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`)

These define the Encryption and Decryption algorithms used by the program both to encrypt and decrypt the Master Key, and to encrypt and decrypt messages *with* the Master Key.

Primitives

Our design utilizes a secure block cipher/PRP, F. F will be the AES block cipher with a 128-bit key. Our program utilizes a Python library for the AES block cipher implementation called PyAES. The key to the block cipher will be derived by hashing the text password entered by the user (hence, it must have 128-bit output).

$$
\boxed{\begin{array}{l}
\quad \text{klen} = 128 \\
\\
\underline{F_{AES}(k,d):} \\
\quad \text{BLACK BOX} \\
\underline{F_{AES}^{-1}(k,d):} \\
\quad \text{BLACK BOX}
\end{array}}
$$

The hash we will be using is a Davies-Meyer compression function with our AES block cipher, F. A Davies-Meyer compression function functionally turns a block cipher into a hashing function. No key is needed by the scheme; the "keys" are the blocks of the message itself. The algorithm is defined below:

$$
\boxed{\begin{array}{l}
\quad \text{blen} = 128 \\
\\
\underline{\textsc{Hash}_{D-M}(m_1||...||m_l):} \\
\quad h := \{0\}^{blen} \\
\quad \text{for } i = 1 \text{ to } l: \\
\quad\quad h := F(m_i, h) \oplus h \\
\quad \text{return } h
\end{array}}
$$

Formal Scheme Definition

Our symmetric encryption mode will be a modified CTR mode. For the Decryption algorithm, we really don't need to have r or c_0 as it's simply concatenated with m_i but we have kept it in the definition to better show how our modification resembles true CTR mode. Additionally, r can be used to verify that the decryption was successful, as the resulting block would be $m_i||r$.

blen = 128	$\underline{\text{ENC}_{CTR}(k, m_1\|\|...\|\|m_l):}$ $r \leftarrow \{0,1\}^{blen}$ $c_0 := r$ for $i = 1$ to l: $c_i := F(k, m_i\|\|r)$ $r := r + 1\%2^{blen}$ return $c_0\|\|...\|\|c_l$	$\underline{\text{DEC}_{CTR}(k, c_0\|\|...\|\|c_l):}$ $r := c_0$ for $i = 1$ to l: $m_i := F^{-1}(k, c_i)$ [blen:] $r := r + 1\%2^{blen}$ return $m_1\|\|...\|\|m_l$

Main

```
klen, blen = 128

# 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
```

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. We assume that F is a secure PRP.

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:

$$\begin{array}{|l|}\hline \mathcal{L}^{\Sigma}_{\textsf{CCA-L}} \\ \hline k \leftarrow \Sigma.\textsf{KeyGen} \\ \mathcal{S} := \emptyset \\ \\ \underline{\textsc{Eavesdrop}(m_L, m_R):} \\ \quad \text{if } |m_L| \neq |m_R|: \\ \quad\quad \text{return } \texttt{err} \\ \quad c := \Sigma.\textsf{Enc}(k, \boxed{m_L}) \\ \quad \mathcal{S} := \mathcal{S} \cup c \\ \quad \text{return } c \\ \\ \underline{\textsc{Decrypt}(c):} \\ \quad \text{if } c \in S \text{ return } \texttt{err} \\ \quad \text{return } \Sigma.\textsf{Dec}(k, c) \\ \hline \end{array} \approx \begin{array}{|l|}\hline \mathcal{L}^{\Sigma}_{\textsf{CCA-R}} \\ \hline k \leftarrow \Sigma.\textsf{KeyGen} \\ \mathcal{S} := \emptyset \\ \\ \underline{\textsc{Eavesdrop}(m_L, m_R):} \\ \quad \text{if } |m_L| \neq |m_R|: \\ \quad\quad \text{return } \texttt{err} \\ \quad c := \Sigma.\textsf{Enc}(k, \boxed{m_R}) \\ \quad \mathcal{S} := \mathcal{S} \cup c \\ \quad \text{return } c \\ \\ \underline{\textsc{Decrypt}(c):} \\ \quad \text{if } c \in S \text{ return } \texttt{err} \\ \quad \text{return } \Sigma.\textsf{Dec}(k, c) \\ \hline \end{array}$$

From here, we will walk through the proof for the left library.

$$\begin{array}{|l|}\hline \mathcal{L}^{\Sigma}_{\textsf{CCA-L}} \\ \hline k \leftarrow \Sigma.\textsf{KeyGen} \\ \mathcal{S} := \emptyset \\ \\ \underline{\textsc{Eavesdrop}(m_L, m_R):} \\ \quad \text{if } |m_L| \neq |m_R|: \\ \quad\quad \text{return } \texttt{err} \\ \quad c := \Sigma.\textsf{Enc}(k, \boxed{m_{1L}||...||m_{lL}}) \\ \quad \mathcal{S} := \mathcal{S} \cup c \\ \quad \text{return } c \\ \\ \underline{\textsc{Decrypt}(c):} \\ \quad \text{if } c \in S: \\ \quad\quad \text{return } \texttt{err} \\ \quad \text{return } \Sigma.\textsf{Dec}(k, c) \\ \hline \end{array} \diamond \begin{array}{|l|}\hline \underline{\textsc{Enc}_{CTR}(k, m_{1L}||...||m_{lL}):} \\ \quad r \leftarrow \{0,1\}^{blen} \\ \quad c_0 := r \\ \quad \text{for } i = 1 \text{ to } l: \\ \quad\quad c_i := F(k, m_{iL}||r) \\ \quad\quad r := r + 1\%2^{blen} \\ \quad \text{return } c_0||...||c_l \\ \hline \end{array} \approx \begin{array}{|c|}\hline \mathcal{L}^{\Sigma}_{\textsf{CCA-R}} \\ \hline \text{" "} \\ \hline \end{array}$$

Next, we can turn our attention to the linked encryption scheme. Here we see that for each block, we calculate $F(k, m_i||r)$ for the corresponding ciphertext block. r is sampled randomly, so the chance of collision is $\frac{1}{2^{blen}}$. However, we are doing counter mode, so r for each subsequent block in the message is deterministic, for l blocks in the message. Still, the rate of collision comes to $\frac{l}{2^{blen}}$. The l increases much slower than the 2^{blen}, which means the rate of collisions is still negligible.

Because r is sampled randomly and has a neglible rate of collisions, $m_i||r$ also has a collision rate of $\frac{l}{2^{blen}}$ even when the same m_i is inputted. It does not matter what m_i is when we concatenate it with r and put it through the PRP F. To illustrate this, we can apply the following transformation:

$$\boxed{\begin{array}{l} \mathcal{L}^{\Sigma}_{\textsf{CCA-L}} \\ \hline k \leftarrow \Sigma.\textsf{KeyGen} \\ \mathcal{S} := \emptyset \\ \\ \underline{\textsc{Eavesdrop}(m_L, m_R):} \\ \quad \text{if } |m_L| \neq |m_R|: \\ \quad\quad \text{return } \texttt{err} \\ \quad c := \Sigma.\textsf{Enc}(k, m_{1L}||...||m_{lL}) \\ \quad \mathcal{S} := \mathcal{S} \cup c \\ \quad \text{return } c \\ \\ \underline{\textsc{Decrypt}(c):} \\ \quad \text{if } c \in S: \\ \quad\quad \text{return } \texttt{err} \\ \quad \text{return } \Sigma.\textsf{Dec}(k, c) \end{array}} \diamond \boxed{\begin{array}{l} \underline{\textsc{Enc}_{CTR}(k, m_{1L}||...||m_{lL}):} \\ \quad x \leftarrow \{0, 1\}^{blen} \\ \quad c_0 := r \\ \quad \text{for } i = 1 \text{ to } l: \\ \quad\quad c_i := F(k, x) \\ \quad\quad r := r + 1\%2^{blen} \\ \quad \text{return } c_0||...||c_l \end{array}} \approx \boxed{\begin{array}{c} \mathcal{L}^{\Sigma}_{\textsf{CCA-R}} \\ \hline \text{" "} \end{array}}$$

Now, $m_{1L}||...||m_{lL}$ is not being used by the Enc_{CTR} function; we can change it to some other name without disrupting the function of the encryption scheme. We can rename this to $m_{1R}||...||m_{lR}$ and inline it into the library.

$$\boxed{\begin{array}{l} \mathcal{L}^{\Sigma}_{\textsf{CCA-L}} \\ \hline k \leftarrow \Sigma.\textsf{KeyGen} \\ \mathcal{S} := \emptyset \\ \\ \underline{\textsc{Eavesdrop}(m_L, m_R):} \\ \quad \text{if } |m_L| \neq |m_R|: \\ \quad\quad \text{return } \texttt{err} \\ \quad c := \Sigma.\textsf{Enc}(k, m_{1R}||...||m_{lR}) \\ \quad \mathcal{S} := \mathcal{S} \cup c \\ \quad \text{return } c \\ \\ \underline{\textsc{Decrypt}(c):} \\ \quad \text{if } c \in S: \\ \quad\quad \text{return } \texttt{err} \\ \quad \text{return } \Sigma.\textsf{Dec}(k, c) \end{array}} \diamond \boxed{\begin{array}{l} \underline{\textsc{Enc}_{CTR}(k, m_{1R}||...||m_{lR}):} \\ \quad x \leftarrow \{0, 1\}^{blen} \\ \quad c_0 := r \\ \quad \text{for } i = 1 \text{ to } l: \\ \quad\quad c_i := F(k, x) \\ \quad\quad r := r + 1\%2^{blen} \\ \quad \text{return } c_0||...||c_l \end{array}} \approx \boxed{\begin{array}{c} \mathcal{L}^{\Sigma}_{\textsf{CCA-R}} \\ \hline \text{" "} \end{array}}$$

Let's inline the whole linked function, and re-consider the right library.

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

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

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

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

$\approx$

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

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

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

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

Here we can see in this function, the left and right libraries are indistinguishable. For any calling program A, it will not be able to distinguish between the two libraries - aka, it will not be able to obtain any partial information from the scheme.

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