

CS427 Final Project - Stream File Encryption & Key Management

Casey Colley

Robert Detjens

Lyell Read

CS 427, Winter 2022

Contents

Abstract	2
Stream Encryption and Decryption (enc, dec)	3
Primitives	3
Formal Scheme Definition	3
Main	3
Security Proof and Reasoning	4
Key Generation and Storage (keygen)	6
Primitives	6
Shoving this here for now sorry	6
Formal Scheme Definition	6
Security Proof and Reasoning	6
Conclusion and Discussion	7

Abstract

placeholder

Stream Encryption and Decryption (enc, dec)

placeholder

Primitives

Our design utilizes F , a Block Cipher (PRP). F will be the AES block cipher with a 256-bit key. This key will be derived using a common hashing algorithm, SHA256 based on the text password entered by the user.

- <https://www.geeksforgeeks.org/advanced-encryption-standard-aes/>
- <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>

klen = 256 TODO: types declared here	$\frac{F_{AES}(k, d):}{\text{TODO}}$
---	--------------------------------------

Formal Scheme Definition

Our symmetric encryption mode will be CTR mode.

klen = 256 TODO: types declared here	$\frac{\text{ENC}_{CTR}(k, m_1 \dots m_l):}{r \leftarrow \{0, 1\}^{klen}}$	
	$c_0 := r$	
	for $i = 1$ to l :	$\frac{\text{DEC}_{CTR}(k, c_0 \dots c_l):}{\text{TODO}}$
	$c_i := F(k, r) \oplus m_i$	
	$r := r + 1 \% 2^{klen}$	
	return $c_0 \dots c_l$	

The hashing function we will use is SHA-256.

klen = 256 TODO: types declared here	$\frac{\text{HASH}_{SHA-256}(m):}{\text{TODO}}$
---	---

Main

```
from getpass import getpass

klen, blen = 256

# Stored persistently, in file or otherwise
s = ''
K = ''
H = ''

Init():
    k = KeyGen()
    s = KeyGen()
    print("You will make a new password.")
    H = Pass2Key()
```

```

print("You will enter the password again.")
K = EncKey()
print("Vault has been initialized.")

Main():
    if:
        Init()

    k = DecKey()
    # Decrypt vault with k
    print("Vault has been decrypted.")

    #Encryption and Decryption behavior here

    # Re-encrypt vault files with k
    # k is not persistant on shutdown

```

getpass: <https://stackoverflow.com/questions/43673886/python-2-7-how-to-get-input-but-dont-show-keys-entered/43673912>

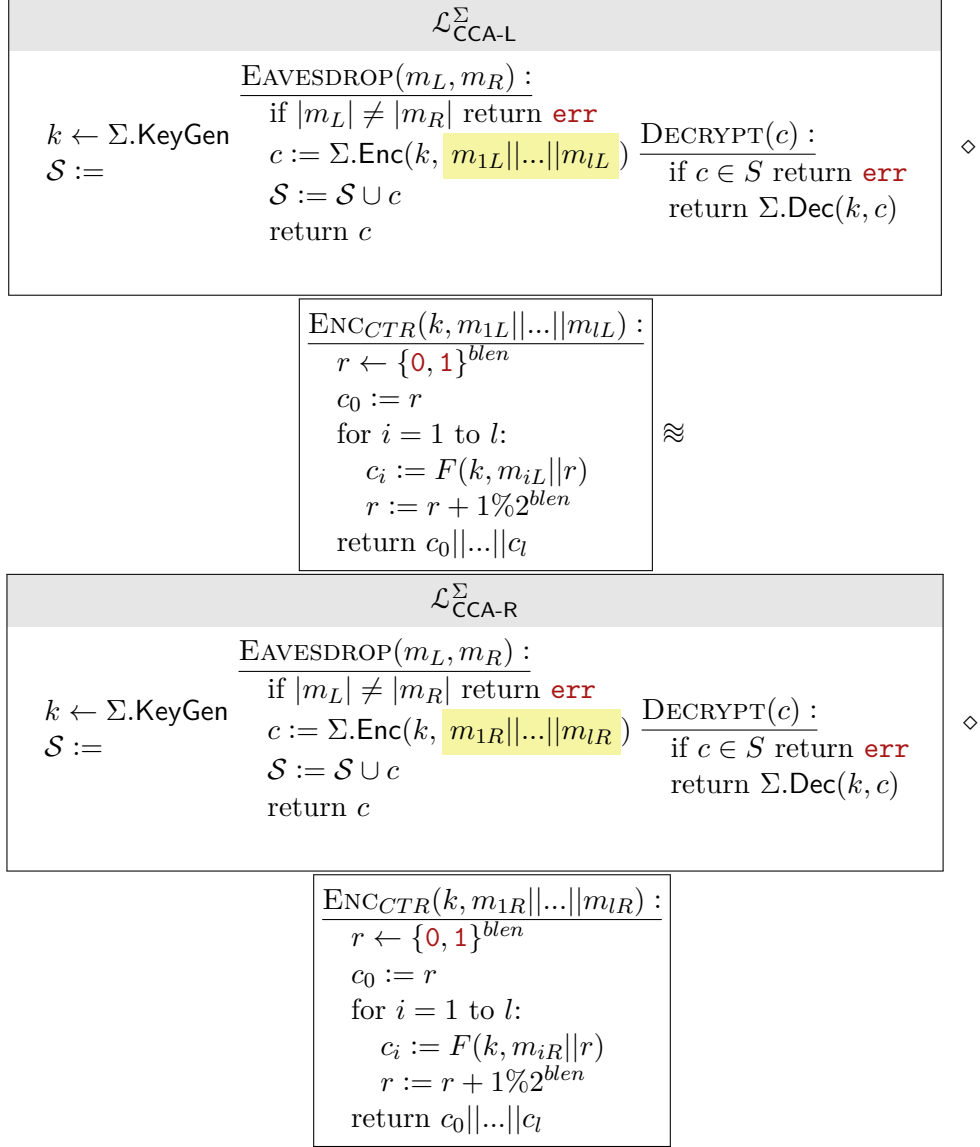
Security Proof and Reasoning

We will prove that the encryption scheme of our key manager, a modified CTR mode, has security against chosen ciphertext attacks.

To prove that a scheme has CCA security, we must prove that two random plaintexts (L & R) cannot be distinguished from each other, including any partial information, like so:

$$\begin{array}{c}
 \mathcal{L}_{\text{CCA-L}}^{\Sigma} \\
 \begin{array}{c}
 k \leftarrow \Sigma.\text{KeyGen} \\
 \mathcal{S} :=
 \end{array}
 \begin{array}{c}
 \text{EAVESDROP}(m_L, m_R) : \\
 \text{if } |m_L| \neq |m_R| \text{ return } \text{err} \\
 c := \Sigma.\text{Enc}(k, m_L) \\
 \mathcal{S} := \mathcal{S} \cup c \\
 \text{return } c
 \end{array}
 \begin{array}{c}
 \text{DECRYPT}(c) : \\
 \text{if } c \in \mathcal{S} \text{ return } \text{err} \\
 \text{return } \Sigma.\text{Dec}(k, c)
 \end{array}
 \approx
 \end{array}$$

$$\begin{array}{c}
 \mathcal{L}_{\text{CCA-R}}^{\Sigma} \\
 \begin{array}{c}
 k \leftarrow \Sigma.\text{KeyGen} \\
 \mathcal{S} :=
 \end{array}
 \begin{array}{c}
 \text{EAVESDROP}(m_L, m_R) : \\
 \text{if } |m_L| \neq |m_R| \text{ return } \text{err} \\
 c := \Sigma.\text{Enc}(k, m_R) \\
 \mathcal{S} := \mathcal{S} \cup c \\
 \text{return } c
 \end{array}
 \begin{array}{c}
 \text{DECRYPT}(c) : \\
 \text{if } c \in \mathcal{S} \text{ return } \text{err} \\
 \text{return } \Sigma.\text{Dec}(k, c)
 \end{array}
 \end{array}$$



Key Generation and Storage (keygen)

Primitives

placeholder

Shoving this here for now sorry

$m_1 \dots m_l := \text{DecStore}$ $c_0 \dots c_l := \text{EncStore}$	$\frac{\text{EncStore}(k):}{c_0 \dots c_l := \text{ENC}_{CTR}(k, m_1 \dots m_l)}$ return $c_0 \dots c_l$	$\frac{\text{DecStore}(k):}{m_1 \dots m_l := \text{DEC}_{CTR}(k, c_0 \dots c_l)}$ return $m_1 \dots m_l$
--	---	---

Formal Scheme Definition

$k := \text{DecKey}()$	$\frac{\text{KeyGen}():}{k \leftarrow \{0, 1\}^{klen}}$ return k	$\frac{\text{Pass2Key}():}{p := \text{get_passphrase}()$ $h := \text{Hash}_{SHA-256}(p s)$ return h	$\frac{\text{EncKey}(k):}{h := H}$ $K := \text{Enc}_{CTR}(h, k)$ return K
$s := \text{KeyGen}()$ $H := \text{Pass2Key}()$ $K := \text{EncKey}(h, k)$			

$\frac{\text{DecKey}(K):}{h := \text{Pass2Key}()}$ if $h \neq H$: return err $k = \text{Dec}_{CTR}(h, K)$ return k

TODO: Define types and formalize scheme in tex

Security Proof and Reasoning

Here we define a library of functions that will handle the generation and storage of the Master Key that will be used to encrypt and decrypt the stored keys in the manager. The Master Key is generated with function **KeyGen**, which samples a string of length **klen**. This sampling will come from the machine's built-in random device, such as **/dev/urandom**.

This Master Key will be stored on the machine, encrypted. The encryption and decryption of the Master Key will be done with a password and in the CTR mode, as shown in the remaining two functions, **Pass2Key()** and **EncKey()**. The correct, salted hash of the password will be stored alongside the encrypted Master Key.

EncKey() begins with **Pass2Key()**, where it will prompt the user for the password, salt it, and then return the SHA-256 hash. **EncKey** will compare this hash with the stored, correct hash. If they do not match (it is the wrong password), then an error is returned. Otherwise, **EncKey** will call the CTR mode, using the hashed password as a key/seed to the PRP F.

Conclusion and Discussion

placeholder