PIT API Documentation

Directory: core/crypto/pit_crypto.h

pit_keygenstate(macro)

Generate ECC keys for the PIT module.

int pit_keygenstate (size_t key_length, struct ecc_private_key *privkey, struct ecc_public_key *pubkey, int *state)

Parameters

Name	Type	Description
key_length	size_t	The length of the key to using in bytes. 256, 381, 521 bits (so X / 8 bytes) are the supported lengths
privkey	ecc_private_key *	Output for the initialized private key.
pubkey	ecc_public_key *	Output for the initialized public key.
state	int	An int to hold the numerical value of the state

Returns

Value	Description
1	return 1 on success

Description

Generates an ECC key pair and sets the state appropriately.

pit_secretkey (macro)

Generates a secret key.

int pit_secretkey (struct ecc_private_key *privkey, struct ecc_public_key *pubkey, uint8_t *secret, int *state)

Parameters

Name	Type	Description
privkey	ecc_private_key *	The private key used to generate the secret.

pubkey	ecc_public_key *	The public key used to generate the secret.
secret	uint8_t *	A non-null output buffer to hold the generated shared secret.
state	int	An int to hold the numerical value of the state

Returns

Value	Description
1	return 1 on success

Description

Generates a secret key using a private key and a public key. These should not belong to the same keypair.

pit_encryption (macro)

AES-GCM encryption. Encrypts a message into ciphertext using a secret key.

```
int pit_encryption (uint8_t *msg, size_t msg_size, uint8_t *secret, size_t secret_length, uint8_t *AESIV, size_t AESIV_SIZE, uint8_t *tag, uint8_t *ciphertext, int *state);
```

Parameters

Name	Type	Description
msg	uint8_t *	A plaintext message you would like to encrypt.
msg_size	size_t	The size of the plaintext message.
secret	uint8_t *	A secret key to use for encryption.
secret_length	size_t	The size of the secret key.
AESIV	uint8_t *	An IV to use for encryption. A 12-byte IV is best (meets NIST standards).
AESIV_SIZE	size_t	The size of the IV used for encryption.
tag	uint8_t *	The buffer to hold the GCM authentication tag. All tags will be 16 bytes.
ciphertext	uint8_t *	An empty output buffer to hold the encrypted ciphertext (The ciphertext will be the same length as the plaintext).
state	int	An int to hold the numerical value of the state

Returns

Value	Description
1	return 1 on success

Description

This API encrypts a plaintext of any size, then writes the ciphertext to the provided ciphertext buffer.

pit_decryption (macro)

AES-GCM decryption decrypts the ciphertext into plaintext using a secret key.

```
int pit_decryption(uint8_t *ciphertext, size_t ciphertext_size, uint8_t *secret, size_t secret_length, uint8_t *AESIV, size_t AESIV_SIZE, uint8_t *tag, uint8_t *plaintext, int state);
```

Parameters

Name	Type	Description
ciphertext	uint8_t *	The ciphertext you would like to decrypt.
ciphertext_size	size_t	The size of the ciphertext message.
secret	uint8_t *	A secret key to use for encryption.
secret_length	size_t	The size of the secret key.
AESIV	uint8_t *	An IV to use for encryption. A 12-byte IV is best (meets NIST standards).
AESIV_SIZE	size_t	The size of the IV used for encryption.
tag	uint8_t *	The buffer to hold the GCM authentication tag. All tags will be 16 bytes.
plaintext	uint8_t *	The buffer to hold the decrypted ciphertext (Will be the same size as the ciphertext)
state	int	An int to hold the numerical value of the state

Returns

Value	Description
1	return 1 on success

Description

This API decrypts a ciphertext and loads the plaintext to the provided plaintext buffer.

pit_OTPgen (macro)

A function to generate a random string representing OTP. Additionally, this function will encrypt that OTP using AES-GCM encryption, using the secret key for the AES.

int pit_OTPgen (uint8_t *secret, size_t secret_size, uint8_t *AESIV, size_t AESIV_SIZE, uint8_t *tag, uint8_t *OTP, size_t OTPSize, uint8_t *OTPs, int *state)

Parameters

Name	Type	Description
secret	uint8_t *	The secret key to encrypt the OTP.
secret_size	size_t	The size of the secret key.
AESIV	uint8_t *	An IV to use for encryption. A 12-byte IV is best (meets NIST standards).
AESIV_SIZE	size_t	The size of the IV used for encryption.
tag	uint8_t *	The output buffer to hold the GCM authentication tag. All tags will be 16 bytes.
OTP	uint8_t *	An output buffer to hold a randomly generated OTP.
OTPSize	size_t	The size of the OTP.
OTPs	uint8_t *	A buffer to hold the encrypted OTP
state	int	An int to hold the numerical value of the state

Returns

Value	Description
1	return 1 on success

Description

This will generate the OTP and encrypt the OTP. The OTP is generated and stored into the OTP argument, and the encrypted OTP is stored in the OTPs argument. OTPs is guaranteed to be the same size as OTP.

pit_OTPvalidation (macro)

Decrypts an encrypted OTP and validate the OTP

int pit_OTPvalidation (uint8_t * secret, size_t secret_size, uint8_t *AESIV, size_t AESIV_SIZE, uint8_t *tag, uint8_t *OTPs, size_t OTPs_size, uint8_t *valOTP, bool *result, int *state);

Parameters

Name	Type	Description
secret	uint8_t *	The secret key used to decrypt OTPs.
secret_size	size_t	The size of the secret key.
AESIV	uint8_t *	An IV to use for encryption. Must be the same as the IV provided to encrypt the OTP.

AESIV_SIZE	size_t	The size of the IV used for encryption.
tag	uint8_t *	The buffer to hold the GCM authentication tag. All tags will be 16 bytes.
OTPs	uint8_t *	Encrypted OTP to be decrypted.
OTPs_size	size_t	The size of the encrypted OTP.
valOTP	uint8_t *	OTP to be validated against the decrypted OTP.
result	bool *	A boolean value to check whether the OTP was successfully validated.
state	int	An int to hold the numerical value of the state

Returns

Value	Description
1	return 1 on success

Description

This API will validate the OTP. It does this by taking OTPs, decrypting it, and comparing it against valOTP. If valid, the function will return 1 and result will hold true. Otherwise, the function will return 0 and result will hold false.

Directory: core/pit.h

lock(macro)

Sets up needed variables and sets the system's state to lock.

int lock (uint8_t *secret)

Parameters

Name	Type	Description
secret	uint8_t *	A 32-byte empty array that will be loaded with the shared secret.

Returns

Value	Description
1	return 1 on success

Description

This API does all the operations needed for locking. It will exchange keys with another party to generate a shared secret.

unlock(macro)

Unlocks the state of the machine by validating OTP.

int unlock ()

Returns

Value	Description
1	return 1 on success

Description

Do All the unlocking operations. These operations include creating an OTP, encrypting it, and sending the ciphertext to the receiving party. The receiving party will receive this encrypted OTP and encrypt it AGAIN. They will send back the “doubly” encrypted OTP, and the lock function will validate. If the secret key is not the same, the validation will fail.

unlock(macro)

Gets the state of the system

int get_state()

Returns

Value	Description
1 to 7	The numerical value of the state of the system at the moment of calling.

Description

Get the state of the system.

get_OTPs (macro)

Get the encrypted OTP (OTPs) from the system.

int get_OTPs (uint8_t *OTPs)

Parameters

Name	Type	Description
OTPs	uint8_t *	OTPs Empty buffer to hold the encrypted OTP.

Returns

Value	Description
1	return 1 on success

Description

Get the encrypted OTP.

Directory: core/i2c/pit_i2c.h

pit_connect (macro)

Initiate a connection to the desired server. Dependent on implementation (Socket vs i2c).

int pit_connect (int desired_port)

Parameters

Name	Type	Description
desired_port	int	The desired port to connect to the server.

Returns

Value	Description
integer	Returns an int pointing to the file descriptor (socket) which can be used to send/receive from the server.

Description

This function was used for our server implementation. It can be changed or disregarded entirely - whichever you want.

keyexchangestate (macro)

On success, keyexchangestate should initialize pubkey_serv with the server's public key.

```
int keyexchangestate (uint8_t *pubkey_cli, size_t pubkey_der_length, uint8_t *pubkey_serv)
```

Parameters

Name	Type	Description
pubkey_cli	uint8_t *	A DER encoded version of the client's public key.
pubkey_der_length	size_t	Length of the DER encodes public key.
pubkey_serv	uint8_t *	An uint_8t pointer (which was already set up in the lock function), which is empty but will be overwritten to contain the server's DER encoded public key.

Returns

Value	Description
integer	return 1 on success

Description

Keyexchangestate will load the pubkey_serv variable with a public key received from the server.

send_unlock_info (macro)

Sends OTPs, AES IV, and the AES-GCM Tag for OTP encryption to the server, receives back the server's encrypted message, and tag for that message.

```
int send_unlock_info (uint8_t *OTPs, size_t OTPs_size, uint8_t *unlock_aes_iv, size_t unlock_aes_iv_size, uint8_t *OTP_tag, uint8_t *server_encrypted_message, uint8_t *server_tag)
```

Parameters

Name	Type	Description
OTPs	uint8_t *	The Encrypted OTP needs to send to the server.
OTPs_size	size_t	Size (in bytes) of the OTPs.
unlock_aes_iv	uint8_t *	The AES IV used to encrypt the OTP into OTPs.
unlock_aes_iv_size	size_t	Size (in bytes) of the unlock_aes_iv parameter.

OTP_tag	uint8_t *	Tag generated when encrypting OTP into OTPs.
server_encrypted_message	uint8_t *	An empty buffer to hold the server's response message (which will then be validated in the unlock API).
server_tag	uint8_t *	tag for the server's encrypted message.

Returns

Value	Description
integer	return 1 on success

Description

You will not need to change any of the setups for the various arguments. As long as the server_encrypted_message and server_tag contain the server's message and relevant tag by the time the function concludes, unlock will work appropriately.

receive_product_info (macro)

This API receives the product information from the server.

int receive_product_info (uint8_t *EncryptedProductID, uint8_t *EncryptedProductIDTag, size_t ProductIDSize, uint8_t *aes_iv, size_t aes_iv_size);

Parameters

Name	Type	Description
EncryptedProductID	uint8_t *	The Encrypted Product ID that received from the server.
EncryptedProductIDTag	uint8_t *	Tag generated when encrypting the Product ID.
ProductIDSize	size_t	Size (in bytes) of the encrypted Product ID
aes_iv	uint8_t *	An IV to use for encryption. Must be the same as the IV provided to encrypt the Product ID.
aes_iv_size	size_t	Size (in bytes) of the aes_iv parameter.

Returns

Value	Description
integer	return 1 on success

Description

This API is used to get all the Encrypted data regarding the Product ID. It has all the parameter that one needs to get to decrypt and verify the Product ID with the existing Product ID.