Project Part 1: Symmetric Cryptography Report

TCSS 487 Spring 2024

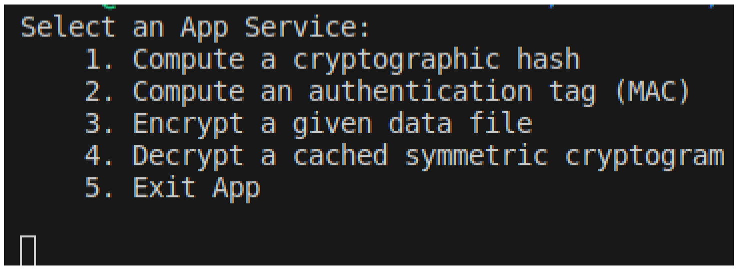
Steven Tieu

Instructions

App

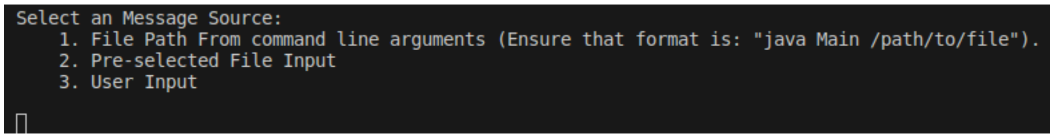
Compile the java files in any manner, preferably have the class files within the same directory as the pre-set text files.

When starting the app you will be greeted with the main selection menu, as seen below. Here you can select from 4 app services. Selecting 5 will end the app, otherwise the app will continue to loop back to this service selection menu after each service completion.



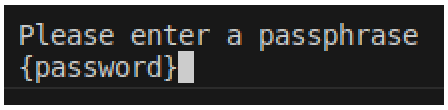
```
Select an App Service:
    1. Compute a cryptographic hash
    2. Compute an authentication tag (MAC)
    3. Encrypt a given data file
    4. Decrypt a cached symmetric cryptogram
    5. Exit App
```

Upon selecting service 1, 2, or 3. You will be prompted to select a message source, as seen below. Option 1 will search a file name specified in the command line arguments, option 2 will use a pre-selected text file, and option 3 will prompt the user for a text input on the CLI.



```
Select an Message Source:
    1. File Path From command line arguments (Ensure that format is: "java Main /path/to/file").
    2. Pre-selected File Input
    3. User Input
```

After finishing the message selection for either service 2, or 3. You will be prompted to enter a passphrase. For service 3, encryption, remember this passphrase for successful decryption in service 4.

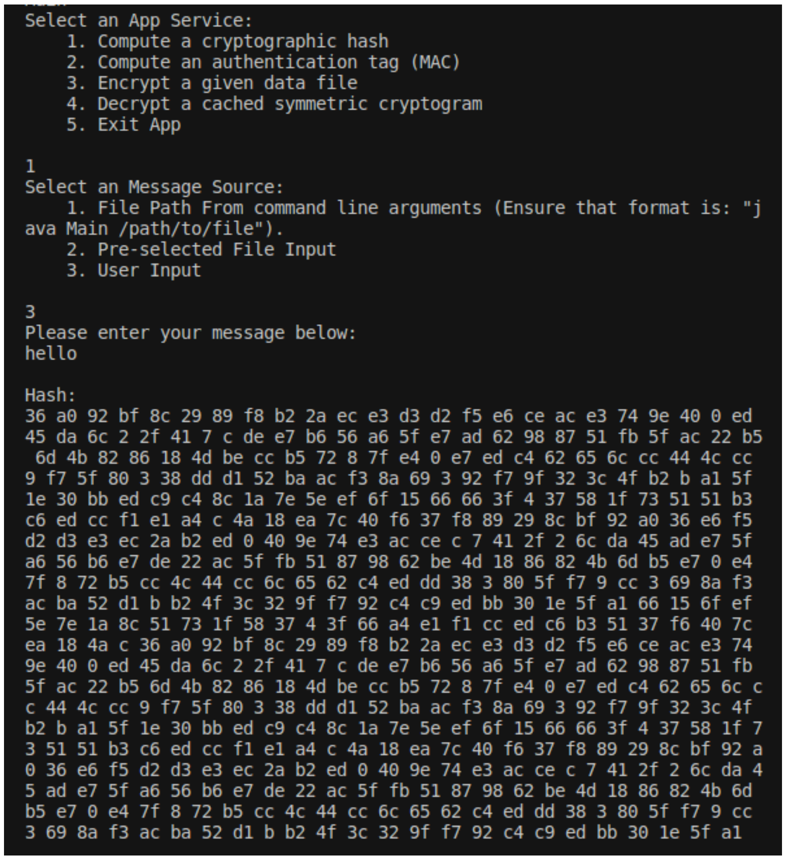


```
Please enter a passphrase
{password}
```

After receiving all inputs, the services will print out its outputs or any actions taken by the service.

Service 1: Compute a Cryptographic Hash

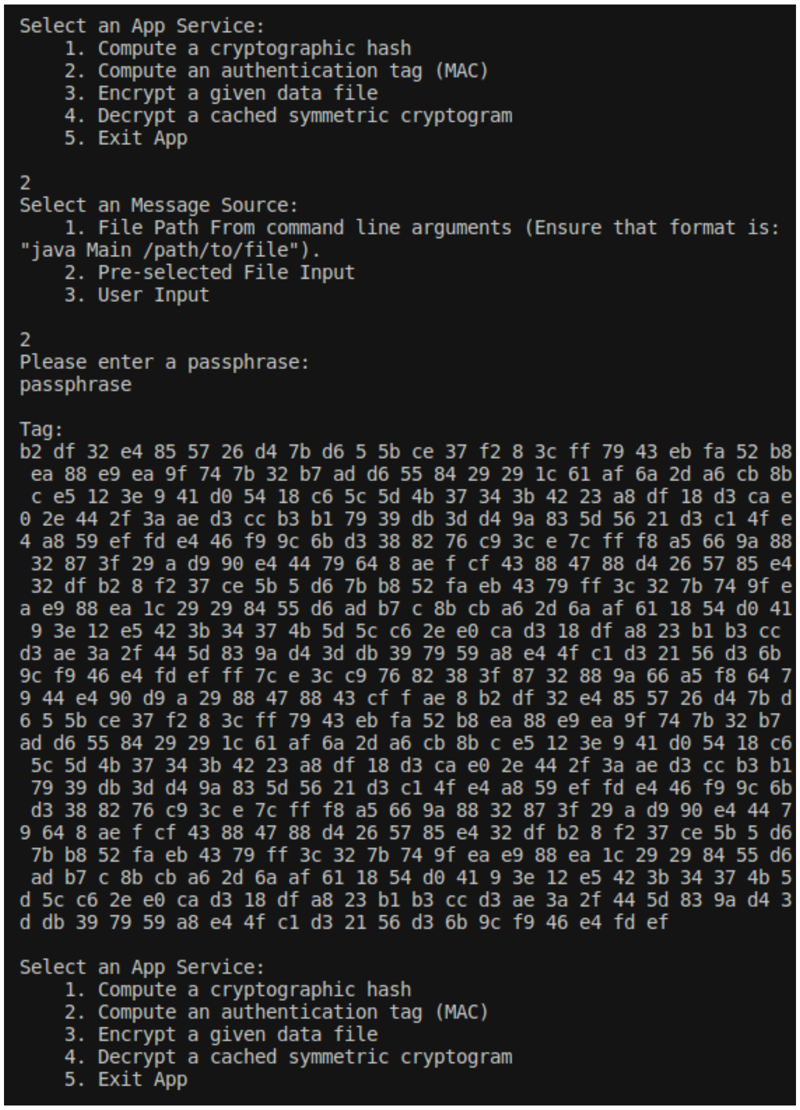
To compute a cryptographic hash function, enter 1 on the service selection prompt, then select a message source. If you use user input, you will be prompted for a message input. The hash output will be printed in bytes after all inputs are received.



```
Select an App Service:
    1. Compute a cryptographic hash
    2. Compute an authentication tag (MAC)
    3. Encrypt a given data file
    4. Decrypt a cached symmetric cryptogram
    5. Exit App

1
Select an Message Source:
    1. File Path From command line arguments (Ensure that format is: "j
ava Main /path/to/file").
    2. Pre-selected File Input
    3. User Input

3
Please enter your message below:
hello

Hash:
36 a0 92 bf 8c 29 89 f8 b2 2a ec e3 d3 d2 f5 e6 ce ac e3 74 9e 40 0 ed
45 da 6c 2 2f 41 7 c de e7 b6 56 a6 5f e7 ad 62 98 87 51 fb 5f ac 22 b5
 6d 4b 82 86 18 4d be cc b5 72 8 7f e4 0 e7 ed c4 62 65 6c cc 44 4c cc
9 f7 5f 80 3 38 dd d1 52 ba ac f3 8a 69 3 92 f7 9f 32 3c 4f b2 b a1 5f
1e 30 bb ed c9 c4 8c 1a 7e 5e ef 6f 15 66 66 3f 4 37 58 1f 73 51 51 b3
c6 ed cc f1 e1 a4 c 4a 18 ea 7c 40 f6 37 f8 89 29 8c bf 92 a0 36 e6 f5
d2 d3 e3 ec 2a b2 ed 0 40 9e 74 e3 ac ce c 7 41 2f 2 6c da 45 ad e7 5f
a6 56 b6 e7 de 22 ac 5f fb 51 87 98 62 be 4d 18 86 82 4b 6d b5 e7 0 e4
7f 8 72 b5 cc 4c 44 cc 6c 65 62 c4 ed dd 38 3 80 5f f7 9 cc 3 69 8a f3
ac ba 52 d1 b b2 4f 3c 32 9f f7 92 c4 c9 ed bb 30 1e 5f a1 66 15 6f ef
5e 7e 1a 8c 51 73 1f 58 37 4 3f 66 a4 e1 f1 cc ed c6 b3 51 37 f6 40 7c
ea 18 4a c 36 a0 92 bf 8c 29 89 f8 b2 2a ec e3 d3 d2 f5 e6 ce ac e3 74
9e 40 0 ed 45 da 6c 2 2f 41 7 c de e7 b6 56 a6 5f e7 ad 62 98 87 51 fb
5f ac 22 b5 6d 4b 82 86 18 4d be cc b5 72 8 7f e4 0 e7 ed c4 62 65 6c c
c 44 4c cc 9 f7 5f 80 3 38 dd d1 52 ba ac f3 8a 69 3 92 f7 9f 32 3c 4f
b2 b a1 5f 1e 30 bb ed c9 c4 8c 1a 7e 5e ef 6f 15 66 66 3f 4 37 58 1f 7
3 51 51 b3 c6 ed cc f1 e1 a4 c 4a 18 ea 7c 40 f6 37 f8 89 29 8c bf 92 a
0 36 e6 f5 d2 d3 e3 ec 2a b2 ed 0 40 9e 74 e3 ac ce c 7 41 2f 2 6c da 4
5 ad e7 5f a6 56 b6 e7 de 22 ac 5f fb 51 87 98 62 be 4d 18 86 82 4b 6d
b5 e7 0 e4 7f 8 72 b5 cc 4c 44 cc 6c 65 62 c4 ed dd 38 3 80 5f f7 9 cc
3 69 8a f3 ac ba 52 d1 b b2 4f 3c 32 9f f7 92 c4 c9 ed bb 30 1e 5f a1
```

Service 2: Compute an Authentication Tag

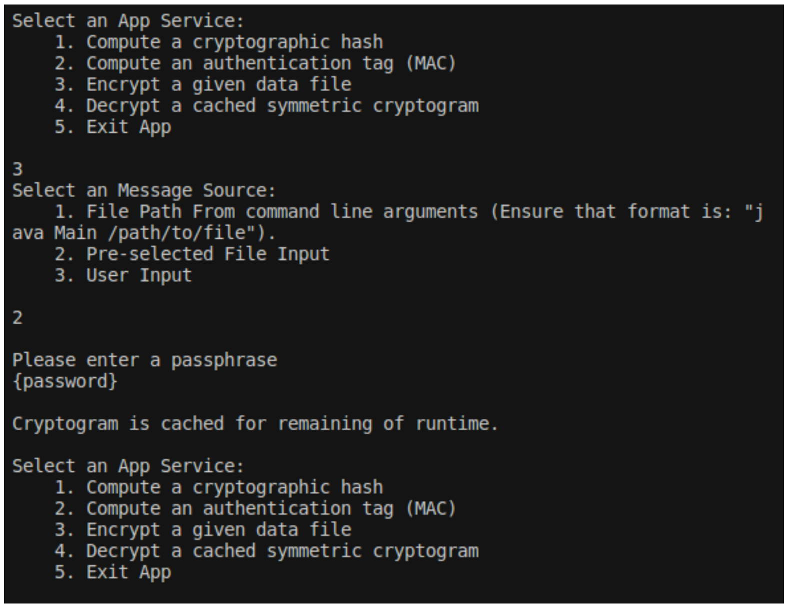
To commute an authentication tag, enter 2 on the service selection prompt, then select a message source. If you use user input, you will be prompted for a message input. A passphrase will also be prompted for user input.



```
Select an App Service:
    1. Compute a cryptographic hash
    2. Compute an authentication tag (MAC)
    3. Encrypt a given data file
    4. Decrypt a cached symmetric cryptogram
    5. Exit App

2
Select an Message Source:
    1. File Path From command line arguments (Ensure that format is:
"java Main /path/to/file").
    2. Pre-selected File Input
    3. User Input

2
Please enter a passphrase:
passphrase

Tag:
b2 df 32 e4 85 57 26 d4 7b d6 5 5b ce 37 f2 8 3c ff 79 43 eb fa 52 b8
 ea 88 e9 ea 9f 74 7b 32 b7 ad d6 55 84 29 29 1c 61 af 6a 2d a6 cb 8b
 c e5 12 3e 9 41 d0 54 18 c6 5c 5d 4b 37 34 3b 42 23 a8 df 18 d3 ca e
0 2e 44 2f 3a ae d3 cc b3 b1 79 39 db 3d d4 9a 83 5d 56 21 d3 c1 4f e
4 a8 59 ef fd e4 46 f9 9c 6b d3 38 82 76 c9 3c e 7c ff f8 a5 66 9a 88
 32 87 3f 29 a d9 90 e4 44 79 64 8 ae f cf 43 88 47 88 d4 26 57 85 e4
 32 df b2 8 f2 37 ce 5b 5 d6 7b b8 52 fa eb 43 79 ff 3c 32 7b 74 9f e
a e9 88 ea 1c 29 29 84 55 d6 ad b7 c 8b cb a6 2d 6a af 61 18 54 d0 41
 9 3e 12 e5 42 3b 34 37 4b 5d 5c c6 2e e0 ca d3 18 df a8 23 b1 b3 cc
d3 ae 3a 2f 44 5d 83 9a d4 3d db 39 79 59 a8 e4 4f c1 d3 21 56 d3 6b
9c f9 46 e4 fd ef ff 7c e 3c c9 76 82 38 3f 87 32 88 9a 66 a5 f8 64 7
9 44 e4 90 d9 a 29 88 47 88 43 cf f ae 8 b2 df 32 e4 85 57 26 d4 7b d
6 5 5b ce 37 f2 8 3c ff 79 43 eb fa 52 b8 ea 88 e9 ea 9f 74 7b 32 b7
ad d6 55 84 29 29 1c 61 af 6a 2d a6 cb 8b c e5 12 3e 9 41 d0 54 18 c6
 5c 5d 4b 37 34 3b 42 23 a8 df 18 d3 ca e0 2e 44 2f 3a ae d3 cc b3 b1
 79 39 db 3d d4 9a 83 5d 56 21 d3 c1 4f e4 a8 59 ef fd e4 46 f9 9c 6b
 d3 38 82 76 c9 3c e 7c ff f8 a5 66 9a 88 32 87 3f 29 a d9 90 e4 44 7
9 64 8 ae f cf 43 88 47 88 d4 26 57 85 e4 32 df b2 8 f2 37 ce 5b 5 d6
 7b b8 52 fa eb 43 79 ff 3c 32 7b 74 9f ea e9 88 ea 1c 29 29 84 55 d6
 ad b7 c 8b cb a6 2d 6a af 61 18 54 d0 41 9 3e 12 e5 42 3b 34 37 4b 5
d 5c c6 2e e0 ca d3 18 df a8 23 b1 b3 cc d3 ae 3a 2f 44 5d 83 9a d4 3
d db 39 79 59 a8 e4 4f c1 d3 21 56 d3 6b 9c f9 46 e4 fd ef

Select an App Service:
    1. Compute a cryptographic hash
    2. Compute an authentication tag (MAC)
    3. Encrypt a given data file
    4. Decrypt a cached symmetric cryptogram
    5. Exit App
```

Service 3: Encryption

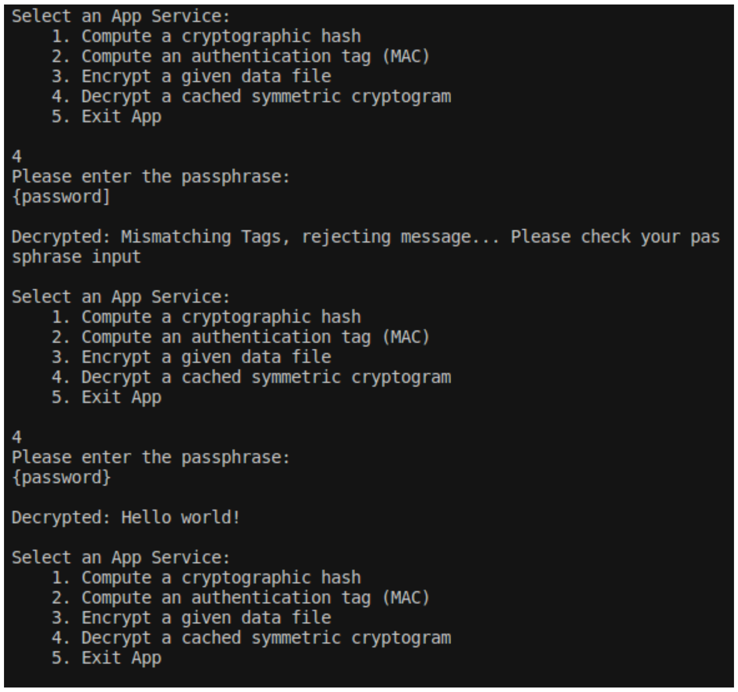
To encrypt text, enter 3 on the service selection prompt, then select a message source. If you choose user input, you will be prompted for a message input. A passphrase will also be prompted for user input. At the end of encryption, the cryptogram will be cached for the remaining runtime of the program, you can replace this cryptogram by running the encryption service again.



```
Select an App Service:
    1. Compute a cryptographic hash
    2. Compute an authentication tag (MAC)
    3. Encrypt a given data file
    4. Decrypt a cached symmetric cryptogram
    5. Exit App

3
Select an Message Source:
    1. File Path From command line arguments (Ensure that format is: "j
ava Main /path/to/file").
    2. Pre-selected File Input
    3. User Input

2

Please enter a passphrase
{password}

Cryptogram is cached for remaining of runtime.

Select an App Service:
    1. Compute a cryptographic hash
    2. Compute an authentication tag (MAC)
    3. Encrypt a given data file
    4. Decrypt a cached symmetric cryptogram
    5. Exit App
```

Service 4: Decrypt a Cached Symmetric Cryptogram

To decrypt the cached cryptogram, enter 4 on the service selection prompt. You will only be prompted for the passphrase. Entering the incorrect passphrase will result in a tag mismatch. Entering the correct passphrase will reveal the original plaintext.



```
Select an App Service:
    1. Compute a cryptographic hash
    2. Compute an authentication tag (MAC)
    3. Encrypt a given data file
    4. Decrypt a cached symmetric cryptogram
    5. Exit App

4
Please enter the passphrase:
{password]

Decrypted: Mismatching Tags, rejecting message... Please check your pas
sphrase input

Select an App Service:
    1. Compute a cryptographic hash
    2. Compute an authentication tag (MAC)
    3. Encrypt a given data file
    4. Decrypt a cached symmetric cryptogram
    5. Exit App

4
Please enter the passphrase:
{password}

Decrypted: Hello world!

Select an App Service:
    1. Compute a cryptographic hash
    2. Compute an authentication tag (MAC)
    3. Encrypt a given data file
    4. Decrypt a cached symmetric cryptogram
    5. Exit App
```

Description of Solution by Function

Kmacxof256.java

computeHash(String message)

Computes a hash function for a given message using Kmacxof256. Follows the notation in the NIST Publication 800-185 except for parameter m which is converted to a byte array KMACX. Parameter message is the message to compute the hash. Return the hash as bytes.

computeAuthTag(String message, String pw)

Computes an authentication tag for a given message using Kmacxof256 and a passphrase. Follows the notation in the NIST Publication 800-185 except for parameter m which is converted to a byte array KMACX. Parameter message is the message to compute the tag. Return the tag as bytes.

encrypt(String message, String pw)

Symmetrically encrypts a given message using a passphrase and Kmacof256. Follows the notation in the NIST Publication 800-185 except for parameter m which is converted to a byte array KMACX. Parameter message is the message to encrypt. Return the cryptogram a 2-d array of bytes.

decrypt(byte[][] cryptogram, String pw)

Symmetrically decrypts a given cryptogram using the given pass phrase. Parameter cryptogram is the symmetric cryptogram(z, c, t). z is the randomly selected number from 0 to (2^512) - 1 during encryption. c is the ciphertext. t is the authentication tag. Parameter pw is the pass phrase used for decrypting the cryptogram. Return the decrypted plaintext.

KMACXOF256(String K, String X, int L, String S)

KMACXOF256 primitive according to NIST Special Publication 800-185, the only exception being that L is the byte width rather than bit width. Additional support function used to convert between bytes and Strings in Java. Parameter K is the key string of any length between zero and 2^2040. Parameter X is the main input string, the message. Parameter L is the length of final output in bytes. Parameter S is an optional customization string. Returns output of Kmacof256 as bytes.

cShake256(byte[] X, int L, String N, String S)

Performs cShake256 function. Inspired by Github user mjosaarinen's Sha-3 implementation in C at https://github.com/mjosaarinen/tiny_sha3/tree/master. Due to the requirements of the project, this function only calls Keccak() unlike the NIST specification in SP 800-185. Specifically, only "KECCAK[512](bytepad(encode_string(N) || encode_string(S), 136) || X || 00, L)" is done here. Parameter X is the main input string, the message. Parameter L is the length of final output in bytes. Parameter N is the NIST function-name input Only values defined by NIST should be used here. Parameter S is an optional customization string. Returns the output of cShake function.

encodeString(String S)

Encode Strings into byte arrays that may be parsed unambiguously from the beginning of the string. Parameter S is the String to encode. Returns encoded byte array of S.

leftEncode(int x)

Encodes integer x as a byte array and inserts the length of the byte array in the first indices. Parameter x is the integer to encode. Returns encoded x as a byte array, length inserted at start.

rightEncode(int x)

Encodes integer x as a byte array and inserts the length of the byte array in the last indices. Parameter x is the integer to encode. Returns encoded x as a byte array, length inserted at end.

getByteSize(int x)

Detects how many bytes are needed to represent integer x. Parameter x is the integer to represent. Returns how many bytes needed to represent x.

bytePad(byte[] X, int w)

Prepends an encoding of integer w to an input byte array X then pads the result with zeros until its length in bytes is a multiple of w. Used on encoded strings. This implementation is provided by class slides. Parameter X is the encoded string, encoded from encodeString(). Parameter w is the output length multiple. Returns padded version of byte array X.

byteXor(byte x, byte y)

Performs exclusive or bit operation on the two input bytes. Normal XOR operations will not work on byte types without casting in Java. Parameter x is the first input byte. Parameter y is the second input byte. Returns is the XOR result of the two input bytes.

byteArrToLongArr(byte[] input)

Converts an array of bytes (8-bit) to an equivalent array of longs (64-bit). Used at the beginning of keccakf() function. Parameter input is the byte array to convert to a long array. Returns a long array equivalent of input.

longArrToByteArr(long[] input)

Converts an array of longs (64-bit) to an equivalent array of bytes (8-bit). Used at ending of keccakf() function. Parameter input is the byte array to convert to a long array. Returns a long array equivalent to input.

keccakf(byte[] state)

Performs keccakf permutation, directly translated from Github user mjosaarinen's Sha-3 implementation in C at https://github.com/mjosaarinen/tiny_sha3/tree/master. Parameter state is the sponge state. Returns state after permutation.

rotateLeft(long num, int shiftAmount)

Performs rotate left bit operation. Parameter num is the number to rotate left. Parameter shiftAmount is how bits are rotated. Returns num shifted left by {shiftAmount} bits.

swapWordEndian(byte[] state)

Swaps the 64-bit word or long endianness in the byte array state. Ensure that the state array can be exactly divided into 64-bit word chunks. Parameter state is the state array to swap endianness in.

strToByteArr(String S)

Converts a string into a byte array using ISO-8859-1 charset. Parameter S is the string to convert to bytes. Returns byte array representing S in ISO-8859-1 charset.

byteArrToStr(byte[] b)

Converts a byte array into a string using ISO-8859-1 charset. Parameter b bytes to convert to a string. Returns string representing b in ISO-8859-1 charset.

concateByteArr(byte[] x, byte[] y)

Concatenates two byte arrays together. Parameter left is the left byte array. Parameter right is the right byte array. Returns the left byte array concatenated with the right array.

Known Bugs

No known bugs at the time of writing this report.

Sources

"FIPS 202." *NIST*, Aug. 2015, nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf. Accessed 05 May 2024.

Kelsey, John, et al. "NIST SP 800-185." *NIST*, Dec. 2016, nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf. Accessed 05 May 2024.

"KMAC-Samples." *NIST CSRC*, NIST, csrc.nist.gov/CSRC/media/Projects/Cryptographic-Standards-and-Guidelines/documents/examples/KMAC_samples.pdf. Accessed 05 May 2024.

Saarinen, Markku-Juhani. "Mjosaarinen/Tiny_sha3: Very Small, Readable Implementation of the SHA3 Hash Function." *Tiny_sha3*, Github, 2016, github.com/mjosaarinen/tiny_sha3. Accessed 04 May 2024.