Encrypto

Programmer's guide

Allgeimeine Aufbau

Die Code des Applikations ist in logische Einheiten (und anhand diesen Foldern) verteilt. Für jede *.c File gehört eine *.h File mit gleichem Name, die die Include-Sachen enthalten, und die Funktionen des Files deklarieren. Außer diesen Foldern findet man nur drei Dateien:

main (.c und .h)

Diese beschreiben die Anfangspunkt des Applikations. Aus dem Terminal die CLI-parametern werden zuerst in einem Datenstruktur (viele char* -s) eingeladet. Einige müssen aus Äußerem erreichbar sein, diese sind globale Variablen (und später mit Keyword `extern` erreicht), andere brauchen wir nur Lokal.

Hier installieren wir auch einem Signalhandler für Signal `SIGINT`, mithilfe von wem wir `^C` korrekt behandeln können, um die Applikation zum Halt kommen zu lassen.

Im Funktion `start` allokieren wir 5 FIFO-s, je für eine spezifizierten Task - n_miso ist für Netwerk Master-In-Slave-Out, i_mosi ist für Interface Master-Out-Slave-In, usw. Diese sind je `NxL` groß, also `N` unterschiedliche Dateien können gespeichert werden, je `L` lang.

Auch hier werden zwei Funktionen angerufen, eine für den Initialisation des Interfaces, andere für des Netzwerkes.

defines (.h)

Diese ist die "Settings" file des Programmes. Jede #define Direktiv ist hier angegeben.

Folders

file - Für Fileoperationen

1. **io.c**
    1. void log_file(char* tag, char* client, char* message) - Screibt eine Zeile in die Logfile aus.
    2. char* load_log() - Ladet die ausgeschriebene Zeilen aus der Logfile ein in einem neu allokierten char*.
    3. void write_pem(const char* type, char* data, const char* file) - schreibt eine PEM-formattiert File aus.
    4. char* read_key(char* file) - Ladet eine PEM-formattiert Datei ein.

interfaces - Für Benutzeroberflächen

1. **interfaces.c**
    1. void interfaces(char* param) - eine 'switch' für CLI oder GUI. Jede nicht `cli` Eingabe werden als `gui` behandelt. Die neue Threads werden gestartet, und diese Thread wird zu eineim Halt gebracht - bis einem SIGINT.
2. **cli.c**
    1. void *cli(void* params) - wenn es Daten in dem i_mosi gibt, diese Daten ausschreiben (formel: carriage return, eine Zeile nach oben, löschen diese Zeile, ausscreiben, zurück nach unten). Startet auch eine neues Thread:

    2. void *listener(void* params) - wenn der Benutzer aus dem Tastatur Daten eingeben, speichern in i_miso.
3. **gui.c**
    1. static void start(GtkApplication* app, gpointer user_data) - Für den Aussicht des GUIs
    2. void* gui(void* params) - Eintrittspunkt des GUI-kodes.
    3. void send_clicked() - wenn ENTER oder Send gedrückt wird, speichern wir die Eingabe in i_miso.
    4. void* handler(void* params) - falls i_mosi nicht leer ist, schreiben wir seine Daten aus.

misomosi - Für memory

1. **memory.c**
    1. void allocate_str_array(char*** array, int size, int piece_size) - eine zweidimensionellen Char-Array allokieren aus einem char***
    2. void free_str_array(char*** array, int size) - die oben allokierten array freilassen.
    3. void shift(char*** array, int size) - Die ganze FIFO nach links schiften.
2. **misomosi.c**
    1. void write_comm(char*** ch, char* in) - eine der FIFOs mit Daten schreiben.
    2. int read_comm(char*** ch, char** out) - eine der FIFOs auslesen, falls es Daten innerhalb gibt, true zurücktreten, und diese Element löschen (durch schiften).

networking - Für Server- und Clientoperationen

1. **networking.c**

    1. void networking(char* param) - abhängig von dem CLI-parametern entweder Server oder Client-Thread starten, und auch eine Router-Thread:
    2. void* router(void* param) - die FIFO-s handeln

2. **server.c**

    1. void* server(void* params) - Eintrittspunkt für den Server, fängt an zu listen-en (auf dem spezifierten Port), started andere Threads(callback, serverread).
    2. void* host_info(void* params) - Eintrittspunkt für den Information-server (der ermöglicht den Admin-Oberfläche).
    3. void* serverread(void* params) - Für einkommene Daten (-> n_miso)
    4. void s_handle_input(char* in) - Für einkommene Daten (Hilfsfunktion)
    5. void* callback(void* params) - Verteilung des einkommenden Daten (nach jede verbundene Client)

3. **client.c**

    Sehr ähnlic zu server, aber:

    1. void handle_input(char* in) - parsieren den einkommenden Daten - message, status, key request, key response, key negotiation -> entsprechende Hilfsfunktionen, sehr trivial (sehen Sie den Grafikonen unten).

text - für Textmanipulation

Jede file kann auch als eine Library benutzt werden.

1. **encryption**
    1. **aes.c**
        1. void handle_aes_key(char* key) - Diese verarbeitet den Session-Key, den unsere Partner uns gesendet hat - eine globale `aes-key` wird dafür verwendet.
        2. char* get_aes_key() - Diese gibt den Key (oben gespeichert) zurück.

3. void encrypt_aes(char** content) - Eine string verschlüsseln mit dem oben gespeicherten Key.
4. void encrypt_private(char** rsa, int len) - Den private-key mit dem Hash eines freigewählten Passwords mit AES256 verschlüsseln.
5. void decrypt_private(char* passwd, char* rsa, int len) - Den verschlüsselten private-key zurückbekommen
6. void generate_aes() - abhängig von den `#define MODE` generiert eine Verschlüsselungsschlüssel für AES256, und macht den Handler daraus auch.

2. **randomdata.c** - enthielt die 3 verschieden Weisen von Randomzahlgenerierung - /dev/random /dev/urandom, und RDRAND. Für diese letzte eine Assembly Funtkion benutzt, diese steht hier:

```
rdrand:
RDRAND %ax
JNC rdrand

ret
```

3. **rsa.c**
    1. char* load_public_key() - Ladet den public key aus dem File ein.
    2. char* encrypt_rsa(char* key, char* content, int len) - Gibt den verschlüsselten Wert des Contents zurück.
    3. void decrypt_rsa(char* content, int len) - Decryptiert den Content mit dem eigenen Private-Key des Benutzers
    4. void generate_keypair() - eine neue Keypair generieren

2. **base64**
    1. char* encode_base64(uint8_t* str, int len) - die enkodierte Representation des Striges zurückgeben (mit Hilfe von char to_base64_char(uint8_t c))
    2. uint8_t* decode_base64(char* base64) - die dekodierte String des Base64-Representations zurückgeben (mit Hilfe von)uint8_t get_char(char c).
3. **http.c**
    1. void build_http(char* header, char* content) - Content zum Header hinfügen.
    2. char* get_http(char* str) - Content aus einem HTTP-Struktur