Cryptography App Documentation

Contents

Module `crypto_functions`

Functions

Function `add_header`

```
def add_header(
    file_localization: str,
    algorithm: str,
    mode: str,
    additional: bytes = None,
    shape: (<class 'int'>, <class 'int'>) = None
)
```

The function adds all the information provided as parameters to the metadata of the file specified as a path.

:param shape: if file is image, width and height :param file_localization: str :param algorithm: str :param mode: str :param additional: bytes :return: None

Function `add_padding`

```
def add_padding(
    data: bytes,
    block_size: int
)
```

The function to add padding to data given as parameter. The size of data is expended by null bytes, so it is divisible by the block_size.

:param data: bytes :param block_size: int :return: bytes

Function `cbc_decrypt`

```
def cbc_decrypt(
    ct,
        iv,
        key
    )
```

The function to decrypt data, encrypted with AES algorithm and CBC mode. The padding is removed from data after decryption. The initialization vector generated before encryption must be provided as parameter.

:param ct: bytes :param iv: bytes :param key: bytes :return: decrypted data: bytes

Function `cbc_encrypt`

```
    def cbc_encrypt(
        data,
        key_size=128
    )
```

The function to encrypt data with AES algorithm and CBC mode. The padding is added to data before encryption. Key size must be one of (128, 192, 256). Key and initialization vector is generated using system’s RNG.

:param data: bytes :param key_size: int = 128 :return: {‘ciphertext’: encrypted data, ‘key’: key used in encryption, ‘additional’: initialization vector}

Function `ctr_decrypt`

```
    def ctr_decrypt(
        data,
        nonce,
        key
    )
```

The function to decrypt data, encrypted with AES algorithm and CTR mode. The padding is removed from data after decryption. The nonce generated before encryption must be provided as parameter.

:param data: bytes :param nonce: bytes :param key: bytes :return: bytes

Function `ctr_encrypt`

```
    def ctr_encrypt(
        data,
        key_size=128
    )
```

The function to encrypt data with AES algorithm and CTR mode. Key size must be one of (128, 192, 256). Key and nonce is generated using system’s RNG.

:param data: bytes :param key_size: int :return: {‘ciphertext’: encrypted data, ‘key’: key used in encryption, ‘additional’: initialization vector}

Function `decrypt`

```
    def decrypt(
        algorithm: str,
        mode: str,
        key: bytes,
        additional: bytes,
        ct
    )
```

The function to decrypt data with given algorithm, mode and key. Raises ValueError if algorithm or mode is unknown.

:raises ValueError: :param algorithm: str :param mode: str :param key: bytes :param additional: bytes :param ct: bytes :return: Optional[bytes]

Function `ecb_decrypt`

```
def ecb_decrypt(
    ct: bytes,
    key: bytes
)
```

The function to decrypt data, encrypted with AES algorithm and ECB mode. The padding is removed from data after decryption.

:param ct: bytes :param key: bytes :return: decrypted data: bytes

Function `ecb_encrypt`

```
def ecb_encrypt(
    data: bytes,
    key_size: int = 128
)
```

The function to encrypt data with AES algorithm and ECB mode. The padding is added to data before encryption. Key size must be one of (128, 192, 256). Key is generated using system's RNG.

:param data: bytes :param key_size: int = 128 :return: {'ciphertext': encrypted data, 'key': key used in encryption, 'additional': None}

Function `encrypt`

```
def encrypt(
    algorithm: str,
    mode: str,
    key_len: int,
    data
)
```

The function to encrypt data with given algorithm and mode. Raises ValueError if algorithm or mode is unknown.

:raises ValueError: :param algorithm: str :param mode: str :param key_len: int :param data: bytes :return: None

Function `read_header`

```
def read_header(
    file_localization: str
)
```

The function reads all the information needed to decryption, such as algorithm, mode and additional data e.g. initialization vector, nonce, and returns it as dictionary.

:param file_localization: str :return: {'algorithm': algorithm, 'mode': mode, 'additional': additional data, 'shape': shape if encrypted file is image}

Function `remove_padding`

```
def remove_padding(
    data: bytes,
    block_size: int
)
```

The function to remove earlier added padding. Removes null bytes added by padder.

:param data: bytes :param block_size: int :return: bytes
