

SEcube

Generated by Doxygen 1.8.11

Contents

1	Module Index	1
1.1	Modules	1
2	Data Structure Index	3
2.1	Data Structures	3
3	File Index	5
3.1	File List	5
4	Module Documentation	7
4.1	AES return values	7
4.1.1	Detailed Description	7
4.2	AES Key, IV, Block Sizes	8
4.2.1	Detailed Description	8
4.2.2	Macro Definition Documentation	8
4.2.2.1	B5_AES_128	8
4.2.2.2	B5_AES_192	8
4.2.2.3	B5_AES_256	8
4.2.2.4	B5_AES_BLK_SIZE	8
4.2.2.5	B5_AES_IV_SIZE	8
4.3	AES modes	9
4.3.1	Detailed Description	9
4.3.2	Macro Definition Documentation	9
4.3.2.1	B5_AES256_CBC_DEC	9
4.3.2.2	B5_AES256_CBC_ENC	9

4.3.2.3	B5_AES256_CFB_DEC	9
4.3.2.4	B5_AES256_CFB_ENC	9
4.3.2.5	B5_AES256_CTR	9
4.3.2.6	B5_AES256_ECB_DEC	9
4.3.2.7	B5_AES256_ECB_ENC	9
4.3.2.8	B5_AES256_OFB	9
4.4	AES data structures	10
4.4.1	Detailed Description	10
4.5	AES functions	11
4.5.1	Detailed Description	11
4.5.2	Function Documentation	11
4.5.2.1	B5_Aes256_Finit(B5_tAesCtx *ctx)	11
4.5.2.2	B5_Aes256_Init(B5_tAesCtx *ctx, const uint8_t *Key, int16_t keySize, uint8_t ← t aesMode)	11
4.5.2.3	B5_Aes256_SetIV(B5_tAesCtx *ctx, const uint8_t *IV)	12
4.5.2.4	B5_Aes256_Update(B5_tAesCtx *ctx, uint8_t *encData, uint8_t *clrData, int16_t nBlk)	13
4.6	CMAC-AES Key, Blk Sizes	14
4.6.1	Detailed Description	14
4.6.2	Macro Definition Documentation	14
4.6.2.1	B5_CMAC_AES_128	14
4.6.2.2	B5_CMAC_AES_192	14
4.6.2.3	B5_CMAC_AES_256	14
4.6.2.4	B5_CMAC_AES_BLK_SIZE	14
4.7	CMAC-AES return values	15
4.7.1	Detailed Description	15
4.8	CMAC-AES data structures	16
4.8.1	Detailed Description	16
4.9	CMAC-AES functions	17
4.9.1	Detailed Description	17
4.9.2	Function Documentation	17

4.9.2.1	B5_CmacAes256_Finit(B5_tCmacAesCtx *ctx, uint8_t *rSignature)	17
4.9.2.2	B5_CmacAes256_Init(B5_tCmacAesCtx *ctx, const uint8_t *Key, int16_t keySize)	17
4.9.2.3	B5_CmacAes256_Reset(B5_tCmacAesCtx *ctx)	18
4.9.2.4	B5_CmacAes256_Sign(const uint8_t *data, int32_t dataLen, const uint8_t *Key, int16_t keySize, uint8_t *rSignature)	18
4.9.2.5	B5_CmacAes256_Update(B5_tCmacAesCtx *ctx, const uint8_t *data, int32_t dataLen)	18
4.10	AccessLogin	19
4.10.1	Detailed Description	19
4.11	KeyOpEdit	20
4.11.1	Detailed Description	20
4.11.2	Enumeration Type Documentation	20
4.11.2.1	anonymous enum	20
4.12	AlgorithmAvail	21
4.12.1	Detailed Description	21
4.12.2	Enumeration Type Documentation	21
4.12.2.1	anonymous enum	21
4.13	SHA256 return values	22
4.13.1	Detailed Description	22
4.14	SHA256 digest and block sizes	23
4.14.1	Detailed Description	23
4.15	SHA256 data structures	24
4.15.1	Detailed Description	24
4.16	SHA256 functions	25
4.16.1	Detailed Description	25
4.16.2	Function Documentation	25
4.16.2.1	B5_Sha256_Finit(B5_tSha256Ctx *ctx, uint8_t *rDigest)	25
4.16.2.2	B5_Sha256_Init(B5_tSha256Ctx *ctx)	25
4.16.2.3	B5_Sha256_Update(B5_tSha256Ctx *ctx, const uint8_t *data, int32_t dataLen)	25
4.17	HMAC-SHA256 return values	27
4.17.1	Detailed Description	27
4.18	HMAC-SHA256 data structures	28
4.18.1	Detailed Description	28
4.19	HMAC-SHA256 functions	29
4.19.1	Detailed Description	29
4.19.2	Function Documentation	29
4.19.2.1	B5_HmacSha256_Finit(B5_tHmacSha256Ctx *ctx, uint8_t *rDigest)	29
4.19.2.2	B5_HmacSha256_Init(B5_tHmacSha256Ctx *ctx, const uint8_t *Key, int16_t keySize)	29
4.19.2.3	B5_HmacSha256_Update(B5_tHmacSha256Ctx *ctx, const uint8_t *data, int32_t dataLen)	29

5	Data Structure Documentation	31
5.1	AesHmacSha256s_ctx Struct Reference	31
5.2	B5_tAesCtx Struct Reference	31
5.2.1	Field Documentation	32
5.2.1.1	InitVector	32
5.2.1.2	mode	32
5.2.1.3	Nr	32
5.2.1.4	rk	32
5.3	B5_tCmacAesCtx Struct Reference	32
5.4	B5_tHmacSha256Ctx Struct Reference	32
5.5	B5_tSha256Ctx Struct Reference	33
5.6	s3_storage_range_ Struct Reference	33
5.6.1	Detailed Description	33
5.7	se3_algo_descriptor_ Struct Reference	33
5.7.1	Detailed Description	34
5.8	SE3_COMM_STATUS_ Struct Reference	34
5.8.1	Detailed Description	35
5.9	SE3_FLASH_INFO_ Struct Reference	35
5.9.1	Detailed Description	35
5.10	se3_flash_it_ Struct Reference	36
5.10.1	Detailed Description	36
5.11	se3_flash_key_ Struct Reference	36
5.11.1	Detailed Description	36
5.12	SE3_L0_GLOBALS_ Struct Reference	37
5.12.1	Detailed Description	37
5.13	SE3_L1_GLOBALS_ Struct Reference	37
5.13.1	Detailed Description	38
5.14	SE3_LOGIN_STATUS_ Struct Reference	38
5.14.1	Detailed Description	38
5.15	se3_mem_ Struct Reference	38
5.15.1	Detailed Description	39
5.16	se3_payload_cryptoctx_ Struct Reference	39
5.17	SE3_RECORD_INFO_ Struct Reference	39
5.17.1	Detailed Description	40
5.18	SE3_SERIAL_ Struct Reference	40
5.18.1	Detailed Description	40
5.19	se3c0_req_header_ Struct Reference	40
5.19.1	Detailed Description	41
5.20	se3c0_resp_header_ Struct Reference	41
5.20.1	Detailed Description	41

6	File Documentation	43
6.1	src/Common/crc16.h File Reference	43
6.1.1	Detailed Description	43
6.1.2	Function Documentation	43
6.1.2.1	se3_crc16_update(size_t length, const uint8_t *data, uint16_t crc)	43
6.2	src/Common/se3_common.h File Reference	44
6.2.1	Detailed Description	44
6.2.2	Function Documentation	44
6.2.2.1	se3_nblocks(uint16_t len)	44
6.2.2.2	se3_req_len_data(uint16_t len_data_and_headers)	45
6.2.2.3	se3_req_len_data_and_headers(uint16_t len_data)	45
6.2.2.4	se3_resp_len_data(uint16_t len_data_and_headers)	45
6.2.2.5	se3_resp_len_data_and_headers(uint16_t len_data)	46
6.3	src/Common/se3c1def.h File Reference	46
6.3.1	Detailed Description	48
6.3.2	Enumeration Type Documentation	48
6.3.2.1	anonymous enum	48
6.3.2.2	anonymous enum	48
6.3.2.3	anonymous enum	48
6.3.2.4	anonymous enum	48
6.3.2.5	anonymous enum	48
6.3.2.6	anonymous enum	49
6.3.2.7	anonymous enum	49
6.3.2.8	anonymous enum	49
6.3.2.9	anonymous enum	49
6.3.2.10	anonymous enum	49
6.3.2.11	anonymous enum	49
6.3.2.12	anonymous enum	49
6.3.2.13	anonymous enum	49
6.3.2.14	anonymous enum	49

6.3.2.15	anonymous enum	49
6.3.2.16	anonymous enum	50
6.3.2.17	anonymous enum	50
6.3.2.18	anonymous enum	50
6.3.2.19	anonymous enum	50
6.3.2.20	anonymous enum	50
6.3.2.21	anonymous enum	50
6.3.2.22	anonymous enum	50
6.3.2.23	anonymous enum	50
6.4	src/Device/se3_algo_Aes.h File Reference	51
6.4.1	Detailed Description	51
6.4.2	Function Documentation	51
6.4.2.1	se3_algo_Aes_init(se3_flash_key *key, uint16_t mode, uint8_t *ctx)	51
6.4.2.2	se3_algo_Aes_update(uint8_t *ctx, uint16_t flags, uint16_t datain1_len, const uint8_t *datain1, uint16_t datain2_len, const uint8_t *datain2, uint16_t t *dataout_len, uint8_t *dataout)	51
6.5	src/Device/se3_algo_AesHmacSha256s.c File Reference	52
6.5.1	Detailed Description	52
6.5.2	Function Documentation	52
6.5.2.1	se3_algo_AesHmacSha256s_init(se3_flash_key *key, uint16_t mode, uint8_t *ctx)	52
6.5.2.2	se3_algo_AesHmacSha256s_update(uint8_t *ctx, uint16_t flags, uint16_t t datain1_len, const uint8_t *datain1, uint16_t datain2_len, const uint8_t *datain2, uint16_t *dataout_len, uint8_t *dataout)	53
6.6	src/Device/se3_algo_AesHmacSha256s.h File Reference	53
6.6.1	Detailed Description	53
6.6.2	Function Documentation	53
6.6.2.1	se3_algo_AesHmacSha256s_init(se3_flash_key *key, uint16_t mode, uint8_t *ctx)	53
6.6.2.2	se3_algo_AesHmacSha256s_update(uint8_t *ctx, uint16_t flags, uint16_t t datain1_len, const uint8_t *datain1, uint16_t datain2_len, const uint8_t *datain2, uint16_t *dataout_len, uint8_t *dataout)	54
6.7	src/Device/se3_cmd.c File Reference	54
6.7.1	Detailed Description	54
6.7.2	Function Documentation	54

6.7.2.1	<code>se3_cmd_execute()</code>	54
6.8	<code>src/Device/se3_cmd.h</code> File Reference	55
6.8.1	Detailed Description	55
6.8.2	Function Documentation	55
6.8.2.1	<code>se3_cmd_execute()</code>	55
6.9	<code>src/Device/se3_cmd0.c</code> File Reference	55
6.9.1	Detailed Description	56
6.9.2	Function Documentation	56
6.9.2.1	<code>L0d_echo(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)</code>	56
6.9.2.2	<code>L0d_factory_init(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)</code>	56
6.10	<code>src/Device/se3_cmd0.h</code> File Reference	56
6.10.1	Detailed Description	56
6.10.2	Function Documentation	57
6.10.2.1	<code>L0d_echo(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)</code>	57
6.10.2.2	<code>L0d_factory_init(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)</code>	57
6.11	<code>src/Device/se3_cmd1.c</code> File Reference	57
6.11.1	Detailed Description	57
6.11.2	Function Documentation	58
6.11.2.1	<code>L0d_cmd1(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)</code>	58
6.11.3	Variable Documentation	58
6.11.3.1	<code>L1d_handlers</code>	58
6.12	<code>src/Device/se3_cmd1.h</code> File Reference	58
6.12.1	Detailed Description	58
6.12.2	Function Documentation	59
6.12.2.1	<code>L0d_cmd1(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)</code>	59
6.13	<code>src/Device/se3_cmd1_config.c</code> File Reference	59
6.13.1	Detailed Description	59
6.13.2	Function Documentation	59
6.13.2.1	<code>L1d_config(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)</code>	59

6.14	src/Device/se3_cmd1_config.h File Reference	59
6.14.1	Detailed Description	60
6.14.2	Function Documentation	60
6.14.2.1	L1d_config(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	60
6.15	src/Device/se3_cmd1_crypto.c File Reference	60
6.15.1	Detailed Description	60
6.15.2	Function Documentation	61
6.15.2.1	L1d_crypto_init(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	61
6.15.2.2	L1d_crypto_list(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	61
6.15.2.3	L1d_crypto_set_time(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	61
6.15.2.4	L1d_crypto_update(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	61
6.16	src/Device/se3_cmd1_crypto.h File Reference	61
6.16.1	Detailed Description	62
6.16.2	Function Documentation	62
6.16.2.1	L1d_crypto_init(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	62
6.16.2.2	L1d_crypto_list(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	62
6.16.2.3	L1d_crypto_set_time(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	62
6.16.2.4	L1d_crypto_update(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	63
6.17	src/Device/se3_cmd1_keys.c File Reference	63
6.17.1	Detailed Description	63
6.17.2	Function Documentation	63
6.17.2.1	L1d_key_edit(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	63
6.17.2.2	L1d_key_list(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	64
6.18	src/Device/se3_cmd1_keys.h File Reference	64

6.18.1 Detailed Description	64
6.18.2 Function Documentation	64
6.18.2.1 L1d_key_edit(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	64
6.18.2.2 L1d_key_list(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	65
6.19 src/Device/se3_cmd1_login.c File Reference	65
6.19.1 Detailed Description	65
6.19.2 Function Documentation	65
6.19.2.1 L1d_challenge(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	65
6.19.2.2 L1d_login(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	66
6.19.2.3 L1d_logout(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	66
6.20 src/Device/se3_cmd1_login.h File Reference	66
6.20.1 Detailed Description	66
6.20.2 Function Documentation	67
6.20.2.1 L1d_challenge(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	67
6.20.2.2 L1d_login(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	67
6.20.2.3 L1d_logout(uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)	67
6.21 src/Device/se3_flash.c File Reference	67
6.21.1 Detailed Description	68
6.21.2 Function Documentation	68
6.21.2.1 se3_flash_canfit(size_t size)	68
6.21.2.2 se3_flash_info_setup(uint32_t sector, const uint8_t *base)	69
6.21.2.3 se3_flash_init()	69
6.21.2.4 se3_flash_it_delete(se3_flash_it *it)	69
6.21.2.5 se3_flash_it_init(se3_flash_it *it)	69
6.21.2.6 se3_flash_it_new(se3_flash_it *it, uint8_t type, uint16_t size)	70
6.21.2.7 se3_flash_it_next(se3_flash_it *it)	70
6.21.2.8 se3_flash_it_write(se3_flash_it *it, uint16_t off, const uint8_t *data, uint16_t size)	70

6.21.2.9	<code>se3_flash_pos_delete(size_t pos)</code>	71
6.21.2.10	<code>se3_flash_unused()</code>	71
6.22	<code>src/Device/se3_flash.h</code> File Reference	71
6.22.1	Detailed Description	72
6.22.2	Enumeration Type Documentation	73
6.22.2.1	anonymous enum	73
6.22.2.2	anonymous enum	73
6.22.3	Function Documentation	73
6.22.3.1	<code>se3_flash_canfit(size_t size)</code>	73
6.22.3.2	<code>se3_flash_info_setup(uint32_t sector, const uint8_t *base)</code>	73
6.22.3.3	<code>se3_flash_init()</code>	74
6.22.3.4	<code>se3_flash_it_delete(se3_flash_it *it)</code>	74
6.22.3.5	<code>se3_flash_it_init(se3_flash_it *it)</code>	74
6.22.3.6	<code>se3_flash_it_new(se3_flash_it *it, uint8_t type, uint16_t size)</code>	74
6.22.3.7	<code>se3_flash_it_next(se3_flash_it *it)</code>	75
6.22.3.8	<code>se3_flash_it_write(se3_flash_it *it, uint16_t off, const uint8_t *data, uint16_t size)</code>	75
6.22.3.9	<code>se3_flash_pos_delete(size_t pos)</code>	75
6.22.3.10	<code>se3_flash_unused()</code>	76
6.23	<code>src/Device/se3_keys.c</code> File Reference	76
6.23.1	Detailed Description	77
6.23.2	Function Documentation	77
6.23.2.1	<code>se3_key_equal(se3_flash_it *it, se3_flash_key *key)</code>	77
6.23.2.2	<code>se3_key_find(uint32_t id, se3_flash_it *it)</code>	77
6.23.2.3	<code>se3_key_new(se3_flash_it *it, se3_flash_key *key)</code>	77
6.23.2.4	<code>se3_key_read(se3_flash_it *it, se3_flash_key *key)</code>	78
6.23.2.5	<code>se3_key_read_data(se3_flash_it *it, uint16_t data_size, uint8_t *data)</code>	78
6.23.2.6	<code>se3_key_remove(se3_flash_it *it)</code>	78
6.23.2.7	<code>se3_key_write(se3_flash_it *it, se3_flash_key *key)</code>	79
6.24	<code>src/Device/se3_keys.h</code> File Reference	79
6.24.1	Detailed Description	80

6.24.2	Typedef Documentation	80
6.24.2.1	se3_flash_key	80
6.24.3	Enumeration Type Documentation	81
6.24.3.1	anonymous enum	81
6.24.4	Function Documentation	81
6.24.4.1	se3_key_equal(se3_flash_it *it, se3_flash_key *key)	81
6.24.4.2	se3_key_find(uint32_t id, se3_flash_it *it)	81
6.24.4.3	se3_key_new(se3_flash_it *it, se3_flash_key *key)	81
6.24.4.4	se3_key_read(se3_flash_it *it, se3_flash_key *key)	82
6.24.4.5	se3_key_read_data(se3_flash_it *it, uint16_t data_size, uint8_t *data)	82
6.24.4.6	se3_key_remove(se3_flash_it *it)	82
6.24.4.7	se3_key_write(se3_flash_it *it, se3_flash_key *key)	83
6.25	src/Device/se3_memory.c File Reference	83
6.25.1	Detailed Description	84
6.25.2	Function Documentation	84
6.25.2.1	se3_mem_alloc(se3_mem *mem, size_t size)	84
6.25.2.2	se3_mem_free(se3_mem *mem, int32_t id)	84
6.25.2.3	se3_mem_init(se3_mem *mem, size_t index_size, uint8_t **index, size_t buf_size, uint8_t *buf)	84
6.25.2.4	se3_mem_ptr(se3_mem *mem, int32_t id)	85
6.25.2.5	se3_mem_reset(se3_mem *mem)	85
6.26	src/Device/se3_memory.h File Reference	85
6.26.1	Detailed Description	86
6.26.2	Enumeration Type Documentation	86
6.26.2.1	anonymous enum	86
6.26.3	Function Documentation	86
6.26.3.1	se3_mem_alloc(se3_mem *mem, size_t size)	86
6.26.3.2	se3_mem_free(se3_mem *mem, int32_t id)	87
6.26.3.3	se3_mem_init(se3_mem *mem, size_t index_size, uint8_t **index, size_t buf_size, uint8_t *buf)	87
6.26.3.4	se3_mem_ptr(se3_mem *mem, int32_t id)	87

6.26.3.5	se3_mem_reset(se3_mem *mem)	87
6.27	src/Device/se3_proto.c File Reference	87
6.27.1	Detailed Description	88
6.27.2	Function Documentation	88
6.27.2.1	block_is_magic(const uint8_t *buf)	88
6.27.2.2	find_magic_index(uint32_t block)	89
6.27.2.3	handle_req_rcv(int index, const uint8_t *blockdata)	89
6.27.2.4	handle_resp_send(int index, uint8_t *blockdata)	89
6.27.2.5	se3_proto_rcv(uint8_t lun, const uint8_t *buf, uint32_t blk_addr, uint16_t blk_len)	90
6.27.2.6	se3_proto_request_reset()	90
6.27.2.7	se3_proto_send(uint8_t lun, uint8_t *buf, uint32_t blk_addr, uint16_t blk_len)	90
6.27.2.8	se3_storage_range_add(s3_storage_range *range, uint8_t lun, uint8_t *buf, uint32_t block, enum s3_storage_range_direction direction)	90
6.28	src/Device/se3_proto.h File Reference	90
6.28.1	Detailed Description	91
6.28.2	Enumeration Type Documentation	91
6.28.2.1	anonymous enum	91
6.28.3	Function Documentation	91
6.28.3.1	se3_proto_rcv(uint8_t lun, const uint8_t *buf, uint32_t blk_addr, uint16_t blk_len)	91
6.28.3.2	se3_proto_send(uint8_t lun, uint8_t *buf, uint32_t blk_addr, uint16_t blk_len)	91
6.29	src/Device/se3c0.c File Reference	92
6.29.1	Detailed Description	92
6.29.2	Variable Documentation	92
6.29.2.1	se3_hello	92
6.30	src/Device/se3c0.h File Reference	92
6.30.1	Detailed Description	94
6.30.2	Typedef Documentation	94
6.30.2.1	se3_cmd_func	94
6.30.2.2	SE3_COMM_STATUS	94
6.31	src/Device/se3c1.c File Reference	94
6.31.1	Detailed Description	95

6.31.2	Function Documentation	95
6.31.2.1	se3c1_login_cleanup()	95
6.31.2.2	se3c1_record_get(uint16_t type, uint8_t *data)	95
6.31.2.3	se3c1_record_set(uint16_t type, const uint8_t *data)	95
6.31.3	Variable Documentation	96
6.31.3.1	L1d_algo_table	96
6.31.3.2	se3_sessions_buf	96
6.31.3.3	se3_sessions_index	96
6.32	src/Device/se3c1.h File Reference	96
6.32.1	Detailed Description	97
6.32.2	Enumeration Type Documentation	98
6.32.2.1	anonymous enum	98
6.32.2.2	anonymous enum	98
6.32.2.3	anonymous enum	98
6.32.3	Function Documentation	98
6.32.3.1	se3c1_login_cleanup()	98
6.32.3.2	se3c1_record_get(uint16_t type, uint8_t *data)	98
6.32.3.3	se3c1_record_set(uint16_t type, const uint8_t *data)	99
6.32.4	Variable Documentation	99
6.32.4.1	L1d_algo_table	99
6.32.4.2	se3_sessions_buf	99
6.32.4.3	se3_sessions_index	99

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

AES return values	7
AES Key, IV, Block Sizes	8
AES modes	9
AES data structures	10
AES functions	11
CMAC-AES Key, Blk Sizes	14
CMAC-AES return values	15
CMAC-AES data structures	16
CMAC-AES functions	17
AccessLogin	19
KeyOpEdit	20
AlgorithmAvail	21
SHA256 return values	22
SHA256 digest and block sizes	23
SHA256 data structures	24
SHA256 functions	25
HMAC-SHA256 return values	27
HMAC-SHA256 data structures	28
HMAC-SHA256 functions	29

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

AesHmacSha256s_ctx	31
B5_tAesCtx	31
B5_tCmacAesCtx	32
B5_tHmacSha256Ctx	32
B5_tSha256Ctx	33
s3_storage_range_ SDIO read/write request buffer context	33
se3_algo_descriptor_ Algorithm descriptor type	33
SE3_COMM_STATUS_ Structure holding host-device communication status and buffers	34
SE3_FLASH_INFO_ Flash management structure	35
se3_flash_it_ Flash node iterator structure	36
se3_flash_key_ Flash key structure	36
SE3_L0_GLOBALS_ L0 globals structure	37
SE3_L1_GLOBALS_ L1 globals structure	37
SE3_LOGIN_STATUS_ L1 login status data	38
se3_mem_ Memory allocator structure	38
se3_payload_cryptoctx_	39
SE3_RECORD_INFO_ Record information	39
SE3_SERIAL_ Serial number data and state	40
se3c0_req_header_ Decoded request header	40
se3c0_resp_header_ Response header to be encoded	41

Chapter 3

File Index

3.1 File List

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

src/Common/ aes256.h	??
src/Common/ crc16.h	
This file contains defines and functions for computing CRC	43
src/Common/ pbkdf2.h	??
src/Common/ se3_common.h	
This file contains defines and functions common for L0 and L1	44
src/Common/ se3c0def.h	??
src/Common/ se3c1def.h	
This file contains defines to be used both for L1 and L0 functions	46
src/Common/ sha256.h	??
src/Device/ se3_algo_Aes.h	
SE3_ALGO_AES crypto handlers	51
src/Device/ se3_algo_aes256hmacsha256.h	??
src/Device/ se3_algo_AesHmacSha256s.c	
SE3_ALGO_AES_HMACSHA256 crypto handlers	52
src/Device/ se3_algo_AesHmacSha256s.h	
SE3_ALGO_AES_HMACSHA256 crypto handlers	53
src/Device/ se3_algo_HmacSha256.h	??
src/Device/ se3_algo_sha256.h	??
src/Device/ se3_cmd.c	
L0 command dispatch and execute	54
src/Device/ se3_cmd.h	
L0 command dispatch and execute	55
src/Device/ se3_cmd0.c	
L0 command handlers	55
src/Device/ se3_cmd0.h	
L0 command handlers	56
src/Device/ se3_cmd1.c	
L1 command dispatch and execute	57
src/Device/ se3_cmd1.h	
L1 command dispatch and execute	58
src/Device/ se3_cmd1_config.c	
L1 handlers for configuration record operations	59
src/Device/ se3_cmd1_config.h	
L1 handlers for configuration record operations	59

src/Device/ se3_cmd1_crypto.c	
L1 handlers for crypto operations	60
src/Device/ se3_cmd1_crypto.h	
L1 handlers for crypto operations	61
src/Device/ se3_cmd1_keys.c	
L1 handlers for key management operations	63
src/Device/ se3_cmd1_keys.h	
L1 handlers for key management operations	64
src/Device/ se3_cmd1_login.c	
L1 handlers for login operations	65
src/Device/ se3_cmd1_login.h	
L1 handlers for login operations	66
src/Device/ se3_flash.c	
Flash management	67
src/Device/ se3_flash.h	
Flash management	71
src/Device/ se3_keys.c	
Key management	76
src/Device/ se3_keys.h	
Key management	79
src/Device/ se3_memory.c	
Memory management (session allocator)	83
src/Device/ se3_memory.h	
Memory management (session allocator)	85
src/Device/ se3_proto.c	
USB read/write handlers	87
src/Device/ se3_proto.h	
USB read/write handlers	90
src/Device/ se3c0.c	
L0 structures and functions	92
src/Device/ se3c0.h	
L0 structures and functions	92
src/Device/ se3c1.c	
L1 structures and functions	94
src/Device/ se3c1.h	
L1 structures and functions	96

Chapter 4

Module Documentation

4.1 AES return values

AES return values

- `#define B5_AES256_RES_OK (0)`
- `#define B5_AES256_RES_INVALID_CONTEXT (-1)`
- `#define B5_AES256_RES_CANNOT_ALLOCATE_CONTEXT (-2)`
- `#define B5_AES256_RES_INVALID_KEY_SIZE (-3)`
- `#define B5_AES256_RES_INVALID_ARGUMENT (-4)`
- `#define B5_AES256_RES_INVALID_MODE (-5)`

4.1.1 Detailed Description

4.2 AES Key, IV, Block Sizes

AES Key, IV, Block Sizes

- `#define B5_AES_256` 32
- `#define B5_AES_192` 24
- `#define B5_AES_128` 16
- `#define B5_AES_IV_SIZE` 16
- `#define B5_AES_BLK_SIZE` 16

4.2.1 Detailed Description

4.2.2 Macro Definition Documentation

4.2.2.1 `#define B5_AES_128` 16

Key Size in Bytes.

4.2.2.2 `#define B5_AES_192` 24

Key Size in Bytes.

4.2.2.3 `#define B5_AES_256` 32

Key Size in Bytes.

4.2.2.4 `#define B5_AES_BLK_SIZE` 16

Block Size in Bytes.

4.2.2.5 `#define B5_AES_IV_SIZE` 16

IV Size in Bytes.

4.3 AES modes

AES modes

- `#define B5_AES256_OFB 1`
- `#define B5_AES256_ECB_ENC 2`
- `#define B5_AES256_ECB_DEC 3`
- `#define B5_AES256_CBC_ENC 4`
- `#define B5_AES256_CBC_DEC 5`
- `#define B5_AES256_CFB_ENC 6`
- `#define B5_AES256_CFB_DEC 7`
- `#define B5_AES256_CTR 8`

4.3.1 Detailed Description

4.3.2 Macro Definition Documentation

4.3.2.1 `#define B5_AES256_CBC_DEC 5`

CBC decryption

4.3.2.2 `#define B5_AES256_CBC_ENC 4`

CBC encryption

4.3.2.3 `#define B5_AES256_CFB_DEC 7`

CFB decryption

4.3.2.4 `#define B5_AES256_CFB_ENC 6`

CFB decryption

4.3.2.5 `#define B5_AES256_CTR 8`

CTR counter mode encryption-decryption

4.3.2.6 `#define B5_AES256_ECB_DEC 3`

ECB decryption

4.3.2.7 `#define B5_AES256_ECB_ENC 2`

ECB encryption

4.3.2.8 `#define B5_AES256_OFB 1`

OFB full feedback encryption-decryption

4.4 AES data structures

Data Structures

- struct [B5_tAesCtx](#)

4.4.1 Detailed Description

4.5 AES functions

AES functions

- `int32_t B5_Aes256_Init (B5_tAesCtx *ctx, const uint8_t *Key, int16_t keySize, uint8_t aesMode)`
Initialize the AES context.
- `int32_t B5_Aes256_SetIV (B5_tAesCtx *ctx, const uint8_t *IV)`
Set the IV for the current AES context.
- `int32_t B5_Aes256_Update (B5_tAesCtx *ctx, uint8_t *encData, uint8_t *clrData, int16_t nBlk)`
Encrypt/Decrypt data based on the status of current AES context.
- `int32_t B5_Aes256_Finit (B5_tAesCtx *ctx)`
De-initialize the current AES context.

4.5.1 Detailed Description

4.5.2 Function Documentation

4.5.2.1 `int32_t B5_Aes256_Finit (B5_tAesCtx * ctx)`

De-initialize the current AES context.

Parameters

<code>ctx</code>	Pointer to the AES context to de-initialize.
------------------	--

Returns

See [AES return values](#) .

4.5.2.2 `int32_t B5_Aes256_Init (B5_tAesCtx * ctx, const uint8_t * Key, int16_t keySize, uint8_t aesMode)`

Initialize the AES context.

Parameters

<code>ctx</code>	Pointer to the AES data structure to be initialized.
<code>Key</code>	Pointer to the Key that must be used for encryption/decryption.
<code>keySize</code>	Key size. See AES Key, IV, Block Sizes for supported sizes.
<code>aesMode</code>	AES mode. See AES modes for supported modes.

Returns

See [AES return values](#) .

4.5.2.3 `int32_t B5_Aes256_SetIV (B5_tAesCtx * ctx, const uint8_t * IV)`

Set the IV for the current AES context.

Parameters

<i>ctx</i>	Pointer to the AES data structure to be initialized.
<i>IV</i>	Pointer to the IV.

Returns

See [AES return values](#) .

4.5.2.4 `int32_t B5_Aes256_Update (B5_tAesCtx * ctx, uint8_t * encData, uint8_t * clrData, int16_t nBlk)`

Encrypt/Decrypt data based on the status of current AES context.

Parameters

<i>ctx</i>	Pointer to the current AES context.
<i>encData</i>	Encrypted data.
<i>clrData</i>	Clear data.
<i>nBlk</i>	Number of AES blocks to process.

Returns

See [AES return values](#) .

4.6 CMAC-AES Key, Blk Sizes

CMAC-AES Key, Block Sizes

- `#define B5_CMAC_AES_256` 32
- `#define B5_CMAC_AES_192` 24
- `#define B5_CMAC_AES_128` 16
- `#define B5_CMAC_AES_BLK_SIZE` 16

4.6.1 Detailed Description

4.6.2 Macro Definition Documentation

4.6.2.1 `#define B5_CMAC_AES_128` 16

Key Size in Bytes

4.6.2.2 `#define B5_CMAC_AES_192` 24

Key Size in Bytes

4.6.2.3 `#define B5_CMAC_AES_256` 32

Key Size in Bytes

4.6.2.4 `#define B5_CMAC_AES_BLK_SIZE` 16

Block Size in Bytes

4.7 CMAC-AES return values

CMAC-AES return values

- `#define B5_CMAC_AES256_RES_OK (0)`
- `#define B5_CMAC_AES256_RES_INVALID_CONTEXT (-1)`
- `#define B5_CMAC_AES256_RES_CANNOT_ALLOCATE_CONTEXT (-2)`
- `#define B5_CMAC_AES256_RES_INVALID_KEY_SIZE (-3)`
- `#define B5_CMAC_AES256_RES_INVALID_ARGUMENT (-4)`

4.7.1 Detailed Description

4.8 CMAC-AES data structures

Data Structures

- struct [B5_tCmacAesCtx](#)

4.8.1 Detailed Description

4.9 CMAC-AES functions

CMAC-AES functions

- `int32_t B5_CmacAes256_Init (B5_tCmacAesCtx *ctx, const uint8_t *Key, int16_t keySize)`
Initialize the CMAC-AES context.
- `int32_t B5_CmacAes256_Update (B5_tCmacAesCtx *ctx, const uint8_t *data, int32_t dataLen)`
Compute the CMAC-AES algorithm on input data depending on the current status of the CMAC-AES context.
- `int32_t B5_CmacAes256_Finit (B5_tCmacAesCtx *ctx, uint8_t *rSignature)`
De-initialize the current CMAC-AES context.
- `int32_t B5_CmacAes256_Reset (B5_tCmacAesCtx *ctx)`
Reset the current CMAC-AES context.
- `int32_t B5_CmacAes256_Sign (const uint8_t *data, int32_t dataLen, const uint8_t *Key, int16_t keySize, uint8_t *rSignature)`
Compute the signature through the CMAC-AES algorithm.

4.9.1 Detailed Description

4.9.2 Function Documentation

4.9.2.1 `int32_t B5_CmacAes256_Finit (B5_tCmacAesCtx * ctx, uint8_t * rSignature)`

De-initialize the current CMAC-AES context.

Parameters

<i>ctx</i>	Pointer to the CMAC-AES context to de-initialize.
<i>rSignature</i>	Pointer to a blank memory area that can store the computed output signature.

Returns

See [CMAC-AES return values](#) .

4.9.2.2 `int32_t B5_CmacAes256_Init (B5_tCmacAesCtx * ctx, const uint8_t * Key, int16_t keySize)`

Initialize the CMAC-AES context.

Parameters

<i>ctx</i>	Pointer to the CMAC-AES data structure to be initialized.
<i>Key</i>	Pointer to the Key that must be used.
<i>keySize</i>	Key size. See CMAC-AES Key, Blk Sizes for supported sizes.

Returns

See [CMAC-AES return values](#) .

4.9.2.3 `int32_t B5_CmacAes256_Reset (B5_tCmacAesCtx * ctx)`

Reset the current CMAC-AES context.

Parameters

<i>ctx</i>	Pointer to the CMAC-AES context to reset.
------------	---

Returns

See [CMAC-AES return values](#) .

4.9.2.4 `int32_t B5_CmacAes256_Sign (const uint8_t * data, int32_t dataLen, const uint8_t * Key, int16_t keySize, uint8_t * rSignature)`

Compute the signature through the CMAC-AES algorithm.

Parameters

<i>data</i>	Pointer to the input data.
<i>dataLen</i>	Input data length (in Bytes).
<i>Key</i>	Pointer to the Key that must be used.
<i>keySize</i>	Key size. See CMAC-AES Key, Blk Sizes for supported sizes.
<i>rSignature</i>	Pointer to a blank memory area that can store the computed output signature.

Returns

See [CMAC-AES return values](#) .

4.9.2.5 `int32_t B5_CmacAes256_Update (B5_tCmacAesCtx * ctx, const uint8_t * data, int32_t dataLen)`

Compute the CMAC-AES algorithm on input data depending on the current status of the CMAC-AES context.

Parameters

<i>ctx</i>	Pointer to the current CMAC-AES context.
<i>data</i>	Pointer to the input data.
<i>dataLen</i>	Bytes to be processed.

Returns

See [CMAC-AES return values](#) .

4.10 AccessLogin

Use this values as access parameter when using L1_login.

Enumerations

- enum { **SE3_ACCESS_USER** = 100, **SE3_ACCESS_ADMIN** = 1000, **SE3_ACCESS_MAX** = 0xFFFF }

4.10.1 Detailed Description

Use this values as access parameter when using L1_login.

4.11 KeyOpEdit

Use these values when using L1_key_edit.

Enumerations

- enum { [SE3_KEY_OP_INSERT](#) = 1, [SE3_KEY_OP_DELETE](#) = 2, [SE3_KEY_OP_UPSERT](#) = 3 }

4.11.1 Detailed Description

Use these values when using L1_key_edit.

4.11.2 Enumeration Type Documentation

4.11.2.1 anonymous enum

Enumerator

- SE3_KEY_OP_INSERT*** Use this value to insert a new key
SE3_KEY_OP_DELETE Use this value to delete a new key
SE3_KEY_OP_UPSERT Use this value to update/insert a key

4.12 AlgorithmAvail

Enumerations

- enum {
SE3_ALGO_AES = 0, SE3_ALGO_SHA256 = 1, SE3_ALGO_HMACSHA256 = 2, SE3_ALGO_AES_HMACSHA256 = 3,
SE3_ALGO_AES_HMAC = 4, SE3_ALGO_MAX = 8 }

4.12.1 Detailed Description

4.12.2 Enumeration Type Documentation

4.12.2.1 anonymous enum

Enumerator

SE3_ALGO_AES AES.

SE3_ALGO_SHA256 SHA256.

SE3_ALGO_HMACSHA256 HMAC-SHA256.

SE3_ALGO_AES_HMACSHA256 AES + HMAC-SHA256.

SE3_ALGO_AES_HMAC AES 256 + HMAC Auth TODO remove.

4.13 SHA256 return values

SHA256 return values

- `#define B5_SHA256_RES_OK (0)`
- `#define B5_SHA256_RES_INVALID_CONTEXT (-1)`
- `#define B5_SHA256_RES_CANNOT_ALLOCATE_CONTEXT (-2)`
- `#define B5_SHA256_RES_INVALID_ARGUMENT (-3)`
- `#define B5_HMAC_SHA256_RES_OK (0)`
- `#define B5_HMAC_SHA256_RES_INVALID_CONTEXT (-1)`
- `#define B5_HMAC_SHA256_RES_CANNOT_ALLOCATE_CONTEXT (-2)`
- `#define B5_HMAC_SHA256_RES_INVALID_ARGUMENT (-3)`

4.13.1 Detailed Description

4.14 SHA256 digest and block sizes

SHA256 digest and block sizes

- #define **B5_SHA256_DIGEST_SIZE** 32
- #define **B5_SHA256_BLOCK_SIZE** 64

4.14.1 Detailed Description

4.15 SHA256 data structures

Data Structures

- struct [B5_tSha256Ctx](#)

4.15.1 Detailed Description

4.16 SHA256 functions

SHA256 functions

- `int32_t B5_Sha256_Init (B5_tSha256Ctx *ctx)`
Initialize the SHA256 context.
- `int32_t B5_Sha256_Update (B5_tSha256Ctx *ctx, const uint8_t *data, int32_t dataLen)`
Compute the SHA256 algorithm on input data depending on the current status of the SHA256 context.
- `int32_t B5_Sha256_Finit (B5_tSha256Ctx *ctx, uint8_t *rDigest)`
De-initialize the current SHA256 context.

4.16.1 Detailed Description

4.16.2 Function Documentation

4.16.2.1 `int32_t B5_Sha256_Finit (B5_tSha256Ctx * ctx, uint8_t * rDigest)`

De-initialize the current SHA256 context.

Parameters

<i>ctx</i>	Pointer to the SHA context to de-initialize.
<i>rDigest</i>	Pointer to a blank memory area that can store the computed output digest.

Returns

See [SHA256 return values](#) .

4.16.2.2 `int32_t B5_Sha256_Init (B5_tSha256Ctx * ctx)`

Initialize the SHA256 context.

Parameters

<i>ctx</i>	Pointer to the SHA256 data structure to be initialized.
------------	---

Returns

See [SHA256 return values](#) .

4.16.2.3 `int32_t B5_Sha256_Update (B5_tSha256Ctx * ctx, const uint8_t * data, int32_t dataLen)`

Compute the SHA256 algorithm on input data depending on the current status of the SHA256 context.

Parameters

<i>ctx</i>	Pointer to the current SHA context.
<i>data</i>	Pointer to the input data.
<i>dataLen</i>	Bytes to be processed.

Returns

See [SHA256 return values](#) .

4.17 HMAC-SHA256 return values

SHA256 return values

- `#define B5_HMAC_SHA256_RES_OK (0)`
- `#define B5_HMAC_SHA256_RES_INVALID_CONTEXT (-1)`
- `#define B5_HMAC_SHA256_RES_CANNOT_ALLOCATE_CONTEXT (-2)`
- `#define B5_HMAC_SHA256_RES_INVALID_ARGUMENT (-3)`

4.17.1 Detailed Description

4.18 HMAC-SHA256 data structures

Data Structures

- struct [B5_tHmacSha256Ctx](#)

4.18.1 Detailed Description

4.19 HMAC-SHA256 functions

HMAC-SHA256 functions

- `int32_t B5_HmacSha256_Init (B5_tHmacSha256Ctx *ctx, const uint8_t *Key, int16_t keySize)`
Initialize the HMAC-SHA256 context.
- `int32_t B5_HmacSha256_Update (B5_tHmacSha256Ctx *ctx, const uint8_t *data, int32_t dataLen)`
Compute the HMAC-SHA256 algorithm on input data depending on the current status of the HMAC-SHA256 context.
- `int32_t B5_HmacSha256_Finit (B5_tHmacSha256Ctx *ctx, uint8_t *rDigest)`
De-initialize the current HMAC-SHA256 context.

4.19.1 Detailed Description

4.19.2 Function Documentation

4.19.2.1 `int32_t B5_HmacSha256_Finit (B5_tHmacSha256Ctx * ctx, uint8_t * rDigest)`

De-initialize the current HMAC-SHA256 context.

Parameters

<i>ctx</i>	Pointer to the HMAC-SHA256 context to de-initialize.
<i>rDigest</i>	Pointer to a blank memory area that can store the computed output digest.

Returns

See [HMAC-SHA256 return values](#) .

4.19.2.2 `int32_t B5_HmacSha256_Init (B5_tHmacSha256Ctx * ctx, const uint8_t * Key, int16_t keySize)`

Initialize the HMAC-SHA256 context.

Parameters

<i>ctx</i>	Pointer to the HMAC-SHA256 data structure to be initialized.
<i>Key</i>	Pointer to the Key that must be used.
<i>keySize</i>	Key size.

Returns

See [HMAC-SHA256 return values](#) .

4.19.2.3 `int32_t B5_HmacSha256_Update (B5_tHmacSha256Ctx * ctx, const uint8_t * data, int32_t dataLen)`

Compute the HMAC-SHA256 algorithm on input data depending on the current status of the HMAC-SHA256 context.

Parameters

<i>ctx</i>	Pointer to the current HMAC-SHA256 context.
<i>data</i>	Pointer to the input data.
<i>dataLen</i>	Bytes to be processed.

Returns

See [HMAC-SHA256 return values](#) .

Chapter 5

Data Structure Documentation

5.1 AesHmacSha256s_ctx Struct Reference

Data Fields

- [B5_tAesCtx](#) * **aes**
- [B5_tHmacSha256Ctx](#) * **hmac**
- [uint8_t](#) * **keys**
- [uint16_t](#) * **key_size**
- [uint8_t](#) * **state**
- [uint8_t](#) * **mode**
- [uint8_t](#) * **direction**

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

- [src/Device/se3_algo_AesHmacSha256s.c](#)

5.2 B5_tAesCtx Struct Reference

Data Fields

- [uint32_t](#) [rk](#) [4 *(14+1)]
- [uint8_t](#) [Nr](#)
- [uint8_t](#) [InitVector](#) [16]
- [uint8_t](#) [mode](#)
- [uint32_t](#) const * **Te0**
- [uint32_t](#) const * **Te1**
- [uint32_t](#) const * **Te2**
- [uint32_t](#) const * **Te3**
- [uint32_t](#) const * **Te4**
- [uint32_t](#) const * **Td0**
- [uint32_t](#) const * **Td1**
- [uint32_t](#) const * **Td2**
- [uint32_t](#) const * **Td3**
- [uint32_t](#) const * **Td4**

5.2.1 Field Documentation

5.2.1.1 `uint8_t B5_tAesCtx::InitVector[16]`

IV for OFB, CBC, CTR

5.2.1.2 `uint8_t B5_tAesCtx::mode`

Active mode

5.2.1.3 `uint8_t B5_tAesCtx::Nr`

Number of rounds

5.2.1.4 `uint32_t B5_tAesCtx::rk[4 * (14+1)]`

Precomputed round keys

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

- `src/Common/aes256.h`

5.3 B5_tCmacAesCtx Struct Reference

Data Fields

- [B5_tAesCtx](#) `aesCtx`
- `uint8_t K1` [32]
- `uint8_t K2` [32]
- `uint8_t tmpBlk` [[B5_AES_BLK_SIZE](#)]
- `uint8_t tmpBlkLen`
- `uint8_t C` [[B5_AES_BLK_SIZE](#)]

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

- `src/Common/aes256.h`

5.4 B5_tHmacSha256Ctx Struct Reference

Data Fields

- [B5_tSha256Ctx](#) `shaCtx`
- `uint8_t iPad` [64]
- `uint8_t oPad` [64]

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

- `src/Common/sha256.h`

5.5 B5_tSha256Ctx Struct Reference

Data Fields

- uint32_t **total** [2]
- uint32_t **state** [8]
- uint8_t **buffer** [64]
- uint32_t **W** [64]

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

- src/Common/sha256.h

5.6 s3_storage_range_ Struct Reference

SDIO read/write request buffer context.

Data Fields

- uint8_t * **buf**
- uint32_t **first**
- uint32_t **count**

5.6.1 Detailed Description

SDIO read/write request buffer context.

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

- src/Device/[se3_proto.c](#)

5.7 se3_algo_descriptor_ Struct Reference

algorithm descriptor type

```
#include <se3cl.h>
```

Data Fields

- [se3_crypto_init_handler](#) `init`
L1_crypto_init function.
- [se3_crypto_update_handler](#) `update`
L1_crypto_update function.
- `uint16_t` [size](#)
context size size
- `char` [display_name](#) [16]
name for the algorithm list API
- `uint16_t` [display_type](#)
type for the algorithm list API
- `uint16_t` [display_block_size](#)
block size for the algorithm list API
- `uint16_t` [display_key_size](#)
key size for the algorithm list API

5.7.1 Detailed Description

algorithm descriptor type

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

- `src/Device/se3c1.h`

5.8 SE3_COMM_STATUS_ Struct Reference

structure holding host-device communication status and buffers

```
#include <se3c0.h>
```

Data Fields

- `bool` [magic_ready](#)
magic written flag
- `uint32_t` [magic_bmap](#)
bit map of written magic sectors
- `uint32_t` [blocks](#) [SE3_COMM_N]
map of blocks
- `uint32_t` [block_guess](#)
guess for next block that will be accessed
- `bool` [locked](#)
prevent magic initialization
- `volatile bool` [req_ready](#)
request ready flag
- `uint32_t` [req_bmap](#)
map of received request blocks
- `uint8_t *` [req_data](#)

- received data buffer*
- uint8_t * [req_hdr](#)
- received header buffer*
- volatile bool [resp_ready](#)
- response ready flag*
- uint32_t [resp_bmap](#)
- map of sent response blocks*
- uint8_t * [resp_data](#)
- buffer for data to be sent*
- uint8_t * [resp_hdr](#)
- buffer for header to be sent*

5.8.1 Detailed Description

structure holding host-device communication status and buffers

req_ready and resp_ready must be volatile, otherwise -O3 optimization will not work.

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

- src/Device/[se3c0.h](#)

5.9 SE3_FLASH_INFO_ Struct Reference

Flash management structure.

```
#include <se3c0.h>
```

Data Fields

- uint32_t [sector](#)
- active sector number*
- const uint8_t * **base**
- const uint8_t * **index**
- const uint8_t * **data**
- size_t **first_free_pos**
- size_t **used**
- size_t **allocated**

5.9.1 Detailed Description

Flash management structure.

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

- src/Device/[se3c0.h](#)

5.10 se3_flash_it_ Struct Reference

Flash node iterator structure.

```
#include <se3_flash.h>
```

Data Fields

- `const uint8_t * addr`
- `uint8_t type`
- `uint16_t size`
- `uint16_t blocks`
- `size_t pos`

5.10.1 Detailed Description

Flash node iterator structure.

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

- `src/Device/se3_flash.h`

5.11 se3_flash_key_ Struct Reference

Flash key structure.

```
#include <se3_keys.h>
```

Data Fields

- `uint32_t id`
- `uint32_t validity`
- `uint16_t data_size`
- `uint16_t name_size`
- `uint8_t * data`
- `uint8_t * name`

5.11.1 Detailed Description

Flash key structure.

Disposition of the fields within the flash node: 0:3 id 4:7 validity 8:9 data_size 10:11 name_size 12:(12+data_size-1) data (12+data_size):(12+data_size+name_size-1) name

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

- `src/Device/se3_keys.h`

5.12 SE3_L0_GLOBALS_ Struct Reference

L0 globals structure.

```
#include <se3c0.h>
```

Data Fields

- [SE3_SERIAL](#) **serial**
- [SE3_FLASH_INFO](#) **flash**
- [SE3_COMM_STATUS](#) **comm**
- [se3c0_req_header](#) **req_hdr**
- [se3c0_resp_header](#) **resp_hdr**
- union {
 - [B5_tSha256Ctx](#) **sha**
 - [B5_tAesCtx](#) **aes**
- **ctx**
- [uint16_t](#) **hwerror**
- [uint64_t](#) **now**
 - current UNIX time in seconds*
- [bool](#) **now_initialized**
 - time was initialized*

5.12.1 Detailed Description

L0 globals structure.

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

- `src/Device/se3c0.h`

5.13 SE3_L1_GLOBALS_ Struct Reference

L1 globals structure.

```
#include <se3c1.h>
```

Data Fields

- [SE3_LOGIN_STATUS](#) **login**
- [SE3_RECORD_INFO](#) **records** [SE3_RECORD_MAX]
- [se3_mem](#) **sessions**
- [uint16_t](#) **sessions_algo** [SE3_SESSIONS_MAX]

5.13.1 Detailed Description

L1 globals structure.

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

- `src/Device/se3c1.h`

5.14 SE3_LOGIN_STATUS_ Struct Reference

L1 login status data.

```
#include <se3c1.h>
```

Data Fields

- bool `y`
logged in
- uint16_t `access`
access level
- uint16_t `challenge_access`
access level of the last offered challenge
- union {
 uint8_t `token` [SE3_L1_TOKEN_SIZE]
 login token
 uint8_t `challenge` [SE3_L1_CHALLENGE_SIZE]
 login challenge response expected
};
- uint8_t `key` [SE3_L1_KEY_SIZE]
session key for protocol encryption
- `se3_payload_cryptoctx` `cryptoctx`
context for protocol encryption
- bool `cryptoctx_initialized`
context initialized flag

5.14.1 Detailed Description

L1 login status data.

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

- `src/Device/se3c1.h`

5.15 se3_mem_ Struct Reference

memory allocator structure

```
#include <se3_memory.h>
```

Data Fields

- `size_t max_count`
- `uint8_t ** ptr`
- `uint8_t * dat`
- `size_t dat_size`
- `size_t used`

5.15.1 Detailed Description

memory allocator structure

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

- `src/Device/se3_memory.h`

5.16 se3_payload_cryptoctx_ Struct Reference

Data Fields

- `B5_tAesCtx aesenc`
- `B5_tAesCtx aesdec`
- `B5_tHmacSha256Ctx hmac`
- `uint8_t hmac_key [B5_AES_256]`
- `uint8_t auth [B5_SHA256_DIGEST_SIZE]`

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

- `src/Common/se3_common.h`

5.17 SE3_RECORD_INFO_ Struct Reference

Record information.

```
#include <se3c1.h>
```

Data Fields

- `uint16_t read_access`
required access level for read
- `uint16_t write_access`
required access level for write

5.17.1 Detailed Description

Record information.

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

- [src/Device/se3c1.h](#)

5.18 SE3_SERIAL_ Struct Reference

serial number data and state

```
#include <se3c0.h>
```

Data Fields

- `uint8_t data` [SE3_SERIAL_SIZE]
- `bool written`

Indicates whether the serial number has been set (by FACTORY_INIT)

5.18.1 Detailed Description

serial number data and state

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

- [src/Device/se3c0.h](#)

5.19 se3c0_req_header_ Struct Reference

decoded request header

```
#include <se3c0.h>
```

Data Fields

- `uint16_t cmd`
- `uint16_t cmd_flags`
- `uint16_t len`
- `uint32_t cmdtok` [SE3_COMM_N-1]

5.19.1 Detailed Description

decoded request header

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

- [src/Device/se3c0.h](#)

5.20 se3c0_resp_header_ Struct Reference

response header to be encoded

```
#include <se3c0.h>
```

Data Fields

- `uint16_t ready`
- `uint16_t status`
- `uint16_t len`
- `uint32_t cmdtok [SE3_COMM_N-1]`

5.20.1 Detailed Description

response header to be encoded

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

- [src/Device/se3c0.h](#)

Chapter 6

File Documentation

6.1 src/Common/crc16.h File Reference

This file contains defines and functions for computing CRC.

```
#include <stddef.h>
#include <stdint.h>
```

Functions

- uint16_t [se3_crc16_update](#) (size_t length, const uint8_t *data, uint16_t crc)
Compute CRC.

Variables

- const uint16_t **se3_crc16_table** [0x100]

6.1.1 Detailed Description

This file contains defines and functions for computing CRC.

6.1.2 Function Documentation

6.1.2.1 uint16_t se3_crc16_update (size_t length, const uint8_t * data, uint16_t crc)

Compute CRC.

Parameters

in	<i>length</i>	Data length
in	<i>data</i>	Data on which CRC is computed
in	<i>crc</i>	CRC

Returns

CRC computed

6.2 src/Common/se3_common.h File Reference

This file contains defines and functions common for L0 and L1.

```
#include "se3c0def.h"
#include "aes256.h"
#include "sha256.h"
#include "pbkdf2.h"
```

Data Structures

- struct [se3_payload_cryptctx](#)

Typedefs

- typedef struct [se3_payload_cryptctx](#) **se3_payload_cryptctx**

Functions

- uint16_t [se3_req_len_data](#) (uint16_t len_data_and_headers)
Compute length of data in a request in terms of SE3_COMM_BLOCK blocks.
- uint16_t [se3_req_len_data_and_headers](#) (uint16_t len_data)
Compute length of data in a request accounting for headers.
- uint16_t [se3_resp_len_data](#) (uint16_t len_data_and_headers)
Compute length of data in a request in terms of SE3_COMM_BLOCK blocks.
- uint16_t [se3_resp_len_data_and_headers](#) (uint16_t len_data)
Compute length of data in a response accounting for headers.
- uint16_t [se3_nblocks](#) (uint16_t len)
Compute number of SE3_COMM_BLOCK blocks, given length in Bytes.
- void **se3_payload_cryptinit** ([se3_payload_cryptctx](#) *ctx, const uint8_t *key)
- void **se3_payload_encrypt** ([se3_payload_cryptctx](#) *ctx, uint8_t *auth, uint8_t *iv, uint8_t *data, uint16_t nblocks, uint16_t flags)
- bool **se3_payload_decrypt** ([se3_payload_cryptctx](#) *ctx, const uint8_t *auth, const uint8_t *iv, uint8_t *data, uint16_t nblocks, uint16_t flags)

Variables

- const uint8_t **se3_magic** [SE3_MAGIC_SIZE]

6.2.1 Detailed Description

This file contains defines and functions common for L0 and L1.

6.2.2 Function Documentation

6.2.2.1 uint16_t se3_nblocks (uint16_t len)

Compute number of SE3_COMM_BLOCK blocks, given length in Bytes.

Parameters

in	<i>len</i>	Length
----	------------	--------

Returns

Number of Blocks

6.2.2.2 `uint16_t se3_req_len_data (uint16_t len_data_and_headers)`

Compute length of data in a request in terms of SE3_COMM_BLOCK blocks.

Parameters

in	<i>len_data_and_headers</i>	Data length
----	-----------------------------	-------------

Returns

Number of SE3_COMM_BLOCK blocks

6.2.2.3 `uint16_t se3_req_len_data_and_headers (uint16_t len_data)`

Compute length of data in a request accounting for headers.

Parameters

in	<i>len_data</i>	Data length
----	-----------------	-------------

Returns

Number of Bytes

6.2.2.4 `uint16_t se3_resp_len_data (uint16_t len_data_and_headers)`

Compute length of data in a request in terms of SE3_COMM_BLOCK blocks.

Parameters

in	<i>len_data_and_headers</i>	Data length
----	-----------------------------	-------------

Returns

Number of SE3_COMM_BLOCK blocks

6.2.2.5 uint16_t se3_resp_len_data_and_headers (uint16_t len_data)

Compute length of data in a response accounting for headers.

Parameters

in	<i>len_data</i>	Data Length
----	-----------------	-------------

Returns

Number of Bytes

6.3 src/Common/se3c1def.h File Reference

This file contains defines to be used both for L1 and L0 functions.

```
#include "se3c0def.h"
```

Macros

- #define **SE3_DIR_SHIFT** (8)

Enumerations

- enum {
SE3_ERR_ACCESS = 100, **SE3_ERR_PIN** = 101, **SE3_ERR_RESOURCE** = 200, **SE3_ERR_EXPIRED** = 201,
SE3_ERR_MEMORY = 400, **SE3_ERR_AUTH** = 401 }
- enum { **SE3_ACCESS_USER** = 100, **SE3_ACCESS_ADMIN** = 1000, **SE3_ACCESS_MAX** = 0xFFFF }
- enum { **SE3_RECORD_SIZE** = 32, **SE3_RECORD_MAX** = 2 }
- enum { **SE3_RECORD_TYPE_ADMINPIN** = 0, **SE3_RECORD_TYPE_USERPIN** = 1 }
- enum {
SE3_L1_PIN_SIZE = 32, **SE3_L1_KEY_SIZE** = 32, **SE3_L1_AUTH_SIZE** = 16, **SE3_L1_CRYPTOBLOCK_SIZE** = 16,
SE3_L1_CHALLENGE_SIZE = 32, **SE3_L1_CHALLENGE_ITERATIONS** = 32, **SE3_L1_IV_SIZE** = 16, **SE3_L1_TOKEN_SIZE** = 16 }
- enum {
SE3_REQ1_OFFSET_AUTH = 0, **SE3_REQ1_OFFSET_IV** = 16, **SE3_REQ1_OFFSET_TOKEN** = 32, **SE3_REQ1_OFFSET_LEN** = 48,
SE3_REQ1_OFFSET_CMD = 50, **SE3_REQ1_OFFSET_DATA** = 64, **SE3_REQ1_MAX_DATA** = (SE3_REQ1_MAX_DATA - SE3_REQ1_OFFSET_DATA) }
- enum {
SE3_RESP1_OFFSET_AUTH = 0, **SE3_RESP1_OFFSET_IV** = 16, **SE3_RESP1_OFFSET_TOKEN** = 32,
SE3_RESP1_OFFSET_LEN = 48,
SE3_RESP1_OFFSET_STATUS = 50, **SE3_RESP1_OFFSET_DATA** = 64, **SE3_RESP1_MAX_DATA** = (SE3_RESP1_MAX_DATA - SE3_RESP1_OFFSET_DATA) }

- enum {
SE3_CMD1_CHALLENGE = 1, **SE3_CMD1_LOGIN** = 2, **SE3_CMD1_LOGOUT** = 3, **SE3_CMD1_CONFIG** = 4,
SE3_CMD1_KEY_EDIT = 5, **SE3_CMD1_KEY_LIST** = 6, **SE3_CMD1_CRYPT0_INIT** = 7, **SE3_CMD1_CRYPT0_UPDATE** = 8,
SE3_CMD1_CRYPT0_LIST = 9, **SE3_CMD1_CRYPT0_SET_TIME** = 10 }
- enum { **SE3_CONFIG_OP_GET** = 1, **SE3_CONFIG_OP_SET** = 2 }
- enum { **SE3_CMD1_CONFIG_REQ_OFF_ID** = 0, **SE3_CMD1_CONFIG_REQ_OFF_OP** = 2, **SE3_CMD1_CONFIG_REQ_OFF_VALUE** = 4, **SE3_CMD1_CONFIG_RESP_OFF_VALUE** = 0 }
- enum {
SE3_CMD1_CHALLENGE_REQ_OFF_CC1 = 0, **SE3_CMD1_CHALLENGE_REQ_OFF_CC2** = 32, **SE3_CMD1_CHALLENGE_REQ_OFF_ACCESS** = 64, **SE3_CMD1_CHALLENGE_REQ_SIZE** = 66,
SE3_CMD1_CHALLENGE_RESP_OFF_SC = 0, **SE3_CMD1_CHALLENGE_RESP_OFF_SRESP** = 32, **SE3_CMD1_CHALLENGE_RESP_SIZE** = 64 }
- enum { **SE3_CMD1_LOGIN_REQ_OFF_CRESP** = 0, **SE3_CMD1_LOGIN_REQ_SIZE** = 32, **SE3_CMD1_LOGIN_RESP_OFF_TOKEN** = 0, **SE3_CMD1_LOGIN_RESP_SIZE** = 16 }
- enum { **SE3_KEY_DATA_MAX** = 2048, **SE3_KEY_NAME_MAX** = 32 }
- enum { **SE3_KEY_OP_INSERT** = 1, **SE3_KEY_OP_DELETE** = 2, **SE3_KEY_OP_UPSERT** = 3 }
- enum {
SE3_CMD1_KEY_EDIT_REQ_OFF_OP = 0, **SE3_CMD1_KEY_EDIT_REQ_OFF_ID** = 2, **SE3_CMD1_KEY_EDIT_REQ_OFF_VALIDITY** = 6, **SE3_CMD1_KEY_EDIT_REQ_OFF_DATA_LEN** = 10,
SE3_CMD1_KEY_EDIT_REQ_OFF_NAME_LEN = 12, **SE3_CMD1_KEY_EDIT_REQ_OFF_DATA_AND_NAME** = 14 }
- enum {
SE3_CMD1_KEY_LIST_REQ_SIZE = 4, **SE3_CMD1_KEY_LIST_REQ_OFF_SKIP** = 0, **SE3_CMD1_KEY_LIST_REQ_OFF_NMAX** = 2, **SE3_CMD1_KEY_LIST_RESP_OFF_COUNT** = 0,
SE3_CMD1_KEY_LIST_RESP_OFF_KEYINFO = 2, **SE3_CMD1_KEY_LIST_KEYINFO_OFF_ID** = 0, **SE3_CMD1_KEY_LIST_KEYINFO_OFF_VALIDITY** = 4, **SE3_CMD1_KEY_LIST_KEYINFO_OFF_DATA_LEN** = 8,
SE3_CMD1_KEY_LIST_KEYINFO_OFF_NAME_LEN = 10, **SE3_CMD1_KEY_LIST_KEYINFO_OFF_NAME** = 12 }
- enum { **SE3_ALGO_INVALID** = 0xFFFF, **SE3_SESSION_INVALID** = 0xFFFFFFFF, **SE3_KEY_INVALID** = 0xFFFFFFFF }
- enum {
SE3_ALGO_AES = 0, **SE3_ALGO_SHA256** = 1, **SE3_ALGO_HMACSHA256** = 2, **SE3_ALGO_AES_HMACSHA256** = 3,
SE3_ALGO_AES_HMAC = 4, **SE3_ALGO_MAX** = 8 }
- enum {
SE3_CMD1_CRYPT0_INIT_REQ_SIZE = 8, **SE3_CMD1_CRYPT0_INIT_REQ_OFF_ALGO** = 0, **SE3_CMD1_CRYPT0_INIT_REQ_OFF_MODE** = 2, **SE3_CMD1_CRYPT0_INIT_REQ_OFF_KEY_ID** = 4,
SE3_CMD1_CRYPT0_INIT_RESP_SIZE = 4, **SE3_CMD1_CRYPT0_INIT_RESP_OFF_SID** = 0 }
- enum {
SE3_CMD1_CRYPT0_UPDATE_REQ_OFF_SID = 0, **SE3_CMD1_CRYPT0_UPDATE_REQ_OFF_FLAGS** = 4, **SE3_CMD1_CRYPT0_UPDATE_REQ_OFF_DATAIN1_LEN** = 6, **SE3_CMD1_CRYPT0_UPDATE_REQ_OFF_DATAIN2_LEN** = 8,
SE3_CMD1_CRYPT0_UPDATE_REQ_OFF_DATA = 16, **SE3_CMD1_CRYPT0_UPDATE_RESP_OFF_DATAOUT_LEN** = 0, **SE3_CMD1_CRYPT0_UPDATE_RESP_OFF_DATA** = 16 }
- enum {
SE3_CRYPT0_FLAG_INIT = (1 << 15), **SE3_CRYPT0_FLAG_RESET** = (1 << 14), **SE3_CRYPT0_FLAG_SETIV** = **SE3_CRYPT0_FLAG_RESET**, **SE3_CRYPT0_FLAG_SETNONCE** = (1 << 13),
SE3_CRYPT0_FLAG_AUTH = (1 << 12) }
- enum { **SE3_CRYPT0_MAX_DATAIN** = (**SE3_REQ1_MAX_DATA** - **SE3_CMD1_CRYPT0_UPDATE_REQ_OFF_DATA**), **SE3_CRYPT0_MAX_DATAOUT** = (**SE3_RESP1_MAX_DATA** - **SE3_CMD1_CRYPT0_UPDATE_RESP_OFF_DATA**) }
- enum { **SE3_CMD1_CRYPT0_SET_TIME_REQ_SIZE** = 4, **SE3_CMD1_CRYPT0_SET_TIME_REQ_OFF_DEVTIME** = 0 }

- enum {
SE3_CMD1_CRYPTΟ_LIST_REQ_SIZE = 0, SE3_CMD1_CRYPTΟ_LIST_RESP_OFF_COUNT = 0, SE3_CMD1_CRYPTΟ_LIST_RESP_OFF_ALGOINFO = 2, SE3_CMD1_CRYPTΟ_ALGOINFO_SIZE = 22, SE3_CMD1_CRYPTΟ_ALGOINFO_OFF_NAME = 0, SE3_CMD1_CRYPTΟ_ALGOINFO_OFF_TYPE = 16, SE3_CMD1_CRYPTΟ_ALGOINFO_OFF_BLOCK_SIZE = 18, SE3_CMD1_CRYPTΟ_ALGOINFO_OFF_KEY_SIZE = 20, SE3_CMD1_CRYPTΟ_ALGOINFO_NAME_SIZE = 16 }
- enum {
SE3_CRYPTΟ_TYPE_BLOCKCIPHER = 0, SE3_CRYPTΟ_TYPE_STREAMCIPHER = 1, SE3_CRYPTΟ_TYPE_DIGEST = 2, SE3_CRYPTΟ_TYPE_BLOCKCIPHER_AUTH = 3, SE3_CRYPTΟ_TYPE_OTHER = 0xFFFF }
- enum {
SE3_FEEDBACK_ECB = 1, SE3_FEEDBACK_CBC = 2, SE3_FEEDBACK_OFB = 3, SE3_FEEDBACK_CTR = 4, SE3_FEEDBACK_CFB = 5, SE3_DIR_ENCRYPT = (1 << SE3_DIR_SHIFT), SE3_DIR_DECRYPT = (2 << SE3_DIR_SHIFT) }

L1_crypto_init default modes.

6.3.1 Detailed Description

This file contains defines to be used both for L1 and L0 functions.

6.3.2 Enumeration Type Documentation

6.3.2.1 anonymous enum

Configuration records definitions

6.3.2.2 anonymous enum

Default configuration record types

6.3.2.3 anonymous enum

L1 field size definitions

6.3.2.4 anonymous enum

L1 request fields definitions

6.3.2.5 anonymous enum

L1 response fields definitions

6.3.2.6 anonymous enum

L1 command codes

6.3.2.7 anonymous enum

L1_config operations

6.3.2.8 anonymous enum

L1_config fields

6.3.2.9 anonymous enum

L1_challenge fields

6.3.2.10 anonymous enum

L1_login fields

6.3.2.11 anonymous enum

Keys: maximum sizes for variable fields

6.3.2.12 anonymous enum

L1_key_edit fields

6.3.2.13 anonymous enum

L1_key_list fields

6.3.2.14 anonymous enum

Invalid handle values

6.3.2.15 anonymous enum

L1_crypto_init fields

6.3.2.16 anonymous enum

L1_crypto_update fields

6.3.2.17 anonymous enum

L1_crypto_update default flags

6.3.2.18 anonymous enum

L1_crypto_update maximum buffer sizes

6.3.2.19 anonymous enum

L1_crypto_set_time fields

6.3.2.20 anonymous enum

L1_crypto_list fields

6.3.2.21 anonymous enum

L1_crypto_list default cipher types

6.3.2.22 anonymous enum

L1_crypto_init default modes.

One FEEDBACK and one DIR may be combined to specify the desired mode Example: Encrypt in CBC mode (SE3_FEEDBACK_CBC | SE3_DIR_ENCRYPT)

6.3.2.23 anonymous enum

L1 errors

Enumerator

- SE3_ERR_ACCESS** insufficient privileges
- SE3_ERR_PIN** pin rejected
- SE3_ERR_RESOURCE** resource not found
- SE3_ERR_EXPIRED** resource expired
- SE3_ERR_MEMORY** no more space to allocate resource
- SE3_ERR_AUTH** SHA256HMAC Authentication failed.

6.4 src/Device/se3_algo_Aes.h File Reference

SE3_ALGO_AES crypto handlers.

```
#include "se3c1.h"
```

Functions

- `uint16_t se3_algo_Aes_init (se3_flash_key *key, uint16_t mode, uint8_t *ctx)`
SE3_ALGO_AES init handler.
- `uint16_t se3_algo_Aes_update (uint8_t *ctx, uint16_t flags, uint16_t datain1_len, const uint8_t *datain1, uint16_t datain2_len, const uint8_t *datain2, uint16_t *dataout_len, uint8_t *dataout)`
SE3_ALGO_AES update handler.

6.4.1 Detailed Description

SE3_ALGO_AES crypto handlers.

Author

Nicola Ferri

6.4.2 Function Documentation

6.4.2.1 `uint16_t se3_algo_Aes_init (se3_flash_key * key, uint16_t mode, uint8_t * ctx)`

SE3_ALGO_AES init handler.

Supported modes Any combination of one of {SE3_DIR_ENCRYPT, SE3_DIR_DECRYPT} and one of {SE3_FEE←DBACK_ECB, SE3_FEEDBACK_CBC, SE3_FEEDBACK_CFB, SE3_FEEDBACK_OFB, SE3_FEEDBACK_CTR}

Supported key sizes 128-bit, 192-bit, 256-bit

6.4.2.2 `uint16_t se3_algo_Aes_update (uint8_t * ctx, uint16_t flags, uint16_t datain1_len, const uint8_t * datain1, uint16_t datain2_len, const uint8_t * datain2, uint16_t * dataout_len, uint8_t * dataout)`

SE3_ALGO_AES update handler.

Supported operations (default): encrypt/decrypt datain2 and update HmacSha256 context with datain2. Not executed if datain2 is empty (zero-length) SE3_CRYPTOP_FLAG_SETIV: set new IV from datain1 SE3_CRYPTOP_FLAG_FINISH: release session

Combined operations are executed in the following order: SE3_CRYPTOP_FLAG_SETIV (default) SE3_CRYPTOP_FLAG_FINISH

Contribution of each operation to the output size: (default): + datain2_len Others: + 0

6.5 src/Device/se3_algo_AesHmacSha256s.c File Reference

SE3_ALGO_AES_HMACSHA256 crypto handlers.

```
#include "se3_algo_AesHmacSha256s.h"
```

Data Structures

- struct [AesHmacSha256s_ctx](#)

Enumerations

- enum { **SE3_ALGO_STATE_KEYS_NOT_INITIALIZED** = 0, **SE3_ALGO_STATE_KEYS_INITIALIZED** = 1 }

Functions

- uint16_t [se3_algo_AesHmacSha256s_init](#) ([se3_flash_key](#) *key, uint16_t mode, uint8_t *ctx)
SE3_ALGO_AES_HMACSHA256 init handler.
- uint16_t [se3_algo_AesHmacSha256s_update](#) (uint8_t *ctx, uint16_t flags, uint16_t datain1_len, const uint8_t *datain1, uint16_t datain2_len, const uint8_t *datain2, uint16_t *dataout_len, uint8_t *dataout)
SE3_ALGO_AES_HMACSHA256 update handler.

6.5.1 Detailed Description

SE3_ALGO_AES_HMACSHA256 crypto handlers.

Author

Nicola Ferri

6.5.2 Function Documentation

6.5.2.1 uint16_t [se3_algo_AesHmacSha256s_init](#) ([se3_flash_key](#) * *key*, uint16_t *mode*, uint8_t * *ctx*)

SE3_ALGO_AES_HMACSHA256 init handler.

Supported modes Any combination of one of {SE3_DIR_ENCRYPT, SE3_DIR_DECRYPT} and one of {SE3_FEE←DBACK_ECB, SE3_FEEDBACK_CBC, SE3_FEEDBACK_CFB, SE3_FEEDBACK_OFB, SE3_FEEDBACK_CTR}

Supported key sizes 128-bit, 192-bit, 256-bit

6.5.2.2 `uint16_t se3_algo_AesHmacSha256s_update (uint8_t * ctx, uint16_t flags, uint16_t datain1_len, const uint8_t * datain1, uint16_t datain2_len, const uint8_t * datain2, uint16_t * dataout_len, uint8_t * dataout)`

SE3_ALGO_AES_HMACSHA256 update handler.

Supported operations SE3_CRYPTΟ_FLAG_SETNONCE: set nonce for AES and HMAC key derivation. Optional. If used, it must be the first operation performed after initialization. Otherwise, the keys will be derived from the master key without any salt. Cannot be combined with any other operation. (default): encrypt/decrypt datain2 and update HmacSha256 context with datain2. Not executed if datain2 is empty (zero-length) SE3_CRYPTΟ_FLAG_RESET: set new IV from datain1 and reset the HmacSha256 context, also authenticating the IV. If the IV is empty (zero-length), no IV will be set, and the HmacSha256 will be reset. SE3_CRYPTΟ_FLAG_AUTH: produce authentication tag and append to dataout SE3_CRYPTΟ_FLAG_FINISH: release session

Combined operations are executed in the following order: SE3_CRYPTΟ_FLAG_RESET (default) SE3_CRYPTΟ_FLAG_AUTH SE3_CRYPTΟ_FLAG_FINISH

Contribution of each operation to the output size: (default): + datain2_len SE3_CRYPTΟ_FLAG_AUTH: + B5_SHA256_DIGEST_SIZE Others: + 0

6.6 src/Device/se3_algo_AesHmacSha256s.h File Reference

SE3_ALGO_AES_HMACSHA256 crypto handlers.

```
#include "se3c1.h"
#include "pbkdf2.h"
```

Functions

- `uint16_t se3_algo_AesHmacSha256s_init (se3_flash_key *key, uint16_t mode, uint8_t *ctx)`
SE3_ALGO_AES_HMACSHA256 init handler.
- `uint16_t se3_algo_AesHmacSha256s_update (uint8_t *ctx, uint16_t flags, uint16_t datain1_len, const uint8_t *datain1, uint16_t datain2_len, const uint8_t *datain2, uint16_t *dataout_len, uint8_t *dataout)`
SE3_ALGO_AES_HMACSHA256 update handler.

6.6.1 Detailed Description

SE3_ALGO_AES_HMACSHA256 crypto handlers.

Author

Nicola Ferri

6.6.2 Function Documentation

6.6.2.1 `uint16_t se3_algo_AesHmacSha256s_init (se3_flash_key * key, uint16_t mode, uint8_t * ctx)`

SE3_ALGO_AES_HMACSHA256 init handler.

Supported modes Any combination of one of {SE3_DIR_ENCRYPT, SE3_DIR_DECRYPT} and one of {SE3_FEEDBACK_ECB, SE3_FEEDBACK_CBC, SE3_FEEDBACK_CFB, SE3_FEEDBACK_OFB, SE3_FEEDBACK_CTR}

Supported key sizes 128-bit, 192-bit, 256-bit

6.6.2.2 `uint16_t se3_algo_AesHmacSha256s_update (uint8_t * ctx, uint16_t flags, uint16_t datain1_len, const uint8_t * datain1, uint16_t datain2_len, const uint8_t * datain2, uint16_t * dataout_len, uint8_t * dataout)`

SE3_ALGO_AES_HMACSHA256 update handler.

Supported operations SE3_CRYPTΟ_FLAG_SETNONCE: set nonce for AES and HMAC key derivation. Optional. If used, it must be the first operation performed after initialization. Otherwise, the keys will be derived from the master key without any salt. Cannot be combined with any other operation. (default): encrypt/decrypt datain2 and update HmacSha256 context with datain2. Not executed if datain2 is empty (zero-length) SE3_CRYPTΟ_FLAG_RESET: set new IV from datain1 and reset the HmacSha256 context, also authenticating the IV. If the IV is empty (zero-length), no IV will be set, and the HmacSha256 will be reset. SE3_CRYPTΟ_FLAG_AUTH: produce authentication tag and append to dataout SE3_CRYPTΟ_FLAG_FINISH: release session

Combined operations are executed in the following order: SE3_CRYPTΟ_FLAG_RESET (default) SE3_CRYPTΟ_FLAG_AUTH SE3_CRYPTΟ_FLAG_FINISH

Contribution of each operation to the output size: (default): + datain2_len SE3_CRYPTΟ_FLAG_AUTH: + B5_SHA256_DIGEST_SIZE Others: + 0

6.7 src/Device/se3_cmd.c File Reference

L0 command dispatch and execute.

```
#include "se3_cmd.h"
#include "se3_cmd0.h"
#include "se3_cmd1.h"
#include "se3c1.h"
#include "crc16.h"
```

Functions

- static uint16_t **invalid_cmd_handler** (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)
- static uint16_t **se3_exec** (se3_cmd_func handler)
- void **se3_cmd_execute** ()

Execute received command.

6.7.1 Detailed Description

L0 command dispatch and execute.

Author

Nicola Ferri

6.7.2 Function Documentation

6.7.2.1 void **se3_cmd_execute** ()

Execute received command.

Process the last received request and produce a response

6.8 src/Device/se3_cmd.h File Reference

L0 command dispatch and execute.

```
#include "se3c0.h"
```

Functions

- void [se3_cmd_execute](#) ()
Execute received command.

6.8.1 Detailed Description

L0 command dispatch and execute.

Author

Nicola Ferri

6.8.2 Function Documentation

6.8.2.1 void [se3_cmd_execute](#) ()

Execute received command.

Process the last received request and produce a response

6.9 src/Device/se3_cmd0.c File Reference

L0 command handlers.

```
#include "se3_cmd0.h"  
#include "se3_flash.h"
```

Functions

- uint16_t [L0d_echo](#) (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)
L0 ECHO command handler.
- uint16_t [L0d_factory_init](#) (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)
L0 FACTORY_INIT command handler.

6.9.1 Detailed Description

L0 command handlers.

Author

Nicola Ferri

6.9.2 Function Documentation

6.9.2.1 `uint16_t L0d_echo (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)`

L0 ECHO command handler.

Send back received data

6.9.2.2 `uint16_t L0d_factory_init (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)`

L0 FACTORY_INIT command handler.

Initialize device's serial number

6.10 src/Device/se3_cmd0.h File Reference

L0 command handlers.

```
#include "se3c0.h"
```

Functions

- `uint16_t L0d_echo (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)`
L0 ECHO command handler.
- `uint16_t L0d_factory_init (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)`
L0 FACTORY_INIT command handler.

6.10.1 Detailed Description

L0 command handlers.

Author

Nicola Ferri

6.10.2 Function Documentation

6.10.2.1 uint16_t L0d_echo (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)

L0 ECHO command handler.

Send back received data

6.10.2.2 uint16_t L0d_factory_init (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)

L0 FACTORY_INIT command handler.

Initialize device's serial number

6.11 src/Device/se3_cmd1.c File Reference

L1 command dispatch and execute.

```
#include "se3_cmd1.h"
#include "se3_cmd1_login.h"
#include "se3_cmd1_config.h"
#include "se3_cmd1_keys.h"
#include "se3_cmd1_crypto.h"
#include "se3_rand.h"
```

Macros

- `#define SE3_CMD1_MAX` (16)

Functions

- static uint16_t **L1d_error** (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)
- uint16_t **L0d_cmd1** (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)

L0 command which executes an L1 command.

Variables

- static `se3_cmd_func` **L1d_handlers** [SE3_CMD1_MAX]

6.11.1 Detailed Description

L1 command dispatch and execute.

Author

Nicola Ferri

6.11.2 Function Documentation

6.11.2.1 uint16_t L0d_cmd1 (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)

L0 command which executes an L1 command.

L0 CMD1 command handler.

This handler also manages encryption and login token check

6.11.3 Variable Documentation

6.11.3.1 se3_cmd_func L1d_handlers[SE3_CMD1_MAX] [static]

Initial value:

```
= {
    NULL,
    L1d_challenge,
    L1d_login,
    L1d_logout,
    L1d_config,
    L1d_key_edit,
    L1d_key_list,
    L1d_crypto_init,
    L1d_crypto_update,
    L1d_crypto_list,
    L1d_crypto_set_time,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL
}
```

6.12 src/Device/se3_cmd1.h File Reference

L1 command dispatch and execute.

```
#include "se3c1.h"
```

Functions

- uint16_t [L0d_cmd1](#) (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)
L0 CMD1 command handler.

6.12.1 Detailed Description

L1 command dispatch and execute.

Author

Nicola Ferri

6.12.2 Function Documentation

6.12.2.1 uint16_t L0d_cmd1 (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)

L0 CMD1 command handler.

Execute a L1 command

L0 CMD1 command handler.

This handler also manages encryption and login token check

6.13 src/Device/se3_cmd1_config.c File Reference

L1 handlers for configuration record operations.

```
#include "se3_cmd1_config.h"
```

Functions

- uint16_t [L1d_config](#) (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)
set or get configuration record

6.13.1 Detailed Description

L1 handlers for configuration record operations.

Author

Nicola Ferri

6.13.2 Function Documentation

6.13.2.1 uint16_t L1d_config (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)

set or get configuration record

L1 CONFIG command handler.

config : (type:ui16, op:ui16, value[32]) => (value[32])

6.14 src/Device/se3_cmd1_config.h File Reference

L1 handlers for configuration record operations.

```
#include "se3c1.h"
```

Functions

- uint16_t [L1d_config](#) (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)
L1 CONFIG command handler.

6.14.1 Detailed Description

L1 handlers for configuration record operations.

Author

Nicola Ferri

6.14.2 Function Documentation

6.14.2.1 uint16_t [L1d_config](#) (uint16_t *req_size*, const uint8_t * *req*, uint16_t * *resp_size*, uint8_t * *resp*)

L1 CONFIG command handler.

Get or set a configuration record

L1 CONFIG command handler.

config : (type:ui16, op:ui16, value[32]) => (value[32])

6.15 src/Device/se3_cmd1_crypto.c File Reference

L1 handlers for crypto operations.

```
#include "se3_cmd1_crypto.h"
```

Functions

- uint16_t [L1d_crypto_init](#) (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)
initialize a crypto context
- uint16_t [L1d_crypto_update](#) (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)
use a crypto context
- uint16_t [L1d_crypto_set_time](#) (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)
set device time for key validity
- uint16_t [L1d_crypto_list](#) (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)
get list of available algorithms

6.15.1 Detailed Description

L1 handlers for crypto operations.

Author

Nicola Ferri

6.15.2 Function Documentation

6.15.2.1 `uint16_t L1d_crypto_init (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)`

initialize a crypto context

L1 CRYPTO_INIT handler.

`L1_crypto_init : (algo:ui16, mode:ui16, key_id:ui32) => (sid:ui32)`

6.15.2.2 `uint16_t L1d_crypto_list (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)`

get list of available algorithms

L1 CRYPTO_SET_TIME handler.

`crypto_list : () => (count:ui16, algoinfo0, algoinfo1, ...) algoinfo : (name[16], type:u16, block_size:u16, key_size:u16)`

6.15.2.3 `uint16_t L1d_crypto_set_time (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)`

set device time for key validity

L1 CRYPTO_SET_TIME handler.

`crypto_set_time : (devtime:ui32) => ()`

6.15.2.4 `uint16_t L1d_crypto_update (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)`

use a crypto context

L1 CRYPTO_UPDATE handler.

`L1_crypto_update : (sid:ui32, flags:ui16, datain1-len:ui16, datain2-len:ui16, pad-to-16[6], datain1[datain1-len], pad-to-16[...], datain2[datain2-len]) => (dataout-len, pad-to-16[14], dataout[dataout-len])`

6.16 src/Device/se3_cmd1_crypto.h File Reference

L1 handlers for crypto operations.

```
#include "se3c1.h"
#include "se3_keys.h"
```

Functions

- uint16_t [L1d_crypto_init](#) (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)
L1 CRYPTO_INIT handler.
- uint16_t [L1d_crypto_update](#) (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)
L1 CRYPTO_UPDATE handler.
- uint16_t [L1d_crypto_set_time](#) (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)
L1 CRYPTO_SET_TIME handler.
- uint16_t [L1d_crypto_list](#) (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)
L1 CRYPTO_SET_TIME handler.

6.16.1 Detailed Description

L1 handlers for crypto operations.

Author

Nicola Ferri

6.16.2 Function Documentation

6.16.2.1 uint16_t L1d_crypto_init (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)

L1 CRYPTO_INIT handler.

Initialize a cryptographic context

L1 CRYPTO_INIT handler.

L1_crypto_init : (algo:ui16, mode:ui16, key_id:ui32) => (sid:ui32)

6.16.2.2 uint16_t L1d_crypto_list (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)

L1 CRYPTO_SET_TIME handler.

Get list of available algorithms

L1 CRYPTO_SET_TIME handler.

crypto_list : () => (count:ui16, algoinfo0, algoinfo1, ...) algoinfo : (name[16], type:u16, block_size:u16, key_size↔:u16)

6.16.2.3 uint16_t L1d_crypto_set_time (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)

L1 CRYPTO_SET_TIME handler.

Set device time for key validity

L1 CRYPTO_SET_TIME handler.

crypto_set_time : (devtime:ui32) => ()

6.16.2.4 `uint16_t L1d_crypto_update (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)`

L1 CRYPTO_UPDATE handler.

Use a cryptographic context

L1 CRYPTO_UPDATE handler.

`L1_crypto_update : (sid:ui32, flags:ui16, datain1-len:ui16, datain2-len:ui16, pad-to-16[6], datain1[datain1-len], pad-to-16[...], datain2[datain2-len]) => (dataout-len, pad-to-16[14], dataout[dataout-len])`

6.17 src/Device/se3_cmd1_keys.c File Reference

L1 handlers for key management operations.

```
#include "se3_cmd1_keys.h"
#include "se3_keys.h"
```

Functions

- `uint16_t L1d_key_edit (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)`
insert, delete or update key
- `uint16_t L1d_key_list (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)`
list all keys in device

6.17.1 Detailed Description

L1 handlers for key management operations.

Author

Nicola Ferri

6.17.2 Function Documentation

6.17.2.1 `uint16_t L1d_key_edit (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)`

insert, delete or update key

L1 KEY_EDIT.

`key_edit : (op:ui16, id:ui32, validity:ui32, data-len:ui16, name-len:ui16, data[data-len], name[name-len]) => ()`

6.17.2.2 `uint16_t L1d_key_list (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)`

list all keys in device

L1 KEY_LIST.

key_list : (skip:ui16, nmax:ui16) => (count:ui16, keyinfo0, keyinfo1, ...) keyinfo: (id:ui32, validity:ui32, data-len:ui16, name-len:ui16, name[name-len])

6.18 src/Device/se3_cmd1_keys.h File Reference

L1 handlers for key management operations.

```
#include "se3c1.h"
#include "sha256.h"
#include "aes256.h"
```

Functions

- `uint16_t L1d_key_edit (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)`
L1 KEY_EDIT.
- `uint16_t L1d_key_list (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)`
L1 KEY_LIST.

6.18.1 Detailed Description

L1 handlers for key management operations.

Author

Nicola Ferri

6.18.2 Function Documentation

6.18.2.1 `uint16_t L1d_key_edit (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)`

L1 KEY_EDIT.

Insert, delete or update a key

L1 KEY_EDIT.

key_edit : (op:ui16, id:ui32, validity:ui32, data-len:ui16, name-len:ui16, data[data-len], name[name-len]) => ()

6.18.2.2 `uint16_t L1d_key_list (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)`

L1 KEY_LIST.

Get a list of keys in the device

L1 KEY_LIST.

`key_list` : (skip:ui16, nmax:ui16) => (count:ui16, keyinfo0, keyinfo1, ...) `keyinfo`: (id:ui32, validity:ui32, data-len:ui16, name-len:ui16, name[name-len])

6.19 src/Device/se3_cmd1_login.c File Reference

L1 handlers for login operations.

```
#include "se3_cmd1_login.h"
#include "se3_rand.h"
```

Functions

- `uint16_t L1d_challenge` (`uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp`)
Get a login challenge from the server.
- `uint16_t L1d_login` (`uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp`)
respond to challenge, completing login
- `uint16_t L1d_logout` (`uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp`)
Log out and release resources.

6.19.1 Detailed Description

L1 handlers for login operations.

Author

Nicola Ferri

6.19.2 Function Documentation

6.19.2.1 `uint16_t L1d_challenge (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)`

Get a login challenge from the server.

L1 CHALLENGE command handler.

`challenge` : (cc1[32], cc2[32], access:ui16) => (sc[32], sresp[32])

6.19.2.2 `uint16_t L1d_login (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)`

respond to challenge, completing login

L1 LOGIN command handler.

login : (cresp[32]) => (tok[16])

6.19.2.3 `uint16_t L1d_logout (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)`

Log out and release resources.

L1 LOGOUT command handler.

logout : () => ()

6.20 src/Device/se3_cmd1_login.h File Reference

L1 handlers for login operations.

```
#include "se3c1.h"
#include "sha256.h"
#include "aes256.h"
#include "pbkdf2.h"
```

Functions

- `uint16_t L1d_challenge` (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)
L1 CHALLENGE command handler.
- `uint16_t L1d_login` (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)
L1 LOGIN command handler.
- `uint16_t L1d_logout` (uint16_t req_size, const uint8_t *req, uint16_t *resp_size, uint8_t *resp)
L1 LOGOUT command handler.

6.20.1 Detailed Description

L1 handlers for login operations.

Author

Nicola Ferri

6.20.2 Function Documentation

6.20.2.1 `uint16_t L1d_challenge (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)`

L1 CHALLENGE command handler.

Get a login challenge from the device

L1 CHALLENGE command handler.

challenge : (cc1[32], cc2[32], access:ui16) => (sc[32], sresp[32])

6.20.2.2 `uint16_t L1d_login (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)`

L1 LOGIN command handler.

Respond to challenge and complete the login

L1 LOGIN command handler.

login : (cresp[32]) => (tok[16])

6.20.2.3 `uint16_t L1d_logout (uint16_t req_size, const uint8_t * req, uint16_t * resp_size, uint8_t * resp)`

L1 LOGOUT command handler.

Log out and release resources

L1 LOGOUT command handler.

logout : () => ()

6.21 src/Device/se3_flash.c File Reference

Flash management.

```
#include "se3_flash.h"
```

Functions

- static bool **flash_fill** (uint32_t addr, uint8_t val, size_t size)
- static bool **flash_zero** (uint32_t addr, size_t size)
- static bool **flash_program** (uint32_t addr, const uint8_t *data, size_t size)
- static bool **flash_erase** (uint32_t sector)
- static bool **flash_swap** ()
- void **se3_flash_info_setup** (uint32_t sector, const uint8_t *base)
Initialize flash structures.
- bool **se3_flash_canfit** (size_t size)
Check if enough space for new node.
- bool **se3_flash_init** ()
Initialize flash.
- bool **se3_flash_it_write** (se3_flash_it *it, uint16_t off, const uint8_t *data, uint16_t size)
Write to flash node.
- void **se3_flash_it_init** (se3_flash_it *it)
Initialize flash iterator.
- bool **se3_flash_it_next** (se3_flash_it *it)
Increment flash iterator.
- size_t **se3_flash_unused** ()
Get unused space.
- bool **se3_flash_it_new** (se3_flash_it *it, uint8_t type, uint16_t size)
Allocate new node.
- bool **se3_flash_pos_delete** (size_t pos)
Delete flash node by index.
- bool **se3_flash_it_delete** (se3_flash_it *it)
Delete flash node.

6.21.1 Detailed Description

Flash management.

Author

Nicola Ferri

6.21.2 Function Documentation

6.21.2.1 bool se3_flash_canfit (size_t size)

Check if enough space for new node.

Check if there is enough space

Parameters

<i>size</i>	size of the data to be stored inside the new node
-------------	---

Returns

true if the node will fit into the flash, else false

6.21.2.2 void se3_flash_info_setup (uint32_t *sector*, const uint8_t * *base*)

Initialize flash structures.

Initializes the structures for flash management, selecting a sector and its base address.

Parameters

<i>sector</i>	active sector number
<i>base</i>	active sector base address

6.21.2.3 bool se3_flash_init ()

Initialize flash.

Selects the active flash sector or initializes one

6.21.2.4 bool se3_flash_it_delete (se3_flash_it * *it*)

Delete flash node.

Delete a flash node and its data

Remarks

if a flash operation fails, the hwerror flag (se3c0.hwerror) is set.

Parameters

<i>it</i>	flash iterator structure
-----------	--------------------------

Returns

true on success, else false

6.21.2.5 void se3_flash_it_init (se3_flash_it * *it*)

Initialize flash iterator.

Parameters

<i>it</i>	flash iterator structure
-----------	--------------------------

6.21.2.6 `bool se3_flash_it_new (se3_flash_it * it, uint8_t type, uint16_t size)`

Allocate new node.

Allocates a new node in the flash and points the iterator to the new node.

Remarks

if a flash operation fails, the hwerror flag (se3c0.hwerror) is set.

Parameters

<i>it</i>	flash iterator structure
<i>type</i>	type of the new flash node
<i>size</i>	size of the data in the new flash node

Returns

true if the function succeeds, false if there is no more space, or a flash operation fails

6.21.2.7 `bool se3_flash_it_next (se3_flash_it * it)`

Increment flash iterator.

Increment iterator and read information of the next node in flash

Parameters

<i>it</i>	flash iterator structure
-----------	--------------------------

Returns

false if end of iteration, else true

6.21.2.8 `bool se3_flash_it_write (se3_flash_it * it, uint16_t off, const uint8_t * data, uint16_t size)`

Write to flash node.

Write data to flash node.

Remarks

if a flash operation fails, the hwerror flag (se3c0.hwerror) is set.

Parameters

<i>it</i>	flash iterator structure
<i>off</i>	offset of data
<i>data</i>	pointer to data to be written
<i>size</i>	size of data to be written

6.21.2.9 bool se3_flash_pos_delete (size_t pos)

Delete flash node by index.

Delete a flash node given its index

Remarks

if a flash operation fails, the hwerror flag (se3c0.hwerror) is set.

Parameters

<i>pos</i>	the index of the node
------------	-----------------------

Returns

true on success, else false

6.21.2.10 size_t se3_flash_unused ()

Get unused space.

Get unused space in the flash memory, including the space marked as invalid. If space is available, it does not mean that flash sectors will not be swapped.

Returns

unused space in bytes

6.22 src/Device/se3_flash.h File Reference

Flash management.

```
#include "se3c0.h"
#include "stm32f4xx.h"
#include "stm32f4xx_hal.h"
```

Data Structures

- struct [se3_flash_it_](#)
Flash node iterator structure.

Macros

- #define **SE3_FLASH_S0** (FLASH_SECTOR_10)
- #define **SE3_FLASH_S1** (FLASH_SECTOR_11)
- #define **SE3_FLASH_S0_ADDR** ((uint32_t)0x080C0000)
- #define **SE3_FLASH_S1_ADDR** ((uint32_t)0x080E0000)
- #define **SE3_FLASH_SECTOR_SIZE** (128*1024)

Typedefs

- typedef struct [se3_flash_it](#) [se3_flash_it](#)
Flash node iterator structure.

Enumerations

- enum { [SE3_FLASH_TYPE_INVALID](#) = 0, [SE3_FLASH_TYPE_SERIAL](#) = 1, [SE3_FLASH_TYPE_CONT](#) = 0xFE, [SE3_FLASH_TYPE_EMPTY](#) = 0xFF }
- enum {
[SE3_FLASH_MAGIC_SIZE](#) = 32, [SE3_FLASH_INDEX_SIZE](#) = 2016, [SE3_FLASH_BLOCK_SIZE](#) = 64,
[SE3_FLASH_NODE_MAX](#) = (4 * 1024),
[SE3_FLASH_NODE_DATA_MAX](#) = (SE3_FLASH_NODE_MAX - 2) }

Functions

- bool [se3_flash_init](#) ()
Initialize flash.
- void [se3_flash_it_init](#) ([se3_flash_it](#) *it)
Initialize flash iterator.
- bool [se3_flash_it_next](#) ([se3_flash_it](#) *it)
Increment flash iterator.
- bool [se3_flash_it_new](#) ([se3_flash_it](#) *it, uint8_t type, uint16_t size)
Allocate new node.
- bool [se3_flash_it_write](#) ([se3_flash_it](#) *it, uint16_t off, const uint8_t *data, uint16_t size)
Write to flash node.
- bool [se3_flash_it_delete](#) ([se3_flash_it](#) *it)
Delete flash node.
- bool [se3_flash_pos_delete](#) (size_t pos)
Delete flash node by index.
- size_t [se3_flash_unused](#) ()
Get unused space.
- bool [se3_flash_canfit](#) (size_t size)
Check if enough space for new node.
- void [se3_flash_info_setup](#) (uint32_t sector, const uint8_t *base)
Initialize flash structures.

6.22.1 Detailed Description

Flash management.

Author

Nicola Ferri

6.22.2 Enumeration Type Documentation

6.22.2.1 anonymous enum

Flash nodes' default and reserved types

Enumerator

SE3_FLASH_TYPE_INVALID Invalid node.
SE3_FLASH_TYPE_SERIAL Device's serial number.
SE3_FLASH_TYPE_CONT Continuation of previous node.
SE3_FLASH_TYPE_EMPTY Not written yet.

6.22.2.2 anonymous enum

Flash fields sizes

6.22.3 Function Documentation

6.22.3.1 bool se3_flash_canfit (size_t size)

Check if enough space for new node.

Check if there is enough space

Parameters

<i>size</i>	size of the data to be stored inside the new node
-------------	---

Returns

true if the node will fit into the flash, else false

6.22.3.2 void se3_flash_info_setup (uint32_t sector, const uint8_t * base)

Initialize flash structures.

Initializes the structures for flash management, selecting a sector and its base address.

Parameters

<i>sector</i>	active sector number
<i>base</i>	active sector base address

6.22.3.3 `bool se3_flash_init ()`

Initialize flash.

Selects the active flash sector or initializes one

6.22.3.4 `bool se3_flash_it_delete (se3_flash_it * it)`

Delete flash node.

Delete a flash node and its data

Remarks

if a flash operation fails, the hwerror flag (se3c0.hwerror) is set.

Parameters

<i>it</i>	flash iterator structure
-----------	--------------------------

Returns

true on success, else false

6.22.3.5 `void se3_flash_it_init (se3_flash_it * it)`

Initialize flash iterator.

Parameters

<i>it</i>	flash iterator structure
-----------	--------------------------

6.22.3.6 `bool se3_flash_it_new (se3_flash_it * it, uint8_t type, uint16_t size)`

Allocate new node.

Allocates a new node in the flash and points the iterator to the new node.

Remarks

if a flash operation fails, the hwerror flag (se3c0.hwerror) is set.

Parameters

<i>it</i>	flash iterator structure
<i>type</i>	type of the new flash node
<i>size</i>	size of the data in the new flash node

Returns

true if the function succeeds, false if there is no more space, or a flash operation fails

6.22.3.7 `bool se3_flash_it_next (se3_flash_it * it)`

Increment flash iterator.

Increment iterator and read information of the next node in flash

Parameters

<i>it</i>	flash iterator structure
-----------	--------------------------

Returns

false if end of iteration, else true

6.22.3.8 `bool se3_flash_it_write (se3_flash_it * it, uint16_t off, const uint8_t * data, uint16_t size)`

Write to flash node.

Write data to flash node.

Remarks

if a flash operation fails, the hwerror flag (se3c0.hwerror) is set.

Parameters

<i>it</i>	flash iterator structure
<i>off</i>	offset of data
<i>data</i>	pointer to data to be written
<i>size</i>	size of data to be written

6.22.3.9 `bool se3_flash_pos_delete (size_t pos)`

Delete flash node by index.

Delete a flash node given its index

Remarks

if a flash operation fails, the hwerror flag (se3c0.hwerror) is set.

Parameters

<i>pos</i>	the index of the node
------------	-----------------------

Returns

true on success, else false

6.22.3.10 size_t se3_flash_unused ()

Get unused space.

Get unused space in the flash memory, including the space marked as invalid. If space is available, it does not mean that flash sectors will not be swapped.

Returns

unused space in bytes

6.23 src/Device/se3_keys.c File Reference

Key management.

```
#include "se3_keys.h"
```

Enumerations

- enum {
SE3_KEY_OFFSET_ID = 0, **SE3_KEY_OFFSET_VALIDITY** = 4, **SE3_KEY_OFFSET_DATALEN** = 8, **SE3_KEY_OFFSET_NAMELEN** = 10,
SE3_KEY_OFFSET_DATA = 12 }

Functions

- bool [se3_key_find](#) (uint32_t id, [se3_flash_it](#) *it)
Find a key.
- bool [se3_key_remove](#) ([se3_flash_it](#) *it)
Remove a key.
- bool [se3_key_new](#) ([se3_flash_it](#) *it, [se3_flash_key](#) *key)
Add a new key.
- void [se3_key_read](#) ([se3_flash_it](#) *it, [se3_flash_key](#) *key)
Read a key.
- bool [se3_key_equal](#) ([se3_flash_it](#) *it, [se3_flash_key](#) *key)
Check if key is equal.
- void [se3_key_read_data](#) ([se3_flash_it](#) *it, uint16_t data_size, uint8_t *data)
Read data from key node.
- bool [se3_key_write](#) ([se3_flash_it](#) *it, [se3_flash_key](#) *key)
Write key data.

6.23.1 Detailed Description

Key management.

Author

Nicola Ferri

6.23.2 Function Documentation

6.23.2.1 `bool se3_key_equal (se3_flash_it * it, se3_flash_key * key)`

Check if key is equal.

Check if the supplied key is equal to a key stored in the flash.

Parameters

<i>it</i>	a flash iterator pointing to a key
<i>key</i>	a flash key structure to compare

Returns

true if equal, else false

6.23.2.2 `bool se3_key_find (uint32_t id, se3_flash_it * it)`

Find a key.

Find a key in the flash memory

Parameters

<i>id</i>	identifier of the key
<i>it</i>	a flash iterator that will be set to the key's position

Returns

true on success

6.23.2.3 `bool se3_key_new (se3_flash_it * it, se3_flash_key * key)`

Add a new key.

Create a new node with the necessary amount of space for the key, then write the key.

Remarks

if a flash operation fails, the hwerror flag (se3c0.hwerror) is set.

Parameters

<i>it</i>	a flash iterator which will receive the position of the new node
<i>key</i>	a flash key structure containing the key information The data and name fields must point to a valid memory region, unless their size (data_size, name_size) is zero.

Returns

true on success, else false

6.23.2.4 void se3_key_read (se3_flash_it * *it*, se3_flash_key * *key*)

Read a key.

Read a key from a flash node

Parameters

<i>it</i>	a flash iterator pointing to the key
<i>key</i>	a flash key structure which will receive the key's information. The data and name fields will be filled only if not NULL.

6.23.2.5 void se3_key_read_data (se3_flash_it * *it*, uint16_t *data_size*, uint8_t * *data*)

Read data from key node.

Read the key's data from a ket node

Parameters

<i>it</i>	a flash iterator pointing to the key
<i>data_size</i>	the number of bytes to read
<i>data</i>	output data buffer

6.23.2.6 bool se3_key_remove (se3_flash_it * *it*)

Remove a key.

Delete a key from the flash

Remarks

if a flash operation fails, the hwerror flag (se3c0.hwerror) is set.

Parameters

<i>it</i>	a flash iterator pointing to the key
-----------	--------------------------------------

Returns

true on success

6.23.2.7 `bool se3_key_write (se3_flash_it * it, se3_flash_key * key)`

Write key data.

Write key data to a flash node

Remarks

if a flash operation fails, the hwerror flag (se3c0.hwerror) is set.

Parameters

<i>it</i>	a flash iterator pointing to a newly created flash node of key type
<i>key</i>	a flash key structure containing the key information The data and name fields must point to a valid memory region, unless their size (data_size, name_size) is zero.

Returns

true on success, else false

6.24 src/Device/se3_keys.h File Reference

Key management.

```
#include "se3_flash.h"
```

Data Structures

- struct [se3_flash_key_](#)
Flash key structure.

Macros

- #define **SE3_TYPE_KEY** 100

Typedefs

- typedef struct [se3_flash_key](#) [se3_flash_key](#)
Flash key structure.

Enumerations

- enum {
SE3_FLASH_KEY_OFF_ID = 0, **SE3_FLASH_KEY_OFF_VALIDITY** = 4, **SE3_FLASH_KEY_OFF_DATA_LEN** = 8, **SE3_FLASH_KEY_OFF_NAME_LEN** = 10,
SE3_FLASH_KEY_OFF_NAME_AND_DATA = 12, **SE3_FLASH_KEY_SIZE_HEADER** = SE3_FLASH_KEY_OFF_NAME_AND_DATA }

Functions

- bool [se3_key_find](#) (uint32_t id, [se3_flash_it](#) *it)
Find a key.
- bool [se3_key_remove](#) ([se3_flash_it](#) *it)
Remove a key.
- bool [se3_key_new](#) ([se3_flash_it](#) *it, [se3_flash_key](#) *key)
Add a new key.
- void [se3_key_read](#) ([se3_flash_it](#) *it, [se3_flash_key](#) *key)
Read a key.
- bool [se3_key_equal](#) ([se3_flash_it](#) *it, [se3_flash_key](#) *key)
Check if key is equal.
- void [se3_key_read_data](#) ([se3_flash_it](#) *it, uint16_t data_size, uint8_t *data)
Read data from key node.
- bool [se3_key_write](#) ([se3_flash_it](#) *it, [se3_flash_key](#) *key)
Write key data.

6.24.1 Detailed Description

Key management.

Author

Nicola Ferri

6.24.2 Typedef Documentation

6.24.2.1 typedef struct [se3_flash_key](#) [se3_flash_key](#)

Flash key structure.

Disposition of the fields within the flash node: 0:3 id 4:7 validity 8:9 data_size 10:11 name_size 12:(12+data_size-1) data (12+data_size):(12+data_size+name_size-1) name

6.24.3 Enumeration Type Documentation

6.24.3.1 anonymous enum

Flash key fields

6.24.4 Function Documentation

6.24.4.1 `bool se3_key_equal (se3_flash_it * it, se3_flash_key * key)`

Check if key is equal.

Check if the supplied key is equal to a key stored in the flash.

Parameters

<i>it</i>	a flash iterator pointing to a key
<i>key</i>	a flash key structure to compare

Returns

true if equal, else false

6.24.4.2 `bool se3_key_find (uint32_t id, se3_flash_it * it)`

Find a key.

Find a key in the flash memory

Parameters

<i>id</i>	identifier of the key
<i>it</i>	a flash iterator that will be set to the key's position

Returns

true on success

6.24.4.3 `bool se3_key_new (se3_flash_it * it, se3_flash_key * key)`

Add a new key.

Create a new node with the necessary amount of space for the key, then write the key.

Remarks

if a flash operation fails, the hwerror flag (se3c0.hwerror) is set.

Parameters

<i>it</i>	a flash iterator which will receive the position of the new node
<i>key</i>	a flash key structure containing the key information The data and name fields must point to a valid memory region, unless their size (data_size, name_size) is zero.

Returns

true on success, else false

6.24.4.4 void se3_key_read (se3_flash_it * *it*, se3_flash_key * *key*)

Read a key.

Read a key from a flash node

Parameters

<i>it</i>	a flash iterator pointing to the key
<i>key</i>	a flash key structure which will receive the key's information. The data and name fields will be filled only if not NULL.

6.24.4.5 void se3_key_read_data (se3_flash_it * *it*, uint16_t *data_size*, uint8_t * *data*)

Read data from key node.

Read the key's data from a ket node

Parameters

<i>it</i>	a flash iterator pointing to the key
<i>data_size</i>	the number of bytes to read
<i>data</i>	output data buffer

6.24.4.6 bool se3_key_remove (se3_flash_it * *it*)

Remove a key.

Delete a key from the flash

Remarks

if a flash operation fails, the hwerror flag (se3c0.hwerror) is set.

Parameters

<i>it</i>	a flash iterator pointing to the key
-----------	--------------------------------------

Returns

true on success

6.24.4.7 `bool se3_key_write (se3_flash_it * it, se3_flash_key * key)`

Write key data.

Write key data to a flash node

Remarks

if a flash operation fails, the hwerror flag (se3c0.hwerror) is set.

Parameters

<i>it</i>	a flash iterator pointing to a newly created flash node of key type
<i>key</i>	a flash key structure containing the key information The data and name fields must point to a valid memory region, unless their size (data_size, name_size) is zero.

Returns

true on success, else false

6.25 src/Device/se3_memory.c File Reference

Memory management (session allocator)

```
#include "se3_memory.h"
```

Macros

- `#define SE3_MEM_SIZE_GET(x, val) SE3_GET16(x, 0, (val))`
- `#define SE3_MEM_SIZE_SET(x, val) SE3_SET16(x, 0, (val))`
- `#define SE3_MEM_INFO_GET(x, info) SE3_GET16(x, 2, info)`
- `#define SE3_MEM_INFO_SET(x, info) SE3_SET16(x, 2, info)`
- `#define SE3_MEM_INFO_MAKE(id, valid) ((id) | ((valid)?(1):(0)) << 15)`
- `#define SE3_MEM_INFO_ISVALID(info) SE3_BIT_TEST(info, 15)`
- `#define SE3_MEM_INFO_ID(info) ((info) & ~(1<<15))`

Functions

- void `se3_mem_reset` (`se3_mem` *mem)
release all entries
- void `se3_mem_init` (`se3_mem` *mem, `size_t` index_size, `uint8_t` **index, `size_t` buf_size, `uint8_t` *buf)
initialize memory allocator
- static void `se3_mem_compact` (`uint8_t` *p, `uint8_t` *end)
- static `uint8_t` * `se3_mem_defrag` (`se3_mem` *mem)
- `int32_t` `se3_mem_alloc` (`se3_mem` *mem, `size_t` size)
allocate one entry
- `uint8_t` * `se3_mem_ptr` (`se3_mem` *mem, `int32_t` id)
get pointer to entry in buffer
- void `se3_mem_free` (`se3_mem` *mem, `int32_t` id)
release single entry

6.25.1 Detailed Description

Memory management (session allocator)

Author

Nicola Ferri

6.25.2 Function Documentation

6.25.2.1 `int32_t se3_mem_alloc (se3_mem * mem, size_t size)`

allocate one entry

Parameters

<i>mem</i>	memory buffer object
<i>size</i>	allocation size

6.25.2.2 `void se3_mem_free (se3_mem * mem, int32_t id)`

release single entry

Parameters

<i>mem</i>	memory buffer object
<i>id</i>	of the entry

6.25.2.3 `void se3_mem_init (se3_mem * mem, size_t index_size, uint8_t ** index, size_t buf_size, uint8_t * buf)`

initialize memory allocator

Parameters

<i>mem</i>	memory buffer object
<i>index_size</i>	number of elements in index
<i>index</i>	pointer to the index buffer (array[index_size] of pointers)
<i>buf_size</i>	number of bytes in data buffer
<i>buf</i>	pointer to data buffer

6.25.2.4 `uint8_t* se3_mem_ptr (se3_mem * mem, int32_t id)`

get pointer to entry in buffer

Parameters

<i>mem</i>	memory buffer object
<i>id</i>	of the entry

6.25.2.5 `void se3_mem_reset (se3_mem * mem)`

release all entries

Parameters

<i>mem</i>	memory buffer object
------------	----------------------

6.26 src/Device/se3_memory.h File Reference

Memory management (session allocator)

```
#include "se3c0.h"
```

Data Structures

- struct [se3_mem_](#)
memory allocator structure

Typedefs

- typedef struct [se3_mem_ se3_mem](#)
memory allocator structure

Enumerations

- enum { `SE3_MEM_HEADER` = 4, `SE3_MEM_BLOCK` = 32 }

Functions

- void `se3_mem_init` (`se3_mem` *mem, size_t index_size, uint8_t **index, size_t buf_size, uint8_t *buf)
initialize memory allocator
- int32_t `se3_mem_alloc` (`se3_mem` *mem, size_t size)
allocate one entry
- uint8_t * `se3_mem_ptr` (`se3_mem` *mem, int32_t id)
get pointer to entry in buffer
- void `se3_mem_free` (`se3_mem` *mem, int32_t id)
release single entry
- void `se3_mem_reset` (`se3_mem` *mem)
release all entries

6.26.1 Detailed Description

Memory management (session allocator)

Author

Nicola Ferri

6.26.2 Enumeration Type Documentation

6.26.2.1 anonymous enum

Enumerator

`SE3_MEM_HEADER` entry header size

`SE3_MEM_BLOCK` memory alignment

6.26.3 Function Documentation

6.26.3.1 int32_t se3_mem_alloc (se3_mem * mem, size_t size)

allocate one entry

Parameters

<i>mem</i>	memory buffer object
<i>size</i>	allocation size

6.26.3.2 void se3_mem_free (se3_mem * mem, int32_t id)

release single entry

Parameters

<i>mem</i>	memory buffer object
<i>id</i>	of the entry

6.26.3.3 void se3_mem_init (se3_mem * mem, size_t index_size, uint8_t ** index, size_t buf_size, uint8_t * buf)

initialize memory allocator

Parameters

<i>mem</i>	memory buffer object
<i>index_size</i>	number of elements in index
<i>index</i>	pointer to the index buffer (array[index_size] of pointers)
<i>buf_size</i>	number of bytes in data buffer
<i>buf</i>	pointer to data buffer

6.26.3.4 uint8_t* se3_mem_ptr (se3_mem * mem, int32_t id)

get pointer to entry in buffer

Parameters

<i>mem</i>	memory buffer object
<i>id</i>	of the entry

6.26.3.5 void se3_mem_reset (se3_mem * mem)

release all entries

Parameters

<i>mem</i>	memory buffer object
------------	----------------------

6.27 src/Device/se3_proto.c File Reference

USB read/write handlers.

```
#include "se3_proto.h"
#include <se3_sdio.h>
```

Data Structures

- struct [s3_storage_range_](#)
SDIO read/write request buffer context.

Typedefs

- typedef struct [s3_storage_range_s3_storage_range](#)
SDIO read/write request buffer context.

Enumerations

- enum [s3_storage_range_direction](#) { [range_write](#), [range_read](#) }

Functions

- static bool [block_is_magic](#) (const uint8_t *buf)
Check if block contains the magic sequence.
- static int [find_magic_index](#) (uint32_t block)
Check if block belongs to the special protocol file.
- static int32_t [se3_storage_range_add](#) ([s3_storage_range](#) *range, uint8_t lun, uint8_t *buf, uint32_t block, enum [s3_storage_range_direction](#) direction)
add request to SDIO read/write buffer
- void [se3_proto_request_reset](#) ()
Reset protocol request buffer.
- static void [handle_req_rcv](#) (int index, const uint8_t *blockdata)
Handle request for incoming protocol block.
- int32_t [se3_proto_rcv](#) (uint8_t lun, const uint8_t *buf, uint32_t blk_addr, uint16_t blk_len)
USB data receive handler.
- static void [handle_resp_send](#) (int index, uint8_t *blockdata)
Handle request for outgoing protocol block.
- int32_t [se3_proto_send](#) (uint8_t lun, uint8_t *buf, uint32_t blk_addr, uint16_t blk_len)
USB data send handler.

6.27.1 Detailed Description

USB read/write handlers.

Author

Nicola Ferri

6.27.2 Function Documentation

6.27.2.1 static bool [block_is_magic](#) (const uint8_t * *buf*) [static]

Check if block contains the magic sequence.

Parameters

<i>buf</i>	pointer to block data
------------	-----------------------

Returns

true if the block contains the magic sequence, otherwise false

Check if a block of data contains the magic sequence, used to initialize the special protocol file.

6.27.2.2 static int find_magic_index (uint32_t *block*) [static]

Check if block belongs to the special protocol file.

Parameters

<i>block</i>	block number
--------------	--------------

Returns

the index of the corresponding protocol file block, or -1 if the block does not belong to the protocol file.

The special protocol file is made up of multiple blocks. Each block is mapped to a block on the physical storage

6.27.2.3 static void handle_req_rcv (int *index*, const uint8_t* *blockdata*) [static]

Handle request for incoming protocol block.

Parameters

<i>index</i>	index of block in the special protocol file
<i>blockdata</i>	data

Handle a single block belonging to a protocol request. The data is stored in the request buffer. As soon as the request data is received completely, the device will start processing the request

6.27.2.4 static void handle_resp_send (int *index*, uint8_t* *blockdata*) [static]

Handle request for outgoing protocol block.

Parameters

<i>index</i>	index of block in the special protocol file
<i>blockdata</i>	output data

Output a single block of a protocol response. If the response is ready, the data is taken from the response buffer. Otherwise the 'not ready' state is returned.

6.27.2.5 `int32_t se3_proto_rcv (uint8_t lun, const uint8_t * buf, uint32_t blk_addr, uint16_t blk_len)`

USB data receive handler.

SEcube API requests are filtered and data is stored in the request buffer. The function also takes care of the initialization of the special protocol file. Other requests are passed to the SDIO interface.

6.27.2.6 `void se3_proto_request_reset ()`

Reset protocol request buffer.

Reset the protocol request buffer, making the device ready for a new request.

6.27.2.7 `int32_t se3_proto_send (uint8_t lun, uint8_t * buf, uint32_t blk_addr, uint16_t blk_len)`

USB data send handler.

SEcube API requests are filtered and data is sent from the response buffer. Other requests are passed to the SDIO interface.

6.27.2.8 `static int32_t se3_storage_range_add (s3_storage_range * range, uint8_t lun, uint8_t * buf, uint32_t block, enum s3_storage_range_direction direction) [static]`

add request to SDIO read/write buffer

Parameters

<i>range</i>	context; the count field must be initialized to zero on first usage
<i>lun</i>	parameter from USB handler
<i>buf</i>	pointer to request data
<i>block</i>	request block index
<i>direction</i>	read or write

Contiguous requests are processed with a single call to the SDIO interface, as soon as a non-contiguous request is added.

6.28 src/Device/se3_proto.h File Reference

USB read/write handlers.

```
#include "se3c0.h"
#include "se3_common.h"
```

Enumerations

- enum { [SE3_PROTO_OK](#) = 0, [SE3_PROTO_FAIL](#) = 1, [SE3_PROTO_BUSY](#) = 2 }

Functions

- int32_t [se3_proto_rcv](#) (uint8_t lun, const uint8_t *buf, uint32_t blk_addr, uint16_t blk_len)
USB data receive handler.
- int32_t [se3_proto_send](#) (uint8_t lun, uint8_t *buf, uint32_t blk_addr, uint16_t blk_len)
USB data send handler.

6.28.1 Detailed Description

USB read/write handlers.

Author

Nicola Ferri

6.28.2 Enumeration Type Documentation

6.28.2.1 anonymous enum

USB data handlers return values

Enumerator

- [SE3_PROTO_OK](#)** Report OK to the USB HAL.
[SE3_PROTO_FAIL](#) Report FAIL to the USB HAL.
[SE3_PROTO_BUSY](#) Report BUSY to the USB HAL.

6.28.3 Function Documentation

6.28.3.1 int32_t se3_proto_rcv (uint8_t lun, const uint8_t * buf, uint32_t blk_addr, uint16_t blk_len)

USB data receive handler.

SEcube API requests are filtered and data is stored in the request buffer. The function also takes care of the initialization of the special protocol file. Other requests are passed to the SDIO interface.

6.28.3.2 int32_t se3_proto_send (uint8_t lun, uint8_t * buf, uint32_t blk_addr, uint16_t blk_len)

USB data send handler.

SEcube API requests are filtered and data is sent from the response buffer Other requests are passed to the SDIO interface.

6.29 src/Device/se3c0.c File Reference

L0 structures and functions.

```
#include "se3c0.h"
```

Functions

- void **se3c0_init** ()
- uint64_t **se3c0_time_get** ()
- void **se3c0_time_set** (uint64_t t)
- void **se3c0_time_inc** ()

Variables

- [SE3_L0_GLOBALS](#) **se3c0**
- uint8_t **se3_comm_request_buffer** [SE3_COMM_N * SE3_COMM_BLOCK]
- uint8_t **se3_comm_response_buffer** [SE3_COMM_N * SE3_COMM_BLOCK]
- const uint8_t **se3_hello** [SE3_HELLO_SIZE]

6.29.1 Detailed Description

L0 structures and functions.

Author

Nicola Ferri

6.29.2 Variable Documentation

6.29.2.1 const uint8_t se3_hello[SE3_HELLO_SIZE]

Initial value:

```
= {
    'H', 'e', 'l', 'l', 'o', ' ', 'S', 'E',
    'c', 'u', 'b', 'e', 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0
}
```

6.30 src/Device/se3c0.h File Reference

L0 structures and functions.

```
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include "sha256.h"
#include "aes256.h"
#include "se3c0def.h"
#include "se3_common.h"
```

Data Structures

- struct [SE3_COMM_STATUS_](#)
structure holding host-device communication status and buffers
- struct [SE3_FLASH_INFO_](#)
Flash management structure.
- struct [SE3_SERIAL_](#)
serial number data and state
- struct [se3c0_req_header_](#)
decoded request header
- struct [se3c0_resp_header_](#)
response header to be encoded
- struct [SE3_L0_GLOBALS_](#)
L0 globals structure.

Macros

- `#define SE3_ALIGN_16`
- `#define SE3_BMAP_MAKE(n) ((uint32_t)(0xFFFFFFFF >> (32 - (n))))`

Typedefs

- typedef struct [SE3_COMM_STATUS_ SE3_COMM_STATUS](#)
structure holding host-device communication status and buffers
- typedef struct [SE3_FLASH_INFO_ SE3_FLASH_INFO](#)
Flash management structure.
- typedef struct [SE3_SERIAL_ SE3_SERIAL](#)
serial number data and state
- typedef struct [se3c0_req_header_ se3c0_req_header](#)
decoded request header
- typedef struct [se3c0_resp_header_ se3c0_resp_header](#)
response header to be encoded
- typedef uint16_t(* [se3_cmd_func](#)) (uint16_t, const uint8_t *, uint16_t *, uint8_t *)
- typedef struct [SE3_L0_GLOBALS_ SE3_L0_GLOBALS](#)
L0 globals structure.

Functions

- void [se3c0_init](#) ()
- uint64_t [se3c0_time_get](#) ()
- void [se3c0_time_set](#) (uint64_t t)
- void [se3c0_time_inc](#) ()

Variables

- const uint8_t [se3_hello](#) [SE3_HELLO_SIZE]
- uint8_t [se3_comm_request_buffer](#) [SE3_COMM_N *SE3_COMM_BLOCK]
- uint8_t [se3_comm_response_buffer](#) [SE3_COMM_N *SE3_COMM_BLOCK]
- [SE3_L0_GLOBALS se3c0](#)

6.30.1 Detailed Description

L0 structures and functions.

Author

Nicola Ferri

6.30.2 Typedef Documentation

6.30.2.1 typedef uint16_t(* se3_cmd_func)(uint16_t, const uint8_t *, uint16_t *, uint8_t *)

L0 command handler

6.30.2.2 typedef struct SE3_COMM_STATUS SE3_COMM_STATUS

structure holding host-device communication status and buffers

req_ready and resp_ready must be volatile, otherwise -O3 optimization will not work.

6.31 src/Device/se3c1.c File Reference

L1 structures and functions.

```
#include "se3c1.h"
#include "se3_flash.h"
#include "se3_cmd1.h"
#include "se3_algo_Aes.h"
#include "se3_algo_sha256.h"
#include "se3_algo_HmacSha256.h"
#include "se3_algo_AesHmacSha256s.h"
#include "se3_algo_aes256hmacsha256.h"
```

Functions

- static bool **se3c1_record_find**(uint16_t record_type, [se3_flash_it](#) *it)
- bool [se3c1_record_set](#)(uint16_t type, const uint8_t *data)
Write record.
- bool [se3c1_record_get](#)(uint16_t type, uint8_t *data)
Read record.
- void [se3c1_login_cleanup](#)()
Clear login session data.
- void [se3c1_init](#)()
Initialize L1 structures.

Variables

- `uint8_t se3_sessions_buf` [SE3_SESSIONS_BUF]
- `uint8_t * se3_sessions_index` [SE3_SESSIONS_MAX]
- `SE3_L1_GLOBALS se3c1`
L1 globals.
- `se3_algo_descriptor L1d_algo_table` [SE3_ALGO_MAX]

6.31.1 Detailed Description

L1 structures and functions.

Author

Nicola Ferri

6.31.2 Function Documentation

6.31.2.1 `void se3c1_login_cleanup ()`

Clear login session data.

Cleans all data associated with the login session, making SEcube ready for a new login.

6.31.2.2 `bool se3c1_record_get (uint16_t type, uint8_t * data)`

Read record.

Get data of a record.

Parameters

<i>type</i>	type of record
<i>data</i>	output buffer

Returns

true on success; false if the record does not exist or has never been written

6.31.2.3 `bool se3c1_record_set (uint16_t type, const uint8_t * data)`

Write record.

Set data of a record

Remarks

if a flash operation fails, the hwerror flag (se3c0.hwerror) is set.

Parameters

<i>type</i>	type of record
<i>data</i>	new data to be written to record

Returns

true on success; false if the record does not exist

6.31.3 Variable Documentation**6.31.3.1 `se3_algo_descriptor` `L1d_algo_table[SE3_ALGO_MAX]`**

algorithm description table

6.31.3.2 `uint8_t se3_sessions_buf[SE3_SESSIONS_BUF]`

session buffer

6.31.3.3 `uint8_t* se3_sessions_index[SE3_SESSIONS_MAX]`

session index

6.32 `src/Device/se3c1.h` File Reference

L1 structures and functions.

```
#include "se3c0.h"
#include "se3c1def.h"
#include "se3_memory.h"
#include "se3_keys.h"
```

Data Structures

- struct [SE3_RECORD_INFO_](#)
Record information.
- struct [se3_algo_descriptor_](#)
algorithm descriptor type
- struct [SE3_LOGIN_STATUS_](#)
L1 login status data.
- struct [SE3_L1_GLOBALS_](#)
L1 globals structure.

Typedefs

- typedef struct [SE3_RECORD_INFO_ SE3_RECORD_INFO](#)
Record information.
- typedef uint16_t(* [se3_crypto_init_handler](#)) (se3_flash_key *key, uint16_t mode, uint8_t *ctx)
L1_crypto_init function type.
- typedef uint16_t(* [se3_crypto_update_handler](#)) (uint8_t *ctx, uint16_t flags, uint16_t datain1_len, const uint8_t *datain1, uint16_t datain2_len, const uint8_t *datain2, uint16_t *dataout_len, uint8_t *dataout)
L1_crypto_update function type.
- typedef struct [se3_algo_descriptor_ se3_algo_descriptor](#)
algorithm descriptor type
- typedef struct [SE3_LOGIN_STATUS_ SE3_LOGIN_STATUS](#)
L1 login status data.
- typedef struct [SE3_L1_GLOBALS_ SE3_L1_GLOBALS](#)
L1 globals structure.

Enumerations

- enum { [SE3_FLASH_TYPE_RECORD](#) = 0xF0 }
- enum { [SE3_RECORD_SIZE_TYPE](#) = 2, [SE3_RECORD_OFFSET_TYPE](#) = 0, [SE3_RECORD_OFFSET_↔DATA](#) = 2 }
- enum { [SE3_SESSIONS_BUF](#) = (32*1024), [SE3_SESSIONS_MAX](#) = 100 }

Functions

- bool [se3c1_record_set](#) (uint16_t type, const uint8_t *data)
Write record.
- bool [se3c1_record_get](#) (uint16_t type, uint8_t *data)
Read record.
- void [se3c1_login_cleanup](#) ()
Clear login session data.
- void [se3c1_init](#) ()
Initialize L1 structures.

Variables

- uint8_t [se3_sessions_buf](#) [[SE3_SESSIONS_BUF](#)]
- uint8_t * [se3_sessions_index](#) [[SE3_SESSIONS_MAX](#)]
- [se3_algo_descriptor](#) L1d_algo_table [[SE3_ALGO_MAX](#)]
- [SE3_L1_GLOBALS](#) se3c1
L1 globals.

6.32.1 Detailed Description

L1 structures and functions.

Author

Nicola Ferri

6.32.2 Enumeration Type Documentation

6.32.2.1 anonymous enum

Enumerator

SE3_FLASH_TYPE_RECORD flash node type: record

6.32.2.2 anonymous enum

Enumerator

SE3_RECORD_SIZE_TYPE record.type field size

SE3_RECORD_OFFSET_TYPE record.type field offset

SE3_RECORD_OFFSET_DATA record.data field offset

6.32.2.3 anonymous enum

Enumerator

SE3_SESSIONS_BUF session buffer size

SE3_SESSIONS_MAX maximum number of sessions

6.32.3 Function Documentation

6.32.3.1 void se3c1_login_cleanup ()

Clear login session data.

Cleans all data associated with the login session, making SEcube ready for a new login.

6.32.3.2 bool se3c1_record_get (uint16_t type, uint8_t * data)

Read record.

Get data of a record.

Parameters

<i>type</i>	type of record
<i>data</i>	output buffer

Returns

true on success; false if the record does not exist or has never been written

6.32.3.3 `bool se3c1_record_set (uint16_t type, const uint8_t * data)`

Write record.

Set data of a record

Remarks

if a flash operation fails, the hwerror flag (se3c0.hwerror) is set.

Parameters

<i>type</i>	type of record
<i>data</i>	new data to be written to record

Returns

true on success; false if the record does not exist

6.32.4 Variable Documentation

6.32.4.1 `se3_algo_descriptor L1d_algo_table[SE3_ALGO_MAX]`

algorithm description table

6.32.4.2 `uint8_t se3_sessions_buf[SE3_SESSIONS_BUF]`

session buffer

6.32.4.3 `uint8_t* se3_sessions_index[SE3_SESSIONS_MAX]`

session index

Index

- AES data structures, [10](#)
- AES functions, [11](#)
 - B5_Aes256_Finit, [11](#)
 - B5_Aes256_Init, [11](#)
 - B5_Aes256_SetIV, [11](#)
 - B5_Aes256_Update, [13](#)
- AES Key, IV, Block Sizes, [8](#)
 - B5_AES_128, [8](#)
 - B5_AES_192, [8](#)
 - B5_AES_256, [8](#)
 - B5_AES_BLK_SIZE, [8](#)
 - B5_AES_IV_SIZE, [8](#)
- AES modes, [9](#)
 - B5_AES256_CBC_DEC, [9](#)
 - B5_AES256_CBC_ENC, [9](#)
 - B5_AES256_CFB_DEC, [9](#)
 - B5_AES256_CFB_ENC, [9](#)
 - B5_AES256_CTR, [9](#)
 - B5_AES256_ECB_DEC, [9](#)
 - B5_AES256_ECB_ENC, [9](#)
 - B5_AES256_OFB, [9](#)
- AES return values, [7](#)
- AccessLogin, [19](#)
- AesHmacSha256s_ctx, [31](#)
- AlgorithmAvail, [21](#)
 - SE3_ALGO_AES_HMACSHA256, [21](#)
 - SE3_ALGO_AES_HMAC, [21](#)
 - SE3_ALGO_AES, [21](#)
 - SE3_ALGO_HMACSHA256, [21](#)
 - SE3_ALGO_SHA256, [21](#)
- B5_AES256_CBC_DEC
 - AES modes, [9](#)
- B5_AES256_CBC_ENC
 - AES modes, [9](#)
- B5_AES256_CFB_DEC
 - AES modes, [9](#)
- B5_AES256_CFB_ENC
 - AES modes, [9](#)
- B5_AES256_CTR
 - AES modes, [9](#)
- B5_AES256_ECB_DEC
 - AES modes, [9](#)
- B5_AES256_ECB_ENC
 - AES modes, [9](#)
- B5_AES256_OFB
 - AES modes, [9](#)
- B5_AES_128
 - AES Key, IV, Block Sizes, [8](#)
- B5_AES_192
 - AES Key, IV, Block Sizes, [8](#)
- B5_AES_256
 - AES Key, IV, Block Sizes, [8](#)
- B5_AES_BLK_SIZE
 - AES Key, IV, Block Sizes, [8](#)
- B5_AES_IV_SIZE
 - AES Key, IV, Block Sizes, [8](#)
- B5_Aes256_Finit
 - AES functions, [11](#)
- B5_Aes256_Init
 - AES functions, [11](#)
- B5_Aes256_SetIV
 - AES functions, [11](#)
- B5_Aes256_Update
 - AES functions, [13](#)
- B5_CMAC_AES_128
 - CMAC-AES Key, Blk Sizes, [14](#)
- B5_CMAC_AES_192
 - CMAC-AES Key, Blk Sizes, [14](#)
- B5_CMAC_AES_256
 - CMAC-AES Key, Blk Sizes, [14](#)
- B5_CMAC_AES_BLK_SIZE
 - CMAC-AES Key, Blk Sizes, [14](#)
- B5_CmacAes256_Finit
 - CMAC-AES functions, [17](#)
- B5_CmacAes256_Init
 - CMAC-AES functions, [17](#)
- B5_CmacAes256_Reset
 - CMAC-AES functions, [17](#)
- B5_CmacAes256_Sign
 - CMAC-AES functions, [18](#)
- B5_CmacAes256_Update
 - CMAC-AES functions, [18](#)
- B5_HmacSha256_Finit
 - HMAC-SHA256 functions, [29](#)
- B5_HmacSha256_Init
 - HMAC-SHA256 functions, [29](#)
- B5_HmacSha256_Update
 - HMAC-SHA256 functions, [29](#)
- B5_Sha256_Finit
 - SHA256 functions, [25](#)
- B5_Sha256_Init
 - SHA256 functions, [25](#)
- B5_Sha256_Update
 - SHA256 functions, [25](#)
- B5_tAesCtx, [31](#)
 - InitVector, [32](#)
 - mode, [32](#)
 - Nr, [32](#)

- rk, [32](#)
- B5_tCmacAesCtx, [32](#)
- B5_tHmacSha256Ctx, [32](#)
- B5_tSha256Ctx, [33](#)
- block_is_magic
 - se3_proto.c, [88](#)
- CMAC-AES data structures, [16](#)
- CMAC-AES functions, [17](#)
 - B5_CmacAes256_Finit, [17](#)
 - B5_CmacAes256_Init, [17](#)
 - B5_CmacAes256_Reset, [17](#)
 - B5_CmacAes256_Sign, [18](#)
 - B5_CmacAes256_Update, [18](#)
- CMAC-AES Key, Blk Sizes, [14](#)
 - B5_CMACEES_128, [14](#)
 - B5_CMACEES_192, [14](#)
 - B5_CMACEES_256, [14](#)
 - B5_CMACEES_BLK_SIZE, [14](#)
- CMAC-AES return values, [15](#)
- crc16.h
 - se3_crc16_update, [43](#)
- find_magic_index
 - se3_proto.c, [89](#)
- HMAC-SHA256 data structures, [28](#)
- HMAC-SHA256 functions, [29](#)
 - B5_HmacSha256_Finit, [29](#)
 - B5_HmacSha256_Init, [29](#)
 - B5_HmacSha256_Update, [29](#)
- HMAC-SHA256 return values, [27](#)
- handle_req_recv
 - se3_proto.c, [89](#)
- handle_resp_send
 - se3_proto.c, [89](#)
- InitVector
 - B5_tAesCtx, [32](#)
- KeyOpEdit, [20](#)
 - SE3_KEY_OP_DELETE, [20](#)
 - SE3_KEY_OP_INSERT, [20](#)
 - SE3_KEY_OP_UPSERT, [20](#)
- L0d_cmd1
 - se3_cmd1.c, [58](#)
 - se3_cmd1.h, [59](#)
- L0d_echo
 - se3_cmd0.c, [56](#)
 - se3_cmd0.h, [57](#)
- L0d_factory_init
 - se3_cmd0.c, [56](#)
 - se3_cmd0.h, [57](#)
- L1d_algo_table
 - se3c1.c, [96](#)
 - se3c1.h, [99](#)
- L1d_challenge
 - se3_cmd1_login.c, [65](#)
 - se3_cmd1_login.h, [67](#)
- L1d_config
 - se3_cmd1_config.c, [59](#)
 - se3_cmd1_config.h, [60](#)
- L1d_crypto_init
 - se3_cmd1_crypto.c, [61](#)
 - se3_cmd1_crypto.h, [62](#)
- L1d_crypto_list
 - se3_cmd1_crypto.c, [61](#)
 - se3_cmd1_crypto.h, [62](#)
- L1d_crypto_set_time
 - se3_cmd1_crypto.c, [61](#)
 - se3_cmd1_crypto.h, [62](#)
- L1d_crypto_update
 - se3_cmd1_crypto.c, [61](#)
 - se3_cmd1_crypto.h, [62](#)
- L1d_handlers
 - se3_cmd1.c, [58](#)
- L1d_key_edit
 - se3_cmd1_keys.c, [63](#)
 - se3_cmd1_keys.h, [64](#)
- L1d_key_list
 - se3_cmd1_keys.c, [63](#)
 - se3_cmd1_keys.h, [64](#)
- L1d_login
 - se3_cmd1_login.c, [65](#)
 - se3_cmd1_login.h, [67](#)
- L1d_logout
 - se3_cmd1_login.c, [66](#)
 - se3_cmd1_login.h, [67](#)
- mode
 - B5_tAesCtx, [32](#)
- Nr
 - B5_tAesCtx, [32](#)
- rk
 - B5_tAesCtx, [32](#)
- s3_storage_range_, [33](#)
- SE3_ALGO_AES_HMACSHA256
 - AlgorithmAvail, [21](#)
- SE3_ALGO_AES_HMAC
 - AlgorithmAvail, [21](#)
- SE3_ALGO_AES
 - AlgorithmAvail, [21](#)
- SE3_ALGO_HMACSHA256
 - AlgorithmAvail, [21](#)
- SE3_ALGO_SHA256
 - AlgorithmAvail, [21](#)
- SE3_COMM_STATUS_, [34](#)
- SE3_COMM_STATUS
 - se3c0.h, [94](#)
- SE3_ERR_ACCESS
 - se3c1def.h, [50](#)
- SE3_ERR_AUTH
 - se3c1def.h, [50](#)
- SE3_ERR_EXPIRED
 - se3c1def.h, [50](#)

SE3_ERR_MEMORY
 se3c1def.h, [50](#)

SE3_ERR_PIN
 se3c1def.h, [50](#)

SE3_ERR_RESOURCE
 se3c1def.h, [50](#)

SE3_FLASH_INFO_, [35](#)

SE3_FLASH_TYPE_CONT
 se3_flash.h, [73](#)

SE3_FLASH_TYPE_EMPTY
 se3_flash.h, [73](#)

SE3_FLASH_TYPE_INVALID
 se3_flash.h, [73](#)

SE3_FLASH_TYPE_RECORD
 se3c1.h, [98](#)

SE3_FLASH_TYPE_SERIAL
 se3_flash.h, [73](#)

SE3_KEY_OP_DELETE
 KeyOpEdit, [20](#)

SE3_KEY_OP_INSERT
 KeyOpEdit, [20](#)

SE3_KEY_OP_UPSERT
 KeyOpEdit, [20](#)

SE3_L0_GLOBALS_, [37](#)

SE3_L1_GLOBALS_, [37](#)

SE3_LOGIN_STATUS_, [38](#)

SE3_MEM_BLOCK
 se3_memory.h, [86](#)

SE3_MEM_HEADER
 se3_memory.h, [86](#)

SE3_PROTO_BUSY
 se3_proto.h, [91](#)

SE3_PROTO_FAIL
 se3_proto.h, [91](#)

SE3_PROTO_OK
 se3_proto.h, [91](#)

SE3_RECORD_INFO_, [39](#)

SE3_RECORD_OFFSET_DATA
 se3c1.h, [98](#)

SE3_RECORD_OFFSET_TYPE
 se3c1.h, [98](#)

SE3_RECORD_SIZE_TYPE
 se3c1.h, [98](#)

SE3_SERIAL_, [40](#)

SE3_SESSIONS_BUF
 se3c1.h, [98](#)

SE3_SESSIONS_MAX
 se3c1.h, [98](#)

SHA256 data structures, [24](#)

SHA256 digest and block sizes, [23](#)

SHA256 functions, [25](#)
 B5_Sha256_Finit, [25](#)
 B5_Sha256_Init, [25](#)
 B5_Sha256_Update, [25](#)

SHA256 return values, [22](#)

se3_algo_Aes.h
 se3_algo_Aes_init, [51](#)
 se3_algo_Aes_update, [51](#)

se3_algo_Aes_init
 se3_algo_Aes.h, [51](#)

se3_algo_Aes_update
 se3_algo_Aes.h, [51](#)

se3_algo_AesHmacSha256s.c
 se3_algo_AesHmacSha256s_init, [52](#)
 se3_algo_AesHmacSha256s_update, [52](#)

se3_algo_AesHmacSha256s.h
 se3_algo_AesHmacSha256s_init, [53](#)
 se3_algo_AesHmacSha256s_update, [53](#)

se3_algo_AesHmacSha256s_init
 se3_algo_AesHmacSha256s.c, [52](#)
 se3_algo_AesHmacSha256s.h, [53](#)

se3_algo_AesHmacSha256s_update
 se3_algo_AesHmacSha256s.c, [52](#)
 se3_algo_AesHmacSha256s.h, [53](#)

se3_algo_descriptor_, [33](#)

se3_cmd.c
 se3_cmd_execute, [54](#)

se3_cmd.h
 se3_cmd_execute, [55](#)

se3_cmd0.c
 L0d_echo, [56](#)
 L0d_factory_init, [56](#)

se3_cmd0.h
 L0d_echo, [57](#)
 L0d_factory_init, [57](#)

se3_cmd1.c
 L0d_cmd1, [58](#)
 L1d_handlers, [58](#)

se3_cmd1.h
 L0d_cmd1, [59](#)

se3_cmd1_config.c
 L1d_config, [59](#)

se3_cmd1_config.h
 L1d_config, [60](#)

se3_cmd1_crypto.c
 L1d_crypto_init, [61](#)
 L1d_crypto_list, [61](#)
 L1d_crypto_set_time, [61](#)
 L1d_crypto_update, [61](#)

se3_cmd1_crypto.h
 L1d_crypto_init, [62](#)
 L1d_crypto_list, [62](#)
 L1d_crypto_set_time, [62](#)
 L1d_crypto_update, [62](#)

se3_cmd1_keys.c
 L1d_key_edit, [63](#)
 L1d_key_list, [63](#)

se3_cmd1_keys.h
 L1d_key_edit, [64](#)
 L1d_key_list, [64](#)

se3_cmd1_login.c
 L1d_challenge, [65](#)
 L1d_login, [65](#)
 L1d_logout, [66](#)

se3_cmd1_login.h
 L1d_challenge, [67](#)

- L1d_login, [67](#)
- L1d_logout, [67](#)
- se3_cmd_execute
 - se3_cmd.c, [54](#)
 - se3_cmd.h, [55](#)
- se3_cmd_func
 - se3c0.h, [94](#)
- se3_common.h
 - se3_nblocks, [44](#)
 - se3_req_len_data, [45](#)
 - se3_req_len_data_and_headers, [45](#)
 - se3_resp_len_data, [45](#)
 - se3_resp_len_data_and_headers, [45](#)
- se3_crc16_update
 - crc16.h, [43](#)
- se3_flash.c
 - se3_flash_canfit, [68](#)
 - se3_flash_info_setup, [69](#)
 - se3_flash_init, [69](#)
 - se3_flash_it_delete, [69](#)
 - se3_flash_it_init, [69](#)
 - se3_flash_it_new, [70](#)
 - se3_flash_it_next, [70](#)
 - se3_flash_it_write, [70](#)
 - se3_flash_pos_delete, [71](#)
 - se3_flash_unused, [71](#)
- se3_flash.h
 - SE3_FLASH_TYPE_CONT, [73](#)
 - SE3_FLASH_TYPE_EMPTY, [73](#)
 - SE3_FLASH_TYPE_INVALID, [73](#)
 - SE3_FLASH_TYPE_SERIAL, [73](#)
 - se3_flash_canfit, [73](#)
 - se3_flash_info_setup, [73](#)
 - se3_flash_init, [73](#)
 - se3_flash_it_delete, [74](#)
 - se3_flash_it_init, [74](#)
 - se3_flash_it_new, [74](#)
 - se3_flash_it_next, [75](#)
 - se3_flash_it_write, [75](#)
 - se3_flash_pos_delete, [75](#)
 - se3_flash_unused, [76](#)
- se3_flash_canfit
 - se3_flash.c, [68](#)
 - se3_flash.h, [73](#)
- se3_flash_info_setup
 - se3_flash.c, [69](#)
 - se3_flash.h, [73](#)
- se3_flash_init
 - se3_flash.c, [69](#)
 - se3_flash.h, [73](#)
- se3_flash_it_, [36](#)
- se3_flash_it_delete
 - se3_flash.c, [69](#)
 - se3_flash.h, [74](#)
- se3_flash_it_init
 - se3_flash.c, [69](#)
 - se3_flash.h, [74](#)
- se3_flash_it_new
 - se3_flash.c, [70](#)
 - se3_flash.h, [74](#)
- se3_flash_it_next
 - se3_flash.c, [70](#)
 - se3_flash.h, [75](#)
- se3_flash_it_write
 - se3_flash.c, [70](#)
 - se3_flash.h, [75](#)
- se3_flash_key
 - se3_keys.h, [80](#)
- se3_flash_key_, [36](#)
- se3_flash_pos_delete
 - se3_flash.c, [71](#)
 - se3_flash.h, [75](#)
- se3_flash_unused
 - se3_flash.c, [71](#)
 - se3_flash.h, [76](#)
- se3_hello
 - se3c0.c, [92](#)
- se3_key_equal
 - se3_keys.c, [77](#)
 - se3_keys.h, [81](#)
- se3_key_find
 - se3_keys.c, [77](#)
 - se3_keys.h, [81](#)
- se3_key_new
 - se3_keys.c, [77](#)
 - se3_keys.h, [81](#)
- se3_key_read
 - se3_keys.c, [78](#)
 - se3_keys.h, [82](#)
- se3_key_read_data
 - se3_keys.c, [78](#)
 - se3_keys.h, [82](#)
- se3_key_remove
 - se3_keys.c, [78](#)
 - se3_keys.h, [82](#)
- se3_key_write
 - se3_keys.c, [79](#)
 - se3_keys.h, [83](#)
- se3_keys.c
 - se3_key_equal, [77](#)
 - se3_key_find, [77](#)
 - se3_key_new, [77](#)
 - se3_key_read, [78](#)
 - se3_key_read_data, [78](#)
 - se3_key_remove, [78](#)
 - se3_key_write, [79](#)
- se3_keys.h
 - se3_flash_key, [80](#)
 - se3_key_equal, [81](#)
 - se3_key_find, [81](#)
 - se3_key_new, [81](#)
 - se3_key_read, [82](#)
 - se3_key_read_data, [82](#)
 - se3_key_remove, [82](#)
 - se3_key_write, [83](#)
- se3_mem_, [38](#)
- se3_flash.c, [70](#)
- se3_flash.h, [74](#)

- se3_mem_alloc
 - se3_memory.c, [84](#)
 - se3_memory.h, [86](#)
- se3_mem_free
 - se3_memory.c, [84](#)
 - se3_memory.h, [86](#)
- se3_mem_init
 - se3_memory.c, [84](#)
 - se3_memory.h, [87](#)
- se3_mem_ptr
 - se3_memory.c, [85](#)
 - se3_memory.h, [87](#)
- se3_mem_reset
 - se3_memory.c, [85](#)
 - se3_memory.h, [87](#)
- se3_memory.c
 - se3_mem_alloc, [84](#)
 - se3_mem_free, [84](#)
 - se3_mem_init, [84](#)
 - se3_mem_ptr, [85](#)
 - se3_mem_reset, [85](#)
- se3_memory.h
 - SE3_MEM_BLOCK, [86](#)
 - SE3_MEM_HEADER, [86](#)
 - se3_mem_alloc, [86](#)
 - se3_mem_free, [86](#)
 - se3_mem_init, [87](#)
 - se3_mem_ptr, [87](#)
 - se3_mem_reset, [87](#)
- se3_nblocks
 - se3_common.h, [44](#)
- se3_payload_cryptoctx_, [39](#)
- se3_proto.c
 - block_is_magic, [88](#)
 - find_magic_index, [89](#)
 - handle_req_rcv, [89](#)
 - handle_resp_send, [89](#)
 - se3_proto_rcv, [90](#)
 - se3_proto_request_reset, [90](#)
 - se3_proto_send, [90](#)
 - se3_storage_range_add, [90](#)
- se3_proto.h
 - SE3_PROTO_BUSY, [91](#)
 - SE3_PROTO_FAIL, [91](#)
 - SE3_PROTO_OK, [91](#)
 - se3_proto_rcv, [91](#)
 - se3_proto_send, [91](#)
- se3_proto_rcv
 - se3_proto.c, [90](#)
 - se3_proto.h, [91](#)
- se3_proto_request_reset
 - se3_proto.c, [90](#)
- se3_proto_send
 - se3_proto.c, [90](#)
 - se3_proto.h, [91](#)
- se3_req_len_data
 - se3_common.h, [45](#)
- se3_req_len_data_and_headers
 - se3_common.h, [45](#)
- se3_common.h, [45](#)
- se3_resp_len_data
 - se3_common.h, [45](#)
- se3_resp_len_data_and_headers
 - se3_common.h, [45](#)
- se3_sessions_buf
 - se3c1.c, [96](#)
 - se3c1.h, [99](#)
- se3_sessions_index
 - se3c1.c, [96](#)
 - se3c1.h, [99](#)
- se3_storage_range_add
 - se3_proto.c, [90](#)
- se3c0.c
 - se3_hello, [92](#)
- se3c0.h
 - SE3_COMM_STATUS, [94](#)
 - se3_cmd_func, [94](#)
- se3c0_req_header_, [40](#)
- se3c0_resp_header_, [41](#)
- se3c1.c
 - L1d_algo_table, [96](#)
 - se3_sessions_buf, [96](#)
 - se3_sessions_index, [96](#)
 - se3c1_login_cleanup, [95](#)
 - se3c1_record_get, [95](#)
 - se3c1_record_set, [95](#)
- se3c1.h
 - L1d_algo_table, [99](#)
 - SE3_FLASH_TYPE_RECORD, [98](#)
 - SE3_RECORD_OFFSET_DATA, [98](#)
 - SE3_RECORD_OFFSET_TYPE, [98](#)
 - SE3_RECORD_SIZE_TYPE, [98](#)
 - SE3_SESSIONS_BUF, [98](#)
 - SE3_SESSIONS_MAX, [98](#)
 - se3_sessions_buf, [99](#)
 - se3_sessions_index, [99](#)
 - se3c1_login_cleanup, [98](#)
 - se3c1_record_get, [98](#)
 - se3c1_record_set, [98](#)
- se3c1_login_cleanup
 - se3c1.c, [95](#)
 - se3c1.h, [98](#)
- se3c1_record_get
 - se3c1.c, [95](#)
 - se3c1.h, [98](#)
- se3c1_record_set
 - se3c1.c, [95](#)
 - se3c1.h, [98](#)
- se3c1def.h
 - SE3_ERR_ACCESS, [50](#)
 - SE3_ERR_AUTH, [50](#)
 - SE3_ERR_EXPIRED, [50](#)
 - SE3_ERR_MEMORY, [50](#)
 - SE3_ERR_PIN, [50](#)
 - SE3_ERR_RESOURCE, [50](#)
- src/Common/crc16.h, [43](#)
- src/Common/se3_common.h, [44](#)

src/Common/se3c1def.h, [46](#)
src/Device/se3_algo_Aes.h, [51](#)
src/Device/se3_algo_AesHmacSha256s.c, [52](#)
src/Device/se3_algo_AesHmacSha256s.h, [53](#)
src/Device/se3_cmd.c, [54](#)
src/Device/se3_cmd.h, [55](#)
src/Device/se3_cmd0.c, [55](#)
src/Device/se3_cmd0.h, [56](#)
src/Device/se3_cmd1.c, [57](#)
src/Device/se3_cmd1.h, [58](#)
src/Device/se3_cmd1_config.c, [59](#)
src/Device/se3_cmd1_config.h, [59](#)
src/Device/se3_cmd1_crypto.c, [60](#)
src/Device/se3_cmd1_crypto.h, [61](#)
src/Device/se3_cmd1_keys.c, [63](#)
src/Device/se3_cmd1_keys.h, [64](#)
src/Device/se3_cmd1_login.c, [65](#)
src/Device/se3_cmd1_login.h, [66](#)
src/Device/se3_flash.c, [67](#)
src/Device/se3_flash.h, [71](#)
src/Device/se3_keys.c, [76](#)
src/Device/se3_keys.h, [79](#)
src/Device/se3_memory.c, [83](#)
src/Device/se3_memory.h, [85](#)
src/Device/se3_proto.c, [87](#)
src/Device/se3_proto.h, [90](#)
src/Device/se3c0.c, [92](#)
src/Device/se3c0.h, [92](#)
src/Device/se3c1.c, [94](#)
src/Device/se3c1.h, [96](#)