Security Essentials for Systems Programmers

CS 3214

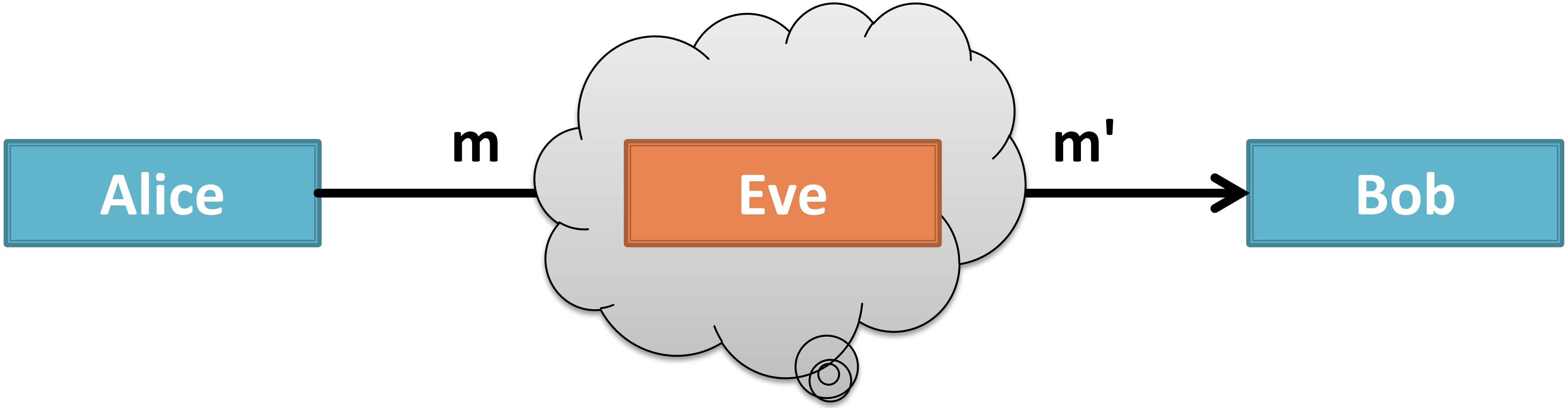
Security properties you should learn

- Confidentiality
- Integrity
- Availability
- Authenticity

Known as **CIA**

Confidentiality is preventing others from reading your data

- **Goal:** Communicate across an untrusted medium, but only Alice and Bob know what is being communicated



- **Adversary:** Eve is an eavesdropper who can read every message sent between Alice and Bob

Confidentiality is preventing others from reading your data

- **Goal:** Communicate across an untrusted medium, but only Alice and Bob know what is being communicated

Alice

Encryption provides confidentiality

Bob

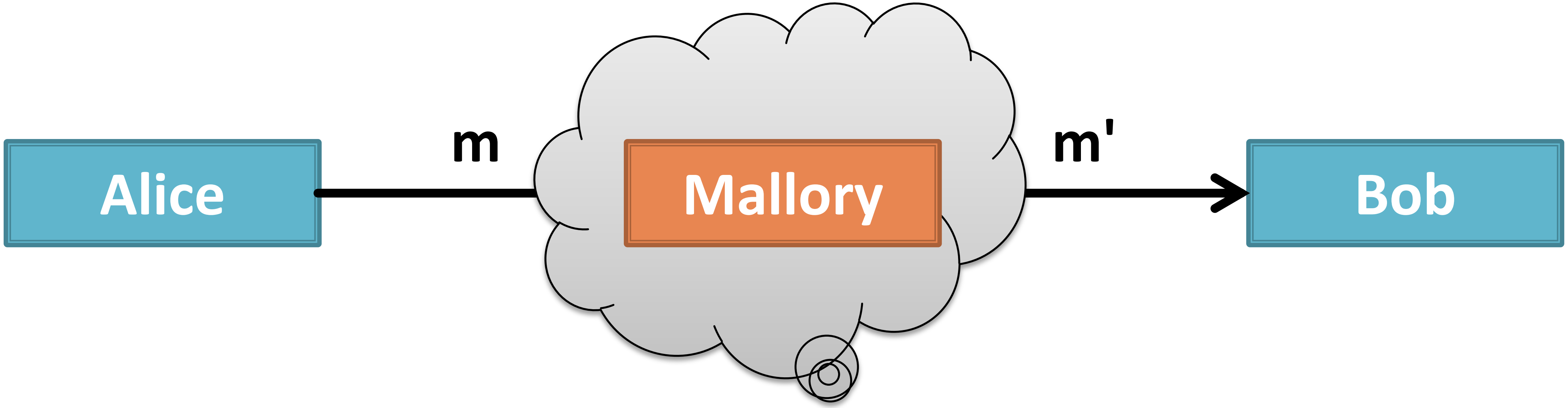
- **Adversary:** Eve is an eavesdropper who can read every message sent between Alice and Bob

Confidentiality Use Cases

- Internet communication
  - Web pages
  - Email
  - Audio and video
  - Conference calls
- File storage
  - On a managed system
  - On a system you may lose control of
    - Theft and hacking
- General communication
  - Phone
  - WiFi
  - Bluetooth
- Extortion (see ransomware)

Integrity is preventing others from modifying your data (with you being aware of it)

- **Goal:** Communicate across an untrusted medium, but every message received must be exactly what was sent



- **Adversary:** Mallory is an malicious entity who can modify, reorder, and replay every message sent between Alice and Bob

Integrity is preventing others from modifying your data (with you being aware of it)

- **Goal:** Communicate across an untrusted medium, but every message received must be exactly what was sent

Alice

Hashing provides integrity

Bob

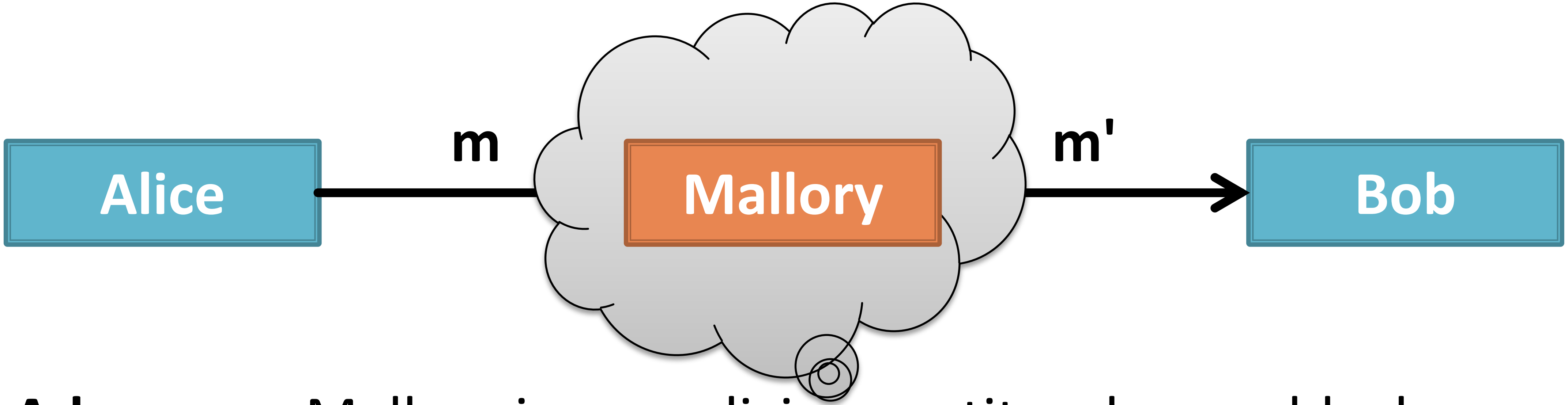
- **Adversary:** Mallory is an malicious entity who can modify, reorder, and replay every message sent between Alice and Bob

(cryptographic) Integrity use cases

- Web pages
- Email
- Password storage and comparison
- Secure system memory
- Software downloads
- Bitcoin et al.
  - Block chain
- SSH
- Malware checking

Availability is making sure your data/service is accessible

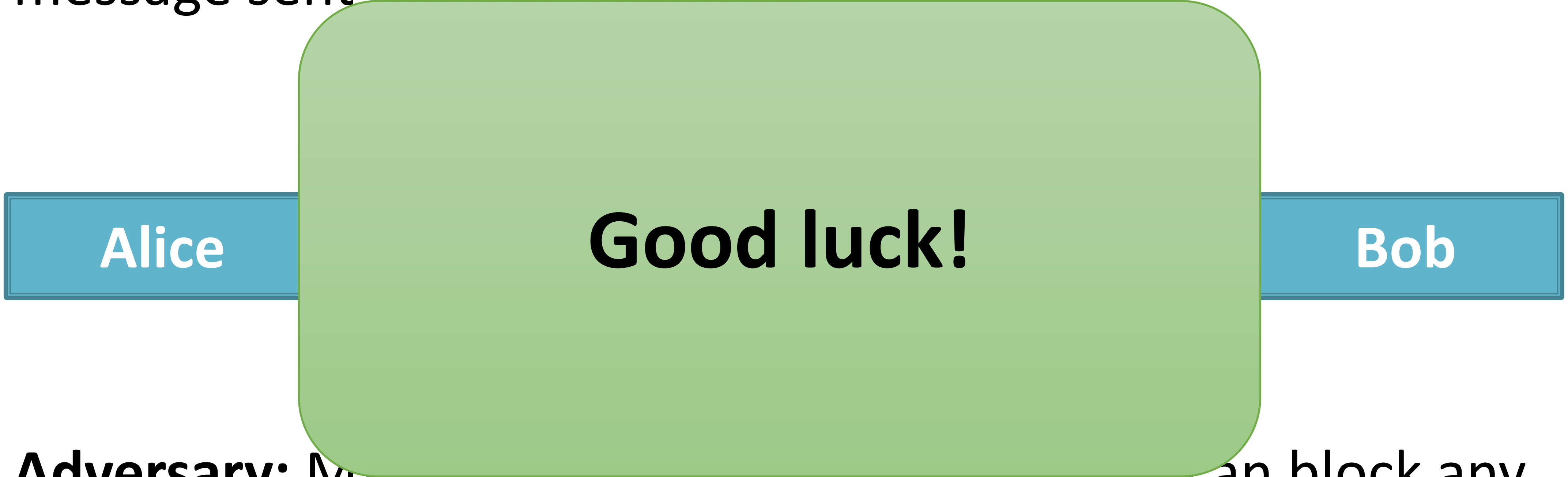
- **Goal:** Communicate across an untrusted medium, but every message sent must be received



- **Adversary:** Mallory is an malicious entity who can block any message sent between Alice and Bob or flood them with messages

Availability is making sure your data/service is accessible

- **Goal:** Communicate across an <u>untrusted</u> medium, but every message sent must be received



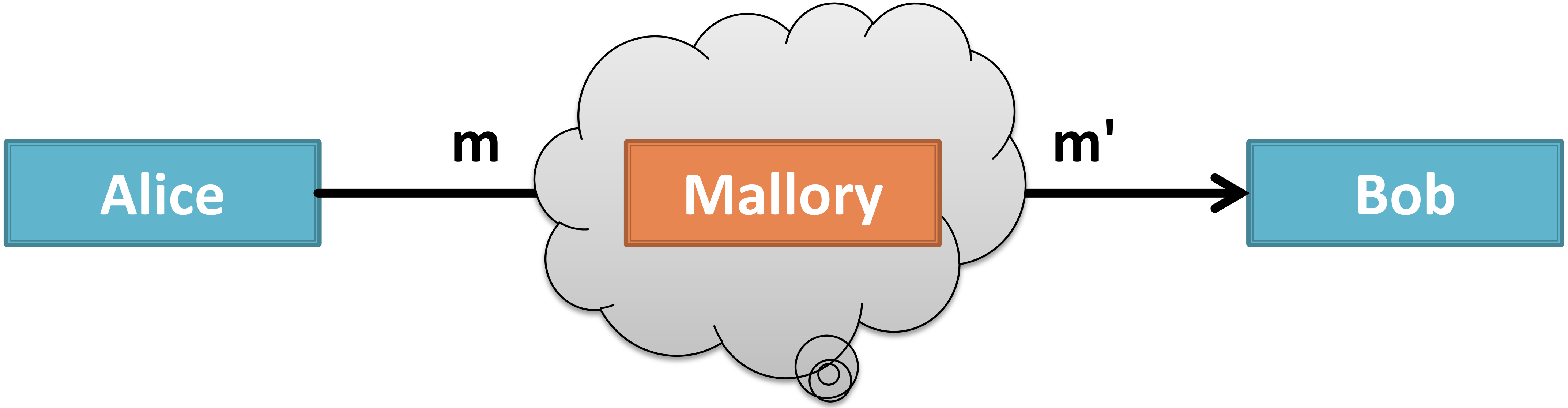
- **Adversary:** Mallory is an malicious entity who can block any message sent between Alice and Bob or flood them with messages

Availability use case

- If you cannot access it, it doesn’t exist

Authenticity is ensuring the other party is who you think they are

- **Goal:** Communicate across an <u>untrusted</u> medium, but Bob must be able to prove the message came from Alice



- **Adversary:** Mallory is an malicious entity who pretends to be Alice

Authenticity is ensuring the other party is who you think they are

- **Goal:** Communicate across an <u>untrusted</u> medium, but Bob must be able to prove the message came from Alice

Shared keys, public keys, passwords/tokens/fingerprints provide authenticity

- **Adversary:** Mallory is an malicious entity who pretends to be Alice

Authentication use case

- Access control
  - Physical access
    - Buildings, cars, parking, airplanes…
  - System access
    - Your computer, rlogin…
  - Service access
    - Hokiespa, library, Zoom…
  - Data access
- Bitcoin
  - Money
- Communication channel access
  - Public Key Certificates

Security Easy Buttons (card subject to change)

- Confidentiality – AES (gov. uses 192-bit+)
- Integrity – SHA256
  - Never md5 or SHA1
- Availability – duplication and distribution, client filtering, push work/state on to client
- Authentication
  - Host
    - Salted hash (SHA256) password comparison
  - Network
    - Shared keys + SHA256
    - Public keys: ECDSA (P-256) + SHA256

How do you implement these in your own programs?

Don’t roll your own!

Choose a **trusted+open source** crypto library

- What most people use: Openssl
  - C/C++ library
  - Command line tool
  - Python: pyopenssl
  - Lack of diversity in crypto. Implementations is problematic: see HeartBleed
- Another option: Crypto++
  - C/C++ library
  - Python: pycryptopp wrapper
- Note: SSL is the predecessor to the modern day TLS

Prerequisites

- What I tested this on
  - Ubuntu 20.04
  - Openssl 1.1.1f
  - Intel x86_64
- Required packages
  - sudo apt install clang libssl-dev
- How I compile
  - clang -Wall -O3 -o example.bin example.c -lssl –lcrypto
- Assumes plaintext.txt in the same directory

Using Openssl for AES Encryption

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <openssl/evp.h>
#include <openssl/aes.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

static const unsigned char key[] =  "thiskeyisverybad12345678"; // 192 bits
static const unsigned char iv[] =   "publicButUnique";          // 128 bits (aka block size)
```

```c
long int getFileSize(FILE *fp) {
    struct stat fInfo;
    fstat(fileno(fp), &fInfo);
    off_t size = fInfo.st_size;
    return (long int)size;
}

int main(int argc, char *argv[])
{
    FILE *fIN, *fOUT;


    // Encrypt
    fIN = fopen("plaintext.txt", "rb");
    fOUT = fopen("ciphertext.txt", "wb");
    encrypt(fIN, fOUT);
    fclose(fIN);
    fclose(fOUT);


    // Decrypt
    fIN = fopen("ciphertext.txt", "rb");
    fOUT = fopen("plaintext2.txt", "wb");
    decrypt(fIN, fOUT);
    fclose(fIN);
    fclose(fOUT);
}
```

```c
void encrypt(FILE *ifp, FILE *ofp)
{
    long int fsize = getFileSize(ifp);

    int outLen1 = 0; int outLen2 = 0;
    unsigned char *indata = malloc(fsize);
    unsigned char *outdata = malloc(fsize + AES_BLOCK_SIZE); // May need an extra block's worth of bytes

    // Read entire file as bytes
    fread(indata, sizeof(char), fsize, ifp);

    // Setup
    EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
    EVP_CIPHER_CTX_init(ctx);

    // Encrypt plaintext
    EVP_EncryptInit(ctx, EVP_aes_192_cbc(), key, iv);
    EVP_EncryptUpdate(ctx, outdata, &outLen1, indata, fsize);
    EVP_EncryptFinal(ctx, outdata + outLen1, &outLen2); // Handle padding

    // Write ciphertext to file as bytes
    fwrite(outdata, sizeof(char), outLen1 + outLen2, ofp);

    // Cleanup
    EVP_CIPHER_CTX_free(ctx);
}
```

```c
void decrypt(FILE *ifp, FILE *ofp)
{
    int fsize = getFileSize(ifp);

    int outLen1 = 0; int outLen2 = 0;
    unsigned char *indata = malloc(fsize);
    unsigned char *outdata = malloc(fsize); // Ciphertext always larger than plaintext due to padding

    //Read entire ciphertext as bytes
    fread(indata, sizeof(char), fsize, ifp);

    // Setup
    EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
    EVP_CIPHER_CTX_init(ctx);

    // Decrypt ciphertext
    EVP_DecryptInit(ctx, EVP_aes_192_cbc(), key, iv);
    EVP_DecryptUpdate(ctx, outdata, &outLen1, indata, fsize);
    EVP_DecryptFinal(ctx, outdata + outLen1, &outLen2); // Last block will always have padding

    // Write plaintext to file as bytes
    fwrite(outdata, sizeof(char), outLen1 + outLen2, ofp);

    // Cleanup
    EVP_CIPHER_CTX_free(ctx);
}
```

Using Openssl for SHA Hashing

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <openssl/evp.h>
#include <openssl/sha.h>
#include <openssl/hmac.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>


static const unsigned char key[] =  "thiskeyisverybad12345678"; // 192 bits


long int getFileSize(FILE *fp) {
    struct stat fInfo;
    fstat(fileno(fp), &fInfo);
    off_t size = fInfo.st_size;
    return (long int)size;
}
```

```c
int main(int argc, char *argv[])
{
    FILE *fIN;

    // Get a hash digest of the file
    fIN = fopen("plaintext.txt", "rb");
    unsigned char digest[SHA256_DIGEST_LENGTH + 1];
    digest[32] = '\0';

    // Hash
    hash(fIN, digest);
    printf("%s\n", digest);

    fseek(fIN, 0L, SEEK_SET);

    // HMAC ~ keyed hash
    hmac(fIN, digest);
    printf("%s\n", digest);

    fclose(fIN);
}
```

```
void hash(FILE *fp, unsigned char *digest) {
    long int fsize = getFileSize(fp);

    unsigned char *indata = malloc(fsize);
    fread(indata, sizeof(char), fsize, fp);

    unsigned int outlen = 0;
    // CTK creation, Init, Update, and Final all in one
    EVP_Digest(indata, fsize, digest, &outlen, EVP_sha256(), NULL);
}
```

```c
void hmac(FILE *fp, unsigned char *digest) {
    long int fsize = getFileSize(fp);

    unsigned char *indata = malloc(fsize);
    fread(indata, sizeof(char), fsize, fp);

    unsigned int outlen = 0;
    // CTK creation, Init, Update, and Final all in one
    HMAC(EVP_sha256(), key, sizeof(key) - 1, indata, fsize, digest, &outlen);
}
```

Using Openssl for Symmetric Key Generation

```
…
#include <openssl/rand.h>
…

#define KEY_BITS 192
static unsigned char key[KEY_BITS/8];

int main(int argc, char *argv[])
{
    …
    // Get random key
    RAND_bytes(key, sizeof(key));
    …
}
```

CS 4264

Principles of Computer Security

Lots more details

Lots more topics

…and public key fun