TII TLS1.3

Generated by Doxygen 1.9.6

| 1 TIIGER TLS C++   | 1    |
|--|------|
| 1.1 Building   | . 1  |
| 1.1.1 Miracl   | . 1  |
| 1.1.2 Miracl + LibSodium   | . 1  |
| 1.2 Try it out   | . 2  |
| 1.2.1 Client side Authentication                                     | . 4  |
| 1.2.2 Testing Pre-shared keys  | . 5  |
| 2 Configure the Arduino Nano RP2040                                  | 7    |
| 2.1 Building the client application on the Arduino Nano RP2040 board | . 7  |
| 3 Data Structure Index   | 9    |
| 3.1 Data Structures  | . 9  |
| 4 File Index   | 11   |
| 4.1 File List  | . 11 |
| 5 Data Structure Documentation                                       | 13   |
| 5.1 crypto Struct Reference  | . 13 |
| 5.1.1 Detailed Description   | . 13 |
| 5.1.2 Field Documentation  | . 13 |
| 5.1.2.1 active   | . 13 |
| 5.1.2.2 k  | . 14 |
| 5.1.2.3 iv   | . 14 |
| 5.1.2.4 K  | . 14 |
| 5.1.2.5 IV   | . 14 |
| 5.1.2.6 record   | . 14 |
| 5.1.2.7 suite  | . 14 |
| 5.1.2.8 taglen   | . 14 |
| 5.2 ECCX08Class Class Reference                                      | . 15 |
| 5.3 ee_status Struct Reference                                       | . 15 |
| 5.3.1 Detailed Description   | . 16 |
| 5.3.2 Field Documentation  | . 16 |
| 5.3.2.1 early_data   | . 16 |
| 5.3.2.2 alpn   | . 16 |
| 5.3.2.3 server_name  | . 16 |
| 5.3.2.4 max_frag_length  | . 16 |
| 5.4 octad Struct Reference   | . 16 |
| 5.4.1 Detailed Description   | . 17 |
| 5.4.2 Field Documentation  | . 17 |
| 5.4.2.1 len  | . 17 |
| 5.4.2.2 max  | . 17 |
| 5.4.2.3 val  | . 17 |
| 5.5 pktype Struct Reference  | . 17 |

| 5.5.1 Detailed Description       | 18 |
|----------------------------------|----|
| 5.5.2 Field Documentation        | 18 |
| 5.5.2.1 type                     | 18 |
| 5.5.2.2 hash                     | 18 |
| 5.5.2.3 curve                    | 18 |
| 5.6 ret Struct Reference         | 18 |
| 5.6.1 Detailed Description       | 19 |
| 5.6.2 Field Documentation        | 19 |
| 5.6.2.1 val                      | 19 |
| 5.6.2.2 err                      | 19 |
| 5.7 Socket Class Reference       | 19 |
| 5.7.1 Detailed Description       | 20 |
| 5.8 ticket Struct Reference      | 20 |
| 5.8.1 Detailed Description       | 20 |
| 5.8.2 Field Documentation        | 20 |
| 5.8.2.1 valid                    | 21 |
| 5.8.2.2 tick                     | 21 |
| 5.8.2.3 nonce                    | 21 |
| 5.8.2.4 psk                      | 21 |
| 5.8.2.5 TICK                     | 21 |
| 5.8.2.6 NONCE                    | 21 |
| 5.8.2.7 PSK                      | 21 |
| 5.8.2.8 age_obfuscator           | 21 |
| 5.8.2.9 max_early_data           | 22 |
| 5.8.2.10 birth                   | 22 |
| 5.8.2.11 lifetime                | 22 |
| 5.8.2.12 cipher_suite            | 22 |
| 5.8.2.13 favourite_group         | 22 |
| 5.8.2.14 origin                  | 22 |
| 5.9 TLS_session Struct Reference | 22 |
| 5.9.1 Detailed Description       | 23 |
| 5.9.2 Field Documentation        | 23 |
| 5.9.2.1 status                   | 23 |
| 5.9.2.2 max_record               | 23 |
| 5.9.2.3 sockptr                  | 24 |
| 5.9.2.4 id                       | 24 |
| 5.9.2.5 hostname                 | 24 |
| 5.9.2.6 cipher_suite             | 24 |
| 5.9.2.7 favourite_group          | 24 |
| 5.9.2.8 server_cert_type         | 24 |
| 5.9.2.9 client_cert_type         | 24 |
| 5.9.2.10 K_send                  | 24 |

| 5.9.2.11 K_recv                      | <br>. 25 |
|--------------------------------------|----------|
| 5.9.2.12 HS                          | <br>. 25 |
| 5.9.2.13 hs                          | <br>. 25 |
| 5.9.2.14 RMS                         | <br>. 25 |
| 5.9.2.15 rms                         | <br>. 25 |
| 5.9.2.16 STS                         | <br>. 25 |
| 5.9.2.17 sts                         | <br>. 25 |
| 5.9.2.18 CTS                         | <br>. 25 |
| 5.9.2.19 cts                         | <br>. 26 |
| 5.9.2.20 CTX                         | <br>. 26 |
| 5.9.2.21 ctx                         | <br>. 26 |
| 5.9.2.22 IBUFF                       | <br>. 26 |
| 5.9.2.23 OBUFF                       | <br>. 26 |
| 5.9.2.24 ibuff                       | <br>. 26 |
| 5.9.2.25 obuff                       | <br>. 26 |
| 5.9.2.26 ptr                         | <br>. 26 |
| 5.9.2.27 tlshash                     | <br>. 27 |
| 5.9.2.28 T                           | <br>. 27 |
| 5.10 unihash Struct Reference        | <br>. 27 |
| 5.10.1 Detailed Description          | <br>. 27 |
| 5.10.2 Field Documentation           | <br>. 27 |
| 5.10.2.1 state                       | <br>. 27 |
| 5.10.2.2 htype                       | <br>. 27 |
| 6 File Documentation                 | 29       |
| 6.1 tls1_3.h File Reference          | _        |
| 6.1.1 Detailed Description           |          |
| 6.1.2 Macro Definition Documentation |          |
| 6.1.2.1 IO_NONE                      |          |
| 6.1.2.2 IO_APPLICATION               |          |
| 6.1.2.3 IO_PROTOCOL                  |          |
| 6.1.2.4 IO DEBUG                     |          |
| 6.1.2.5 IO_WIRE                      |          |
| 6.1.2.6 TINY_ECC                     |          |
| 6.1.2.7 TYPICAL                      |          |
|                                      |          |
| 6.1.2.8 POST_QUANTUM                 |          |
| 6.1.2.9 HYBRID                       |          |
| 6.1.2.10 NOCERT                      |          |
| 6.1.2.11 RSA_SS                      |          |
| 6.1.2.12 ECC_SS                      |          |
| 6.1.2.13 DLT_SS                      |          |
| 6.1.2.14 HYB_SS                      | <br>. 35 |

| 6.1.2.15 HW_1                        |
|--------------------------------------|
| 6.1.2.16 HW_2                        |
| 6.1.2.17 VERBOSITY                   |
| 6.1.2.18 THIS_YEAR                   |
| 6.1.2.19 POST_HS_AUTH                |
| 6.1.2.20 CLIENT_CERT                 |
| 6.1.2.21 CRYPTO_SETTING              |
| 6.1.2.22 TLS_APPLICATION_PROTOCOL    |
| 6.1.2.23 ALLOW_SELF_SIGNED           |
| 6.1.2.24 TRY_EARLY_DATA              |
| 6.1.2.25 TLS_SHA256_T                |
| 6.1.2.26 TLS_SHA384_T                |
| 6.1.2.27 TLS_SHA512_T                |
| 6.1.2.28 TLS_MAX_HASH_STATE          |
| 6.1.2.29 TLS_MAX_HASH                |
| 6.1.2.30 TLS_MAX_KEY                 |
| 6.1.2.31 TLS_X509_MAX_FIELD          |
| 6.1.2.32 TLS_MAX_EXT_LABEL           |
| 6.1.2.33 TLS_MAX_FRAG                |
| 6.1.2.34 TLS_MAX_IBUFF_SIZE          |
| 6.1.2.35 TLS_MAX_PLAIN_FRAG          |
| 6.1.2.36 TLS_MAX_CIPHER_FRAG         |
| 6.1.2.37 TLS_MAX_CERT_SIZE           |
| 6.1.2.38 TLS_MAX_CERT_B64            |
| 6.1.2.39 TLS_MAX_HELLO               |
| 6.1.2.40 TLS_MAX_SIG_PUB_KEY_SIZE    |
| 6.1.2.41 TLS_MAX_SIG_SECRET_KEY_SIZE |
| 6.1.2.42 TLS_MAX_SIGNATURE_SIZE      |
| 6.1.2.43 TLS_MAX_KEX_PUB_KEY_SIZE    |
| 6.1.2.44 TLS_MAX_KEX_CIPHERTEXT_SIZE |
| 6.1.2.45 TLS_MAX_KEX_SECRET_KEY_SIZE |
| 6.1.2.46 TLS_MAX_SERVER_CHAIN_LEN    |
| 6.1.2.47 TLS_MAX_SERVER_CHAIN_SIZE   |
| 6.1.2.48 TLS_MAX_CLIENT_CHAIN_LEN    |
| 6.1.2.49 TLS_MAX_CLIENT_CHAIN_SIZE   |
| 6.1.2.50 TLS_MAX_SHARED_SECRET_SIZE  |
| 6.1.2.51 TLS_MAX_TICKET_SIZE         |
| 6.1.2.52 TLS_MAX_EXTENSIONS          |
| 6.1.2.53 TLS_MAX_ECC_FIELD           |
| 6.1.2.54 TLS_MAX_IV_SIZE             |
| 6.1.2.55 TLS_MAX_TAG_SIZE            |
| 6.1.2.56 TLS MAX COOKIE              |

| 6.1.2.57 TLS_MAX_OUTPUT_RECORD_SIZE   |
|---------------------------------------|
| 6.1.2.58 TLS_MAX_OBUFF_SIZE           |
| 6.1.2.59 TLS_MAX_SERVER_NAME          |
| 6.1.2.60 TLS_MAX_SUPPORTED_GROUPS     |
| 6.1.2.61 TLS_MAX_SUPPORTED_SIGS       |
| 6.1.2.62 TLS_MAX_PSK_MODES            |
| 6.1.2.63 TLS_MAX_CIPHER_SUITES        |
| 6.1.2.64 TLS_AES_128_GCM_SHA256       |
| 6.1.2.65 TLS_AES_256_GCM_SHA384       |
| 6.1.2.66 TLS_CHACHA20_POLY1305_SHA256 |
| 6.1.2.67 TLS_AES_128_CCM_SHA256       |
| 6.1.2.68 TLS_AES_128_CCM_8_SHA256     |
| 6.1.2.69 X25519                       |
| 6.1.2.70 SECP256R1                    |
| 6.1.2.71 SECP384R1                    |
| 6.1.2.72 SECP521R1                    |
| 6.1.2.73 X448                         |
| 6.1.2.74 KYBER768                     |
| 6.1.2.75 HYBRID_KX                    |
| 6.1.2.76 ECDSA_SECP256R1_SHA256       |
| 6.1.2.77 ECDSA_SECP256R1_SHA384       |
| 6.1.2.78 ECDSA_SECP384R1_SHA384       |
| 6.1.2.79 RSA_PSS_RSAE_SHA256          |
| 6.1.2.80 RSA_PSS_RSAE_SHA384          |
| 6.1.2.81 RSA_PSS_RSAE_SHA512          |
| 6.1.2.82 RSA_PKCS1_SHA256             |
| 6.1.2.83 RSA_PKCS1_SHA384             |
| 6.1.2.84 RSA_PKCS1_SHA512             |
| 6.1.2.85 ED25519                      |
| 6.1.2.86 DILITHIUM2                   |
| 6.1.2.87 DILITHIUM3                   |
| 6.1.2.88 DILITHIUM2_P256              |
| 6.1.2.89 PSKOK                        |
| 6.1.2.90 PSKWECDHE                    |
| 6.1.2.91 TLS_FULL_HANDSHAKE           |
| 6.1.2.92 TLS_EXTERNAL_PSK             |
| 6.1.2.93 TLS1_0                       |
| 6.1.2.94 TLS1_2                       |
| 6.1.2.95 TLS1_3                       |
| 6.1.2.96 TLS13_UPDATE_NOT_REQUESTED   |
| 6.1.2.97 TLS13_UPDATE_REQUESTED       |
| 6.1.2.98 SERVER_NAME                  |

| 6.1.2.99 SUPPORTED_GROUPS         |
|-----------------------------------|
| 6.1.2.100 SIG_ALGS                |
| 6.1.2.101 POST_HANDSHAKE_AUTH     |
| 6.1.2.102 SIG_ALGS_CERT           |
| 6.1.2.103 KEY_SHARE               |
| 6.1.2.104 PSK_MODE                |
| 6.1.2.105 PRESHARED_KEY           |
| 6.1.2.106 TLS_VER                 |
| 6.1.2.107 COOKIE                  |
| 6.1.2.108 EARLY_DATA              |
| 6.1.2.109 MAX_FRAG_LENGTH         |
| 6.1.2.110 PADDING                 |
| 6.1.2.111 APP_PROTOCOL            |
| 6.1.2.112 RECORD_SIZE_LIMIT       |
| 6.1.2.113 CLIENT_CERT_TYPE        |
| 6.1.2.114 SERVER_CERT_TYPE        |
| 6.1.2.115 HSHAKE                  |
| 6.1.2.116 APPLICATION             |
| 6.1.2.117 ALERT                   |
| 6.1.2.118 CHANGE_CIPHER           |
| 6.1.2.119 TIMED_OUT               |
| 6.1.2.120 CLIENT_HELLO            |
| 6.1.2.121 SERVER_HELLO            |
| 6.1.2.122 CERTIFICATE             |
| 6.1.2.123 CERT_REQUEST            |
| 6.1.2.124 CERT_VERIFY             |
| 6.1.2.125 FINISHED                |
| 6.1.2.126 ENCRYPTED_EXTENSIONS    |
| 6.1.2.127 TICKET                  |
| 6.1.2.128 KEY_UPDATE              |
| 6.1.2.129 MESSAGE_HASH            |
| 6.1.2.130 END_OF_EARLY_DATA       |
| 6.1.2.131 HANDSHAKE_RETRY         |
| 6.1.2.132 NOT_TLS1_3              |
| 6.1.2.133 BAD_CERT_CHAIN          |
| 6.1.2.134 ID_MISMATCH             |
| 6.1.2.135 UNRECOGNIZED_EXT        |
| 6.1.2.136 BAD_HELLO               |
| 6.1.2.137 WRONG_MESSAGE           |
| 6.1.2.138 MISSING_REQUEST_CONTEXT |
| 6.1.2.139 AUTHENTICATION_FAILURE  |
| 6.1.2.140 BAD_RECORD              |

| 6.1.2.141 BAD_TICKET              |
|-----------------------------------|
| 6.1.2.142 NOT_EXPECTED            |
| 6.1.2.143 CA_NOT_FOUND            |
| 6.1.2.144 CERT_OUTOFDATE          |
| 6.1.2.145 MEM_OVERFLOW            |
| 6.1.2.146 FORBIDDEN_EXTENSION     |
| 6.1.2.147 MAX_EXCEEDED            |
| 6.1.2.148 EMPTY_CERT_CHAIN        |
| 6.1.2.149 SELF_SIGNED_CERT        |
| 6.1.2.150 ERROR_ALERT_RECEIVED    |
| 6.1.2.151 BAD_MESSAGE             |
| 6.1.2.152 CERT_VERIFY_FAIL        |
| 6.1.2.153 BAD_HANDSHAKE           |
| 6.1.2.154 BAD_REQUEST_UPDATE      |
| 6.1.2.155 CLOSURE_ALERT_RECEIVED  |
| 6.1.2.156 MISSING_EXTENSIONS      |
| 6.1.2.157 ILLEGAL_PARAMETER       |
| 6.1.2.158 UNEXPECTED_MESSAGE      |
| 6.1.2.159 DECRYPT_ERROR           |
| 6.1.2.160 BAD_CERTIFICATE         |
| 6.1.2.161 UNSUPPORTED_EXTENSION   |
| 6.1.2.162 UNKNOWN_CA              |
| 6.1.2.163 CERTIFICATE_EXPIRED     |
| 6.1.2.164 PROTOCOL_VERSION        |
| 6.1.2.165 DECODE_ERROR            |
| 6.1.2.166 RECORD_OVERFLOW         |
| 6.1.2.167 BAD_RECORD_MAC          |
| 6.1.2.168 HANDSHAKE_FAILURE       |
| 6.1.2.169 CLOSE_NOTIFY            |
| 6.1.2.170 MISSING_EXTENSION       |
| 6.1.2.171 LOG_OUTPUT_TRUNCATION   |
| 6.1.2.172 TLS13_DISCONNECTED      |
| 6.1.2.173 TLS13_CONNECTED         |
| 6.1.2.174 TLS13_HANDSHAKING       |
| 6.1.2.175 TLS_FAILURE             |
| 6.1.2.176 TLS_SUCCESS             |
| 6.1.2.177 TLS_RESUMPTION_REQUIRED |
| 6.1.2.178 TLS_EARLY_DATA_ACCEPTED |
| 6.1.2.179 PSK_NOT                 |
| 6.1.2.180 PSK_KEY                 |
| 6.1.2.181 PSK_IBE                 |
| 6.1.2.182 X509_CERT               |

| 6.1.2.183 RAW_PUBLIC_KEY                  | 57 |
|---|----|
| 6.1.3 Typedef Documentation               | 57 |
| 6.1.3.1 byte                              | 57 |
| 6.1.3.2 sign8                             | 57 |
| 6.1.3.3 sign16                            | 57 |
| 6.1.3.4 sign32                            | 57 |
| 6.1.3.5 sign64                            | 57 |
| 6.1.3.6 unsign32                          | 58 |
| 6.1.3.7 unsign64                          | 58 |
| 6.2 tls1_3.h                              | 58 |
| 6.3 tls_bfibe.h File Reference            | 63 |
| 6.3.1 Detailed Description                | 63 |
| 6.3.2 Macro Definition Documentation      | 63 |
| 6.3.2.1 PGS_BLS12381                      | 63 |
| 6.3.2.2 PFS_BLS12381                      | 63 |
| 6.3.3 Function Documentation              | 63 |
| 6.3.3.1 BFIBE_CCA_ENCRYPT()               | 63 |
| 6.3.3.2 BFIBE_CCA_DECRYPT()               | 64 |
| 6.4 tls_bfibe.h                           | 64 |
| 6.5 tls_cert_chain.h File Reference       | 65 |
| 6.5.1 Detailed Description                | 65 |
| 6.5.2 Function Documentation              | 65 |
| 6.5.2.1 checkServerCertChain()            | 65 |
| 6.5.2.2 getClientPrivateKeyandCertChain() | 66 |
| 6.6 tls_cert_chain.h                      | 66 |
| 6.7 tls_certs.h File Reference            | 67 |
| 6.7.1 Detailed Description                | 67 |
| 6.7.2 Function Documentation              | 67 |
| 6.7.2.1 getSigRequirements()              | 67 |
| 6.7.3 Variable Documentation              | 68 |
| 6.7.3.1 myprivate                         | 68 |
| 6.7.3.2 mycert                            | 68 |
| 6.7.3.3 cacerts                           | 68 |
| 6.8 tls_certs.h                           | 68 |
| 6.9 tls_client_recv.h File Reference      | 68 |
| 6.9.1 Detailed Description                | 69 |
| 6.9.2 Function Documentation              | 69 |
| 6.9.2.1 parseoctad()                      | 70 |
| 6.9.2.2 parsebytes()                      | 70 |
| 6.9.2.3 parseInt()                        | 70 |
| 6.9.2.4 parseoctadptr()                   | 71 |
| 6.9.2.5 getServerRecord()                 | 71 |

| 6.9.2.6 parseIntorPull()                  | . 72 |
|---|------|
| 6.9.2.7 parseoctadorPull()                | . 72 |
| 6.9.2.8 parsebytesorPull()                | . 73 |
| 6.9.2.9 parseoctadorPullptrX()            | . 73 |
| 6.9.2.10 badResponse()                    | . 73 |
| 6.9.2.11 seeWhatsNext()                   | . 74 |
| 6.9.2.12 getServerEncryptedExtensions()   | . 74 |
| 6.9.2.13 getServerCertVerify()            | . 75 |
| 6.9.2.14 getServerFinished()              | . 75 |
| 6.9.2.15 getServerHello()                 | . 75 |
| 6.9.2.16 getCheckServerCertificateChain() | . 76 |
| 6.9.2.17 getCertificateRequest()          | . 76 |
| 6.10 tls_client_recv.h                    | . 77 |
| 6.11 tls_client_send.h File Reference     | . 77 |
| 6.11.1 Detailed Description               | . 79 |
| 6.11.2 Function Documentation             | . 79 |
| 6.11.2.1 sendCCCS()                       | . 79 |
| 6.11.2.2 addPreSharedKeyExt()             | . 79 |
| 6.11.2.3 addServerNameExt()               | . 80 |
| 6.11.2.4 addSupportedGroupsExt()          | . 80 |
| 6.11.2.5 addServerRawPublicKey()          | . 80 |
| 6.11.2.6 addClientRawPublicKey()          | . 81 |
| 6.11.2.7 addSigAlgsExt()                  | . 81 |
| 6.11.2.8 addSigAlgsCertExt()              | . 81 |
| 6.11.2.9 addKeyShareExt()                 | . 82 |
| 6.11.2.10 addALPNExt()                    | . 82 |
| 6.11.2.11 addMFLExt()                     | . 82 |
| 6.11.2.12 addRSLExt()                     | . 83 |
| 6.11.2.13 addPSKModesExt()                | . 83 |
| 6.11.2.14 addVersionExt()                 | . 83 |
| 6.11.2.15 addPadding()                    | . 84 |
| 6.11.2.16 addCookieExt()                  | . 84 |
| 6.11.2.17 addEarlyDataExt()               | . 84 |
| 6.11.2.18 addPostHSAuth()                 | . 84 |
| 6.11.2.19 clientRandom()                  | . 85 |
| 6.11.2.20 cipherSuites()                  | . 85 |
| 6.11.2.21 sendClientMessage()             | . 86 |
| 6.11.2.22 sendBinder()                    | . 86 |
| 6.11.2.23 sendClientHello()               | . 86 |
| 6.11.2.24 sendAlert()                     | . 87 |
| 6.11.2.25 sendKeyUpdate()                 | . 87 |
| 6.11.2.26 sendClientFinish()              | . 87 |

| 6.15.2.8 logServerResponse()       | )5 |
|------------------------------------|----|
| 6.15.2.9 logAlert()                | )5 |
| 6.15.2.10 logCipherSuite()         | )5 |
| 6.15.2.11 logKeyExchange()         | )5 |
| 6.15.2.12 logSigAlg()              | )6 |
| 6.16 tls_logger.h                  | )6 |
| 6.17 tls_octads.h File Reference   | )6 |
| 6.17.1 Detailed Description        | )7 |
| 6.17.2 Function Documentation      | )8 |
| 6.17.2.1 millis()                  | )8 |
| 6.17.2.2 OCT_append_int()          | )8 |
| 6.17.2.3 OCT_append_octad()        | )8 |
| 6.17.2.4 OCT_compare()             | )9 |
| 6.17.2.5 OCT_shift_left()          | )9 |
| 6.17.2.6 OCT_kill()                | )9 |
| 6.17.2.7 OCT_from_hex()            | 10 |
| 6.17.2.8 OCT_append_string()       | 10 |
| 6.17.2.9 OCT_append_byte()         | 10 |
| 6.17.2.10 OCT_append_bytes()       | 11 |
| 6.17.2.11 OCT_from_base64()        | 11 |
| 6.17.2.12 OCT_reverse()            | 11 |
| 6.17.2.13 OCT_truncate()           | 11 |
| 6.17.2.14 OCT_copy()               | 12 |
| 6.17.2.15 OCT_output_hex()         | 12 |
| 6.17.2.16 OCT_output_string()      | 12 |
| 6.17.2.17 OCT_output_base64()      | 13 |
| 6.18 tls_octads.h                  | 13 |
| 6.19 tls_pqibe.h File Reference    | 14 |
| 6.19.1 Detailed Description        | 14 |
| 6.19.2 Function Documentation      | 14 |
| 6.19.2.1 PQIBE_CCA_ENCRYPT()       | 14 |
| 6.19.2.2 PQIBE_CCA_DECRYPT()       | 15 |
| 6.20 tls_pqibe.h                   | 15 |
| 6.21 tls_protocol.h File Reference | 15 |
| 6.21.1 Detailed Description        | 16 |
| 6.21.2 Function Documentation      | 16 |
| 6.21.2.1 TLS13_start()             | 16 |
| 6.21.2.2 TLS13_end()               | 17 |
| 6.21.2.3 TLS13_stop()              | 17 |
| 6.21.2.4 TLS13_connect()           | 17 |
| 6.21.2.5 TLS13_send()              | 18 |
| 6.21.2.6 TLS13_recv()              | 18 |

| 6.21.2.7 TLS13_clean()               |
|--------------------------------------|
| 6.22 tls_protocol.h                  |
| 6.23 tls_sal.h File Reference        |
| 6.23.1 Detailed Description          |
| 6.23.2 Function Documentation        |
| 6.23.2.1 SAL_name()                  |
| 6.23.2.2 SAL_ciphers()               |
| 6.23.2.3 SAL_groups()                |
| 6.23.2.4 SAL_sigs()                  |
| 6.23.2.5 SAL_sigCerts()              |
| 6.23.2.6 SAL_initLib()               |
| 6.23.2.7 SAL_endLib()                |
| 6.23.2.8 SAL_hashType()              |
| 6.23.2.9 SAL_hashLen()               |
| 6.23.2.10 SAL_aeadKeylen()           |
| 6.23.2.11 SAL_aeadTaglen()           |
| 6.23.2.12 SAL_randomByte()           |
| 6.23.2.13 SAL_randomOctad()          |
| 6.23.2.14 SAL_hkdfExtract()          |
| 6.23.2.15 SAL_hkdfExpand()           |
| 6.23.2.16 SAL_hmac()                 |
| 6.23.2.17 SAL_hashNull()             |
| 6.23.2.18 SAL_hashInit()             |
| 6.23.2.19 SAL_hashProcessArray()     |
| 6.23.2.20 SAL_hashOutput()           |
| 6.23.2.21 SAL_aeadEncrypt()          |
| 6.23.2.22 SAL_aeadDecrypt()          |
| 6.23.2.23 SAL_generateKeyPair()      |
| 6.23.2.24 SAL_generateSharedSecret() |
| 6.23.2.25 SAL_tlsSignatureVerify()   |
| 6.23.2.26 SAL_tlsSignature()         |
| 6.24 tls_sal.h                       |
| 6.25 tls_sockets.h File Reference    |
| 6.25.1 Detailed Description          |
| 6.25.2 Function Documentation        |
| 6.25.2.1 setclientsock()             |
| 6.25.2.2 getIPaddress()              |
| 6.25.2.3 sendOctad()                 |
| 6.25.2.4 sendLen()                   |
| 6.25.2.5 getBytes()                  |
| 6.25.2.6 getInt16()                  |
| 6.25.2.7 getInt24()                  |

| 6.25.2.8 getByte()                    |
|---------------------------------------|
| 6.25.2.9 getOctad()                   |
| 6.26 tls_sockets.h                    |
| 6.27 tls_tickets.h File Reference     |
| 6.27.1 Detailed Description           |
| 6.27.2 Function Documentation         |
| 6.27.2.1 parseTicket()                |
| 6.27.2.2 initTicketContext()          |
| 6.27.2.3 endTicketContext()           |
| 6.27.2.4 ticket_still_good()          |
| 6.28 tls_tickets.h                    |
| 6.29 tls_wifi.h File Reference        |
| 6.29.1 Detailed Description           |
| 6.30 tls_wifi.h                       |
| 6.31 tls_x509.h File Reference        |
| 6.31.1 Detailed Description           |
| 6.31.2 Macro Definition Documentation |
| 6.31.2.1 X509_ECC                     |
| 6.31.2.2 X509_RSA                     |
| 6.31.2.3 X509_ECD                     |
| 6.31.2.4 X509_PQ                      |
| 6.31.2.5 X509_HY                      |
| 6.31.2.6 X509_H256                    |
| 6.31.2.7 X509_H384                    |
| 6.31.2.8 X509_H512                    |
| 6.31.2.9 USE_NIST256                  |
| 6.31.2.10 USE_C25519                  |
| 6.31.2.11 USE_NIST384                 |
| 6.31.2.12 USE_NIST521                 |
| 6.31.3 Function Documentation         |
| 6.31.3.1 ecdsa_sig_encode()           |
| 6.31.3.2 ecdsa_sig_decode()           |
| 6.31.3.3 X509_extract_private_key()   |
| 6.31.3.4 X509_extract_cert_sig()      |
| 6.31.3.5 X509_extract_cert()          |
| 6.31.3.6 X509_find_public_key()       |
| 6.31.3.7 X509_get_public_key()        |
| 6.31.3.8 X509_extract_public_key()    |
| 6.31.3.9 X509_find_issuer()           |
| 6.31.3.10 X509_find_validity()        |
| 6.31.3.11 X509_find_subject()         |
| 6.31.3.12 X509_self_signed()          |

| 6.31.3.13 X509_find_entity_property() | 148 |
|---------------------------------------|-----|
| 6.31.3.14 X509_find_start_date()      | 149 |
| 6.31.3.15 X509_find_expiry_date()     | 149 |
| 6.31.3.16 X509_find_extensions()      | 149 |
| 6.31.3.17 X509_find_extension()       | 150 |
| 6.31.3.18 X509_find_alt_name()        | 150 |
| 6.31.4 Variable Documentation         | 150 |
| 6.31.4.1 X509_CN                      | 151 |
| 6.31.4.2 X509_ON                      | 151 |
| 6.31.4.3 X509_EN                      | 151 |
| 6.31.4.4 X509_LN                      | 151 |
| 6.31.4.5 X509_UN                      | 151 |
| 6.31.4.6 X509_MN                      | 151 |
| 6.31.4.7 X509_SN                      | 151 |
| 6.31.4.8 X509_AN                      | 151 |
| 6.31.4.9 X509_KU                      | 152 |
| 6.31.4.10 X509_BC                     | 152 |
| 6.32 tls_x509.h                       | 152 |
| 6.33 ECCX08.h                         | 153 |
| Index                                 | 155 |

## TIIGER TLS C++

This C++ project implements a TLS1.3 client. There is also a Rust version available from this site. This C++ version is really just C plus namespaces plus pass-by-reference. These are the only features of C++ that are used. Documentation can be found in the doxygen generated file doc/refman.pdf

## 1.1 Building

The TLS library is designed to support crypto agility by allowing a mix of cryptographic providers. This functionality is provided by the SAL (Security Abstraction Layer). Below are two examples to choose from. The SAL API documentation is provided in sal/sal.pdf, and guided by this it should be possible to create your own SAL. To build the client on an IoT node like the Arduino RP2040, see the readme file in the src/arduino directory.

Private keys, server/client certificate chains, and CA root stores are all fixed in the code.

Ideally keys, chains and key stores should be kept in external files, but in an IoT setting there may not be a file system. In this C++ code the client private key and certificate (only required for client-side authentication) are stored in the source code file *tls\_client\_cert.cpp*. The root certificate store is stored in the file *tls\_cacert.cpp*. When using secure hardware, the client private key may not be embedded in the source code, rather it exists in secure on-board memory.

The installation process requires the *cmake* utility to be installed. Copy all files and subdirectories from this directory to a working directory.

#### 1.1.1 Miracl

This build gets all of it cryptography from the MIRACL core library https://github.com/miracl/core/cpp

bash ./scripts/build.sh -1

### 1.1.2 Miracl + LibSodium

bash ./scripts/build.sh -2

2 TIIGER TLS C++

## 1.2 Try it out

After the build complete successfully, the example executable *client* and the TiigerTLS library *libtiitls.a* are generated in the build directory

To see the Security Abstraction Layer (SAL) capabilities, navigate to the build directory

```
./client -s
```

#### To connect to a Website

```
./client swifttls.org
```

Signature Algorithm is RSA\_PKCS1\_SHA256 Cert Signature Verification succeeded

Root Certificate sig is OK

The output should (if VERBOSITY has been set to IO DEBUG in tls1 3.h) look something like this

```
Hostname= swifttls.org
Private key= 0373AF7D060E0E80959254DC071A068FCBEDA5F0C1B6FFFC02C7EB56AE6B00CD
Client Public key= 93CDD4247C90CBC1920E53C4333BE444C0F13E96A077D8D1EF485FE0F9D9D703
Client Hello sent
Cipher Suite is TLS AES 128 GCM SHA256
Server HelloRetryRequest= 020000540303CF21AD74E59A6111BE1D8C021E65B891C2A211167ABB8C5E079E09E2C8A8339C20557742
Client Hello re-sent
Server Hello= 020000970303268C697006F0AC66287680A88C6DB34C2804CD9884B2B0BD087A0F3DE2495F5120A0E658C6A5BB912768
Parsing serverHello
Cipher Suite is TLS_AES_128_GCM_SHA256
Key Exchange Group is SECP256R1
Server Public Key= 04F87B11F808F92B9D4DAE8AE83389257F04B3697181F3CD1479B7214E7D76B108B650A57494D15C5F673EDB05I
Shared Secret= 99A5F3B6F8BE0938AB6D74A99E8FD42DEFD71F25445BD703F0D429DA6CC4AA12
Handshake Secret= 093388E25C3F8468DF3A0544683036CBACF5157874CE995C080807559834CBCA
Client handshake traffic secret= 5B383ED973C7324E267B16A1A7507C380846FFB5397B41E3199C305C23A2C430
Server handshake traffic secret= 71A23E7184F1AA8F228504D3FA735EC8E70FFEC54E0922D553A64800A32C2853
Warning - ALPN extension NOT acknowledged by server
Server Name NOT acknowledged
Max frag length request NOT acknowledged
Encrypted Extensions Processed
Certificate Chain Length= 2458
Parsing Server certificate
Signature is 0A5C155DB6DD9F7F6ABE005D351D6E3FF9DEBA799F7479BD33E1C784B63CB4CA695A76815C9B666C24B6E989EE85009A6
RSA signature of length 2048
Public key= E2AB76AE1A676E3268E39BB9B8AE9CA19DD8BC0BFED0A4275E13C191D716794B48F47766A6B6AD17F19764F48D459E8271
RSA public key of length 2048
Issuer is R3/Let's Encrypt/
Subject is swifttls.org//
Parsing Intermediate certificate
Signature is D94CE0C9F584883731DBBB13E2B3FC8B6B62126C58B7497E3C02B7A81F2861EBCEE02E73EF49077A35841F1DAD68F0D8F
RSA signature of length 2048
Public key= BB021528CCF6A094D30F12EC8D5592C3F882F199A67A4288A75D26AAB52BB9C54CB1AF8E6BF975C8A3D70F479414553557
RSA public key of length 2048
Issuer is DST Root CA X3/Digital Signature Trust Co./
Subject is R3/Let's Encrypt/
Public key = BB021528CCF6A094D30F12EC8D5592C3F882F199A67A4288A75D26AAB52BB9C54CB1AF8E6BF975C8A3D70F47941455355
Checking Signature on Cert
Signature Algorithm is RSA_PKCS1_SHA256
Cert Signature Verification succeeded
Intermediate Certificate Chain sig is OK
Public Key from root cert= DFAFE99750088357B4CC6265F69082ECC7D32C6B30CA5BECD9C37DC740C118148BE0E83376492AE33F2
Public key = DFAFE99750088357B4CC6265F69082ECC7D32C6B30CA5BECD9C37DC740C118148BE0E83376492AE33F214993AC4E0EAF3
Checking Signature on Cert
```

1.2 Try it out 3

```
Certificate Chain is valid
Transcript Hash (CH+SH+EE+CT) = 7CECF69D794C20FB7551BA5C4B986E1F501011328225CDD740A8EB54B728E31B
Transcript Hash (CH+SH+EE+SCT+SCV) = 8EC0EE587717BAEB401992622E3F31CBE151CC6C489104E68B5A83E96284E1E7
Server Certificate Signature= B5B74CF6026CF16FA866BA7E7562C53F67A74949FF040319B0BD2149CF4EF97CAD482463F1746D20
Signature Algorithm is RSA_PSS_RSAE_SHA256
Server Cert Verification OK
Server Data is verified
Transcript Hash (CH+SH+EE+SCT+SCV+SF+[CCT+CSV]) = 299C505CBD66E8CCCF1934AC5398EFAB7DCF239D9A9C95CF0A5384B5902E
Client Verify Data= 9D20AD7C24238C5B77B72D40EC355C41C5859B6851639EA9920986EDF50DF032
Transcript Hash (CH+SH+EE+SCT+SCV+SF+[CCT+CSV]+CF) = 50AC5EA2A163FD5A3CE92D7D98E8CB56D763514148A30213784612F9E
Client application traffic secret= 7DE3D4B470FBCA72FEECBA1A1B938F4AF85F0E4D84C8E06E4218A92DF3EE67CF
Server application traffic secret= 11FFA6345BE788BBF8C1948E4F499D852A07A77B74C74F560BC9E399AB41ABC8
Full Handshake concluded
... after handshake resumption
Sending Application Message
GET / HTTP/1.1
Host: swifttls.org
Waiting for Server input
Got a ticket
Waiting for Server input
```

To attempt a fast resumption, based on a resumption ticket (generated and stored in a file cookie.txt), connect again

./client swifttls.org

Connection closed

#### The output should look something like

Alert sent to Server - Close notify

```
Attempting resumption
Hostname= swifttls.org
Parsing Ticket
Ticket = 6CE7CD561F03F6E3CDD9A0DD4A7F37181861F51A17E8FF6930AAA02C6C5DAFD9
life time in minutes = 3600
Pre-Shared Key = 41301AAD7DAADCF43D700CD71E1198DD2C8DFF5C61B91BEA35116B96762C8B7E
max_early_data = 40960
PSK= 41301AAD7DAADCF43D700CD71E1198DD2C8DFF5C61B91BEA35116B96762C8B7E
Binder Key= 3CC796B38A7FEB226D9B0CD6B6BB4994253298DDF9FF43060C5C30834D75EE79
Early Secret= 610B9D95E512F6E199046C93E600D5CE10BB98517F9A81096E653C13B2D0F17D
Private key= 7373AF7D060E0E80959254DC071A06905E067B07367C49D86B48A10F3923CC49
Client Public key= 04EA04CDA74C1A1942BB8C56C0BD8AE1A4CB9D9B76B5AC64C24CFE7C367B46FA6F06037D945835019D3F1220803
Ticket age= feff
obfuscated age = 447e2e62
Client Hello sent
BND= 258FA2CE9D69253C83646641266B2A81FCEED47348D60E0C7BBB27D2557D1BD2
Sending Binders
Client Early Traffic Secret= CF7D980E8213205CFD35C2194FB75F6D1E98215860BB1F7FA5CFDC8DAE48E9F5
Sending some early data
Sending Application Message
GET / HTTP/1.1
Host: swifttls.org
Parsing serverHello
Cipher Suite is TLS_AES_128_GCM_SHA256
Key Exchange Group is SECP256R1
PSK Identity= 0
Server Public Key= 0401D908F018811AF140E2D417EB2713492C146C2B73F78A81DEC6C3F6E2A31D5114207D93EC92AEB03D64DAD11
{\tt serverHello=0200009D0303268C69B38026464DFFE72A496662627EC35798DA3F98437042E39CAF404C888520557742FB051C8ADC0A48BBC0A48BBC0A48BBC0A48BBC0A48BBC0A48BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4BBC0A4
```

TIIGER TLS C++

```
Shared Secret= 8C7784C539C0144B8FADCBF065637418F190C49995E79660919E204F05287C2D
Handshake Secret= 4025A7EE2C1B634C9FC83FDF5CFB2FCB5498EA3F5D019EEDC6D3C1D751C87C47
Client handshake traffic secret= 5FC1307F4E7ED84B4196B83EA19D69724812C25A571061FB53B5B6E9FD7FCABE
Server handshake traffic secret= 1E84FEBA7F8D75F756408906C608925F9A6445292BA614BB398E634CF5854B2A
Early Data Accepted
Warning - ALPN extension NOT acknowledged by server
Server Name NOT acknowledged
Max frag length request NOT acknowledged
Transcript Hash (CH+SH+EE) = DCB73D7B5416D91546EF7D625FBB6A84105CCCE5F054D753275325A822D394E9
Send End of Early Data
Transcript Hash (CH+SH+EE+SF+ED) = FE1FADC8085B3B41A9146647FC9A40F6F2A303533B237112564A2F51F82B64C4
Server Data is verified
Client Verify Data= 350E968A15D36F16BC20D80789E9DB2792A2975765F9BE537407165F7E7366B8
Client application traffic secret= 536F912C98CF4C2D9672DEA57AC8136519607014EFEBBA289FCED97929EA9633
Server application traffic secret= 6B797DBC7FB2D9F75A877F1D34EE7CACC6D65C847C085331F8941C81F2884E83
Resumption Handshake concluded
Early data was accepted
Waiting for Server input
Alert sent to Server - Close notify
Connection closed
```

Try it out on your favourite websites. It will abort if TLS1.3 is not supported. In a small number of cases it will fail due to receiving a malformed certificate chain from the Server. It is not forgiving of badly formed certificate chains, and makes no attempt to fix them.

#### Also try

```
./client tls13.1d.pw
```

Try it a few times - it randomly asks for a HelloRetryRequest and a Key Update, testing this code (but it does not allow resumption). This site also requires that short records are not padded (that is PAD\_SHORT\_RECORDS is not defined in *tls1 3.h*).

A resumption ticket can be deleted by

```
./client -r
```

See doc/list.txt for some websites that work OK and test different functionality.

### 1.2.1 Client side Authentication

A self-signed client certificate and private key can be generated by

```
openssl req -x509 -nodes -days 365 -newkey ec:<(openssl ecparam -name secp256r1) -keyout mykey.pem -out mycert
```

and inserted into the file tls client cert.cpp

A way to test less common options is to set up a local openssl server. First generate a self-signed server certificate using something like

```
openssl req -x509 -nodes -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365
```

#### then for example

```
openssl s_server -tls1_3 -key key.pem -cert cert.pem -accept 4433 -www
```

acts as a normal Website, while

```
openssl s_server -tls1_3 -verify 0 -key key.pem -cert cert.pem -accept 4433 -www
```

looks for client side certificate authentication - and the server makes a Certificate Request to the client. We can't control the openssl debug output, but its better than nothing! The client connects to this local server via

```
./client localhost
```

1.2 Try it out 5

## 1.2.2 Testing Pre-shared keys

Again we will use OpenSSL to mimic a TLS1.3 server

openssl s\_server -tls1\_3 -cipher PSK-AES128-GCM-SHA256 -psk\_identity 42 -psk 0102030405060708090a0b0c0d0e0f10

and connect via

./client -p 42 localhost

An important setting in *tls1\_3.h* is CRYPTO\_SETTING. For the above tests is assumed that this is set to the default TYPICAL, which allows interaction with standard websites. However it may also be set to TINY\_ECC, POST\_QUANTUM and HYBRID. These last three support interaction with our own rust server. This setting impacts code size and memory resource allocation. It also controls the type of the self-signed certificate provided by the client if it is asked to authenticate.

The client choice of key exchange algorithms, and their preferred ordering, is set in the sal (*tls\_sal.cpp*). The chosen CRYPTO\_SETTING impacts on this ordering. With the default setting the X25519 elliptic curve is preferred.

6 TIIGER TLS C++

# **Configure the Arduino Nano RP2040**

This build is specifically for the Arduino Nano version of the Raspberry Pi Pico (RP2040). Please use version 2.x.x of the Arduino IDE.

First the board needs to be initialised and locked. To do this install the ArduinoECCX08 library and run the ECCX08SelfSignedCert example program.

(This example program appears when an MKR1000 board is suggested, and may not appear for the RP2040. However it runs fine on the RP2040).

This program (a) locks the board, and (b) generates a self-signed X.509 certificate, with an associated private key hidden in Slot 0. Copy the self-signed certificate and place it into tls client cert.cpp where indicated.

Note that the ECC608A chip does a lot of the heavy crypto lifting, especially if the secp256r1 curve is used for certificate signature verification.

The key exchange secret is generated in Slot 1. Slot 9 is used for the HMAC calculation. See the ECC608A documentation for more detail.

## 2.1 Building the client application on the Arduino Nano RP2040 board.

- 1. Create working directory directory with name tiitls
- 2. Copy in all from the cpp directory of <a href="https://github.com/miracl/core">https://github.com/miracl/core</a>
- 3. Copy in all from the arduino directory of https://github.com/miracl/core
- 4. (If ever asked to overwrite a file, go ahead and overwrite it)
- 5. Copy in all of the TLS1.3 C++ code from the lib/, lib/ibe, include/, sal/ and src/arduino directories (but not from subdirectories)
- 6. Edit the file core.h to define CORE\_ARDUINO (line 31)
- 7. Edit the file tls\_octads.h to define TLS\_ARDUINO (line 13).
- 8. Edit tls1\_3.h. Define VERBOSITY as IO\_DEBUG for more debug output. Decide on CRYPTO\_SETTING. Stack only, or Stack plus heap.
- 9. Edit the file client.cpp to set your wifi SSID and password (near line 150)

- 10. Run py config.py, and select options 2, 8, 31, 41 and 43. This creates the default SAL (in this case using miracl + ECC608A hardware).
- 11. Drop the working directory into where the Arduino IDE expects it.
- 12. (In the IDE select File->Preferences and find the Sketchbook location its the libraries directory off that.)
- 13. Open the Arduino app, and look in File->Examples->tiitls, and look for the example "client"
- 14. Upload to the board and run it. Open Tools->Serial Monitor to see the output.
- 15. Enter URL (e.g. www.bbc.co.uk) when prompted, and press return. A full TLS1.3 handshake followed by a resumption is attempted.
- 16. Click on Clear Output and Send to repeat for a different URL (or click Send again to see SAL capabilities).

or before executing step 10, search for \$\*\\$\*\\$\*\\$\* in config.py (around line 1020) and make changes as indicated. If using miracl alone, without hardware support, option 3 must be selected as well. If using assembly language code for X25519, copy x25519.S from <a href="https://github.com/pornin/x25519-cm0/blob/main/src/x25519-cm0.">https://github.com/pornin/x25519-cm0/blob/main/src/x25519-cm0.</a> into working directory and remove option 2. This creates the SAL (in this case using miracl + ECC608A hardware + Pornin's x25519). If experimenting with post-quantum primitives, also select options 45 and 46, for Dilithium and Kyber support.

The example TLS1.3 client code first connects to the wireless network, and after that it should connect to standard websites, as long as they support TLS1.3. The example program first makes a full TLS handshake, and exits after receiving some HTML from the server. Then after a few seconds, if it has received a resumption ticket, it attempts a resumption handshake.

The client can also be run in conjunction with our Rust server. Make sure that the CRYPTO\_SETTING parameter is the same for both client and server. In our experimental set-up, the rust server runs from Windows, looking for connections on port 4433. Run ipconfig to get the IP address of the server on the local network, which might look something like 192.168.1.186. Then run the client from the Arduino IDE, and when prompted enter for example 192.168.1.186:4433. The client should now connect to the server.

# **Data Structure Index**

## 3.1 Data Structures

Here are the data structures with brief descriptions:

| crypto   |  |    |
|----------|--|----|
|          | Crypto context structure   | 13 |
| ECCX08   | Class  | 15 |
| ee_statu | IS Control of the con |    |
|          | Server encrypted extensions expectations/responses   | 15 |
| octad    |  |    |
|          | Safe representation of an octad  | 16 |
| pktype   |  |    |
|          | Public key type  | 17 |
| ret      |  |    |
|          | Function return structure  | 18 |
| Socket   |  |    |
|          | Socket instance  | 19 |
| ticket   |  |    |
|          | Ticket context structure   | 20 |
| TLS_ses  | ssion  |    |
|          | TLS1.3 session state   | 22 |
| unihash  |  |    |
|          | Universal Hash Function  | 27 |

10 Data Structure Index

# File Index

## 4.1 File List

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

| tls1_3.h   |
|--|
| Main TLS 1.3 Header File for constants and structures          |
| tls_bfibe.h  |
| Boneh and Franklin IBE   |
| tls_cert_chain.h   |
| Process Certificate Chain                                      |
| tls_certs.h  |
| Certificate Authority root certificate store                   |
| tls_client_recv.h  |
| Process Input received from the Server                         |
| tls_client_send.h  |
| Process Output to be sent to the Server                        |
| tls_keys_calc.h  |
| TLS 1.3 crypto support functions                               |
| tls_logger.h   |
| TLS 1.3 logging  |
| tls_octads.h   |
| Octad handling routines - octads don't overflow, they truncate |
| tls_pqibe.h  |
| Ducas et al. IBE   |
| tls_protocol.h   |
| TLS 1.3 main client-side protocol functions                    |
| tls_sal.h  |
| Security Abstraction Layer for TLS                             |
| tls_sockets.h  |
| Set up sockets for reading and writing                         |
| tls_tickets.h  |
| TLS 1.3 process resumption tickets                             |
| tls_wifi.h   |
| Define Socket structure depending on processor context         |
| tls_x509.h   |
| X509 function Header File                                      |
| ECCX08.h   |

12 File Index

## **Data Structure Documentation**

## 5.1 crypto Struct Reference

crypto context structure

#include <tls1\_3.h>

## **Data Fields**

- bool active
- char k [TLS\_MAX\_KEY]
- char iv [12]
- octad K
- octad IV
- unsign32 record
- int suite
- int taglen

## 5.1.1 Detailed Description

crypto context structure

## 5.1.2 Field Documentation

### 5.1.2.1 active

bool crypto::active

Indicates if encryption has been activated

## 5.1.2.2 k

```
char crypto::k[TLS_MAX_KEY]
```

AEAD cryptographic Key bytes

### 5.1.2.3 iv

```
char crypto::iv[12]
```

AEAD cryptographic IV bytes

#### 5.1.2.4 K

```
octad crypto::K
```

Key as octad

## 5.1.2.5 IV

```
octad crypto::IV
```

IV as octad

## 5.1.2.6 record

```
unsign32 crypto::record
```

current record number - to be incremented

## 5.1.2.7 suite

```
int crypto::suite
```

Cipher Suite

## 5.1.2.8 taglen

```
int crypto::taglen
```

## Tag Length

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

• tls1\_3.h

## 5.2 ECCX08Class Class Reference

#### **Public Member Functions**

- ECCX08Class (TwoWire &wire, uint8\_t address)
- int begin ()
- void end ()
- int serialNumber (byte sn[])
- String serialNumber ()
- long random (long max)
- long random (long min, long max)
- int random (byte data[], size t length)
- int generatePrivateKey (int slot, byte publicKey[])
- int generatePublicKey (int slot, byte publicKey[])
- int generateSharedKey (int slot, byte publicKey[], byte sharedKey[])
- int ecdsaVerify (const byte message[], const byte signature[], const byte pubkey[])
- int ecSign (int slot, const byte message[], byte signature[])
- int challenge (const byte message[])
- int aesEncrypt (byte block[])
- int aesGFM (byte state[], byte H[])
- int beginSHA256 ()
- int beginHMAC (int slot)
- int updateSHA256 (const byte data[], int len)
- int endSHA256 (byte result[])
- int endSHA256 (const byte data[], int length, byte result[])
- int readSHA256 (byte context[])
- int writeSHA256 (byte context[], int length)
- int readSlot (int slot, byte data[], int length)
- int writeSlot (int slot, const byte data[], int length)
- · int locked ()
- int writeConfiguration (const byte data[])
- int readConfiguration (byte data[])
- int lock ()

The documentation for this class was generated from the following files:

- ECCX08.h
- ECCX08.cpp

## 5.3 ee status Struct Reference

server encrypted extensions expectations/responses

```
#include <tls1_3.h>
```

### **Data Fields**

- · bool early\_data
- bool alpn
- bool server\_name
- bool max\_frag\_length

## 5.3.1 Detailed Description

server encrypted extensions expectations/responses

## 5.3.2 Field Documentation

## 5.3.2.1 early\_data

```
bool ee_status::early_data
```

true if early data accepted

### 5.3.2.2 alpn

```
bool ee_status::alpn
```

true if ALPN accepted

### 5.3.2.3 server\_name

```
bool ee_status::server_name
```

true if server name accepted

### 5.3.2.4 max\_frag\_length

```
bool ee_status::max_frag_length
```

true if max frag length respected

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

• tls1\_3.h

## 5.4 octad Struct Reference

Safe representation of an octad.

```
#include <tls_octads.h>
```

## **Data Fields**

- int len
- int max
- char \* val

## 5.4.1 Detailed Description

Safe representation of an octad.

## 5.4.2 Field Documentation

### 5.4.2.1 len

int octad::len

length in bytes

### 5.4.2.2 max

int octad::max

max length allowed - enforce truncation

### 5.4.2.3 val

char\* octad::val

byte array

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

• tls\_octads.h

## 5.5 pktype Struct Reference

Public key type.

#include <tls\_x509.h>

## **Data Fields**

- int type
- int hash
- int curve

## 5.5.1 Detailed Description

Public key type.

## 5.5.2 Field Documentation

### 5.5.2.1 type

```
int pktype::type
```

signature type (ECC or RSA)

### 5.5.2.2 hash

int pktype::hash

hash type

## 5.5.2.3 curve

```
int pktype::curve
```

elliptic curve used or RSA key length in bits

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

• tls\_x509.h

## 5.6 ret Struct Reference

function return structure

#include <tls1\_3.h>

5.7 Socket Class Reference 19

## **Data Fields**

- unsign32 val
- int err

## 5.6.1 Detailed Description

function return structure

#### 5.6.2 Field Documentation

#### 5.6.2.1 val

unsign32 ret::val

return value

### 5.6.2.2 err

int ret::err

error return

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

• tls1\_3.h

## 5.7 Socket Class Reference

Socket instance.

```
#include <tls_sockets.h>
```

## **Public Member Functions**

- bool connect (char \*host, int port)
- void setTimeout (int to)
- int write (char \*buf, int len)
- int read (char \*buf, int len)
- void stop ()

## **Static Public Member Functions**

- static Socket InetSocket ()
- static Socket UnixSocket ()

## 5.7.1 Detailed Description

Socket instance.

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

· tls sockets.h

## 5.8 ticket Struct Reference

ticket context structure

```
#include <tls1_3.h>
```

### **Data Fields**

- bool valid
- char tick [TLS\_MAX\_TICKET\_SIZE]
- char nonce [256]
- char psk [TLS\_MAX\_HASH]
- octad TICK
- octad NONCE
- octad PSK
- unsign32 age\_obfuscator
- unsign32 max\_early\_data
- unsign32 birth
- int lifetime
- int cipher\_suite
- int favourite\_group
- int origin

## 5.8.1 Detailed Description

ticket context structure

### 5.8.2 Field Documentation

5.8 ticket Struct Reference 21

### 5.8.2.1 valid

bool ticket::valid

Is ticket valid?

#### 5.8.2.2 tick

char ticket::tick[TLS\_MAX\_TICKET\_SIZE]

Ticket bytes

### 5.8.2.3 nonce

char ticket::nonce[256]

nonce

#### 5.8.2.4 psk

char ticket::psk[TLS\_MAX\_HASH]

pre-shared key

## 5.8.2.5 TICK

octad ticket::TICK

Ticket or external PSK label as octad

# 5.8.2.6 NONCE

octad ticket::NONCE

Nonce as octad

# 5.8.2.7 PSK

octad ticket::PSK

PSK as octad

# 5.8.2.8 age\_obfuscator

unsign32 ticket::age\_obfuscator

ticket age obfuscator - 0 for external PSK

### 5.8.2.9 max\_early\_data

```
unsign32 ticket::max_early_data
```

Maximum early data allowed for this ticket

### 5.8.2.10 birth

```
unsign32 ticket::birth
```

Birth time of this ticket

#### 5.8.2.11 lifetime

```
int ticket::lifetime
```

ticket lifetime

### 5.8.2.12 cipher\_suite

```
int ticket::cipher_suite
```

Cipher suite used

### 5.8.2.13 favourite\_group

```
int ticket::favourite_group
```

the server's favourite group

# 5.8.2.14 origin

```
int ticket::origin
```

Origin of initial handshake - Full or PSK?

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

• tls1\_3.h

# 5.9 TLS\_session Struct Reference

TLS1.3 session state.

#include <tls1\_3.h>

## **Data Fields**

- · int status
- int max\_record
- Socket \* sockptr
- char id [32]
- char hostname [TLS\_MAX\_SERVER\_NAME]
- int cipher\_suite
- int favourite\_group
- · int server\_cert\_type
- int client\_cert\_type
- crypto K\_send
- crypto K\_recv
- octad HS
- char hs [TLS\_MAX\_HASH]
- octad RMS
- char rms [TLS\_MAX\_HASH]
- octad STS
- char sts [TLS\_MAX\_HASH]
- octad CTS
- char cts [TLS\_MAX\_HASH]
- octad CTX
- char ctx [TLS\_MAX\_HASH]
- octad IBUFF
- octad OBUFF
- char ibuff [TLS\_MAX\_IBUFF\_SIZE]
- char obuff [TLS\_MAX\_OBUFF\_SIZE]
- int ptr
- · unihash tlshash
- ticket T

# 5.9.1 Detailed Description

TLS1.3 session state.

# 5.9.2 Field Documentation

### 5.9.2.1 status

int TLS\_session::status

Connection status

### 5.9.2.2 max\_record

int TLS\_session::max\_record

max record size I should send

### 5.9.2.3 sockptr

```
Socket* TLS_session::sockptr
```

Pointer to socket

#### 5.9.2.4 id

```
char TLS_session::id[32]
```

Session ID

### 5.9.2.5 hostname

```
char TLS_session::hostname[TLS_MAX_SERVER_NAME]
```

Server name for connection

## 5.9.2.6 cipher\_suite

```
int TLS_session::cipher_suite
```

agreed cipher suite

# 5.9.2.7 favourite\_group

```
int TLS_session::favourite_group
```

favourite key exchange group - may be changed on handshake retry

### 5.9.2.8 server\_cert\_type

```
int TLS_session::server_cert_type
```

server certificate type

# 5.9.2.9 client\_cert\_type

```
int TLS_session::client_cert_type
```

client certificate type

# 5.9.2.10 K\_send

```
crypto TLS_session::K_send
```

Sending Key

## 5.9.2.11 K\_recv

crypto TLS\_session::K\_recv

Receiving Key

#### 5.9.2.12 HS

octad TLS\_session::HS

Handshake secret

### 5.9.2.13 hs

char TLS\_session::hs[TLS\_MAX\_HASH]

Handshake secret data

#### 5.9.2.14 RMS

octad TLS\_session::RMS

**Resumption Master Secret** 

### 5.9.2.15 rms

char TLS\_session::rms[TLS\_MAX\_HASH]

Resumption Master Secret data

### 5.9.2.16 STS

octad TLS\_session::STS

Server Traffic secret

# 5.9.2.17 sts

char TLS\_session::sts[TLS\_MAX\_HASH]

Server Traffic secret data

### 5.9.2.18 CTS

octad TLS\_session::CTS

Client Traffic secret

### 5.9.2.19 cts

```
char TLS_session::cts[TLS_MAX_HASH]
```

Client Traffic secret data

#### 5.9.2.20 CTX

```
octad TLS_session::CTX
```

Certificate Request Context

### 5.9.2.21 ctx

```
char TLS_session::ctx[TLS_MAX_HASH]
```

Certificate Request Context data

#### 5.9.2.22 IBUFF

```
octad TLS_session::IBUFF
```

Main input buffer for this connection

## 5.9.2.23 OBUFF

```
octad TLS_session::OBUFF
```

output buffer for this connection

### 5.9.2.24 ibuff

```
char TLS_session::ibuff[TLS_MAX_IBUFF_SIZE]
```

Byte array for main input buffer for this connection

### 5.9.2.25 obuff

```
char TLS_session::obuff[TLS_MAX_OBUFF_SIZE]
```

output buffer for this connection

# 5.9.2.26 ptr

```
int TLS_session::ptr
```

pointer into IBUFF buffer

### 5.9.2.27 tlshash

```
unihash TLS_session::tlshash
```

Transcript hash recorder

#### 5.9.2.28 T

```
ticket TLS_session::T
```

resumption ticket

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

• tls1\_3.h

# 5.10 unihash Struct Reference

Universal Hash Function.

```
#include <tls1_3.h>
```

## **Data Fields**

- char state [TLS\_MAX\_HASH\_STATE]
- int htype

# 5.10.1 Detailed Description

Universal Hash Function.

# 5.10.2 Field Documentation

### 5.10.2.1 state

```
char unihash::state[TLS_MAX_HASH_STATE]
```

hash function state

### 5.10.2.2 htype

```
int unihash::htype
```

The hash type (typically SHA256)

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

• tls1\_3.h

# **Chapter 6**

# **File Documentation**

# 6.1 tls1\_3.h File Reference

Main TLS 1.3 Header File for constants and structures.

```
#include <stdint.h>
#include "tls_octads.h"
#include "tls_sockets.h"
```

# **Data Structures**

struct ret

function return structure

• struct ee\_status

server encrypted extensions expectations/responses

• struct crypto

crypto context structure

struct ticket

ticket context structure

· struct unihash

Universal Hash Function.

• struct TLS\_session

TLS1.3 session state.

### **Macros**

- #define IO\_NONE 0
- #define IO\_APPLICATION 1
- #define IO\_PROTOCOL 2
- #define IO\_DEBUG 3
- #define IO WIRE 4
- #define TINY\_ECC 0
- #define TYPICAL 1
- #define POST\_QUANTUM 2

- #define HYBRID 3
- #define NOCERT 0
- #define RSA SS 1
- #define ECC SS 2
- #define DLT SS 3
- #define HYB SS 6
- #define HW 14
- #define HW\_2 5
- #define VERBOSITY IO\_PROTOCOL
- #define THIS YEAR 2023
- #define POST HS AUTH
- #define CLIENT CERT ECC SS
- #define CRYPTO\_SETTING TYPICAL
- #define TLS APPLICATION PROTOCOL (char \*)("http/1.1")
- #define ALLOW\_SELF\_SIGNED
- #define TRY EARLY DATA
- #define TLS SHA256 T 1
- #define TLS SHA384 T 2
- #define TLS SHA512 T 3
- #define TLS\_MAX\_HASH\_STATE 768
- #define TLS\_MAX\_HASH 64
- #define TLS MAX KEY 32
- #define TLS X509 MAX FIELD 256
- #define TLS\_MAX\_EXT\_LABEL 256
- #define TLS MAX FRAG 2
- #define TLS\_MAX\_IBUFF\_SIZE (16384+256)
- #define TLS MAX PLAIN FRAG 16384
- #define TLS\_MAX\_CIPHER\_FRAG (16384+256)
- #define TLS\_MAX\_CERT\_SIZE 2048
- #define TLS\_MAX\_CERT\_B64 2800
- #define TLS MAX HELLO 1024
- #define TLS MAX SIG PUB KEY SIZE 512
- #define TLS\_MAX\_SIG\_SECRET\_KEY\_SIZE 512
- #define TLS\_MAX\_SIGNATURE\_SIZE 512
- #define TLS\_MAX\_KEX\_PUB\_KEY\_SIZE 97
- #define TLS\_MAX\_KEX\_CIPHERTEXT\_SIZE 97
- #define TLS\_MAX\_KEX\_SECRET\_KEY\_SIZE 48
- #define TLS\_MAX\_SERVER\_CHAIN\_LEN 2
- #define TLS\_MAX\_SERVER\_CHAIN\_SIZE (TLS\_MAX\_SERVER\_CHAIN\_LEN\*TLS\_MAX\_CERT\_SIZE)
- #define TLS MAX CLIENT CHAIN LEN 1
- #define TLS MAX CLIENT CHAIN SIZE (TLS MAX CLIENT CHAIN LEN\*TLS MAX CERT SIZE)
- #define TLS\_MAX\_SHARED\_SECRET\_SIZE 256
- #define TLS\_MAX\_TICKET\_SIZE 4196
- #define TLS\_MAX\_EXTENSIONS 6144
- #define TLS\_MAX\_ECC\_FIELD 66
- #define TLS\_MAX\_IV\_SIZE 12
- #define TLS\_MAX\_TAG\_SIZE 16
- #define TLS\_MAX\_COOKIE 128
- #define TLS\_MAX\_OUTPUT\_RECORD\_SIZE 1024
- #define TLS\_MAX\_OBUFF\_SIZE (TLS\_MAX\_OUTPUT\_RECORD\_SIZE+TLS\_MAX\_TAG\_SIZE+6)
- #define TLS MAX SERVER NAME 128
- #define TLS\_MAX\_SUPPORTED\_GROUPS 10
- #define TLS MAX SUPPORTED SIGS 16
- #define TLS MAX PSK MODES 2
- #define TLS\_MAX\_CIPHER\_SUITES 5

- #define TLS\_AES\_128\_GCM\_SHA256 0x1301
- #define TLS\_AES\_256\_GCM\_SHA384 0x1302
- #define TLS\_CHACHA20\_POLY1305\_SHA256 0x1303
- #define TLS\_AES\_128\_CCM\_SHA256 0x1304
- #define TLS AES 128 CCM 8 SHA256 0x1305
- #define X25519 0x001d
- #define SECP256R1 0x0017
- #define SECP384R1 0x0018
- #define SECP521R1 0x0019
- #define X448 0x001e
- #define KYBER768 0x4242
- #define HYBRID KX 0x421d
- #define ECDSA\_SECP256R1\_SHA256 0x0403
- #define ECDSA SECP256R1 SHA384 0x0413
- #define ECDSA\_SECP384R1\_SHA384 0x0503
- #define RSA PSS RSAE SHA256 0x0804
- #define RSA PSS RSAE SHA384 0x0805
- #define RSA PSS\_RSAE\_SHA512 0x0806
- #define RSA\_PKCS1\_SHA256 0x0401
- #define RSA\_PKCS1\_SHA384 0x0501
- #define RSA\_PKCS1\_SHA512 0x0601
- #define ED25519 0x0807
- #define DILITHIUM2 0x0902
- #define DILITHIUM3 0x0903
- #define DILITHIUM2 P256 0x09F2
- #define PSKOK 0x00
- #define PSKWECDHE 0x01
- #define TLS\_FULL\_HANDSHAKE 1
- #define TLS EXTERNAL PSK 2
- #define TLS1\_0 0x0301
- #define TLS1\_2 0x0303
- #define TLS1 3 0x0304
- #define TLS13\_UPDATE\_NOT\_REQUESTED 0
- #define TLS13\_UPDATE\_REQUESTED 1
- #define SERVER\_NAME 0x0000
- #define SUPPORTED\_GROUPS 0x000a
- #define SIG\_ALGS 0x000d
- #define POST\_HANDSHAKE\_AUTH 0x0031
- #define SIG\_ALGS\_CERT 0x0032
- #define KEY SHARE 0x0033
- #define PSK MODE 0x002d
- #define PRESHARED\_KEY 0x0029
- #define TLS\_VER 0x002b
- #define COOKIE 0x002c
- #define EARLY DATA 0x002a
- #define MAX\_FRAG\_LENGTH 0x0001
- #define PADDING 0x0015
- #define APP\_PROTOCOL 0x0010
- #define RECORD\_SIZE\_LIMIT 0x001c
- #define CLIENT\_CERT\_TYPE 0x0013
- #define SERVER CERT TYPE 0x0014
- #define HSHAKE 0x16
- #define APPLICATION 0x17
- #define ALERT 0x15
- #define CHANGE\_CIPHER 0x14

- #define TIMED\_OUT 0x01
- #define CLIENT HELLO 0x01
- #define SERVER\_HELLO 0x02
- #define CERTIFICATE 0x0b
- #define CERT REQUEST 0x0d
- #define CERT\_VERIFY 0x0f
- #define FINISHED 0x14
- #define ENCRYPTED\_EXTENSIONS 0x08
- #define TICKET 0x04
- #define KEY UPDATE 0x18
- #define MESSAGE HASH 0xFE
- #define END OF EARLY DATA 0x05
- #define HANDSHAKE\_RETRY 0x102
- #define NOT TLS1 3-2
- #define BAD\_CERT\_CHAIN -3
- #define ID MISMATCH -4
- #define UNRECOGNIZED EXT -5
- #define BAD\_HELLO -6
- #define WRONG MESSAGE -7
- #define MISSING\_REQUEST\_CONTEXT -8
- #define AUTHENTICATION\_FAILURE -9
- #define BAD RECORD -10
- #define BAD TICKET -11
- #define NOT\_EXPECTED -12
- #define CA NOT FOUND -13
- #define CERT\_OUTOFDATE -14
- #define MEM OVERFLOW -15
- #define FORBIDDEN\_EXTENSION -16
- #define MAX\_EXCEEDED -17
- #define EMPTY\_CERT\_CHAIN -18
- #define SELF\_SIGNED\_CERT -20
- #define ERROR ALERT RECEIVED -22
- #define BAD\_MESSAGE -23
- #define CERT\_VERIFY\_FAIL -24
- #define BAD\_HANDSHAKE -26
- #define BAD\_REQUEST\_UPDATE -27
- #define CLOSURE\_ALERT\_RECEIVED -28
- #define MISSING\_EXTENSIONS -30
- #define ILLEGAL\_PARAMETER 0x2F
- #define UNEXPECTED MESSAGE 0x0A
- #define DECRYPT ERROR 0x33
- #define BAD\_CERTIFICATE 0x2A
- #define UNSUPPORTED\_EXTENSION 0x6E
- #define UNKNOWN\_CA 0x30
- #define CERTIFICATE EXPIRED 0x2D
- #define PROTOCOL\_VERSION 0x46
- #define DECODE ERROR 0x32
- #define RECORD\_OVERFLOW 0x16
- #define BAD\_RECORD\_MAC 0x14
- #define HANDSHAKE\_FAILURE 0x28
- #define CLOSE NOTIFY 0x00
- #define MISSING\_EXTENSION 0x6D;
- #define LOG\_OUTPUT\_TRUNCATION 256
- #define TLS13 DISCONNECTED 0
- #define TLS13\_CONNECTED 1

- #define TLS13\_HANDSHAKING 2
- #define TLS\_FAILURE 0
- #define TLS SUCCESS 1
- #define TLS\_RESUMPTION\_REQUIRED 2
- #define TLS\_EARLY\_DATA\_ACCEPTED 3
- #define PSK\_NOT 0
- #define PSK\_KEY 1
- #define PSK\_IBE 2
- #define X509\_CERT 0
- #define RAW\_PUBLIC\_KEY 2

# **Typedefs**

- typedef uint8\_t byte
- typedef int8\_t sign8
- typedef int16\_t sign16
- typedef int32\_t sign32
- typedef int64\_t sign64
- typedef uint32\_t unsign32
- typedef uint64\_t unsign64

# 6.1.1 Detailed Description

Main TLS 1.3 Header File for constants and structures.

Author

Mike Scott

### 6.1.2 Macro Definition Documentation

### 6.1.2.1 IO\_NONE

#define IO\_NONE 0

Run silently

# 6.1.2.2 IO\_APPLICATION

#define IO\_APPLICATION 1

just print application traffic

## 6.1.2.3 IO\_PROTOCOL

```
#define IO_PROTOCOL 2
```

print protocol progress + application traffic

## 6.1.2.4 IO\_DEBUG

```
#define IO_DEBUG 3
```

print lots of debug information + protocol progress + application progress

# 6.1.2.5 IO\_WIRE

```
#define IO_WIRE 4
```

print lots of debug information + protocol progress + application progress + bytes on the wire

### 6.1.2.6 TINY\_ECC

```
#define TINY_ECC 0
```

ECC keys only

## 6.1.2.7 TYPICAL

```
#define TYPICAL 1
```

Mixture of RSA and ECC - for use with most standard web servers

### 6.1.2.8 POST\_QUANTUM

```
#define POST_QUANTUM 2
```

Post quantum (Dilithium+Kyber?)

### 6.1.2.9 HYBRID

#define HYBRID 3

Hybrid, Kyber/Dilithium + X25519

### 6.1.2.10 NOCERT

#define NOCERT 0

Don't have a Client Cert

## 6.1.2.11 RSA\_SS

#define RSA\_SS 1

self signed RSA cert

# 6.1.2.12 ECC\_SS

#define ECC\_SS 2

self signed ECC cert

## 6.1.2.13 DLT\_SS

#define DLT\_SS 3

self signed Dilithium cert

## 6.1.2.14 HYB\_SS

#define HYB\_SS 6

self signed Hybrid cert (Dilithium+ECC)

# 6.1.2.15 HW\_1

#define HW\_1 4

RP2040 1 Hardware cert

## 6.1.2.16 HW\_2

#define HW\_2 5

RP2040 2 Hardware cert

# 6.1.2.17 VERBOSITY

#define VERBOSITY IO\_PROTOCOL

Set to level of output information desired - see above

# 6.1.2.18 THIS\_YEAR

#define THIS\_YEAR 2023

Set to this year - crudely used to deprecate old certificates

# 6.1.2.19 POST\_HS\_AUTH

```
#define POST_HS_AUTH
```

Willing to do post handshake authentication

### 6.1.2.20 CLIENT\_CERT

```
#define CLIENT_CERT ECC_SS
```

Indicate capability of authenticating with a cert plus signing key

## 6.1.2.21 CRYPTO\_SETTING

```
#define CRYPTO_SETTING TYPICAL
```

**Determine Cryptography settings** 

## 6.1.2.22 TLS\_APPLICATION\_PROTOCOL

```
#define TLS_APPLICATION_PROTOCOL (char *)("http/1.1")
```

Support ALPN protocol

# 6.1.2.23 ALLOW\_SELF\_SIGNED

```
#define ALLOW_SELF_SIGNED
```

allow self-signed server cert

# 6.1.2.24 TRY\_EARLY\_DATA

```
#define TRY_EARLY_DATA
```

Try to send early data on resumptions

## 6.1.2.25 TLS\_SHA256\_T

#define TLS\_SHA256\_T 1

SHA256 hash

# 6.1.2.26 TLS\_SHA384\_T

#define TLS\_SHA384\_T 2

SHA384 hash

# 6.1.2.27 TLS\_SHA512\_T

#define TLS\_SHA512\_T 3

SHA512 hash

## 6.1.2.28 TLS\_MAX\_HASH\_STATE

#define TLS\_MAX\_HASH\_STATE 768

Maximum memory required to store hash function state

## 6.1.2.29 TLS\_MAX\_HASH

#define TLS\_MAX\_HASH 64

Maximum hash output length in bytes

## 6.1.2.30 TLS\_MAX\_KEY

#define TLS\_MAX\_KEY 32

Maximum key length in bytes

## 6.1.2.31 TLS\_X509\_MAX\_FIELD

#define TLS\_X509\_MAX\_FIELD 256

Maximum X.509 field size

# 6.1.2.32 TLS\_MAX\_EXT\_LABEL

#define TLS\_MAX\_EXT\_LABEL 256

Max external psk label size

## 6.1.2.33 TLS\_MAX\_FRAG

```
#define TLS_MAX_FRAG 2
```

Max Fragment length desired - 1 for 512, 2 for 1024, 3 for 2048, 4 for 4096, 0 for 16384

## 6.1.2.34 TLS\_MAX\_IBUFF\_SIZE

```
#define TLS_MAX_IBUFF_SIZE (16384+256)
```

Maximum Input/Output buffer size. We will want to reduce this as much as possible! But must be large enough to take full certificate chain

### 6.1.2.35 TLS\_MAX\_PLAIN\_FRAG

```
#define TLS_MAX_PLAIN_FRAG 16384
```

Max Plaintext Fragment size

## 6.1.2.36 TLS\_MAX\_CIPHER\_FRAG

```
#define TLS_MAX_CIPHER_FRAG (16384+256)
```

Max Ciphertext Fragment size

# 6.1.2.37 TLS\_MAX\_CERT\_SIZE

```
#define TLS_MAX_CERT_SIZE 2048
```

I checked - current max for root CAs is 2016

## 6.1.2.38 TLS\_MAX\_CERT\_B64

```
#define TLS_MAX_CERT_B64 2800
```

In base64 - current max for root CAs is 2688

### 6.1.2.39 TLS\_MAX\_HELLO

```
#define TLS_MAX_HELLO 1024
```

Max client hello size (less extensions) KEX public key is largest component

### 6.1.2.40 TLS\_MAX\_SIG\_PUB\_KEY\_SIZE

#define TLS\_MAX\_SIG\_PUB\_KEY\_SIZE 512

Max signature public key size in bytes RSA

## 6.1.2.41 TLS\_MAX\_SIG\_SECRET\_KEY\_SIZE

#define TLS\_MAX\_SIG\_SECRET\_KEY\_SIZE 512

Max signature private key size in bytes RSA

## 6.1.2.42 TLS\_MAX\_SIGNATURE\_SIZE

#define TLS\_MAX\_SIGNATURE\_SIZE 512

Max digital signature size in bytes RSA

#### 6.1.2.43 TLS\_MAX\_KEX\_PUB\_KEY\_SIZE

#define TLS\_MAX\_KEX\_PUB\_KEY\_SIZE 97

Max key exchange public key size in bytes ECC

# 6.1.2.44 TLS\_MAX\_KEX\_CIPHERTEXT\_SIZE

#define TLS\_MAX\_KEX\_CIPHERTEXT\_SIZE 97

Max key exchange (KEM) ciphertext size ECC

### 6.1.2.45 TLS\_MAX\_KEX\_SECRET\_KEY\_SIZE

#define TLS\_MAX\_KEX\_SECRET\_KEY\_SIZE 48

Max key exchange private key size in bytes ECC

### 6.1.2.46 TLS\_MAX\_SERVER\_CHAIN\_LEN

#define TLS\_MAX\_SERVER\_CHAIN\_LEN 2

Maximum Server Certificate chain length - omitting root CA

# 6.1.2.47 TLS\_MAX\_SERVER\_CHAIN\_SIZE

#define TLS\_MAX\_SERVER\_CHAIN\_SIZE (TLS\_MAX\_SERVER\_CHAIN\_LEN\*TLS\_MAX\_CERT\_SIZE)

Maximum Server Certificate chain length in bytes

### 6.1.2.48 TLS\_MAX\_CLIENT\_CHAIN\_LEN

```
#define TLS_MAX_CLIENT_CHAIN_LEN 1
```

Maximum Client Certificate chain length - one self signed here

## 6.1.2.49 TLS\_MAX\_CLIENT\_CHAIN\_SIZE

```
#define TLS_MAX_CLIENT_CHAIN_SIZE (TLS_MAX_CLIENT_CHAIN_LEN*TLS_MAX_CERT_SIZE)
```

Maximum Client Certificate chain length in bytes

## 6.1.2.50 TLS\_MAX\_SHARED\_SECRET\_SIZE

```
#define TLS_MAX_SHARED_SECRET_SIZE 256
```

Max key exchange Shared secret size

### 6.1.2.51 TLS\_MAX\_TICKET\_SIZE

```
#define TLS_MAX_TICKET_SIZE 4196
```

maximum resumption ticket size - beware some servers send much bigger tickets!

# 6.1.2.52 TLS\_MAX\_EXTENSIONS

#define TLS\_MAX\_EXTENSIONS 6144

Max extensions size

### 6.1.2.53 TLS\_MAX\_ECC\_FIELD

#define TLS\_MAX\_ECC\_FIELD 66

Max ECC field size in bytes

# 6.1.2.54 TLS\_MAX\_IV\_SIZE

#define TLS\_MAX\_IV\_SIZE 12

Max IV size in bytes

# 6.1.2.55 TLS\_MAX\_TAG\_SIZE

#define TLS\_MAX\_TAG\_SIZE 16

Max HMAC tag length in bytes

### 6.1.2.56 TLS\_MAX\_COOKIE

#define TLS\_MAX\_COOKIE 128

Max Cookie size

## 6.1.2.57 TLS\_MAX\_OUTPUT\_RECORD\_SIZE

#define TLS\_MAX\_OUTPUT\_RECORD\_SIZE 1024

Max output record size

### 6.1.2.58 TLS\_MAX\_OBUFF\_SIZE

#define TLS\_MAX\_OBUFF\_SIZE (TLS\_MAX\_OUTPUT\_RECORD\_SIZE+TLS\_MAX\_TAG\_SIZE+6)

Max output buffer size

#### 6.1.2.59 TLS\_MAX\_SERVER\_NAME

#define TLS\_MAX\_SERVER\_NAME 128

Max server name size in bytes

# 6.1.2.60 TLS\_MAX\_SUPPORTED\_GROUPS

#define TLS\_MAX\_SUPPORTED\_GROUPS 10

Max number of supported crypto groups

### 6.1.2.61 TLS\_MAX\_SUPPORTED\_SIGS

#define TLS\_MAX\_SUPPORTED\_SIGS 16

Max number of supported signature schemes

# 6.1.2.62 TLS\_MAX\_PSK\_MODES

#define TLS\_MAX\_PSK\_MODES 2

Max preshared key modes

# 6.1.2.63 TLS\_MAX\_CIPHER\_SUITES

#define TLS\_MAX\_CIPHER\_SUITES 5

Max number of supported cipher suites

### 6.1.2.64 TLS\_AES\_128\_GCM\_SHA256

#define TLS\_AES\_128\_GCM\_SHA256 0x1301

AES128/SHA256/GCM cipher suite - this is only one which MUST be implemented

## 6.1.2.65 TLS\_AES\_256\_GCM\_SHA384

#define TLS\_AES\_256\_GCM\_SHA384 0x1302

AES256/SHA384/GCM cipher suite

## 6.1.2.66 TLS\_CHACHA20\_POLY1305\_SHA256

#define TLS\_CHACHA20\_POLY1305\_SHA256 0x1303

CHACHA20/SHA256/POLY1305 cipher suite

#### 6.1.2.67 TLS\_AES\_128\_CCM\_SHA256

#define TLS\_AES\_128\_CCM\_SHA256 0x1304

AES/SHA256/CCM cipher suite - optional

# 6.1.2.68 TLS\_AES\_128\_CCM\_8\_SHA256

#define TLS\_AES\_128\_CCM\_8\_SHA256 0x1305

AES/SHA256/CCM 8 cipher suite - optional

# 6.1.2.69 X25519

#define X25519 0x001d

X25519 elliptic curve key exchange

### 6.1.2.70 SECP256R1

#define SECP256R1 0x0017

NIST SECP256R1 elliptic curve key exchange

### 6.1.2.71 SECP384R1

#define SECP384R1 0x0018

NIST SECP384R1 elliptic curve key exchange

#### 6.1.2.72 SECP521R1

#define SECP521R1 0x0019

NIST SECP521R1 elliptic curve key exchange

#### 6.1.2.73 X448

#define X448 0x001e

X448 elliptic curve key exchange

#### 6.1.2.74 KYBER768

#define KYBER768 0x4242

Kyber PQ key exchange - NOTE I just made this up! Not generally recognised!

### 6.1.2.75 HYBRID\_KX

#define HYBRID\_KX 0x421d

Hybrid key exchange, Kyber+X25519

# 6.1.2.76 ECDSA\_SECP256R1\_SHA256

#define ECDSA\_SECP256R1\_SHA256 0x0403

Supported ECDSA Signature algorithm

### 6.1.2.77 ECDSA\_SECP256R1\_SHA384

#define ECDSA\_SECP256R1\_SHA384 0x0413

Non-standard ECDSA Signature algorithm

## 6.1.2.78 ECDSA\_SECP384R1\_SHA384

#define ECDSA\_SECP384R1\_SHA384 0x0503

Supported ECDSA Signature algorithm

# 6.1.2.79 RSA\_PSS\_RSAE\_SHA256

#define RSA\_PSS\_RSAE\_SHA256 0x0804

Supported RSA Signature algorithm

### 6.1.2.80 RSA\_PSS\_RSAE\_SHA384

#define RSA\_PSS\_RSAE\_SHA384 0x0805

Supported RSA Signature algorithm

## 6.1.2.81 RSA\_PSS\_RSAE\_SHA512

#define RSA\_PSS\_RSAE\_SHA512 0x0806

Supported RSA Signature algorithm

## 6.1.2.82 RSA\_PKCS1\_SHA256

#define RSA\_PKCS1\_SHA256 0x0401

Supported RSA Signature algorithm

### 6.1.2.83 RSA\_PKCS1\_SHA384

#define RSA\_PKCS1\_SHA384 0x0501

Supported RSA Signature algorithm

# 6.1.2.84 RSA\_PKCS1\_SHA512

#define RSA\_PKCS1\_SHA512 0x0601

Supported RSA Signature algorithm

### 6.1.2.85 ED25519

#define ED25519 0x0807

Ed25519 EdDSA Signature algorithm

### 6.1.2.86 DILITHIUM2

#define DILITHIUM2 0x0902

Dilithium2 Signature algorithm

## 6.1.2.87 DILITHIUM3

#define DILITHIUM3 0x0903

Dilithium3 Signature algorithm

### 6.1.2.88 DILITHIUM2\_P256

#define DILITHIUM2\_P256 0x09F2

Dilithium2+SECP256R1 Signature algorithms - this type can be negotiated, but always implemented seperately by SAL

### 6.1.2.89 PSKOK

#define PSKOK 0x00

Preshared Key only mode

#### 6.1.2.90 PSKWECDHE

#define PSKWECDHE 0x01

Preshared Key with Diffie-Hellman key exchange mode

# 6.1.2.91 TLS\_FULL\_HANDSHAKE

#define TLS\_FULL\_HANDSHAKE 1

Came from Full Handshake

# 6.1.2.92 TLS\_EXTERNAL\_PSK

#define TLS\_EXTERNAL\_PSK 2

External Pre-Shared Key

## 6.1.2.93 TLS1\_0

#define TLS1\_0 0x0301

TLS 1.0 version

### 6.1.2.94 TLS1\_2

#define TLS1\_2 0x0303

TLS 1.2 version

## 6.1.2.95 TLS1\_3

#define TLS1\_3 0x0304

TLS 1.3 version

## 6.1.2.96 TLS13\_UPDATE\_NOT\_REQUESTED

#define TLS13\_UPDATE\_NOT\_REQUESTED 0

Updating my keys

## 6.1.2.97 TLS13\_UPDATE\_REQUESTED

#define TLS13\_UPDATE\_REQUESTED 1

Updating my keys and telling you to update yours

### 6.1.2.98 SERVER\_NAME

#define SERVER\_NAME 0x0000

Server Name extension

### 6.1.2.99 SUPPORTED\_GROUPS

#define SUPPORTED\_GROUPS 0x000a

Supported Group extension

## 6.1.2.100 SIG\_ALGS

#define SIG\_ALGS 0x000d

Signature algorithms extension

# 6.1.2.101 POST\_HANDSHAKE\_AUTH

#define POST\_HANDSHAKE\_AUTH 0x0031

Post Handshake Authentication

# 6.1.2.102 SIG\_ALGS\_CERT

#define SIG\_ALGS\_CERT 0x0032

Signature algorithms Certificate extension

## 6.1.2.103 KEY\_SHARE

#define KEY\_SHARE 0x0033

Key Share extension

## 6.1.2.104 PSK\_MODE

#define PSK\_MODE 0x002d

Preshared key mode extension

## 6.1.2.105 PRESHARED\_KEY

#define PRESHARED\_KEY 0x0029

Preshared key extension

## 6.1.2.106 TLS\_VER

#define TLS\_VER 0x002b

TLS version extension

## 6.1.2.107 COOKIE

#define COOKIE 0x002c

Cookie extension

### 6.1.2.108 EARLY\_DATA

#define EARLY\_DATA 0x002a

Early Data extension

# 6.1.2.109 MAX\_FRAG\_LENGTH

#define MAX\_FRAG\_LENGTH 0x0001

max fragmentation length extension

### 6.1.2.110 PADDING

#define PADDING 0x0015

Padding extension

# 6.1.2.111 APP\_PROTOCOL

#define APP\_PROTOCOL 0x0010

Application Layer Protocol Negotiation (ALPN)

# 6.1.2.112 RECORD\_SIZE\_LIMIT

#define RECORD\_SIZE\_LIMIT 0x001c

Record Size Limit

## 6.1.2.113 CLIENT\_CERT\_TYPE

#define CLIENT\_CERT\_TYPE 0x0013

Client Certificate type

## 6.1.2.114 SERVER\_CERT\_TYPE

#define SERVER\_CERT\_TYPE 0x0014

Server Certificate type

## 6.1.2.115 HSHAKE

#define HSHAKE 0x16

Handshake record

# **6.1.2.116 APPLICATION**

#define APPLICATION 0x17

Application record

# 6.1.2.117 ALERT

#define ALERT 0x15

Alert record

# 6.1.2.118 CHANGE\_CIPHER

#define CHANGE\_CIPHER 0x14

Change Cipher record

## 6.1.2.119 TIMED\_OUT

#define TIMED\_OUT 0x01

Time-out

## 6.1.2.120 CLIENT\_HELLO

#define CLIENT\_HELLO 0x01

Client Hello message

### 6.1.2.121 SERVER\_HELLO

#define SERVER\_HELLO 0x02

Server Hello message

#### 6.1.2.122 CERTIFICATE

#define CERTIFICATE 0x0b

Certificate message

### 6.1.2.123 CERT\_REQUEST

#define CERT\_REQUEST 0x0d

Certificate Request

## 6.1.2.124 CERT\_VERIFY

#define CERT\_VERIFY 0x0f

Certificate Verify message

# 6.1.2.125 FINISHED

#define FINISHED 0x14

Handshake Finished message

## 6.1.2.126 ENCRYPTED\_EXTENSIONS

#define ENCRYPTED\_EXTENSIONS 0x08

Encrypted Extensions message

#### 6.1.2.127 TICKET

#define TICKET 0x04

Ticket message

## 6.1.2.128 KEY\_UPDATE

#define KEY\_UPDATE 0x18

Key Update message

### 6.1.2.129 MESSAGE\_HASH

#define MESSAGE\_HASH 0xFE

Special synthetic message hash message

# 6.1.2.130 END\_OF\_EARLY\_DATA

#define END\_OF\_EARLY\_DATA 0x05

End of Early Data message

## 6.1.2.131 HANDSHAKE\_RETRY

#define HANDSHAKE\_RETRY 0x102

Handshake retry

# 6.1.2.132 NOT\_TLS1\_3

#define NOT\_TLS1\_3 -2

Wrong version error, not TLS1.3

# 6.1.2.133 BAD\_CERT\_CHAIN

#define BAD\_CERT\_CHAIN -3

Bad Certificate Chain error

### 6.1.2.134 ID\_MISMATCH

#define ID\_MISMATCH -4

Session ID mismatch error

## 6.1.2.135 UNRECOGNIZED\_EXT

#define UNRECOGNIZED\_EXT -5

Unrecognised extension error

## 6.1.2.136 BAD\_HELLO

#define BAD\_HELLO -6

badly formed Hello message error

### 6.1.2.137 WRONG\_MESSAGE

#define WRONG\_MESSAGE -7

Message out-of-order error

# 6.1.2.138 MISSING\_REQUEST\_CONTEXT

#define MISSING\_REQUEST\_CONTEXT -8

Request context missing error

### 6.1.2.139 AUTHENTICATION\_FAILURE

#define AUTHENTICATION\_FAILURE -9

Authentication error - AEAD Tag incorrect

### 6.1.2.140 BAD\_RECORD

#define BAD\_RECORD -10

Badly formed Record received

# 6.1.2.141 BAD\_TICKET

#define BAD\_TICKET -11

Badly formed Ticket received

# 6.1.2.142 NOT\_EXPECTED

#define NOT\_EXPECTED -12

Received ack for something not requested

# 6.1.2.143 CA\_NOT\_FOUND

#define CA\_NOT\_FOUND -13

Certificate Authority not found

## 6.1.2.144 CERT\_OUTOFDATE

#define CERT\_OUTOFDATE -14

Certificate Expired

### 6.1.2.145 MEM\_OVERFLOW

#define MEM\_OVERFLOW -15

Memory Overflow

# 6.1.2.146 FORBIDDEN\_EXTENSION

#define FORBIDDEN\_EXTENSION -16

Forbidden Encrypted Extension

# 6.1.2.147 MAX\_EXCEEDED

#define MAX\_EXCEEDED -17

Maximum record size exceeded

# 6.1.2.148 EMPTY\_CERT\_CHAIN

#define EMPTY\_CERT\_CHAIN -18

**Empty Certificate Message** 

# 6.1.2.149 SELF\_SIGNED\_CERT

#define SELF\_SIGNED\_CERT -20

Self signed certificate

## 6.1.2.150 ERROR\_ALERT\_RECEIVED

#define ERROR\_ALERT\_RECEIVED -22

Alert has been received

## 6.1.2.151 BAD\_MESSAGE

#define BAD\_MESSAGE -23

Badly formed message

## 6.1.2.152 CERT\_VERIFY\_FAIL

#define CERT\_VERIFY\_FAIL -24

Certificate Verification failure

#### 6.1.2.153 BAD\_HANDSHAKE

#define BAD\_HANDSHAKE -26

Could not agree

# 6.1.2.154 BAD\_REQUEST\_UPDATE

#define BAD\_REQUEST\_UPDATE -27

Bad Request Update value

### 6.1.2.155 CLOSURE\_ALERT\_RECEIVED

#define CLOSURE\_ALERT\_RECEIVED -28

Alert has been received

## 6.1.2.156 MISSING\_EXTENSIONS

#define MISSING\_EXTENSIONS -30

Some mandatory extensions are missing

# 6.1.2.157 ILLEGAL\_PARAMETER

#define ILLEGAL\_PARAMETER 0x2F

Illegal parameter alert

## 6.1.2.158 UNEXPECTED\_MESSAGE

#define UNEXPECTED\_MESSAGE 0x0A

Unexpected message alert

## 6.1.2.159 DECRYPT\_ERROR

#define DECRYPT\_ERROR 0x33

Decryption error alert

## 6.1.2.160 BAD\_CERTIFICATE

#define BAD\_CERTIFICATE 0x2A

Bad certificate alert

### 6.1.2.161 UNSUPPORTED\_EXTENSION

#define UNSUPPORTED\_EXTENSION 0x6E

Unsupported extension alert

# 6.1.2.162 UNKNOWN\_CA

#define UNKNOWN\_CA 0x30

**Unrecognised Certificate Authority** 

### 6.1.2.163 CERTIFICATE\_EXPIRED

#define CERTIFICATE\_EXPIRED 0x2D

Certificate Expired

# 6.1.2.164 PROTOCOL\_VERSION

#define PROTOCOL\_VERSION 0x46

Wrong TLS version

# 6.1.2.165 DECODE\_ERROR

#define DECODE\_ERROR 0x32

Decode error alert

## 6.1.2.166 RECORD\_OVERFLOW

#define RECORD\_OVERFLOW 0x16

Record Overflow

## 6.1.2.167 BAD\_RECORD\_MAC

#define BAD\_RECORD\_MAC 0x14

Bad Record Mac

## 6.1.2.168 HANDSHAKE\_FAILURE

#define HANDSHAKE\_FAILURE 0x28

Could not agree

#### 6.1.2.169 CLOSE\_NOTIFY

#define CLOSE\_NOTIFY 0x00

Orderly shut down of connection

# 6.1.2.170 MISSING\_EXTENSION

#define MISSING\_EXTENSION 0x6D;

Missing extension

### 6.1.2.171 LOG\_OUTPUT\_TRUNCATION

#define LOG\_OUTPUT\_TRUNCATION 256

Output Hex digits before truncation

## 6.1.2.172 TLS13\_DISCONNECTED

#define TLS13\_DISCONNECTED 0

TLS1.3 Connection is broken

# 6.1.2.173 TLS13\_CONNECTED

#define TLS13\_CONNECTED 1

TLS1.3 Connection is made

### 6.1.2.174 TLS13\_HANDSHAKING

```
#define TLS13_HANDSHAKING 2
```

TLS1.3 is handshaking

# 6.1.2.175 TLS\_FAILURE

```
#define TLS_FAILURE 0
```

Failed to cmake TLS1.3 connection

## 6.1.2.176 TLS\_SUCCESS

```
#define TLS_SUCCESS 1
```

Succeeded in making TLS1.3 connection

## 6.1.2.177 TLS\_RESUMPTION\_REQUIRED

```
#define TLS_RESUMPTION_REQUIRED 2
```

Connection succeeded, but handshake retry was needed

# 6.1.2.178 TLS\_EARLY\_DATA\_ACCEPTED

```
#define TLS_EARLY_DATA_ACCEPTED 3
```

Connection succeeded, and early data was accepted

### 6.1.2.179 PSK\_NOT

```
#define PSK_NOT 0
```

No PSK

# 6.1.2.180 PSK\_KEY

#define PSK\_KEY 1

Using PSK from database

# 6.1.2.181 PSK\_IBE

#define PSK\_IBE 2

Using IBE based PSK

### 6.1.2.182 X509\_CERT

```
#define X509_CERT 0
```

X509 Certificate-based authentication

# 6.1.2.183 RAW\_PUBLIC\_KEY

```
#define RAW_PUBLIC_KEY 2
```

Raw Public Key based authentication

# 6.1.3 Typedef Documentation

### 6.1.3.1 byte

```
typedef uint8_t byte
```

8-bit unsigned integer

### 6.1.3.2 sign8

```
typedef int8_t sign8
```

8-bit signed integer

### 6.1.3.3 sign16

```
typedef int16_t sign16
```

16-bit signed integer

#### 6.1.3.4 sign32

```
typedef int32_t sign32
```

32-bit signed integer

### 6.1.3.5 sign64

```
typedef int64_t sign64
```

64-bit signed integer

#### 6.1.3.6 unsign32

```
typedef uint32_t unsign32
```

32-bit unsigned integer

#### 6.1.3.7 unsign64

```
typedef uint64_t unsign64
```

64-bit unsigned integer

# 6.2 tls1 3.h

#### Go to the documentation of this file.

```
00008 #ifndef TLS1_3_H
00009 #define TLS1_3_H
00010
00011 #include <stdint.h>
00012 #include "tls_octads.h"
00013 #include "tls_sockets.h"
00015 typedef uint8_t byte;
00016 typedef int8_t sign8 ;
00017 typedef int16_t sign16;
00018 typedef int32 t sign32;
00019 typedef int64_t sign64;
00020 typedef uint32_t unsign32;
00021 typedef uint64_t unsign64;
00023 // Terminal Output
00024 #define IO_NONE 0
00025 #define IO_APPLICATION 1
00026 #define IO_PROTOCOL 2
00027 #define IO_DEBUG 3
00028 #define IO_WIRE 4
00030 // Cryptographic Environment
00031 #define TINY_ECC 0
00032 #define TYPICAL 1
00033 #define POST_QUANTUM 2
00034 #define HYBRID 3
00036 // Client Certificate Chain + Key
00037 #define NOCERT 0
00038 #define RSA_SS 1
00039 #define ECC_SS 2
00040 #define DLT_SS
00041 #define HYB_SS 6
00042 #define HW_1 4
00043 #define HW_2 5
00046
00047 // Note that favourite group (as used in client hello) is determined by the SAL ordering - see
     tls sal.cpp
00048 // If server does not support it, an expensive Handshake Retry will be required
00049 // So best to place a popular group (such as X25519) at top of list in SAL
00050
00051 #define VERBOSITY IO PROTOCOL
00052 #define THIS_YEAR 2023
00054 #define POST_HS_AUTH
00055 #define CLIENT_CERT ECC_SS
00057 #define CRYPTO_SETTING TYPICAL
00058 // Supported protocols
00059 #define TLS_APPLICATION_PROTOCOL (char \star)("http/1.1")
00060 #define ALLOW_SELF_SIGNED
00061 //#define NO_CERT_CHECKS
                                      /**< Don't do any checks on server certs - useful for Anvil testing
00062 #define TRY_EARLY_DATA
00064 // Note that BUFF, Certificates and crypto keys can be quite large, and therefore maybe better taken
      from the heap
\tt 00065 // on systems with a shallow stack. Define this to use the heap.
00066
00067 //#define SHALLOW_STACK
                                       /**< Get large arrays from heap, else stack */
00068
```

6.2 tls1\_3.h 59

```
00069 // comment out if no max record size. In practise TLS1.3 doesn't seem to support this
     record_size_limit extension, so use with caution
00070 // #define MAX_RECORD 1024
                                    /\star\star< Maximum record size client is willing to receive - should be less
     than TLS_MAX_IBUFF_SIZE below */
00071 // Note that if this is not used, max_fragment_size extension is tried instead, see TLS_MAX_FRAG below
00072
00073 // define this so that all encrypted records are padded with 0s to full length
00074 // #define PAD_SHORT_RECORDS
                                         /**< Pad short output records */
00075
00076 //#define PREFER_RAW_SERVER_PUBLIC_KEY
                                              // Would be happy with raw public key from server
00077 //#define PREFER_RAW_CLIENT_PUBLIC_KEY
                                              // Would prefer server to accept raw public key from client
00078
08000
00081
00082 // Standard Hash Types
00083
00084 #define TLS SHA256 T
00085 #define TLS_SHA384_T 2
00086 #define TLS SHA512 T 3
00088 // Some maximum sizes for stack allocated memory. Handshake will fail if these sizes are exceeded!
00089
00090 #define TLS_MAX_HASH_STATE 768
00091 #define TLS_MAX_HASH 64
00092 #define TLS_MAX_KEY 32
00093 #define TLS_X509_MAX_FIELD 256
00094 #define TLS_MAX_EXT_LABEL 256
00096 // Max Frag length must be less than TLS_MAX_IBUFF_SIZE
00097 #define TLS_MAX_FRAG 2
00099 #if CRYPTO_SETTING==TYPICAL
00100 #define TLS_MAX_IBUFF_SIZE (16384+256)
00101 #define TLS_MAX_PLAIN_FRAG 16384
00102 #define TLS_MAX_CIPHER_FRAG (16384+256)
00104
       #define TLS_MAX_CERT_SIZE 2048
00105
      #define TLS_MAX_CERT_B64 2800
00106 #define TLS_MAX_HELLO 1024
00108 #define TLS_MAX_SIG_PUB_KEY_SIZE 512
       #define TLS_MAX_SIG_SECRET_KEY_SIZE 512
00110
       #define TLS_MAX_SIGNATURE_SIZE 512
       #define TLS_MAX_KEX_PUB_KEY_SIZE 97
00111
00112
      #define TLS_MAX_KEX_CIPHERTEXT_SIZE 97
00113 #define TLS_MAX_KEX_SECRET_KEY_SIZE 48
00114 #endif
00115
00116 #if CRYPTO SETTING == POST QUANTUM
00117
00118 #define TLS_MAX_IBUFF_SIZE (16384+256)
00119  #define TLS_MAX_PLAIN_FRAG 16384
00120  #define TLS_MAX_CIPHER_FRAG (16384+256)
00122 #define TLS_MAX_CERT_SIZE 6144
00123 #define TLS_MAX_CERT_B64 8192
00124 #define TLS_MAX_HELLO 2048
00126 // These all blow up post quantum
00127 #define TLS_MAX_SIG_PUB_KEY_SIZE 1952
00128 #define TLS_MAX_SIG_SECRET_KEY_SIZE 4000
       #define TLS_MAX_SIGNATURE_SIZE 3296
00129
00130 #define TLS_MAX_KEX_PUB_KEY_SIZE 1184
       #define TLS_MAX_KEX_CIPHERTEXT_SIZE 1088
00131
00132 #define TLS_MAX_KEX_SECRET_KEY_SIZE 2400
00133 #endif
00134
00135 #if CRYPTO SETTING == HYBRID
00136
00137 #define TLS_MAX_IBUFF_SIZE (16384+256)
00138 #define TLS_MAX_PLAIN_FRAG 16384
00139 #define TLS_MAX_CIPHER_FRAG (16384+256)
00141 #define TLS_MAX_CERT_SIZE 6144
00142 #define TLS_MAX_CERT_B64 8192
00143 #define TLS_MAX_HELLO 2048
00145 // These all blow up post quantum
00146 #define TLS_MAX_SIG_PUB_KEY_SIZE 1312+65
00147 #define TLS_MAX_SIG_SECRET_KEY_SIZE 2528+200
00148 #define TLS_MAX_SIGNATURE_SIZE 2420+100
00149 #define TLS_MAX_KEX_PUB_KEY_SIZE 1184+32
00150 #define TLS_MAX_KEX_CIPHERTEXT_SIZE 1088+32
00151 #define TLS_MAX_KEX_SECRET_KEY_SIZE 2400+32
00152 #endif
00153
00154
00155 #if CRYPTO SETTING==TINY ECC
00156 #define TLS_MAX_IBUFF_SIZE (4096+256)
       #define TLS_MAX_PLAIN_FRAG 4096
       #define TLS_MAX_CIPHER_FRAG (4096+256)
00158
00160
       #define TLS_MAX_CERT_SIZE 2048
00161 #define TLS_MAX_CERT_B64 2800
00162 #define TLS_MAX_HELLO 1024
00164 #define TLS_MAX_SIG_PUB_KEY_SIZE 133
```

```
00165 #define TLS_MAX_SIG_SECRET_KEY_SIZE 66
00166 #define TLS_MAX_SIGNATURE_SIZE 132
00167 #define TLS_MAX_KEX_PUB_KEY_SIZE 97
00168 #define TLS_MAX_KEX_CIPHERTEXT_SIZE 97
00169 #define TLS_MAX_KEX_SECRET_KEY_SIZE 48
00170 #endif
00171
00172 // Certificate size limits
00173 #define TLS_MAX_SERVER_CHAIN_LEN 2
00174 #define TLS_MAX_SERVER_CHAIN_SIZE (TLS_MAX_SERVER_CHAIN_LEN*TLS_MAX_CERT_SIZE)
00175 #define TLS_MAX_CLIENT_CHAIN_LEN 1
00176 #define TLS_MAX_CLIENT_CHAIN_SIZE (TLS_MAX_CLIENT_CHAIN_LEN*TLS_MAX_CERT_SIZE)
00178 #define TLS_MAX_SHARED_SECRET_SIZE 256
00180 // Both of these are bumped up by PQ IBE and Hybrid
00181 #define TLS_MAX_TICKET_SIZE 4196
00182 #define TLS_MAX_EXTENSIONS 6144
00184 #define TLS_MAX_ECC_FIELD 66
00185 #define TLS_MAX_IV_SIZE 12
00186 #define TLS_MAX_TAG_SIZE 16
00187 #define TLS_MAX_COOKIE 128
00189 #define TLS_MAX_OUTPUT_RECORD_SIZE 1024
00190 #define TLS_MAX_OBUFF_SIZE (TLS_MAX_OUTPUT_RECORD_SIZE+TLS_MAX_TAG_SIZE+6)
00192 #define TLS_MAX_SERVER_NAME 128
00193 #define TLS_MAX_SUPPORTED_GROUPS 10
00194 #define TLS_MAX_SUPPORTED_SIGS 16
00195 #define TLS_MAX_PSK_MODES
00196 #define TLS_MAX_CIPHER_SUITES 5
00198 // Cipher Suites
00199 #define TLS_AES_128_GCM_SHA256 0x1301
00200 #define TLS_AES_256_GCM_SHA384 0x1302
00201 #define TLS_CHACHA20_POLY1305_SHA256 0x1303
00202 #define TLS_AES_128_CCM_SHA256 0x1304
00203 #define TLS_AES_128_CCM_8_SHA256 0x1305
00205 // Key exchange groups
00206 #define X25519 0x001d
00207 #define SECP256R1 0x0017
00208 #define SECP384R1 0x0018
00209 #define SECP521R1 0x0019
00210 #define X448 0x001e
00211 #define KYBER768 0x4242
00212 #define HYBRID KX 0x421d
00214 // Signature algorithms for TLS1.3 and Certs that we can handle
00215 #define ECDSA_SECP256R1_SHA256 0x0403 00216 #define ECDSA_SECP256R1_SHA384 0x0413
00217 #define ECDSA_SECP384R1_SHA384 0x0503
00218 #define RSA_PSS_RSAE_SHA256 0x0804
00219 #define RSA_PSS_RSAE_SHA384 0x0805
00220 #define RSA_PSS_RSAE_SHA512 0x0806
00221 #define RSA_PKCS1_SHA256 0x0401
00222 #define RSA_PKCS1_SHA384 0x0501
00223 #define RSA_PKCS1_SHA512 0x0601
00224 #define ED25519 0x0807
00225 #define DILITHIUM2 0x0902
00226 #define DILITHIUM3 0x0903
00227 #define DILITHIUM2_P256 0x09F2
00229 // pre-shared Key (PSK) modes
00230 #define PSKOK 0x00
00231 #define PSKWECDHE 0x01
00233 // ticket origin
00234 #define TLS_FULL_HANDSHAKE 1 00235 #define TLS_EXTERNAL_PSK 2
00237 // TLS versions
00238 #define TLS1_0 0x0301
00239 #define TLS1_2 0x0303
00240 #define TLS1_3 0x0304
00242 #define TLS13_UPDATE_NOT_REQUESTED 0
00243 #define TLS13_UPDATE_REQUESTED 1
00245 // Extensions
00246 #define SERVER_NAME 0x0000
00247 #define SUPPORTED_GROUPS 0x000a
00248 #define SIG_ALGS 0x000d
00249 #define POST_HANDSHAKE_AUTH 0x0031
00250 #define SIG_ALGS_CERT 0x0032
00251 #define KEY_SHARE 0x0033
00252 #define PSK_MODE 0x002d
00253 #define PRESHARED_KEY 0x0029
00254 #define TLS_VER 0x002b
00255 #define COOKIE 0x002c
00256 #define EARLY_DATA 0x002a
00257 #define MAX_FRAG_LENGTH 0x0001
00258 #define PADDING 0x0015
00259 #define APP_PROTOCOL 0x0010
00260 #define RECORD_SIZE_LIMIT 0x001c
00261 #define CLIENT_CERT_TYPE 0x0013
00262 #define SERVER_CERT_TYPE 0x0014
00265 // record types
00266 #define HSHAKE 0x16
```

6.2 tls1\_3.h

```
00267 #define APPLICATION 0x17
00268 #define ALERT 0x15
00269 #define CHANGE_CIPHER 0x14
00270 // pseudo record types
00271 #define TIMED OUT 0x01
00273 // message types
00274 #define CLIENT_HELLO 0x01
00275 #define SERVER_HELLO 0x02
00276 #define CERTIFICATE 0x0b
00277 #define CERT_REQUEST 0x0d
00278 #define CERT_VERIFY 0x0f
00279 #define FINISHED 0x14
00280 #define ENCRYPTED_EXTENSIONS 0x08
00281 #define TICKET 0x04
00282 #define KEY_UPDATE 0x18
00283 #define MESSAGE_HASH 0xFE
00284 #define END_OF_EARLY_DATA 0x05
00285 // pseudo message types
00286 #define HANDSHAKE_RETRY 0x102
00288 // Causes of server error - which should generate a client alert
00289 #define NOT_TLS1_3 -2
00290 #define BAD_CERT_CHAIN -3
00291 #define ID_MISMATCH -4
00292 #define UNRECOGNIZED EXT -5
00293 #define BAD_HELLO -6
00294 #define WRONG_MESSAGE -7
00295 #define MISSING_REQUEST_CONTEXT -8
00296 #define AUTHENTICATION_FAILURE -9
00297 #define BAD_RECORD -10
00298 #define BAD_TICKET -11
00299 #define NOT_EXPECTED -12
00300 #define CA_NOT_FOUND -13
00301 #define CERT_OUTOFDATE -14
00302 #define MEM_OVERFLOW -15
00303 #define FORBIDDEN_EXTENSION -16
00304 #define MAX_EXCEEDED -17
00305 #define EMPTY_CERT_CHAIN -18
00306 #define SELF_SIGNED_CERT -20
00307 #define ERROR_ALERT_RECEIVED -22
00308 #define BAD_MESSAGE -23
00309 #define CERT_VERIFY_FAIL -24
00310 #define BAD_HANDSHAKE -26
00311 #define BAD REQUEST UPDATE -27
00312 #define CLOSURE_ALERT_RECEIVED -28
00313 #define MISSING_EXTENSIONS -30
00314 // client alerts
00315 #define ILLEGAL_PARAMETER 0x2F
00316 #define UNEXPECTED_MESSAGE 0x0A
00317 #define DECRYPT_ERROR 0x33
00318 #define BAD_CERTIFICATE 0x2A
00319 #define UNSUPPORTED_EXTENSION 0x6E
00320 #define UNKNOWN_CA 0x30
00321 #define CERTIFICATE_EXPIRED 0x2D
00322 #define PROTOCOL VERSION 0x46
00323 #define DECODE_ERROR 0x32
00324 #define RECORD_OVERFLOW 0x16
00325 #define BAD_RECORD_MAC 0x14
00326 #define HANDSHAKE_FAILURE 0x28
00327 #define CLOSE_NOTIFY 0x00
00328 #define MISSING_EXTENSION 0x6D;
00330 #define LOG_OUTPUT_TRUNCATION 256
00332 #define TLS13_DISCONNECTED 0
00333 #define TLS13_CONNECTED 1
00334 #define TLS13_HANDSHAKING 2
00336 // protocol returns..
00337 #define TLS_FAILURE 0
00338 #define TLS_SUCCESS 1
00339 #define TLS_RESUMPTION_REQUIRED 2
00340 #define TLS_EARLY_DATA_ACCEPTED 3
00342 // PSK modes
00343 #define PSK_NOT 0
00344 #define PSK_KEY 1
00345 #define PSK_IBE 2
00347 // Certificate types
00348 #define X509_CERT 0
00349 #define RAW_PUBLIC_KEY 2
00353 typedef struct
00354 {
00355
          unsign32 val;
00356
          int err;
00357 } ret;
00358
00361 typedef struct
00362 {
00363
          bool early_data;
00364
          bool alpn;
00365
          bool server name:
```

```
bool max_frag_length;
00367 } ee_status;
00368
00371 typedef struct
00372 {
00373
         bool active;
00374
         char k[TLS_MAX_KEY];
00375
         char iv[12];
00376
         octad K;
00377
         octad IV;
00378
         unsign32 record;
00379
        int suite;
int taglen;
00380
00381 } crypto;
00382
00385 typedef struct
00386 {
00387
         bool valid;
         char tick[TLS_MAX_TICKET_SIZE];
00388
00389
         char nonce[256];
00390
         char psk[TLS_MAX_HASH];
00391
         octad TICK;
         octad NONCE;
00392
         octad PSK;
00393
         unsign32 age_obfuscator;
unsign32 max_early_data;
00394
00395
00396
         unsign32 birth;
00397
         int lifetime;
00398
         int cipher_suite;
         int favourite_group;
00399
00400
         int origin;
00401 } ticket;
00402
00405 typedef struct
00406 {
         char state[TLS_MAX_HASH_STATE];
00407
00408
         int htype;
00409 } unihash;
00410
00413 typedef struct
00414 {
00415
         int status;
         int max_record;
Socket *sockptr;
00416
00417
00418
         char id[32];
00419
         char hostname[TLS_MAX_SERVER_NAME];
00420
         int cipher_suite;
00421
         int favourite_group;
00422
         int server_cert_type;
00423
         int client_cert_type;
00424
         crypto K_send;
00425
         crypto K_recv;
00426
         octad HS;
         char hs[TLS_MAX_HASH];
00427
00428
         octad RMS;
00429
         char rms[TLS_MAX_HASH];
00430
         octad STS;
00431
         char sts[TLS_MAX_HASH];
00432
         octad CTS;
00433
         char cts[TLS_MAX_HASH];
00434
         octad CTX;
         char ctx[TLS_MAX_HASH];
00435
00436
         octad IBUFF;
00437
         octad OBUFF;
00438 #ifndef SHALLOW_STACK
00439 char ibuff[TLS_MAX_IBUFF_SIZE];
         char obuff[TLS_MAX_OBUFF_SIZE];
00440
00441 #endif
00442
       int ptr;
         unihash tlshash;
00444
         ticket T;
00445 } TLS_session;
00446
00447 // IBUFF buffer
00450 //
00451 // when ptr becomes equal to IBUFF.len, pull in another record (and maybe decrypt it)
00452
00453 #endif
```

# 6.3 tls bfibe.h File Reference

Boneh and Franklin IBE.

```
#include "pair_BLS12381.h"
```

#### **Macros**

- #define PGS\_BLS12381 MODBYTES\_B384\_58
- #define PFS\_BLS12381 MODBYTES\_B384\_58

#### **Functions**

```
• bool BFIBE_CCA_ENCRYPT (char *ID, octet *R32, octet *SSK, octet *CT)
```

Create key SSK encapsulated in ciphertext CT to be sent to ID.

bool BFIBE\_CCA\_DECRYPT (octet \*SK, octet \*CT, octet \*SSK)

Create key SSK encapsulated in ciphertext CT to be sent to ID.

### 6.3.1 Detailed Description

Boneh and Franklin IBE.

**Author** 

Mike Scott

#### 6.3.2 Macro Definition Documentation

#### 6.3.2.1 PGS BLS12381

```
#define PGS_BLS12381 MODBYTES_B384_58
BF Group Size
```

# 6.3.2.2 PFS\_BLS12381

```
#define PFS_BLS12381 MODBYTES_B384_58
BF Field Size
```

#### 6.3.3 Function Documentation

### 6.3.3.1 BFIBE\_CCA\_ENCRYPT()

Create key SSK encapsulated in ciphertext CT to be sent to ID.

#### **Parameters**

| ID  | the entity to receive encapsulated key |
|-----|--|
| R32 | 32 random bytes                        |
| SSK | is the encapsulated key                |
| CT  | is the ciphertext                      |

#### Returns

true if OK

### 6.3.3.2 BFIBE\_CCA\_DECRYPT()

Create key SSK encapsulated in ciphertext CT to be sent to ID.

#### **Parameters**

| SK  | the secret key of ID    |
|-----|-------------------------|
| CT  | is the ciphertext       |
| SSK | is the decapsulated key |

#### Returns

true if OK

# 6.4 tls bfibe.h

# Go to the documentation of this file. 00001

```
00007 //
00008 // Pairing-based B&F IBE 128-bit API Functions
00009 // Uses MIRACL
00010 //
00011 #ifndef BFIBE_BLS381_H
00012 #define BFIBE_BLS381_H
00013
00014 #include "pair_BLS12381.h"
00015
00016 using namespace core;
00017
00018 /* Field size is assumed to be greater than or equal to group size */
00019
00020 #if CHUNK == 32
00021 #define PGS_BLS12381 MODBYTES_B384_29
00022 #define PFS_BLS12381 MODBYTES_B384_29
00023 #else
00024 #define PGS_BLS12381 MODBYTES_B384_58
00025 #define PFS_BLS12381 MODBYTES_B384_58
00026 #endif
```

```
00027

00028 /* IBE primitives */

00029

00038 bool BFIBE_CCA_ENCRYPT(char *ID,octet *R32,octet *SSK,octet *CT);

00039

00047 bool BFIBE_CCA_DECRYPT(octet *SK,octet *CT,octet *SSK);

00048

00049 #endif

00050
```

# 6.5 tls\_cert\_chain.h File Reference

Process Certificate Chain.

```
#include "tls1_3.h"
#include "tls_x509.h"
#include "tls_sal.h"
#include "tls_client_recv.h"
#include "tls_logger.h"
#include "tls_certs.h"
```

#### **Functions**

- int checkServerCertChain (octad \*CERTCHAIN, char \*hostname, int cert\_type, octad \*PUBKEY, octad \*SIG)

  Check Certificate Chain for hostname, and extract public key.
- int getClientPrivateKeyandCertChain (octad \*PRIVKEY, int cert\_type, octad \*CERTCHAIN)

  Get Client private key and Certificate chain from .pem files.

# 6.5.1 Detailed Description

Process Certificate Chain.

**Author** 

Mike Scott

### 6.5.2 Function Documentation

#### 6.5.2.1 checkServerCertChain()

Check Certificate Chain for hostname, and extract public key.

#### **Parameters**

| CERTCHAIN | the input certificate chain                                  |
|-----------|--|
| hostname  | the input Server name associated with the Certificate chain  |
| cert_type | the certificate type (a certificate or a raw key)            |
| PUBKEY    | the Server's public key extracted from the Certificate chain |
| SIG       | signature (supplied as workspace)                            |

#### Returns

0 if certificate chain is OK, else returns negative failure reason

#### 6.5.2.2 getClientPrivateKeyandCertChain()

Get Client private key and Certificate chain from .pem files.

#### **Parameters**

| PRIVKEY   | the Client's private key                          |
|-----------|---|
| cert_type | the certificate type (a certificate or a raw key) |
| CERTCHAIN | the Client's certificate chain                    |

#### Returns

type of private key, ECC or RSA

# 6.6 tls cert chain.h

#### Go to the documentation of this file.

```
00001
00008 // TLS1.3 Server Certificate Chain Code
00009
00010 #ifndef TLS_CERT_CHAIN_H
00011 #define TLS_CERT_CHAIN_H
00012 #include "tls1_3.h"
00013 #include "tls_x509.h"
00014 #include "tls_sal.h"
00015 #include "tls_client_recv.h"
00016 #include "tls_client_recv.h"
00016 #include "tls_logger.h"
00017 #include "tls_certs.h"
00018
00019 using namespace std;
00020
00030 extern int checkServerCertChain(octad *CERTCHAIN, char *hostname, int cert_type, octad *PUBKEY, octad *SIG);
00031
00039 extern int getClientPrivateKeyandCertChain(octad *PRIVKEY, int cert_type, octad *CERTCHAIN);
00040
00041 #endif
```

# 6.7 tls\_certs.h File Reference

Certificate Authority root certificate store.

```
#include "tls1_3.h"
```

### **Functions**

int getSigRequirements (int \*sigReqs)
 Get Client Certificate chain requirements.

#### **Variables**

```
const char * myprivate = NULLconst char * mycert
```

• const char \* cacerts

# 6.7.1 Detailed Description

Certificate Authority root certificate store.

**Author** 

Mike Scott

### 6.7.2 Function Documentation

# 6.7.2.1 getSigRequirements()

Get Client Certificate chain requirements.

#### **Parameters**

| sigReqs | list of signature requirements |
|---------|--------------------------------|

### Returns

number of such requirements

#### 6.7.3 Variable Documentation

#### 6.7.3.1 myprivate

```
const char * myprivate =NULL [extern]
Client private key
```

#### 6.7.3.2 mycert

```
const char * mycert [extern]
```

```
Initial value:
= (char *)
"----BEGIN CERTIFICATE----\n"
"MIIBKzCB0aADAqECAqEBMAoGCCqGSM49BAMCMB0xGzAZBqNVBAMTEjAxMjM0NjI0QjIwMjYwRDdF\n"
"RTAeFw0yMTExMTgxMTAwMDBaFw0yNjExMTgxMTAwMDBaMB0xGzAZBgNVBAMTEjAxMjM0NjI0QjIw\n"
"MjYwRDdFRTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABDOFj/SnArwqM15cZs/bXppfTuAxgMzB\n"
"N3LS48xHSqpLhHlVnv0vWqyhE8v+ZX4Jz1o7Z9LGOG537EeldBeGjYijAjAAMAoGCCqGSM49BAMC\n"
"A0kAMEYCIQC901185YX1+9vZ0t/SHQ3zFH5e7Vc8XtrZ+mTtMc5riwIhAI/SektrG3C0JwII0VV5\n"
"psR9RRnuwo810km81P4S56/m\n"
"----END CERTIFICATE----\n"
```

Client certificate

#### 6.7.3.3 cacerts

```
const char* cacerts [extern]
```

The Root Certificate store

# 6.8 tls certs.h

### Go to the documentation of this file.

```
00008 #ifndef TLS_CA_CERTS_H
00009 #define TLS_CA_CERTS_H
00010
00011 #include "tls1_3.h"
00012
00013 extern const char *myprivate;
00014 extern const char *mycert;
00015 extern const char *cacerts;
00022 extern int getSigRequirements(int *sigReqs);
00023
00024 #endif
```

#### 6.9 tls client recv.h File Reference

Process Input received from the Server.

```
#include "tls_sal.h"
#include "tls1_3.h"
#include "tls_sockets.h"
#include "tls_keys_calc.h"
#include "tls_client_send.h"
```

#### **Functions**

ret parseoctad (octad \*E, int len, octad \*M, int &ptr)

Parse out an octad from a pointer into an octad.

ret parsebytes (char \*e, int len, octad \*M, int &ptr)

Parse out byte array from a pointer into an octad.

• ret parseInt (octad \*M, int len, int &ptr)

Parse out an unsigned integer from a pointer into an octad.

ret parseoctadptr (octad \*E, int len, octad \*M, int &ptr)

Return a pointer to an octad from a pointer into an octad.

int getServerRecord (TLS session \*session)

Read a record from the Server, a fragment of a full protocol message.

ret parseIntorPull (TLS\_session \*session, int len)

Parse out an unsigned integer from a pointer into an octad, if necessary pulling in a new fragment.

ret parseoctadorPull (TLS\_session \*session, octad \*O, int len)

Parse out an octad from a pointer into an octad, if necessary pulling in a new fragment.

• ret parsebytesorPull (TLS\_session \*session, char \*o, int len)

Parse out a byte array from a pointer into an octad, if necessary pulling in a new fragment.

ret parseoctadorPullptrX (TLS session \*session, octad \*O, int len)

Return a pointer to an octad from a pointer into an octad, if necessary pulling in a new fragment.

bool badResponse (TLS\_session \*session, ret r)

Process response from server input.

ret seeWhatsNext (TLS session \*session)

Identify type of incoming message.

ret getServerEncryptedExtensions (TLS\_session \*session, ee\_status \*enc\_ext\_expt, ee\_status \*enc\_ext←
 \_resp)

Receive and parse Server Encrypted Extensions.

ret getServerCertVerify (TLS\_session \*session, octad \*SCVSIG, int &sigalg)

Get Server proof that he owns the Certificate, by receiving and verifying its signature on transcript hash.

• ret getServerFinished (TLS\_session \*session, octad \*HFIN)

Get final handshake message from Server, a HMAC on the transcript hash.

ret getServerHello (TLS\_session \*session, int &kex, octad \*CK, octad \*PK, int &pskid)

Receive and parse initial Server Hello.

• ret getCheckServerCertificateChain (TLS\_session \*session, octad \*PUBKEY, octad \*SIG)

Receive and check certificate chain.

ret getCertificateRequest (TLS\_session \*session, bool context)

process a Certificate Request

#### 6.9.1 Detailed Description

Process Input received from the Server.

**Author** 

Mike Scott

#### 6.9.2 Function Documentation

### 6.9.2.1 parseoctad()

Parse out an octad from a pointer into an octad.

#### **Parameters**

| Ε   | the output octad copied out from the octad M |
|-----|--|
| len | the expected length of the output octad E    |
| М   | the input octad                              |
| ptr | a pointer into M, which advances after use   |

#### Returns

the actual length of E extracted, and an error flag

### 6.9.2.2 parsebytes()

Parse out byte array from a pointer into an octad.

#### **Parameters**

| е   | the output byte array copied out from the octad M |
|-----|---|
| len | the expected length of e                          |
| М   | the input octad                                   |
| ptr | a pointer into M, which advances after use        |

### Returns

the actual length of e extracted, and an error flag

### 6.9.2.3 parseInt()

```
int len,
int & ptr )
```

Parse out an unsigned integer from a pointer into an octad.

#### **Parameters**

| М   | the input octad                            |
|-----|--|
| len | the number of bytes in integer             |
| ptr | a pointer into M, which advances after use |

#### Returns

the integer value, and an error flag

#### 6.9.2.4 parseoctadptr()

Return a pointer to an octad from a pointer into an octad.

#### **Parameters**

| Ε   | a pointer to an octad contained within an octad M |  |
|-----|---|--|
| len | the expected length of the octad E                |  |
| М   | the input octad                                   |  |
| ptr | a pointer into M, which advances after use        |  |

#### Returns

the actual length of E, and an error flag

# 6.9.2.5 getServerRecord()

Read a record from the Server, a fragment of a full protocol message.

#### **Parameters**

| session the TLS session str | ucture |
|-----------------------------|--------|
|-----------------------------|--------|

### Returns

a positive indication of the record type, or a negative error return

### 6.9.2.6 parseIntorPull()

Parse out an unsigned integer from a pointer into an octad, if necessary pulling in a new fragment.

#### **Parameters**

| session | the TLS session structure      |
|---------|--------------------------------|
| len     | the number of bytes in integer |

#### Returns

the unsigned integer, and an error flag

### 6.9.2.7 parseoctadorPull()

Parse out an octad from a pointer into an octad, if necessary pulling in a new fragment.

#### **Parameters**

| session | the TLS session structure                 |
|---------|---|
| 0       | the output octad                          |
| len     | the expected length of the output octad O |

#### Returns

the actual length of O extracted, and an error flag

#### 6.9.2.8 parsebytesorPull()

```
ret parsebytesorPull (
          TLS_session * session,
          char * o,
          int len )
```

Parse out a byte array from a pointer into an octad, if necessary pulling in a new fragment.

#### **Parameters**

| session | the TLS session structure         |
|---------|-----------------------------------|
| 0       | the output bytes                  |
| len     | the expected length of the output |

#### Returns

the actual length of o extracted, and an error flag

#### 6.9.2.9 parseoctadorPullptrX()

Return a pointer to an octad from a pointer into an octad, if necessary pulling in a new fragment.

#### **Parameters**

| session | the TLS session structure                          |
|---------|--|
| 0       | a pointer to an octad contained within an octad IO |
| len     | the expected length of the octad O                 |

#### Returns

the actual length of O extracted, and an error flag

#### 6.9.2.10 badResponse()

Process response from server input.

#### **Parameters**

| session | the TLS1.3 session structure |
|---------|------------------------------|
| r       | return value to be processed |

#### Returns

true, if its a bad response requiring an abort

### 6.9.2.11 seeWhatsNext()

Identify type of incoming message.

#### **Parameters**

| session | the TLS session structure |
|---------|---------------------------|
|---------|---------------------------|

#### Returns

negative error, zero for OK, or positive for message type

### 6.9.2.12 getServerEncryptedExtensions()

Receive and parse Server Encrypted Extensions.

#### **Parameters**

| session      | the TLS session structure                    |  |
|--------------|--|--|
| enc_ext_expt | ext structure containing server expectations |  |
| enc_ext_resp | ext structure containing server responses    |  |

#### Returns

response structure

#### 6.9.2.13 getServerCertVerify()

Get Server proof that he owns the Certificate, by receiving and verifying its signature on transcript hash.

#### **Parameters**

| session | the TLS session structure                     |
|---------|---|
| SCVSIG  | the received signature on the transcript hash |
| sigalg  | the type of the received signature            |

#### Returns

response structure

### 6.9.2.14 getServerFinished()

Get final handshake message from Server, a HMAC on the transcript hash.

#### Parameters

| session | the TLS session structure                                      |
|---------|--|
| HFIN    | an octad containing HMAC on transcript as calculated by Server |

#### Returns

response structure

#### 6.9.2.15 getServerHello()

Receive and parse initial Server Hello.

#### **Parameters**

| session | the TLS session structure                                |
|---------|--|
| kex     | key exchange data  |
| CK      | an output Cookie   |
| PK      | the key exchange public value supplied by the Server     |
| pskid   | indicates if a pre-shared key was accepted, otherwise -1 |

#### Returns

response structure

### 6.9.2.16 getCheckServerCertificateChain()

Receive and check certificate chain.

#### **Parameters**

| session | the TLS session structure                            |
|---------|--|
| PUBKEY  | the public key extracted from the Server certificate |
| SIG     | signature (supplied as workspace)                    |

#### Returns

response structure

# 6.9.2.17 getCertificateRequest()

process a Certificate Request

#### **Parameters**

| session | the TLS session structure   |
|---------|-----------------------------|
| context | true if expecting a context |

6.10 tls\_client\_recv.h 77

Returns

response structure

# 6.10 tls client recv.h

#### Go to the documentation of this file.

```
00001
00007 // Process input received from Server
80000
00009 #ifndef TLS_CLIENT_RECV_H
00010 #define TLS_CLIENT_RECV_H
00011
00012 #include "tls_sal.h"
00012 "Include "tls1_3.h"
00014 #include "tls_sockets.h"
00015 #include "tls_keys_calc.h"
00016 #include "tls_client_send.h"
00017
00026 extern ret parseoctad(octad \starE, int len, octad \starM, int &ptr);
00027
00036 extern ret parsebytes(char *e,int len,octad *M,int &ptr);
00037
00045 extern ret parseInt(octad *M, int len, int &ptr);
00046
00055 extern ret parseoctadptr(octad *E,int len,octad *M,int &ptr);
00056
00062 extern int getServerRecord(TLS_session *session);
00063
00070 extern ret parseIntorPull(TLS_session *session,int len);
00071
00079 extern ret parseoctadorPull(TLS_session *session,octad *0,int len);
08000
00081
00089 extern ret parsebytesorPull(TLS_session *session, char *o, int len);
00098 extern ret parseoctadorPullptrX(TLS_session *session,octad *0,int len);
00099
00106 extern bool badResponse(TLS_session *session,ret r);
00107
00113 extern ret seeWhatsNext(TLS_session *session);
00114
00122 extern ret getServerEncryptedExtensions(TLS_session *session,ee_status *enc_ext_expt,ee_status
      *enc_ext_resp);
00123
00131 extern ret getServerCertVerify(TLS_session *session,octad *SCVSIG,int &sigalg);
00132
00139 extern ret getServerFinished(TLS_session *session,octad *HFIN);
00150 extern ret getServerHello(TLS_session *session,/*int &cipher,*/int &kex,octad *CK,octad *PK,int
00151
00159 extern ret getCheckServerCertificateChain(TLS_session *session,octad *PUBKEY,octad *SIG);
00160
00167 extern ret getCertificateRequest(TLS_session *session,bool context);
00168
00169
00170
00171 #endif
```

# 6.11 tls\_client\_send.h File Reference

Process Output to be sent to the Server.

```
#include "tls_sal.h"
#include "tls1_3.h"
#include "tls_sockets.h"
#include "tls_keys_calc.h"
```

#### **Functions**

void sendCCCS (TLS\_session \*session)

Send Change Cipher Suite message.

int addPreSharedKeyExt (octad \*EXT, unsign32 age, octad \*IDS, int sha)

Add PreShared Key extension to under-construction Extensions Octet (omitting binder)

void addServerNameExt (octad \*EXT, char \*servername)

Add Server name extension to under-construction Extensions Octet.

void addSupportedGroupsExt (octad \*EXT, int nsg, int \*supportedGroups)

Add Supported Groups extension to under-construction Extensions Octet.

void addServerRawPublicKey (octad \*EXT)

indicate acceptance of raw server public key

void addClientRawPublicKey (octad \*EXT)

indicate acceptance of raw client public key

void addSigAlgsExt (octad \*EXT, int nsa, int \*sigAlgs)

Add Supported TLS1.3 Signature algorithms to under-construction Extensions Octet.

void addSigAlgsCertExt (octad \*EXT, int nsac, int \*sigAlgsCert)

Add Supported X.509 Certificate Signature algorithms to under-construction Extensions Octet.

void addKeyShareExt (octad \*EXT, int alg, octad \*PK)

Add Key Share extension to under-construction Extensions Octet.

void addALPNExt (octad \*EXT, octad \*AP)

Add Application Layer Protocol Negotiation (ALPN) extension to under-construction Extensions Octet.

void addMFLExt (octad \*EXT, int mode)

Add Maximum Fragment Length extension to under-construction Extensions Octet.

void addRSLExt (octad \*EXT, int size)

Add Record Size Limit extension to under-construction Extensions Octet.

void addPSKModesExt (octad \*EXT, int mode)

Add Preshared Key exchange modes extension to under-construction Extensions Octet.

void addVersionExt (octad \*EXT, int version)

Add Version extension to under-construction Extensions Octet.

void addPadding (octad \*EXT, int n)

Add padding extension to under-construction Extensions Octet.

void addCookieExt (octad \*EXT, octad \*CK)

Add Cookie extension to under-construction Extensions Octet.

void addEarlyDataExt (octad \*EXT)

Indicate desire to send Early Data in under-construction Extensions Octet.

void addPostHSAuth (octad \*EXT)

indicate willingness to do post handshake authentication

int clientRandom (octad \*RN)

Generate 32-byte random octad.

• int cipherSuites (octad \*CS, int ncs, int \*ciphers)

Build a cipher-suites octad from supported ciphers.

 $\bullet \ \ void\ sendClientMessage\ (TLS\_session\ *session,\ int\ rectype,\ int\ version,\ octad\ *CM,\ octad\ *EXT,\ bool\ flush)$ 

void sendBinder (TLS session \*session, octad \*BND)

Send a preshared key binder message to the Server.

Send a generic client message (as a single record) to the Server.

void sendClientHello (TLS\_session \*session, int version, octad \*CH, octad \*CRN, bool already\_agreed, octad \*EXTENSIONS, int extra, bool resume, bool flush)

Prepare and send Client Hello message to the Server, appending prepared extensions.

• void sendAlert (TLS session \*session, int type)

Prepare and send an Alert message to the Server.

• void sendKeyUpdate (TLS\_session \*session, int type)

Prepare and send a key update message to the Server.

void sendClientFinish (TLS\_session \*session, octad \*CHF)

Prepare and send a final handshake Verification message to the Server.

void sendClientCertificateChain (TLS\_session \*session, octad \*CERTCHAIN)

Prepare and send client certificate message to the Server.

void sendClientCertVerify (TLS\_session \*session, int sigAlg, octad \*CCVSIG)

Send client Certificate Verify message to the Server.

void sendEndOfEarlyData (TLS\_session \*session)

Indicate End of Early Data in message to the Server.

• int alert\_from\_cause (int rtn)

Maps problem cause to Alert.

### 6.11.1 Detailed Description

Process Output to be sent to the Server.

**Author** 

Mike Scott

#### 6.11.2 Function Documentation

#### 6.11.2.1 sendCCCS()

Send Change Cipher Suite message.

**Parameters** 

```
session the TLS session structure
```

#### 6.11.2.2 addPreSharedKeyExt()

Add PreShared Key extension to under-construction Extensions Octet (omitting binder)

#### **Parameters**

| EXT | the extensions octad which is being built            |
|-----|--|
| age | the obfuscated age of the preshared key              |
| IDS | the proposed preshared key identity                  |
| sha | the hash algorithm used to calculate the HMAC binder |

#### Returns

length of binder to be sent later

# 6.11.2.3 addServerNameExt()

Add Server name extension to under-construction Extensions Octet.

#### **Parameters**

| EXT        | the extensions octad which is being built |
|------------|---|
| servername | the Host name (URL) of the Server         |

### 6.11.2.4 addSupportedGroupsExt()

Add Supported Groups extension to under-construction Extensions Octet.

#### **Parameters**

| EXT             | the extensions octad which is being built |  |
|-----------------|---|--|
| nsg             | Number of supported groups                |  |
| supportedGroups | an array of supported groups              |  |

# 6.11.2.5 addServerRawPublicKey()

```
\verb"void addServerRawPublicKey" (
```

```
octad * EXT )
```

indicate acceptance of raw server public key

#### **Parameters**

| EXT   the extensions octad which is be | eing built |
|--|------------|
|--|------------|

### 6.11.2.6 addClientRawPublicKey()

indicate acceptance of raw client public key

#### **Parameters**

#### 6.11.2.7 addSigAlgsExt()

Add Supported TLS1.3 Signature algorithms to under-construction Extensions Octet.

#### **Parameters**

| EXT     | the extensions octad which is being built  |
|---------|--|
| nsa     | Number of supported signature algorithms   |
| sigAlgs | an array of supported signature algorithms |

#### 6.11.2.8 addSigAlgsCertExt()

Add Supported X.509 Certificate Signature algorithms to under-construction Extensions Octet.

#### **Parameters**

| EXT         | the extensions octad which is being built  |
|-------------|--|
| nsac        | Number of supported signature algorithms   |
| sigAlgsCert | an array of supported signature algorithms |

### 6.11.2.9 addKeyShareExt()

Add Key Share extension to under-construction Extensions Octet.

#### **Parameters**

| EXT | the extensions octad which is being built              |
|-----|--|
| alg | the suggested key exchange algorithm                   |
| PK  | the key exchange public value to be sent to the Server |

#### 6.11.2.10 addALPNExt()

Add Application Layer Protocol Negotiation (ALPN) extension to under-construction Extensions Octet.

#### **Parameters**

| EXT | the extensions octad which is being built               |
|-----|---|
| AP  | the IANA sequence associated with the expected protocol |

### 6.11.2.11 addMFLExt()

Add Maximum Fragment Length extension to under-construction Extensions Octet.

#### **Parameters**

| EXT  | the extensions octad which is being built |
|------|---|
| mode | the proposed maximum fragment size        |

# 6.11.2.12 addRSLExt()

Add Record Size Limit extension to under-construction Extensions Octet.

#### **Parameters**

| EXT  | the extensions octad which is being built |
|------|---|
| size | the demanded maximum fragment size        |

#### 6.11.2.13 addPSKModesExt()

Add Preshared Key exchange modes extension to under-construction Extensions Octet.

#### **Parameters**

| EXT  | the extensions octad which is being built |
|------|---|
| mode | the proposed preshared key mode           |

# 6.11.2.14 addVersionExt()

Add Version extension to under-construction Extensions Octet.

### **Parameters**

| EXT     | the extensions octad which is being built |
|---------|---|
| version | the supported TLS version                 |

### 6.11.2.15 addPadding()

Add padding extension to under-construction Extensions Octet.

#### **Parameters**

| EXT | the extensions octad which is being built |
|-----|---|
| n   | the zero padding length                   |

### 6.11.2.16 addCookieExt()

Add Cookie extension to under-construction Extensions Octet.

#### **Parameters**

| EXT | the extensions octad which is being built |
|-----|---|
| CK  | the cookie octad to be added              |

### 6.11.2.17 addEarlyDataExt()

Indicate desire to send Early Data in under-construction Extensions Octet.

#### **Parameters**

| EXT | the extensions octad which is being built |
|-----|---|

#### 6.11.2.18 addPostHSAuth()

```
{\tt void} \ {\tt addPostHSAuth} \ (
```

```
octad * EXT )
```

indicate willingness to do post handshake authentication

#### **Parameters**

| EXT | the extensions octad which is being built |
|-----|---|
|-----|---|

### 6.11.2.19 clientRandom()

Generate 32-byte random octad.

#### **Parameters**

```
RN the output 32-byte octad
```

#### Returns

length of output octad

### 6.11.2.20 cipherSuites()

Build a cipher-suites octad from supported ciphers.

#### **Parameters**

| CS      | the output cipher-suite octad         |  |
|---------|---------------------------------------|--|
| ncs     | the number of supported cipher-suites |  |
| ciphers | an array of supported cipher-suites   |  |

### Returns

length of the output octad

### 6.11.2.21 sendClientMessage()

Send a generic client message (as a single record) to the Server.

#### **Parameters**

| session | the TLS session structure                          |  |
|---------|--|--|
| rectype | the record type                                    |  |
| version | TLS version indication                             |  |
| СМ      | the client message to be sent                      |  |
| EXT     | extensions to be added (or NULL if there are none) |  |
| flush   | transmit immediately if true                       |  |

### 6.11.2.22 sendBinder()

Send a preshared key binder message to the Server.

# **Parameters**

| session | the TLS session structure                 |
|---------|---|
| BND     | binding HMAC of truncated transcript hash |

#### 6.11.2.23 sendClientHello()

```
void sendClientHello (
          TLS_session * session,
          int version,
          octad * CH,
          octad * CRN,
          bool already_agreed,
          octad * EXTENSIONS,
          int extra,
          bool resume,
          bool flush )
```

Prepare and send Client Hello message to the Server, appending prepared extensions.

#### **Parameters**

| session        | the TLS session structure                              |
|----------------|--|
| version        | TLS version indication                                 |
| CH             | workspace octad in which to build client Hello         |
| CRN            | Random bytes   |
| already_agreed | true if cipher suite previously negotiated, else false |
| EXTENSIONS     | pre-prepared extensions                                |
| extra          | length of preshared key binder to be sent later        |
| resume         | true if this hello is for handshae resumption          |
| flush          | transmit immediately                                   |

#### 6.11.2.24 sendAlert()

Prepare and send an Alert message to the Server.

#### **Parameters**

| session | the TLS session structure |
|---------|---------------------------|
| type    | the type of the Alert     |

# 6.11.2.25 sendKeyUpdate()

Prepare and send a key update message to the Server.

### Parameters

| session | the TLS session structure |
|---------|---------------------------|
| type    | the type of the update    |

# 6.11.2.26 sendClientFinish()

```
\verb"void sendClientFinish" (
```

```
TLS_session * session,
octad * CHF )
```

Prepare and send a final handshake Verification message to the Server.

#### **Parameters**

| session | the TLS session structure   |
|---------|-----------------------------|
| CHF     | the client verify data HMAC |

# 6.11.2.27 sendClientCertificateChain()

Prepare and send client certificate message to the Server.

#### **Parameters**

| session   | the TLS session structure    |
|-----------|------------------------------|
| CERTCHAIN | the client certificate chain |

# 6.11.2.28 sendClientCertVerify()

Send client Certificate Verify message to the Server.

### **Parameters**

| session | the TLS session structure                |
|---------|--|
| sigAlg  | the client's digital signature algorithm |
| CCVSIG  | the client's signature                   |

# 6.11.2.29 sendEndOfEarlyData()

```
void sendEndOfEarlyData ( {\tt TLS\_session} \ * \ session \ )
```

Indicate End of Early Data in message to the Server.

#### **Parameters**

```
session the TLS session structure
```

#### 6.11.2.30 alert from cause()

Maps problem cause to Alert.

#### **Parameters**

rtn the cause of a problem (a function error return)

#### Returns

type of Alert that should be sent to Server

# 6.12 tls\_client\_send.h

#### Go to the documentation of this file.

```
00001
00008 // Process output sent to Server
00009 #ifndef TLS_CLIENT_SEND_H
00010 #define TLS_CLIENT_SEND_H
00011
00012 #include "tls_sal.h"
00012 #Include "tls1_3.h"
00014 #include "tls_sockets.h"
00015 #include "tls_keys_calc.h"
00016
00021 extern void sendCCCS(TLS_session *session);
00022
00031 extern int addPreSharedKeyExt(octad *EXT,unsign32 age,octad *IDS,int sha);
00032
00038 extern void addServerNameExt(octad *EXT,char *servername);
00039
00046 extern void addSupportedGroupsExt(octad *EXT,int nsg,int *supportedGroups);
00047
00052 extern void addServerRawPublicKey(octad *EXT); 00057 extern void addClientRawPublicKey(octad *EXT);
00058
00066 extern void addSigAlgsExt(octad *EXT,int nsa,int *sigAlgs);
00067
00074 extern void addSigAlgsCertExt(octad *EXT, int nsac, int *sigAlgsCert);
00075
00076
00083 extern void addKeyShareExt(octad *EXT, int alg, octad *PK);
00084
00085
00091 extern void addALPNExt(octad *EXT,octad *AP);
00092
00093
00099 extern void addMFLExt(octad *EXT,int mode);
00100
00106 extern void addRSLExt(octad *EXT,int size);
00107
00113 extern void addPSKModesExt(octad *EXT,int mode);
00114
00120 extern void addVersionExt(octad *EXT, int version);
00121
```

```
00127 extern void addPadding(octad *EXT,int n);
00134 extern void addCookieExt(octad *EXT,octad *CK);
00135
00140 extern void addEarlyDataExt(octad *EXT);
00141
00146 extern void addPostHSAuth(octad *EXT);
00147
00153 extern int clientRandom(octad *RN);
00154
00162 extern int cipherSuites(octad *CS, int ncs, int *ciphers);
00163
00173 extern void sendClientMessage(TLS_session *session,int rectype,int version,octad *CM,octad *EXT,bool
00174
00180 extern void sendBinder(TLS_session *session,octad *BND);
00181
00194 extern void sendClientHello(TLS session *session,int version,octad *CH,octad *CRN,bool
      already_agreed,octad *EXTENSIONS,int extra,bool resume,bool flush);
00195
00201 extern void sendAlert(TLS_session *session, int type);
00202
00203
00209 extern void sendKeyUpdate(TLS_session *session, int type);
00210
00211
00217 extern void sendClientFinish(TLS_session *session,octad *CHF);
00218
00224 extern void sendClientCertificateChain(TLS_session *session,octad *CERTCHAIN);
00225
00232 extern void sendClientCertVerify(TLS session *session, int sigAlg, octad *CCVSIG);
00233
00234
00239 extern void sendEndOfEarlyData(TLS_session *session);
00240
00246 extern int alert_from_cause(int rtn);
00247 #endif
```

# 6.13 tls keys calc.h File Reference

TLS 1.3 crypto support functions.

```
#include "tls1_3.h"
#include "tls_sal.h"
#include "tls_client_recv.h"
```

#### **Functions**

• void initTranscriptHash (TLS\_session \*session)

Initialise Transcript hash.

void runningHash (TLS\_session \*session, octad \*O)

Accumulate octad into ongoing hashing.

void runningHashIO (TLS\_session \*session)

Accumulate transcript hash from IO buffer.

• void rewindIO (TLS\_session \*session)

rewind the IO buffer

void runningHashlOrewind (TLS session \*session)

Accumulate transcript hash and from IO buffer, and rewind IO buffer.

void transcriptHash (TLS\_session \*session, octad \*O)

Output current hash value.

void runningSyntheticHash (TLS session \*session, octad \*O, octad \*E)

Calculate special synthetic hash calculation for first clientHello after retry request (RFC 8446 section 4.4.1)

void initCryptoContext (crypto \*C)

Initiate a Crypto Context.

void updateCryptoContext (crypto \*C, octad \*K, octad \*IV)

Build a Crypto Context.

void incrementCryptoContext (crypto \*C)

Increment a Crypto Context for the next record, updating IV.

void createCryptoContext (int cipher, octad \*TS, crypto \*context)

Create a crypto context from an input raw Secret and an agreed cipher\_suite.

void createSendCryptoContext (TLS\_session \*session, octad \*TS)

Build a crypto context for transmission from an input raw Secret and an agreed cipher\_suite.

void createRecvCryptoContext (TLS session \*session, octad \*TS)

Build a crypto context for reception from an input raw Secret and an agreed cipher\_suite.

• void recoverPSK (TLS\_session \*session)

Recover pre-shared key from the Resumption Master Secret and store with ticket.

void deriveEarlySecrets (int htype, octad \*PSK, octad \*ES, octad \*BKE, octad \*BKR)

Extract Early Secret Key and Binder Key from Preshared Key (External or Resumption)

void deriveLaterSecrets (int htype, octad \*H, octad \*ES, octad \*CETS, octad \*EEMS)

Extract more secrets from Early Secret.

void deriveHandshakeSecrets (TLS session \*session, octad \*SS, octad \*ES, octad \*H)

Extract Handshake Secret from Shared Secret and Early Secret. Use Handshake Secret to extract Client and Server Handshake Traffic secrets.

• void deriveApplicationSecrets (TLS\_session \*session, octad \*SFH, octad \*CFH, octad \*EMS)

Extract Application Secret from Handshake Secret and Early Secret. Use Handshake Secret to extract Client and Server Application Traffic secrets.

void deriveUpdatedKeys (crypto \*context, octad \*TS)

Perform a Key Update on a crypto context.

bool checkVeriferData (int htype, octad \*SF, octad \*STS, octad \*H)

Test if data from Server is verified using server traffic secret and a transcript hash.

void deriveVeriferData (int htype, octad \*SF, octad \*CTS, octad \*H)

Create handshake verification data for Client to send to Server from client traffic secret and a transcript hash.

bool checkServerCertVerifier (int sigalg, octad \*SCVSIG, octad \*H, octad \*CERTPK)

verify Server's signature on protocol transcript

void createClientCertVerifier (int sigAlg, octad \*H, octad \*KEY, octad \*CCVSIG)

Create Cert Verify message, as a digital signature on some TLS1.3 specific message+transcript hash.

#### 6.13.1 Detailed Description

TLS 1.3 crypto support functions.

Author

Mike Scott

#### 6.13.2 Function Documentation

#### 6.13.2.1 initTranscriptHash()

Initialise Transcript hash.

# **Parameters**

| session  | the TLS session structure |
|----------|---------------------------|
| 00001011 | the red decolor structure |

# 6.13.2.2 runningHash()

Accumulate octad into ongoing hashing.

### **Parameters**

| session | the TLS session structure       |
|---------|---------------------------------|
| 0       | an octad to be included in hash |

# 6.13.2.3 runningHashIO()

Accumulate transcript hash from IO buffer.

### **Parameters**

| session | the TLS session structure |
|---------|---------------------------|
|---------|---------------------------|

# 6.13.2.4 rewindIO()

rewind the IO buffer

| session | the TLS session structure |
|---------|---------------------------|

# 6.13.2.5 runningHashlOrewind()

```
void runningHashIOrewind ( {\tt TLS\_session} \ * \ session \ )
```

Accumulate transcript hash and from IO buffer, and rewind IO buffer.

### **Parameters**

| session | the TLS session structure |
|---------|---------------------------|
|---------|---------------------------|

### 6.13.2.6 transcriptHash()

```
void transcriptHash (
          TLS_session * session,
          octad * 0 )
```

Output current hash value.

### **Parameters**

| session | the TLS session structure               |
|---------|---|
| 0       | an output octad containing current hash |

# 6.13.2.7 runningSyntheticHash()

```
void runningSyntheticHash (
          TLS_session * session,
          octad * O,
          octad * E )
```

Calculate special synthetic hash calculation for first clientHello after retry request (RFC 8446 section 4.4.1)

#### **Parameters**

| session | the TLS session structure                  |
|---------|--|
| 0       | an octad containing clientHello            |
| Ε       | an octad containing clientHello extensions |

# 6.13.2.8 initCryptoContext()

Initiate a Crypto Context.

#### **Parameters**

```
C an AEAD encryption context
```

### 6.13.2.9 updateCryptoContext()

Build a Crypto Context.

### **Parameters**

| С  | an AEAD encryption context          |
|----|-------------------------------------|
| K  | an encryption key                   |
| IV | an encryption Initialisation Vector |

### 6.13.2.10 incrementCryptoContext()

Increment a Crypto Context for the next record, updating IV.

### **Parameters**

```
C an AEAD encryption context
```

# 6.13.2.11 createCryptoContext()

```
void createCryptoContext (
          int cipher,
          octad * TS,
          crypto * context )
```

Create a crypto context from an input raw Secret and an agreed cipher\_suite.

#### **Parameters**

| cipher  | the chosen cipher site     |
|---------|----------------------------|
| TS      | the input raw secret       |
| context | the output crypto conetext |

# 6.13.2.12 createSendCryptoContext()

Build a crypto context for transmission from an input raw Secret and an agreed cipher\_suite.

#### **Parameters**

| session | TLS session structure |
|---------|-----------------------|
| TS      | the input raw secret  |

### 6.13.2.13 createRecvCryptoContext()

Build a crypto context for reception from an input raw Secret and an agreed cipher\_suite.

# **Parameters**

| session | TLS session structure |
|---------|-----------------------|
| TS      | the input raw secret  |

# 6.13.2.14 recoverPSK()

Recover pre-shared key from the Resumption Master Secret and store with ticket.

| session | the TLS session structure |
|---------|---------------------------|

# 6.13.2.15 deriveEarlySecrets()

```
void deriveEarlySecrets (
    int htype,
    octad * PSK,
    octad * ES,
    octad * BKE,
    octad * BKR )
```

Extract Early Secret Key and Binder Key from Preshared Key (External or Resumption)

### **Parameters**

| htype | hash algorithm   |  |
|-------|--|--|
| PSK   | the input pre-shared key, or NULL if not available           |  |
| ES    | the output early secret key                                  |  |
| BKE   | BKE the output external binder key (or NULL if not required) |  |
| BKR   | the output resumption binder key (or NULL if not required)   |  |

### 6.13.2.16 deriveLaterSecrets()

```
void deriveLaterSecrets (
    int htype,
    octad * H,
    octad * ES,
    octad * CETS,
    octad * EEMS )
```

Extract more secrets from Early Secret.

### **Parameters**

| htype | hash algorithm  |  |
|-------|---|--|
| Н     | a partial transcript hash   |  |
| ES    | the input early secret key  |  |
| CETS  | CETS the output Client Early Traffic Secret (or NULL if not required) |  |
| EEMS  | the output Early Exporter Master Secret (or NULL if not required)     |  |

# 6.13.2.17 deriveHandshakeSecrets()

```
void deriveHandshakeSecrets ( {\tt TLS\_session} \ * \ session,
```

```
octad * SS,
octad * ES,
octad * H )
```

Extract Handshake Secret from Shared Secret and Early Secret. Use Handshake Secret to extract Client and Server Handshake Traffic secrets.

#### **Parameters**

| session | the TLS session structure  |
|---------|----------------------------|
| SS      | input Shared Secret        |
| ES      | the input early secret key |
| Н       | a partial transcript hash  |

# 6.13.2.18 deriveApplicationSecrets()

Extract Application Secret from Handshake Secret and Early Secret. Use Handshake Secret to extract Client and Server Application Traffic secrets.

### **Parameters**

| session | the TLS session structure                                   |  |
|---------|---|--|
|         |   |  |
| SFH     | an input partial transcript hash                            |  |
| CFH     | FH an input partial transcript hash                         |  |
| EMS     | the output External Master Secret (or NULL if not required) |  |

# 6.13.2.19 deriveUpdatedKeys()

Perform a Key Update on a crypto context.

| context | an AEAD encryption context |
|---------|----------------------------|
| TS      | the updated Traffic secret |

#### 6.13.2.20 checkVeriferData()

```
bool checkVeriferData (
    int htype,
    octad * SF,
    octad * STS,
    octad * H )
```

Test if data from Server is verified using server traffic secret and a transcript hash.

### **Parameters**

| htype | hash algorithm                          |
|-------|---|
| SF    | the input verification data from Server |
| STS   | the input Server Traffic Secret         |
| Н     | the input partial transcript hash       |

### Returns

true is data is verified, else false

# 6.13.2.21 deriveVeriferData()

```
void deriveVeriferData (
    int htype,
    octad * SF,
    octad * CTS,
    octad * H )
```

Create handshake verification data for Client to send to Server from client traffic secret and a transcript hash.

# **Parameters**

| htype | hash algorithm                    |
|-------|-----------------------------------|
| SF    | the output verification data      |
| CTS   | the input Client Traffic Secret   |
| Н     | the input partial transcript hash |

### 6.13.2.22 checkServerCertVerifier()

```
bool checkServerCertVerifier ( int \ sigalg,
```

```
octad * SCVSIG,
octad * H,
octad * CERTPK )
```

verify Server's signature on protocol transcript

#### **Parameters**

| sigalg | the algorithm used for digital signature |
|--------|--|
| SCVSIG | the input signature on the transcript    |
| Н      | the transcript hash                      |
| CERTPK | the Server's public key                  |

#### Returns

true if signature is verified, else returns false

# 6.13.2.23 createClientCertVerifier()

```
void createClientCertVerifier (
    int sigAlg,
    octad * H,
    octad * KEY,
    octad * CCVSIG )
```

Create Cert Verify message, as a digital signature on some TLS1.3 specific message+transcript hash.

#### **Parameters**

| sigAlg | the signature algorithm        |
|--------|--------------------------------|
| Н      | a transcript hash to be signed |
| KEY    | the Client's private key       |
| CCVSIG | the output digital signature   |

# 6.14 tls\_keys\_calc.h

# Go to the documentation of this file.

```
00001
00008 // TLS1.3 crypto support functions
00009 #ifndef TLS_KEYS_CALC_H
00010 #define TLS_KEYS_CALC_H
00011
00012 #include "tls1_3.h"
00013 #include "tls_sal.h"
00014 #include "tls_client_recv.h"
00015
00016 // transcript hash support
00021 extern void initTranscriptHash(TLS_session *session);
00022
00029 extern void runningHash(TLS_session *session,octad *0);
00030
```

```
00036 extern void runningHashIO(TLS_session *session);
00043 extern void rewindIO(TLS_session *session);
00044
00045
00051 extern void runningHashIOrewind(TLS_session *session);
00053
00059 extern void transcriptHash(TLS_session *session,octad *0);
00060
00068 extern void runningSyntheticHash(TLS_session *session,octad *O,octad *E);
00069
00074 extern void initCryptoContext(crypto *C);
00075
00082 extern void updateCryptoContext(crypto *C,octad *K,octad *IV);
00083
00088 extern void incrementCryptoContext(crypto *C);
00089
00090
00097 extern void createCryptoContext(int cipher,octad *TS,crypto *context);
00098
00104 extern void createSendCryptoContext(TLS_session *session,octad *TS);
00105
00111 extern void createRecvCryptoContext(TLS_session *session,octad *TS);
00112
00117 extern void recoverPSK(TLS_session *session);
00118
00127 extern void deriveEarlySecrets(int htype,octad *PSK,octad *ES,octad *BKE,octad *BKR);
00128
00137 extern void deriveLaterSecrets(int htype,octad *H,octad *ES,octad *CETS,octad *EEMS);
00138
00146 extern void deriveHandshakeSecrets(TLS_session *session,octad *SS,octad *ES, octad *H);
00147
00155 extern void deriveApplicationSecrets(TLS_session *session,octad *SFH,octad *CFH,octad *EMS);
00156
00162 extern void deriveUpdatedKeys(crypto *context,octad *TS);
00163
00172 extern bool checkVeriferData(int htype,octad *SF,octad *STS,octad *H);
00173
00181 extern void deriveVeriferData(int htype,octad *SF,octad *CTS,octad *H);
00182
00191 extern bool checkServerCertVerifier(int sigalg,octad *SCVSIG,octad *H,octad *CERTPK);
00192
00200 extern void createClientCertVerifier(int sigAlg,octad *H,octad *KEY,octad *CCVSIG);
00202 #endif
```

# 6.15 tls\_logger.h File Reference

```
TLS 1.3 logging.

#include <string.h>
#include "tls1_3.h"
#include "tls x509.h"
```

### **Functions**

```
    void myprintf (char *s)
```

internal printf function - all output funnels through this function

void log (int logit, char \*preamble, char \*string, unsign32 info, octad \*O)

basic logging function

• void logServerHello (int cipher suite, int pskid, octad \*PK, octad \*CK)

logging the Server hello

void logTicket (ticket \*T)

logging a resumption ticket

void logEncExt (ee\_status \*e, ee\_status \*r)

logging server extended extensions responses vs expectations

```
    void logCert (octad *CERT)
```

logging a Certificate in standard base 64 format

• void logCertDetails (octad \*PUBKEY, pktype pk, octad \*SIG, pktype sg, octad \*ISSUER, octad \*SUBJECT) logging Certificate details

• void logServerResponse (ret r)

log client processing of a Server response

void logAlert (int detail)

log Server Alert

• void logCipherSuite (int cipher\_suite)

log Cipher Suite

void logKeyExchange (int kex)

log Key Exchange Group

• void logSigAlg (int sigAlg)

log Signature Algorithm

# 6.15.1 Detailed Description

TLS 1.3 logging.

Author

Mike Scott

# 6.15.2 Function Documentation

# 6.15.2.1 myprintf()

```
void myprintf ( {\rm char} \ * \ s \ )
```

internal printf function - all output funnels through this function

**Parameters** 

s a string to be output

# 6.15.2.2 log()

```
void log (
                int logit,
                char * preamble,
                char * string,
```

```
unsign32 info,
octad * 0 )
```

# basic logging function

### **Parameters**

| logit    | logging level   |
|----------|---|
| preamble | a string to be output                                   |
| string   | another string, or a format specifier for info, or NULL |
| info     | an integer to be output                                 |
| 0        | an octad to be output (or NULL)                         |

# 6.15.2.3 logServerHello()

```
void logServerHello (
                int cipher_suite,
                int pskid,
                octad * PK,
                     octad * CK )
```

# logging the Server hello

### **Parameters**

| cipher_suite | the chosen cipher suite                  |
|--------------|--|
| pskid        | the chosen preshared key (or -1 if none) |
| PK           | the Server Public Key                    |
| CK           | a Cookie (if any)                        |

# 6.15.2.4 logTicket()

# logging a resumption ticket

```
T a resumption ticket
```

# 6.15.2.5 logEncExt()

logging server extended extensions responses vs expectations

### **Parameters**

| е | structure containing server expectations |
|---|--|
| r | structure containing server responses    |

# 6.15.2.6 logCert()

logging a Certificate in standard base 64 format

#### **Parameters**

```
CERT | the certificate to be logged
```

# 6.15.2.7 logCertDetails()

logging Certificate details

| PUBKEY  | the certificate public key octad    |
|---------|-------------------------------------|
| pk      | the public key type                 |
| SIG     | the signature on the certificate    |
| sg      | the signature type                  |
| ISSUER  | the (composite) certificate issuer  |
| SUBJECT | the (composite) certificate subject |

# 6.15.2.8 logServerResponse()

```
void logServerResponse (  {\tt ret} \ r \ )
```

log client processing of a Server response

**Parameters** 

r the Server response

# 6.15.2.9 logAlert()

```
void logAlert (
          int detail )
```

log Server Alert

**Parameters** 

detail the server's alert code

# 6.15.2.10 logCipherSuite()

log Cipher Suite

**Parameters** 

cipher\_suite the Cipher Suite to be logged

# 6.15.2.11 logKeyExchange()

```
void logKeyExchange (
          int kex )
```

log Key Exchange Group

#### **Parameters**

kex the Key Exchange Group to be logged

### 6.15.2.12 logSigAlg()

#### log Signature Algorithm

#### **Parameters**

sigAlg the Signature Algorithm to be logged

# 6.16 tls\_logger.h

#### Go to the documentation of this file.

```
00007 // Log protocol progress
00008 #ifndef TLS_LOGGER_H
00009 #define TLS_LOGGER_H
00010
00011 #include <string.h>
00012 #include "tls1_3.h"
00013 #include "tls_x509.h"
00014
00019 extern void myprintf(char *s);
00020
00029 extern void log(int logit,char *preamble,char *string,unsign32 info,octad *0);
00038 extern void logServerHello(int cipher_suite,int pskid,octad *PK,octad *CK);
00039
00044 extern void logTicket(ticket *T);
00045
00051 extern void logEncExt(ee_status *e,ee_status *r);
00052
00057 extern void logCert(octad *CERT);
00058
00068 extern void logCertDetails(octad *PUBKEY,pktype pk,octad *SIG,pktype sg,octad *ISSUER,octad *SUBJECT);
00069
00074 extern void logServerResponse(ret r);
00080 extern void logAlert(int detail);
00081
00086 extern void logCipherSuite(int cipher_suite);
00087
00092 extern void logKeyExchange(int kex);
00098 extern void logSigAlg(int sigAlg);
00099
00100 #endif
```

# 6.17 tls\_octads.h File Reference

octad handling routines - octads don't overflow, they truncate

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

#### **Data Structures**

· struct octad

Safe representation of an octad.

### **Functions**

• unsigned long millis ()

read milliseconds from a stop-watch

• void OCT\_append\_int (octad \*O, unsigned int x, int len)

Join len bytes of integer x to end of octad O (big endian)

void OCT\_append\_octad (octad \*O, octad \*P)

Join one octad to the end of another.

bool OCT\_compare (octad \*O, octad \*P)

Compare two octads.

void OCT\_shift\_left (octad \*O, int n)

Shifts octad left by n bytes.

void OCT\_kill (octad \*O)

Wipe clean an octad.

void OCT\_from\_hex (octad \*O, char \*src)

Convert a hex number to an octad.

void OCT\_append\_string (octad \*O, char \*s)

Join from a C string to end of an octad.

void OCT\_append\_byte (octad \*O, int b, int n)

Join single byte to end of an octad, repeated n times.

void OCT append bytes (octad \*O, char \*s, int n)

Join bytes to end of an octad.

void OCT\_from\_base64 (octad \*O, char \*b)

Create an octad from a base64 number.

void OCT\_reverse (octad \*O)

Reverse bytes in an octad.

void OCT\_truncate (octad \*O, int n)

Reverse bytes in an octad.

void OCT\_copy (octad \*O, octad \*P)

Copy one octad into another.

bool OCT\_output\_hex (octad \*O, int max, char \*s)

Output octad as hex string.

bool OCT\_output\_string (octad \*O, int max, char \*s)

Output octad as C ascii string.

void OCT\_output\_base64 (octad \*O, int max, char \*s)

Output octad as base64 string.

### 6.17.1 Detailed Description

octad handling routines - octads don't overflow, they truncate

Author

Mike Scott

# 6.17.2 Function Documentation

# 6.17.2.1 millis()

```
unsigned long millis ( )
```

read milliseconds from a stop-watch

#### Returns

milliseconds read from stop-watch

# 6.17.2.2 OCT\_append\_int()

Join len bytes of integer x to end of octad O (big endian)

### Parameters

| 0   | octad to be appended to     |
|-----|-----------------------------|
| X   | integer to be appended to O |
| len | number of bytes in m        |

# 6.17.2.3 OCT\_append\_octad()

Join one octad to the end of another.

| 0 | octad to be appended to            |
|---|------------------------------------|
| Р | octad to be joined to the end of O |

# 6.17.2.4 OCT\_compare()

Compare two octads.

# **Parameters**

| 0 | first octad to be compared  |
|---|-----------------------------|
| Р | second octad to be compared |

#### Returns

true if equal, else false

# 6.17.2.5 OCT\_shift\_left()

Shifts octad left by n bytes.

Leftmost bytes disappear

#### **Parameters**

| 0 | octad to be shifted      |
|---|--------------------------|
| n | number of bytes to shift |

# 6.17.2.6 OCT\_kill()

Wipe clean an octad.

### **Parameters**

O octad to be cleared

# 6.17.2.7 OCT\_from\_hex()

Convert a hex number to an octad.

#### **Parameters**

| 0   | octad                      |
|-----|----------------------------|
| src | Hex string to be converted |

# 6.17.2.8 OCT\_append\_string()

Join from a C string to end of an octad.

### **Parameters**

| 0 | octad to be written to                       |
|---|--|
| s | zero terminated string to be joined to octad |

# 6.17.2.9 OCT\_append\_byte()

Join single byte to end of an octad, repeated n times.

| 0 | octad to be written to            |
|---|-----------------------------------|
| b | byte to be joined to end of octad |
| n | number of times b is to be joined |

# 6.17.2.10 OCT\_append\_bytes()

Join bytes to end of an octad.

#### **Parameters**

| 0 | octad to be written to                  |
|---|---|
| s | byte array to be joined to end of octad |
| n | number of bytes to join                 |

# 6.17.2.11 OCT\_from\_base64()

Create an octad from a base64 number.

### **Parameters**

| 0 | octad to be populated         |
|---|-------------------------------|
| b | zero terminated base64 string |

# 6.17.2.12 OCT\_reverse()

Reverse bytes in an octad.

### **Parameters**

O octad to be reversed

# 6.17.2.13 OCT\_truncate()

```
void OCT_truncate (
```

```
octad * O,
int n )
```

Reverse bytes in an octad.

### **Parameters**

| 0 | octad to be truncated  |
|---|------------------------|
| n | the new shorter length |

# 6.17.2.14 OCT\_copy()

Copy one octad into another.

### **Parameters**

| 0 | octad to be copied to   |
|---|-------------------------|
| Р | octad to be copied from |

# 6.17.2.15 OCT\_output\_hex()

Output octad as hex string.

# **Parameters**

| 0   | octad to be output               |
|-----|----------------------------------|
| max | the maximum output length        |
| s   | the char array to receive output |

# 6.17.2.16 OCT\_output\_string()

6.18 tls\_octads.h 113

```
int max,
char * s )
```

Output octad as C ascii string.

#### **Parameters**

| 0   | octad to be output               |
|-----|----------------------------------|
| max | the maximum output length        |
| s   | the char array to receive output |

### 6.17.2.17 OCT\_output\_base64()

Output octad as base64 string.

#### **Parameters**

| 0   | octad to be output               |
|-----|----------------------------------|
| max | the maximum output length        |
| s   | the char array to receive output |

# 6.18 tls\_octads.h

### Go to the documentation of this file.

```
00008 #ifndef TLS_OCTADS_H
00009 #define TLS_OCTADS_H
00010
00011 // An octad - "a group or set of eight" - Oxford dictionary
00012
00013 //#define TLS_ARDUINO
                                      /**< Define for Arduino-based implementation */
00014
00015 #include <stddef.h>
00016
00021 #ifndef TLS_ARDUINO
00022 extern unsigned long millis();
00023 #endif
00027 typedef struct
00028 {
00029
         int len;
00030
         int max;
         char *val;
00031
00032 } octad;
00041 extern void OCT_append_int(octad \star0, unsigned int x, int len);
00042
00048 extern void OCT_append_octad(octad *0, octad *P);
00049
00056 extern bool OCT_compare(octad *0, octad *P);
00057
00065 extern void OCT_shift_left(octad \star0, int n);
00066
```

```
00071 extern void OCT_kill(octad *O);
00078 extern void OCT_from_hex(octad *0, char *src);
00079
00085 extern void OCT_append_string(octad *0, char *s);
00086
00093 extern void OCT_append_byte(octad *0, int b, int n);
00094
00101 extern void OCT_append_bytes(octad *O, char *s, int n);
00102
00109 extern void OCT_from_base64(octad *0, char *b);
00110
00115 extern void OCT_reverse(octad *0);
00116
00122 extern void OCT_truncate(octad *O,int n);
00123
00130 extern void OCT_copy(octad *O, octad *P);
00131
00139 extern bool OCT_output_hex(octad *O,int max,char *s);
00148 extern bool OCT_output_string(octad *O,int max,char *s);
00149
00157 extern void OCT_output_base64(octad *O,int max,char *s);
00158 #endif
```

# 6.19 tls\_pqibe.h File Reference

```
Ducas et al. IBE.
```

```
#include "core.h"
```

### **Functions**

```
    void PQIBE_CCA_ENCRYPT (char *ID, octet *R32, octet *KEY, octet *CT)
    IBE KEM CCA encrypt.
```

void PQIBE\_CCA\_DECRYPT (char \*ID, const int16\_t \*csk, octet \*CT, octet \*KEY)
 IBE KEM CCA decrypt.

# 6.19.1 Detailed Description

Ducas et al. IBE.

**Author** 

Mike Scott

### 6.19.2 Function Documentation

### 6.19.2.1 PQIBE CCA ENCRYPT()

IBE KEM CCA encrypt.

6.20 tls\_pqibe.h 115

#### **Parameters**

| ID  | Identity                     |
|-----|------------------------------|
| R32 | 32 random bytes              |
| KEY | random session key generated |
| CT  | encapsulating ciphertext     |

# 6.19.2.2 PQIBE\_CCA\_DECRYPT()

### IBE KEM CCA decrypt.

#### **Parameters**

| ID  | Identity           |
|-----|--------------------|
| csk | secret key         |
| CT  | ciphertext         |
| KEY | output session key |

# 6.20 tls\_pqibe.h

### Go to the documentation of this file.

```
00001
00007 //
00008 // Lattice-based B&F IBE 128-bit API Functions
00009 // Ducas et al. Method
00010 // Implementation by M.Scott
00011 //
00012
00013 #ifndef PQIBE_H
00015
00016 #define PQIBE_H
00015
00016 #include "core.h"
00017
00018 using namespace core;
00019
00027 extern void PQIBE_CCA_ENCRYPT(char *ID,octet *R32,octet *KEY,octet *CT);
00028
00036 extern void PQIBE_CCA_DECRYPT(char *ID,const int16_t *csk,octet *CT,octet *KEY);
00037
00038 #endif
```

# 6.21 tls\_protocol.h File Reference

TLS 1.3 main client-side protocol functions.

```
#include "tls_keys_calc.h"
#include "tls_cert_chain.h"
#include "tls_client_recv.h"
#include "tls_client_send.h"
#include "tls_tickets.h"
#include "tls_logger.h"
```

### **Functions**

```
    TLS_session TLS13_start (Socket *client, char *hostname)
        initialise a TLS 1.3 session structure
    void TLS13_end (TLS_session *session)
        terminate a session structure
    void TLS13_stop (TLS_session *session)
        stop sending - send CLOSE_NOTIFY and DISCONNECT
    bool TLS13_connect (TLS_session *session, octad *EARLY)
        TLS 1.3 forge connection.
    void TLS13_send (TLS_session *session, octad *DATA)
        TLS 1.3 send data.
    int TLS13_recv (TLS_session *session, octad *DATA)
        TLS 1.3 receive data.
```

# 6.21.1 Detailed Description

TLS 1.3 main client-side protocol functions.

void TLS13\_clean (TLS\_session \*session)

TLS 1.3 end session, delete keys, clean up buffers.

**Author** 

Mike Scott

### 6.21.2 Function Documentation

### 6.21.2.1 TLS13\_start()

initialise a TLS 1.3 session structure

| client   | the socket connection to the Server |
|----------|-------------------------------------|
| hostname | the host name (URL) of the server   |

Returns

an initialised TLS1.3 session structure

# 6.21.2.2 TLS13\_end()

terminate a session structure

# **Parameters**

| session  | the session structure  |
|----------|------------------------|
| 30331011 | lilo sossion structuro |

### 6.21.2.3 TLS13\_stop()

stop sending - send CLOSE\_NOTIFY and DISCONNECT

### **Parameters**

```
session the session structure
```

### 6.21.2.4 TLS13\_connect()

# TLS 1.3 forge connection.

### **Parameters**

| session | an initialised TLS session structure |
|---------|--------------------------------------|
| EARLY   | some early data to be transmitted    |

### Returns

false for failure, true for success

# 6.21.2.5 TLS13\_send()

# TLS 1.3 send data.

#### **Parameters**

| session | an initialised TLS session structure |
|---------|--------------------------------------|
| DATA    | some data to be transmitted          |

# 6.21.2.6 TLS13\_recv()

#### TLS 1.3 receive data.

### **Parameters**

| session | an initialised TLS session structure |
|---------|--------------------------------------|
| DATA    | that has been received               |

### Returns

0 for failure, otherwise success

# 6.21.2.7 TLS13\_clean()

TLS 1.3 end session, delete keys, clean up buffers.

| :       | an initialized TLC assertant structure |
|---------|--|
| session | an initialised TLS session structure   |

6.22 tls\_protocol.h 119

# 6.22 tls\_protocol.h

#### Go to the documentation of this file.

```
00007 // Main TLS 1.3 Protocol
00009 #ifndef TLS_PROTOCOL_H
00010 #define TLS_PROTOCOL_H
00011
00012 #include "tls_keys_calc.h"
00013 #include "tls_cert_chain.h"
00014 #include "tls_client_recv.h"
00015 #include "tls_client_send.h"
00016 #include "tls_tickets.h"
00017 #include "tls_logger.h"
00018
00025 extern TLS_session TLS13_start(Socket *client,char *hostname);
00026
00031 extern void TLS13_end(TLS_session *session);
00032
00033
00038 extern void TLS13_stop(TLS_session *session);
00039
00040
00047 extern bool TLS13_connect(TLS_session *session,octad *EARLY);
00048
00054 extern void TLS13_send(TLS_session *session,octad *DATA);
00055
00062 extern int TLS13_recv(TLS_session *session,octad *DATA);
00063
00068 extern void TLS13_clean(TLS_session *session);
```

# 6.23 tls\_sal.h File Reference

Security Abstraction Layer for TLS.

```
#include "tls1_3.h"
```

### **Functions**

```
char * SAL_name ()
```

Return name of SAL provider.

• int SAL\_ciphers (int \*ciphers)

Return supported ciphers.

int SAL\_groups (int \*groups)

Return supported groups in preferred order.

int SAL\_sigs (int \*sigAlgs)

Return supported TLS signature algorithms in preferred order.

• int SAL\_sigCerts (int \*sigAlgsCert)

Return supported TLS signature algorithms for Certificates in preferred order.

• bool SAL\_initLib ()

Initialise library for use.

void SAL endLib ()

finish use of library

int SAL\_hashType (int cipher\_suite)

return hash type asspciated with a cipher suite

• int SAL\_hashLen (int hash\_type)

return output length of hash function associated with a hash type

```
• int SAL_aeadKeylen (int cipher_suite)
     return key length associated with a cipher suite

    int SAL_aeadTaglen (int cipher_suite)

     return authentication tag length associated with a cipher suite

    int SAL_randomByte ()

     get a random byte

    void SAL randomOctad (int len, octad *R)

     get a random octad

    void SAL_hkdfExtract (int sha, octad *PRK, octad *SALT, octad *IKM)

     HKDF Extract function.

    void SAL_hkdfExpand (int htype, int olen, octad *OKM, octad *PRK, octad *INFO)

     Special HKDF Expand function (for TLS)

    void SAL_hmac (int htype, octad *T, octad *K, octad *M)

     simple HMAC function

    void SAL_hashNull (int sha, octad *H)

     simple HASH of nothing function

    void SAL_hashInit (int hlen, unihash *h)

     Initiate Hashing context.

    void SAL hashProcessArray (unihash *h, char *b, int len)

     Hash process an array of bytes.

    int SAL hashOutput (unihash *h, char *d)

     Hash finish and output.

    void SAL_aeadEncrypt (crypto *send, int hdrlen, char *hdr, int ptlen, char *pt, octad *TAG)

     AEAD encryption.

    bool SAL aeadDecrypt (crypto *recv, int hdrlen, char *hdr, int ctlen, char *ct, octad *TAG)

     AEAD decryption.

    void SAL_generateKeyPair (int group, octad *SK, octad *PK)

     generate a public/private key pair in an approved group for a key exchange

    bool SAL_generateSharedSecret (int group, octad *SK, octad *PK, octad *SS)

     generate a Diffie-Hellman shared secret

    bool SAL_tlsSignatureVerify (int sigAlg, octad *TRANS, octad *SIG, octad *PUBKEY)

      Verify a generic TLS signature.

    void SAL_tlsSignature (int sigAlg, octad *KEY, octad *TRANS, octad *SIG)

     Apply a generic TLS transcript signature.
```

# 6.23.1 Detailed Description

Security Abstraction Layer for TLS.

**Author** 

Mike Scott

#### 6.23.2 Function Documentation

# 6.23.2.1 SAL\_name()

```
char * SAL_name ( )
```

Return name of SAL provider.

Returns

name of SAL provider

# 6.23.2.2 SAL\_ciphers()

Return supported ciphers.

#### **Parameters**

### Returns

number of supported ciphers

# 6.23.2.3 SAL\_groups()

```
int SAL_groups (
          int * groups )
```

Return supported groups in preferred order.

# **Parameters**

| s array of su | pported groups |
|---------------|----------------|
|---------------|----------------|

# Returns

number of supported groups

### 6.23.2.4 SAL\_sigs()

```
int SAL_sigs (
          int * sigAlgs )
```

Return supported TLS signature algorithms in preferred order.

**Parameters** 

```
sigAlgs array of supported signature algorithms
```

Returns

number of supported groups

# 6.23.2.5 SAL\_sigCerts()

Return supported TLS signature algorithms for Certificates in preferred order.

**Parameters** 

```
sigAlgsCert array of supported signature algorithms for Certificates
```

Returns

number of supported groups

# 6.23.2.6 SAL\_initLib()

```
bool SAL_initLib ( )
```

Initialise library for use.

Returns

return true if successful, else false

# 6.23.2.7 SAL\_endLib()

```
void SAL_endLib ( )
```

finish use of library

### 6.23.2.8 SAL\_hashType()

return hash type asspciated with a cipher suite

**Parameters** 

```
cipher_suite a TLS cipher suite
```

#### Returns

hash function output length

# 6.23.2.9 SAL\_hashLen()

return output length of hash function associated with a hash type

#### **Parameters**

```
hash_type a TLS hash type
```

# Returns

hash function output length

# 6.23.2.10 SAL\_aeadKeylen()

return key length associated with a cipher suite

### **Parameters**

| cipher_suite | a TLS cipher suite |
|--------------|--------------------|
|--------------|--------------------|

Returns

key length

# 6.23.2.11 SAL\_aeadTaglen()

return authentication tag length associated with a cipher suite

**Parameters** 

```
cipher_suite a TLS cipher suite
```

Returns

tag length

# 6.23.2.12 SAL\_randomByte()

```
int SAL_randomByte ( )
```

get a random byte

Returns

a random byte

# 6.23.2.13 SAL\_randomOctad()

```
void SAL_randomOctad ( int \ len, \\ octad * R )
```

get a random octad

### **Parameters**

| len | number of random bytes               |
|-----|--------------------------------------|
| R   | octad to be filled with random bytes |

# 6.23.2.14 SAL\_hkdfExtract()

```
void SAL_hkdfExtract (
    int sha,
    octad * PRK,
    octad * SALT,
    octad * IKM )
```

HKDF Extract function.

### **Parameters**

| sha  | hash algorithm             |
|------|----------------------------|
| PRK  | an output Key              |
| SALT | public input salt          |
| IKM  | raw secret keying material |

# 6.23.2.15 SAL\_hkdfExpand()

```
void SAL_hkdfExpand (
    int htype,
    int olen,
    octad * OKM,
    octad * PRK,
    octad * INFO )
```

Special HKDF Expand function (for TLS)

| htype | hash algorithm                            |
|-------|---|
| olen  | is the desired length of the expanded key |
| OKM   | an expanded output Key                    |
| PRK   | is the fixed length input key             |
| INFO  | is public label information               |

# 6.23.2.16 SAL\_hmac()

### simple HMAC function

#### **Parameters**

| htype | hash algorithm        |
|-------|-----------------------|
| T     | an output tag         |
| K     | an input key, or salt |
| М     | an input message      |

# 6.23.2.17 SAL\_hashNull()

```
void SAL_hashNull (
          int sha,
          octad * H )
```

simple HASH of nothing function

#### **Parameters**

| sha | the SHA2 function output length (32,48 or 64) |
|-----|---|
| Н   | the output hash                               |

# 6.23.2.18 SAL\_hashInit()

```
void SAL_hashInit ( int \ hlen, unihash * h \ )
```

Initiate Hashing context.

| hlen | length in bytes of SHA2 hashing output |
|------|--|
| h    | a hashing context                      |

# 6.23.2.19 SAL\_hashProcessArray()

Hash process an array of bytes.

#### **Parameters**

| h   | a hashing context                     |
|-----|---------------------------------------|
| b   | the byte array to be included in hash |
| len | the array length                      |

### 6.23.2.20 SAL\_hashOutput()

Hash finish and output.

### **Parameters**

| h | a hashing context   |
|---|---|
| d | the current output digest of an ongoing hashing operation |

### Returns

hash output length

# 6.23.2.21 SAL\_aeadEncrypt()

AEAD encryption.

### **Parameters**

| send   | the AES key and IV                        |
|--------|---|
| hdrlen | the length of the header                  |
| hdr    | the header bytes                          |
| ptlen  | the plaintext length                      |
| pt     | the input plaintext and output ciphertext |
| TAG    | the output authentication tag             |

# 6.23.2.22 SAL\_aeadDecrypt()

# AEAD decryption.

#### **Parameters**

| recv   | the AES key and IV                        |
|--------|---|
| hdrlen | the length of the header                  |
| hdr    | the header bytes                          |
| ctlen  | the ciphertext length                     |
| ct     | the input ciphertext and output plaintext |
| TAG    | the expected authentication tag           |

### Returns

false if tag is wrong, else true

# 6.23.2.23 SAL\_generateKeyPair()

generate a public/private key pair in an approved group for a key exchange

#### **Parameters**

| group | roup the cryptographic group used to generate the key pair |  |
|-------|--|--|
| SK    | the output Private Key                                     |  |
| PK    | the output Public Key                                      |  |

# 6.23.2.24 SAL\_generateSharedSecret()

```
bool SAL_generateSharedSecret (
    int group,
    octad * SK,
    octad * PK,
    octad * SS )
```

generate a Diffie-Hellman shared secret

#### **Parameters**

| group | the cryptographic group used to generate the shared secret |  |
|-------|--|--|
| SK    | the input client private key                               |  |
| PK    | the input server public Key                                |  |
| SS    | the output shared secret                                   |  |

# Returns

false for all zeros, else true

#### 6.23.2.25 SAL\_tlsSignatureVerify()

```
bool SAL_tlsSignatureVerify (  & \text{int } sigAlg, \\ & \text{octad} * TRANS, \\ & \text{octad} * SIG, \\ & \text{octad} * PUBKEY )
```

Verify a generic TLS signature.

#### **Parameters**

| sigAlg | the signature type                          |
|--------|---|
| TRANS  | the signed input transcript hash            |
| SIG    | the input signature                         |
| PUBKEY | the public key used to verify the signature |

#### Returns

true if signature is valid, else false

#### 6.23.2.26 SAL\_tlsSignature()

Apply a generic TLS transcript signature.

#### **Parameters**

| sigAlg | the signature type                         |
|--------|--|
| KEY    | the private key used to form the signature |
| TRANS  | the input transcript hash to be signed     |
| SIG    | the output signature                       |

# 6.24 tls\_sal.h

# Go to the documentation of this file.

```
00007 // Process input received from Server
80000
00009 #ifndef TLS_SAL_H
00010 #define TLS_SAL_H
00011
00012 // Use MIRACL core library
00013
00014 #include "tls1_3.h"
00015
00020 extern char *SAL_name();
00021
00027 extern int SAL_ciphers(int *ciphers);
00028
00034 extern int SAL_groups(int *groups);
00035
00041 extern int SAL_sigs(int *sigAlgs);
00042
00048 extern int SAL_sigCerts(int *sigAlgsCert);
00049
00054 extern bool SAL_initLib();
00055
00059 extern void SAL_endLib();
00060
00066 extern int SAL_hashType(int cipher_suite);
00073 extern int SAL_hashLen(int hash_type);
00074
00080 int SAL_aeadKeylen(int cipher_suite);
00081
00087 int SAL_aeadTaglen(int cipher_suite);
00088
00093 extern int SAL_randomByte();
00094
00100 extern void SAL_randomOctad(int len, octad \star R);
00101
00109 extern void SAL_hkdfExtract(int sha,octad *PRK,octad *SALT,octad *IKM);
00110
```

```
00119 extern void SAL_hkdfExpand(int htype, int olen, octad *OKM,octad *PRK, octad *INFO);
00128 extern void SAL_hmac(int htype,octad *T,octad *K,octad *M);
00129
00135 extern void SAL hashNull(int sha,octad *H);
00136
00137 // hash functions
00138
00144 extern void SAL_hashInit(int hlen,unihash *h);
00145
00152 extern void SAL_hashProcessArray(unihash *h,char *b,int len);
00153
00154
00161 extern int SAL_hashOutput(unihash *h,char *d);
00162
00172 extern void SAL_aeadEncrypt(crypto *send,int hdrlen,char *hdr,int ptlen,char *pt,octad *TAG);
00173
00184 extern bool SAL_aeadDecrypt(crypto *recv,int hdrlen,char *hdr,int ctlen,char *ct,octad *TAG);
00192 extern void SAL_generateKeyPair(int group,octad *SK,octad *PK);
00193
00202 extern bool SAL_generateSharedSecret(int group,octad *SK,octad *PK,octad *SS);
00203
00204
00213 extern bool SAL_tlsSignatureVerify(int sigAlg,octad *TRANS,octad *SIG,octad *PUBKEY);
00222 extern void SAL_tlsSignature(int sigAlg,octad *KEY,octad *TRANS,octad *SIG);
00223
00224
00225 #endif
```

# 6.25 tls\_sockets.h File Reference

set up sockets for reading and writing

```
#include <string.h>
#include "tls_octads.h"
#include <time.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/un.h>
```

#### **Data Structures**

class Socket
 Socket instance.

#### **Functions**

- int setclientsock (int port, char \*ip, int toms)
   create a client socket
   int getlPaddress (char \*ip, char \*hostname)
   get the IP address from a URL
- void sendOctad (Socket \*client, octad \*B)

send an octad over a socket

```
    void sendLen (Socket *client, int len)
```

send a 16-bit integer as an octad to Server

• int getBytes (Socket \*client, char \*b, int expected)

receive bytes over a socket sonnection

• int getInt16 (Socket \*client)

receive 16-bit integer from a socket

• int getInt24 (Socket \*client)

receive 24-bit integer from a socket

int getByte (Socket \*client)

receive a single byte from a socket

• int getOctad (Socket \*client, octad \*B, int expected)

receive an octad from a socket

# 6.25.1 Detailed Description

set up sockets for reading and writing

**Author** 

Mike Scott

# 6.25.2 Function Documentation

# 6.25.2.1 setclientsock()

```
int setclientsock (
    int port,
    char * ip,
    int toms )
```

create a client socket

#### **Parameters**

| port | the TCP/IP port on which to connect  |
|------|--------------------------------------|
| ip   | the IP address with which to connect |
| toms | the time-out period in milliseconds  |

#### Returns

the socket handle

# 6.25.2.2 getlPaddress()

```
int getIPaddress ( \label{eq:char} \mbox{char} \; * \; ip, \\ \mbox{char} \; * \; hostname \; )
```

get the IP address from a URL

#### **Parameters**

| ip       | the IP address              |
|----------|-----------------------------|
| hostname | the input Server name (URL) |

#### Returns

1 for success, 0 for failure

# 6.25.2.3 sendOctad()

send an octad over a socket

#### **Parameters**

| client | the socket connection to the Server |
|--------|-------------------------------------|
| В      | the octad to be transmitted         |

# 6.25.2.4 sendLen()

send a 16-bit integer as an octad to Server

# **Parameters**

| client | the socket connection to the Server                       |
|--------|---|
| len    | the 16-bit integer to be encoded as octad and transmitted |

# 6.25.2.5 getBytes()

receive bytes over a socket sonnection

#### **Parameters**

| client   | the socket connection to the Server |
|----------|-------------------------------------|
| b        | the received bytes                  |
| expected | the number of bytes expected        |

#### Returns

-1 on failure, 0 on success

# 6.25.2.6 getInt16()

receive 16-bit integer from a socket

#### **Parameters**

| client the socket connection to the Server |
|--|
|--|

#### Returns

a 16-bit integer

# 6.25.2.7 getInt24()

receive 24-bit integer from a socket

#### **Parameters**

| client | the socket connection to the Server |
|--------|-------------------------------------|
|--------|-------------------------------------|

6.26 tls\_sockets.h

#### Returns

a 24-bit integer

# 6.25.2.8 getByte()

receive a single byte from a socket

#### **Parameters**

#### Returns

a byte

# 6.25.2.9 getOctad()

receive an octad from a socket

# **Parameters**

| client   | the socket connection to the Server |
|----------|-------------------------------------|
| В        | the output octad                    |
| expected | the number of bytes expected        |

#### Returns

-1 on failure, 0 on success

# 6.26 tls\_sockets.h

# Go to the documentation of this file. $_{\tt 00001}$

```
00001
00008 // Set up and read/write sockets
00009
00010 #ifndef TLS_SOCKETS_H
```

```
00011 #define TLS_SOCKETS_H
00012
00013 #include <string.h>
00014 #include "tls_octads.h"
00015
00016 #ifdef TLS_ARDUINO
00017 #include "tls_wifi.h"
00018 #else
00019 #include <time.h>
00020 #include <unistd.h>
00021 #include <stdio.h>
00022 #include <sys/socket.h>
00023 #include <arpa/inet.h>
00024 #include <stdlib.h>
00025 #include <netinet/in.h>
00026 #include <netdb.h>
00027 #include <netinet/in.h>
00028 #include <sys/un.h>
00029 #endif
00030
00031 #ifndef TLS_ARDUINO
00032
00040 extern int setclientsock(int port, char *ip, int toms);
00041
00048 extern int getIPaddress(char *ip,char *hostname);
00050 // Simple socket class, mimics Arduino
00053 class Socket
00054 {
00055
          int sock;
00056
          int toms:
00057
          bool is_af_unix;
00059 private:
00060
          Socket(bool is_af_unix) {
              this->sock = 0;
this->toms = 5000;
00061
00062
00063
              this->is_af_unix = is_af_unix;
00064
          }
00065
00066
          static int afunix_setclientsock(const char *const socket_path)
00067
00068
              int sock:
00069
              struct sockaddr un serv addr;
              if ((sock = socket(AF_UNIX, SOCK_STREAM, 0)) < 0)</pre>
00070
                  return -2;
00071
00072
00073
              serv_addr.sun_family = AF_UNIX;
00074
              strcpy(serv_addr.sun_path, socket_path);
00075
00076
              if (::connect(sock, (struct sockaddr *)&serv addr, sizeof(serv addr)) < 0)</pre>
                  return -1;
00078
00079
              return sock;
08000
          }
00081
00082 public:
00084
          bool connect(char *host,int port) {
00085
             if(!this->is_af_unix) {
00086
                  char ip[40];
00087
                  sock = 0:
                  if (!getIPaddress(ip, host))
00088
00089
                       return false;
00090
                  sock = setclientsock(port, ip, toms);
00091
                  if (sock <= 0)
00092
                       return false;
00093
                  return true;
00094
              } else {
00095
                 bool connected = true;
                  sock = afunix_setclientsock(host);
00096
00097
                  if (sock <= 0) {</pre>
00098
                       connected = false;
00099
00100
                   return connected;
              }
00101
00102
          }
00103
00104
          return Socket (false);
}
          static Socket InetSocket() {
00105
00106
00107
00108
          static Socket UnixSocket() {
00109
            return Socket (true);
00110
00111
          void setTimeout(int to) {toms=to;}
00112
00113
          int write (char *buf, int len) {return ::send(sock, buf, len, 0);}
```

```
int read(char *buf,int len) {return ::recv(sock,buf,len,0);}
00115
         void stop() {::close(sock);}
00116
00117
         ~Socket() {::close(sock);}
00118 };
00119 #else
00120
00126 extern void clearsoc(Socket &client,octad *IO);
00127
00128 #endif
00129
00135 extern void sendOctad(Socket *client,octad *B);
00136
00142 extern void sendLen(Socket *client,int len);
00143
00151 extern int getBytes(Socket *client, char *b, int expected);
00152
00158 extern int getInt16(Socket *client);
00165 extern int getInt24(Socket *client);
00166
00172 extern int getByte(Socket *client);
00173
00181 extern int getOctad(Socket *client,octad *B,int expected);
00182
00183 #endif
```

# 6.27 tls\_tickets.h File Reference

TLS 1.3 process resumption tickets.

```
#include "tls1_3.h"
#include "tls_client_recv.h"
```

#### **Functions**

int parseTicket (octad \*TICK, unsign32 birth, ticket \*T)

parse a received ticket octad into a ticket structure

void initTicketContext (ticket \*T)

initialize a ticket structure

void endTicketContext (ticket \*T)

terminate a ticket structure

bool ticket\_still\_good (ticket \*T)

Check that a ticket is still good, and not out-of-date.

# 6.27.1 Detailed Description

TLS 1.3 process resumption tickets.

**Author** 

Mike Scott

### 6.27.2 Function Documentation

# 6.27.2.1 parseTicket()

parse a received ticket octad into a ticket structure

#### **Parameters**

| TICK  | the input ticket octad       |
|-------|------------------------------|
| T     | the output ticket structure  |
| birth | the birth time of the ticket |

#### Returns

bad ticket error, or 0 if ticket is good

# 6.27.2.2 initTicketContext()

initialize a ticket structure

#### **Parameters**

T the ticket structure

# 6.27.2.3 endTicketContext()

terminate a ticket structure

# **Parameters**

T the ticket structure

6.28 tls\_tickets.h 139

#### 6.27.2.4 ticket\_still\_good()

Check that a ticket is still good, and not out-of-date.

**Parameters** 

```
T the ticket structure
```

Returns

true if ticket is still good

# 6.28 tls\_tickets.h

#### Go to the documentation of this file.

```
00001
00007 // Process Resumption Tickets
80000
00009 #ifndef TLS_TICKETS_H
00010 #define TLS_TICKETS_H
00011
00012 #include "tls1_3.h"
00013 #include "tls_client_recv.h"
00014
00022 extern int parseTicket(octad *TICK,unsign32 birth,ticket *T);
00023
00028 extern void initTicketContext(ticket *T);
00029
00034 extern void endTicketContext(ticket *T);
00035
00041 extern bool ticket_still_good(ticket *T);
00042
00043 #endif
```

# 6.29 tls\_wifi.h File Reference

define Socket structure depending on processor context

```
#include "tls1_3.h"
```

#### 6.29.1 Detailed Description

define Socket structure depending on processor context

Author

Mike Scott

# 6.30 tls wifi.h

# Go to the documentation of this file.

```
00001
00007 // Set up WiFi environment for Arduino boards
00008
00009 #ifndef TLS_WIFI_H
00010 #define TLS_WIFI_H
00011
00012 #include "tls1_3.h"
00013
00014 #ifdef TLS_ARDUINO
00015
00016 #ifdef FISHINO_PIRANHA
00017 // Fishino Piranha board
00018 #define PARTICULAR_BOARD
00019 #include <Fishino.h>
00020 #include <SPI.h>
00021 typedef FishinoClient Socket;
00022 #define WiFi Fishino
00023 #define FISHINO
00024
00025 #endif
00026
00027 #ifdef ESP32
00028 // ESP32 board
00029 #define PARTICULAR_BOARD
00030 #include <WiFi.h>
00031 typedef WiFiClient Socket;
00032
00033 #endif
00034
00035 #ifndef PARTICULAR_BOARD
00036 // any other board
00037 #include <WiFiNINA.h>
00038 typedef WiFiClient Socket;
00039
00040 #endif
00041 #endif
00042
00043 #endif
```

# 6.31 tls\_x509.h File Reference

X509 function Header File.

#### **Data Structures**

struct pktype
 Public key type.

#### **Macros**

- #define X509\_ECC 1
- #define X509 RSA 2
- #define X509\_ECD 3
- #define X509\_PQ 4
- #define X509 HY 5
- #define X509\_H256 2
- #define X509 H384 3
- #define X509\_H512 4
- #define USE\_NIST256 0
- #define USE\_C25519 1
- #define USE\_NIST384 10
- #define USE\_NIST521 12

#### **Functions**

```
    void ecdsa_sig_encode (octad *c)
```

in-place ECDSA signature encoding

int ecdsa\_sig\_decode (octad \*c)

in-place ECDSA signature decoding

pktype X509\_extract\_private\_key (octad \*c, octad \*pk)

Extract private key.

pktype X509\_extract\_cert\_sig (octad \*c, octad \*s)

Extract certificate signature.

- int X509 extract cert (octad \*sc, octad \*c)
- int X509\_find\_public\_key (octad \*c, int \*ptr)
- pktype X509 get public key (octad \*c, octad \*key)
- pktype X509 extract public key (octad \*c, octad \*k)
- int X509 find issuer (octad \*c)
- int X509\_find\_validity (octad \*c)
- int X509\_find\_subject (octad \*c)
- int X509\_self\_signed (octad \*c)
- int X509\_find\_entity\_property (octad \*c, octad \*S, int s, int \*f)
- int X509\_find\_start\_date (octad \*c, int s)
- int X509\_find\_expiry\_date (octad \*c, int s)
- int X509\_find\_extensions (octad \*c)
- int X509\_find\_extension (octad \*c, octad \*S, int s, int \*f)
- int X509\_find\_alt\_name (octad \*c, int s, char \*name)

#### **Variables**

- octad X509 CN
- octad X509\_ON
- octad X509 EN
- octad X509\_LN
- octad X509 UN
- octad X509\_MN
- octad X509\_SN
- octad X509\_AN
- octad X509\_KUoctad X509\_BC

# 6.31.1 Detailed Description

X509 function Header File.

**Author** 

Mike Scott

defines structures declares functions

#### 6.31.2 Macro Definition Documentation

# 6.31.2.1 X509\_ECC

#define X509\_ECC 1

Elliptic Curve data type detected

# 6.31.2.2 X509\_RSA

#define X509\_RSA 2

RSA data type detected

# 6.31.2.3 X509\_ECD

#define X509\_ECD 3

Elliptic Curve (Ed25519) detected

#### 6.31.2.4 X509\_PQ

#define X509\_PQ 4

Post Quantum method

# 6.31.2.5 X509\_HY

#define X509\_HY 5

Hybrid Post\_Quantum

# 6.31.2.6 X509\_H256

#define X509\_H256 2

SHA256 hash algorithm used

# 6.31.2.7 X509\_H384

#define X509\_H384 3

SHA384 hash algorithm used

# 6.31.2.8 X509\_H512

#define X509\_H512 4

SHA512 hash algorithm used

#### 6.31.2.9 USE\_NIST256

```
#define USE_NIST256 0
```

For the NIST 256-bit standard curve - WEIERSTRASS only

# 6.31.2.10 USE\_C25519

```
#define USE_C25519 1
```

Bernstein's Modulus 2<sup>255-19</sup> - EDWARDS or MONTGOMERY only

#### 6.31.2.11 USE\_NIST384

```
#define USE_NIST384 10
```

For the NIST 384-bit standard curve - WEIERSTRASS only

#### 6.31.2.12 USE\_NIST521

```
#define USE_NIST521 12
```

For the NIST 521-bit standard curve - WEIERSTRASS only

### 6.31.3 Function Documentation

#### 6.31.3.1 ecdsa\_sig\_encode()

```
void ecdsa_sig_encode ( octad * c )
```

in-place ECDSA signature encoding

**Parameters** 

 $c \mid$  an ecdsa signature to be converted from r|s form to ASN.1

# 6.31.3.2 ecdsa\_sig\_decode()

```
int ecdsa_sig_decode ( octad * c )
```

in-place ECDSA signature decoding

#### **Parameters**

 $c \mid$  an ecdsa signature to be converted from ASN.1 to simple r|s form

#### Returns

index into c where conversion ended

# 6.31.3.3 X509\_extract\_private\_key()

Extract private key.

#### **Parameters**

| С  | an X.509 private key  |
|----|---|
| pk | the extracted private key - for RSA octad = $p q dp dq c$ , for ECC octad = k |

# Returns

0 on failure, or indicator of private key type (ECC or RSA)

# 6.31.3.4 X509\_extract\_cert\_sig()

Extract certificate signature.

#### **Parameters**

| С | an X.509 certificate    |
|---|-------------------------|
| s | the extracted signature |

#### Returns

0 on failure, or indicator of signature type (ECC or RSA)

# 6.31.3.5 X509\_extract\_cert()

#### **Parameters**

| sc | a signed certificate      |
|----|---------------------------|
| С  | the extracted certificate |

#### Returns

0 on failure

# 6.31.3.6 X509\_find\_public\_key()

#### **Parameters**

| С   | an X.509 certificate            |
|-----|---------------------------------|
| ptr | pointer to ASN.1 raw public key |

#### Returns

length of raw public key

# 6.31.3.7 X509\_get\_public\_key()

### **Parameters**

| С   | an ASN.1 encoded public key |
|-----|-----------------------------|
| key | the extracted public key    |

#### Returns

indicator of public key type (ECC or RSA)

# 6.31.3.8 X509\_extract\_public\_key()

#### **Parameters**

| С | an X.509 certificate |
|---|----------------------|
| k | the extracted key    |

#### Returns

0 on failure, or indicator of public key type (ECC or RSA)

#### 6.31.3.9 X509\_find\_issuer()

```
int X509_find_issuer ( octad * c )
```

#### **Parameters**

c an X.509 certificate

# Returns

0 on failure, or pointer to issuer field in cert

# 6.31.3.10 X509\_find\_validity()

#### **Parameters**

c an X.509 certificate

#### Returns

0 on failure, or pointer to validity field in cert

# 6.31.3.11 X509\_find\_subject()

```
int X509_find_subject ( octad * c )
```

#### **Parameters**

```
c an X.509 certificate
```

#### Returns

0 on failure, or pointer to subject field in cert

# 6.31.3.12 X509\_self\_signed()

```
int X509_self_signed ( octad * c )
```

#### **Parameters**

```
c an X.509 certificate
```

# Returns

true if self-signed, else false

#### 6.31.3.13 X509\_find\_entity\_property()

### **Parameters**

| С | an X.509 certificate                                |
|---|---|
| S | is OID of property we are looking for               |
| s | is a pointer to the section of interest in the cert |
| f | is pointer to the length of the property            |

#### Returns

0 on failure, or pointer to the property

# 6.31.3.14 X509\_find\_start\_date()

#### **Parameters**

| С | an X.509 certificate                            |
|---|---|
| s | is a pointer to the start of the validity field |

#### Returns

0 on failure, or pointer to the start date

# 6.31.3.15 X509\_find\_expiry\_date()

#### **Parameters**

| С | an X.509 certificate                            |
|---|---|
| s | is a pointer to the start of the validity field |

# Returns

0 on failure, or pointer to the expiry date

# 6.31.3.16 X509\_find\_extensions()

```
int X509_find_extensions ( octad * c )
```

#### **Parameters**

c an X.509 certificate

#### Returns

0 on failure (or no extensions), or pointer to extensions field in cert

# 6.31.3.17 X509\_find\_extension()

# **Parameters**

| С | an X.509 certificate                                |
|---|---|
| S | is OID of particular extension we are looking for   |
| s | is a pointer to the section of interest in the cert |
| f | is pointer to the length of the extension           |

#### Returns

0 on failure, or pointer to the extension

# 6.31.3.18 X509\_find\_alt\_name()

### **Parameters**

| С    | an X.509 certificate                                  |
|------|---|
| s    | is a pointer to certificate extension SubjectAltNames |
| name | is a URL  |

#### Returns

0 on failure, 1 if URL is in list of alt names

# 6.31.4 Variable Documentation

# 6.31.4.1 X509\_CN

```
octad X509_CN [extern]
```

Country Name

# 6.31.4.2 X509\_ON

```
octad X509_ON [extern]
```

organisation Name

# 6.31.4.3 X509\_EN

```
octad X509_EN [extern]
```

email

#### 6.31.4.4 X509\_LN

```
octad X509_LN [extern]
```

local name

# 6.31.4.5 X509\_UN

```
octad X509_UN [extern]
```

Unit name (aka Organisation Unit OU)

# 6.31.4.6 X509\_MN

```
octad X509_MN [extern]
```

My Name (aka Common Name)

# 6.31.4.7 X509\_SN

```
octad X509_SN [extern]
```

State Name

# 6.31.4.8 X509\_AN

```
octad X509_AN [extern]
```

Alternate Name

#### 6.31.4.9 X509\_KU

```
octad X509_KU [extern]
Key Usage
```

#### 6.31.4.10 X509 BC

```
octad X509_BC [extern]
```

**Basic Constraints** 

# 6.32 tls x509.h

#### Go to the documentation of this file.

```
00011 #ifndef TLS_X509_H
00012 #define TLS_X509_H
00013
00014 // Supported Encryption Methods
00015
00016 #define X509_ECC 1
00017 #define X509_RSA 2
00018 #define X509_ECD 3
00019 #define X509_PQ 4
00020 #define X509_HY 5
00022 // Supported Hash functions
00023
00024 #define X509_H256 2
00025 #define X509_H384 3
00026 #define X509_H512 4
00028 // Supported Curves
00029
00030 #define USE_NIST256 0
00031 #define USE_C25519
00032 //#define USE_BRAINPOOL 2 /**< For Brainpool 256-bit curve - WEIERSTRASS only */
00033 //#define USE_ANSSI 3
                                  /**< For French 256-bit standard curve - WEIERSTRASS only */
00034 #define USE_NIST384 10
00035 #define USE_NIST521 12
00037 extern octad X509_CN;
00038 extern octad X509_ON;
00039 extern octad X509_EN;
00040 extern octad X509_LN;
00041 extern octad X509_UN;
00042 extern octad X509_MN;
00043 extern octad X509_SN;
00045 extern octad X509_AN;
00046 extern octad X509_KU;
00047 extern octad X509_BC;
00052 typedef struct
00053 {
00054
          int type;
00055
         int hash;
00056
          int curve;
00057 } pktype;
00058
00059
00064 extern void ecdsa_sig_encode(octad *c);
00065
00071 extern int ecdsa_sig_decode(octad *c);
00072
00073 /* X.509 functions */
00074
00081 extern pktype X509_extract_private_key(octad *c,octad *pk);
00082
00090 extern pktype X509_extract_cert_sig(octad *c, octad *s);
00097 extern int X509_extract_cert(octad *sc, octad *c);
00098
00105 extern int X509_find_public_key(octad *c,int *ptr);
00106
00113 extern pktype X509_get_public_key(octad *c,octad *key);
00114
00115
```

6.33 ECCX08.h 153

```
00122 extern pktype X509_extract_public_key(octad *c, octad *k);
00128 extern int X509_find_issuer(octad *c);
00134 extern int X509_find_validity(octad *c);
00140 extern int X509_find_subject(octad *c);
00141
00147 extern int X509_self_signed(octad *c);
00157 extern int X509_find_entity_property(octad *c, octad *S, int s, int *f);
00164 extern int X509_find_start_date(octad *c, int s);
00171 extern int X509_find_expiry_date(octad *c, int s);
00172
00178 extern int X509 find extensions(octad *c);
00187 extern int X509_find_extension(octad *c, octad *S, int s, int *f);
00188
00196 extern int X509_find_alt_name(octad *c,int s,char *name);
00197
00198 #endif
```

### 6.33 ECCX08.h

```
00002
        This file is part of the ArduinoECCX08 library.
00003
        Copyright (c) 2018 Arduino SA. All rights reserved.
00004
00005
        This library is free software: you can redistribute it and/or
        modify it under the terms of the GNU Lesser General Public
00006
00007
        License as published by the Free Software Foundation; either
80000
        version 2.1 of the License, or (at your option) any later version.
00009
00010
        This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
00011
00012
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
00013
        Lesser General Public License for more details.
00014
00015
        You should have received a copy of the GNU Lesser General Public
       License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
00016
00017
00018 */
00019
00020 #ifndef _ECCX08_H_
00021 #define _ECCX08_H_
00022
00023 #include <Arduino.h>
00024 #include <Wire.h>
00026 class ECCX08Class
00027 {
00028 public:
00029
        ECCX08Class(TwoWire& wire, uint8_t address);
        virtual ~ECCX08Class();
00030
00031
00032
        int begin();
00033
        void end();
00034
00035
        int serialNumber(byte sn[]);
00036
        String serialNumber();
00037
00038
        long random (long max);
00039
        long random(long min, long max);
00040
        int random(byte data[], size_t length);
00041
00042
        int generatePrivateKey(int slot, byte publicKey[]);
int generatePublicKey(int slot, byte publicKey[]);
int generateSharedKey(int slot, byte publicKey[], byte sharedKey[]); // M.Scott 12/7/2021
00043
00044
00045
00046
        int ecdsaVerify(const byte message[], const byte signature[], const byte pubkey[]);
00047
        int ecSign(int slot, const byte message[], byte signature[]);
00048
00049
        int challenge (const byte message[]);
00050
        int aesEncrypt(byte block[]);
00051
        int aesGFM(byte state[],byte H[]);
00052
        int beginSHA256();
00053
00054
        int beginHMAC(int slot);
00055
        int updateSHA256(const byte data[], int len); // variable
00056
        int endSHA256(byte result[]);
00057
        int endSHA256(const byte data[], int length, byte result[]);
00058
        int readSHA256(byte context[]);
00059
        int writeSHA256(byte context[],int length);
00060
00061
        int readSlot(int slot, byte data[], int length);
00062
        int writeSlot(int slot, const byte data[], int length);
00063
```

```
00064
         int locked();
00065
         int writeConfiguration(const byte data[]);
00066
         int readConfiguration(byte data[]);
00067
         int lock();
00068
00069 private:
00070
        int wakeup();
00071
         int sleep();
00072
         int idle();
00073
00074
         long version();
00075
         int verify(const byte signature[], const byte pubkey[]);
00076
         int sign(int slot, byte signature[]);
00077
00078
         int read(int zone, int address, byte buffer[], int length);
00079
         int write(int zone, int address, const byte buffer[], int length);
00080
         int lock(int zone);
00081
00082
         int addressForSlotOffset(int slot, int offset);
00083
00084
         int sendCommand(uint8_t opcode, uint8_t param1, uint16_t param2, const byte data[] = NULL, size_t
dataLength = 0);

00085 int receiveResponse(void* response, size_t length);

00086 int receiveResponse(void* response);

00087 uint16_t crc16(const byte data[], size_t length);
00088
00089 private:
00090 TwoWire* _wire;
00091 uint8_t _address;
00092
        static const uint32_t _wakeupFrequency;
static const uint32_t _normalFrequency;
00093
00094
00095 };
00096
00097 extern ECCX08Class ECCX08;
00098
00099 #endif
```

# Index

| active                | tls1_3.h, 48              |
|-----------------------|---------------------------|
| crypto, 13            | AUTHENTICATION_FAILURE    |
| addALPNExt            | tls1_3.h, 51              |
| tls_client_send.h, 82 |                           |
| addClientRawPublicKey | BAD_CERT_CHAIN            |
| tls_client_send.h, 81 | tls1_3.h, <mark>50</mark> |
| addCookieExt          | BAD_CERTIFICATE           |
| tls_client_send.h, 84 | tls1_3.h, 54              |
| addEarlyDataExt       | BAD_HANDSHAKE             |
| tls_client_send.h, 84 | tls1_3.h, <mark>53</mark> |
| addKeyShareExt        | BAD_HELLO                 |
| tls client send.h, 82 | tls1_3.h, <mark>51</mark> |
| addMFLExt             | BAD_MESSAGE               |
| tls_client_send.h, 82 | tls1_3.h, <mark>53</mark> |
| addPadding            | BAD_RECORD                |
| tls_client_send.h, 84 | tls1_3.h, 51              |
| addPostHSAuth         | BAD_RECORD_MAC            |
| tls client send.h, 84 | tls1_3.h, <mark>55</mark> |
| addPreSharedKeyExt    | BAD_REQUEST_UPDATE        |
| tls_client_send.h, 79 | tls1_3.h, <mark>53</mark> |
| addPSKModesExt        | BAD_TICKET                |
| tls_client_send.h, 83 | tls1_3.h, 51              |
| addRSLExt             | badResponse               |
| tls_client_send.h, 83 | tls_client_recv.h, 73     |
| addServerNameExt      | BFIBE_CCA_DECRYPT         |
| tls_client_send.h, 80 | tls_bfibe.h, 64           |
| addServerRawPublicKey | BFIBE_CCA_ENCRYPT         |
| tls_client_send.h, 80 | tls_bfibe.h, 63           |
| addSigAlgsCertExt     | birth                     |
| tls_client_send.h, 81 | ticket, 22                |
| addSigAlgsExt         | byte                      |
| tls_client_send.h, 81 | tls1_3.h, 57              |
| addSupportedGroupsExt |                           |
| tls_client_send.h, 80 | CA_NOT_FOUND              |
| addVersionExt         | tls1_3.h, <mark>52</mark> |
| tls_client_send.h, 83 | cacerts                   |
| age_obfuscator        | tls_certs.h, 68           |
| ticket, 21            | CERT_OUTOFDATE            |
| ALERT                 | tls1_3.h, <mark>52</mark> |
| tls1_3.h, 48          | CERT_REQUEST              |
| alert_from_cause      | tls1_3.h, 49              |
| tls_client_send.h, 90 | CERT_VERIFY               |
| ALLOW_SELF_SIGNED     | tls1_3.h, 49              |
| tls1 3.h, 36          | CERT_VERIFY_FAIL          |
| alpn                  | tls1_3.h, 53              |
| ee_status, 16         | CERTIFICATE               |
| APP_PROTOCOL          | tls1_3.h, 49              |
| tls1_3.h, 47          | CERTIFICATE_EXPIRED       |
| APPLICATION           | tls1_3.h, <mark>54</mark> |
| ALL LIGATION          | CHANGE CIPHER             |

| tls1_3.h, 48             | tls1_3.h, <mark>54</mark> |
|--------------------------|---------------------------|
| checkServerCertChain     | DECRYPT_ERROR             |
| tls_cert_chain.h, 65     | tls1_3.h, <mark>54</mark> |
| checkServerCertVerifier  | deriveApplicationSecrets  |
| tls_keys_calc.h, 99      | tls_keys_calc.h, 98       |
| checkVeriferData         | deriveEarlySecrets        |
| tls_keys_calc.h, 99      | tls_keys_calc.h, 97       |
| cipher_suite             | deriveHandshakeSecrets    |
| ticket, 22               | tls_keys_calc.h, 97       |
| TLS_session, 24          | deriveLaterSecrets        |
| cipherSuites             | tls_keys_calc.h, 97       |
| tls_client_send.h, 85    | deriveUpdatedKeys         |
| CLIENT_CERT              | tls_keys_calc.h, 98       |
| tls1_3.h, 36             | deriveVeriferData         |
| CLIENT_CERT_TYPE         | tls_keys_calc.h, 99       |
| tls1_3.h, 48             | DILITHIUM2                |
| client_cert_type         | tls1_3.h, 44              |
| TLS_session, 24          | DILITHIUM2_P256           |
| CLIENT_HELLO             | tls1_3.h, 44              |
| tls1_3.h, 49             | DILITHIUM3                |
| clientRandom             | tls1_3.h, 44              |
| tls_client_send.h, 85    | DLT_SS                    |
| CLOSE_NOTIFY             | tls1_3.h, 35              |
| tls1_3.h, 55             |                           |
| CLOSURE ALERT RECEIVED   | EARLY_DATA                |
| tls1_3.h, 53             | tls1_3.h, 47              |
| COOKIE                   | early_data                |
| tls1_3.h, 47             | ee_status, 16             |
| createClientCertVerifier | ECC_SS                    |
| tls_keys_calc.h, 100     | tls1_3.h, <mark>35</mark> |
| createCryptoContext      | ECCX08.h, 153             |
| tls_keys_calc.h, 95      | ECCX08Class, 15           |
| createRecvCryptoContext  | ECDSA_SECP256R1_SHA256    |
| tls_keys_calc.h, 96      | tls1_3.h, 43              |
| createSendCryptoContext  | ECDSA_SECP256R1_SHA384    |
| tls_keys_calc.h, 96      | tls1_3.h, 43              |
| crypto, 13               | ECDSA_SECP384R1_SHA384    |
| active, 13               | tls1_3.h, 43              |
| IV, 14                   | ecdsa_sig_decode          |
| iv, 14                   | tls_x509.h, 143           |
| K, 14                    | ecdsa_sig_encode          |
| k, 13                    | tls_x509.h, 143           |
| record, 14               | ED25519                   |
| suite, 14                | tls1_3.h, 44              |
| taglen, 14               | ee_status, 15             |
| CRYPTO_SETTING           | alpn, 16                  |
| tls1_3.h, 36             | early_data, 16            |
| CTS                      | max_frag_length, 16       |
| TLS_session, 25          | server_name, 16           |
| cts                      | EMPTY_CERT_CHAIN          |
| TLS_session, 25          | tls1_3.h, <mark>52</mark> |
| CTX                      | ENCRYPTED_EXTENSIONS      |
| TLS_session, 26          | tls1_3.h, 49              |
| ctx                      | END_OF_EARLY_DATA         |
| TLS_session, 26          | tls1_3.h, <mark>50</mark> |
| curve                    | endTicketContext          |
| pktype, 18               | tls_tickets.h, 138        |
| المراهم المراها          | err                       |
| DECODE_ERROR             | ret, 19                   |
|                          |                           |

| ERROR ALERT RECEIVED  | tls1_3.h, 35  |
|---|---|
| tls1 3.h, 52  | HW 2  |
| 1131_0.11, 32   | <del></del>   |
| f   | tls1_3.h, 35  |
| favourite_group   | HYB_SS  |
| ticket, 22  | tls1_3.h, <mark>35</mark>   |
| TLS_session, 24   | HYBRID  |
| FINISHED  | tls1_3.h, 34  |
| tls1_3.h, 49  |   |
|   | HYBRID_KX   |
| FORBIDDEN_EXTENSION   | tls1_3.h, 43  |
| tls1_3.h, 52  |   |
|   | IBUFF   |
| getByte   | TLS session, 26   |
| tls_sockets.h, 135  | ibuff   |
| getBytes  | TLS session, 26   |
| tls sockets.h, 133  | id  |
| getCertificateRequest   | =   |
| <del>-</del>  | TLS_session, 24   |
| tls_client_recv.h, 76   | ID_MISMATCH   |
| getCheckServerCertificateChain  | tls1_3.h, 50  |
| tls_client_recv.h, 76   | ILLEGAL_PARAMETER   |
| getClientPrivateKeyandCertChain   | tls1_3.h, 53  |
| tls_cert_chain.h, 66  |   |
|   | incrementCryptoContext  |
| getInt16  | tls_keys_calc.h, 95   |
| tls_sockets.h, 134  | initCryptoContext   |
| getInt24  | tls_keys_calc.h, 94   |
| tls_sockets.h, 134  | initTicketContext   |
| getlPaddress  | tls_tickets.h, 138  |
| tls_sockets.h, 132  |   |
|   | initTranscriptHash  |
| getOctad  | tls_keys_calc.h, <mark>92</mark>  |
| tls_sockets.h, 135  | IO_APPLICATION  |
| getServerCertVerify   | tls1_3.h, 33  |
| tls_client_recv.h, 74   | IO DEBUG  |
| getServerEncryptedExtensions  | tls1_3.h, 34  |
| tls_client_recv.h, 74   |   |
|   | IO_NONE   |
| getServerFinished   | tls1_3.h, <mark>33</mark>   |
| tls_client_recv.h, 75   | IO_PROTOCOL   |
| getServerHello  | tls1_3.h, 33  |
| tls_client_recv.h, 75   | IO WIRE   |
| getServerRecord   | tls1 3.h, 34  |
| tls_client_recv.h, 71   | <del>-</del> :  |
|   | IV  |
| getSigRequirements  | crypto, 14  |
| tls_certs.h, 67   | iv  |
|   | crypto, 14  |
| HANDSHAKE_FAILURE   |   |
| tls1_3.h, 55  | K   |
| HANDSHAKE RETRY   | crypto, 14  |
| tls1_3.h, <del>50</del>   | k   |
| hash  |   |
|   |   |
| pktype, 18  | crypto, 13  |
|   | Crypto, 13<br>K_recv  |
| hostname  |   |
| hostname TLS_session, 24  | K_recv<br>TLS_session, 24   |
|   | K_recv<br>TLS_session, 24<br>K_send   |
| TLS_session, 24<br>HS   | K_recv<br>TLS_session, 24<br>K_send<br>TLS_session, 24  |
| TLS_session, 24 HS TLS_session, 25  | K_recv<br>TLS_session, 24<br>K_send<br>TLS_session, 24<br>KEY_SHARE   |
| TLS_session, 24 HS TLS_session, 25 hs   | K_recv<br>TLS_session, 24<br>K_send<br>TLS_session, 24<br>KEY_SHARE<br>tls1_3.h, 46   |
| TLS_session, 24 HS TLS_session, 25 hs TLS_session, 25                                       | K_recv<br>TLS_session, 24<br>K_send<br>TLS_session, 24<br>KEY_SHARE<br>tls1_3.h, 46<br>KEY_UPDATE   |
| TLS_session, 24 HS TLS_session, 25 hs TLS_session, 25 HSHAKE                                | K_recv<br>TLS_session, 24<br>K_send<br>TLS_session, 24<br>KEY_SHARE<br>tls1_3.h, 46   |
| TLS_session, 24 HS TLS_session, 25 hs TLS_session, 25                                       | K_recv<br>TLS_session, 24<br>K_send<br>TLS_session, 24<br>KEY_SHARE<br>tls1_3.h, 46<br>KEY_UPDATE   |
| TLS_session, 24 HS TLS_session, 25 hs TLS_session, 25 HSHAKE                                | K_recv<br>TLS_session, 24<br>K_send<br>TLS_session, 24<br>KEY_SHARE<br>tls1_3.h, 46<br>KEY_UPDATE<br>tls1_3.h, 50<br>KYBER768                 |
| TLS_session, 24 HS TLS_session, 25 hs TLS_session, 25 HSHAKE tls1_3.h, 48 htype             | K_recv<br>TLS_session, 24<br>K_send<br>TLS_session, 24<br>KEY_SHARE<br>tls1_3.h, 46<br>KEY_UPDATE<br>tls1_3.h, 50                             |
| TLS_session, 24 HS TLS_session, 25 hs TLS_session, 25 HSHAKE tls1_3.h, 48 htype unihash, 27 | K_recv<br>TLS_session, 24<br>K_send<br>TLS_session, 24<br>KEY_SHARE<br>tls1_3.h, 46<br>KEY_UPDATE<br>tls1_3.h, 50<br>KYBER768<br>tls1_3.h, 43 |
| TLS_session, 24 HS TLS_session, 25 hs TLS_session, 25 HSHAKE tls1_3.h, 48 htype             | K_recv<br>TLS_session, 24<br>K_send<br>TLS_session, 24<br>KEY_SHARE<br>tls1_3.h, 46<br>KEY_UPDATE<br>tls1_3.h, 50<br>KYBER768                 |

|                          | NOOFDT                |
|--------------------------|-----------------------|
| octad, 17                | NOCERT                |
| lifetime                 | tls1_3.h, 34<br>NONCE |
| ticket, 22               |                       |
| log<br>tls logger.h, 102 | ticket, 21            |
| LOG OUTPUT TRUNCATION    | nonce ticket, 21      |
| <u> </u>                 | NOT EXPECTED          |
| tls1_3.h, 55<br>logAlert | tls1_3.h, 51          |
| tls logger.h, 105        | NOT TLS1 3            |
| logCert                  | tls1_3.h, 50          |
| tls_logger.h, 104        | 1131_3.11, 30         |
| logCertDetails           | OBUFF                 |
| tls_logger.h, 104        | TLS_session, 26       |
| logCipherSuite           | obuff                 |
| tls_logger.h, 105        | TLS_session, 26       |
| logEncExt                | OCT_append_byte       |
| tls_logger.h, 103        | tls_octads.h, 110     |
| logKeyExchange           | OCT_append_bytes      |
| tls_logger.h, 105        | tls_octads.h, 110     |
| logServerHello           | OCT_append_int        |
| tls logger.h, 103        | tls_octads.h, 108     |
| logServerResponse        | OCT_append_octad      |
| tls_logger.h, 105        | tls_octads.h, 108     |
| logSigAlg                | OCT_append_string     |
| tls_logger.h, 106        | tls_octads.h, 110     |
| logTicket                | OCT_compare           |
| tls_logger.h, 103        | tls_octads.h, 108     |
| <u>-</u>                 | OCT_copy              |
| max                      | tls_octads.h, 112     |
| octad, 17                | OCT_from_base64       |
| max_early_data           | tls_octads.h, 111     |
| ticket, 21               | OCT_from_hex          |
| MAX EXCEEDED             | tls_octads.h, 109     |
| tls1_3.h, 52             | OCT kill              |
| MAX_FRAG_LENGTH          | tls_octads.h, 109     |
| tls1_3.h, 47             | OCT_output_base64     |
| max_frag_length          | tls_octads.h, 113     |
| ee_status, 16            | OCT_output_hex        |
| max_record               | tls_octads.h, 112     |
| TLS_session, 23          | OCT_output_string     |
| MEM_OVERFLOW             | tls_octads.h, 112     |
| tls1_3.h, 52             | OCT_reverse           |
| MESSAGE_HASH             | tls_octads.h, 111     |
| tls1_3.h, 50             | OCT_shift_left        |
| millis                   | tls_octads.h, 109     |
| tls_octads.h, 108        | OCT_truncate          |
| MISSING_EXTENSION        | tls_octads.h, 111     |
| tls1_3.h, 55             | octad, 16             |
| MISSING_EXTENSIONS       | len, 17               |
| tls1_3.h, 53             | max, 17               |
| MISSING_REQUEST_CONTEXT  | val, 17               |
| tls1_3.h, 51             | origin                |
| mycert                   | ticket, 22            |
| tls_certs.h, 68          | DADDING               |
| myprintf                 | PADDING               |
| tls_logger.h, 102        | tls1_3.h, 47          |
| myprivate                | parsebytes            |
| tls_certs.h, 68          | tls_client_recv.h, 70 |
|                          | parsebytesorPull      |

| tls_client_recv.h, 72 | crypto, 14                   |
|-----------------------|------------------------------|
| parseInt              | RECORD_OVERFLOW              |
| tls_client_recv.h, 70 | tls1_3.h, <mark>54</mark>    |
| parseIntorPull        | RECORD_SIZE_LIMIT            |
| tls_client_recv.h, 72 | tls1_3.h, 48                 |
| parseoctad            | recoverPSK                   |
| tls_client_recv.h, 69 | tls_keys_calc.h, 96          |
| parseoctadorPull      | ret, 18                      |
| tls_client_recv.h, 72 | err, 19                      |
| parseoctadorPullptrX  | val, 19                      |
| tls_client_recv.h, 73 | rewindIO                     |
| parseoctadptr         | tls_keys_calc.h, 93          |
| tls_client_recv.h, 71 | RMS                          |
| parseTicket           | TLS_session, 25              |
| tls_tickets.h, 137    | rms                          |
| PFS_BLS12381          | TLS_session, 25              |
| tls_bfibe.h, 63       | RSA_PKCS1_SHA256             |
| PGS BLS12381          | tls1_3.h, 44                 |
| tls_bfibe.h, 63       | RSA_PKCS1_SHA384             |
| pktype, 17            | tls1 3.h, 44                 |
| curve, 18             | RSA PKCS1 SHA512             |
| hash, 18              | tls1_3.h, 44                 |
| type, 18              | RSA_PSS_RSAE_SHA256          |
| POST HANDSHAKE AUTH   | <br>tls1_3.h, 43             |
| tls1_3.h, 46          | RSA_PSS_RSAE_SHA384          |
| POST_HS_AUTH          | tls1_3.h, 43                 |
| tls1_3.h, 35          | RSA_PSS_RSAE_SHA512          |
| POST QUANTUM          | tls1 3.h, 44                 |
| tls1_3.h, 34          | RSA SS                       |
| PQIBE CCA DECRYPT     | tls1 3.h, 34                 |
| tls_pqibe.h, 115      | runningHash                  |
| PQIBE_CCA_ENCRYPT     | tls_keys_calc.h, 93          |
| tls_pqibe.h, 114      | runningHashIO                |
| PRESHARED KEY         | tls_keys_calc.h, 93          |
| tls1_3.h, 47          | runningHashIOrewind          |
| PROTOCOL VERSION      | tls keys calc.h, 93          |
| tls1_3.h, 54          | runningSyntheticHash         |
| PSK                   | tls_keys_calc.h, 94          |
| ticket, 21            | <u></u>                      |
| psk                   | SAL_aeadDecrypt              |
| ticket, 21            | tls_sal.h, 128               |
| PSK IBE               | SAL_aeadEncrypt              |
| tls1_3.h, 56          | tls_sal.h, 127               |
| PSK KEY               | SAL_aeadKeylen               |
| tls1_3.h, 56          | tls_sal.h, 123               |
| PSK MODE              | SAL_aeadTaglen               |
| tls1_3.h, 47          | tls_sal.h, 124               |
| PSK NOT               | SAL_ciphers                  |
| tls1_3.h, 56          | tls_sal.h, 121               |
| PSKOK                 | SAL_endLib                   |
| tls1_3.h, 45          | tls_sal.h, 122               |
| PSKWECDHE             | SAL_generateKeyPair          |
| tls1_3.h, 45          | tls_sal.h, 128               |
| ptr                   | SAL_generateSharedSecret     |
| TLS_session, 26       | -                            |
|                       | tls_sal.h, 129               |
| 120_0001011, 20       | tls_sal.h, 129<br>SAL_groups |
| RAW_PUBLIC_KEY        |                              |
|                       | SAL_groups                   |
| RAW_PUBLIC_KEY        | SAL_groups<br>tls_sal.h, 121 |

| SAL_hashLen                      | sendEndOfEarlyData        |
|----------------------------------|---------------------------|
| tls_sal.h, 123                   | tls_client_send.h, 88     |
| SAL_hashNull                     | sendKeyUpdate             |
| tls_sal.h, 126                   | tls_client_send.h, 87     |
| SAL_hashOutput                   | sendLen                   |
| tls_sal.h, 127                   | tls_sockets.h, 133        |
| SAL_hashProcessArray             | sendOctad                 |
| tls_sal.h, 126                   | tls_sockets.h, 133        |
| SAL_hashType                     | SERVER_CERT_TYPE          |
| tls_sal.h, 123                   | tls1_3.h, 48              |
| SAL_hkdfExpand                   | server_cert_type          |
| tls_sal.h, 125                   | TLS_session, 24           |
| SAL_hkdfExtract                  | SERVER_HELLO              |
| tls_sal.h, 125                   | tls1_3.h, 49              |
| SAL_hmac                         | SERVER_NAME               |
| tls_sal.h, 125                   | tls1_3.h, 46              |
| SAL_initLib                      | server_name               |
| tls_sal.h, 122                   | ee_status, 16             |
| SAL_name                         | setclientsock             |
| tls_sal.h, 120                   | tls_sockets.h, 132        |
| SAL_randomByte                   | SIG_ALGS                  |
| tls_sal.h, 124                   | tls1_3.h, 46              |
| SAL_randomOctad                  | SIG_ALGS_CERT             |
| tls_sal.h, 124                   | tls1_3.h, 46              |
| SAL_sigCerts                     | sign16                    |
| tls_sal.h, 122                   | tls1_3.h, 57              |
| SAL_sigs                         | sign32                    |
| tls_sal.h, 121                   | tls1_3.h, 57              |
| SAL_tlsSignature                 | sign64                    |
| tls_sal.h, 130                   | tls1_3.h, 57              |
| SAL_tlsSignatureVerify           | sign8                     |
| tls_sal.h, 129                   | tls1_3.h, 57              |
| SECP256R1                        | Socket, 19                |
| tls1_3.h, 42                     | sockptr                   |
| SECP384R1                        | TLS_session, 23           |
| tls1_3.h, 42                     | state                     |
| SECP521R1                        | unihash, 27               |
| tls1_3.h, 42                     | status                    |
| seeWhatsNext                     | TLS_session, 23           |
| tls_client_recv.h, 74            | STS                       |
| SELF_SIGNED_CERT<br>tls1_3.h, 52 | TLS_session, 25           |
| sendAlert                        |                           |
| tls_client_send.h, 87            | TLS_session, 25 suite     |
| sendBinder                       | crypto, 14                |
| tls_client_send.h, 86            | SUPPORTED_GROUPS          |
| sendCCCS                         | tls1_3.h, 46              |
| tls_client_send.h, 79            | 1151_3.11, 40             |
| sendClientCertificateChain       | Т                         |
| tls_client_send.h, 88            | TLS_session, 27           |
| sendClientCertVerify             | taglen                    |
| tls_client_send.h, 88            | crypto, 14                |
| sendClientFinish                 | THIS_YEAR                 |
| tls_client_send.h, 87            | tls1_3.h, <mark>35</mark> |
| sendClientHello                  | TICK                      |
| tls_client_send.h, 86            | ticket, 21                |
| sendClientMessage                | tick                      |
| tls client send.h, 85            | ticket, 21                |
|                                  | TICKET                    |
|                                  |                           |

| tls1_3.h, 50               | BAD_CERT_CHAIN, 50         |
|----------------------------|----------------------------|
| ticket, 20                 | BAD_CERTIFICATE, 54        |
| age_obfuscator, 21         | BAD HANDSHAKE, 53          |
| birth, 22                  | BAD HELLO, 51              |
| cipher_suite, 22           | BAD_MESSAGE, 53            |
| favourite group, 22        | BAD RECORD, 51             |
|                            |                            |
| lifetime, 22               | BAD_RECORD_MAC, 55         |
| max_early_data, 21         | BAD_REQUEST_UPDATE, 53     |
| NONCE, 21                  | BAD_TICKET, 51             |
| nonce, 21                  | byte, <del>57</del>        |
| origin, 22                 | CA_NOT_FOUND, 52           |
| PSK, 21                    | CERT_OUTOFDATE, 52         |
| psk, 21                    | CERT_REQUEST, 49           |
| TICK, 21                   | CERT VERIFY, 49            |
|                            |                            |
| tick, 21                   | CERT_VERIFY_FAIL, 53       |
| valid, 20                  | CERTIFICATE, 49            |
| ticket_still_good          | CERTIFICATE_EXPIRED, 54    |
| tls_tickets.h, 138         | CHANGE_CIPHER, 48          |
| TIMED_OUT                  | CLIENT_CERT, 36            |
| tls1_3.h, 48               | CLIENT CERT TYPE, 48       |
| TINY ECC                   | CLIENT HELLO, 49           |
| tls1_3.h, 34               | CLOSE_NOTIFY, 55           |
| TLS13_clean                | CLOSURE_ALERT_RECEIVED, 53 |
|                            |                            |
| tls_protocol.h, 118        | COOKIE, 47                 |
| TLS13_connect              | CRYPTO_SETTING, 36         |
| tls_protocol.h, 117        | DECODE_ERROR, 54           |
| TLS13_CONNECTED            | DECRYPT_ERROR, 54          |
| tls1_3.h, 55               | DILITHIUM2, 44             |
| TLS13_DISCONNECTED         | DILITHIUM2_P256, 44        |
| tls1_3.h, 55               | DILITHIUM3, 44             |
| TLS13 end                  | DLT_SS, 35                 |
| tls_protocol.h, 117        | EARLY_DATA, 47             |
| <del></del>                |                            |
| TLS13_HANDSHAKING          | ECC_SS, 35                 |
| tls1_3.h, 55               | ECDSA_SECP256R1_SHA256, 43 |
| TLS13_recv                 | ECDSA_SECP256R1_SHA384, 43 |
| tls_protocol.h, 118        | ECDSA_SECP384R1_SHA384, 43 |
| TLS13_send                 | ED25519, 44                |
| tls_protocol.h, 118        | EMPTY_CERT_CHAIN, 52       |
| TLS13 start                | ENCRYPTED EXTENSIONS, 49   |
| tls protocol.h, 116        | END_OF_EARLY_DATA, 50      |
| TLS13_stop                 | ERROR_ALERT_RECEIVED, 52   |
| tls_protocol.h, 117        | FINISHED, 49               |
| TLS13_UPDATE_NOT_REQUESTED | FORBIDDEN_EXTENSION, 52    |
|                            |                            |
| tls1_3.h, 46               | HANDSHAKE_FAILURE, 55      |
| TLS13_UPDATE_REQUESTED     | HANDSHAKE_RETRY, 50        |
| tls1_3.h, 46               | HSHAKE, 48                 |
| TLS1_0                     | HW_1, 35                   |
| tls1_3.h, 45               | HW_2, 35                   |
| TLS1 2                     | HYB SS, 35                 |
| <br>tls1_3.h, 45           | HYBRID, 34                 |
| TLS1 3                     | HYBRID_KX, 43              |
| tls1_3.h, 45               | ID MISMATCH, 50            |
|                            | ILLEGAL PARAMETER, 53      |
| tls1_3.h, 29, 58           | <del>-</del>               |
| ALERT, 48                  | IO_APPLICATION, 33         |
| ALLOW_SELF_SIGNED, 36      | IO_DEBUG, 34               |
| APP_PROTOCOL, 47           | IO_NONE, 33                |
| APPLICATION, 48            | IO_PROTOCOL, 33            |
| AUTHENTICATION_FAILURE, 51 | IO_WIRE, 34                |
|                            |                            |

| KEY_SHARE, 46                  | TLS13_UPDATE_REQUESTED, 46       |
|--------------------------------|----------------------------------|
| KEY_UPDATE, 50                 | TLS1_0, 45                       |
| KYBER768, 43                   | TLS1_2, 45                       |
| LOG_OUTPUT_TRUNCATION, 55      | TLS1 3, 45                       |
| MAX EXCEEDED, 52               | TLS AES 128 CCM 8 SHA256, 42     |
| MAX_FRAG_LENGTH, 47            | TLS_AES_128_CCM_SHA256, 42       |
| MEM OVERFLOW, 52               | TLS_AES_128_GCM_SHA256, 41       |
| MESSAGE HASH, 50               | TLS AES 256 GCM SHA384, 42       |
| MISSING EXTENSION, 55          | TLS APPLICATION PROTOCOL, 36     |
| MISSING EXTENSIONS, 53         | TLS CHACHA20 POLY1305 SHA256, 42 |
|                                |                                  |
| MISSING_REQUEST_CONTEXT, 51    | TLS_EARLY_DATA_ACCEPTED, 56      |
| NOCERT, 34                     | TLS_EXTERNAL_PSK, 45             |
| NOT_EXPECTED, 51               | TLS_FAILURE, 56                  |
| NOT_TLS1_3, 50                 | TLS_FULL_HANDSHAKE, 45           |
| PADDING, 47                    | TLS_MAX_CERT_B64, 38             |
| POST_HANDSHAKE_AUTH, 46        | TLS_MAX_CERT_SIZE, 38            |
| POST_HS_AUTH, 35               | TLS_MAX_CIPHER_FRAG, 38          |
| POST_QUANTUM, 34               | TLS_MAX_CIPHER_SUITES, 41        |
| PRESHARED_KEY, 47              | TLS_MAX_CLIENT_CHAIN_LEN, 39     |
| PROTOCOL_VERSION, 54           | TLS_MAX_CLIENT_CHAIN_SIZE, 40    |
| PSK_IBE, 56                    | TLS_MAX_COOKIE, 40               |
| PSK_KEY, 56                    | TLS_MAX_ECC_FIELD, 40            |
| PSK_MODE, 47                   | TLS_MAX_EXT_LABEL, 37            |
| PSK NOT, 56                    | TLS_MAX_EXTENSIONS, 40           |
| PSKOK, 45                      | TLS MAX FRAG, 37                 |
| PSKWECDHE, 45                  | TLS MAX HASH, 37                 |
| RAW_PUBLIC_KEY, 57             | TLS MAX HASH STATE, 37           |
| RECORD OVERFLOW, 54            | TLS MAX HELLO, 38                |
| RECORD SIZE LIMIT, 48          | TLS MAX IBUFF SIZE, 38           |
| RSA PKCS1 SHA256, 44           | TLS MAX IV SIZE, 40              |
| RSA_PKCS1_SHA384, 44           | TLS MAX KEX CIPHERTEXT SIZE, 39  |
| RSA_PKCS1_SHA512, 44           | TLS_MAX_KEX_PUB_KEY_SIZE, 39     |
| RSA_PSS_RSAE_SHA256, 43        | TLS MAX KEX SECRET KEY SIZE, 39  |
|                                | TLS MAX KEY, 37                  |
| RSA_PSS_RSAE_SHA384, 43        |                                  |
| RSA_PSS_RSAE_SHA512, 44        | TLS_MAX_OBUFF_SIZE, 41           |
| RSA_SS, 34                     | TLS_MAX_OUTPUT_RECORD_SIZE, 41   |
| SECP256R1, 42                  | TLS_MAX_PLAIN_FRAG, 38           |
| SECP384R1, 42                  | TLS_MAX_PSK_MODES, 41            |
| SECP521R1, 42                  | TLS_MAX_SERVER_CHAIN_LEN, 39     |
| SELF_SIGNED_CERT, 52           | TLS_MAX_SERVER_CHAIN_SIZE, 39    |
| SERVER_CERT_TYPE, 48           | TLS_MAX_SERVER_NAME, 41          |
| SERVER_HELLO, 49               | TLS_MAX_SHARED_SECRET_SIZE, 40   |
| SERVER_NAME, 46                | TLS_MAX_SIG_PUB_KEY_SIZE, 38     |
| SIG_ALGS, 46                   | TLS_MAX_SIG_SECRET_KEY_SIZE, 39  |
| SIG_ALGS_CERT, 46              | TLS_MAX_SIGNATURE_SIZE, 39       |
| sign16, 57                     | TLS_MAX_SUPPORTED_GROUPS, 41     |
| sign32, 57                     | TLS_MAX_SUPPORTED_SIGS, 41       |
| sign64, 57                     | TLS_MAX_TAG_SIZE, 40             |
| sign8, 57                      | TLS_MAX_TICKET_SIZE, 40          |
| SUPPORTED_GROUPS, 46           | TLS_RESUMPTION_REQUIRED, 56      |
| THIS_YEAR, 35                  | TLS_SHA256_T, 36                 |
| TICKET, 50                     | TLS_SHA384_T, 36                 |
| TIMED_OUT, 48                  | TLS_SHA512_T, 37                 |
| TINY_ECC, 34                   | TLS_SUCCESS, 56                  |
| TLS13_CONNECTED, 55            | TLS_VER, 47                      |
| TLS13_DISCONNECTED, 55         | TLS_X509_MAX_FIELD, 37           |
| TLS13_HANDSHAKING, 55          | TRY EARLY DATA, 36               |
| TLS13_UPDATE_NOT_REQUESTED, 46 | TYPICAL, 34                      |
| ,,,,                           | ,                                |

| UNEXPECTED_MESSAGE, 53              | addEarlyDataExt, 84            |
|-------------------------------------|--------------------------------|
| UNKNOWN_CA, 54                      | addKeyShareExt, 82             |
| UNRECOGNIZED_EXT, 51                | addMFLExt, 82                  |
| unsign32, 57                        | addPadding, 84                 |
| unsign64, 58                        | addPostHSAuth, 84              |
| UNSUPPORTED_EXTENSION, 54           | addPreSharedKeyExt, 79         |
| VERBOSITY, 35                       | addPSKModesExt, 83             |
| WRONG_MESSAGE, 51                   | addRSLExt, 83                  |
| X25519, 42                          | addServerNameExt, 80           |
| X448, 43                            | addServerRawPublicKey, 80      |
| X509 CERT, 56                       | addSigAlgsCertExt, 81          |
| TLS_AES_128_CCM_8_SHA256            | addSigAlgsExt, 81              |
| tls1 3.h, 42                        | addSupportedGroupsExt, 80      |
| TLS_AES_128_CCM_SHA256              | addVersionExt, 83              |
| tls1 3.h, 42                        | alert_from_cause, 90           |
| TLS_AES_128_GCM_SHA256              | cipherSuites, 85               |
|                                     | •                              |
| tls1_3.h, 41                        | clientRandom, 85               |
| TLS_AES_256_GCM_SHA384              | sendAlert, 87                  |
| tls1_3.h, 42                        | sendBinder, 86                 |
| TLS_APPLICATION_PROTOCOL            | sendCCCS, 79                   |
| tls1_3.h, 36                        | sendClientCertificateChain, 88 |
| tls_bfibe.h, 63, 64                 | sendClientCertVerify, 88       |
| BFIBE_CCA_DECRYPT, 64               | sendClientFinish, 87           |
| BFIBE_CCA_ENCRYPT, 63               | sendClientHello, 86            |
| PFS_BLS12381, 63                    | sendClientMessage, 85          |
| PGS_BLS12381, 63                    | sendEndOfEarlyData, 88         |
| tls_cert_chain.h, 65, 66            | sendKeyUpdate, 87              |
| checkServerCertChain, 65            | TLS_EARLY_DATA_ACCEPTED        |
| getClientPrivateKeyandCertChain, 66 | tls1_3.h, <mark>56</mark>      |
| tls_certs.h, 67, 68                 | TLS_EXTERNAL_PSK               |
| cacerts, 68                         | tls1_3.h, 45                   |
| getSigRequirements, 67              | TLS FAILURE                    |
| mycert, 68                          | tls1_3.h, <u>56</u>            |
| myprivate, 68                       | TLS_FULL_HANDSHAKE             |
| TLS_CHACHA20_POLY1305_SHA256        | tls1 3.h, 45                   |
| tls1_3.h, 42                        | tls_keys_calc.h, 91, 100       |
| tls_client_recv.h, 68, 77           | checkServerCertVerifier, 99    |
| badResponse, 73                     | checkVeriferData, 99           |
| getCertificateRequest, 76           | createClientCertVerifier, 100  |
| getCheckServerCertificateChain, 76  | createCryptoContext, 95        |
| getServerCertVerify, 74             | createRecvCryptoContext, 96    |
| getServerEncryptedExtensions, 74    | createSendCryptoContext, 96    |
| getServerFinished, 75               | deriveApplicationSecrets, 98   |
| getServerHello, 75                  | deriveApplicationSecrets, 90   |
| •                                   |                                |
| getServerRecord, 71                 | deriveHandshakeSecrets, 97     |
| parsebytes, 70                      | deriveLaterSecrets, 97         |
| parsebytesorPull, 72                | deriveUpdatedKeys, 98          |
| parseInt, 70                        | deriveVeriferData, 99          |
| parseIntorPull, 72                  | incrementCryptoContext, 95     |
| parseoctad, 69                      | initCryptoContext, 94          |
| parseoctadorPull, 72                | initTranscriptHash, 92         |
| parseoctadorPullptrX, 73            | recoverPSK, 96                 |
| parseoctadptr, 71                   | rewindIO, 93                   |
| seeWhatsNext, 74                    | runningHash, 93                |
| tls_client_send.h, 77, 90           | runningHashIO, 93              |
| addALPNExt, 82                      | runningHashIOrewind, 93        |
| addClientRawPublicKey, 81           | runningSyntheticHash, 94       |
| addCookieExt, 84                    | transcriptHash, 94             |
|                                     |                                |

| updateCryptoContext, 95     | TLS MAX PLAIN FRAG          |
|-----------------------------|-----------------------------|
| tls_logger.h, 101, 106      | tls1_3.h, 38                |
| log, 102                    | TLS_MAX_PSK_MODES           |
| logAlert, 105               | tls1_3.h, 41                |
| logCert, 104                | TLS_MAX_SERVER_CHAIN_LEN    |
| logCertDetails, 104         | tls1 3.h, 39                |
| logCipherSuite, 105         | TLS_MAX_SERVER_CHAIN_SIZE   |
| logEncExt, 103              | tls1_3.h, 39                |
| logKeyExchange, 105         | TLS_MAX_SERVER_NAME         |
| logServerHello, 103         | tls1_3.h, 41                |
| logServerResponse, 105      | TLS_MAX_SHARED_SECRET_SIZE  |
| logSigAlg, 106              | tls1_3.h, 40                |
| logTicket, 103              | TLS_MAX_SIG_PUB_KEY_SIZE    |
| myprintf, 102               | tls1_3.h, 38                |
| TLS_MAX_CERT_B64            | TLS_MAX_SIG_SECRET_KEY_SIZE |
| tls1_3.h, 38                | tls1_3.h, 39                |
| TLS_MAX_CERT_SIZE           | TLS_MAX_SIGNATURE_SIZE      |
| tls1_3.h, 38                | tls1 3.h, 39                |
| TLS_MAX_CIPHER_FRAG         | TLS MAX SUPPORTED GROUPS    |
| tls1_3.h, 38                | tls1_3.h, 41                |
| TLS MAX CIPHER SUITES       | TLS_MAX_SUPPORTED_SIGS      |
| tls1_3.h, 41                | tls1_3.h, 41                |
| TLS_MAX_CLIENT_CHAIN_LEN    | TLS_MAX_TAG_SIZE            |
| tls1_3.h, 39                | tls1_3.h, 40                |
|                             |                             |
| TLS_MAX_CLIENT_CHAIN_SIZE   | TLS_MAX_TICKET_SIZE         |
| tls1_3.h, 40                | tls1_3.h, 40                |
| TLS_MAX_COOKIE              | tls_octads.h, 106, 113      |
| tls1_3.h, 40                | millis, 108                 |
| TLS_MAX_ECC_FIELD           | OCT_append_byte, 110        |
| tls1_3.h, 40                | OCT_append_bytes, 110       |
| TLS_MAX_EXT_LABEL           | OCT_append_int, 108         |
| tls1_3.h, 37                | OCT_append_octad, 108       |
| TLS_MAX_EXTENSIONS          | OCT_append_string, 110      |
| tls1_3.h, 40                | OCT_compare, 108            |
| TLS_MAX_FRAG                | OCT_copy, 112               |
| tls1_3.h, 37                | OCT_from_base64, 111        |
| TLS_MAX_HASH                | OCT_from_hex, 109           |
| tls1_3.h, 37                | OCT_kill, 109               |
| TLS_MAX_HASH_STATE          | OCT_output_base64, 113      |
| tls1_3.h, 37                | OCT_output_hex, 112         |
| TLS_MAX_HELLO               | OCT_output_string, 112      |
| tls1_3.h, 38                | OCT_reverse, 111            |
| TLS_MAX_IBUFF_SIZE          | OCT_shift_left, 109         |
| tls1_3.h, 38                | OCT_truncate, 111           |
| TLS_MAX_IV_SIZE             | tls_pqibe.h, 114, 115       |
| tls1_3.h, 40                | PQIBE_CCA_DECRYPT, 115      |
| TLS_MAX_KEX_CIPHERTEXT_SIZE | PQIBE_CCA_ENCRYPT, 114      |
| tls1_3.h, 39                | tls_protocol.h, 115, 119    |
| TLS_MAX_KEX_PUB_KEY_SIZE    | TLS13_clean, 118            |
| tls1_3.h, 39                | TLS13_connect, 117          |
| TLS_MAX_KEX_SECRET_KEY_SIZE | TLS13_end, 117              |
| tls1_3.h, <mark>39</mark>   | TLS13_recv, 118             |
| TLS_MAX_KEY                 | TLS13_send, 118             |
| tls1_3.h, 37                | TLS13_start, 116            |
| TLS_MAX_OBUFF_SIZE          | TLS13_stop, 117             |
| tls1_3.h, 41                | TLS_RESUMPTION_REQUIRED     |
| TLS_MAX_OUTPUT_RECORD_SIZE  | tls1_3.h, 56                |
| tls1 3.h. 41                | tls sal.h. 119, 130         |

|      | SAL_aeadDecrypt, 128          | tls1_3.h, <mark>36</mark>      |
|------|-------------------------------|--------------------------------|
|      | SAL_aeadEncrypt, 127          | TLS_SHA512_T                   |
|      | SAL_aeadKeylen, 123           | tls1_3.h, <mark>37</mark>      |
|      | SAL_aeadTaglen, 124           | tls_sockets.h, 131, 135        |
|      | SAL_ciphers, 121              | getByte, 135                   |
|      | SAL_endLib, 122               | getBytes, 133                  |
|      | SAL_generateKeyPair, 128      | getInt16, 134                  |
|      | SAL_generateSharedSecret, 129 | getInt24, 134                  |
|      | SAL_groups, 121               | getlPaddress, 132              |
|      | SAL_hashInit, 126             | getOctad, 135                  |
|      | SAL_hashLen, 123              | sendLen, 133                   |
|      | SAL hashNull, 126             | sendOctad, 133                 |
|      | SAL_hashOutput, 127           | setclientsock, 132             |
|      | SAL_hashProcessArray, 126     | TLS SUCCESS                    |
|      | SAL_hashType, 123             | tls1_3.h, 56                   |
|      | SAL_hkdfExpand, 125           | tls_tickets.h, 137, 139        |
|      |                               |                                |
|      | SAL_hkdfExtract, 125          | endTicketContext, 138          |
|      | SAL_hmac, 125                 | initTicketContext, 138         |
|      | SAL_initLib, 122              | parseTicket, 137               |
|      | SAL_name, 120                 | ticket_still_good, 138         |
|      | SAL_randomByte, 124           | TLS_VER                        |
|      | SAL_randomOctad, 124          | tls1_3.h, 47                   |
|      | SAL_sigCerts, 122             | tls_wifi.h, 139, 140           |
|      | SAL_sigs, 121                 | tls_x509.h, 140, 152           |
|      | SAL_tlsSignature, 130         | ecdsa_sig_decode, 143          |
|      | SAL_tlsSignatureVerify, 129   | ecdsa_sig_encode, 143          |
| TLS_ | _session, 22                  | USE_C25519, 143                |
|      | cipher_suite, 24              | USE_NIST256, 142               |
|      | client_cert_type, 24          | USE_NIST384, 143               |
|      | CTS, 25                       | USE_NIST521, 143               |
|      | cts, 25                       | X509_AN, 151                   |
|      | CTX, 26                       | X509_BC, 152                   |
|      | ctx, 26                       | X509_CN, 150                   |
|      | favourite_group, 24           | X509_ECC, 141                  |
|      | hostname, 24                  | X509_ECD, 142                  |
|      | HS, 25                        | X509_EN, 151                   |
|      | hs, 25                        | X509_extract_cert, 145         |
|      | IBUFF, 26                     | X509_extract_cert_sig, 145     |
|      | ibuff, 26                     | X509_extract_private_key, 145  |
|      | id, 24                        | X509_extract_public_key, 147   |
|      | K_recv, 24                    | X509_find_alt_name, 150        |
|      | K_send, 24                    |                                |
|      |                               | X509_find_entity_property, 148 |
|      | max_record, 23                | X509_find_expiry_date, 149     |
|      | OBUFF, 26                     | X509_find_extension, 150       |
|      | obuff, 26                     | X509_find_extensions, 149      |
|      | ptr, 26                       | X509_find_issuer, 147          |
|      | RMS, 25                       | X509_find_public_key, 146      |
|      | rms, 25                       | X509_find_start_date, 149      |
|      | server_cert_type, 24          | X509_find_subject, 147         |
|      | sockptr, 23                   | X509_find_validity, 147        |
|      | status, 23                    | X509_get_public_key, 146       |
|      | STS, 25                       | X509_H256, 142                 |
|      | sts, 25                       | X509_H384, 142                 |
|      | T, 27                         | X509_H512, 142                 |
|      | tlshash, 26                   | X509_HY, 142                   |
| TLS  | _SHA256_T                     | X509_KU, 151                   |
|      | tls1_3.h, <mark>36</mark>     | X509_LN, 151                   |
|      | _SHA384_T                     | X509_MN, 151                   |
|      |                               |                                |
|      |                               |                                |

| X509_ON, 151              | tls1_3.h, 43                  |
|---------------------------|-------------------------------|
| X509_PQ, 142              | X509_AN                       |
| X509_RSA, 142             | tls_x509.h, 151               |
| X509_self_signed, 148     | X509_BC                       |
| X509_SN, 151              | tls_x509.h, 1 <mark>52</mark> |
| X509_UN, 151              | X509_CERT                     |
| TLS_X509_MAX_FIELD        | tls1_3.h, <mark>56</mark>     |
| tls1_3.h, 37              | X509_CN                       |
| tlshash                   | tls_x509.h, 150               |
| TLS_session, 26           | X509_ECC                      |
| transcriptHash            | tls x509.h, 141               |
| tls_keys_calc.h, 94       | X509_ECD                      |
| TRY EARLY DATA            | tls_x509.h, 142               |
| tls1 3.h, 36              | X509_EN                       |
| type                      | tls_x509.h, 151               |
| pktype, 18                | X509_extract_cert             |
| TYPICAL                   | tls_x509.h, 145               |
| tls1_3.h, 34              | X509 extract cert sig         |
| 1101_0.11, 01             | tls x509.h, 145               |
| UNEXPECTED_MESSAGE        | X509 extract private key      |
| tls1 3.h, 53              | tls x509.h, 145               |
| unihash, 27               | X509_extract_public_key       |
| htype, 27                 | tls x509.h, 147               |
| state, 27                 | <del>-</del>                  |
| UNKNOWN CA                | X509_find_alt_name            |
| tls1_3.h, 54              | tls_x509.h, 150               |
| UNRECOGNIZED EXT          | X509_find_entity_property     |
| tls1_3.h, 51              | tls_x509.h, 148               |
|                           | X509_find_expiry_date         |
| unsign32                  | tls_x509.h, 149               |
| tls1_3.h, 57              | X509_find_extension           |
| unsign64                  | tls_x509.h, 150               |
| tls1_3.h, 58              | X509_find_extensions          |
| UNSUPPORTED_EXTENSION     | tls_x509.h, 149               |
| tls1_3.h, 54              | X509_find_issuer              |
| updateCryptoContext       | tls_x509.h, 147               |
| tls_keys_calc.h, 95       | X509_find_public_key          |
| USE_C25519                | tls_x509.h, 146               |
| tls_x509.h, 143           | X509_find_start_date          |
| USE_NIST256               | tls_x509.h, 149               |
| tls_x509.h, 142           | X509_find_subject             |
| USE_NIST384               | tls_x509.h, 147               |
| tls_x509.h, 143           | X509_find_validity            |
| USE_NIST521               | tls_x509.h, 147               |
| tls_x509.h, 143           | X509_get_public_key           |
|                           | tls x509.h, 146               |
| val                       | X509_H256                     |
| octad, 17                 | tls_x509.h, 142               |
| ret, 19                   | X509_H384                     |
| valid                     | _<br>tls_x509.h, 142          |
| ticket, 20                | X509_H512                     |
| VERBOSITY                 | _<br>tls_x509.h, 142          |
| tls1_3.h, <mark>35</mark> | X509_HY                       |
|                           | tls_x509.h, 142               |
| WRONG_MESSAGE             | X509_KU                       |
| tls1_3.h, 51              | tls_x509.h, 151               |
| Vocation                  | X509_LN                       |
| X25519                    | tls_x509.h, 151               |
| tls1_3.h, 42              | X509_MN                       |
| X448                      | 7.000_IVILY                   |
|                           |                               |

tls\_x509.h, 151
X509\_ON
 tls\_x509.h, 151
X509\_PQ
 tls\_x509.h, 142
X509\_RSA
 tls\_x509.h, 142
X509\_self\_signed
 tls\_x509.h, 148
X509\_SN
 tls\_x509.h, 151
X509\_UN
 tls\_x509.h, 151