## Linux下的Ecryptfs-util编译：

依赖模块：keyutils 、 Linux-PAM 、flex、nss

**arm-linux-gcc 环境配置：**

下载arm-linux-gcc 4.3.2 解压后，将：4.3.2文件夹拷贝到 /home/当前用户/

运行命令行程序：

添加环境变量：export PATH=$PATH:/home/当前用户/4.3.2/bin

检查环境变量是否正确：$PATH

如果回显的字符串中有刚刚配置的路径：/home/当前用户/4.3.2/bin

则输入：arm-linux-gcc –v

如果命令行回显最后一行有：arm-linux-gcc v4.3.2

则表示环境变量以及arm-linux-gcc交叉编译环境配置正确。

**Keyutils模块编译：**

由于源代码中没有提供configure文件，而且Makefile文件已经生成好，

所以直接 make CC=arm-linux-gcc （可以直接编译通过，如果编译不通过，则检查错误。）

将编译完成后的.a 文件拷贝到：/home/当前用户/4.3.2/arm-none-linux-gnueabi/lib

到此，Keyutils模块所需要做的工作完成。

注意：在执行make命令时，一定要指定CC=arm-linux-gcc 否则编译的时候会使用默认的linux系统的gcc。

**Linux-PAM模块编译：**

编译ecryptfs-utils，执行./configure 命令时，无法找到pam模块，开始的时候为了简单，直接将模块禁用了，用 ./configure --host=arm-linux CC=arm-linux-gcc --disable-pam

来将pam模块从ecryptfs-util中禁用，这个会造成很多功能不能使用，但是起初没有更好的解决方法。

后来从网上下载了新的Linux-PAM模块代码，并编译成功：

模块给提供了configure，所以最好重新生成一下Makefile（原有Makefile是linux下的交叉编译不能使用）

命令行中：

./configure –host=arm -linux CC=arm-linux-gcc

注： --host=arm-none-linux 所指定的是，目标环境为arm环境。CC=arm-linux-gcc说明需要用arm-linux-gcc进行目标文件的编译。

等待命令行执行完成以后，就可以生成新的Makefile文件。

直接运行：make

就可以对模块进行编译。在命令执行过程中要注意，如果命令行里面回显的内容里面有gcc 而不是arm-linux-gcc 则需要用：

Make CC=arm-linux-gcc 进行编译。

编译成功以后，所生成的.so文件在隐藏文件夹中，（CTRL+H用来显示隐藏文件夹）将libpam、libpamc、libpammisc中.libs文件夹中的两个.so的快捷方式，一个.so（如libpammisc.so.0.82.0）文件，拷贝到系统搜索路径中：`/home/当前用户/4.3.2/arm-none-linux-gnueabi/lib

**Flex模块：（需要特定的代码，代码版本2.5.4里面只包含代码文件与一个MISC文件夹）**

编译ecryptfs-util时，提示：yylex未定义，所以从网上下载了一个arm下编译的flex.deb文件

用ar -x <package>.deb解压后，将文件夹里面的data与control解压，找到里面的.a文件与.h文件，拷贝到/home/当前用户/4.3.2/arm-none-linux-gnueabi/lib 与 /home/当前用户/4.3.2/arm-none-linux-gnueabi/include 下，无明显效果。

下载2.5.4版的源代码后，

./configure –host=arm-none-linux

执行后会生成新的makefile文件，用make CC=arm-linux-gcc RANLIB=arm-linux-ranlib AR=arm-linux-ar

即可完成对flex源文件的编译，注意后面的两个：RANLIB与AR

AR的作用是将编译生成的动态库文件打包成静态的.a文件。

RANLIB的作用是将.a中的所有库文件进行索引，用来调用里面的函数。（如果目标系统为Android，1 必须用静态库，静态编译 ；2 必须对静态库.a文件进行RANLIB处理，否则编译的时候会提示：无法找到方法）。

将生成的.a文件放到搜索目录，再次编译ecryptfs-utils时，可以搜索到模块相关的信息。

**NSS模块**

完成flex模块的编译以后，ecryptfs-utils的编译继续进行，提示：PK11\_ImportSymKey未定义。这个定义是属于NSS模块中的。

Linux里面有libnss-dev、libnss-tools 但是无法在arm平台下使用。

网上所能找到的NSS模块源代码全部是由Mozillia提供的，它依赖的库文件很多，目录里面列出来的有21个（因为卡在了中间模块的编译上，所以也不能确定是不是所有的21个模块都是NSS所需要依赖的模块），但是有一个模块是需要提前编译的，那就是nsinstall模块，用gcc编译以后会生成一个可执行文件，下面的所有模块的编译都会去调用这个nsinstall文件。

NSS与其它模块依赖关系如下：

NSS->libpkjx->util、certdb

编译过程进行到这里时，有些文件的路径引用是错误的，所以，需要手动修改。修改完成以后继续编译的话会有下面的错误提示：error: declaration for parameter ‘\*\*\*\*\*\*’ but no such parameter

这个错误在google上搜索了，大部分是有相同的问题，但是没有提供解决方法。

有一个关于SkypeforAsterisk on Trixbo的帖子，里面也有相同的问题，回复给的答案是：安装一个asterisk16-devel的工具，而且，有人用这个办法成功过。

下载、安装都没有问题，但是，编译的错误仍然健在！

也有说需要安装openssl的，也不能解决问题。（未能解决）

## Linux下的Android Kernel编译

Linux下编译模拟器可以使用的img文件。

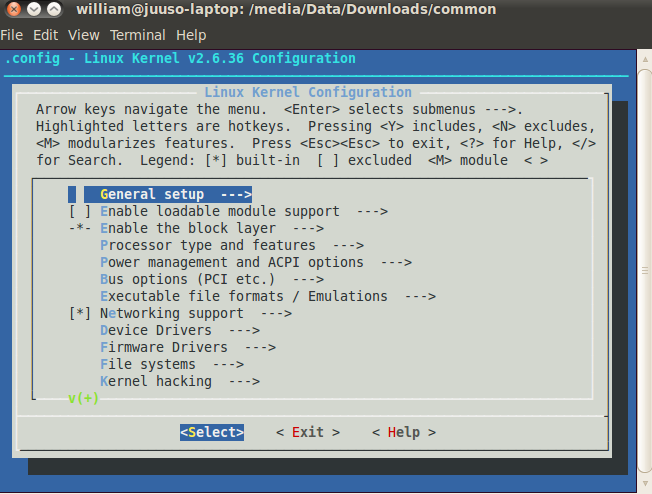
前提：

1 Android Kernel版本可以从kernel.git.android.org 下载到，一定要是 common 版的。

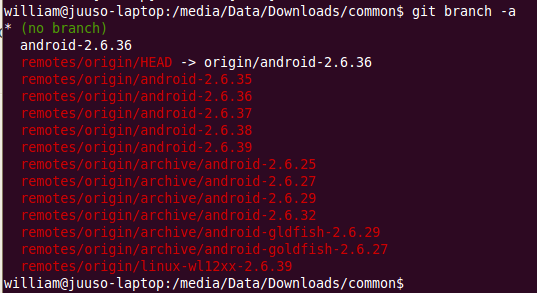
2 配置文件：必须是 Gold\_fish，只有用这个版本的配置文件进行编译，编译出来的img文件才能在模拟器里面使用。

在命令行里面进入到common目录，输入：make menuconfig 然后回车，

会弹出一个蓝色的框，如图：

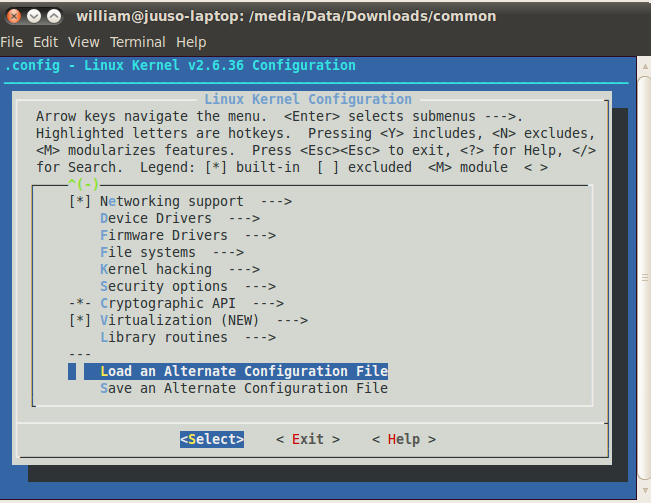


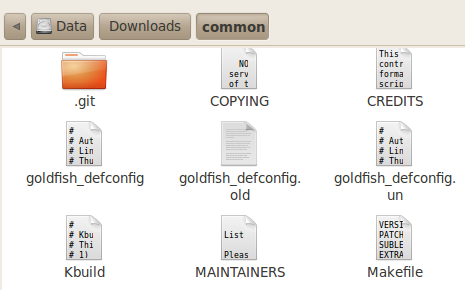
下载gold\_fish配置文件：

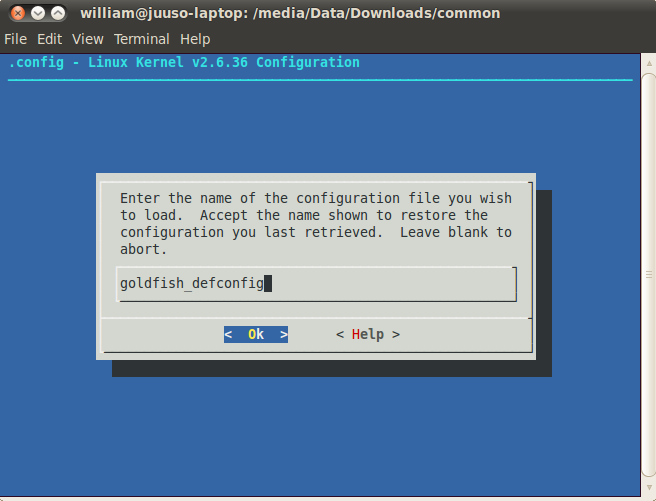


然后用 git checkout remotes/origin/HEAD 即可以下载到最新的 gold\_fish 配置文件

下载好gold\_fish的配置文件后，放到根目录下，然后从上面图中的倒数第二项：加载配置文件，







之后，进入各个选项，进行模块的增减，到最后的时候，在第一个画面，选择：exit，然后选择：保存配置。

编译的时候需要指定：make zImage ARCH=arm CROSS\_COMPILE=arm-none-linux-gnueabi-

第一个参数指定的是最终生成的img文件名。

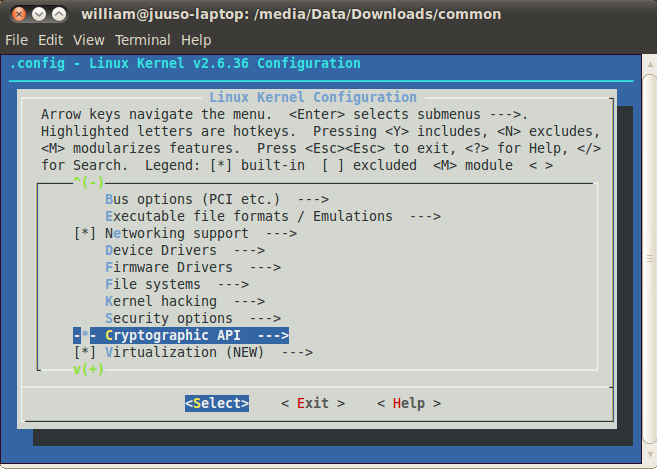
第二个参数指定的是目标平台。

第三个参数指定的是编译器所在的文件夹，make会在后面自动添加上“gcc”.

编译完成以后，就可以将common/arch/arm/boot下的zImage文件拷贝到工具目录下，并用它启动一个模拟器了。

Emulator.exe –avd avd-001 –kernel zImage

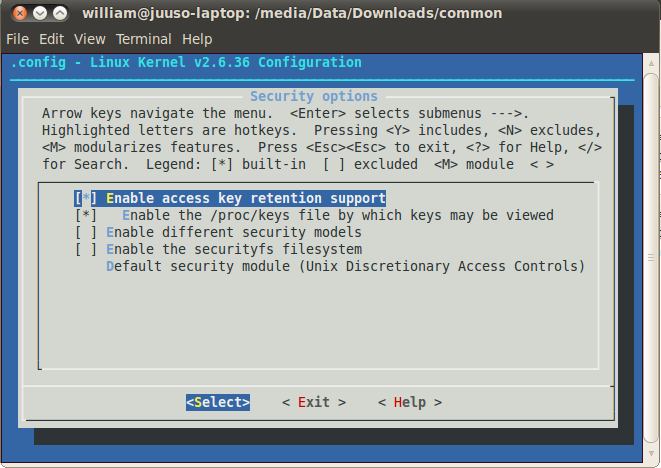
这个一般不容易出问题。但是需要注意的是：如果想编译 eCryptfs模块（在fs目录下）它是依赖两个模块的，Crypto 与 Key 。



Crypto在第一页下方，Key不太好找，而且，名字也并不是Key，搜寻路径如下 ：

Cryptographic API上面的一项是：Security options --->

进入后，将第一项启用，这时，第一项的后面会多出一个新的项，这个新的项就是 Key



之后，在 File Systems-> Miscellaneous filesystem下找到

eCryptfs filesystem layer support (EXPERIMENTAL)，并将它选中[\*]

再进行make操作，就可以将eCryptfs编译到内核中。

## linux下编译Android平台可直接运行的二进制文件

在Linux下编译的二进制文件，只用用arm-linux-gcc静态编译的文件才可以在Android下运行，动态编译的文件因为搜索不到所依赖的库文件，会提示“not found”。

拿helloworld为例：

如果只有一个文件，那么可以直接指定静态编译：

arm-linux-gcc –c helloworld.c -static

如果有多个相互依赖的文件，则需要先将main函数所在文件以外的文件进行编译，然后将生成的.o文件，打包成静态库文件，再用生成的静态库文件，对main函数所在的文件进行编译。

例如：main.c 里面引用到了hello.h 和 world.h

hello.c 中只有一句输出：hello

world.c 中只有一句输出：world

arm-linux-gcc –c hello.c // 生成hello.o 文件

arm-linux-gcc –c world.c // 生成world.o 文件

arm-linux-gcc –c main.c //生成main.o文件

arm-linux-gcc –o lib.o main.o hello.o world.o // 生成 lib.o 文件这个文件是一个动态库文件，不能在Android上运行，所以不可用！

arm-linux-ar –r lib.a \*.o // 生成静态编译的库文件lib.a

arm-linux-ranlib lib.a // 为lib.a中的函数进行索引，否则是无法在程序中调用的。

arm-linux-gcc –o binfile lib.a –static //生成静态文件。

将静态文件拷贝到android环境中，就可以用adbshell中的命令执行了：

进入到相应目录，./binfile

即可以看到输出的文字：hello world

## Makefile实例

Makefile 其实就是一系列的命令的集合（类似于bat命令），它可以很方便的对复杂的工程进行自动编译。

还拿上面的例子来说明，main.c hello.c world.c hello.h world.h

编译的详细过程上面已经给出了，所以直接开始makefile，先大致说一下makefile的语法：

目标文件：生成目标文件的依赖1 2 3 4 …

生成目标文件的命令行。

上例就可以把Makefile写成：

binfile:main.o hello.o world.o

arm-linux-ar –r lib.a main.o hello.o world.o

arm-linux-ar -ranlib lib.a

arm-linux-gcc –o binfile lib.a –static

main.o:main.c hello.o world.o

arm-linux-gcc –c main.c

hello.o:hello.c hello.h

arm-linux-gcc –c hello.c

world.o:world.c world.h

arm-linux-gcc –c world.c

执行过程是这样的：先检查目标文件，如果目标文件所依赖的文件都存在，则往下执行，否则的话执行每一个依赖项。类似于递归调用。

Makefile还有一点是值得学习的，就是它的一些公共定义，比如说上面的例子，重复的东西很多，有些繁琐，那么，同样的东西，我们可以用同一个定义代替。如下：

CC=arm-linux-gcc # 这个是CROSS\_COMPILE的简写，

OBJ=main.o hello.o world.o #OBJ = objects 是Makefile里面的一个规约。

$@ 代表目标文件

$^代表所有依赖

$< 代表第一个依赖

调用的时候用$CC 调用 ，但是为了排除有 CC= arm-linux-gcc –o 这样的情况产生不可知的错误，都用：$(CC)调用。

这样的话，上面的例子就可以简写为：

CC=arm-linux-gcc

AR=arm-linux-ar -r

RANLIB=arm-linux-ranlib

OBJ=main.o hello.o world.o

binfile:$(OBJ)

$(AR) lib.a $(OBJ)

$(RANLIB) lib.a

$(CC) –o $@ lib.a –static

main.o:main.c hello.o world.o

$(CC) –c $<

hello.o:hello.c hello.h

$(CC) –c $<

world.o:world.c world.h

$(CC) –c $<

将Makefile保存以后，通过命令行进入到当前目录，输入：make 回车

binfile就可以生成成功了。