Garage Opener

1 Purpose

Purpose of the Garage Opener soft- and -hardware is a device, which allows to open and close the garage door of my house remotely and to monitor remotely whether the garage is open.

The product requires a certain amount of security to prevent unauthorized access to the house.

Remote access is provided through the in-house LAN and WiFi.

Remote clients should run on mobile devices (Android or Apple) and on PC's.

An administration tool is used for basic network configuration of the setup and for device- /user-administration.

2 Development History

The project started with an Arduino UNO, with an Ethernet shield and a rather simple Arduino sketch on the hardware side. For the clients I decided to use Delphi which would allow me to have shared source code for the administration and all clients on different platforms.

Adding the security layer and the administration led rather quickly to a much more complex software. Which caused me to break the Arduino part in to several C++ modules, which allowed me and develop and test them independently.

The final step will be an independent software framework for Arduino and Delphi, which will allow to run RPC like commands on an Arduino.

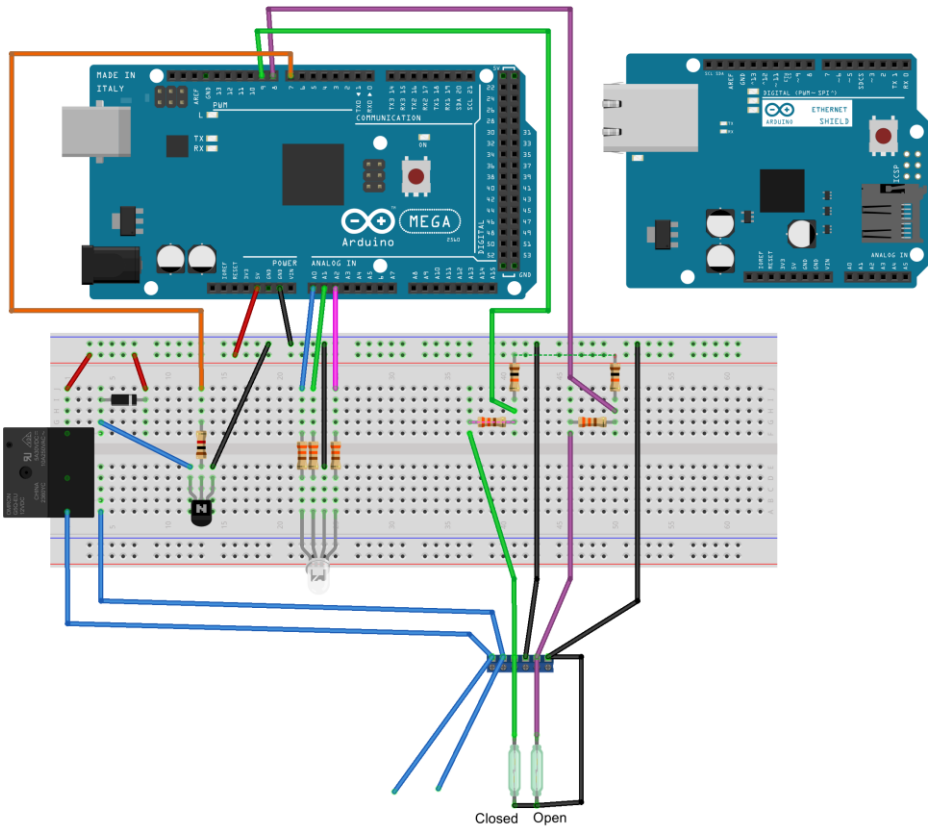
3 Hardware

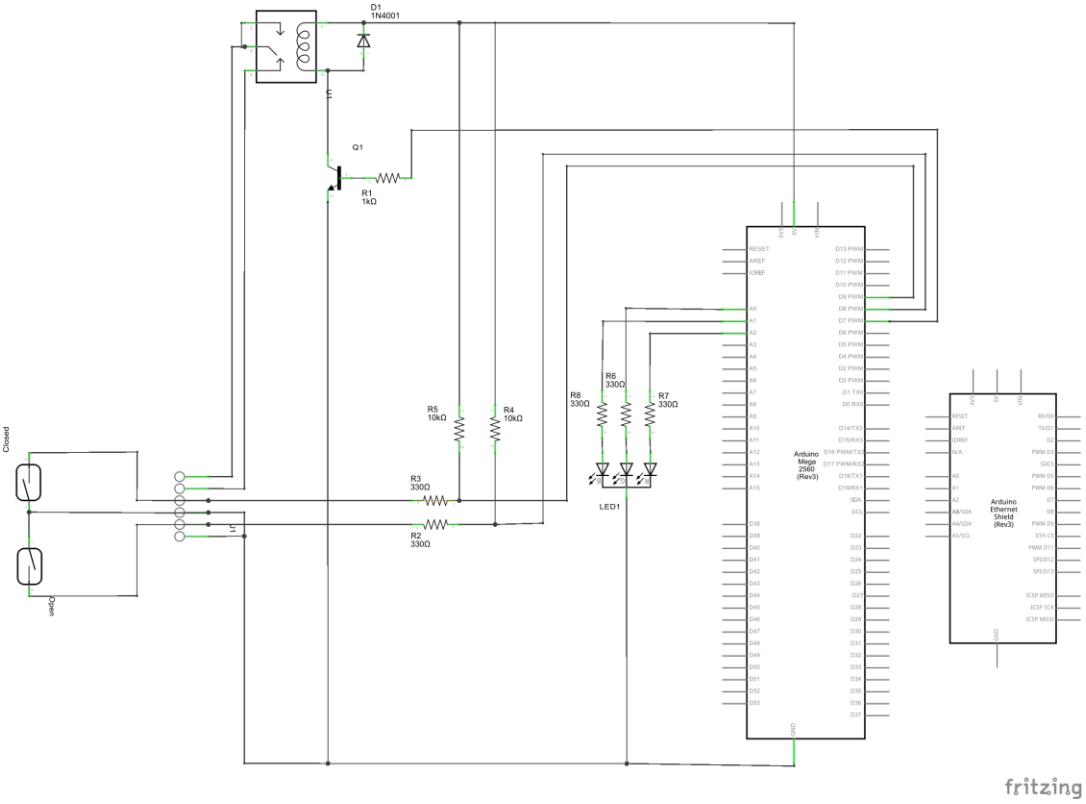
The project started with an Arduino UNO and an Ethernet shield. Code- and memory size later required to use an Arduino Mega. Later I added the Adafruit CC3000 WiFi shield. Initialization of this shield has only been tested in an earlier stage.

A SD-Card slot is required for network configuration.

External contacts are the two reed switches as sensors for the garage door status. The relay output is connected to the garage door button in the garage.

There is a RGB status LED, used to display the status of the door.


MADE IN ITALY
MEGA
Arduino
ETHERNET SHIELD
Closed
Open
fritzing


D1
1N4001
Q1
R1
1kΩ
R2
330Ω
R3
330Ω
R4
10kΩ
R5
10kΩ
R6
330Ω
R7
330Ω
R8
330Ω
LED1
Closed
Open
Arduino Mega 2560 (Rev3)
Arduino Ethernet Shield (Rev3)
fritzing

4 Software

4.1 Modules

There are three software modules:

- Arduino control module
- User client
- Administration tool.

4.1.1 Arduino control

This C++ software contains:

- Initialization module for reading configuration data and initializing the module
- HTTP Server
- Sensor module to detect the garage door state
- RGB LED module to display the device status
- Motor control module to operate the garage motor

4.1.2 Administration tool

This Delphi software contains:

- Configuration editor to edit basic configuration
- User/Device editor for user administration
- HTTP userdata update for the Arduino
- HTTP client initialization to send configuration data to the clients

4.1.3 User client

This multi-platform Delphi software is used to poll the garage status from the Arduino and to open and close the garage door.

4.2 Environment

The Arduino part was developed using Visual Studio 2015 with the Visual Micro Arduino plugin (http://www.visualmicro.com) and the Arduino IDE 1.6. (www.arduino.cc).

For the client module and the admin tool Delphi XE5 was used – in my case the professional edition with the mobile add on.

4.3 Data transfer protocol

There is a simple HTTP server implemented on the Arduino. Requests are transferred as HTML-like GET commands. POST should also work with little changes.

The command URL has the following structure:

```
<Command>?<ParamName1>=<ParamValue1>&<ParamName2>=>ParamValue2>...
```

Example.

```
CONNECT?USERID=12345
```

The reply from Arduino is kept in a simple XML format – following this ddt:

```
<!ELEMENT html (head?, RESULT, DATA*)>
<!ELEMENT head (title?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT RESULT (#PCDATA)>
<!ATTLIST RESULT
      STATUS (OK|NO_SESSION|NO_USER|FAILED|UNDEF) #REQUIRED>
<!ELEMENT DATA (#PCDATA)>
<!ATTLIST DATA
      ID ID #REQUIRED>
```

Example:

```
<html>
      <head>
            <title>Garage Door</title>
      </head>
      <RESULT STATUS="OK"></RESULT>
      <DATA ID="SESSIONID">12345</DATA>
</html>
```

4.4 Configuration

The Arduino is reading the initial configuration at startup, if a SD card is present and the configuration file "Arduino.cfg" is located in the root directory of the SD card.

When no configuration file is found, then the software tries to find the configuration in the Arduino-EEPROM.

The configuration is set up as a packed structure – in the file and in EEPROM. Start and end keys are used to validate the structure.

The basic configuration is used to initialize the network connection and the secret key.

4.5 Security

The security implementation allows for authentication and data integrity – but not confidentiality and availability.

I'm not a security expert – so please don't rely on my implementation. There might be serious flaws, which could provide easy access for experienced intruders.

There are some assumptions, on which the security implementation is based:

- The secret keys are saved in unencrypted configuration files on all devices. It is assumed, that no third party can access these files.

- During device configuration the secret keys are transferred over the LAN as plain text. It is assumed, that no third party is listening into this communication.
- The software uses the SipHash (https://131002.net/siphash/) procedure to verify integrity and authenticity of messages. It is assumed, that this hash function is secure.
- The software does not distinguish between users and devices. So each device which is participating in the communication only has one user-/device id. In the following I only use the term "client id".
- The Arduino can handle only one communication stream at a time. Two or more devices trying to communicate with the Arduino, can finally prevent any communication, by breaking off the other client's communication sessions.

4.5.1 Protocol

1. Communication is started by the client which is sends a CONNECT message to the Arduino, which contains the client id.
2. If the client id is known to the Arduino, the it will reply with a random 31 bit session id. In all future messages within this session, the client will use this session id. The session is valid until the next CONNECT message is sent to the Arduino.
   There is a message counter, which will give an index for each message.
3. The client can now send command messages. (See the data transfer protocol above). The messages contain a command and a number of parameters as key-value pairs. There are also virtual parameters, which are not sent:
   - Session Id
   - Message index
   - Client ID
   - (Client password)

   The command, the non virtual keys and values and the the virtual parameters (also keys and values) are then fed into the hash function – using this sequence. The resulting 32 char hexadecimal hash value also has to be one parameter of the command string.
4. The Arduino will evaluate the command string and will also calculate the hash value. The message is valid, when both hash values are equal. Otherwise it will reply with a "NO_SESSION" error.
   The client password is only required for special commands (at the moment only for user administration). Passwords are not saved on client devices. So it can add some security, if a passwords are required for certain commands.

4.6 Software modules

4.7 Third Party

The software contains several source code modules from third parties:

- C++ SipHash 2-4 for Arduino: Forward Computing and Control Pty. Ltd. , www.forward.com.au

- C++ Webserver derived from WebDuino: Ben Combee, Ran Talbott, Christopher Lee, Martin Lormes, Francisco M Cuenca-Acuna, github.com/sirleech/Webduino
- Delphi XmlParser: Stefan Heymann, www.destructor.de
- Delphi SipHash is derived from the java SipHash package: Martin Bosslet , Martin.Bosslet@googlemail.com

5 Open Issues

- Modularization is not finished yet: There are still some source code files which contain both garage related code and framework related code.
- The Arduino sensor module should be converted to uses the debouncing approach described by Elliot Williams (http://hackaday.com/2015/12/10/embed-with-elliot-debounce-your-noisy-buttons-part-ii/)
- Security should be protected against replay attacks, by adding expiration time for sessions.
- Messages from the Arduino module to clients are not yet secure.
- At the moment only local ip addresses are used to connect to the Arduino. To allow access over the internet, the clients need to be capable to distinguish between local WiFi and external internet addresses and would need to use a different URL if outside the LAN.

6 Disclaimer / Licenses

6.1 Usage

You may use the source codes for the development of free and/or commercially sold applications. You may also modify the codes and distribute them.

6.2 Distribution

You may freely distribute these source codes. However, if you distribute the originals, you must distribute all original files as well. If you distribute modifications of the *source* codes, these must also be free.

You must not charge any fees or royalties for the usage of the source codes or their derivates.

6.3 Third party source codes

This software uses source codes from third parties. If you use this software or these source codes, then it is your responsibility to meet their license requirements.

6.4 The AS IS paragraphs

Please note that this package is provided "AS IS" and without any express or implied warranties, including, but not without limitation, the implied warranties of merchantability and fitness for a particular purpose. In other words, I accept no liability for any damage that may result from using the provided codes or programs.

I do not warrant that the functions contained in the source codes will meet your requirements or that the software operation will be uninterrupted or error free.

In no event will I be liable to you for any damages, including any lost profits, lost savings or other incidental or consequential damages arising out of the use or inability to use these codes, or for any claim by any other party. The entire risk as to the results and performance of the codes is assumed by you.