CS631 - Advanced Programming in the UNIX Environment
–
Advanced I/O / Encryption in a Nutshell

Department of Computer Science
Stevens Institute of Technology
Jan Schaumann
jschauma@stevens.edu
http://www.cs.stevens.edu/~jschauma/631/

Nonblocking I/O

Recall from our lecture on signals that certain system calls can block forever:

- `read(2)` from a particular file, if data isn't present (pipes, terminals, network devices)
- `write(2)` to the same kind of file
- `open(2)` of a particular file until a specific condition occurs
- `read(2)` and `write(2)` of files that have mandatory locking enabled
- certain `ioctls(2)`
- some IPC functions (such as `sendto(2)` or `recv(2)`)

See `eintr.c` from that lecture.

Nonblocking I/O

Recall from our lecture on signals that certain system calls can block forever:

- `read(2)` from a particular file, if data isn't present (pipes, terminals, network devices)
- `write(2)` to the same kind of file
- `open(2)` of a particular file until a specific condition occurs
- `read(2)` and `write(2)` of files that have mandatory locking enabled
- certain `ioctls(2)`
- some IPC functions (such as `sendto(2)` or `recv(2)`)

Nonblocking I/O lets us issue an I/O operation and not have it block forever. If the operation cannot be completed, return is made immediately with an error noting that the operating would have blocked (`EWOULDBLOCK` or `EAGAIN`).

Nonblocking I/O

Ways to specify nonblocking mode:

- pass `O_NONBLOCK` to `open(2)`:

```
open(path, O_RDRW|O_NONBLOCK);
```

- set `O_NONBLOCK` via `fcntl(2)`:

```
flags = fcntl(fd, F_GETFL, 0);
fcntl(fd, F_SETFL, flags|O_NONBLOCK);
```

Nonblocking I/O

```
$ cc -Wall nonblock.c -o block
$ cc -DNONBLOCK -Wall nonblock.c -o nonblock
$ ./nonblock >/dev/null
wrote 100000 bytes
[...]
$ ./block | ( sleep 3; cat >/dev/null )
[...]
$ ./nonblock | ( sleep 3; cat >/dev/null )
[...]
$ ( ./nonblock | cat >/dev/null ) 2>&1 | more
[...]
$ nc -l 8080 >/dev/null &

$ ./nonblock | nc hostname 8080
[...]
```

Resource Locking

Ways we have learned so far to ensure only one process has exclusive access to a resource:

- open file using `O_CREAT|O_EXCL`, then immediately `unlink(2)` it
- create a “lockfile” – if file exists, somebody else is using the resource
- use of a semaphore

What are some problems with each of these?

Advisory Locking

```
#include <fcntl.h>

int flock(int fd,int operation);
```

Returns: 0 if OK, -1 otherwise

- applies or removes an advisory lock on the file associated with the file descriptor fd
- *operation* can be `LOCK_NB` and any one of:
  - `LOCK_SH`
  - `LOCK_EX`
  - `LOCK_UN`
- locks entire file

Advisory Locking

```
$ cc -Wall flock.c
1$ ./a.out
Shared lock established - sleeping for 10 seconds.
[...]
Giving up all locks.

2$ ./a.out
Shared lock established - sleeping for 10 seconds.
Now trying to get an exclusive lock.
Unable to get an exclusive lock.
[...]
Exclusive lock established.

1$ ./a.out
[blocks until the other process terminates]
```

Advisory "Record" Locking

Record locking is done using `fcntl(2)`, using one of `F_GETLK`, `F_SETLK` or `F_SETLKW` and passing a

```
struct flock {
    short l_type;   /* F_RDLCK, F_WRLCK, or F_UNLCK */
    off_t l_start;  /* offset in bytes from l_whence */
    short l_whence; /* SEEK_SET, SEEK_CUR, or SEEK_END */
    off_t l_len;    /* length, in bytes; 0 means "lock to EOF" */
    pid_t l_pid;    /* returned by F_GETLK */
}
```

Lock types are:

- `F_RDLCK` – Non-exclusive (read) lock; fails if write lock exists.
- `F_WRLCK` – Exclusive (write) lock; fails if any lock exists.
- `F_UNLCK` – Releases our lock on specified range.

Advisory "Record" locking

```
#include <unistd.h>

int lockf(int fd, int value, off_t size);
```

Returns: 0 on success, -1 on error

value can be:

- `F_ULOCK` – unlock locked sections
- `F_LOCK` – lock a section for exclusive use
- `F_TLOCK` – test and lock a section for exclusive use
- `F_TEST` – test a section for locks by other processes

		Request for	
		read lock	write lock
Region currently has	no locks	OK	OK
	one or more read locks	OK	denied
	one write lock	denied	denied

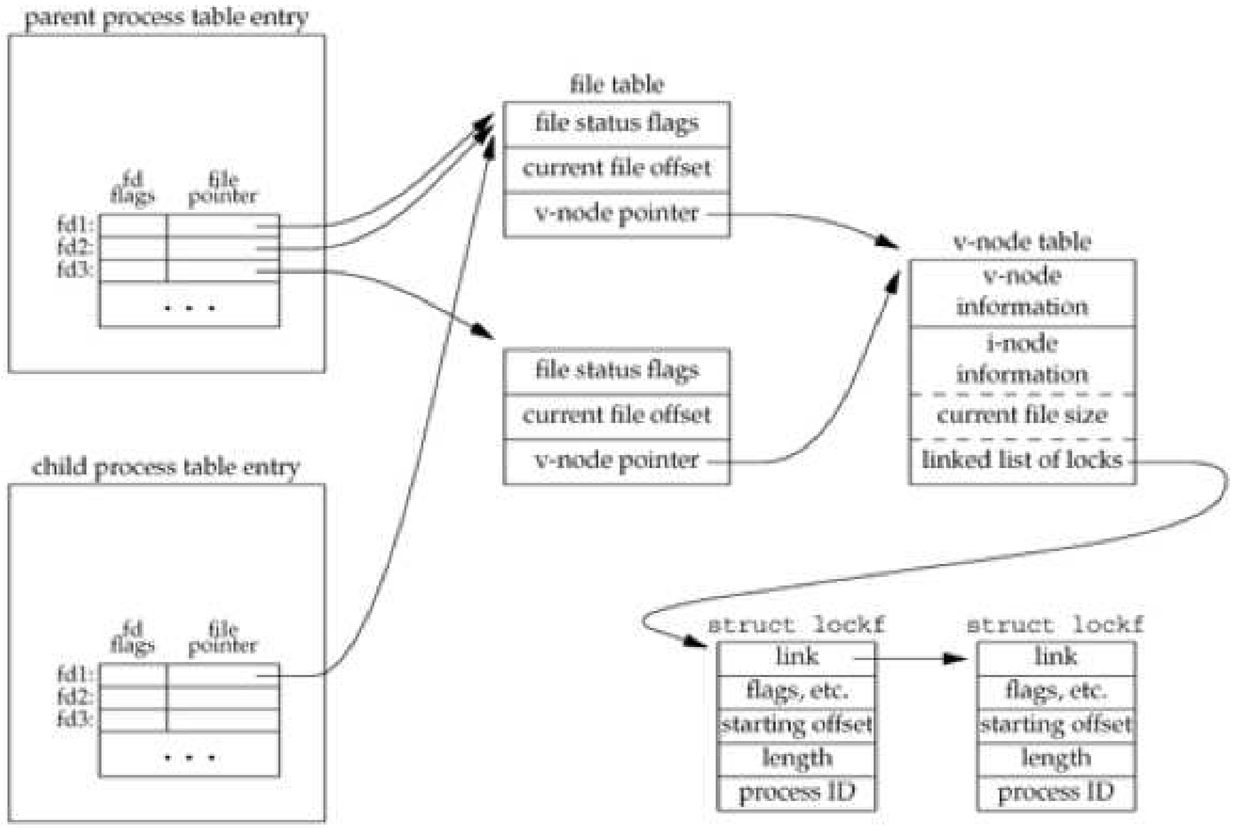
Advisory "Record" locking

Locks are:

- released if a process terminates
- released if a filedescriptor is closed (!)
- not inherited across `fork(2)`
- inherited across `exec(2)`
- released upon `exec(2)` if close-on-exec is set

Advisory “Record” locking

Locks are associated with a *file and process pair*, not with a *filedescriptor*!



Mandatory locking

- not implemented on all UNIX flavors
  - `chmod g+s,g-x file`
- possible to be circumvented:

```
$ mandatory-lock /tmp/file &
$ echo foo > /tmp/file2
$ rm /tmp/file
$ mv /tmp/file2 /tmp/file
```

Asynchronous I/O

- System V derived async I/O
  - limited to STREAMS
  - enabled via `ioctl(2)`
  - uses `SIGPOLL`
- BSD derived async I/O
  - limited to terminals and networks
  - enabled via `fcntl(2)` (`O_ASYNC`, `F_SETOWN`)
  - uses `SIGIO` and `SIGURG`

Mentioned here for completeness's sake only.
See `aio(7)` for an example of POSIX AIO.

Memory Mapped I/O

```
#include <sys/types.h>
#include <sys/mman.h>

void *mmap(void *addr, size_t len, int prot, int flags, int fd, off_t offset);
```

Returns: pointer to mapped region if OK

Protection specified for a region:

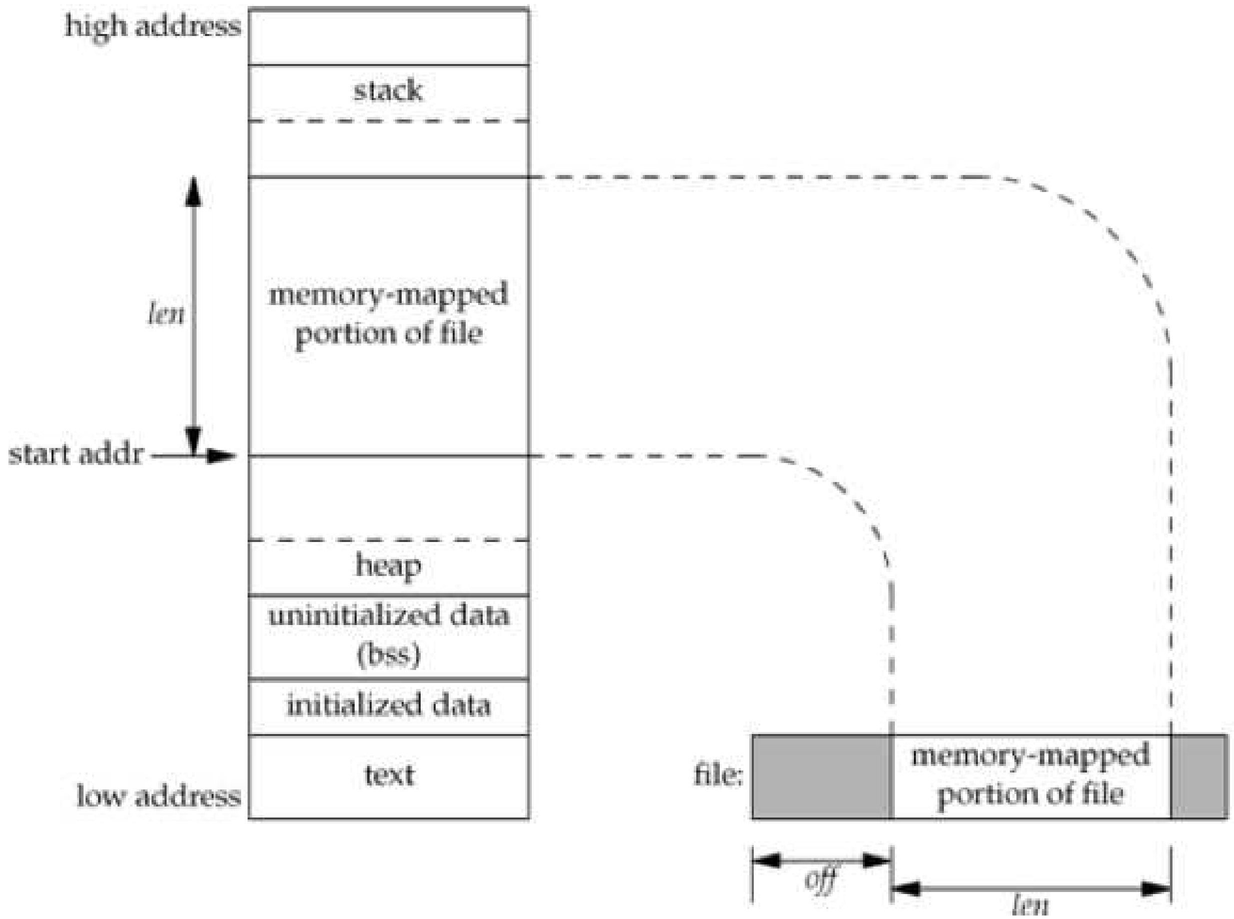
- `PROT_READ` – region can be read
- `PROT_WRITE` – region can be written
- `PROT_EXEC` – region can be executed
- `PROT_NONE` – region can not be accessed

flag needs to be one of

- `MAP_SHARED`
- `MAP_PRIVATE`
- `MAP_COPY`

which may be OR'd with other flags (see `mmap(2)` for details).

Memory Mapped I/O



Memory Mapped I/O

Operation	Linux 2.4.22 (Intel x86)			Solaris 9 (SPARC)		
	User	System	Clock	User	System	Clock
`read/write`	0.04	1.02	39.76	0.18	9.70	41.66
`mmap/memcpy`	0.64	1.31	24.26	1.68	7.94	28.53

Exercise: write a program that benchmarks this performance and run it on the systems you have access to.

Memory Mapped I/O

`http://cvsweb.netbsd.org/bsdweb.cgi/src/bin/cp/utils.c?rev=HEAD`

Hooray!

5 Minute Break

Cryptography

Cryptography can provide “security” in the areas of:

- Authenticity
  - *Is the party I’m talking to actually who I* think *it is?*
- Accuracy or Integrity
  - *Is the message I received in fact what was sent?*
- Secrecy or Confidentiality
  - *Did/could anybody else see (parts of) the message?*

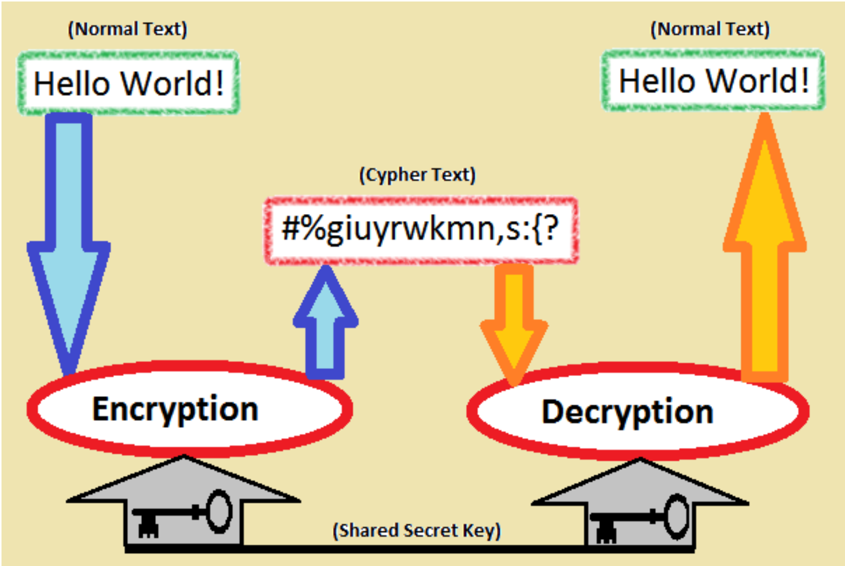
How does encryption work?

Secrecy: Make sure that the data can only be read by those intended.

How does encryption work?

Secrecy: Make sure that the data can only be read by those intended.

- Alice and Bob agree on a way to transform data
- transformed data is sent over insecure channel
- Alice and Bob are able to get data out of the transformation



How does encryption work?

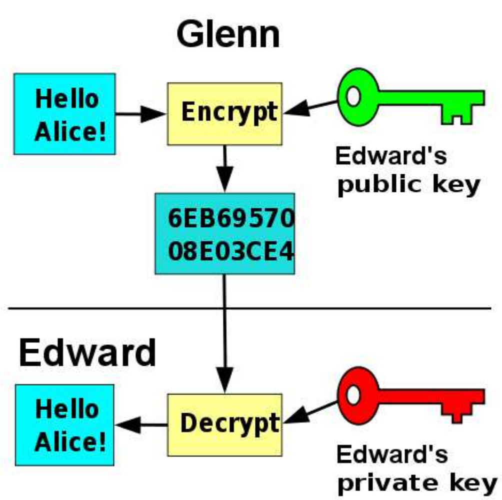
Different approaches:

- public key cryptography
- secret key cryptography

How does encryption work?

Different approaches:

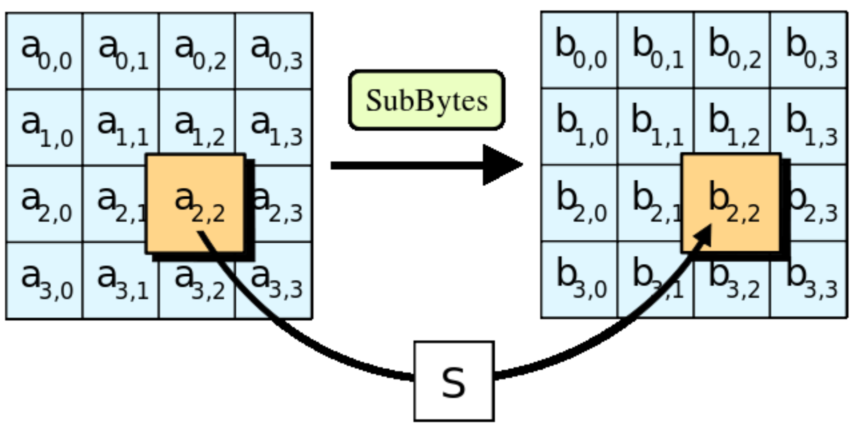
- public key cryptography (example: *RSA*, your ssh keys)
  - Alice has a private and a public key
  - data encrypted with her private key can only be decrypted by her public key and vice versa
  - public key can be shared with Bob



How does encryption work?

Different approaches:

- secret key cryptography (example: *AES*)
  - Alice and Bob share a secret key
  - for authentication purposes, Alice may prove to Bob that he knows the secret key
  - any data encrypted with this key can also be decrypted using the same key



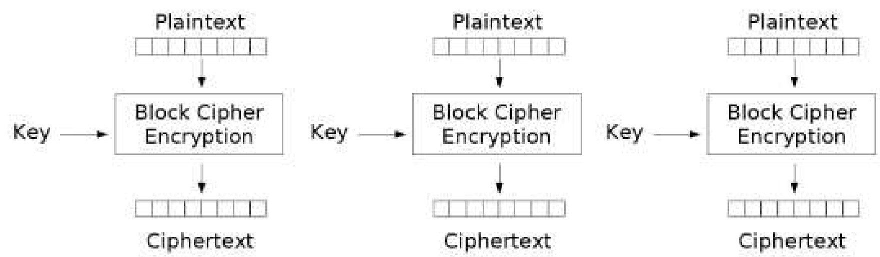
Cipher Modes

Encryption entails transformation of input data (“plain” or “clear” text) into encrypted output data (“ciphertext”). Input data is generally transformed in one of two ways:

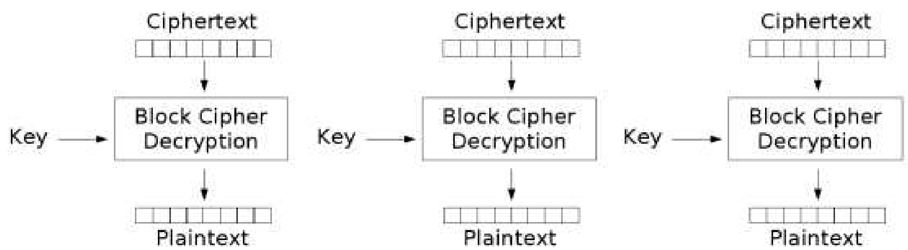
Stream Cipher: each bit on plaintext is combined with a pseudo-random cipher digit stream (or *keystream*)

Block Cipher: fixed-length blocks of plaintext are transformed into same-sized blocks of ciphertext; may require padding

Electronic Codebook Mode

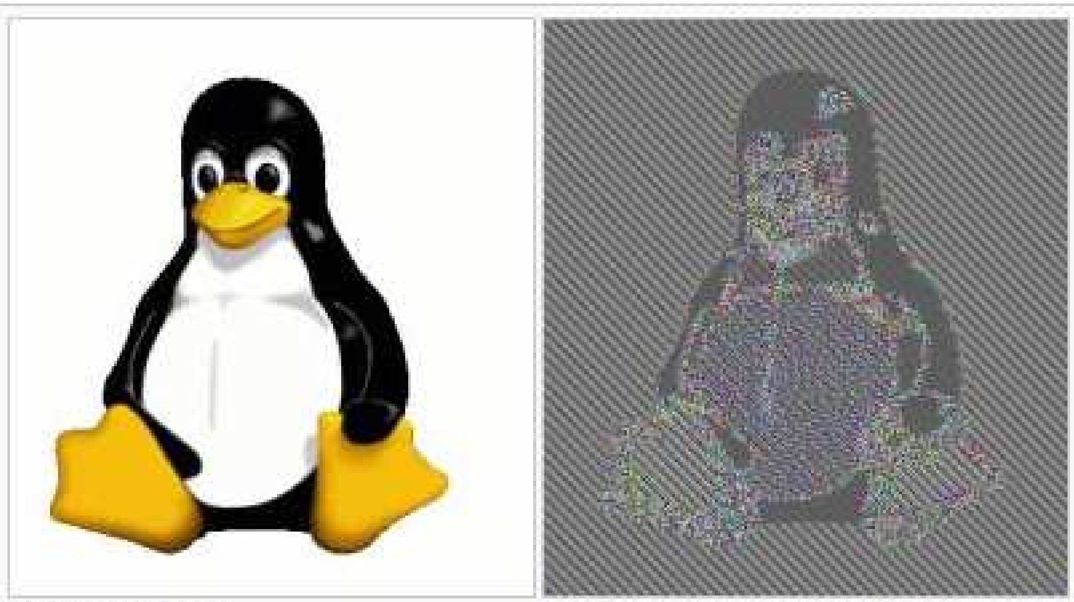


Electronic Codebook (ECB) mode encryption



Electronic Codebook (ECB) mode decryption

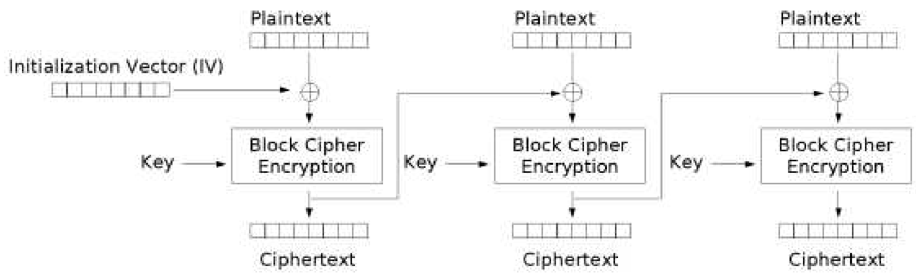
Electronic Codebook Mode



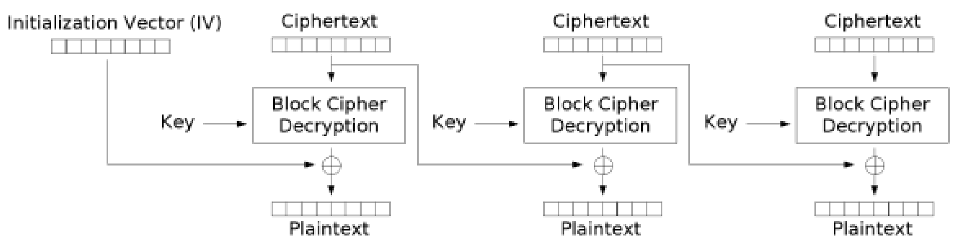
Original image

Encrypted using ECB mode

Cipher Block Chaining



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

Random String generation

Random numbers can be generated using `/dev/random`, `/dev/urandom`, `rand(3)`, `random(3)`, `BN_rand(3)` etc.

Map numbers to printable characters (for use as a salt, for example):

```
static const unsigned char itoa64[] =
        "./0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";

char salt[16];
for (i=0; i<16; i++)
        salt[i] = itoa64[(int)random()%64];
```

Practical AES

- a symmetric block cipher
- variable key length
- consists of a key setup phase and the actual encryption or decryption
- keying material use of `ivec`, which needs to be shared
- useful code examples in `EVP_EncryptInit(3)`

HW#4

```
https://www.cs.stevens.edu/~jschauma/631/f16-hw4.html

aed - perform aes256cbc encryption/decryption

aed reads data from stdin and either encrypts or decrypts it using
AES 256bit CBC mode with a SHA1 digest with keying material derived
from the passphrase using the EVP_BytesToKey(3) function, generating
a suitable salt via RAND_bytes(3).

aed reads the password from which to derive the key material from the
AED_PASS environment variable.

Output is written to stdout.

When encrypting, the output is prefixed by the string "Salted__",
followed by the 8 byte salt.
```

HW#4

https://www.cs.stevens.edu/~jschauma/631/f16-hw4.html

To encrypt the contents of the file file and storing the encrypted output in file.enc:

```
aed -e <file >file.enc
```

To decrypt the contents of that file again:

```
aed -d <file.enc
```

Since aed operates on stdin and stdout, the above two commands could also be chained:

```
cat file | aed -e | aed -d
```

References

- crypto(3)
- EVP_EncryptInit(3)
- EVP_BytesToKey(3)
- http://tldp.org/LDP/LG/issue87/vinayak.html
- http://en.wikipedia.org/wiki/Cipher_Block_Chaining
- http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html