CS427 Final Project - Stream File Encryption & Key Management

Casey Colley Robert Detjens Lyell Read

CS 427, Winter 2022

Contents

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	$\underline{F_{AES}(k, d)}$: TODO

Formal Scheme Definition

Our symmetric encryption mode will be CTR mode.

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

The hashing function we will use is SHA-256.

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

$\mathcal{L}^{\Sigma}_{\text{CCA-L}}$

$k \leftarrow \Sigma.\mathsf{KeyGen}$
$\mathcal{S} :=$

$\underline{\textsc{Eavesdrop}(m_L, m_R):}$
if $|m_L| \neq |m_R|$ return err
$c := \Sigma.\mathsf{Enc}(k, m_L)$
$\mathcal{S} := \mathcal{S} \cup c$
return c

$\underline{\textsc{Decrypt}(c):}$
if $c \in S$ return err
return $\Sigma.\mathsf{Dec}(k, c)$

$\approx$

$\mathcal{L}^{\Sigma}_{\text{CCA-R}}$

$k \leftarrow \Sigma.\mathsf{KeyGen}$
$\mathcal{S} :=$

$\underline{\textsc{Eavesdrop}(m_L, m_R):}$
if $|m_L| \neq |m_R|$ return err
$c := \Sigma.\mathsf{Enc}(k, m_R)$
$\mathcal{S} := \mathcal{S} \cup c$
return c

$\underline{\textsc{Decrypt}(c):}$
if $c \in S$ return err
return $\Sigma.\mathsf{Dec}(k, c)$

$\mathcal{L}^{\Sigma}_{\text{CCA-L}}$

$k \leftarrow \Sigma.\mathsf{KeyGen}$
$\mathcal{S} :=$

$\underline{\textsc{Eavesdrop}(m_L, m_R):}$
if $|m_L| \neq |m_R|$ return err
$c := \Sigma.\mathsf{Enc}(k, m_{1L}||...||m_{lL})$
$\mathcal{S} := \mathcal{S} \cup c$
return c

$\underline{\textsc{Decrypt}(c):}$
if $c \in S$ return err
return $\Sigma.\mathsf{Dec}(k, c)$

$\diamond$

$\underline{\textsc{Enc}_{CTR}(k, m_{1L}||...||m_{lL}):}$
$r \leftarrow \{0,1\}^{blen}$
$c_0 := r$
for $i = 1$ to l:
$c_i := F(k, m_{iL}||r)$
$r := r + 1 \% 2^{blen}$
return $c_0||...||c_l$

$\approx$

$\mathcal{L}^{\Sigma}_{\text{CCA-R}}$

$k \leftarrow \Sigma.\mathsf{KeyGen}$
$\mathcal{S} :=$

$\underline{\textsc{Eavesdrop}(m_L, m_R):}$
if $|m_L| \neq |m_R|$ return err
$c := \Sigma.\mathsf{Enc}(k, m_{1R}||...||m_{lR})$
$\mathcal{S} := \mathcal{S} \cup c$
return c

$\underline{\textsc{Decrypt}(c):}$
if $c \in S$ return err
return $\Sigma.\mathsf{Dec}(k, c)$

$\diamond$

$\underline{\textsc{Enc}_{CTR}(k, m_{1R}||...||m_{lR}):}$
$r \leftarrow \{0,1\}^{blen}$
$c_0 := r$
for $i = 1$ to l:
$c_i := F(k, m_{iR}||r)$
$r := r + 1 \% 2^{blen}$
return $c_0||...||c_l$

Key Generation and Storage (`keygen`)

Primitives

placeholder

Shoving this here for now sorry

$m_1||...||m_l$:= DecStore
$c_0||...||c_l$:= EncStore

EncStore (k):
  $c_0||...||c_l := \text{Enc}_{CTR}(k, m_1||...||m_l)$
  return $c_0||...||c_l$

DecStore(k):
  $m_1||...||m_l := \text{Dec}_{CTR}(k, c_0||...||c_l)$
  return $m_1||...||m_l$

Formal Scheme Definition

$k := DecKey()$

$s := KeyGen()$
$H := Pass2Key()$
$K := EncKey(h, k)$

KeyGen():
  $k \leftarrow \{0,1\}^{klen}$
  return k

Pass2Key():
  $p := get\_passphrase()$
  $h := Hash_{SHA-256}(p||s)$
  return h

EncKey(k):
  $h := H$
  $K := Enc_{CTR}(h, k)$
  return K

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