libacvp

Generated by Doxygen 1.8.14

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

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

Chapter 3

Data Structure Documentation

3.1 ACVP_CIPHER Struct Reference

This enum lists the various algorithms supported by the ACVP library.

`#include <acvp.h>`

3.1.1 Detailed Description

This enum lists the various algorithms supported by the ACVP library.

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

- src/acvp.h

3.2 ACVP_CMAC_MSG_LEN_INDEX Struct Reference

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

- src/acvp.h

3.3 ACVP_CMAC_PARM Struct Reference

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

- src/acvp.h

3.4 ACVP_CMAC_TC Struct Reference

This struct holds data that represents a single test case for CMAC testing. This data is passed between libacvp and the crypto module.

`#include <acvp.h>`

3.4.1 Detailed Description

This struct holds data that represents a single test case for CMAC testing. This data is passed between libacvp and the crypto module.

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

- src/acvp.h

3.5 acvp_cmac_tc_t Struct Reference

Data Fields

- ACVP_CIPHER **cipher**
- char **direction** [4]
- char **ver_disposition** [5]
- unsigned int **tc_id**
- unsigned char * **msg**
- unsigned int **msg_len**
- unsigned char * **mac**
- unsigned int **mac_len**
- unsigned int **key_len**
- unsigned char * **key**
- unsigned char * **key2**
- unsigned char * **key3**

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

- src/acvp.h

3.6 ACVP_CTX Struct Reference

This opaque structure is used to maintain the state of a test session with an ACVP server. A single instance of this context represents a test session with the ACVP server. This context is used by the application layer to perform the steps to conduct a test. These steps are:

`#include <acvp.h>`

3.6.1 Detailed Description

This opaque structure is used to maintain the state of a test session with an ACVP server. A single instance of this context represents a test session with the ACVP server. This context is used by the application layer to perform the steps to conduct a test. These steps are:

1. Create the context
2. Specify the server hostname
3. Specify the crypto algorithms to test
4. Register with the ACVP server
5. Commence the test with the server
6. Check the test results
7. Free the context

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

- src/acvp.h

3.7 ACVP_DRBG_MODE Struct Reference

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

- src/acvp.h

3.8 ACVP_DRBG_PARM Struct Reference

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

- src/acvp.h

3.9 ACVP_DRBG_TC Struct Reference

This struct holds data that represents a single test case for DRBG testing. This data is passed between libacvp and the crypto module.

```
#include <acvp.h>
```

3.9.1 Detailed Description

This struct holds data that represents a single test case for DRBG testing. This data is passed between libacvp and the crypto module.

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

- src/acvp.h

3.10 acvp_drbg_tc_t Struct Reference

Data Fields

- ACVP_CIPHER **cipher**
- ACVP_DRBG_MODE **mode**
- unsigned int **tc_id**
- unsigned char * **additional_input**
- unsigned char * **entropy_input_pr**
- unsigned char * **additional_input_1**
- unsigned char * **entropy_input_pr_1**
- unsigned char * **perso_string**
- unsigned char * **entropy**
- unsigned char * **nonce**
- unsigned char * **drb**
- unsigned int **additional_input_len**
- unsigned int **pred_resist_enabled**
- unsigned int **perso_string_len**
- unsigned int **der_func_enabled**
- unsigned int **entropy_len**
- unsigned int **nonce_len**
- unsigned int **drb_len**

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

- src/acvp.h

3.11 ACVP_DSA_GEN_PARM Struct Reference

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

- src/acvp.h

3.12 ACVP_DSA_MODE Struct Reference

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

- src/acvp.h

3.13 ACVP_DSA_PARM Struct Reference

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

- src/acvp.h

3.14 ACVP_DSA_PQGGEN_TC Struct Reference

This struct holds data that represents a single test case for DSA testing. This data is passed between libacvp and the crypto module.

```
#include <acvp.h>
```

3.14.1 Detailed Description

This struct holds data that represents a single test case for DSA testing. This data is passed between libacvp and the crypto module.

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

- src/acvp.h

3.15 acvp_dsa_pqggen_tc_t Struct Reference

Data Fields

- int **l**
- int **n**
- int **h**
- int **sha**
- int **gen_pq**
- int **num**
- int **index**
- int **seedlen**
- unsigned char * **p**
- unsigned char * **q**
- unsigned char * **g**
- unsigned char * **seed**
- int **counter**

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

- src/acvp.h

3.16 ACVP_DSA_SHA Struct Reference

these are bit flags

`#include <acvp.h>`

3.16.1 Detailed Description

these are bit flags

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

- src/acvp.h

3.17 ACVP_DSA_TC Struct Reference

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

- src/acvp.h

3.18 acvp_dsa_tc_t Struct Reference

Data Fields

- ACVP_CIPHER **cipher**
- union {
  ACVP_DSA_PQGGEN_TC * **pqggen**
  } **mode_tc**
- ACVP_DSA_MODE **mode**
- int **l**
- int **n**
- int **h**
- int **sha**
- int **gen_pq**
- int **num**
- int **index**
- int **seedlen**
- int **msglen**
- int **result**
- unsigned char * **p**
- unsigned char * **q**
- unsigned char * **g**
- unsigned char * **y**
- unsigned char * **r**
- unsigned char * **s**
- unsigned char * **seed**
- unsigned char * **msg**
- int **counter**

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

- src/acvp.h

3.19 ACVP_ECDSA_PARM Struct Reference

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

- src/acvp.h

3.20 ACVP_ECDSA_TC Struct Reference

This struct holds data that represents a single test case for ECDSA testing. This data is passed between libacvp and the crypto module.

```
#include <acvp.h>
```

3.20.1 Detailed Description

This struct holds data that represents a single test case for ECDSA testing. This data is passed between libacvp and the crypto module.

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

- src/acvp.h

3.21 acvp_ecdsa_tc_t Struct Reference

Data Fields

- char * **hash_alg**
- unsigned int **tc_id**
- ACVP_CIPHER **cipher**
- char * **curve**
- char * **secret_gen_mode**
- unsigned char * **d**
- unsigned char * **qy**
- unsigned char * **qx**
- unsigned char * **r**
- unsigned char * **s**
- char * **ver_disposition**
- unsigned char * **message**

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

- src/acvp.h

3.22 ACVP_ENTROPY_TC Struct Reference

This struct holds data that represents a single test case for entropy testing. This data is passed between libacvp and the crypto module.

`#include <acvp.h>`

3.22.1 Detailed Description

This struct holds data that represents a single test case for entropy testing. This data is passed between libacvp and the crypto module.

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

- src/acvp.h

3.23 acvp_entropy_tc_t Struct Reference

Data Fields

- ACVP_CIPHER **cipher**
- unsigned int **tc_id**
- unsigned int **entropy_len**
- unsigned char * **entropy_data**

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

- src/acvp.h

3.24 ACVP_HASH_TC Struct Reference

This struct holds data that represents a single test case for hash testing. This data is passed between libacvp and the crypto module.

`#include <acvp.h>`

3.24.1 Detailed Description

This struct holds data that represents a single test case for hash testing. This data is passed between libacvp and the crypto module.

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

- src/acvp.h

3.25 acvp_hash_tc_t Struct Reference

Data Fields

- ACVP_CIPHER **cipher**
- unsigned int **tc_id**
- ACVP_HASH_TESTTYPE **test_type**
- unsigned char * **msg**
- unsigned char * **m1**
- unsigned char * **m2**
- unsigned char * **m3**
- unsigned int **msg_len**
- unsigned char * **md**
- unsigned int **md_len**

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

- src/acvp.h

3.26 ACVP_HASH_TESTTYPE Struct Reference

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

- src/acvp.h

3.27 ACVP_HMAC_PARM Struct Reference

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

- src/acvp.h

3.28 ACVP_HMAC_TC Struct Reference

This struct holds data that represents a single test case for HMAC testing. This data is passed between libacvp and the crypto module.

```
#include <acvp.h>
```

3.28.1 Detailed Description

This struct holds data that represents a single test case for HMAC testing. This data is passed between libacvp and the crypto module.

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

- src/acvp.h

3.29 acvp_hmac_tc_t Struct Reference

Data Fields

- ACVP_CIPHER **cipher**
- unsigned int **tc_id**
- unsigned char * **msg**
- unsigned int **msg_len**
- unsigned char * **mac**
- unsigned int **mac_len**
- unsigned int **key_len**
- unsigned char * **key**

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

- src/acvp.h

3.30 ACVP_KDF135_IKEV1_TC Struct Reference

This struct holds data that represents a single test case for kdf135 IKEV1 testing. This data is passed between libacvp and the crypto module.

```
#include <acvp.h>
```

3.30.1 Detailed Description

This struct holds data that represents a single test case for kdf135 IKEV1 testing. This data is passed between libacvp and the crypto module.

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

- src/acvp.h

3.31 acvp_kdf135_ikev1_tc_t Struct Reference

Data Fields

- ACVP_CIPHER **cipher**
- unsigned int **tc_id**
- unsigned char * **hash_alg**
- char **auth_method** [3]
- int **init_nonce_len**
- int **resp_nonce_len**
- int **dh_secret_len**
- int **psk_len**
- unsigned char * **init_nonce**
- unsigned char * **resp_nonce**

- unsigned char * **init_ckey**
- unsigned char * **resp_ckey**
- unsigned char * **gxy**
- unsigned char * **psk**
- unsigned char * **s_key_id**
- unsigned char * **s_key_id_d**
- unsigned char * **s_key_id_a**
- unsigned char * **s_key_id_e**

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

- src/acvp.h

3.32 ACVP_KDF135_IKEV2_TC Struct Reference

`#include <acvp.h>`

3.32.1 Detailed Description

This struct holds data that represents a single test case for kdf135 IKEV2 testing. This data is passed between libacvp and the crypto module.

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

- src/acvp.h

3.33 acvp_kdf135_ikev2_tc_t Struct Reference

Data Fields

- ACVP_CIPHER **cipher**
- unsigned int **tc_id**
- unsigned char * **hash_alg**
- int **init_nonce_len**
- int **resp_nonce_len**
- int **dh_secret_len**
- int **keying_material_len**
- unsigned char * **init_nonce**
- unsigned char * **resp_nonce**
- unsigned char * **init_spi**
- unsigned char * **resp_spi**
- unsigned char * **gir**
- unsigned char * **gir_new**
- unsigned char * **s_key_seed**
- unsigned char * **s_key_seed_rekey**
- unsigned char * **derived_keying_material**
- unsigned char * **derived_keying_material_child**
- unsigned char * **derived_keying_material_child_dh**

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

- src/acvp.h

3.34 ACVP_KDF135_SNMP_TC Struct Reference

This struct holds data that represents a single test case for kdf135 SNMP testing. This data is passed between libacvp and the crypto module.

`#include <acvp.h>`

3.34.1 Detailed Description

This struct holds data that represents a single test case for kdf135 SNMP testing. This data is passed between libacvp and the crypto module.

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

- src/acvp.h

3.35 acvp_kdf135_snmp_tc_t Struct Reference

Data Fields

- ACVP_CIPHER **cipher**
- unsigned int **tc_id**
- const char * **password**
- unsigned int **p_len**
- unsigned char * **s_key**
- unsigned int **skey_len**
- unsigned char * **engine_id**

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

- src/acvp.h

3.36 ACVP_KDF135_SRTP_PARAM Struct Reference

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

- src/acvp.h

3.37 ACVP_KDF135_SRTP_TC Struct Reference

This struct holds data that represents a single test case for kdf135 SRTP testing. This data is passed between libacvp and the crypto module.

`#include <acvp.h>`

3.37.1 Detailed Description

This struct holds data that represents a single test case for kdf135 SRTP testing. This data is passed between libacvp and the crypto module.

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

- src/acvp.h

3.38 acvp_kdf135_srtp_tc_t Struct Reference

Data Fields

- ACVP_CIPHER **cipher**
- unsigned int **tc_id**
- unsigned char * **kdr**
- int **aes_keylen**
- unsigned char * **master_key**
- unsigned char * **master_salt**
- unsigned char * **index**
- unsigned char * **srtcp_index**
- unsigned char * **srtp_ke**
- unsigned char * **srtp_ka**
- unsigned char * **srtp_ks**
- unsigned char * **srtcp_ke**
- unsigned char * **srtcp_ka**
- unsigned char * **srtcp_ks**

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

- src/acvp.h

3.39 ACVP_KDF135_SSH_CAP_PARM Struct Reference

these are bit flags

`#include <acvp.h>`

3.39.1 Detailed Description

these are bit flags

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

- src/acvp.h

3.40 ACVP_KDF135_SSH_METHOD Struct Reference

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

- src/acvp.h

3.41 ACVP_KDF135_SSH_TC Struct Reference

This struct holds data that represents a single test case for kdf135 SSH testing. This data is passed between libacvp and the crypto module.

`#include <acvp.h>`

3.41.1 Detailed Description

This struct holds data that represents a single test case for kdf135 SSH testing. This data is passed between libacvp and the crypto module.

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

- src/acvp.h

3.42 acvp_kdf135_ssh_tc_t Struct Reference

Data Fields

- ACVP_CIPHER **cipher**
- unsigned int **tc_id**
- unsigned int **sha_type**
- unsigned int **sh_sec_len**
- unsigned int **iv_len**
- unsigned int **key_len**
- char * **shared_sec_k**
- char * **hash_h**
- unsigned int **hash_len**
- char * **session_id**
- unsigned int **session_len**
- unsigned char * **cs_init_iv**
- unsigned char * **sc_init_iv**
- unsigned char * **cs_e_key**
- unsigned char * **sc_e_key**
- unsigned char * **cs_i_key**
- unsigned char * **sc_i_key**

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

- src/acvp.h

3.43 ACVP_KDF135_TLS_CAP_PARM Struct Reference

these are bit flags

`#include <acvp.h>`

3.43.1 Detailed Description

these are bit flags

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

- src/acvp.h

3.44 ACVP_KDF135_TLS_METHOD Struct Reference

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

- src/acvp.h

3.45 ACVP_KDF135_TLS_TC Struct Reference

This struct holds data that represents a single test case for kdf135 TLS testing. This data is passed between libacvp and the crypto module.

`#include <acvp.h>`

3.45.1 Detailed Description

This struct holds data that represents a single test case for kdf135 TLS testing. This data is passed between libacvp and the crypto module.

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

- src/acvp.h

3.46 acvp_kdf135_tls_tc_t Struct Reference

Data Fields

- ACVP_CIPHER **cipher**
- unsigned int **tc_id**
- unsigned int **method**
- unsigned int **md**
- unsigned int **pm_len**
- unsigned int **kb_len**
- unsigned char * **pm_secret**
- unsigned char * **sh_rnd**
- unsigned char * **ch_rnd**
- unsigned char * **s_rnd**
- unsigned char * **c_rnd**
- unsigned char * **msecret1**
- unsigned char * **msecret2**
- unsigned char * **kblock1**
- unsigned char * **kblock2**

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

- src/acvp.h

3.47 ACVP_PREREQ_ALG Struct Reference

This enum lists the prerequisities that are available to the library during registration. Whereas an ACVP_CIPHER may specify a certain mode or key size, the prereqs are more generic.

`#include <acvp.h>`

3.47.1 Detailed Description

This enum lists the prerequisities that are available to the library during registration. Whereas an ACVP_CIPHER may specify a certain mode or key size, the prereqs are more generic.

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

- src/acvp.h

3.48 acvp_prereqs_mode_name_t Struct Reference

Data Fields

- ACVP_PREREQ_ALG **alg**
- char * **name**

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

- src/acvp.c

3.49 ACVP_RESULT Struct Reference

This enum is used to indicate error conditions to the application layer. Most libacvp function will return a value from this enum.

`#include <acvp.h>`

3.49.1 Detailed Description

This enum is used to indicate error conditions to the application layer. Most libacvp function will return a value from this enum.

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

- src/acvp.h

3.50 ACVP_RSA_KEYGEN_MODE Struct Reference

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

- src/acvp.h

3.51 ACVP_RSA_KEYGEN_TC Struct Reference

This struct holds data that represents a single test case for RSA keygen testing. The other modes of RSA have their own respective structs. This data is passed between libacvp and the crypto module.

`#include <acvp.h>`

3.51.1 Detailed Description

This struct holds data that represents a single test case for RSA keygen testing. The other modes of RSA have their own respective structs. This data is passed between libacvp and the crypto module.

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

- src/acvp.h

3.52 acvp_rsa_keygen_tc_t Struct Reference

Data Fields

- char * **hash_alg**
- unsigned int **tc_id**
- char * **pub_exp**
- char * **prime_test**
- char * **prime_result**
- int **rand_pq**
- int **info_gen_by_server**
- char * **pub_exp_mode**
- char * **key_format**
- int **modulo**
- unsigned char * **e**
- unsigned char * **p_rand**
- unsigned char * **q_rand**
- unsigned char * **xp1**
- unsigned char * **xp2**
- unsigned char * **xp**
- unsigned char * **xq1**
- unsigned char * **xq2**
- unsigned char * **xq**
- unsigned char * **dmp1**
- unsigned char * **dmq1**
- unsigned char * **iqmp**
- unsigned char * **n**
- unsigned char * **d**
- unsigned char * **p**
- unsigned char * **q**
- unsigned char * **seed**
- int **seed_len**
- int **bitlen1**
- int **bitlen2**
- int **bitlen3**
- int **bitlen4**

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

- src/acvp.h

3.53 ACVP_RSA_PARM Struct Reference

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

- src/acvp.h

3.54 ACVP_RSA_SIG_TC Struct Reference

This struct holds data that represents a single test case for RSA signature testing. Both siggen and sigver use this struct in their testing. This data is passed between libacvp and the crypto module.

`#include <acvp.h>`

3.54.1 Detailed Description

This struct holds data that represents a single test case for RSA signature testing. Both siggen and sigver use this struct in their testing. This data is passed between libacvp and the crypto module.

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

- src/acvp.h

3.55 acvp_rsa_sig_tc_t Struct Reference

Data Fields

- char * **hash_alg**
- char * **sig_type**
- unsigned int **tc_id**
- unsigned int **modulo**
- unsigned char * **e**
- unsigned char * **n**
- int **salt_len**
- unsigned char * **msg**
- int **msg_len**
- unsigned char * **signature**
- int **sig_len**
- ACVP_CIPHER **sig_mode**
- int **ver_disposition**

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

- src/acvp.h

3.56 ACVP_RSA_SIG_TYPE Struct Reference

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

- src/acvp.h

3.57 ACVP_SYM_CIPH_DIR Struct Reference

These are the algorithm direction suppported by libacvp. These are used in conjunction with ACVP_SYM_CIPH when registering the crypto module capabilities with libacvp.

`#include <acvp.h>`

3.57.1 Detailed Description

These are the algorithm direction suppported by libacvp. These are used in conjunction with ACVP_SYM_CIPH when registering the crypto module capabilities with libacvp.

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

- src/acvp.h

3.58 ACVP_SYM_CIPH_IVGEN_MODE Struct Reference

The IV generation mode. It can comply with 8.2.1, 8.2.2, or may not be applicable for some ciphers.

`#include <acvp.h>`

3.58.1 Detailed Description

The IV generation mode. It can comply with 8.2.1, 8.2.2, or may not be applicable for some ciphers.

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

- src/acvp.h

3.59 ACVP_SYM_CIPH_IVGEN_SRC Struct Reference

The IV generation source for AEAD ciphers. This can be internal, external, or not applicable.

`#include <acvp.h>`

3.59.1 Detailed Description

The IV generation source for AEAD ciphers. This can be internal, external, or not applicable.

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

- src/acvp.h

3.60 ACVP_SYM_CIPH_KO Struct Reference

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

- src/acvp.h

3.61 ACVP_SYM_CIPH_TESTTYPE Struct Reference

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

- src/acvp.h

3.62 ACVP_SYM_CIPH_TWEAK_MODE Struct Reference

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

- src/acvp.h

3.63 ACVP_SYM_CIPHER_PARM Struct Reference

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

- src/acvp.h

3.64 ACVP_SYM_CIPHER_TC Struct Reference

This struct holds data that represents a single test case for a symmetric cipher, such as AES or DES. This data is passed between libacvp and the crypto module. libacvp will parse the test case parameters from the JSON encoded test vector, fill in this structure, and pass the struct to the crypto module via the handler that was registered with libacvp. The crypto module will then need to perform the crypto operation and fill in the remaining items in the struct for the given test case. The struct is then passed back to libacvp, where it is then used to build the JSON encoded vector response.

```
#include <acvp.h>
```

3.64.1 Detailed Description

This struct holds data that represents a single test case for a symmetric cipher, such as AES or DES. This data is passed between libacvp and the crypto module. libacvp will parse the test case parameters from the JSON encoded test vector, fill in this structure, and pass the struct to the crypto module via the handler that was registered with libacvp. The crypto module will then need to perform the crypto operation and fill in the remaining items in the struct for the given test case. The struct is then passed back to libacvp, where it is then used to build the JSON encoded vector response.

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

- src/acvp.h

3.65 acvp_sym_cipher_tc_t Struct Reference

Data Fields

- ACVP_CIPHER **cipher**
- ACVP_SYM_CIPH_TESTTYPE **test_type**
- ACVP_SYM_CIPH_DIR **direction**
- ACVP_SYM_CIPH_IVGEN_SRC **ivgen_source**
- ACVP_SYM_CIPH_IVGEN_MODE **ivgen_mode**
- unsigned int **tc_id**
- unsigned char * **key**
- unsigned char * **pt**
- unsigned char * **aad**
- unsigned char * **iv**
- unsigned char * **ct**
- unsigned char * **tag**
- unsigned char * **iv_ret**
- unsigned char * **iv_ret_after**
- unsigned int **kwcipher**
- unsigned int **key_len**
- unsigned int **pt_len**
- unsigned int **aad_len**
- unsigned int **iv_len**
- unsigned int **ct_len**
- unsigned int **tag_len**
- unsigned int **mct_index**

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

- src/acvp.h

3.66 ACVP_SYM_KW_MODE Struct Reference

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

- src/acvp.h

3.67 ACVP_TEST_CASE Struct Reference

This is the abstracted test case representation used for passing test case data to/from the crypto module. Because the callback prototype is generic to all algorithms, we abstract the various classes of test cases using a union. This struct is then used to pass a reference to the test case between libacvp and the crypto module.

```
#include <acvp.h>
```

3.67.1 Detailed Description

This is the abstracted test case representation used for passing test case data to/from the crypto module. Because the callback prototype is generic to all algorithms, we abstract the various classes of test cases using a union. This struct is then used to pass a reference to the test case between libacvp and the crypto module.

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

- src/acvp.h

3.68 acvp_test_case_t Struct Reference

Data Fields

- union {
  ACVP_SYM_CIPHER_TC * **symmetric**
  ACVP_ENTROPY_TC * **entropy**
  ACVP_HASH_TC * **hash**
  ACVP_DRBG_TC * **drbg**
  ACVP_DSA_TC * **dsa**
  ACVP_HMAC_TC * **hmac**
  ACVP_CMAC_TC * **cmac**
  ACVP_RSA_KEYGEN_TC * **rsa_keygen**
  ACVP_RSA_SIG_TC * **rsa_sig**
  ACVP_ECDSA_TC * **ecdsa**
  ACVP_KDF135_TLS_TC * **kdf135_tls**
  ACVP_KDF135_SNMP_TC * **kdf135_snmp**
  ACVP_KDF135_SSH_TC * **kdf135_ssh**
  ACVP_KDF135_SRTP_TC * **kdf135_srtp**
  } **tc**

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

- src/acvp.h

Chapter 4

File Documentation

4.1 src/acvp.c File Reference

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include "acvp.h"
#include "acvp_lcl.h"
```

Data Structures

- struct acvp_prereqs_mode_name_t

Macros

- #define **ACVP_NUM_PREREQS** 5

Typedefs

- typedef struct acvp_prereqs_mode_name_t **ACVP_PREREQ_MODE_NAME**

Functions

- ACVP_RESULT acvp_create_test_session (ACVP_CTX **ctx, ACVP_RESULT(*progress_cb)(char *msg), ACVP_LOG_LVL level)

  acvp_create_test_session() creates a context that can be used to commence a test session with an ACVP server.
- ACVP_RESULT acvp_set_2fa_callback (ACVP_CTX *ctx, ACVP_RESULT(*totp_cb)(char **token))

  acvp_set_2fa_callback() sets a callback function which will create or obtain a TOTP password for the second part of the two-factor authentication.
- ACVP_RESULT acvp_free_test_session (ACVP_CTX *ctx)

  acvp_free_test_session() releases the memory associated with an ACVP_CTX.

- ACVP_RESULT acvp_enable_sym_cipher_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_SYM_CIPH_DIR dir, ACVP_SYM_CIPH_KO keying_option, ACVP_SYM_CIPH_IVGEN_SRC ivgen_source, ACVP_SYM_CIPH_IVGEN_MODE ivgen_mode, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))

  acvp_enable_sym_cipher_cap() allows an application to specify a symmetric cipher capability to be tested by the ACVP server.
- ACVP_RESULT acvp_enable_sym_cipher_cap_parm (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_SYM_CIPH_PARM parm, int length)

  acvp_enable_sym_cipher_cap_parm() allows an application to specify length-based operational parameters to be used for a given cipher during a test session with the ACVP server.
- ACVP_RESULT acvp_enable_prereq_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_PREREQ_ALG pre_req_cap, char *value)

  acvp_enable_prereq_cap() allows an application to specify a prerequisite for a cipher capability that was previously registered.
- ACVP_RESULT acvp_enable_sym_cipher_cap_value (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_SYM_CIPH_PARM param, int value)

  acvp_enable_sym_cipher_cap_parm() allows an application to specify non length-based operational parameters to be used for a given cipher during a test session with the ACVP server.
- ACVP_RESULT acvp_enable_hash_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))

  acvp_enable_hash_cap() allows an application to specify a hash capability to be tested by the ACVP server.
- ACVP_RESULT **acvp_validate_hash_parm_value** (ACVP_HASH_PARM parm, int value)
- ACVP_RESULT acvp_enable_hash_cap_parm (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_HASH_PARM param, int value)

  acvp_enable_hash_cap_parm() allows an application to specify operational parameters to be used for a given hash alg during a test session with the ACVP server.
- ACVP_RESULT **acvp_validate_hmac_parm_value** (ACVP_CIPHER cipher, ACVP_HMAC_PARM parm, int value)
- ACVP_RESULT acvp_enable_hmac_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))

  acvp_enable_hmac_cap() allows an application to specify an HMAC capability to be tested by the ACVP server.
- ACVP_RESULT acvp_enable_hmac_cap_parm (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_HMAC_PARM parm, int value)

  acvp_enable_hmac_cap_parm() allows an application to specify operational parameters for use during a test session with the ACVP server.
- ACVP_RESULT **acvp_validate_cmac_parm_value** (ACVP_CMAC_PARM parm, int value)
- ACVP_RESULT acvp_enable_cmac_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))

  acvp_enable_cmac_cap() allows an application to specify an CMAC capability to be tested by the ACVP server.
- ACVP_RESULT acvp_enable_cmac_cap_parm (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_CMAC_PARM parm, int value)

  acvp_enable_cmac_cap_parm() allows an application to specify operational parameters for use during a test session with the ACVP server.
- ACVP_RESULT **acvp_validate_drbg_parm_value** (ACVP_DRBG_PARM parm, int value)
- ACVP_RESULT **acvp_enable_rsa_keygen_mode** (ACVP_CTX *ctx, ACVP_RSA_KEYGEN_MODE value)
- ACVP_RESULT acvp_enable_rsa_keygen_cap_parm (ACVP_CTX *ctx, ACVP_RSA_PARM param, int value)

  acvp_enable_rsa_*_cap_parm() allows an application to specify operational parameters to be used for a given RSA alg during a test session with the ACVP server.
- ACVP_RESULT **acvp_enable_ecdsa_cap_parm** (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_ECDSA_PARM param, char *value)
- ACVP_RESULT **acvp_enable_rsa_sigver_cap_parm** (ACVP_CTX *ctx, ACVP_RSA_PARM param, int value)
- ACVP_RESULT **acvp_enable_rsa_sigver_type** (ACVP_CTX *ctx, ACVP_RSA_SIG_TYPE value)
- ACVP_RESULT **acvp_enable_rsa_siggen_type** (ACVP_CTX *ctx, ACVP_RSA_SIG_TYPE value)

- ACVP_RESULT acvp_enable_rsa_keygen_exp_parm (ACVP_CTX *ctx, ACVP_RSA_PARM param, char *value)

  acvp_enable_rsa_bignum_parm() allows an application to specify BIGNUM operational parameters to be used for a given RSA alg during a test session with the ACVP server.
- ACVP_RESULT **acvp_enable_rsa_sigver_exp_parm** (ACVP_CTX *ctx, ACVP_RSA_PARM param, char *value)
- ACVP_RESULT acvp_enable_rsa_keygen_primes_parm (ACVP_CTX *ctx, ACVP_RSA_KEYGEN_MODE mode, int mod, char *name)

  acvp_enable_rsa_primes_parm() allows an application to specify RSA key generation provable or probable primes parameters for use during a test session with the ACVP server.
- ACVP_RESULT **acvp_enable_rsa_sigver_caps_parm** (ACVP_CTX *ctx, ACVP_RSA_SIG_TYPE sig_↩ type, int mod, char *hash_name, int salt_len)
- ACVP_RESULT **acvp_enable_rsa_siggen_caps_parm** (ACVP_CTX *ctx, ACVP_RSA_SIG_TYPE sig_↩ type, int mod, char *hash_name, int salt_len)
- ACVP_RESULT acvp_enable_drbg_cap_parm (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_DRBG_MODE mode, ACVP_DRBG_PARM param, int value)

  acvp_enable_drbg_cap_parm() allows an application to specify operational parameters to be used for a given DRBG alg during a test session with the ACVP server.
- ACVP_RESULT acvp_enable_drbg_prereq_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_DRBG_MODE mode, ACVP_PREREQ_ALG pre_req, char *value)

  acvp_enable_drbg_prereq_cap() allows an application to specify a prerequisite algorithm for a given DRBG during a test session with the ACVP server.
- ACVP_RESULT acvp_enable_drbg_length_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_DRBG_MODE mode, ACVP_DRBG_PARM param, int min, int step, int max)

  acvp_enable_drbg_length_cap() allows an application to register a DRBG capability length-based paramter.
- ACVP_RESULT acvp_enable_drbg_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_RESULT(*crypto↩ _handler)(ACVP_TEST_CASE *test_case))

  acvp_enable_drbg_cap() allows an application to specify a hash capability to be tested by the ACVP server.
- ACVP_RESULT acvp_enable_rsa_keygen_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_RESULT(*crypto↩ _handler)(ACVP_TEST_CASE *test_case))

  acvp_enable_rsa_*_cap()
- ACVP_RESULT **acvp_enable_ecdsa_cap** (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_RESULT(*crypto↩ _handler)(ACVP_TEST_CASE *test_case))
- ACVP_RESULT **acvp_enable_rsa_siggen_cap** (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_RESULT(*crypto↩ _handler)(ACVP_TEST_CASE *test_case))
- ACVP_RESULT **acvp_enable_rsa_sigver_cap** (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_RESULT(*crypto↩ _handler)(ACVP_TEST_CASE *test_case))
- ACVP_RESULT acvp_set_vendor_info (ACVP_CTX *ctx, const char *vendor_name, const char *vendor_url, const char *contact_name, const char *contact_email)

  acvp_set_vendor_info() specifies the vendor attributes for the test session.
- ACVP_RESULT acvp_set_module_info (ACVP_CTX *ctx, const char *module_name, const char *module↩ _type, const char *module_version, const char *module_description)

  acvp_set_module_info() specifies the crypto module attributes for the test session.
- ACVP_RESULT acvp_set_server (ACVP_CTX *ctx, char *server_name, int port)

  acvp_set_server() specifies the ACVP server and TCP port number to use when contacting the server.
- ACVP_RESULT acvp_set_path_segment (ACVP_CTX *ctx, char *path_segment)

  acvp_set_path_segment() specifies the URI prefix used by the ACVP server.
- ACVP_RESULT acvp_set_cacerts (ACVP_CTX *ctx, char *ca_file)

  acvp_set_cacerts() specifies PEM encoded certificates to use as the root trust anchors for establishing the TLS session with the ACVP server.
- ACVP_RESULT acvp_set_certkey (ACVP_CTX *ctx, char *cert_file, char *key_file)

  acvp_set_certkey() specifies PEM encoded certificate and private key to use for establishing the TLS session with the ACVP server.
- void acvp_mark_as_sample (ACVP_CTX *ctx)

acvp_mark_as_sample() *marks the registration as a sample.*

- ACVP_RESULT acvp_register (ACVP_CTX *ctx)

  acvp_register() *registers the DUT with the ACVP server.*
- ACVP_RESULT acvp_process_tests (ACVP_CTX *ctx)

  acvp_process_tests() *performs the ACVP testing procedures.*
- ACVP_RESULT **acvp_retry_handler** (ACVP_CTX *ctx, unsigned int retry_period)
- ACVP_RESULT acvp_check_test_results (ACVP_CTX *ctx)

  acvp_check_test_results() *allows the application to fetch vector set results from the server during a test session.*
- ACVP_RESULT acvp_enable_kdf135_tls_cap (ACVP_CTX *ctx, ACVP_KDF135_TLS_METHOD method, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))

  acvp_enable_kdf135_*_cap() *allows an application to specify a kdf cipher capability to be tested by the ACVP server.*
- ACVP_RESULT acvp_enable_kdf135_tls_cap_parm (ACVP_CTX *ctx, ACVP_CIPHER kcap, ACVP_KDF135_TLS_METHOD method, ACVP_KDF135_TLS_CAP_PARM param)

  acvp_enable_kdf135_tls_cap_parm() *allows an application to specify operational parameters to be used during a test session with the ACVP server.*
- ACVP_RESULT **acvp_enable_kdf135_srtp_cap** (ACVP_CTX *ctx, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))
- ACVP_RESULT **acvp_enable_kdf135_snmp_cap** (ACVP_CTX *ctx, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))
- ACVP_RESULT **acvp_enable_kdf135_ssh_cap** (ACVP_CTX *ctx, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))
- ACVP_RESULT acvp_enable_kdf135_ssh_cap_parm (ACVP_CTX *ctx, ACVP_CIPHER kcap, ACVP_KDF135_SSH_METHOD method, ACVP_KDF135_SSH_CAP_PARM param)

  acvp_enable_kdf135_ssh_cap_parm() *allows an application to specify operational parameters to be used during a test session with the ACVP server.*
- ACVP_RESULT acvp_enable_kdf135_srtp_cap_parm (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_KDF135_SRTP_PARA param, int value)

  acvp_enable_kdf135_srtp_cap_parm() *allows an application to specify operational parameters to be used during a test session with the ACVP server.*
- ACVP_RESULT acvp_enable_dsa_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))

  acvp_enable_dsa_cap()
- ACVP_RESULT acvp_enable_dsa_cap_parm (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_DSA_MODE mode, ACVP_DSA_PARM param, int value)

  acvp_enable_dsa_cap_parm() *allows an application to specify operational parameters to be used for a given hash alg during a test session with the ACVP server.*
- void **ctr64_inc** (unsigned char *counter)
- void **ctr128_inc** (unsigned char *counter)

Variables

- ACVP_ALG_HANDLER **alg_tbl** [ACVP_ALG_MAX]
- struct acvp_prereqs_mode_name_t **acvp_prereqs_tbl** [ACVP_NUM_PREREQS]

4.1.1 Function Documentation

4.1.1.1 acvp_check_test_results()

```
ACVP_RESULT acvp_check_test_results (
            ACVP_CTX * ctx )
```

acvp_check_test_results() allows the application to fetch vector set results from the server during a test session.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.

Returns

ACVP_RESULT

4.1.1.2 acvp_create_test_session()

```
ACVP_RESULT acvp_create_test_session (
              ACVP_CTX ** ctx,
              ACVP_RESULT(*)(char *msg) progress_cb,
              ACVP_LOG_LVL level )
```

acvp_create_test_session() creates a context that can be used to commence a test session with an ACVP server.

This function should be called first to create a context that is used to manage all the API calls into libacvp. The context should be released after the test session has completed by invoking acvp_free_test_session().

When creating a new test session, a function pointer can be provided to receive logging messages from libacvp. The application can then forward the log messages to any logging service it desires, such as syslog.

Parameters

ctx	Address of pointer to unallocated ACVP_CTX.
progress_cb	Address of function to receive log messages from libacvp.

Returns

ACVP_RESULT

4.1.1.3 acvp_enable_cmac_cap()

```
ACVP_RESULT acvp_enable_cmac_cap (
              ACVP_CTX * ctx,
              ACVP_CIPHER cipher,
              ACVP_RESULT(*)(ACVP_TEST_CASE *test_case) crypto_handler )
```

acvp_enable_cmac_cap() allows an application to specify an CMAC capability to be tested by the ACVP server.

This function should be called to enable crypto capabilities for cmac algorithms that will be tested by the ACVP server. This includes CMAC-AES-128, CMAC-AES-192, CMAC-AES-256, etc. This function may be called multiple times to specify more than one crypto capability.

When the application enables a crypto capability, such as CMAC-AES-128, it also needs to specify a callback function that will be used by libacvp when that crypto capability is needed during a test session.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
crypto_handler	Address of function implemented by application that is invoked by libacvp when the crypto capablity is needed during a test session.

Returns

ACVP_RESULT

4.1.1.4 acvp_enable_cmac_cap_parm()

```
ACVP_RESULT acvp_enable_cmac_cap_parm (
             ACVP_CTX * ctx,
             ACVP_CIPHER cipher,
             ACVP_CMAC_PARM parm,
             int value )
```

acvp_enable_cmac_cap_parm() allows an application to specify operational parameters for use during a test session with the ACVP server.

This function allows the application to specify parameters for use when registering CMAC capability with the server.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
parm	ACVP_CMAC_PARM enum value specifying parameter
value	Supported value for the corresponding parameter

Returns

ACVP_RESULT

4.1.1.5 acvp_enable_drbg_cap()

```
ACVP_RESULT acvp_enable_drbg_cap (
             ACVP_CTX * ctx,
             ACVP_CIPHER cipher,
             ACVP_RESULT(*)(ACVP_TEST_CASE *test_case) crypto_handler )
```

acvp_enable_drbg_cap() allows an application to specify a hash capability to be tested by the ACVP server.

This function should be called to enable crypto capabilities for hash algorithms that will be tested by the ACVP server. This includes HASHDRBG, HMACDRBG, CTRDRBG. This function may be called multiple times to specify more than one crypto capability.

When the application enables a crypto capability, such as ACVP_HASHDRBG, it also needs to specify a callback function that will be used by libacvp when that crypto capability is needed during a test session.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
crypto_handler	Address of function implemented by application that is invoked by libacvp when the crypto capablity is needed during a test session.

Returns

ACVP_RESULT

4.1.1.6 acvp_enable_drbg_cap_parm()

```
ACVP_RESULT acvp_enable_drbg_cap_parm (
            ACVP_CTX * ctx,
            ACVP_CIPHER cipher,
            ACVP_DRBG_MODE mode,
            ACVP_DRBG_PARM param,
            int value )
```

acvp_enable_drbg_cap_parm() allows an application to specify operational parameters to be used for a given D↩ RBG alg during a test session with the ACVP server.

This function should be called to enable crypto capabilities for hash capabilities that will be tested by the ACVP server. This includes HASHDRBG, HMACDRBG, CTRDRBG. This function may be called multiple times to specify more than one crypto capability.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
mode	ACVP_DRBG_MODE enum value specifying mode. An example would be ACVP_DRBG_SHA_1
param	ACVP_DRBG_PARM enum value identifying the algorithm parameter that is being specified. An example would be prediction resistance.
value	the value corresponding to the parameter being set

Returns

ACVP_RESULT

4.1.1.7 acvp_enable_drbg_length_cap()

```
ACVP_RESULT acvp_enable_drbg_length_cap (
            ACVP_CTX * ctx,
            ACVP_CIPHER cipher,
```

```
                ACVP_DRBG_MODE mode,
                ACVP_DRBG_PARM param,
                int min,
                int step,
                int max )
```

acvp_enable_drbg_length_cap() allows an application to register a DRBG capability length-based paramter.

This function should be used to register a length-based parameter for a DRBG capability. An example would be entropy, nonce, perso where a minimum, step, and maximum can be specified.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
mode	ACVP_DRBG_MODE enum value specifying mode. An example would be ACVP_DRBG_SHA_1
param	ACVP_DRBG_PARM enum value specifying paramter. An example would be ACVP_DRBG_ENTROPY_LEN
min	minimum value
step	increment value
max	maximum value

Returns

ACVP_RESULT

4.1.1.8 acvp_enable_drbg_prereq_cap()

```
ACVP_RESULT acvp_enable_drbg_prereq_cap (
                ACVP_CTX * ctx,
                ACVP_CIPHER cipher,
                ACVP_DRBG_MODE mode,
                ACVP_PREREQ_ALG pre_req,
                char * value )
```

acvp_enable_drbg_prereq_cap() allows an application to specify a prerequisite algorithm for a given DRBG during a test session with the ACVP server.

This function should be called to enable a prerequisite for a DRBG capability that will be tested by the server.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
mode	ACVP_DRBG_MODE enum value specifying mode. An example would be ACVP_DRBG_SHA_1
pre_req	ACVP_PREREQ_ALG enum that the specified cipher/mode depends on
value	"same" or number

Returns

ACVP_RESULT

4.1.1.9 acvp_enable_dsa_cap()

```
ACVP_RESULT acvp_enable_dsa_cap (
             ACVP_CTX * ctx,
             ACVP_CIPHER cipher,
             ACVP_RESULT(*)(ACVP_TEST_CASE *test_case) crypto_handler )
```

acvp_enable_dsa_cap()

This function should be used to enable DSA capabilities. Specific modes and parameters can use acvp_enable_↩ rsa_cap_parm, acvp_enable_rsa_bignum_parm, acvp_enable_rsa_primes_parm depending on the need.

When the application enables a crypto capability, such as RSA, it also needs to specify a callback function that will be used by libacvp when that crypto capability is needed during a test session.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
crypto_handler	Address of function implemented by application that is invoked by libacvp when the crypto capability is needed during a test session.

Returns

ACVP_RESULT

4.1.1.10 acvp_enable_dsa_cap_parm()

```
ACVP_RESULT acvp_enable_dsa_cap_parm (
             ACVP_CTX * ctx,
             ACVP_CIPHER cipher,
             ACVP_DSA_MODE mode,
             ACVP_DSA_PARM param,
             int value )
```

acvp_enable_dsa_cap_parm() allows an application to specify operational parameters to be used for a given hash alg during a test session with the ACVP server.

This function should be called to enable crypto capabilities for hash capabilities that will be tested by the ACVP server. This includes HASHDRBG, HMACDRBG, CTRDRBG. This function may be called multiple times to specify more than one crypto capability.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
mode	ACVP_DSA_MODE enum value specifying mode. An example would be ACVP_DSA_MODE_PQGGEN
param	ACVP_DSA_PARM enum value identifying the algorithm parameter that is being specified. An example would be ACVP_DSA_GENPQ.
value	the value corresponding to the parameter being set

Returns

ACVP_RESULT

4.1.1.11 acvp_enable_hash_cap()

```
ACVP_RESULT acvp_enable_hash_cap (
             ACVP_CTX * ctx,
             ACVP_CIPHER cipher,
             ACVP_RESULT(*)(ACVP_TEST_CASE *test_case) crypto_handler )
```

acvp_enable_hash_cap() allows an application to specify a hash capability to be tested by the ACVP server.

This function should be called to enable crypto capabilities for hash algorithms that will be tested by the ACVP server. This includes SHA-1, SHA-256, SHA-384, etc. This function may be called multiple times to specify more than one crypto capability.

When the application enables a crypto capability, such as SHA-1, it also needs to specify a callback function that will be used by libacvp when that crypto capability is needed during a test session.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
crypto_handler	Address of function implemented by application that is invoked by libacvp when the crypto capablity is needed during a test session.

Returns

ACVP_RESULT

4.1.1.12 acvp_enable_hash_cap_parm()

```
ACVP_RESULT acvp_enable_hash_cap_parm (
             ACVP_CTX * ctx,
```

```
                ACVP_CIPHER cipher,
                ACVP_HASH_PARM param,
                int value )
```

acvp_enable_hash_cap_parm() allows an application to specify operational parameters to be used for a given hash alg during a test session with the ACVP server.

This function should be called to enable crypto capabilities for hash capabilities that will be tested by the ACVP server. This includes SHA-1, SHA-256, SHA-384, etc.

This function may be called multiple times to specify more than one crypto parameter value for the hash algorithm. The ACVP_CIPHER value passed to this function should already have been setup by invoking acvp_enable_hash_cap().

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
param	ACVP_HASH_PARM enum value identifying the algorithm parameter that is being specified. An example would be a flag indicating if empty input values are allowed.
value	the value corresponding to the parameter being set

Returns

ACVP_RESULT

4.1.1.13 acvp_enable_hmac_cap()

```
ACVP_RESULT acvp_enable_hmac_cap (
                ACVP_CTX * ctx,
                ACVP_CIPHER cipher,
                ACVP_RESULT(*)(ACVP_TEST_CASE *test_case) crypto_handler )
```

acvp_enable_hmac_cap() allows an application to specify an HMAC capability to be tested by the ACVP server.

This function should be called to enable crypto capabilities for hmac algorithms that will be tested by the ACVP server. This includes HMAC-SHA-1, HMAC-SHA2-256, HMAC-SHA2-384, etc. This function may be called multiple times to specify more than one crypto capability.

When the application enables a crypto capability, such as HMAC-SHA-1, it also needs to specify a callback function that will be used by libacvp when that crypto capability is needed during a test session.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
crypto_handler	Address of function implemented by application that is invoked by libacvp when the crypto capablity is needed during a test session.

Returns

ACVP_RESULT

4.1.1.14 acvp_enable_hmac_cap_parm()

```
ACVP_RESULT acvp_enable_hmac_cap_parm (
            ACVP_CTX * ctx,
            ACVP_CIPHER cipher,
            ACVP_HMAC_PARM parm,
            int value )
```

acvp_enable_hmac_cap_parm() allows an application to specify operational parameters for use during a test session with the ACVP server.

This function allows the application to specify parameters for use when registering HMAC capability with the server.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
parm	ACVP_HMAC_PARM enum value specifying parameter
value	Supported value for the corresponding parameter

Returns

ACVP_RESULT

4.1.1.15 acvp_enable_kdf135_srtp_cap_parm()

```
ACVP_RESULT acvp_enable_kdf135_srtp_cap_parm (
            ACVP_CTX * ctx,
            ACVP_CIPHER cap,
            ACVP_KDF135_SRTP_PARAM param,
            int value )
```

acvp_enable_kdf135_srtp_cap_parm() allows an application to specify operational parameters to be used during a test session with the ACVP server.

This function should be called after acvp_enable_kdf135_srtp_cap() to specify the parameters for the corresponding KDF.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cap	ACVP_CIPHER enum value identifying the crypto capability, here it will always be ACVP_KDF135_SRTP
param	acvp_enable_kdf135_srtp_cap_parm enum value specifying parameter
value	integer value for parameter

Returns

ACVP_RESULT

4.1.1.16 acvp_enable_kdf135_ssh_cap_parm()

```
ACVP_RESULT acvp_enable_kdf135_ssh_cap_parm (
            ACVP_CTX * ctx,
            ACVP_CIPHER cap,
            ACVP_KDF135_SSH_METHOD method,
            ACVP_KDF135_SSH_CAP_PARM param )
```

acvp_enable_kdf135_ssh_cap_parm() allows an application to specify operational parameters to be used during a test session with the ACVP server.

This function should be called after acvp_enable_kdf135_tls_cap() to specify the parameters for the corresponding KDF.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cap	ACVP_CIPHER enum value identifying the crypto capability, here it will always be ACVP_KDF135_SSH
method	ACVP_KDF135_SSH_METHOD enum value specifying method type
param	ACVP_KDF135_SSH_CAP_PARM enum value

Returns

ACVP_RESULT

4.1.1.17 acvp_enable_kdf135_tls_cap()

```
ACVP_RESULT acvp_enable_kdf135_tls_cap (
            ACVP_CTX * ctx,
            ACVP_KDF135_TLS_METHOD method,
            ACVP_RESULT(*)(ACVP_TEST_CASE *test_case) crypto_handler )
```

acvp_enable_kdf135_*_cap() allows an application to specify a kdf cipher capability to be tested by the ACVP server.

When the application enables a crypto capability, such as KDF135_TLS, it also needs to specify a callback function that will be used by libacvp when that crypto capability is needed during a test session.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
crypto_handler	Address of function implemented by application that is invoked by libacvp when the crypto capablity is needed during a test session.

Returns

ACVP_RESULT

4.1.1.18 acvp_enable_kdf135_tls_cap_parm()

```
ACVP_RESULT acvp_enable_kdf135_tls_cap_parm (
            ACVP_CTX * ctx,
            ACVP_CIPHER cap,
            ACVP_KDF135_TLS_METHOD method,
            ACVP_KDF135_TLS_CAP_PARM param )
```

acvp_enable_kdf135_tls_cap_parm() allows an application to specify operational parameters to be used during a test session with the ACVP server.

This function should be called after acvp_enable_kdf135_tls_cap() to specify the parameters for the corresponding KDF.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cap	ACVP_CIPHER enum value identifying the crypto capability, here it will always be ACVP_KDF135_TLS
method	ACVP_KDF135_TLS_METHOD enum value specifying method type
param	ACVP_KDF135_TLS_CAP_PARM enum value

Returns

ACVP_RESULT

4.1.1.19 acvp_enable_prereq_cap()

```
ACVP_RESULT acvp_enable_prereq_cap (
            ACVP_CTX * ctx,
            ACVP_CIPHER cipher,
            ACVP_PREREQ_ALG pre_req_cap,
            char * value )
```

acvp_enable_prereq_cap() allows an application to specify a prerequisite for a cipher capability that was previously registered.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability that has a prerequisite
pre_req_alg	ACVP_PREREQ_ALG enum identifying the prerequisite
value	value for specified prerequisite

Returns

ACVP_RESULT

4.1.1.20 acvp_enable_rsa_keygen_cap()

```
ACVP_RESULT acvp_enable_rsa_keygen_cap (
             ACVP_CTX * ctx,
             ACVP_CIPHER cipher,
             ACVP_RESULT(*)(ACVP_TEST_CASE *test_case) crypto_handler )
```

acvp_enable_rsa_*_cap()

This function should be used to enable RSA capabilities. Specific modes and parameters can use acvp_enable_↩ rsa_cap_parm, acvp_enable_rsa_bignum_parm, acvp_enable_rsa_primes_parm depending on the need.

When the application enables a crypto capability, such as RSA, it also needs to specify a callback function that will be used by libacvp when that crypto capability is needed during a test session.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
crypto_handler	Address of function implemented by application that is invoked by libacvp when the crypto capablity is needed during a test session.

Returns

ACVP_RESULT

4.1.1.21 acvp_enable_rsa_keygen_cap_parm()

```
ACVP_RESULT acvp_enable_rsa_keygen_cap_parm (
             ACVP_CTX * ctx,
             ACVP_RSA_PARM param,
             int value )
```

acvp_enable_rsa_*_cap_parm() allows an application to specify operational parameters to be used for a given RSA alg during a test session with the ACVP server.

This function should be called to enable parameters for RSA capabilities that will be tested by the ACVP server. This function may be called multiple times to specify more than one crypto capability.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
mode	ACVP_RSA_MODE enum value specifying mode. An example would be ACVP_RSA_MODE_KEYGEN
param	ACVP_RSA_PARM enum value identifying the algorithm parameter that is being specified. An example would be public exponent
value	the value corresponding to the parameter being set

Returns

ACVP_RESULT

4.1.1.22 acvp_enable_rsa_keygen_exp_parm()

```
ACVP_RESULT acvp_enable_rsa_keygen_exp_parm (
            ACVP_CTX * ctx,
            ACVP_RSA_PARM param,
            char * value )
```

acvp_enable_rsa_bignum_parm() allows an application to specify BIGNUM operational parameters to be used for a given RSA alg during a test session with the ACVP server.

This function behaves the same as acvp_enable_rsa_cap_parm() but instead allows the application to specify a BIGNUM parameter

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
mode	ACVP_RSA_MODE enum value specifying mode. An example would be ACVP_RSA_MODE_KEYGEN
param	ACVP_RSA_PARM enum value identifying the algorithm parameter that is being specified. An example would be public exponent
value	BIGNUM value corresponding to the parameter being set

Returns

ACVP_RESULT

4.1.1.23 acvp_enable_rsa_keygen_primes_parm()

```
ACVP_RESULT acvp_enable_rsa_keygen_primes_parm (
            ACVP_CTX * ctx,
            ACVP_RSA_KEYGEN_MODE mode,
            int mod,
            char * name )
```

acvp_enable_rsa_primes_parm() allows an application to specify RSA key generation provable or probable primes parameters for use during a test session with the ACVP server.

The function behaves similarly to acvp_enable_rsa_cap_parm() and acvp_enable_rsa_∗_exp_parm() but allows for a modulo and hash algorithm parameter to be specified alongside the provable or probable parameter.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.

Parameters

cipher	ACVP_CIPHER enum value identifying the crypto capability.
mode	ACVP_RSA_MODE enum value specifying mode. In this case it will always be ACVP_RSA_MODE_KEYGEN
param	ACVP_RSA_PARM enum value identifying the algorithm parameter being specified. Here, it will be one of: ACVP_CAPS_PROV_PRIME, ACVP_CAPS_PROB_PRIME, or ACVP_CAPS_PROV_PROB_PRIME
mod	Supported RSA modulo value for probable or provable prime generation
hash	The corresponding supported hash algorithm for probable or provable prime generation

Returns

ACVP_RESULT

4.1.1.24 acvp_enable_sym_cipher_cap()

```
ACVP_RESULT acvp_enable_sym_cipher_cap (
             ACVP_CTX * ctx,
             ACVP_CIPHER cipher,
             ACVP_SYM_CIPH_DIR dir,
             ACVP_SYM_CIPH_KO keying_options,
             ACVP_SYM_CIPH_IVGEN_SRC ivgen_source,
             ACVP_SYM_CIPH_IVGEN_MODE ivgen_mode,
             ACVP_RESULT(*)(ACVP_TEST_CASE *test_case) crypto_handler )
```

acvp_enable_sym_cipher_cap() allows an application to specify a symmetric cipher capability to be tested by the ACVP server.

This function should be called to enable crypto capabilities for symmetric ciphers that will be tested by the ACVP server. This includes AES and 3DES. This function may be called multiple times to specify more than one crypto capability, such as AES-CBC, AES-CTR, AES-GCM, etc.

When the application enables a crypto capability, such as AES-GCM, it also needs to specify a callback function that will be used by libacvp when that crypto capability is needed during a test session.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
dir	ACVP_SYM_CIPH_DIR enum value identifying the crypto operation (e.g. encrypt or decrypt).
keying_option	ACVP_SYM_CIPH_KO enum value identifying the TDES keying options
ivgen_source	The source of the IV used by the crypto module (e.g. internal or external)
ivgen_mode	The IV generation mode
crypto_handler	Address of function implemented by application that is invoked by libacvp when the crypto capablity is needed during a test session.

Returns

ACVP_RESULT

4.1.1.25 acvp_enable_sym_cipher_cap_parm()

```
ACVP_RESULT acvp_enable_sym_cipher_cap_parm (
            ACVP_CTX * ctx,
            ACVP_CIPHER cipher,
            ACVP_SYM_CIPH_PARM parm,
            int length )
```

acvp_enable_sym_cipher_cap_parm() allows an application to specify length-based operational parameters to be used for a given cipher during a test session with the ACVP server.

This function should be called to enable crypto capabilities for symmetric ciphers that will be tested by the ACVP server. This includes AES and 3DES.

This function may be called multiple times to specify more than one crypto parameter value for the cipher. For instance, if cipher supports plaintext lengths of 0, 128, and 136 bits, then this function would be called three times. Once for 0, once for 128, and once again for 136. The ACVP_CIPHER value passed to this function should already have been setup by invoking acvp_enable_sym_cipher_cap() for that cipher earlier.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
parm	ACVP_SYM_CIPH_PARM enum value identifying the algorithm parameter that is being specified. An example would be the supported plaintext length of the algorithm.
length	The length value for the symmetric cipher parameter being set

Returns

ACVP_RESULT

4.1.1.26 acvp_enable_sym_cipher_cap_value()

```
ACVP_RESULT acvp_enable_sym_cipher_cap_value (
            ACVP_CTX * ctx,
            ACVP_CIPHER cipher,
            ACVP_SYM_CIPH_PARM param,
            int value )
```

acvp_enable_sym_cipher_cap_parm() allows an application to specify non length-based operational parameters to be used for a given cipher during a test session with the ACVP server.

This function should be called to enable crypto capabilities for symmetric ciphers that will be tested by the ACVP server. This includes AES and 3DES.

This function may be called multiple times to specify more than one crypto parameter value for the cipher. The ACVP_CIPHER value passed to this function should already have been setup by invoking acvp_enable_sym_cipher_cap() for that cipher earlier.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
parm	ACVP_SYM_CIPH_PARM enum value identifying the algorithm parameter that is being specified. An example would be the supported key wrap values
value	The length value for the symmetric cipher parameter being set

Returns

ACVP_RESULT

4.1.1.27 acvp_free_test_session()

```
ACVP_RESULT acvp_free_test_session (
            ACVP_CTX * ctx )
```

acvp_free_test_session() releases the memory associated with an ACVP_CTX.

This function will free an ACVP_CTX. Failure to invoke this function will result in a memory leak in the application layer. This function should be invoked after a test session has completed and a reference to the context is no longer needed.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.
level	Select the debug level, see ACVP_LOG_LVL

Returns

ACVP_RESULT

4.1.1.28 acvp_mark_as_sample()

```
void acvp_mark_as_sample (
            ACVP_CTX * ctx )
```

acvp_mark_as_sample() marks the registration as a sample.

This function sets a flag that will allow the client to retrieve the correct answers later on, allowing for comparison and debugging.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.

4.1.1.29 acvp_process_tests()

```
ACVP_RESULT acvp_process_tests (
            ACVP_CTX * ctx )
```

acvp_process_tests() performs the ACVP testing procedures.

This function will commence the test session after the DUT has been registered with the ACVP server. This function should be invoked after acvp_register() finishes. When invoked, this function will download the vector sets from the ACVP server, process the vectors, and upload the results to the server.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.

Returns

ACVP_RESULT

4.1.1.30 acvp_register()

```
ACVP_RESULT acvp_register (
            ACVP_CTX * ctx )
```

acvp_register() registers the DUT with the ACVP server.

This function is used to regitser the DUT with the server. Registration allows the DUT to advertise it's capabilities to the server. The server will respond with a set of vector set identifiers that the client will need to process.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.

Returns

ACVP_RESULT

4.1.1.31 acvp_set_2fa_callback()

```
ACVP_RESULT acvp_set_2fa_callback (
            ACVP_CTX * ctx,
            ACVP_RESULT(*)(char **token) totp_cb )
```

acvp_set_2fa_callback() sets a callback function which will create or obtain a TOTP password for the second part of the two-factor authentication.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.
totp_cb	Function that will get the TOTP password

Returns

ACVP_RESULT

4.1.1.32 acvp_set_cacerts()

```
ACVP_RESULT acvp_set_cacerts (
              ACVP_CTX * ctx,
              char * ca_file )
```

acvp_set_cacerts() specifies PEM encoded certificates to use as the root trust anchors for establishing the TLS session with the ACVP server.

ACVP uses TLS as the transport. In order to verify the identity of the ACVP server, the TLS stack requires one or more root certificates that can be used to verify the identify of the ACVP TLS certificate during the TLS handshake. These root certificates are set using this function. They must be PEM encoded and all contained in the same file.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.
ca_file	Name of file containing all the PEM encoded X.509 certificates used as trust anchors for the TLS session.

Returns

ACVP_RESULT

4.1.1.33 acvp_set_certkey()

```
ACVP_RESULT acvp_set_certkey (
              ACVP_CTX * ctx,
              char * cert_file,
              char * key_file )
```

acvp_set_certkey() specifies PEM encoded certificate and private key to use for establishing the TLS session with the ACVP server.

ACVP uses TLS as the transport. In order for the ACVP server to verify the identity the DUT using libacvp, a certificate needs to be presented during the TLS handshake. The certificate used by libacvp needs to be trusted by the ACVP server. Otherwise the TLS handshake will fail.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.
cert_file	Name of file containing the PEM encoded X.509 certificate to use as the client identity.
key_file	Name of file containing PEM encoded private key associated with the client certificate.

Returns

ACVP_RESULT

4.1.1.34 acvp_set_module_info()

```
ACVP_RESULT acvp_set_module_info (
            ACVP_CTX * ctx,
            const char * module_name,
            const char * module_type,
            const char * module_version,
            const char * module_description )
```

acvp_set_module_info() specifies the crypto module attributes for the test session.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.
module_name	Name of the crypto module under test.
module_type	The crypto module type: software, hardware, or hybrid.
module_version	The version# of the crypto module under test.
module_description	A brief description of the crypto module under test.

Returns

ACVP_RESULT

4.1.1.35 acvp_set_path_segment()

```
ACVP_RESULT acvp_set_path_segment (
            ACVP_CTX * ctx,
            char * path_segment )
```

acvp_set_path_segment() specifies the URI prefix used by the ACVP server.

Some ACVP servers use a prefix in the URI for the path to the ACVP REST interface. Calling this function allows the path segment prefix to be specified. The value provided to this function is prepended to the path segment of the URI used for the ACVP REST calls.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.
path_segment	Value to embed in the URI path after the server name and before the ACVP well-known path.

Returns

ACVP_RESULT

4.1.1.36 acvp_set_server()

```
ACVP_RESULT acvp_set_server (
              ACVP_CTX * ctx,
              char * server_name,
              int port )
```

acvp_set_server() specifies the ACVP server and TCP port number to use when contacting the server.

This function is used to specify the hostname or IP address of the ACVP server. The TCP port number can also be specified if the server doesn't use port 443.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.
server_name	Name or IP address of the ACVP server.
port	TCP port number the server listens on.

Returns

ACVP_RESULT

4.1.1.37 acvp_set_vendor_info()

```
ACVP_RESULT acvp_set_vendor_info (
              ACVP_CTX * ctx,
              const char * vendor_name,
              const char * vendor_url,
              const char * contact_name,
              const char * contact_email )
```

acvp_set_vendor_info() specifies the vendor attributes for the test session.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.
vendor_name	Name of the vendor that owns the crypto module.
vendor_url	The Vendor's URL.
contact_name	Name of contact at Vendor.
contact_email	Email of vendor contact.

Returns

ACVP_RESULT

4.1.2 Variable Documentation

4.1.2.1 acvp_prereqs_tbl

```
struct acvp_prereqs_mode_name_t acvp_prereqs_tbl[ACVP_NUM_PREREQS]
```

Initial value:

```
= {
        {ACVP_PREREQ_AES,  "AES"},
        {ACVP_PREREQ_DRBG, "DRBG"},
        {ACVP_PREREQ_HMAC, "HMAC"},
        {ACVP_PREREQ_SHA,  "SHA"},
        {ACVP_PREREQ_TDES, "TDES"}
}
```

4.2 src/acvp.h File Reference

Data Structures

- struct acvp_sym_cipher_tc_t
- struct acvp_entropy_tc_t
- struct acvp_hash_tc_t
- struct acvp_kdf135_tls_tc_t
- struct acvp_kdf135_ikev2_tc_t
- struct acvp_kdf135_ikev1_tc_t
- struct acvp_kdf135_snmp_tc_t
- struct acvp_kdf135_srtp_tc_t
- struct acvp_kdf135_ssh_tc_t
- struct acvp_hmac_tc_t
- struct acvp_cmac_tc_t
- struct acvp_rsa_keygen_tc_t
- struct acvp_ecdsa_tc_t
- struct acvp_rsa_sig_tc_t
- struct acvp_dsa_pqggen_tc_t
- struct acvp_dsa_tc_t
- struct acvp_drbg_tc_t
- struct acvp_test_case_t

Macros

- #define **ACVP_TOTP_LENGTH** 8
- #define **ACVP_TOTP_TOKEN_MAX** 128
- #define **ACVP_KDF135_SNMP_ENGID_MAX** 32
- #define **ACVP_KDF135_SNMP_SKEY_MAX** 32
- #define **ACVP_STR_SHA_1** "SHA-1"
- #define **ACVP_STR_SHA_224** "SHA-224"
- #define **ACVP_STR_SHA_256** "SHA-256"
- #define **ACVP_STR_SHA_384** "SHA-384"
- #define **ACVP_STR_SHA_512** "SHA-512"
- #define **ACVP_STR_SHA_512_224** "SHA-512/224"
- #define **ACVP_STR_SHA_512_256** "SHA-512/256"
- #define **ACVP_STR_SHA2_224** "SHA2-224"
- #define **ACVP_STR_SHA2_256** "SHA2-256"
- #define **ACVP_STR_SHA2_384** "SHA2-384"
- #define **ACVP_STR_SHA2_512** "SHA2-512"
- #define **ACVP_STR_SHA2_512_224** "SHA2-512/224"
- #define **ACVP_STR_SHA2_512_256** "SHA2-512/256"
- #define **RSA_SIG_TYPE_X931_NAME** "ansx9.31"
- #define **RSA_SIG_TYPE_PKCS1V15_NAME** "pkcs1v1.5"
- #define **RSA_SIG_TYPE_PKCS1PSS_NAME** "pss"
- #define **PRIME_TEST_TBLC2_NAME** "tblC2"
- #define **PRIME_TEST_TBLC3_NAME** "tblC3"
- #define **RSA_PUB_EXP_FIXED** 1
- #define **RSA_PUB_EXP_RANDOM** 0

Typedefs

- typedef enum acvp_log_lvl **ACVP_LOG_LVL**
- typedef struct acvp_ctx_t **ACVP_CTX**
- typedef enum acvp_result **ACVP_RESULT**
- typedef enum acvp_cipher **ACVP_CIPHER**
- typedef enum acvp_prereq_mode_t **ACVP_PREREQ_ALG**
- typedef enum acvp_kdf135_tls_cap_parm **ACVP_KDF135_TLS_CAP_PARM**
- typedef enum acvp_kdf135_ssh_cap_parm **ACVP_KDF135_SSH_CAP_PARM**
- typedef enum acvp_kdf135_ssh_method **ACVP_KDF135_SSH_METHOD**
- typedef enum acvp_kdf135_srtp_param **ACVP_KDF135_SRTP_PARAM**
- typedef enum acvp_capability_type **ACVP_CAP_TYPE**
- typedef enum acvp_sym_cipher_keying_option **ACVP_SYM_CIPH_KO**
- typedef enum acvp_sym_cipher_ivgen_source **ACVP_SYM_CIPH_IVGEN_SRC**
- typedef enum acvp_sym_cipher_ivgen_mode **ACVP_SYM_CIPH_IVGEN_MODE**
- typedef enum acvp_sym_cipher_direction **ACVP_SYM_CIPH_DIR**
- typedef enum acvp_kdf135_tls_method **ACVP_KDF135_TLS_METHOD**
- typedef enum acvp_hash_param **ACVP_HASH_PARM**
- typedef enum acvp_drbg_mode **ACVP_DRBG_MODE**
- typedef enum acvp_drbg_param **ACVP_DRBG_PARM**
- typedef enum acvp_rsa_param **ACVP_RSA_PARM**
- typedef enum acvp_ecdsa_param **ACVP_ECDSA_PARM**
- typedef enum acvp_rsa_keygen_mode_t **ACVP_RSA_KEYGEN_MODE**
- typedef enum acvp_rsa_sig_type **ACVP_RSA_SIG_TYPE**
- typedef enum acvp_sym_cipher_parameter **ACVP_SYM_CIPH_PARM**
- typedef enum acvp_sym_xts_tweak_mode **ACVP_SYM_CIPH_TWEAK_MODE**
- typedef enum acvp_sym_kw_mode **ACVP_SYM_KW_MODE**

- typedef enum acvp_sym_cipher_testtype **ACVP_SYM_CIPH_TESTTYPE**
- typedef enum acvp_hash_testtype **ACVP_HASH_TESTTYPE**
- typedef enum acvp_hmac_parameter **ACVP_HMAC_PARM**
- typedef enum acvp_cmac_parameter **ACVP_CMAC_PARM**
- typedef enum acvp_cmac_msg_len_index **ACVP_CMAC_MSG_LEN_INDEX**
- typedef struct acvp_sym_cipher_tc_t **ACVP_SYM_CIPHER_TC**
- typedef struct acvp_entropy_tc_t **ACVP_ENTROPY_TC**
- typedef struct acvp_hash_tc_t **ACVP_HASH_TC**
- typedef struct acvp_kdf135_tls_tc_t **ACVP_KDF135_TLS_TC**
- typedef struct acvp_kdf135_ikev2_tc_t **ACVP_KDF135_IKEV2_TC**
- typedef struct acvp_kdf135_ikev1_tc_t **ACVP_KDF135_IKEV1_TC**
- typedef struct acvp_kdf135_snmp_tc_t **ACVP_KDF135_SNMP_TC**
- typedef struct acvp_kdf135_srtp_tc_t **ACVP_KDF135_SRTP_TC**
- typedef struct acvp_kdf135_ssh_tc_t **ACVP_KDF135_SSH_TC**
- typedef struct acvp_hmac_tc_t **ACVP_HMAC_TC**
- typedef struct acvp_cmac_tc_t **ACVP_CMAC_TC**
- typedef struct acvp_rsa_keygen_tc_t **ACVP_RSA_KEYGEN_TC**
- typedef struct acvp_ecdsa_tc_t **ACVP_ECDSA_TC**
- typedef struct acvp_rsa_sig_tc_t **ACVP_RSA_SIG_TC**
- typedef enum acvp_dsa_mode **ACVP_DSA_MODE**
- typedef enum acvp_dsa_sha **ACVP_DSA_SHA**
- typedef enum acvp_dsa_parm **ACVP_DSA_PARM**
- typedef enum acvp_dsa_gen_parm **ACVP_DSA_GEN_PARM**
- typedef struct acvp_dsa_pqggen_tc_t **ACVP_DSA_PQGGEN_TC**
- typedef struct acvp_dsa_tc_t **ACVP_DSA_TC**
- typedef struct acvp_drbg_tc_t **ACVP_DRBG_TC**
- typedef struct acvp_test_case_t **ACVP_TEST_CASE**

Enumerations

- enum **acvp_log_lvl** {
  ACVP_LOG_LVL_NONE = 0, **ACVP_LOG_LVL_ERR**, **ACVP_LOG_LVL_WARN**, **ACVP_LOG_LVL_ST↩ ATUS**,
  ACVP_LOG_LVL_INFO, **ACVP_LOG_LVL_VERBOSE** }
- enum **acvp_cipher** {
  ACVP_CIPHER_START = 0, **ACVP_AES_GCM**, **ACVP_AES_CCM**, **ACVP_AES_ECB**,
  ACVP_AES_CBC, **ACVP_AES_CFB1**, **ACVP_AES_CFB8**, **ACVP_AES_CFB128**,
  ACVP_AES_OFB, **ACVP_AES_CTR**, **ACVP_AES_XTS**, **ACVP_AES_KW**,
  ACVP_AES_KWP, **ACVP_TDES_ECB**, **ACVP_TDES_CBC**, **ACVP_TDES_CBCI**,
  ACVP_TDES_OFB, **ACVP_TDES_OFBI**, **ACVP_TDES_CFB1**, **ACVP_TDES_CFB8**,
  ACVP_TDES_CFB64, **ACVP_TDES_CFBP1**, **ACVP_TDES_CFBP8**, **ACVP_TDES_CFBP64**,
  ACVP_TDES_CTR, **ACVP_TDES_KW**, **ACVP_SHA1**, **ACVP_SHA224**,
  ACVP_SHA256, **ACVP_SHA384**, **ACVP_SHA512**, **ACVP_HASHDRBG**,
  ACVP_HMACDRBG, **ACVP_CTRDRBG**, **ACVP_HMAC_SHA1**, **ACVP_HMAC_SHA2_224**,
  ACVP_HMAC_SHA2_256, **ACVP_HMAC_SHA2_384**, **ACVP_HMAC_SHA2_512**, **ACVP_HMAC_SHA2↩ _512_224**,
  ACVP_HMAC_SHA2_512_256, **ACVP_HMAC_SHA3_224**, **ACVP_HMAC_SHA3_256**, **ACVP_HMAC_S↩ HA3_384**,
  ACVP_HMAC_SHA3_512, **ACVP_CMAC_AES**, **ACVP_CMAC_TDES**, **ACVP_DSA_KEYGEN**,
  ACVP_DSA_PQGGEN, **ACVP_DSA_PQGVER**, **ACVP_DSA_SIGGEN**, **ACVP_DSA_SIGVER**,
  ACVP_RSA_KEYGEN, **ACVP_RSA_SIGGEN**, **ACVP_RSA_SIGVER**, **ACVP_ECDSA_KEYGEN**,
  ACVP_ECDSA_KEYVER, **ACVP_ECDSA_SIGGEN**, **ACVP_ECDSA_SIGVER**, **ACVP_KDF135_TLS**,
  ACVP_KDF135_SNMP, **ACVP_KDF135_SSH**, **ACVP_KDF135_SRTP**, **ACVP_CIPHER_END** }
- enum **acvp_prereq_mode_t** {
  ACVP_PREREQ_AES = 1, **ACVP_PREREQ_TDES**, **ACVP_PREREQ_DRBG**, **ACVP_PREREQ_HMAC**,
  ACVP_PREREQ_SHA }

- enum **acvp_kdf135_tls_cap_parm** { **ACVP_KDF135_TLS_CAP_SHA256** = 1, **ACVP_KDF135_TLS_C↩ AP_SHA384**, **ACVP_KDF135_TLS_CAP_SHA512**, **ACVP_KDF135_TLS_CAP_MAX** }
- enum **acvp_kdf135_ssh_cap_parm** { **ACVP_KDF135_SSH_CAP_MIN** = 0, **ACVP_KDF135_SSH_CAP↩ _SHA256** = 1, **ACVP_KDF135_SSH_CAP_SHA384** = 2, **ACVP_KDF135_SSH_CAP_SHA512** = 4 }
- enum **acvp_kdf135_ssh_method** {
  ACVP_SSH_METH_TDES_CBC = 1, **ACVP_SSH_METH_AES_128_CBC**, **ACVP_SSH_METH_AES_↩ 192_CBC**, **ACVP_SSH_METH_AES_256_CBC**,
  ACVP_SSH_METH_MAX }
- enum **acvp_kdf135_srtp_param** {
  ACVP_SRTP_PARAM_MIN, **ACVP_SRTP_AES_KEYLEN**, **ACVP_SRTP_SUPPORT_ZERO_KDR**, **AC↩ VP_SRTP_KDF_EXPONENT**,
  ACVP_SRTP_PARAM_MAX }
- enum **acvp_capability_type** {
  ACVP_SYM_TYPE = 1, **ACVP_HASH_TYPE**, **ACVP_DRBG_TYPE**, **ACVP_HMAC_TYPE**,
  ACVP_CMAC_TYPE, **ACVP_RSA_KEYGEN_TYPE**, **ACVP_RSA_SIGGEN_TYPE**, **ACVP_RSA_SIGVE↩ R_TYPE**,
  ACVP_ECDSA_KEYGEN_TYPE, **ACVP_ECDSA_KEYVER_TYPE**, **ACVP_ECDSA_SIGGEN_TYPE**, **A↩ CVP_ECDSA_SIGVER_TYPE**,
  ACVP_DSA_TYPE, **ACVP_KDF135_TLS_TYPE**, **ACVP_KDF135_SNMP_TYPE**, **ACVP_KDF135_SSH_↩ TYPE**,
  ACVP_KDF135_SRTP_TYPE }
- enum **acvp_sym_cipher_keying_option** { **ACVP_KO_NA** = 0, **ACVP_KO_THREE**, **ACVP_KO_TWO**, **A↩ CVP_KO_BOTH** }
- enum **acvp_sym_cipher_ivgen_source** { **ACVP_IVGEN_SRC_INT** = 0, **ACVP_IVGEN_SRC_EXT**, **AC↩ VP_IVGEN_SRC_NA** }
- enum **acvp_sym_cipher_ivgen_mode** { **ACVP_IVGEN_MODE_821** = 0, **ACVP_IVGEN_MODE_822**, **A↩ CVP_IVGEN_MODE_NA** }
- enum **acvp_sym_cipher_direction** { **ACVP_DIR_ENCRYPT** = 0, **ACVP_DIR_DECRYPT**, **ACVP_DIR_↩ BOTH** }
- enum **acvp_kdf135_tls_method** { **ACVP_KDF135_TLS10_TLS11** = 1, **ACVP_KDF135_TLS12** }
- enum **acvp_hash_param** { **ACVP_HASH_IN_BIT** = 0, **ACVP_HASH_IN_EMPTY** }
- enum **acvp_drbg_mode** {
  ACVP_DRBG_MODE_START = 0, **ACVP_DRBG_SHA_1**, **ACVP_DRBG_SHA_224**, **ACVP_DRBG_SH↩ A_256**,
  ACVP_DRBG_SHA_384, **ACVP_DRBG_SHA_512**, **ACVP_DRBG_SHA_512_224**, **ACVP_DRBG_SHA_↩ 512_256**,
  ACVP_DRBG_3KEYTDEA, **ACVP_DRBG_AES_128**, **ACVP_DRBG_AES_192**, **ACVP_DRBG_AES_256**,
  ACVP_DRBG_MODE_END }
- enum **acvp_drbg_param** {
  ACVP_DRBG_DER_FUNC_ENABLED = 0, **ACVP_DRBG_PRED_RESIST_ENABLED**, **ACVP_DRBG_↩ RESEED_ENABLED**, **ACVP_DRBG_ENTROPY_LEN**,
  ACVP_DRBG_NONCE_LEN, **ACVP_DRBG_PERSO_LEN**, **ACVP_DRBG_ADD_IN_LEN**, **ACVP_DRBG↩ _RET_BITS_LEN**,
  ACVP_DRBG_PRE_REQ_VALS }
- enum **acvp_rsa_param** {
  ACVP_PUB_EXP_MODE = 0, **ACVP_FIXED_PUB_EXP_VAL**, **ACVP_KEY_FORMAT_CRT**, **ACVP_RA↩ ND_PQ**,
  ACVP_RSA_INFO_GEN_BY_SERVER }
- enum **acvp_ecdsa_param** { **ACVP_CURVE**, **ACVP_SECRET_GEN_MODE**, **ACVP_HASH_ALG** }
- enum **acvp_rsa_keygen_mode_t** {
  ACVP_RSA_KEYGEN_START = 0, **ACVP_RSA_KEYGEN_B32**, **ACVP_RSA_KEYGEN_B33**, **ACVP_R↩ SA_KEYGEN_B34**,
  ACVP_RSA_KEYGEN_B35, **ACVP_RSA_KEYGEN_B36** }
- enum **acvp_rsa_sig_type** { **RSA_SIG_TYPE_START** = 0, **RSA_SIG_TYPE_X931**, **RSA_SIG_TYPE_P↩ KCS1V15**, **RSA_SIG_TYPE_PKCS1PSS** }

- enum **acvp_sym_cipher_parameter** {
ACVP_SYM_CIPH_KEYLEN = 0, **ACVP_SYM_CIPH_TAGLEN**, **ACVP_SYM_CIPH_IVLEN**, **ACVP_SYM_CIPH_PTLEN**,
ACVP_SYM_CIPH_TWEAK, **ACVP_SYM_CIPH_AADLEN**, **ACVP_SYM_CIPH_KW_MODE** }
- enum **acvp_sym_xts_tweak_mode** { **ACVP_SYM_CIPH_TWEAK_HEX** = 1, **ACVP_SYM_CIPH_TWEAK_NUM**, **ACVP_SYM_CIPH_TWEAK_NONE** }
- enum **acvp_sym_kw_mode** { **ACVP_SYM_KW_NONE** = 0, **ACVP_SYM_KW_CIPHER**, **ACVP_SYM_KW_INVERSE**, **ACVP_SYM_KW_MAX** }
- enum **acvp_sym_cipher_testtype** { **ACVP_SYM_TEST_TYPE_NONE** = 0, **ACVP_SYM_TEST_TYPE_AFT**, **ACVP_SYM_TEST_TYPE_CTR**, **ACVP_SYM_TEST_TYPE_MCT** }
- enum **acvp_hash_testtype** { **ACVP_HASH_TEST_TYPE_NONE** = 0, **ACVP_HASH_TEST_TYPE_AFT**, **ACVP_HASH_TEST_TYPE_MCT** }
- enum **acvp_hmac_parameter** { **ACVP_HMAC_KEYLEN_MIN** = 0, **ACVP_HMAC_KEYLEN_MAX**, **ACVP_HMAC_KEYBLOCK**, **ACVP_HMAC_MACLEN** }
- enum **acvp_cmac_parameter** {
ACVP_CMAC_MACLEN, **ACVP_CMAC_KEYLEN**, **ACVP_CMAC_KEYING_OPTION**, **ACVP_CMAC_DIRECTION_GEN**,
ACVP_CMAC_DIRECTION_VER, **ACVP_CMAC_BLK_DIVISIBLE_1**, **ACVP_CMAC_BLK_DIVISIBLE_2**, **ACVP_CMAC_BLK_NOT_DIVISIBLE_1**,
ACVP_CMAC_BLK_NOT_DIVISIBLE_2, **ACVP_CMAC_MSG_LEN_MAX** }
- enum **acvp_cmac_msg_len_index** {
CMAC_BLK_DIVISIBLE_1 = 0, **CMAC_BLK_DIVISIBLE_2**, **CMAC_BLK_NOT_DIVISIBLE_1**, **CMAC_BLK_NOT_DIVISIBLE_2**,
CMAC_MSG_LEN_MAX, **CMAC_MSG_LEN_NUM_ITEMS** }
- enum **acvp_dsa_mode** {
ACVP_DSA_MODE_KEYGEN = 1, **ACVP_DSA_MODE_PQGGEN**, **ACVP_DSA_MODE_PQGVER**, **ACVP_DSA_MODE_SIGGEN**,
ACVP_DSA_MODE_SIGVER }
- enum **acvp_dsa_sha** {
ACVP_DSA_SHA1 = 1, **ACVP_DSA_SHA224** = 2, **ACVP_DSA_SHA256** = 4, **ACVP_DSA_SHA384** = 8,
ACVP_DSA_SHA512 = 16, **ACVP_DSA_SHA512_224** = 32, **ACVP_DSA_SHA512_256** = 64 }
- enum **acvp_dsa_parm** {
ACVP_DSA_LN2048_224 = 1, **ACVP_DSA_LN2048_256**, **ACVP_DSA_LN3072_256**, **ACVP_DSA_GENPQ**,
ACVP_DSA_GENG }
- enum **acvp_dsa_gen_parm** { **ACVP_DSA_PROVABLE** = 1, **ACVP_DSA_PROBABLE**, **ACVP_DSA_CANONICAL**, **ACVP_DSA_UNVERIFIABLE** }
- enum acvp_result {
ACVP_SUCCESS = 0, ACVP_MALLOC_FAIL, ACVP_NO_CTX, ACVP_TRANSPORT_FAIL,
ACVP_JSON_ERR, **ACVP_UNSUPPORTED_OP**, **ACVP_CLEANUP_FAIL**, **ACVP_KAT_DOWNLOAD_RETRY**,
ACVP_INVALID_ARG, **ACVP_CRYPTO_MODULE_FAIL**, **ACVP_CRYPTO_TAG_FAIL**, **ACVP_CRYPTO_WRAP_FAIL**,
ACVP_NO_TOKEN, **ACVP_NO_CAP**, **ACVP_MALFORMED_JSON**, **ACVP_DATA_TOO_LARGE**,
ACVP_DUP_CIPHER, **ACVP_TOTP_DECODE_FAIL**, **ACVP_TOTP_MISSING_SEED**, **ACVP_RESULT_MAX** }

Functions

- ACVP_RESULT acvp_enable_sym_cipher_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_SYM_CIPH_DIR dir, ACVP_SYM_CIPH_KO keying_options, ACVP_SYM_CIPH_IVGEN_SRC ivgen_source, ACVP_SYM_CIPH_IVGEN_MODE ivgen_mode, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))

  acvp_enable_sym_cipher_cap() allows an application to specify a symmetric cipher capability to be tested by the ACVP server.
- ACVP_RESULT acvp_enable_sym_cipher_cap_value (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_SYM_CIPH_PARM param, int value)

acvp_enable_sym_cipher_cap_parm() allows an application to specify non length-based operational parameters to be used for a given cipher during a test session with the ACVP server.

- ACVP_RESULT acvp_enable_sym_cipher_cap_parm (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_SYM_CIPH_PARM parm, int length)

  acvp_enable_sym_cipher_cap_parm() allows an application to specify length-based operational parameters to be used for a given cipher during a test session with the ACVP server.
- ACVP_RESULT acvp_enable_hash_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))

  acvp_enable_hash_cap() allows an application to specify a hash capability to be tested by the ACVP server.
- ACVP_RESULT acvp_enable_hash_cap_parm (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_HASH_PARM param, int value)

  acvp_enable_hash_cap_parm() allows an application to specify operational parameters to be used for a given hash alg during a test session with the ACVP server.
- ACVP_RESULT acvp_enable_drbg_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))

  acvp_enable_drbg_cap() allows an application to specify a hash capability to be tested by the ACVP server.
- ACVP_RESULT acvp_enable_drbg_cap_parm (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_DRBG_MODE mode, ACVP_DRBG_PARM param, int value)

  acvp_enable_drbg_cap_parm() allows an application to specify operational parameters to be used for a given DRBG alg during a test session with the ACVP server.
- ACVP_RESULT acvp_enable_drbg_prereq_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_DRBG_MODE mode, ACVP_PREREQ_ALG pre_req, char *value)

  acvp_enable_drbg_prereq_cap() allows an application to specify a prerequisite algorithm for a given DRBG during a test session with the ACVP server.
- ACVP_RESULT acvp_enable_drbg_length_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_DRBG_MODE mode, ACVP_DRBG_PARM param, int min, int step, int max)

  acvp_enable_drbg_length_cap() allows an application to register a DRBG capability length-based paramter.
- ACVP_RESULT acvp_enable_dsa_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))

  acvp_enable_dsa_cap()
- ACVP_RESULT acvp_enable_dsa_cap_parm (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_DSA_MODE mode, ACVP_DSA_PARM param, int value)

  acvp_enable_dsa_cap_parm() allows an application to specify operational parameters to be used for a given hash alg during a test session with the ACVP server.
- ACVP_RESULT acvp_enable_rsa_keygen_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))

  acvp_enable_rsa_*_cap()
- ACVP_RESULT **acvp_enable_rsa_siggen_cap** (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))
- ACVP_RESULT **acvp_enable_rsa_sigver_cap** (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))
- ACVP_RESULT **acvp_enable_ecdsa_cap** (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))
- ACVP_RESULT acvp_enable_rsa_keygen_cap_parm (ACVP_CTX *ctx, ACVP_RSA_PARM param, int value)

  acvp_enable_rsa_*_cap_parm() allows an application to specify operational parameters to be used for a given RSA alg during a test session with the ACVP server.
- ACVP_RESULT **acvp_enable_rsa_siggen_cap_parm** (ACVP_CTX *ctx, ACVP_RSA_PARM param, int value)
- ACVP_RESULT **acvp_enable_rsa_sigver_cap_parm** (ACVP_CTX *ctx, ACVP_RSA_PARM param, int value)
- ACVP_RESULT **acvp_enable_rsa_keygen_mode** (ACVP_CTX *ctx, ACVP_RSA_KEYGEN_MODE value)
- ACVP_RESULT **acvp_enable_rsa_siggen_type** (ACVP_CTX *ctx, ACVP_RSA_SIG_TYPE type)
- ACVP_RESULT **acvp_enable_rsa_sigver_type** (ACVP_CTX *ctx, ACVP_RSA_SIG_TYPE type)

- ACVP_RESULT **acvp_enable_rsa_siggen_caps_parm** (ACVP_CTX *ctx, ACVP_RSA_SIG_TYPE sig_↩ type, int mod, char *hash_name, int salt_len)
- ACVP_RESULT **acvp_enable_rsa_sigver_caps_parm** (ACVP_CTX *ctx, ACVP_RSA_SIG_TYPE sig_↩ type, int mod, char *hash_name, int salt_len)
- ACVP_RESULT **acvp_enable_ecdsa_cap_parm** (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_ECDSA_PARM param, char *value)
- ACVP_RESULT acvp_enable_rsa_keygen_exp_parm (ACVP_CTX *ctx, ACVP_RSA_PARM param, char *value)

  acvp_enable_rsa_bignum_parm() allows an application to specify BIGNUM operational parameters to be used for a given RSA alg during a test session with the ACVP server.
- ACVP_RESULT **acvp_enable_rsa_sigver_exp_parm** (ACVP_CTX *ctx, ACVP_RSA_PARM param, char *value)
- ACVP_RESULT acvp_enable_rsa_keygen_primes_parm (ACVP_CTX *ctx, ACVP_RSA_KEYGEN_MODE mode, int mod, char *name)

  acvp_enable_rsa_primes_parm() allows an application to specify RSA key generation provable or probable primes parameters for use during a test session with the ACVP server.
- ACVP_RESULT acvp_enable_hmac_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_RESULT(*crypto↩ _handler)(ACVP_TEST_CASE *test_case))

  acvp_enable_hmac_cap() allows an application to specify an HMAC capability to be tested by the ACVP server.
- ACVP_RESULT acvp_enable_hmac_cap_parm (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_HMAC_PARM parm, int value)

  acvp_enable_hmac_cap_parm() allows an application to specify operational parameters for use during a test session with the ACVP server.
- ACVP_RESULT acvp_enable_cmac_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_RESULT(*crypto↩ _handler)(ACVP_TEST_CASE *test_case))

  acvp_enable_cmac_cap() allows an application to specify an CMAC capability to be tested by the ACVP server.
- ACVP_RESULT acvp_enable_cmac_cap_parm (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_CMAC_PARM parm, int value)

  acvp_enable_cmac_cap_parm() allows an application to specify operational parameters for use during a test session with the ACVP server.
- ACVP_RESULT acvp_enable_kdf135_tls_cap (ACVP_CTX *ctx, ACVP_KDF135_TLS_METHOD method, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))

  acvp_enable_kdf135_*_cap() allows an application to specify a kdf cipher capability to be tested by the ACVP server.
- ACVP_RESULT **acvp_enable_kdf135_snmp_cap** (ACVP_CTX *ctx, ACVP_RESULT(*crypto_↩ handler)(ACVP_TEST_CASE *test_case))
- ACVP_RESULT **acvp_enable_kdf135_ssh_cap** (ACVP_CTX *ctx, ACVP_RESULT(*crypto_handler)(ACVP_TEST_CASE *test_case))
- ACVP_RESULT **acvp_enable_kdf135_srtp_cap** (ACVP_CTX *ctx, ACVP_RESULT(*crypto_↩ handler)(ACVP_TEST_CASE *test_case))
- ACVP_RESULT acvp_enable_kdf135_tls_cap_parm (ACVP_CTX *ctx, ACVP_CIPHER cap, ACVP_KDF135_TLS_METHOD method, ACVP_KDF135_TLS_CAP_PARM param)

  acvp_enable_kdf135_tls_cap_parm() allows an application to specify operational parameters to be used during a test session with the ACVP server.
- ACVP_RESULT acvp_enable_kdf135_ssh_cap_parm (ACVP_CTX *ctx, ACVP_CIPHER cap, ACVP_KDF135_SSH_METHOD method, ACVP_KDF135_SSH_CAP_PARM param)

  acvp_enable_kdf135_ssh_cap_parm() allows an application to specify operational parameters to be used during a test session with the ACVP server.
- ACVP_RESULT acvp_enable_kdf135_srtp_cap_parm (ACVP_CTX *ctx, ACVP_CIPHER cap, ACVP_KDF135_SRTP_PARAM param, int value)

  acvp_enable_kdf135_srtp_cap_parm() allows an application to specify operational parameters to be used during a test session with the ACVP server.
- ACVP_RESULT acvp_enable_prereq_cap (ACVP_CTX *ctx, ACVP_CIPHER cipher, ACVP_PREREQ_ALG pre_req_cap, char *value)

  acvp_enable_prereq_cap() allows an application to specify a prerequisite for a cipher capability that was previously registered.

- ACVP_RESULT acvp_create_test_session (ACVP_CTX **ctx, ACVP_RESULT(*progress_cb)(char *msg), ACVP_LOG_LVL level)

  acvp_create_test_session() creates a context that can be used to commence a test session with an ACVP server.
- ACVP_RESULT acvp_free_test_session (ACVP_CTX *ctx)

  acvp_free_test_session() releases the memory associated with an ACVP_CTX.
- ACVP_RESULT acvp_set_server (ACVP_CTX *ctx, char *server_name, int port)

  acvp_set_server() specifies the ACVP server and TCP port number to use when contacting the server.
- ACVP_RESULT acvp_set_path_segment (ACVP_CTX *ctx, char *path_segment)

  acvp_set_path_segment() specifies the URI prefix used by the ACVP server.
- ACVP_RESULT acvp_set_cacerts (ACVP_CTX *ctx, char *ca_file)

  acvp_set_cacerts() specifies PEM encoded certificates to use as the root trust anchors for establishing the TLS session with the ACVP server.
- ACVP_RESULT acvp_set_certkey (ACVP_CTX *ctx, char *cert_file, char *key_file)

  acvp_set_certkey() specifies PEM encoded certificate and private key to use for establishing the TLS session with the ACVP server.
- void acvp_mark_as_sample (ACVP_CTX *ctx)

  acvp_mark_as_sample() marks the registration as a sample.
- ACVP_RESULT acvp_register (ACVP_CTX *ctx)

  acvp_register() registers the DUT with the ACVP server.
- ACVP_RESULT acvp_process_tests (ACVP_CTX *ctx)

  acvp_process_tests() performs the ACVP testing procedures.
- ACVP_RESULT acvp_set_vendor_info (ACVP_CTX *ctx, const char *vendor_name, const char *vendor_url, const char *contact_name, const char *contact_email)

  acvp_set_vendor_info() specifies the vendor attributes for the test session.
- ACVP_RESULT acvp_set_module_info (ACVP_CTX *ctx, const char *module_name, const char *module_type, const char *module_version, const char *module_description)

  acvp_set_module_info() specifies the crypto module attributes for the test session.
- ACVP_RESULT acvp_check_test_results (ACVP_CTX *ctx)

  acvp_check_test_results() allows the application to fetch vector set results from the server during a test session.
- ACVP_RESULT acvp_set_2fa_callback (ACVP_CTX *ctx, ACVP_RESULT(*totp_cb)(char **token))

  acvp_set_2fa_callback() sets a callback function which will create or obtain a TOTP password for the second part of the two-factor authentication.
- ACVP_RESULT **acvp_bin_to_hexstr** (const unsigned char *src, unsigned int src_len, unsigned char *dest)
- ACVP_RESULT **acvp_hexstr_to_bin** (const unsigned char *src, unsigned char *dest, int dest_max)
- char * acvp_lookup_error_string (ACVP_RESULT rv)

  acvp_lookup_error_string() is a utility that returns a more descriptive string for an ACVP_RESULT error code
- void **acvp_cleanup** (void)

4.2.1 Detailed Description

This is the public header file to be included by applications using libacvp.

4.2.2 Enumeration Type Documentation

4.2.2.1 acvp_result

```
enum acvp_result
```

Enumerator

ACVP_MALLOC_FAIL	Error allocating memory
ACVP_NO_CTX	No valid context
ACVP_TRANSPORT_FAIL	Error exchanging data with server

4.2.3 Function Documentation

4.2.3.1 acvp_check_test_results()

```
ACVP_RESULT acvp_check_test_results (
            ACVP_CTX * ctx )
```

acvp_check_test_results() allows the application to fetch vector set results from the server during a test session.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.

Returns

ACVP_RESULT

4.2.3.2 acvp_create_test_session()

```
ACVP_RESULT acvp_create_test_session (
            ACVP_CTX ** ctx,
            ACVP_RESULT(*)(char *msg) progress_cb,
            ACVP_LOG_LVL level )
```

acvp_create_test_session() creates a context that can be used to commence a test session with an ACVP server.

This function should be called first to create a context that is used to manage all the API calls into libacvp. The context should be released after the test session has completed by invoking acvp_free_test_session().

When creating a new test session, a function pointer can be provided to receive logging messages from libacvp. The application can then forward the log messages to any logging service it desires, such as syslog.

Parameters

ctx	Address of pointer to unallocated ACVP_CTX.
progress_cb	Address of function to receive log messages from libacvp.

Returns

ACVP_RESULT

4.2.3.3 acvp_enable_cmac_cap()

```
ACVP_RESULT acvp_enable_cmac_cap (
             ACVP_CTX * ctx,
             ACVP_CIPHER cipher,
             ACVP_RESULT(*)(ACVP_TEST_CASE *test_case) crypto_handler )
```

acvp_enable_cmac_cap() allows an application to specify an CMAC capability to be tested by the ACVP server.

This function should be called to enable crypto capabilities for cmac algorithms that will be tested by the ACVP server. This includes CMAC-AES-128, CMAC-AES-192, CMAC-AES-256, etc. This function may be called multiple times to specify more than one crypto capability.

When the application enables a crypto capability, such as CMAC-AES-128, it also needs to specify a callback function that will be used by libacvp when that crypto capability is needed during a test session.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
crypto_handler	Address of function implemented by application that is invoked by libacvp when the crypto capablity is needed during a test session.

Returns

ACVP_RESULT

4.2.3.4 acvp_enable_cmac_cap_parm()

```
ACVP_RESULT acvp_enable_cmac_cap_parm (
             ACVP_CTX * ctx,
             ACVP_CIPHER cipher,
             ACVP_CMAC_PARM parm,
             int value )
```

acvp_enable_cmac_cap_parm() allows an application to specify operational parameters for use during a test session with the ACVP server.

This function allows the application to specify parameters for use when registering CMAC capability with the server.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
parm	ACVP_CMAC_PARM enum value specifying parameter
value	Supported value for the corresponding parameter

Returns

ACVP_RESULT

4.2.3.5 acvp_enable_drbg_cap()

```
ACVP_RESULT acvp_enable_drbg_cap (
             ACVP_CTX * ctx,
             ACVP_CIPHER cipher,
             ACVP_RESULT(*)(ACVP_TEST_CASE *test_case) crypto_handler )
```

acvp_enable_drbg_cap() allows an application to specify a hash capability to be tested by the ACVP server.

This function should be called to enable crypto capabilities for hash algorithms that will be tested by the ACVP server. This includes HASHDRBG, HMACDRBG, CTRDRBG. This function may be called multiple times to specify more than one crypto capability.

When the application enables a crypto capability, such as ACVP_HASHDRBG, it also needs to specify a callback function that will be used by libacvp when that crypto capability is needed during a test session.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
crypto_handler	Address of function implemented by application that is invoked by libacvp when the crypto capablity is needed during a test session.

Returns

ACVP_RESULT

4.2.3.6 acvp_enable_drbg_cap_parm()

```
ACVP_RESULT acvp_enable_drbg_cap_parm (
             ACVP_CTX * ctx,
             ACVP_CIPHER cipher,
             ACVP_DRBG_MODE mode,
             ACVP_DRBG_PARM param,
             int value )
```

acvp_enable_drbg_cap_parm() allows an application to specify operational parameters to be used for a given D↩ RBG alg during a test session with the ACVP server.

This function should be called to enable crypto capabilities for hash capabilities that will be tested by the ACVP server. This includes HASHDRBG, HMACDRBG, CTRDRBG. This function may be called multiple times to specify more than one crypto capability.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
mode	ACVP_DRBG_MODE enum value specifying mode. An example would be ACVP_DRBG_SHA_1
param	ACVP_DRBG_PARM enum value identifying the algorithm parameter that is being specified. An example would be prediction resistance.
value	the value corresponding to the parameter being set

Returns

ACVP_RESULT

4.2.3.7 acvp_enable_drbg_length_cap()

```
ACVP_RESULT acvp_enable_drbg_length_cap (
            ACVP_CTX * ctx,
            ACVP_CIPHER cipher,
            ACVP_DRBG_MODE mode,
            ACVP_DRBG_PARM param,
            int min,
            int step,
            int max )
```

acvp_enable_drbg_length_cap() allows an application to register a DRBG capability length-based paramter.

This function should be used to register a length-based parameter for a DRBG capability. An example would be entropy, nonce, perso where a minimum, step, and maximum can be specified.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
mode	ACVP_DRBG_MODE enum value specifying mode. An example would be ACVP_DRBG_SHA_1
param	ACVP_DRBG_PARM enum value specifying paramter. An example would be ACVP_DRBG_ENTROPY_LEN
min	minimum value
step	increment value
max	maximum value

Returns

ACVP_RESULT

4.2.3.8 acvp_enable_drbg_prereq_cap()

```
ACVP_RESULT acvp_enable_drbg_prereq_cap (
              ACVP_CTX * ctx,
              ACVP_CIPHER cipher,
              ACVP_DRBG_MODE mode,
              ACVP_PREREQ_ALG pre_req,
              char * value )
```

acvp_enable_drbg_prereq_cap() allows an application to specify a prerequisite algorithm for a given DRBG during a test session with the ACVP server.

This function should be called to enable a prerequisite for a DRBG capability that will be tested by the server.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
mode	ACVP_DRBG_MODE enum value specifying mode. An example would be ACVP_DRBG_SHA_1
pre_req	ACVP_PREREQ_ALG enum that the specified cipher/mode depends on
value	"same" or number

Returns

ACVP_RESULT

4.2.3.9 acvp_enable_dsa_cap()

```
ACVP_RESULT acvp_enable_dsa_cap (
              ACVP_CTX * ctx,
              ACVP_CIPHER cipher,
              ACVP_RESULT(*)(ACVP_TEST_CASE *test_case) crypto_handler )
```

acvp_enable_dsa_cap()

This function should be used to enable DSA capabilities. Specific modes and parameters can use acvp_enable_↩ rsa_cap_parm, acvp_enable_rsa_bignum_parm, acvp_enable_rsa_primes_parm depending on the need.

When the application enables a crypto capability, such as RSA, it also needs to specify a callback function that will be used by libacvp when that crypto capability is needed during a test session.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
crypto_handler	Address of function implemented by application that is invoked by libacvp when the crypto capability is needed during a test session.

Returns

ACVP_RESULT

4.2.3.10 acvp_enable_dsa_cap_parm()

```
ACVP_RESULT acvp_enable_dsa_cap_parm (
            ACVP_CTX * ctx,
            ACVP_CIPHER cipher,
            ACVP_DSA_MODE mode,
            ACVP_DSA_PARM param,
            int value )
```

acvp_enable_dsa_cap_parm() allows an application to specify operational parameters to be used for a given hash alg during a test session with the ACVP server.

This function should be called to enable crypto capabilities for hash capabilities that will be tested by the ACVP server. This includes HASHDRBG, HMACDRBG, CTRDRBG. This function may be called multiple times to specify more than one crypto capability.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
mode	ACVP_DSA_MODE enum value specifying mode. An example would be ACVP_DSA_MODE_PQGGEN
param	ACVP_DSA_PARM enum value identifying the algorithm parameter that is being specified. An example would be ACVP_DSA_GENPQ.
value	the value corresponding to the parameter being set

Returns

ACVP_RESULT

4.2.3.11 acvp_enable_hash_cap()

```
ACVP_RESULT acvp_enable_hash_cap (
            ACVP_CTX * ctx,
            ACVP_CIPHER cipher,
            ACVP_RESULT(*)(ACVP_TEST_CASE *test_case) crypto_handler )
```

acvp_enable_hash_cap() allows an application to specify a hash capability to be tested by the ACVP server.

This function should be called to enable crypto capabilities for hash algorithms that will be tested by the ACVP server. This includes SHA-1, SHA-256, SHA-384, etc. This function may be called multiple times to specify more than one crypto capability.

When the application enables a crypto capability, such as SHA-1, it also needs to specify a callback function that will be used by libacvp when that crypto capability is needed during a test session.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
crypto_handler	Address of function implemented by application that is invoked by libacvp when the crypto capablity is needed during a test session.

Returns

ACVP_RESULT

4.2.3.12 acvp_enable_hash_cap_parm()

```
ACVP_RESULT acvp_enable_hash_cap_parm (
            ACVP_CTX * ctx,
            ACVP_CIPHER cipher,
            ACVP_HASH_PARM param,
            int value )
```

acvp_enable_hash_cap_parm() allows an application to specify operational parameters to be used for a given hash alg during a test session with the ACVP server.

This function should be called to enable crypto capabilities for hash capabilities that will be tested by the ACVP server. This includes SHA-1, SHA-256, SHA-384, etc.

This function may be called multiple times to specify more than one crypto parameter value for the hash algorithm. The ACVP_CIPHER value passed to this function should already have been setup by invoking acvp_enable_hash_cap().

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
param	ACVP_HASH_PARM enum value identifying the algorithm parameter that is being specified. An example would be a flag indicating if empty input values are allowed.
value	the value corresponding to the parameter being set

Returns

ACVP_RESULT

4.2.3.13 acvp_enable_hmac_cap()

```
ACVP_RESULT acvp_enable_hmac_cap (
            ACVP_CTX * ctx,
```

```
            ACVP_CIPHER cipher,
            ACVP_RESULT(*)(ACVP_TEST_CASE *test_case) crypto_handler )
```

acvp_enable_hmac_cap() allows an application to specify an HMAC capability to be tested by the ACVP server.

This function should be called to enable crypto capabilities for hmac algorithms that will be tested by the ACVP server. This includes HMAC-SHA-1, HMAC-SHA2-256, HMAC-SHA2-384, etc. This function may be called multiple times to specify more than one crypto capability.

When the application enables a crypto capability, such as HMAC-SHA-1, it also needs to specify a callback function that will be used by libacvp when that crypto capability is needed during a test session.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
crypto_handler	Address of function implemented by application that is invoked by libacvp when the crypto capablity is needed during a test session.

Returns

ACVP_RESULT

4.2.3.14 acvp_enable_hmac_cap_parm()

```
ACVP_RESULT acvp_enable_hmac_cap_parm (
            ACVP_CTX * ctx,
            ACVP_CIPHER cipher,
            ACVP_HMAC_PARM parm,
            int value )
```

acvp_enable_hmac_cap_parm() allows an application to specify operational parameters for use during a test session with the ACVP server.

This function allows the application to specify parameters for use when registering HMAC capability with the server.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
parm	ACVP_HMAC_PARM enum value specifying parameter
value	Supported value for the corresponding parameter

Returns

ACVP_RESULT

4.2.3.15 acvp_enable_kdf135_srtp_cap_parm()

```
ACVP_RESULT acvp_enable_kdf135_srtp_cap_parm (
             ACVP_CTX * ctx,
             ACVP_CIPHER cap,
             ACVP_KDF135_SRTP_PARAM param,
             int value )
```

acvp_enable_kdf135_srtp_cap_parm() allows an application to specify operational parameters to be used during a test session with the ACVP server.

This function should be called after acvp_enable_kdf135_srtp_cap() to specify the parameters for the corresponding KDF.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cap	ACVP_CIPHER enum value identifying the crypto capability, here it will always be ACVP_KDF135_SRTP
param	acvp_enable_kdf135_srtp_cap_parm enum value specifying parameter
value	integer value for parameter

Returns

ACVP_RESULT

4.2.3.16 acvp_enable_kdf135_ssh_cap_parm()

```
ACVP_RESULT acvp_enable_kdf135_ssh_cap_parm (
             ACVP_CTX * ctx,
             ACVP_CIPHER cap,
             ACVP_KDF135_SSH_METHOD method,
             ACVP_KDF135_SSH_CAP_PARM param )
```

acvp_enable_kdf135_ssh_cap_parm() allows an application to specify operational parameters to be used during a test session with the ACVP server.

This function should be called after acvp_enable_kdf135_tls_cap() to specify the parameters for the corresponding KDF.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cap	ACVP_CIPHER enum value identifying the crypto capability, here it will always be ACVP_KDF135_SSH
method	ACVP_KDF135_SSH_METHOD enum value specifying method type
param	ACVP_KDF135_SSH_CAP_PARM enum value

Returns

ACVP_RESULT

4.2.3.17 acvp_enable_kdf135_tls_cap()

```
ACVP_RESULT acvp_enable_kdf135_tls_cap (
              ACVP_CTX * ctx,
              ACVP_KDF135_TLS_METHOD method,
              ACVP_RESULT(*)(ACVP_TEST_CASE *test_case) crypto_handler )
```

acvp_enable_kdf135_*_cap() allows an application to specify a kdf cipher capability to be tested by the ACVP server.

When the application enables a crypto capability, such as KDF135_TLS, it also needs to specify a callback function that will be used by libacvp when that crypto capability is needed during a test session.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
crypto_handler	Address of function implemented by application that is invoked by libacvp when the crypto capablity is needed during a test session.

Returns

ACVP_RESULT

4.2.3.18 acvp_enable_kdf135_tls_cap_parm()

```
ACVP_RESULT acvp_enable_kdf135_tls_cap_parm (
              ACVP_CTX * ctx,
              ACVP_CIPHER cap,
              ACVP_KDF135_TLS_METHOD method,
              ACVP_KDF135_TLS_CAP_PARM param )
```

acvp_enable_kdf135_tls_cap_parm() allows an application to specify operational parameters to be used during a test session with the ACVP server.

This function should be called after acvp_enable_kdf135_tls_cap() to specify the parameters for the corresponding KDF.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cap	ACVP_CIPHER enum value identifying the crypto capability, here it will always be ACVP_KDF135_TLS
method	ACVP_KDF135_TLS_METHOD enum value specifying method type
param	ACVP_KDF135_TLS_CAP_PARM enum value

Returns

ACVP_RESULT

4.2.3.19 acvp_enable_prereq_cap()

```
ACVP_RESULT acvp_enable_prereq_cap (
             ACVP_CTX * ctx,
             ACVP_CIPHER cipher,
             ACVP_PREREQ_ALG pre_req_cap,
             char * value )
```

acvp_enable_prereq_cap() allows an application to specify a prerequisite for a cipher capability that was previously registered.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability that has a prerequisite
pre_req_alg	ACVP_PREREQ_ALG enum identifying the prerequisite
value	value for specified prerequisite

Returns

ACVP_RESULT

4.2.3.20 acvp_enable_rsa_keygen_cap()

```
ACVP_RESULT acvp_enable_rsa_keygen_cap (
             ACVP_CTX * ctx,
             ACVP_CIPHER cipher,
             ACVP_RESULT(*)(ACVP_TEST_CASE *test_case) crypto_handler )
```

acvp_enable_rsa_*_cap()

This function should be used to enable RSA capabilities. Specific modes and parameters can use acvp_enable_↩ rsa_cap_parm, acvp_enable_rsa_bignum_parm, acvp_enable_rsa_primes_parm depending on the need.

When the application enables a crypto capability, such as RSA, it also needs to specify a callback function that will be used by libacvp when that crypto capability is needed during a test session.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
crypto_handler	Address of function implemented by application that is invoked by libacvp when the crypto capablity is needed during a test session.

Returns

ACVP_RESULT

4.2.3.21 acvp_enable_rsa_keygen_cap_parm()

```
ACVP_RESULT acvp_enable_rsa_keygen_cap_parm (
            ACVP_CTX * ctx,
            ACVP_RSA_PARM param,
            int value )
```

acvp_enable_rsa_*_cap_parm() allows an application to specify operational parameters to be used for a given RSA alg during a test session with the ACVP server.

This function should be called to enable parameters for RSA capabilities that will be tested by the ACVP server. This function may be called multiple times to specify more than one crypto capability.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
mode	ACVP_RSA_MODE enum value specifying mode. An example would be ACVP_RSA_MODE_KEYGEN
param	ACVP_RSA_PARM enum value identifying the algorithm parameter that is being specified. An example would be public exponent
value	the value corresponding to the parameter being set

Returns

ACVP_RESULT

4.2.3.22 acvp_enable_rsa_keygen_exp_parm()

```
ACVP_RESULT acvp_enable_rsa_keygen_exp_parm (
            ACVP_CTX * ctx,
            ACVP_RSA_PARM param,
            char * value )
```

acvp_enable_rsa_bignum_parm() allows an application to specify BIGNUM operational parameters to be used for a given RSA alg during a test session with the ACVP server.

This function behaves the same as acvp_enable_rsa_cap_parm() but instead allows the application to specify a BIGNUM parameter

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.

Parameters

mode	ACVP_RSA_MODE enum value specifying mode. An example would be ACVP_RSA_MODE_KEYGEN
param	ACVP_RSA_PARM enum value identifying the algorithm parameter that is being specified. An example would be public exponent
value	BIGNUM value corresponding to the parameter being set

Returns

ACVP_RESULT

4.2.3.23 acvp_enable_rsa_keygen_primes_parm()

```
ACVP_RESULT acvp_enable_rsa_keygen_primes_parm (
            ACVP_CTX * ctx,
            ACVP_RSA_KEYGEN_MODE mode,
            int mod,
            char * name )
```

acvp_enable_rsa_primes_parm() allows an application to specify RSA key generation provable or probable primes parameters for use during a test session with the ACVP server.

The function behaves similarly to acvp_enable_rsa_cap_parm() and acvp_enable_rsa_∗_exp_parm() but allows for a modulo and hash algorithm parameter to be specified alongside the provable or probable parameter.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
mode	ACVP_RSA_MODE enum value specifying mode. In this case it will always be ACVP_RSA_MODE_KEYGEN
param	ACVP_RSA_PARM enum value identifying the algorithm parameter being specified. Here, it will be one of: ACVP_CAPS_PROV_PRIME, ACVP_CAPS_PROB_PRIME, or ACVP_CAPS_PROV_PROB_PRIME
mod	Supported RSA modulo value for probable or provable prime generation
hash	The corresponding supported hash algorithm for probable or provable prime generation

Returns

ACVP_RESULT

4.2.3.24 acvp_enable_sym_cipher_cap()

```
ACVP_RESULT acvp_enable_sym_cipher_cap (
            ACVP_CTX * ctx,
```

```
            ACVP_CIPHER cipher,
            ACVP_SYM_CIPH_DIR dir,
            ACVP_SYM_CIPH_KO keying_options,
            ACVP_SYM_CIPH_IVGEN_SRC ivgen_source,
            ACVP_SYM_CIPH_IVGEN_MODE ivgen_mode,
            ACVP_RESULT(*)(ACVP_TEST_CASE *test_case) crypto_handler )
```

acvp_enable_sym_cipher_cap() allows an application to specify a symmetric cipher capability to be tested by the ACVP server.

This function should be called to enable crypto capabilities for symmetric ciphers that will be tested by the ACVP server. This includes AES and 3DES. This function may be called multiple times to specify more than one crypto capability, such as AES-CBC, AES-CTR, AES-GCM, etc.

When the application enables a crypto capability, such as AES-GCM, it also needs to specify a callback function that will be used by libacvp when that crypto capability is needed during a test session.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
dir	ACVP_SYM_CIPH_DIR enum value identifying the crypto operation (e.g. encrypt or decrypt).
keying_option	ACVP_SYM_CIPH_KO enum value identifying the TDES keying options
ivgen_source	The source of the IV used by the crypto module (e.g. internal or external)
ivgen_mode	The IV generation mode
crypto_handler	Address of function implemented by application that is invoked by libacvp when the crypto capablity is needed during a test session.

Returns

ACVP_RESULT

4.2.3.25 acvp_enable_sym_cipher_cap_parm()

```
ACVP_RESULT acvp_enable_sym_cipher_cap_parm (
            ACVP_CTX * ctx,
            ACVP_CIPHER cipher,
            ACVP_SYM_CIPH_PARM parm,
            int length )
```

acvp_enable_sym_cipher_cap_parm() allows an application to specify length-based operational parameters to be used for a given cipher during a test session with the ACVP server.

This function should be called to enable crypto capabilities for symmetric ciphers that will be tested by the ACVP server. This includes AES and 3DES.

This function may be called multiple times to specify more than one crypto parameter value for the cipher. For instance, if cipher supports plaintext lengths of 0, 128, and 136 bits, then this function would be called three times. Once for 0, once for 128, and once again for 136. The ACVP_CIPHER value passed to this function should already have been setup by invoking acvp_enable_sym_cipher_cap() for that cipher earlier.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
parm	ACVP_SYM_CIPH_PARM enum value identifying the algorithm parameter that is being specified. An example would be the supported plaintext length of the algorithm.
length	The length value for the symmetric cipher parameter being set

Returns

ACVP_RESULT

4.2.3.26 acvp_enable_sym_cipher_cap_value()

```
ACVP_RESULT acvp_enable_sym_cipher_cap_value (
            ACVP_CTX * ctx,
            ACVP_CIPHER cipher,
            ACVP_SYM_CIPH_PARM param,
            int value )
```

acvp_enable_sym_cipher_cap_parm() allows an application to specify non length-based operational parameters to be used for a given cipher during a test session with the ACVP server.

This function should be called to enable crypto capabilities for symmetric ciphers that will be tested by the ACVP server. This includes AES and 3DES.

This function may be called multiple times to specify more than one crypto parameter value for the cipher. The ACVP_CIPHER value passed to this function should already have been setup by invoking acvp_enable_sym_cipher_cap() for that cipher earlier.

Parameters

ctx	Address of pointer to a previously allocated ACVP_CTX.
cipher	ACVP_CIPHER enum value identifying the crypto capability.
parm	ACVP_SYM_CIPH_PARM enum value identifying the algorithm parameter that is being specified. An example would be the supported key wrap values
value	The length value for the symmetric cipher parameter being set

Returns

ACVP_RESULT

4.2.3.27 acvp_free_test_session()

```
ACVP_RESULT acvp_free_test_session (
            ACVP_CTX * ctx )
```

acvp_free_test_session() releases the memory associated with an ACVP_CTX.

This function will free an ACVP_CTX. Failure to invoke this function will result in a memory leak in the application layer. This function should be invoked after a test session has completed and a reference to the context is no longer needed.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.
level	Select the debug level, see ACVP_LOG_LVL

Returns

ACVP_RESULT

4.2.3.28 acvp_lookup_error_string()

```
char* acvp_lookup_error_string (
            ACVP_RESULT rv )
```

acvp_lookup_error_string() is a utility that returns a more descriptive string for an ACVP_RESULT error code

Parameters

rv	ACVP_RESULT error code

Returns

(char *) error string

4.2.3.29 acvp_mark_as_sample()

```
void acvp_mark_as_sample (
            ACVP_CTX * ctx )
```

acvp_mark_as_sample() marks the registration as a sample.

This function sets a flag that will allow the client to retrieve the correct answers later on, allowing for comparison and debugging.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.

4.2.3.30 acvp_process_tests()

```
ACVP_RESULT acvp_process_tests (
            ACVP_CTX * ctx )
```

acvp_process_tests() performs the ACVP testing procedures.

This function will commence the test session after the DUT has been registered with the ACVP server. This function should be invoked after acvp_register() finishes. When invoked, this function will download the vector sets from the ACVP server, process the vectors, and upload the results to the server.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.

Returns

ACVP_RESULT

4.2.3.31 acvp_register()

```
ACVP_RESULT acvp_register (
            ACVP_CTX * ctx )
```

acvp_register() registers the DUT with the ACVP server.

This function is used to regitser the DUT with the server. Registration allows the DUT to advertise it's capabilities to the server. The server will respond with a set of vector set identifiers that the client will need to process.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.

Returns

ACVP_RESULT

4.2.3.32 acvp_set_2fa_callback()

```
ACVP_RESULT acvp_set_2fa_callback (
            ACVP_CTX * ctx,
            ACVP_RESULT(*)(char **token) totp_cb )
```

acvp_set_2fa_callback() sets a callback function which will create or obtain a TOTP password for the second part of the two-factor authentication.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.
totp_cb	Function that will get the TOTP password

Returns

ACVP_RESULT

4.2.3.33 acvp_set_cacerts()

```
ACVP_RESULT acvp_set_cacerts (
              ACVP_CTX * ctx,
              char * ca_file )
```

acvp_set_cacerts() specifies PEM encoded certificates to use as the root trust anchors for establishing the TLS session with the ACVP server.

ACVP uses TLS as the transport. In order to verify the identity of the ACVP server, the TLS stack requires one or more root certificates that can be used to verify the identify of the ACVP TLS certificate during the TLS handshake. These root certificates are set using this function. They must be PEM encoded and all contained in the same file.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.
ca_file	Name of file containing all the PEM encoded X.509 certificates used as trust anchors for the TLS session.

Returns

ACVP_RESULT

4.2.3.34 acvp_set_certkey()

```
ACVP_RESULT acvp_set_certkey (
              ACVP_CTX * ctx,
              char * cert_file,
              char * key_file )
```

acvp_set_certkey() specifies PEM encoded certificate and private key to use for establishing the TLS session with the ACVP server.

ACVP uses TLS as the transport. In order for the ACVP server to verify the identity the DUT using libacvp, a certificate needs to be presented during the TLS handshake. The certificate used by libacvp needs to be trusted by the ACVP server. Otherwise the TLS handshake will fail.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.
cert_file	Name of file containing the PEM encoded X.509 certificate to use as the client identity.
key_file	Name of file containing PEM encoded private key associated with the client certificate.

Returns

ACVP_RESULT

4.2.3.35 acvp_set_module_info()

```
ACVP_RESULT acvp_set_module_info (
            ACVP_CTX * ctx,
            const char * module_name,
            const char * module_type,
            const char * module_version,
            const char * module_description )
```

acvp_set_module_info() specifies the crypto module attributes for the test session.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.
module_name	Name of the crypto module under test.
module_type	The crypto module type: software, hardware, or hybrid.
module_version	The version# of the crypto module under test.
module_description	A brief description of the crypto module under test.

Returns

ACVP_RESULT

4.2.3.36 acvp_set_path_segment()

```
ACVP_RESULT acvp_set_path_segment (
            ACVP_CTX * ctx,
            char * path_segment )
```

acvp_set_path_segment() specifies the URI prefix used by the ACVP server.

Some ACVP servers use a prefix in the URI for the path to the ACVP REST interface. Calling this function allows the path segment prefix to be specified. The value provided to this function is prepended to the path segment of the URI used for the ACVP REST calls.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.
path_segment	Value to embed in the URI path after the server name and before the ACVP well-known path.

Returns

ACVP_RESULT

4.2.3.37 acvp_set_server()

```
ACVP_RESULT acvp_set_server (
              ACVP_CTX * ctx,
              char * server_name,
              int port )
```

acvp_set_server() specifies the ACVP server and TCP port number to use when contacting the server.

This function is used to specify the hostname or IP address of the ACVP server. The TCP port number can also be specified if the server doesn't use port 443.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.
server_name	Name or IP address of the ACVP server.
port	TCP port number the server listens on.

Returns

ACVP_RESULT

4.2.3.38 acvp_set_vendor_info()

```
ACVP_RESULT acvp_set_vendor_info (
              ACVP_CTX * ctx,
              const char * vendor_name,
              const char * vendor_url,
              const char * contact_name,
              const char * contact_email )
```

acvp_set_vendor_info() specifies the vendor attributes for the test session.

Parameters

ctx	Pointer to ACVP_CTX that was previously created by calling acvp_create_test_session.
vendor_name	Name of the vendor that owns the crypto module.
vendor_url	The Vendor's URL.
contact_name	Name of contact at Vendor.
contact_email	Email of vendor contact.

Returns

ACVP_RESULT

Index