set, [218](#)
  - ECP2\_BLS381\_setx, [218](#)
  - ECP2\_BLS381\_sub, [219](#)
  - ECP2\_BLS381\_toOctet, [219](#)
  - Fra\_BLS381, [221](#)
  - Frb\_BLS381, [221](#)
- `ecp_BLS381.h`, [221](#)
  - CURVE\_A\_BLS381, [232](#)
  - CURVE\_B\_BLS381, [232](#)
  - CURVE\_B\_I\_BLS381, [232](#)
  - CURVE\_BB\_BLS381, [232](#)
  - CURVE\_Bnx\_BLS381, [232](#)
  - CURVE\_Cof\_BLS381, [232](#)
  - CURVE\_Cof\_I\_BLS381, [232](#)
  - CURVE\_Cru\_BLS381, [233](#)
  - CURVE\_Gx\_BLS381, [233](#)
  - CURVE\_Gy\_BLS381, [233](#)
  - CURVE\_Order\_BLS381, [233](#)
  - CURVE\_Pxa\_BLS381, [233](#)
  - CURVE\_Pxaa\_BLS381, [233](#)
  - CURVE\_Pxaaa\_BLS381, [233](#)
  - CURVE\_Pxaab\_BLS381, [233](#)
  - CURVE\_Pxab\_BLS381, [234](#)
  - CURVE\_Pxaba\_BLS381, [234](#)
  - CURVE\_Pxabb\_BLS381, [234](#)
  - CURVE\_Pxb\_BLS381, [234](#)
  - CURVE\_Pxba\_BLS381, [234](#)
  - CURVE\_Pxbaa\_BLS381, [234](#)
  - CURVE\_Pxbab\_BLS381, [234](#)
  - CURVE\_Pxbb\_BLS381, [234](#)
  - CURVE\_Pxbba\_BLS381, [235](#)
  - CURVE\_Pxbbb\_BLS381, [235](#)
  - CURVE\_Pya\_BLS381, [235](#)
  - CURVE\_Pyaa\_BLS381, [235](#)
  - CURVE\_Pyaaa\_BLS381, [235](#)
  - CURVE\_Pyaab\_BLS381, [235](#)
  - CURVE\_Pyab\_BLS381, [235](#)
  - CURVE\_Pyaba\_BLS381, [235](#)
  - CURVE\_Pyabb\_BLS381, [236](#)
  - CURVE\_Pyb\_BLS381, [236](#)
  - CURVE\_Pyba\_BLS381, [236](#)
  - CURVE\_Pybaa\_BLS381, [236](#)
  - CURVE\_Pybab\_BLS381, [236](#)
  - CURVE\_Pybb\_BLS381, [236](#)
  - CURVE\_Pybba\_BLS381, [236](#)
  - CURVE\_Pybbb\_BLS381, [236](#)
  - CURVE\_SB\_BLS381, [237](#)
  - CURVE\_W\_BLS381, [237](#)
  - CURVE\_WB\_BLS381, [237](#)
  - ECP\_BLS381\_add, [224](#)
  - ECP\_BLS381\_affine, [224](#)
  - ECP\_BLS381\_cfp, [224](#)
  - ECP\_BLS381\_copy, [224](#)
  - ECP\_BLS381\_dbl, [225](#)
  - ECP\_BLS381\_equals, [225](#)
  - ECP\_BLS381\_fromOctet, [225](#)
  - ECP\_BLS381\_generator, [226](#)
  - ECP\_BLS381\_get, [226](#)
  - ECP\_BLS381\_inf, [226](#)
  - ECP\_BLS381\_isinf, [226](#)
  - ECP\_BLS381\_mapit, [227](#)
  - ECP\_BLS381\_mul, [227](#)
  - ECP\_BLS381\_mul2, [227](#)
  - ECP\_BLS381\_neg, [228](#)
  - ECP\_BLS381\_output, [228](#)
  - ECP\_BLS381\_outputxyz, [228](#)
  - ECP\_BLS381\_pinmul, [228](#)
  - ECP\_BLS381\_rawoutput, [229](#)
  - ECP\_BLS381\_rhs, [229](#)
  - ECP\_BLS381\_set, [229](#)
  - ECP\_BLS381\_setx, [230](#)
  - ECP\_BLS381\_sub, [230](#)
  - ECP\_BLS381\_toOctet, [230](#)
  - Fra\_BLS381, [237](#)
  - Frb\_BLS381, [237](#)
- `ecp_ED25519.h`, [237](#)
  - CURVE\_A\_ED25519, [247](#)
  - CURVE\_B\_ED25519, [247](#)
  - CURVE\_B\_I\_ED25519, [247](#)
  - CURVE\_BB\_ED25519, [247](#)
  - CURVE\_Bnx\_ED25519, [247](#)

CURVE\_Cof\_ED25519, [247](#)  
 CURVE\_Cof\_I\_ED25519, [248](#)  
 CURVE\_Cru\_ED25519, [248](#)  
 CURVE\_Gx\_ED25519, [248](#)  
 CURVE\_Gy\_ED25519, [248](#)  
 CURVE\_Order\_ED25519, [248](#)  
 CURVE\_Pxa\_ED25519, [248](#)  
 CURVE\_Pxaa\_ED25519, [248](#)  
 CURVE\_Pxaaa\_ED25519, [248](#)  
 CURVE\_Pxaab\_ED25519, [249](#)  
 CURVE\_Pxab\_ED25519, [249](#)  
 CURVE\_Pxaba\_ED25519, [249](#)  
 CURVE\_Pxab\_b\_ED25519, [249](#)  
 CURVE\_Pxb\_ED25519, [249](#)  
 CURVE\_Pxba\_ED25519, [249](#)  
 CURVE\_Pxbaa\_ED25519, [249](#)  
 CURVE\_Pxbab\_ED25519, [249](#)  
 CURVE\_Pxbb\_ED25519, [250](#)  
 CURVE\_Pxbba\_ED25519, [250](#)  
 CURVE\_Pxbbb\_ED25519, [250](#)  
 CURVE\_Pya\_ED25519, [250](#)  
 CURVE\_Pyaa\_ED25519, [250](#)  
 CURVE\_Pyaaa\_ED25519, [250](#)  
 CURVE\_Pyaab\_ED25519, [250](#)  
 CURVE\_Pyab\_ED25519, [250](#)  
 CURVE\_Pyaba\_ED25519, [251](#)  
 CURVE\_Pyabb\_ED25519, [251](#)  
 CURVE\_Pyb\_ED25519, [251](#)  
 CURVE\_Pyba\_ED25519, [251](#)  
 CURVE\_Pybaa\_ED25519, [251](#)  
 CURVE\_Pybab\_ED25519, [251](#)  
 CURVE\_Pybb\_ED25519, [251](#)  
 CURVE\_Pybba\_ED25519, [251](#)  
 CURVE\_Pybbb\_ED25519, [252](#)  
 CURVE\_SB\_ED25519, [252](#)  
 CURVE\_W\_ED25519, [252](#)  
 CURVE\_WB\_ED25519, [252](#)  
 ECP\_ED25519\_add, [240](#)  
 ECP\_ED25519\_affine, [240](#)  
 ECP\_ED25519\_cfp, [240](#)  
 ECP\_ED25519\_copy, [240](#)  
 ECP\_ED25519\_dbl, [241](#)  
 ECP\_ED25519\_equals, [241](#)  
 ECP\_ED25519\_fromOctet, [241](#)  
 ECP\_ED25519\_generator, [242](#)  
 ECP\_ED25519\_get, [242](#)  
 ECP\_ED25519\_inf, [242](#)  
 ECP\_ED25519\_isinf, [243](#)  
 ECP\_ED25519\_mapit, [243](#)  
 ECP\_ED25519\_mul, [243](#)  
 ECP\_ED25519\_mul2, [243](#)  
 ECP\_ED25519\_neg, [244](#)  
 ECP\_ED25519\_output, [244](#)  
 ECP\_ED25519\_outputxyz, [244](#)  
 ECP\_ED25519\_pinmul, [245](#)  
 ECP\_ED25519\_rawoutput, [245](#)  
 ECP\_ED25519\_rhs, [245](#)  
 ECP\_ED25519\_set, [245](#)  
 ECP\_ED25519\_setx, [246](#)  
 ECP\_ED25519\_sub, [246](#)  
 ECP\_ED25519\_toOctet, [246](#)  
 Fra\_25519, [252](#)  
 Frb\_25519, [252](#)  
 ecp\_GOLDILOCKS.h, [253](#)  
 CURVE\_A\_GOLDILOCKS, [262](#)  
 CURVE\_B\_GOLDILOCKS, [262](#)  
 CURVE\_B\_I\_GOLDILOCKS, [262](#)  
 CURVE\_BB\_GOLDILOCKS, [262](#)  
 CURVE\_Bnx\_GOLDILOCKS, [263](#)  
 CURVE\_Cof\_GOLDILOCKS, [263](#)  
 CURVE\_Cof\_I\_GOLDILOCKS, [263](#)  
 CURVE\_Cru\_GOLDILOCKS, [263](#)  
 CURVE\_Gx\_GOLDILOCKS, [263](#)  
 CURVE\_Gy\_GOLDILOCKS, [263](#)  
 CURVE\_Order\_GOLDILOCKS, [263](#)  
 CURVE\_Pxa\_GOLDILOCKS, [263](#)  
 CURVE\_Pxaa\_GOLDILOCKS, [264](#)  
 CURVE\_Pxaaa\_GOLDILOCKS, [264](#)  
 CURVE\_Pxaab\_GOLDILOCKS, [264](#)  
 CURVE\_Pxab\_GOLDILOCKS, [264](#)  
 CURVE\_Pxaba\_GOLDILOCKS, [264](#)  
 CURVE\_Pxab\_b\_GOLDILOCKS, [264](#)  
 CURVE\_Pxb\_GOLDILOCKS, [264](#)  
 CURVE\_Pxba\_GOLDILOCKS, [264](#)  
 CURVE\_Pxbaa\_GOLDILOCKS, [265](#)  
 CURVE\_Pxbab\_GOLDILOCKS, [265](#)  
 CURVE\_Pxbb\_GOLDILOCKS, [265](#)  
 CURVE\_Pxbba\_GOLDILOCKS, [265](#)  
 CURVE\_Pxbbb\_GOLDILOCKS, [265](#)  
 CURVE\_Pya\_GOLDILOCKS, [265](#)  
 CURVE\_Pyaa\_GOLDILOCKS, [265](#)  
 CURVE\_Pyaaa\_GOLDILOCKS, [265](#)  
 CURVE\_Pyaab\_GOLDILOCKS, [266](#)  
 CURVE\_Pyab\_GOLDILOCKS, [266](#)  
 CURVE\_Pyaba\_GOLDILOCKS, [266](#)  
 CURVE\_Pyabb\_GOLDILOCKS, [266](#)  
 CURVE\_Pyb\_GOLDILOCKS, [266](#)  
 CURVE\_Pyba\_GOLDILOCKS, [266](#)  
 CURVE\_Pybaa\_GOLDILOCKS, [266](#)  
 CURVE\_Pybab\_GOLDILOCKS, [266](#)  
 CURVE\_Pybb\_GOLDILOCKS, [267](#)  
 CURVE\_Pybba\_GOLDILOCKS, [267](#)  
 CURVE\_Pybbb\_GOLDILOCKS, [267](#)  
 CURVE\_SB\_GOLDILOCKS, [267](#)  
 CURVE\_W\_GOLDILOCKS, [267](#)  
 CURVE\_WB\_GOLDILOCKS, [267](#)  
 ECP\_GOLDILOCKS\_add, [255](#)  
 ECP\_GOLDILOCKS\_affine, [255](#)  
 ECP\_GOLDILOCKS\_cfp, [255](#)  
 ECP\_GOLDILOCKS\_copy, [256](#)  
 ECP\_GOLDILOCKS\_dbl, [256](#)  
 ECP\_GOLDILOCKS\_equals, [256](#)  
 ECP\_GOLDILOCKS\_fromOctet, [257](#)  
 ECP\_GOLDILOCKS\_generator, [257](#)  
 ECP\_GOLDILOCKS\_get, [257](#)  
 ECP\_GOLDILOCKS\_inf, [258](#)



- ECP\_GOLDILOCKS\_isinf, [258](#)
- ECP\_GOLDILOCKS\_mapit, [258](#)
- ECP\_GOLDILOCKS\_mul, [258](#)
- ECP\_GOLDILOCKS\_mul2, [259](#)
- ECP\_GOLDILOCKS\_neg, [259](#)
- ECP\_GOLDILOCKS\_output, [259](#)
- ECP\_GOLDILOCKS\_outputxyz, [260](#)
- ECP\_GOLDILOCKS\_pinmul, [260](#)
- ECP\_GOLDILOCKS\_rawoutput, [260](#)
- ECP\_GOLDILOCKS\_rhs, [260](#)
- ECP\_GOLDILOCKS\_set, [261](#)
- ECP\_GOLDILOCKS\_setx, [261](#)
- ECP\_GOLDILOCKS\_sub, [261](#)
- ECP\_GOLDILOCKS\_toOctet, [262](#)
- Fra\_GOLDILOCKS, [267](#)
- Frb\_GOLDILOCKS, [267](#)
- ecp\_NIST256.h, [268](#)
  - CURVE\_A\_NIST256, [277](#)
  - CURVE\_B\_I\_NIST256, [277](#)
  - CURVE\_B\_NIST256, [277](#)
  - CURVE\_BB\_NIST256, [278](#)
  - CURVE\_Bnx\_NIST256, [278](#)
  - CURVE\_Cof\_I\_NIST256, [278](#)
  - CURVE\_Cof\_NIST256, [278](#)
  - CURVE\_Cru\_NIST256, [278](#)
  - CURVE\_Gx\_NIST256, [278](#)
  - CURVE\_Gy\_NIST256, [278](#)
  - CURVE\_Order\_NIST256, [278](#)
  - CURVE\_Pxa\_NIST256, [279](#)
  - CURVE\_Pxaa\_NIST256, [279](#)
  - CURVE\_Pxaaa\_NIST256, [279](#)
  - CURVE\_Pxaab\_NIST256, [279](#)
  - CURVE\_Pxab\_NIST256, [279](#)
  - CURVE\_Pxaba\_NIST256, [279](#)
  - CURVE\_Pxab\_b\_NIST256, [279](#)
  - CURVE\_Pxb\_NIST256, [279](#)
  - CURVE\_Pxba\_NIST256, [280](#)
  - CURVE\_Pxbaa\_NIST256, [280](#)
  - CURVE\_Pxbab\_NIST256, [280](#)
  - CURVE\_Pxbb\_NIST256, [280](#)
  - CURVE\_Pxbba\_NIST256, [280](#)
  - CURVE\_Pxbbb\_NIST256, [280](#)
  - CURVE\_Pya\_NIST256, [280](#)
  - CURVE\_Pyaa\_NIST256, [280](#)
  - CURVE\_Pyaaa\_NIST256, [281](#)
  - CURVE\_Pyaab\_NIST256, [281](#)
  - CURVE\_Pyab\_NIST256, [281](#)
  - CURVE\_Pyaba\_NIST256, [281](#)
  - CURVE\_Pyabb\_NIST256, [281](#)
  - CURVE\_Pyb\_NIST256, [281](#)
  - CURVE\_Pyba\_NIST256, [281](#)
  - CURVE\_Pybaa\_NIST256, [281](#)
  - CURVE\_Pybab\_NIST256, [282](#)
  - CURVE\_Pybb\_NIST256, [282](#)
  - CURVE\_Pybba\_NIST256, [282](#)
  - CURVE\_Pybbb\_NIST256, [282](#)
  - CURVE\_SB\_NIST256, [282](#)
  - CURVE\_W\_NIST256, [282](#)
  - CURVE\_WB\_NIST256, [282](#)
  - ECP\_NIST256\_add, [270](#)
  - ECP\_NIST256\_affine, [270](#)
  - ECP\_NIST256\_cfp, [271](#)
  - ECP\_NIST256\_copy, [271](#)
  - ECP\_NIST256\_dbl, [271](#)
  - ECP\_NIST256\_equals, [271](#)
  - ECP\_NIST256\_fromOctet, [272](#)
  - ECP\_NIST256\_generator, [272](#)
  - ECP\_NIST256\_get, [272](#)
  - ECP\_NIST256\_inf, [273](#)
  - ECP\_NIST256\_isinf, [273](#)
  - ECP\_NIST256\_mapit, [273](#)
  - ECP\_NIST256\_mul, [274](#)
  - ECP\_NIST256\_mul2, [274](#)
  - ECP\_NIST256\_neg, [274](#)
  - ECP\_NIST256\_output, [274](#)
  - ECP\_NIST256\_outputxyz, [275](#)
  - ECP\_NIST256\_pinmul, [275](#)
  - ECP\_NIST256\_rawoutput, [275](#)
  - ECP\_NIST256\_rhs, [275](#)
  - ECP\_NIST256\_set, [276](#)
  - ECP\_NIST256\_setx, [276](#)
  - ECP\_NIST256\_sub, [277](#)
  - ECP\_NIST256\_toOctet, [277](#)
  - Fra\_NIST256, [282](#)
  - Frb\_NIST256, [283](#)
- ehashit
  - ecdh\_support.h, [209](#)
- f
  - amcl\_aes, [15](#)
- FEXCESS\_25519
  - fp\_25519.h, [346](#)
- FEXCESS\_BLS381
  - fp\_BLS381.h, [357](#)
- FEXCESS\_GOLDILOCKS
  - fp\_GOLDILOCKS.h, [368](#)
- FEXCESS\_NIST256
  - fp\_NIST256.h, [378](#)
- FF\_2048\_add
  - ff\_2048.h, [285](#)
- FF\_2048\_cfactor
  - ff\_2048.h, [286](#)
- FF\_2048\_comp
  - ff\_2048.h, [286](#)
- FF\_2048\_copy
  - ff\_2048.h, [286](#)
- FF\_2048\_dec
  - ff\_2048.h, [287](#)
- FF\_2048\_dmod
  - ff\_2048.h, [287](#)
- FF\_2048\_fromOctet
  - ff\_2048.h, [287](#)
- FF\_2048\_inc
  - ff\_2048.h, [288](#)
- FF\_2048\_init
  - ff\_2048.h, [288](#)
- FF\_2048\_invmodp

ff\_2048.h, [288](#)  
FF\_2048\_iszilch  
ff\_2048.h, [289](#)  
FF\_2048\_lastbits  
ff\_2048.h, [289](#)  
FF\_2048\_mod  
ff\_2048.h, [289](#)  
FF\_2048\_mul  
ff\_2048.h, [290](#)  
FF\_2048\_norm  
ff\_2048.h, [290](#)  
FF\_2048\_one  
ff\_2048.h, [290](#)  
FF\_2048\_output  
ff\_2048.h, [291](#)  
FF\_2048\_parity  
ff\_2048.h, [291](#)  
FF\_2048\_pow  
ff\_2048.h, [291](#)  
FF\_2048\_pow2  
ff\_2048.h, [292](#)  
FF\_2048\_power  
ff\_2048.h, [292](#)  
FF\_2048\_prime  
ff\_2048.h, [293](#)  
FF\_2048\_random  
ff\_2048.h, [293](#)  
FF\_2048\_randomnum  
ff\_2048.h, [293](#)  
FF\_2048\_rawoutput  
ff\_2048.h, [294](#)  
FF\_2048\_shl  
ff\_2048.h, [294](#)  
FF\_2048\_shr  
ff\_2048.h, [294](#)  
FF\_2048\_skpow  
ff\_2048.h, [294](#)  
FF\_2048\_skspow  
ff\_2048.h, [295](#)  
FF\_2048\_sqr  
ff\_2048.h, [295](#)  
FF\_2048\_sub  
ff\_2048.h, [296](#)  
FF\_2048\_toOctet  
ff\_2048.h, [296](#)  
FF\_2048\_zero  
ff\_2048.h, [296](#)  
FF\_3072\_add  
ff\_3072.h, [299](#)  
FF\_3072\_cfactor  
ff\_3072.h, [300](#)  
FF\_3072\_comp  
ff\_3072.h, [300](#)  
FF\_3072\_copy  
ff\_3072.h, [300](#)  
FF\_3072\_dec  
ff\_3072.h, [301](#)  
FF\_3072\_dmod  
ff\_3072.h, [301](#)  
FF\_3072\_fromOctet  
ff\_3072.h, [301](#)  
FF\_3072\_inc  
ff\_3072.h, [302](#)  
FF\_3072\_init  
ff\_3072.h, [302](#)  
FF\_3072\_invmodp  
ff\_3072.h, [302](#)  
FF\_3072\_iszilch  
ff\_3072.h, [303](#)  
FF\_3072\_lastbits  
ff\_3072.h, [303](#)  
FF\_3072\_mod  
ff\_3072.h, [303](#)  
FF\_3072\_mul  
ff\_3072.h, [304](#)  
FF\_3072\_norm  
ff\_3072.h, [304](#)  
FF\_3072\_one  
ff\_3072.h, [304](#)  
FF\_3072\_output  
ff\_3072.h, [305](#)  
FF\_3072\_parity  
ff\_3072.h, [305](#)  
FF\_3072\_pow  
ff\_3072.h, [305](#)  
FF\_3072\_pow2  
ff\_3072.h, [306](#)  
FF\_3072\_power  
ff\_3072.h, [306](#)  
FF\_3072\_prime  
ff\_3072.h, [307](#)  
FF\_3072\_random  
ff\_3072.h, [307](#)  
FF\_3072\_randomnum  
ff\_3072.h, [307](#)  
FF\_3072\_rawoutput  
ff\_3072.h, [308](#)  
FF\_3072\_shl  
ff\_3072.h, [308](#)  
FF\_3072\_shr  
ff\_3072.h, [308](#)  
FF\_3072\_skpow  
ff\_3072.h, [308](#)  
FF\_3072\_skspow  
ff\_3072.h, [309](#)  
FF\_3072\_sqr  
ff\_3072.h, [309](#)  
FF\_3072\_sub  
ff\_3072.h, [310](#)  
FF\_3072\_toOctet  
ff\_3072.h, [310](#)  
FF\_3072\_zero  
ff\_3072.h, [310](#)  
FFLEN\_2048  
config\_ff\_2048.h, [186](#)  
FFLEN\_3072

config\_ff\_3072.h, [187](#)  
 FP12\_BLS381, [22](#)  
   a, [22](#)  
   b, [22](#)  
   c, [22](#)  
   type, [22](#)  
 FP12\_BLS381\_cmove  
   fp12\_BLS381.h, [312](#)  
 FP12\_BLS381\_compow  
   fp12\_BLS381.h, [313](#)  
 FP12\_BLS381\_conj  
   fp12\_BLS381.h, [313](#)  
 FP12\_BLS381\_copy  
   fp12\_BLS381.h, [313](#)  
 FP12\_BLS381\_equals  
   fp12\_BLS381.h, [314](#)  
 FP12\_BLS381\_frob  
   fp12\_BLS381.h, [314](#)  
 FP12\_BLS381\_from\_FP4  
   fp12\_BLS381.h, [314](#)  
 FP12\_BLS381\_from\_FP4s  
   fp12\_BLS381.h, [315](#)  
 FP12\_BLS381\_fromOctet  
   fp12\_BLS381.h, [315](#)  
 FP12\_BLS381\_inv  
   fp12\_BLS381.h, [315](#)  
 FP12\_BLS381\_isunity  
   fp12\_BLS381.h, [316](#)  
 FP12\_BLS381\_iszilch  
   fp12\_BLS381.h, [316](#)  
 FP12\_BLS381\_mul  
   fp12\_BLS381.h, [316](#)  
 FP12\_BLS381\_norm  
   fp12\_BLS381.h, [316](#)  
 FP12\_BLS381\_one  
   fp12\_BLS381.h, [317](#)  
 FP12\_BLS381\_output  
   fp12\_BLS381.h, [317](#)  
 FP12\_BLS381\_pinpow  
   fp12\_BLS381.h, [317](#)  
 FP12\_BLS381\_pow  
   fp12\_BLS381.h, [317](#)  
 FP12\_BLS381\_pow4  
   fp12\_BLS381.h, [318](#)  
 FP12\_BLS381\_reduce  
   fp12\_BLS381.h, [318](#)  
 FP12\_BLS381\_smul  
   fp12\_BLS381.h, [318](#)  
 FP12\_BLS381\_sqr  
   fp12\_BLS381.h, [319](#)  
 FP12\_BLS381\_ssmul  
   fp12\_BLS381.h, [319](#)  
 FP12\_BLS381\_toOctet  
   fp12\_BLS381.h, [319](#)  
 FP12\_BLS381\_trace  
   fp12\_BLS381.h, [319](#)  
 FP12\_BLS381\_usqr  
   fp12\_BLS381.h, [320](#)  
 FP12\_BLS381\_zero  
   fp12\_BLS381.h, [320](#)  
 FP2\_BLS381, [23](#)  
   a, [23](#)  
   b, [23](#)  
 FP2\_BLS381\_add  
   fp2\_BLS381.h, [322](#)  
 FP2\_BLS381\_cmove  
   fp2\_BLS381.h, [323](#)  
 FP2\_BLS381\_conj  
   fp2\_BLS381.h, [323](#)  
 FP2\_BLS381\_copy  
   fp2\_BLS381.h, [323](#)  
 FP2\_BLS381\_div2  
   fp2\_BLS381.h, [324](#)  
 FP2\_BLS381\_div\_ip  
   fp2\_BLS381.h, [324](#)  
 FP2\_BLS381\_div\_ip2  
   fp2\_BLS381.h, [324](#)  
 FP2\_BLS381\_equals  
   fp2\_BLS381.h, [324](#)  
 FP2\_BLS381\_from\_BIGs  
   fp2\_BLS381.h, [325](#)  
 FP2\_BLS381\_from\_BIG  
   fp2\_BLS381.h, [325](#)  
 FP2\_BLS381\_from\_FPs  
   fp2\_BLS381.h, [326](#)  
 FP2\_BLS381\_from\_FP  
   fp2\_BLS381.h, [325](#)  
 FP2\_BLS381\_imul  
   fp2\_BLS381.h, [326](#)  
 FP2\_BLS381\_inv  
   fp2\_BLS381.h, [326](#)  
 FP2\_BLS381\_isunity  
   fp2\_BLS381.h, [327](#)  
 FP2\_BLS381\_iszilch  
   fp2\_BLS381.h, [327](#)  
 FP2\_BLS381\_mul  
   fp2\_BLS381.h, [327](#)  
 FP2\_BLS381\_mul\_ip  
   fp2\_BLS381.h, [328](#)  
 FP2\_BLS381\_neg  
   fp2\_BLS381.h, [328](#)  
 FP2\_BLS381\_norm  
   fp2\_BLS381.h, [328](#)  
 FP2\_BLS381\_one  
   fp2\_BLS381.h, [328](#)  
 FP2\_BLS381\_output  
   fp2\_BLS381.h, [329](#)  
 FP2\_BLS381\_pmul  
   fp2\_BLS381.h, [329](#)  
 FP2\_BLS381\_pow  
   fp2\_BLS381.h, [329](#)  
 FP2\_BLS381\_rawoutput  
   fp2\_BLS381.h, [330](#)  
 FP2\_BLS381\_reduce  
   fp2\_BLS381.h, [330](#)  
 FP2\_BLS381\_sqr

fp2\_BLS381.h, [330](#)  
 FP2\_BLS381\_sqrt  
   fp2\_BLS381.h, [330](#)  
 FP2\_BLS381\_sub  
   fp2\_BLS381.h, [331](#)  
 FP2\_BLS381\_times\_i  
   fp2\_BLS381.h, [331](#)  
 FP2\_BLS381\_zero  
   fp2\_BLS381.h, [331](#)  
 FP4\_BLS381, [23](#)  
   a, [23](#)  
   b, [24](#)  
 FP4\_BLS381\_add  
   fp4\_BLS381.h, [333](#)  
 FP4\_BLS381\_cmove  
   fp4\_BLS381.h, [334](#)  
 FP4\_BLS381\_conj  
   fp4\_BLS381.h, [334](#)  
 FP4\_BLS381\_copy  
   fp4\_BLS381.h, [334](#)  
 FP4\_BLS381\_div2  
   fp4\_BLS381.h, [335](#)  
 FP4\_BLS381\_div\_2i  
   fp4\_BLS381.h, [335](#)  
 FP4\_BLS381\_div\_i  
   fp4\_BLS381.h, [335](#)  
 FP4\_BLS381\_equals  
   fp4\_BLS381.h, [335](#)  
 FP4\_BLS381\_frob  
   fp4\_BLS381.h, [336](#)  
 FP4\_BLS381\_from\_FP2  
   fp4\_BLS381.h, [336](#)  
 FP4\_BLS381\_from\_FP2H  
   fp4\_BLS381.h, [336](#)  
 FP4\_BLS381\_from\_FP2s  
   fp4\_BLS381.h, [337](#)  
 FP4\_BLS381\_imul  
   fp4\_BLS381.h, [337](#)  
 FP4\_BLS381\_inv  
   fp4\_BLS381.h, [337](#)  
 FP4\_BLS381\_isreal  
   fp4\_BLS381.h, [338](#)  
 FP4\_BLS381\_isunity  
   fp4\_BLS381.h, [338](#)  
 FP4\_BLS381\_iszilch  
   fp4\_BLS381.h, [338](#)  
 FP4\_BLS381\_mul  
   fp4\_BLS381.h, [339](#)  
 FP4\_BLS381\_nconj  
   fp4\_BLS381.h, [339](#)  
 FP4\_BLS381\_neg  
   fp4\_BLS381.h, [339](#)  
 FP4\_BLS381\_norm  
   fp4\_BLS381.h, [339](#)  
 FP4\_BLS381\_one  
   fp4\_BLS381.h, [340](#)  
 FP4\_BLS381\_output  
   fp4\_BLS381.h, [340](#)  
 FP4\_BLS381\_pmul  
   fp4\_BLS381.h, [340](#)  
 FP4\_BLS381\_pow  
   fp4\_BLS381.h, [340](#)  
 FP4\_BLS381\_qmul  
   fp4\_BLS381.h, [341](#)  
 FP4\_BLS381\_rawoutput  
   fp4\_BLS381.h, [341](#)  
 FP4\_BLS381\_reduce  
   fp4\_BLS381.h, [341](#)  
 FP4\_BLS381\_sqr  
   fp4\_BLS381.h, [342](#)  
 FP4\_BLS381\_sqrt  
   fp4\_BLS381.h, [342](#)  
 FP4\_BLS381\_sub  
   fp4\_BLS381.h, [342](#)  
 FP4\_BLS381\_times\_i  
   fp4\_BLS381.h, [343](#)  
 FP4\_BLS381\_xtr\_A  
   fp4\_BLS381.h, [343](#)  
 FP4\_BLS381\_xtr\_D  
   fp4\_BLS381.h, [343](#)  
 FP4\_BLS381\_xtr\_pow  
   fp4\_BLS381.h, [343](#)  
 FP4\_BLS381\_xtr\_pow2  
   fp4\_BLS381.h, [344](#)  
 FP4\_BLS381\_zero  
   fp4\_BLS381.h, [344](#)  
 FP\_25519, [24](#)  
   g, [24](#)  
   XES, [24](#)  
 FP\_25519\_add  
   fp\_25519.h, [347](#)  
 FP\_25519\_cmove  
   fp\_25519.h, [347](#)  
 FP\_25519\_copy  
   fp\_25519.h, [348](#)  
 FP\_25519\_cswap  
   fp\_25519.h, [348](#)  
 FP\_25519\_div2  
   fp\_25519.h, [348](#)  
 FP\_25519\_equals  
   fp\_25519.h, [349](#)  
 FP\_25519\_imul  
   fp\_25519.h, [349](#)  
 FP\_25519\_inv  
   fp\_25519.h, [349](#)  
 FP\_25519\_iszilch  
   fp\_25519.h, [350](#)  
 FP\_25519\_mod  
   fp\_25519.h, [350](#)  
 FP\_25519\_mul  
   fp\_25519.h, [350](#)  
 FP\_25519\_neg  
   fp\_25519.h, [351](#)  
 FP\_25519\_norm  
   fp\_25519.h, [351](#)  
 FP\_25519\_nres

- fp\_25519.h, [351](#)
- FP\_25519\_one
  - fp\_25519.h, [351](#)
- FP\_25519\_output
  - fp\_25519.h, [352](#)
- FP\_25519\_pow
  - fp\_25519.h, [352](#)
- FP\_25519\_qr
  - fp\_25519.h, [352](#)
- FP\_25519\_rawoutput
  - fp\_25519.h, [353](#)
- FP\_25519\_rcopy
  - fp\_25519.h, [353](#)
- FP\_25519\_redc
  - fp\_25519.h, [353](#)
- FP\_25519\_reduce
  - fp\_25519.h, [353](#)
- FP\_25519\_sqr
  - fp\_25519.h, [354](#)
- FP\_25519\_sqrt
  - fp\_25519.h, [354](#)
- FP\_25519\_sub
  - fp\_25519.h, [354](#)
- FP\_25519\_zero
  - fp\_25519.h, [355](#)
- FP\_BLS381, [25](#)
  - g, [25](#)
  - XES, [25](#)
- FP\_BLS381\_add
  - fp\_BLS381.h, [358](#)
- FP\_BLS381\_cmove
  - fp\_BLS381.h, [358](#)
- FP\_BLS381\_copy
  - fp\_BLS381.h, [358](#)
- FP\_BLS381\_cswap
  - fp\_BLS381.h, [359](#)
- FP\_BLS381\_div2
  - fp\_BLS381.h, [359](#)
- FP\_BLS381\_equals
  - fp\_BLS381.h, [359](#)
- FP\_BLS381\_imul
  - fp\_BLS381.h, [360](#)
- FP\_BLS381\_inv
  - fp\_BLS381.h, [360](#)
- FP\_BLS381\_iszilch
  - fp\_BLS381.h, [360](#)
- FP\_BLS381\_mod
  - fp\_BLS381.h, [361](#)
- FP\_BLS381\_mul
  - fp\_BLS381.h, [361](#)
- FP\_BLS381\_neg
  - fp\_BLS381.h, [361](#)
- FP\_BLS381\_norm
  - fp\_BLS381.h, [362](#)
- FP\_BLS381\_nres
  - fp\_BLS381.h, [362](#)
- FP\_BLS381\_one
  - fp\_BLS381.h, [362](#)
- FP\_BLS381\_output
  - fp\_BLS381.h, [362](#)
- FP\_BLS381\_pow
  - fp\_BLS381.h, [363](#)
- FP\_BLS381\_qr
  - fp\_BLS381.h, [363](#)
- FP\_BLS381\_rawoutput
  - fp\_BLS381.h, [363](#)
- FP\_BLS381\_rcopy
  - fp\_BLS381.h, [363](#)
- FP\_BLS381\_redc
  - fp\_BLS381.h, [364](#)
- FP\_BLS381\_reduce
  - fp\_BLS381.h, [364](#)
- FP\_BLS381\_sqr
  - fp\_BLS381.h, [364](#)
- FP\_BLS381\_sqrt
  - fp\_BLS381.h, [365](#)
- FP\_BLS381\_sub
  - fp\_BLS381.h, [365](#)
- FP\_BLS381\_zero
  - fp\_BLS381.h, [365](#)
- FP\_GOLDILOCKS\_add
  - fp\_GOLDILOCKS.h, [368](#)
- FP\_GOLDILOCKS\_cmove
  - fp\_GOLDILOCKS.h, [369](#)
- FP\_GOLDILOCKS\_copy
  - fp\_GOLDILOCKS.h, [369](#)
- FP\_GOLDILOCKS\_cswap
  - fp\_GOLDILOCKS.h, [369](#)
- FP\_GOLDILOCKS\_div2
  - fp\_GOLDILOCKS.h, [370](#)
- FP\_GOLDILOCKS\_equals
  - fp\_GOLDILOCKS.h, [370](#)
- FP\_GOLDILOCKS\_imul
  - fp\_GOLDILOCKS.h, [370](#)
- FP\_GOLDILOCKS\_inv
  - fp\_GOLDILOCKS.h, [371](#)
- FP\_GOLDILOCKS\_iszilch
  - fp\_GOLDILOCKS.h, [371](#)
- FP\_GOLDILOCKS\_mod
  - fp\_GOLDILOCKS.h, [371](#)
- FP\_GOLDILOCKS\_mul
  - fp\_GOLDILOCKS.h, [371](#)
- FP\_GOLDILOCKS\_neg
  - fp\_GOLDILOCKS.h, [372](#)
- FP\_GOLDILOCKS\_norm
  - fp\_GOLDILOCKS.h, [372](#)
- FP\_GOLDILOCKS\_nres
  - fp\_GOLDILOCKS.h, [372](#)
- FP\_GOLDILOCKS\_one
  - fp\_GOLDILOCKS.h, [373](#)
- FP\_GOLDILOCKS\_output
  - fp\_GOLDILOCKS.h, [373](#)
- FP\_GOLDILOCKS\_pow
  - fp\_GOLDILOCKS.h, [373](#)
- FP\_GOLDILOCKS\_qr
  - fp\_GOLDILOCKS.h, [373](#)

FP\_GOLDILOCKS\_rawoutput  
     fp\_GOLDILOCKS.h, 374  
 FP\_GOLDILOCKS\_rcopy  
     fp\_GOLDILOCKS.h, 374  
 FP\_GOLDILOCKS\_redc  
     fp\_GOLDILOCKS.h, 374  
 FP\_GOLDILOCKS\_reduce  
     fp\_GOLDILOCKS.h, 375  
 FP\_GOLDILOCKS\_sqr  
     fp\_GOLDILOCKS.h, 375  
 FP\_GOLDILOCKS\_sqrt  
     fp\_GOLDILOCKS.h, 375  
 FP\_GOLDILOCKS\_sub  
     fp\_GOLDILOCKS.h, 375  
 FP\_GOLDILOCKS\_zero  
     fp\_GOLDILOCKS.h, 376  
 FP\_GOLDILOCKS, 25  
     g, 25  
     XES, 26  
 FP\_NIST256, 26  
     g, 26  
     XES, 26  
 FP\_NIST256\_add  
     fp\_NIST256.h, 379  
 FP\_NIST256\_cmove  
     fp\_NIST256.h, 379  
 FP\_NIST256\_copy  
     fp\_NIST256.h, 380  
 FP\_NIST256\_cswap  
     fp\_NIST256.h, 380  
 FP\_NIST256\_div2  
     fp\_NIST256.h, 380  
 FP\_NIST256\_equals  
     fp\_NIST256.h, 381  
 FP\_NIST256\_imul  
     fp\_NIST256.h, 381  
 FP\_NIST256\_inv  
     fp\_NIST256.h, 381  
 FP\_NIST256\_iszilch  
     fp\_NIST256.h, 381  
 FP\_NIST256\_mod  
     fp\_NIST256.h, 382  
 FP\_NIST256\_mul  
     fp\_NIST256.h, 382  
 FP\_NIST256\_neg  
     fp\_NIST256.h, 382  
 FP\_NIST256\_norm  
     fp\_NIST256.h, 383  
 FP\_NIST256\_nres  
     fp\_NIST256.h, 383  
 FP\_NIST256\_one  
     fp\_NIST256.h, 383  
 FP\_NIST256\_output  
     fp\_NIST256.h, 383  
 FP\_NIST256\_pow  
     fp\_NIST256.h, 384  
 FP\_NIST256\_qr  
     fp\_NIST256.h, 384  
 FP\_NIST256\_rawoutput  
     fp\_NIST256.h, 384  
 FP\_NIST256\_rcopy  
     fp\_NIST256.h, 385  
 FP\_NIST256\_redc  
     fp\_NIST256.h, 385  
 FP\_NIST256\_reduce  
     fp\_NIST256.h, 385  
 FP\_NIST256\_sqr  
     fp\_NIST256.h, 385  
 FP\_NIST256\_sqrt  
     fp\_NIST256.h, 386  
 FP\_NIST256\_sub  
     fp\_NIST256.h, 386  
 FP\_NIST256\_zero  
     fp\_NIST256.h, 386  
 ff\_2048.h, 283  
     FF\_2048\_add, 285  
     FF\_2048\_cfactor, 286  
     FF\_2048\_comp, 286  
     FF\_2048\_copy, 286  
     FF\_2048\_dec, 287  
     FF\_2048\_dmod, 287  
     FF\_2048\_fromOctet, 287  
     FF\_2048\_inc, 288  
     FF\_2048\_init, 288  
     FF\_2048\_invmodp, 288  
     FF\_2048\_iszilch, 289  
     FF\_2048\_lastbits, 289  
     FF\_2048\_mod, 289  
     FF\_2048\_mul, 290  
     FF\_2048\_norm, 290  
     FF\_2048\_one, 290  
     FF\_2048\_output, 291  
     FF\_2048\_parity, 291  
     FF\_2048\_pow, 291  
     FF\_2048\_pow2, 292  
     FF\_2048\_power, 292  
     FF\_2048\_prime, 293  
     FF\_2048\_random, 293  
     FF\_2048\_randomnum, 293  
     FF\_2048\_rawoutput, 294  
     FF\_2048\_shl, 294  
     FF\_2048\_shr, 294  
     FF\_2048\_skpow, 294  
     FF\_2048\_skspow, 295  
     FF\_2048\_sqr, 295  
     FF\_2048\_sub, 296  
     FF\_2048\_toOctet, 296  
     FF\_2048\_zero, 296  
     HFLEN\_2048, 285  
     P\_EXCESS\_2048, 285  
     P\_FEXCESS\_2048, 285  
     P\_MBITS\_2048, 285  
     P\_TBITS\_2048, 285  
 ff\_3072.h, 297  
     FF\_3072\_add, 299  
     FF\_3072\_cfactor, 300

- FF\_3072\_comp, [300](#)
- FF\_3072\_copy, [300](#)
- FF\_3072\_dec, [301](#)
- FF\_3072\_dmod, [301](#)
- FF\_3072\_fromOctet, [301](#)
- FF\_3072\_inc, [302](#)
- FF\_3072\_init, [302](#)
- FF\_3072\_invmodp, [302](#)
- FF\_3072\_iszilch, [303](#)
- FF\_3072\_lastbits, [303](#)
- FF\_3072\_mod, [303](#)
- FF\_3072\_mul, [304](#)
- FF\_3072\_norm, [304](#)
- FF\_3072\_one, [304](#)
- FF\_3072\_output, [305](#)
- FF\_3072\_parity, [305](#)
- FF\_3072\_pow, [305](#)
- FF\_3072\_pow2, [306](#)
- FF\_3072\_power, [306](#)
- FF\_3072\_prime, [307](#)
- FF\_3072\_random, [307](#)
- FF\_3072\_randomnum, [307](#)
- FF\_3072\_rawoutput, [308](#)
- FF\_3072\_shl, [308](#)
- FF\_3072\_shr, [308](#)
- FF\_3072\_skpow, [308](#)
- FF\_3072\_skspow, [309](#)
- FF\_3072\_sqr, [309](#)
- FF\_3072\_sub, [310](#)
- FF\_3072\_toOctet, [310](#)
- FF\_3072\_zero, [310](#)
- HFLen\_3072, [299](#)
- P\_EXCESS\_3072, [299](#)
- P\_FEXCESS\_3072, [299](#)
- P\_MBITS\_3072, [299](#)
- P\_TBITS\_3072, [299](#)
- fkey
  - amcl\_aes, [15](#)
- fp12\_BLS381.h, [311](#)
  - FP12\_BLS381\_cmove, [312](#)
  - FP12\_BLS381\_compow, [313](#)
  - FP12\_BLS381\_conj, [313](#)
  - FP12\_BLS381\_copy, [313](#)
  - FP12\_BLS381\_equals, [314](#)
  - FP12\_BLS381\_frob, [314](#)
  - FP12\_BLS381\_from\_FP4, [314](#)
  - FP12\_BLS381\_from\_FP4s, [315](#)
  - FP12\_BLS381\_fromOctet, [315](#)
  - FP12\_BLS381\_inv, [315](#)
  - FP12\_BLS381\_isunity, [316](#)
  - FP12\_BLS381\_iszilch, [316](#)
  - FP12\_BLS381\_mul, [316](#)
  - FP12\_BLS381\_norm, [316](#)
  - FP12\_BLS381\_one, [317](#)
  - FP12\_BLS381\_output, [317](#)
  - FP12\_BLS381\_pinpow, [317](#)
  - FP12\_BLS381\_pow, [317](#)
  - FP12\_BLS381\_pow4, [318](#)
  - FP12\_BLS381\_reduce, [318](#)
  - FP12\_BLS381\_smul, [318](#)
  - FP12\_BLS381\_sqr, [319](#)
  - FP12\_BLS381\_ssmul, [319](#)
  - FP12\_BLS381\_toOctet, [319](#)
  - FP12\_BLS381\_trace, [319](#)
  - FP12\_BLS381\_usqr, [320](#)
  - FP12\_BLS381\_zero, [320](#)
  - Fra\_BLS381, [320](#)
  - Frb\_BLS381, [320](#)
- fp2\_BLS381.h, [321](#)
  - FP2\_BLS381\_add, [322](#)
  - FP2\_BLS381\_cmove, [323](#)
  - FP2\_BLS381\_conj, [323](#)
  - FP2\_BLS381\_copy, [323](#)
  - FP2\_BLS381\_div2, [324](#)
  - FP2\_BLS381\_div\_ip, [324](#)
  - FP2\_BLS381\_div\_ip2, [324](#)
  - FP2\_BLS381\_equals, [324](#)
  - FP2\_BLS381\_from\_BIGs, [325](#)
  - FP2\_BLS381\_from\_BIG, [325](#)
  - FP2\_BLS381\_from\_FPs, [326](#)
  - FP2\_BLS381\_from\_FP, [325](#)
  - FP2\_BLS381\_imul, [326](#)
  - FP2\_BLS381\_inv, [326](#)
  - FP2\_BLS381\_isunity, [327](#)
  - FP2\_BLS381\_iszilch, [327](#)
  - FP2\_BLS381\_mul, [327](#)
  - FP2\_BLS381\_mul\_ip, [328](#)
  - FP2\_BLS381\_neg, [328](#)
  - FP2\_BLS381\_norm, [328](#)
  - FP2\_BLS381\_one, [328](#)
  - FP2\_BLS381\_output, [329](#)
  - FP2\_BLS381\_pmul, [329](#)
  - FP2\_BLS381\_pow, [329](#)
  - FP2\_BLS381\_rawoutput, [330](#)
  - FP2\_BLS381\_reduce, [330](#)
  - FP2\_BLS381\_sqr, [330](#)
  - FP2\_BLS381\_sqrt, [330](#)
  - FP2\_BLS381\_sub, [331](#)
  - FP2\_BLS381\_times\_i, [331](#)
  - FP2\_BLS381\_zero, [331](#)
- fp4\_BLS381.h, [331](#)
  - FP4\_BLS381\_add, [333](#)
  - FP4\_BLS381\_cmove, [334](#)
  - FP4\_BLS381\_conj, [334](#)
  - FP4\_BLS381\_copy, [334](#)
  - FP4\_BLS381\_div2, [335](#)
  - FP4\_BLS381\_div\_2i, [335](#)
  - FP4\_BLS381\_div\_i, [335](#)
  - FP4\_BLS381\_equals, [335](#)
  - FP4\_BLS381\_frob, [336](#)
  - FP4\_BLS381\_from\_FP2, [336](#)
  - FP4\_BLS381\_from\_FP2H, [336](#)
  - FP4\_BLS381\_from\_FP2s, [337](#)
  - FP4\_BLS381\_imul, [337](#)
  - FP4\_BLS381\_inv, [337](#)
  - FP4\_BLS381\_isreal, [338](#)



- FP4\_BLS381\_isunity, 338
- FP4\_BLS381\_iszilch, 338
- FP4\_BLS381\_mul, 339
- FP4\_BLS381\_nconj, 339
- FP4\_BLS381\_neg, 339
- FP4\_BLS381\_norm, 339
- FP4\_BLS381\_one, 340
- FP4\_BLS381\_output, 340
- FP4\_BLS381\_pmul, 340
- FP4\_BLS381\_pow, 340
- FP4\_BLS381\_qmul, 341
- FP4\_BLS381\_rawoutput, 341
- FP4\_BLS381\_reduce, 341
- FP4\_BLS381\_sqr, 342
- FP4\_BLS381\_sqrt, 342
- FP4\_BLS381\_sub, 342
- FP4\_BLS381\_times\_i, 343
- FP4\_BLS381\_xtr\_A, 343
- FP4\_BLS381\_xtr\_D, 343
- FP4\_BLS381\_xtr\_pow, 343
- FP4\_BLS381\_xtr\_pow2, 344
- FP4\_BLS381\_zero, 344
- fp\_25519.h, 345
  - FEXCESS\_25519, 346
  - FP\_25519\_add, 347
  - FP\_25519\_cmove, 347
  - FP\_25519\_copy, 348
  - FP\_25519\_cswap, 348
  - FP\_25519\_div2, 348
  - FP\_25519\_equals, 349
  - FP\_25519\_imul, 349
  - FP\_25519\_inv, 349
  - FP\_25519\_iszilch, 350
  - FP\_25519\_mod, 350
  - FP\_25519\_mul, 350
  - FP\_25519\_neg, 351
  - FP\_25519\_norm, 351
  - FP\_25519\_nres, 351
  - FP\_25519\_one, 351
  - FP\_25519\_output, 352
  - FP\_25519\_pow, 352
  - FP\_25519\_qr, 352
  - FP\_25519\_rawoutput, 353
  - FP\_25519\_rcopy, 353
  - FP\_25519\_redc, 353
  - FP\_25519\_reduce, 353
  - FP\_25519\_sqr, 354
  - FP\_25519\_sqrt, 354
  - FP\_25519\_sub, 354
  - FP\_25519\_zero, 355
  - MConst\_25519, 355
  - MODBITS\_25519, 346
  - Modulus\_25519, 355
  - OMASK\_25519, 347
  - R2modp\_25519, 355
  - TBITS\_25519, 347
  - TMASK\_25519, 347
- fp\_BLS381.h, 355
  - FEXCESS\_BLS381, 357
  - FP\_BLS381\_add, 358
  - FP\_BLS381\_cmove, 358
  - FP\_BLS381\_copy, 358
  - FP\_BLS381\_cswap, 359
  - FP\_BLS381\_div2, 359
  - FP\_BLS381\_equals, 359
  - FP\_BLS381\_imul, 360
  - FP\_BLS381\_inv, 360
  - FP\_BLS381\_iszilch, 360
  - FP\_BLS381\_mod, 361
  - FP\_BLS381\_mul, 361
  - FP\_BLS381\_neg, 361
  - FP\_BLS381\_norm, 362
  - FP\_BLS381\_nres, 362
  - FP\_BLS381\_one, 362
  - FP\_BLS381\_output, 362
  - FP\_BLS381\_pow, 363
  - FP\_BLS381\_qr, 363
  - FP\_BLS381\_rawoutput, 363
  - FP\_BLS381\_rcopy, 363
  - FP\_BLS381\_redc, 364
  - FP\_BLS381\_reduce, 364
  - FP\_BLS381\_sqr, 364
  - FP\_BLS381\_sqrt, 365
  - FP\_BLS381\_sub, 365
  - FP\_BLS381\_zero, 365
  - MConst\_BLS381, 365
  - MODBITS\_BLS381, 357
  - Modulus\_BLS381, 365
  - OMASK\_BLS381, 357
  - R2modp\_BLS381, 366
  - TBITS\_BLS381, 357
  - TMASK\_BLS381, 358
- fp\_GOLDILOCKS.h, 366
  - FEXCESS\_GOLDILOCKS, 368
  - FP\_GOLDILOCKS\_add, 368
  - FP\_GOLDILOCKS\_cmove, 369
  - FP\_GOLDILOCKS\_copy, 369
  - FP\_GOLDILOCKS\_cswap, 369
  - FP\_GOLDILOCKS\_div2, 370
  - FP\_GOLDILOCKS\_equals, 370
  - FP\_GOLDILOCKS\_imul, 370
  - FP\_GOLDILOCKS\_inv, 371
  - FP\_GOLDILOCKS\_iszilch, 371
  - FP\_GOLDILOCKS\_mod, 371
  - FP\_GOLDILOCKS\_mul, 371
  - FP\_GOLDILOCKS\_neg, 372
  - FP\_GOLDILOCKS\_norm, 372
  - FP\_GOLDILOCKS\_nres, 372
  - FP\_GOLDILOCKS\_one, 373
  - FP\_GOLDILOCKS\_output, 373
  - FP\_GOLDILOCKS\_pow, 373
  - FP\_GOLDILOCKS\_qr, 373
  - FP\_GOLDILOCKS\_rawoutput, 374
  - FP\_GOLDILOCKS\_rcopy, 374
  - FP\_GOLDILOCKS\_redc, 374
  - FP\_GOLDILOCKS\_reduce, 375



- FP\_GOLDILOCKS\_sqr, [375](#)
- FP\_GOLDILOCKS\_sqrt, [375](#)
- FP\_GOLDILOCKS\_sub, [375](#)
- FP\_GOLDILOCKS\_zero, [376](#)
- MConst\_GOLDILOCKS, [376](#)
- MODBITS\_GOLDILOCKS, [368](#)
- Modulus\_GOLDILOCKS, [376](#)
- OMASK\_GOLDILOCKS, [368](#)
- R2modp\_GOLDILOCKS, [376](#)
- TBITS\_GOLDILOCKS, [368](#)
- TMASK\_GOLDILOCKS, [368](#)
- fp\_NIST256.h, [376](#)
  - FEXCESS\_NIST256, [378](#)
  - FP\_NIST256\_add, [379](#)
  - FP\_NIST256\_cmove, [379](#)
  - FP\_NIST256\_copy, [380](#)
  - FP\_NIST256\_cswap, [380](#)
  - FP\_NIST256\_div2, [380](#)
  - FP\_NIST256\_equals, [381](#)
  - FP\_NIST256\_imul, [381](#)
  - FP\_NIST256\_inv, [381](#)
  - FP\_NIST256\_iszilch, [381](#)
  - FP\_NIST256\_mod, [382](#)
  - FP\_NIST256\_mul, [382](#)
  - FP\_NIST256\_neg, [382](#)
  - FP\_NIST256\_norm, [383](#)
  - FP\_NIST256\_nres, [383](#)
  - FP\_NIST256\_one, [383](#)
  - FP\_NIST256\_output, [383](#)
  - FP\_NIST256\_pow, [384](#)
  - FP\_NIST256\_qr, [384](#)
  - FP\_NIST256\_rawoutput, [384](#)
  - FP\_NIST256\_rcopy, [385](#)
  - FP\_NIST256\_redc, [385](#)
  - FP\_NIST256\_reduce, [385](#)
  - FP\_NIST256\_sqr, [385](#)
  - FP\_NIST256\_sqrt, [386](#)
  - FP\_NIST256\_sub, [386](#)
  - FP\_NIST256\_zero, [386](#)
  - MConst\_NIST256, [387](#)
  - MODBITS\_NIST256, [378](#)
  - Modulus\_NIST256, [387](#)
  - OMASK\_NIST256, [378](#)
  - R2modp\_NIST256, [387](#)
  - TBITS\_NIST256, [379](#)
  - TMASK\_NIST256, [379](#)
- Fra\_25519
  - ecp\_ED25519.h, [252](#)
- Fra\_BLS381
  - ecp2\_BLS381.h, [221](#)
  - ecp\_BLS381.h, [237](#)
  - fp12\_BLS381.h, [320](#)
- Fra\_GOLDILOCKS
  - ecp\_GOLDILOCKS.h, [267](#)
- Fra\_NIST256
  - ecp\_NIST256.h, [282](#)
- Frb\_25519
  - ecp\_ED25519.h, [252](#)
- Frb\_BLS381
  - ecp2\_BLS381.h, [221](#)
  - ecp\_BLS381.h, [237](#)
  - fp12\_BLS381.h, [320](#)
- Frb\_GOLDILOCKS
  - ecp\_GOLDILOCKS.h, [267](#)
- Frb\_NIST256
  - ecp\_NIST256.h, [283](#)
- g
  - FP\_25519, [24](#)
  - FP\_BLS381, [25](#)
  - FP\_GOLDILOCKS, [25](#)
  - FP\_NIST256, [26](#)
- GET\_TIME
  - pbcsupport.h, [412](#)
- gcm, [27](#)
  - a, [27](#)
  - lenA, [27](#)
  - lenC, [27](#)
  - stateX, [27](#)
  - status, [27](#)
  - table, [28](#)
  - Y\_0, [28](#)
- generateOTP
  - utils.c, [424](#)
  - utils.h, [426](#)
- generateRandom
  - utils.c, [424](#)
  - utils.h, [427](#)
- h
  - hash256, [28](#)
  - hash512, [29](#)
- HASH\_ALL
  - pbcsupport.h, [412](#)
- HASH\_ID
  - pbcsupport.h, [412](#)
- HASH\_TYPE\_RSA\_2048
  - rsa\_2048.h, [415](#)
- HASH\_TYPE\_RSA\_3072
  - rsa\_3072.h, [418](#)
- HASH
  - ecdh\_support.h, [209](#)
- HBITS\_1024\_58
  - big\_1024\_58.h, [43](#)
- HBITS\_256\_56
  - big\_256\_56.h, [71](#)
- HBITS\_384\_56
  - big\_384\_56.h, [98](#)
- HBITS\_384\_58
  - big\_384\_58.h, [126](#)
- HBITS\_448\_58
  - big\_448\_58.h, [154](#)
- HFLEN\_2048
  - ff\_2048.h, [285](#)
- HFLEN\_3072
  - ff\_3072.h, [299](#)
- HMASK\_1024\_58

- big\_1024\_58.h, [43](#)
- HMASK\_256\_56
  - big\_256\_56.h, [71](#)
- HMASK\_384\_56
  - big\_384\_56.h, [99](#)
- HMASK\_384\_58
  - big\_384\_58.h, [127](#)
- HMASK\_448\_58
  - big\_448\_58.h, [155](#)
- HMAC
  - ecdh\_support.h, [210](#)
- hash
  - pktype, [31](#)
- hash256, [28](#)
  - h, [28](#)
  - hlen, [28](#)
  - length, [29](#)
  - w, [29](#)
- hash512, [29](#)
  - h, [29](#)
  - hlen, [29](#)
  - length, [30](#)
  - w, [30](#)
- hlen
  - hash256, [28](#)
  - hash512, [29](#)
- ira
  - csprng, [17](#)
- KDF2
  - ecdh\_support.h, [210](#)
- KILL\_CSPRNG
  - randapi.h, [414](#)
- len
  - octet, [30](#)
  - sha3, [36](#)
- lenA
  - gcm, [27](#)
- lenC
  - gcm, [27](#)
- length
  - hash256, [29](#)
  - hash512, [30](#)
  - sha3, [36](#)
- M\_SIZE\_BLS381
  - mpin\_BLS381.h, [389](#)
- MAX\_RSA\_BYTES
  - rsa\_support.h, [421](#)
- MAXPIN
  - mpin\_BLS381.h, [389](#)
- MConst\_25519
  - fp\_25519.h, [355](#)
- MConst\_BLS381
  - fp\_BLS381.h, [365](#)
- MConst\_GOLDILOCKS
  - fp\_GOLDILOCKS.h, [376](#)
- MConst\_NIST256
  - fp\_NIST256.h, [387](#)
- MESSAGE\_SIZE
  - mpin\_BLS381.h, [389](#)
- MODBITS\_25519
  - fp\_25519.h, [346](#)
- MODBITS\_BLS381
  - fp\_BLS381.h, [357](#)
- MODBITS\_GOLDILOCKS
  - fp\_GOLDILOCKS.h, [368](#)
- MODBITS\_NIST256
  - fp\_NIST256.h, [378](#)
- MODBYTES\_1024\_58
  - config\_big\_1024\_58.h, [183](#)
- MODBYTES\_256\_56
  - config\_big\_256\_56.h, [183](#)
- MODBYTES\_384\_56
  - config\_big\_384\_56.h, [184](#)
- MODBYTES\_384\_58
  - config\_big\_384\_58.h, [185](#)
- MODBYTES\_448\_58
  - config\_big\_448\_58.h, [186](#)
- MPIN\_BAD\_PIN
  - mpin\_BLS381.h, [389](#)
- MPIN\_BLS381\_CLIENT\_1
  - mpin\_BLS381.h, [391](#)
- MPIN\_BLS381\_CLIENT\_2
  - mpin\_BLS381.h, [392](#)
- MPIN\_BLS381\_CLIENT\_KEY
  - mpin\_BLS381.h, [392](#)
- MPIN\_BLS381\_CLIENT
  - mpin\_BLS381.h, [390](#)
- MPIN\_BLS381\_DECODING
  - mpin\_BLS381.h, [393](#)
- MPIN\_BLS381\_ENCODING
  - mpin\_BLS381.h, [393](#)
- MPIN\_BLS381\_EXTRACT\_FACTOR
  - mpin\_BLS381.h, [393](#)
- MPIN\_BLS381\_EXTRACT\_PIN
  - mpin\_BLS381.h, [394](#)
- MPIN\_BLS381\_GET\_CLIENT\_PERMIT
  - mpin\_BLS381.h, [394](#)
- MPIN\_BLS381\_GET\_CLIENT\_SECRET
  - mpin\_BLS381.h, [396](#)
- MPIN\_BLS381\_GET\_DVS\_KEYPAIR
  - mpin\_BLS381.h, [396](#)
- MPIN\_BLS381\_GET\_G1\_MULTIPLE
  - mpin\_BLS381.h, [396](#)
- MPIN\_BLS381\_GET\_G2\_MULTIPLE
  - mpin\_BLS381.h, [397](#)
- MPIN\_BLS381\_GET\_SERVER\_SECRET
  - mpin\_BLS381.h, [397](#)
- MPIN\_BLS381\_GET\_Y
  - mpin\_BLS381.h, [398](#)
- MPIN\_BLS381\_KANGAROO
  - mpin\_BLS381.h, [398](#)
- MPIN\_BLS381\_PRECOMPUTE
  - mpin\_BLS381.h, [398](#)

- MPIN\_BLS381\_RANDOM\_GENERATE
  - mpin\_BLS381.h, [399](#)
- MPIN\_BLS381\_RECOMBINE\_G1
  - mpin\_BLS381.h, [399](#)
- MPIN\_BLS381\_RECOMBINE\_G2
  - mpin\_BLS381.h, [400](#)
- MPIN\_BLS381\_RESTORE\_FACTOR
  - mpin\_BLS381.h, [400](#)
- MPIN\_BLS381\_SERVER\_1
  - mpin\_BLS381.h, [401](#)
- MPIN\_BLS381\_SERVER\_2
  - mpin\_BLS381.h, [402](#)
- MPIN\_BLS381\_SERVER\_KEY
  - mpin\_BLS381.h, [403](#)
- MPIN\_BLS381\_SERVER
  - mpin\_BLS381.h, [400](#)
- MPIN\_INVALID\_POINT
  - mpin\_BLS381.h, [389](#)
- MPIN\_OK
  - mpin\_BLS381.h, [389](#)
- MPIN\_PAS
  - mpin\_BLS381.h, [390](#)
- max
  - octet, [30](#)
- mhashit
  - pbc\_support.h, [413](#)
- mode
  - amcl\_aes, [15](#)
- Modulus\_25519
  - fp\_25519.h, [355](#)
- Modulus\_BLS381
  - fp\_BLS381.h, [365](#)
- Modulus\_GOLDILOCKS
  - fp\_GOLDILOCKS.h, [376](#)
- Modulus\_NIST256
  - fp\_NIST256.h, [387](#)
- mpin\_BLS381.h, [387](#)
  - M\_SIZE\_BLS381, [389](#)
  - MAXPIN, [389](#)
  - MESSAGE\_SIZE, [389](#)
  - MPIN\_BAD\_PIN, [389](#)
  - MPIN\_BLS381\_CLIENT\_1, [391](#)
  - MPIN\_BLS381\_CLIENT\_2, [392](#)
  - MPIN\_BLS381\_CLIENT\_KEY, [392](#)
  - MPIN\_BLS381\_CLIENT, [390](#)
  - MPIN\_BLS381\_DECODING, [393](#)
  - MPIN\_BLS381\_ENCODING, [393](#)
  - MPIN\_BLS381\_EXTRACT\_FACTOR, [393](#)
  - MPIN\_BLS381\_EXTRACT\_PIN, [394](#)
  - MPIN\_BLS381\_GET\_CLIENT\_PERMIT, [394](#)
  - MPIN\_BLS381\_GET\_CLIENT\_SECRET, [396](#)
  - MPIN\_BLS381\_GET\_DVS\_KEYPAIR, [396](#)
  - MPIN\_BLS381\_GET\_G1\_MULTIPLE, [396](#)
  - MPIN\_BLS381\_GET\_G2\_MULTIPLE, [397](#)
  - MPIN\_BLS381\_GET\_SERVER\_SECRET, [397](#)
  - MPIN\_BLS381\_GET\_Y, [398](#)
  - MPIN\_BLS381\_KANGAROO, [398](#)
  - MPIN\_BLS381\_PRECOMPUTE, [398](#)
  - MPIN\_BLS381\_RANDOM\_GENERATE, [399](#)
  - MPIN\_BLS381\_RECOMBINE\_G1, [399](#)
  - MPIN\_BLS381\_RECOMBINE\_G2, [400](#)
  - MPIN\_BLS381\_RESTORE\_FACTOR, [400](#)
  - MPIN\_BLS381\_SERVER\_1, [401](#)
  - MPIN\_BLS381\_SERVER\_2, [402](#)
  - MPIN\_BLS381\_SERVER\_KEY, [403](#)
  - MPIN\_BLS381\_SERVER, [400](#)
  - MPIN\_INVALID\_POINT, [389](#)
  - MPIN\_OK, [389](#)
  - MPIN\_PAS, [390](#)
  - PBLEN, [390](#)
  - PFS\_BLS381, [390](#)
  - PGS\_BLS381, [390](#)
- n
  - rsa\_public\_key\_2048, [34](#)
  - rsa\_public\_key\_3072, [35](#)
- NEXCESS\_1024\_58
  - big\_1024\_58.h, [43](#)
- NEXCESS\_256\_56
  - big\_256\_56.h, [71](#)
- NEXCESS\_384\_56
  - big\_384\_56.h, [99](#)
- NEXCESS\_384\_58
  - big\_384\_58.h, [127](#)
- NEXCESS\_448\_58
  - big\_448\_58.h, [155](#)
- NLEN\_1024\_58
  - big\_1024\_58.h, [43](#)
- NLEN\_256\_56
  - big\_256\_56.h, [72](#)
- NLEN\_384\_56
  - big\_384\_56.h, [99](#)
- NLEN\_384\_58
  - big\_384\_58.h, [127](#)
- NLEN\_448\_58
  - big\_448\_58.h, [155](#)
- Nk
  - amcl\_aes, [16](#)
- Nr
  - amcl\_aes, [16](#)
- OAEP\_DECODE
  - rsa\_support.h, [421](#)
- OAEP\_ENCODE
  - rsa\_support.h, [421](#)
- OMASK\_25519
  - fp\_25519.h, [347](#)
- OMASK\_BLS381
  - fp\_BLS381.h, [357](#)
- OMASK\_GOLDILOCKS
  - fp\_GOLDILOCKS.h, [368](#)
- OMASK\_NIST256
  - fp\_NIST256.h, [378](#)
- octet, [30](#)
  - len, [30](#)
  - max, [30](#)
  - val, [31](#)

## p

- rsa\_private\_key\_2048, [32](#)
- rsa\_private\_key\_3072, [34](#)
- P\_EXCESS\_2048
  - ff\_2048.h, [285](#)
- P\_EXCESS\_3072
  - ff\_3072.h, [299](#)
- P\_FEXCESS\_2048
  - ff\_2048.h, [285](#)
- P\_FEXCESS\_3072
  - ff\_3072.h, [299](#)
- P\_MBITS\_2048
  - ff\_2048.h, [285](#)
- P\_MBITS\_3072
  - ff\_3072.h, [299](#)
- P\_TBITS\_2048
  - ff\_2048.h, [285](#)
- P\_TBITS\_3072
  - ff\_3072.h, [299](#)
- PAIR\_BLS381\_G1mul
  - pair\_BLS381.h, [407](#)
- PAIR\_BLS381\_G2mul
  - pair\_BLS381.h, [407](#)
- PAIR\_BLS381\_GTmember
  - pair\_BLS381.h, [407](#)
- PAIR\_BLS381\_GTpows
  - pair\_BLS381.h, [408](#)
- PAIR\_BLS381\_another
  - pair\_BLS381.h, [404](#)
- PAIR\_BLS381\_ate
  - pair\_BLS381.h, [405](#)
- PAIR\_BLS381\_double\_ate
  - pair\_BLS381.h, [405](#)
- PAIR\_BLS381\_fexp
  - pair\_BLS381.h, [405](#)
- PAIR\_BLS381\_initmp
  - pair\_BLS381.h, [408](#)
- PAIR\_BLS381\_miller
  - pair\_BLS381.h, [408](#)
- PAIR\_BLS381\_nbits
  - pair\_BLS381.h, [408](#)
- PBKDF2
  - ecdh\_support.h, [210](#)
- PBLN
  - mpin\_BLS381.h, [390](#)
- PFS\_BLS381
  - mpin\_BLS381.h, [390](#)
- PGS\_BLS381
  - mpin\_BLS381.h, [390](#)
- PIV
  - wcc\_BLS381.h, [429](#)
- PKCS15
  - rsa\_support.h, [422](#)
- PTAG
  - wcc\_BLS381.h, [429](#)
- pair\_BLS381.h, [403](#)
  - CURVE\_BB\_BLS381, [409](#)
  - CURVE\_Bnx\_BLS381, [409](#)

- CURVE\_Cru\_BLS381, [409](#)
- CURVE\_SB\_BLS381, [409](#)
- CURVE\_W\_BLS381, [409](#)
- CURVE\_WB\_BLS381, [409](#)
- PAIR\_BLS381\_G1mul, [407](#)
- PAIR\_BLS381\_G2mul, [407](#)
- PAIR\_BLS381\_GTmember, [407](#)
- PAIR\_BLS381\_GTpows, [408](#)
- PAIR\_BLS381\_another, [404](#)
- PAIR\_BLS381\_ate, [405](#)
- PAIR\_BLS381\_double\_ate, [405](#)
- PAIR\_BLS381\_fexp, [405](#)
- PAIR\_BLS381\_initmp, [408](#)
- PAIR\_BLS381\_miller, [408](#)
- PAIR\_BLS381\_nbits, [408](#)
- pbcsupport.h, [410](#)
  - AES\_GCM\_DECRYPT, [411](#)
  - AES\_GCM\_ENCRYPT, [411](#)
  - GET\_TIME, [412](#)
  - HASH\_ALL, [412](#)
  - HASH\_ID, [412](#)
  - mhashit, [413](#)
  - TIME\_SLOT\_MINUTES, [410](#)
  - today, [413](#)
- pktype, [31](#)
  - curve, [31](#)
  - hash, [31](#)
  - type, [31](#)
- pool
  - csprng, [17](#)
- pool\_ptr
  - csprng, [17](#)
- q
  - rsa\_private\_key\_2048, [33](#)
  - rsa\_private\_key\_3072, [34](#)
- R2modp\_25519
  - fp\_25519.h, [355](#)
- R2modp\_BLS381
  - fp\_BLS381.h, [366](#)
- R2modp\_GOLDILOCKS
  - fp\_GOLDILOCKS.h, [376](#)
- R2modp\_NIST256
  - fp\_NIST256.h, [387](#)
- RFS\_2048
  - rsa\_2048.h, [415](#)
- RFS\_3072
  - rsa\_3072.h, [418](#)
- RSA\_2048\_DECRYPT
  - rsa\_2048.h, [415](#)
- RSA\_2048\_ENCRYPT
  - rsa\_2048.h, [416](#)
- RSA\_2048\_KEY\_PAIR
  - rsa\_2048.h, [416](#)
- RSA\_2048\_PRIVATE\_KEY\_KILL
  - rsa\_2048.h, [417](#)
- RSA\_2048\_fromOctet
  - rsa\_2048.h, [416](#)

RSA\_3072\_DECRYPT  
  rsa\_3072.h, 418  
RSA\_3072\_ENCRYPT  
  rsa\_3072.h, 419  
RSA\_3072\_KEY\_PAIR  
  rsa\_3072.h, 419  
RSA\_3072\_PRIVATE\_KEY\_KILL  
  rsa\_3072.h, 420  
RSA\_3072\_fromOctet  
  rsa\_3072.h, 419  
randapi.h, 413  
  CREATE\_CSPRNG, 414  
  KILL\_CSPRNG, 414  
rate  
  sha3, 36  
rkey  
  amcl\_aes, 16  
rndptr  
  csprng, 17  
rsa\_2048.h, 414  
  HASH\_TYPE\_RSA\_2048, 415  
  RFS\_2048, 415  
  RSA\_2048\_DECRYPT, 415  
  RSA\_2048\_ENCRYPT, 416  
  RSA\_2048\_KEY\_PAIR, 416  
  RSA\_2048\_PRIVATE\_KEY\_KILL, 417  
  RSA\_2048\_fromOctet, 416  
rsa\_3072.h, 417  
  HASH\_TYPE\_RSA\_3072, 418  
  RFS\_3072, 418  
  RSA\_3072\_DECRYPT, 418  
  RSA\_3072\_ENCRYPT, 419  
  RSA\_3072\_KEY\_PAIR, 419  
  RSA\_3072\_PRIVATE\_KEY\_KILL, 420  
  RSA\_3072\_fromOctet, 419  
rsa\_private\_key\_2048, 32  
  c, 32  
  dp, 32  
  dq, 32  
  p, 32  
  q, 33  
rsa\_private\_key\_3072, 33  
  c, 33  
  dp, 33  
  dq, 33  
  p, 34  
  q, 34  
rsa\_public\_key\_2048, 34  
  e, 34  
  n, 34  
rsa\_public\_key\_3072, 35  
  e, 35  
  n, 35  
rsa\_support.h, 420  
  MAX\_RSA\_BYTES, 421  
  OAEP\_DECODE, 421  
  OAEP\_ENCODE, 421  
  PKCS15, 422  
S  
  sha3, 36  
sha3, 35  
  len, 36  
  length, 36  
  rate, 36  
  S, 36  
sign16  
  arch.h, 38  
sign32  
  arch.h, 38  
sign64  
  arch.h, 38  
sign8  
  arch.h, 38  
stateX  
  gcm, 27  
status  
  gcm, 27  
TBITS\_25519  
  fp\_25519.h, 347  
TBITS\_BLS381  
  fp\_BLS381.h, 357  
TBITS\_GOLDILOCKS  
  fp\_GOLDILOCKS.h, 368  
TBITS\_NIST256  
  fp\_NIST256.h, 379  
TIME\_SLOT\_MINUTES  
  pbc\_support.h, 410  
  wcc\_BLS381.h, 429  
TMASK\_25519  
  fp\_25519.h, 347  
TMASK\_BLS381  
  fp\_BLS381.h, 358  
TMASK\_GOLDILOCKS  
  fp\_GOLDILOCKS.h, 368  
TMASK\_NIST256  
  fp\_NIST256.h, 379  
table  
  gcm, 28  
today  
  pbc\_support.h, 413  
type  
  FP12\_BLS381, 22  
  pktype, 31  
uchar  
  arch.h, 38  
unsign32  
  arch.h, 39  
unsign64  
  arch.h, 39  
utils.c, 422  
  amcl\_bin2hex, 423  
  amcl\_hex2bin, 423  
  amcl\_print\_hex, 424  
  generateOTP, 424  
  generateRandom, 424

- utils.h, [425](#)
  - amcl\_bin2hex, [425](#)
  - amcl\_hex2bin, [426](#)
  - amcl\_print\_hex, [426](#)
  - generateOTP, [426](#)
  - generateRandom, [427](#)
- val
  - octet, [31](#)
- version.c, [427](#)
  - amcl\_version, [428](#)
- w
  - hash256, [29](#)
  - hash512, [30](#)
- WCC\_BLS381\_GET\_G1\_MULTIPLE
  - wcc\_BLS381.h, [430](#)
- WCC\_BLS381\_GET\_G2\_MULTIPLE
  - wcc\_BLS381.h, [431](#)
- WCC\_BLS381\_Hq
  - wcc\_BLS381.h, [431](#)
- WCC\_BLS381\_RANDOM\_GENERATE
  - wcc\_BLS381.h, [432](#)
- WCC\_BLS381\_RECEIVER\_KEY
  - wcc\_BLS381.h, [432](#)
- WCC\_BLS381\_RECOMBINE\_G1
  - wcc\_BLS381.h, [433](#)
- WCC\_BLS381\_RECOMBINE\_G2
  - wcc\_BLS381.h, [433](#)
- WCC\_BLS381\_SENDER\_KEY
  - wcc\_BLS381.h, [433](#)
- WCC\_INVALID\_POINT
  - wcc\_BLS381.h, [429](#)
- WCC\_OK
  - wcc\_BLS381.h, [430](#)
- WCC\_PFS\_BLS381
  - wcc\_BLS381.h, [430](#)
- WCC\_PGS\_BLS381
  - wcc\_BLS381.h, [430](#)
- wcc\_BLS381.h, [428](#)
  - PIV, [429](#)
  - PTAG, [429](#)
  - TIME\_SLOT\_MINUTES, [429](#)
  - WCC\_BLS381\_GET\_G1\_MULTIPLE, [430](#)
  - WCC\_BLS381\_GET\_G2\_MULTIPLE, [431](#)
  - WCC\_BLS381\_Hq, [431](#)
  - WCC\_BLS381\_RANDOM\_GENERATE, [432](#)
  - WCC\_BLS381\_RECEIVER\_KEY, [432](#)
  - WCC\_BLS381\_RECOMBINE\_G1, [433](#)
  - WCC\_BLS381\_RECOMBINE\_G2, [433](#)
  - WCC\_BLS381\_SENDER\_KEY, [433](#)
  - WCC\_INVALID\_POINT, [429](#)
  - WCC\_OK, [430](#)
  - WCC\_PFS\_BLS381, [430](#)
  - WCC\_PGS\_BLS381, [430](#)
- x
  - ECP2\_BLS381, [18](#)
  - ECP\_BLS381, [18](#)
- ECP\_ED25519, [19](#)
- ECP\_GOLDILOCKS, [20](#)
- ECP\_NIST256, [21](#)
- x509.h, [434](#)
  - X509\_extract\_cert, [435](#)
  - X509\_extract\_cert\_sig, [435](#)
  - X509\_extract\_public\_key, [436](#)
  - X509\_find\_entity\_property, [436](#)
  - X509\_find\_expiry\_date, [436](#)
  - X509\_find\_issuer, [437](#)
  - X509\_find\_start\_date, [437](#)
  - X509\_find\_subject, [437](#)
  - X509\_find\_validity, [438](#)
- X509\_extract\_cert
  - x509.h, [435](#)
- X509\_extract\_cert\_sig
  - x509.h, [435](#)
- X509\_extract\_public\_key
  - x509.h, [436](#)
- X509\_find\_entity\_property
  - x509.h, [436](#)
- X509\_find\_expiry\_date
  - x509.h, [436](#)
- X509\_find\_issuer
  - x509.h, [437](#)
- X509\_find\_start\_date
  - x509.h, [437](#)
- X509\_find\_subject
  - x509.h, [437](#)
- X509\_find\_validity
  - x509.h, [438](#)
- XES
  - FP\_25519, [24](#)
  - FP\_BLS381, [25](#)
  - FP\_GOLDILOCKS, [26](#)
  - FP\_NIST256, [26](#)
- y
  - ECP2\_BLS381, [18](#)
  - ECP\_BLS381, [19](#)
  - ECP\_ED25519, [19](#)
  - ECP\_GOLDILOCKS, [20](#)
  - ECP\_NIST256, [21](#)
- Y\_0
  - gcm, [28](#)
- z
  - ECP2\_BLS381, [18](#)
  - ECP\_BLS381, [19](#)
  - ECP\_ED25519, [20](#)
  - ECP\_GOLDILOCKS, [20](#)
  - ECP\_NIST256, [21](#)