Brypt Winter Progress Update

Sean Caster, Vincenzo Piscitello, Adam Barton, Ryan Howerton, and Terri Hewitt

Senior Design, Group 76

Winter 2019

Abstract

The goal of this project will be to implement a secured, decentralized meshnet with low minimum resource requirements to enable the connection of a wide variety of low-end devices which lack a general purpose operating system. Our objective will be to investigate the amount of security offered and resources required by various cryptosystems in order to provide organizations interested in securing their low-end device networks with a quantified estimate of the costs and benefits. In addition, we hope to accomplish graceful degradation on our network, meaning that the elimination or temporary unavailability of a given node will have minimal negative impact on the inter-connectivity of other nodes. This will encourage the securing of many resource limited devices which play a critical role in meeting a wide variety of human needs.

February 25th, 2019

Contents

1 Introduction 2
- 1.1 System purpose . . . 2
- 1.2 System scope . . . 2
- 1.3 System overview . . . 2
  - 1.3.1 System context . . . 2
  - 1.3.2 System functions . . . 2
  - 1.3.3 User characteristics . . . 2

2 Current Project Status 3
- 2.1 Networking . . . 3
  - 2.1.1 Polymorphic Communication . . . 3
  - 2.1.2 Threading . . . 3
  - 2.1.3 Messaging . . . 3
- 2.2 Command Handling . . . 4
  - 2.2.1 Arduino . . . 4
- 2.3 Research . . . 4
- 2.4 Crypto . . . 4
  - 2.4.1 Arduino . . . 4
  - 2.4.2 C++ . . . 6
  - 2.4.3 Electron Framework . . . 6

3 Remaining Items to Finish 7
- 3.1 Networking . . . 7
  - 3.1.1 PeerWatcher . . . 7
  - 3.1.2 Wi-Fi . . . 7
  - 3.1.3 Long Range Radio (LoRa) . . . 8
  - 3.1.4 Bluetooth Low Energy (BLE) . . . 8
- 3.2 Research . . . 8
- 3.3 Crypto . . . 8
  - 3.3.1 Key Exchange . . . 8
  - 3.3.2 Merging Crypto Class into the Message Class . . . 9

4 Problems Encountered and Their Solutions 9
- 4.1 Javascript and Electron . . . 9

5 User Interface Screens 9

6 Interesting Code Snippets 13
- 6.1 Desktop application supports Windows and OS X . . . 13
- 6.2 AwaitObject and AwaitContainer to delay responses to requests that require input from other nodes. 13

7 Relevant Images 16
- 7.1 Brypt Architecture Diagram . . . 16

1 Introduction

1.1 System purpose

As it stands, a wide variety of mission critical tasks for many utility grids (water, power, etc.) are accomplished by low power embedded devices. The mass quantity of devices required to fulfill the operational needs of these networks has created a systemic problem of sacrificing security to reduce the cost of power. The purpose of our system, Brypt, will aim to provide a solution for architects of these networks to have the best of both worlds. In minimizing the cost of power while maximizing the security of ad-hoc mesh networks will demonstrate numerous benefits.

In addition to offering an implementation of what our investigations suggest to be an excellent general purpose meshnet cryptosystem, we will generate useful data about the costs of resource asymmetric cryptosystems that can be applied to the design of more special purpose meshnets by organizations looking to secure their particular resource distribution optimally. These findings will be published in a final report alongside the implemented network to help in the potential design of such projects.

1.2 System scope

The ultimate goal of this project will be to produce the implementation and verification of the secure networking paradigm succinctly named Brypt. Brypt combines the main cornerstones of the software, bridging and encrypting. The implementation will provide a central server, binaries for node endpoints, and a security protocol working on top of the session layer. In completing the work on this project our team aims to provide a novel solution to security in interconnected networks of embedded and general purpose devices. Some of the benefits Brypt will provide include the increased viability of secure cryptographic primitives in power constrained environments across a number of different communication technologies.

1.3 System overview

1.3.1 System context

The primary context of our system will be provided through a cloud based application served through our central server. Within our web application three primary interfaces will be accessible by the user. The first interface will be the base information page for Brypt; this page will contain information about the system, it's requirements, and downloads. The second interface will be the node management screen; these pages will provide node network authorization. The final user interface element of our system is the dashboard page which will contain information pertaining to the status of the user's connected clusters and aggregation of the nodal data. The only requirement of the user will be access to a browser capable device. Outside of our application interface, users will need to interact with our embedded devices and/or binaries for their system of choice.

1.3.2 System functions

Our system capabilities will be provided through our central server, networking implementation, and security protocol. The complete system will need to operate over several communication standards, variable weather conditions, and battery constrained environments. An active portal to a full internet connection will be required for or mesh network to make and maintain connections to their respective clusters.

1.3.3 User characteristics

There is a single user class within our system; this permissions class will act as the manager and maintainer of their organization's clusters. There can be multiple users with access to an association's clusters, although there may be one owner. An individual wanting to use our system and application should be familiar with technology, but may come from all different walks of life. We can expect users to have varying levels education and disability, so accessibility should be built into the user interface.

2 Current Project Status

2.1 Networking

We have made significant progress on the overall networking architecture being used throughout our Brypt ecosystem. This has been a difficult task as we had to devise a method of communicating between WiFi, LoRa, and BLE nodes with the possibility of adding new technologies. Our implementation makes use of a polymorphic communication structure, threading connections, and uniform messages between devices. We hope that this modularity allows our system to be extended to several different hardware configurations; currently we are making use of general purpose computer as our client, Raspberry Pis as our coordinators, and Adafruit Feathers as our nodes.

2.1.1 Polymorphic Communication

We have designed our code to be as polymorphic as possible to allow us to have one set of code that runs on all devices–laptops, Raspberry Pis, or feather nodes. Our system works through a script to identify the parameters of each device, such as what communication technologies are supported. From this the code when started will configure to the type of communication technologies available. The polymorphic classes greatly increase the organization and reduce the complexity to our code.

2.1.2 Threading

Threading was required in both our Brypt operation network and our Electron application for different purposes.

- Raspberry Pi
  In order to allow our Raspberry Pi coordinators to communicate with multiple sub clusters and its own leaf nodes we have developed a method to allow initial connections to a single control process which are then redirected to a new child process for normal communication. Following a handshake process WiFi devices first connect to a pre-designated port on the coordinator device to gain a new connection port for full communication within the network. This method allows the control connection to handle new devices fairly quickly without requiring a significant amount of resources in the main process.
- Electron
  Electron by default utilizes a Main and Renderer process given that it is built of Chromium and Node.js. We had to make use these default processes for our desktop application with the addition of a Brypt connection thread forked from the Main process. The Main process handles the state of the application as well as launching browser windows and other native operating system interactions. Each browser window launched by the Main process is handled by a Renderer process to provide the User Interface visuals and interactions. The Brypt connection thread handles communication with the Brypt network via ZeroMQ in a Request/Reply format over TCP. There are two main forms of interprocess communication (IPC) in use to get these three processes talking to each other. The first method of IPC allows the Renderer to request actions to be taken by the Main process which may include launching a new window or sending and collecting information from the Brypt connection process. The Brypt connection is managed by an interface which forks the connection logic and directs actions to the child process. A JavaScript Map was used to track IDs and callbacks for Brypt connection jobs.

2.1.3 Messaging

Messaging between devices on our network is implemented through a message queue containing instances of messages–a class we have created–which contains all necessary information for the routing of the data as well as authentication and encryption of the data. Additionally, this message queue allows for our coordinator nodes to better handle their many connections. Coordinators can loop through this message queue, use the stored handle to determine the output/input location, and take actions on each individual message. Additionally, this enables the coordinator nodes to take actions on their control connection while other threads handle each of the messages. When the main thread of the program finishes handling the control connection it can then continue where it left off handling the messages in the queue.

2.2 Command Handling

Received messages are directed to a Command handling class that operates within the node instance to respond appropriately. Currently, we are able to handle actions that direct network information and sensor reading responses to the client. In the specific case of a sensor Query request, the receiving coordinator creates an AwaitObject that is placed into a hash table (unordered_map) that uses the MD5 hash of the request as the key. Then the request is flooded down the network using a specialized Notifier class. The Notifier class utilizes a ZeroMQ Publisher socket, however, if a node uses a technology that cannot subscribe, the notifier contacts that node directly. Upon receiving a notification the node then responds to the coordinator with the desired information. When a node response is received by the coordinator it is matched to the AwaitObject and appended to the aggregated response data. Finally, each cycle the coordinator checks for fulfilled messages to be sent off to the requesting client or parent coordinator. An AwaitObject request is considered fulfilled if all expected nodes respond or the expiration time is passed.

2.2.1 Arduino

The Feather devices run code written in Arduino's flavor of C++ and thus makes it a bit more difficult to connect them into the network.

Messaging Class
We currently have an Arduino equivalent version of our messaging class implemented in Arduino allowing us to use the same message class to communicate with Feather devices. This class has been tested to ensure it has the same functionality. The crypto function still needs to be implemented into the Arduino message class.

Wi-Fi Setup
When connecting these feather devices to the network they first scan the network looking for Brypt networks to connect to. Once found, the device then hosts a captive portal for a user to connect and select a network for the Feather to connect to. When the network is connected, the Feather device begins the handshaking process to become a node on the network.

2.3 Research

On the research side, we currently have a baseline program and programs to loop over computing an HMAC, generating ciphertext, and decrypting ciphertext in Arduino for all hashes and encryption algorithms we plan to study. We are able to run the programs on the BlueFruit featherboard and gather serial data output from the Hydra power supply in a CSV file. We also wrote a Python script which could then be used to parse the CSV files and calculate the average supplied voltage, supplied current, watts, and joules over the time period which data was being collected.

2.4 Crypto

2.4.1 Arduino

On the Feather boards, we currently have a crypto test suite which computes the SHA2, Blake2s, HMAC(SHA2), and HMAC(Blake2s) hashes on a static message then logs the result to the console (seen in the figure below). The test suite also encrypts the message using AES-CTR-128 and AES-CTR-256 and then decrypts the corresponding ciphertext and outputs the results.

```
SHA2 Hash:
D7A8FBB37D7809469CA9ABCB082E4F8D5651E46D3CDB762D2D0BF37C9E592
SHA2 HMAC:
3B383C66FE6688CC3DDCC2CA7FA6FD94A326CEFBD1E366C42ECCBB21BDBA
BLAKE2s Hash:
606BEEEC743CCBEFF6CBCDF5D5302AA855C256C29B88C8ED331EA1A6BF3C8812000000000000000000000000000000000000000000000000000000000000000000000000
BLAKE2s HMAC:
E31A2D526D15DD2602D998588294EAE83FDF9ED79397977268FA3E8F78DE
AES-CTR-128 Ciphertext:
B 15 95 9F 61 4F E5 29 8D DA CF EB 69 DB 55 A 0 58 58 FA 7C 6F D0 52 52 53 73 61 1B 8 2E 3 A2 D1 16 C 98 4B F5 C7 43 6C CA D5 0

AES-CTR-128 Decrypted text:
The quick brown fox jumps over the lazy dog

AES-CTR-256 Ciphertext:
9 F1 E4 64 98 3A 7D 25 30 5D 5B 86 53 86 E4 77 AE C3 59 54 FC 8 79 CE BF E9 BD 4 13 27 7 7C 21 FB C9 60 80 7C 86 6C AD 2 85 B5

AES-CTR-256 Decrypted text:
The quick brown fox jumps over the lazy dog
```

Figure 1: Feather outputting correct ciphertexts, decrypted texts, and hashes for static message

2.4.2 C++

Our team currently has, in C++, four hash functions (SHA1, SHA2, HMAC(SHA2), HMAC(Blake2s)) and two block ciphers (AES-CTR-128 and AES-CTR-256) which take in the message: "The quick brown fox jumps over the lazy dog" and outputs the corresponding hash or ciphertext. Each of the block ciphers also have a decryption function which takes in the ciphertext and outputs the corresponding decrypted text.

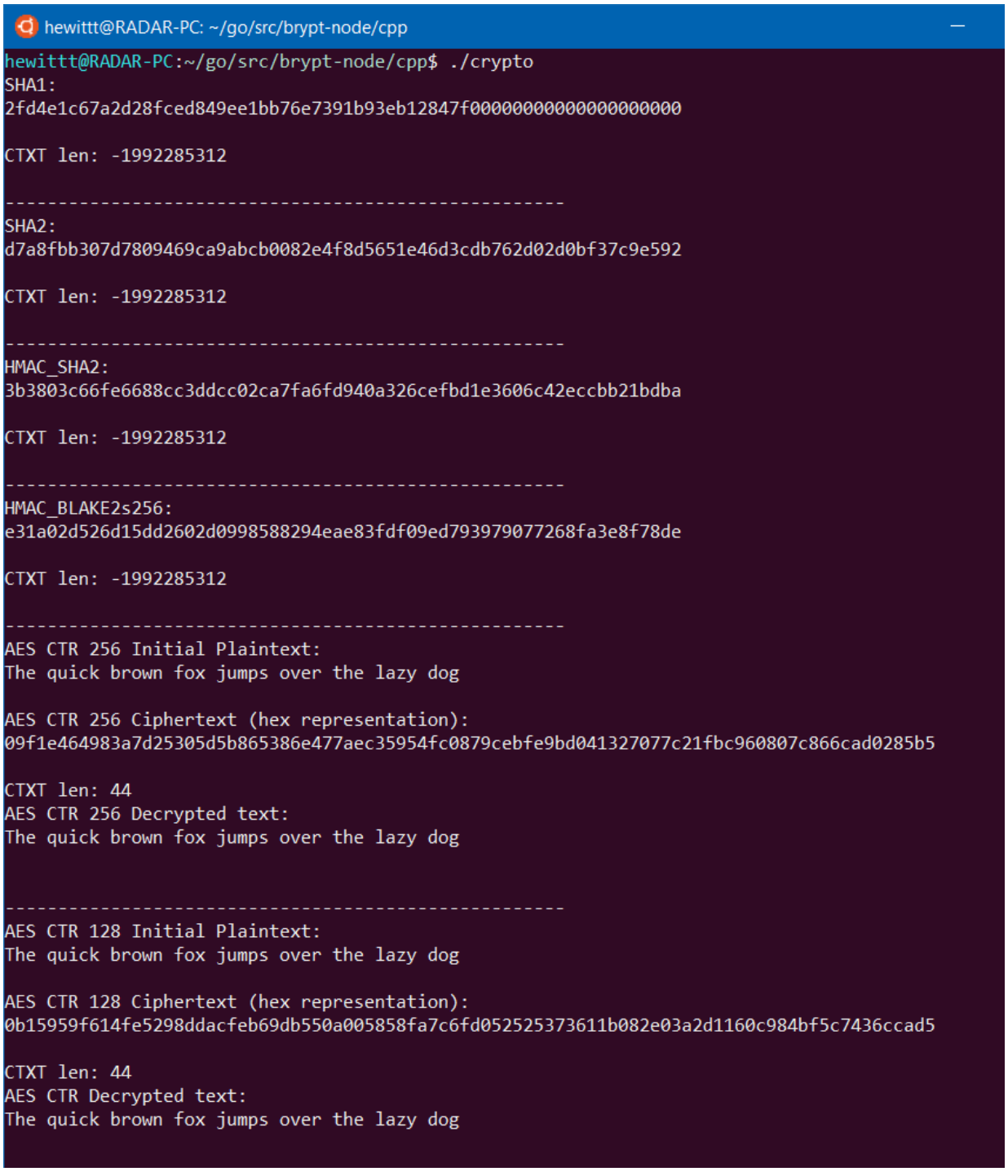

Figure 2: Cryptographic algorithms outputting correct ciphertexts and hashes

2.4.3 Electron Framework

In the desktop application which uses the Electron framework, there is a Crypto class which will get called when the user clicks on the title of the sign-in page. From there, our group has been able to call functions which take the keyed hash of a message using HMAC(SHA2) and HMAC(Blake2s) or encrypt and then decrypt a message using

AES-CTR-128 or AES-CTR-256. The figure below shows the HMAC(Blake2s) of the message: "The quick brown fox jumps over the lazy dog" which matches the HMAC computed in the solely C++ implementation.

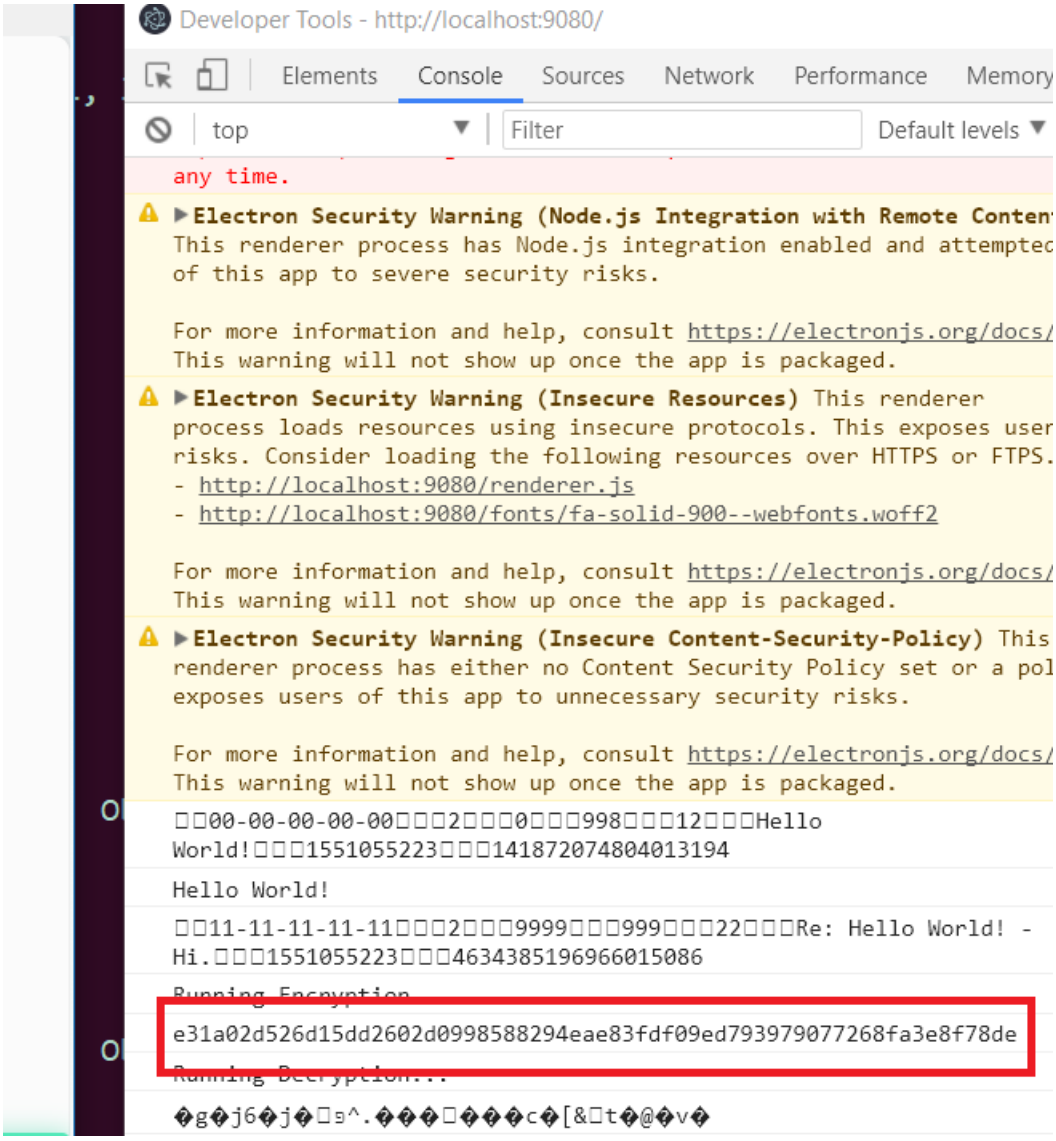

Figure 3: Proof of concept that HMAC(Blake2s) can be computed in Electron framework

3 Remaining Items to Finish

3.1 Networking

3.1.1 PeerWatcher

The PeerWatcher class is the system that a node instance uses to manage its connections state. As of now, the PeerWatcher can spawn a threaded worker to check the last update time point of a connection. If the connection was not updated within a given timeout, a heartbeat is required to ensure that the peer is still alive. Logic is still needed to send heartbeat messages to the marked nodes as well as cleaning up the connections and MessageQueue watched pipes.

3.1.2 Wi-Fi

Currently the Wi-Fi networking code on the coordinator and laptop devices is working well. We have integrated the messaging class into the connection code and have created the handshake process for adding nodes. The

code for the Wi-Fi Feather devices contains all of the necessary components to connect to the network, however, debugging still needs to be done around establishing the initial TCP connection.

3.1.3 Long Range Radio (LoRa)

As of writing this document, the LoRa connection class is still in development. Currently, only the feathers can send transmissions to the Pi's, but not vice versa. We have checked a few possible configuration issues, including frequency and modulation settings, but these were symmetric between the boards. After a period of further research, we determined that the RadioHead libraries (used in the implementation of the feather boards) has a specific format, and our current theory is that the feathers are dropping packets that do not conform to this packet structure. Unfortunately, RadioHead is not compatible with the Raspberry Pi 3B+, so to test this we will have to implement this structure from scratch. Hopefully, once this packet is implemented, we will be able to move on.

Luckily, because of the polymorphic structure of our communication class, once two-way communication is working, the devices should have full functionality (minus whatever bugs come to attention).

3.1.4 Bluetooth Low Energy (BLE)

Due to time constraints, and spurred on by the fact that Bluetooth and Wi-Fi operate on the same radio frequency (therefore making the two technologies somewhat redundant), we decided to reduce the priority of the BLE protocol, and leave it for a future stretch goal. This way, we can focus on getting the other two dissimilar technologies fully operational.

3.2 Research

The next steps for the research portion will be to finalize the Arduino programs for computing the HMACs and encrypting/decrypting the static message. Then, with the finalized programs, run them on the Feather boards and collect approximately 100 minutes worth of data from each board for each program and run the Python script on the CSV files to get an average for watts and joules.

We also plan to run these same tests on the Raspberry Pis as well. However, for that, we need to translate the Arduino programs into C++ code and install a stripped version of Raspbian on one of the Pis. Our intent with installing a stripped version of Raspbian on the Pis is to limit the number of operating system type processes which will be running in the background and give the highest priority to our test process.

3.3 Crypto

3.3.1 Key Exchange

Before exchanging keys over the network, nodes and the server will need to ensure they're talking to the computers they intend to. For this reason, each will have to independently authenticate the identity of the other. Though more robust networks use a combination of Public Key Infrastructures and asymmetric cryptography, for now we plan for ours to solve this base authentication problem with pre-flashed fingerprints/public keys for our devices. The exact nature of these fingerprints will vary depending upon the key exchange methodology being used in a given phase as follows.

- Modified Diffie-Hellman Key Exchange (DHK) Algorithm

  In order to perform authentication during key exchange in a manner that would mimic the costs of PKI usage (which is largely based around use of hash functions for signatures) we'll be flashing the hashes of the expected intermediary keys for Diffie Hellman. I.e. if the node is to send $g^n\%p$ the server will be flashed with hash($g^n\%p$) beforehand, and check that the hash of the key component purportedly from the node matches the pre-stored value. Similarly, if the server is sending $g^s\%p$, the node will be flashed with hash($g^s\%p$) and check the hash of the other key component.

- Modified Elliptic-Curve Diffie Hellman Key Exchange (ECDHK) Algorithm

  This fingerprinting will work similar to the above: if the node is to send some n*(an elliptic curve P), the server will store hash(n*P). Similarly, if the server is to send some s*P, the node will store hash(s*P).

3.3.2 Merging Crypto Class into the Message Class

Currently in the desktop application, we have a proof-of-concept Crypto class which allowed us to output computed hashes/ciphertexts to the console for testing and verification. The final step is to merge these functions into the Message class. So far, HMAC(SHA2), HMAC(Blake2s), and the encryption algorithms for AES-CTR-128 and AES-CTR-256 have been merged into the message.

What remains is to merge the decryption algorithms for AES-CTR-128 and AES-CTR-256 into the message class and setup the calls to the functions so that the class will automatically decrypt the incoming raw data after it is unpacked and encrypt outgoing data before it is packed. After that, some function parameters will need to be modified so that they accept a key and/or IV.

4 Problems Encountered and Their Solutions

4.1 Javascript and Electron

Originally, our intention was to implement the cryptographic primitives, hash functions, and key exchange protocols in Arduino, C++, and Javascript. The Arduino code would be used on the Feather boards, the C++ code would be run on the Raspberry Pi devices, and Javascript would be used on the client. However,the issue we encountered while trying to implement cryptographic algorithms which required a key or a key and an initialization vector (IV) was that all the Javascript cryptographic APIs only allowed us to call a procedure which would generate a random key and IV. This alone made it very difficult for us to figure out how to extract the key and IV which had been used in order to test that we could compute the same hash or ciphertext in Arduino and C++. The other issue we ran across was that we were unable to figure out how to set our own keys and IVs in Javascript which would make it impossible to decipher any messages received from the network. This led us to switch over using the Electron framework which would allow us to essentially reuse our C++ code on the client side.

5 User Interface Screens

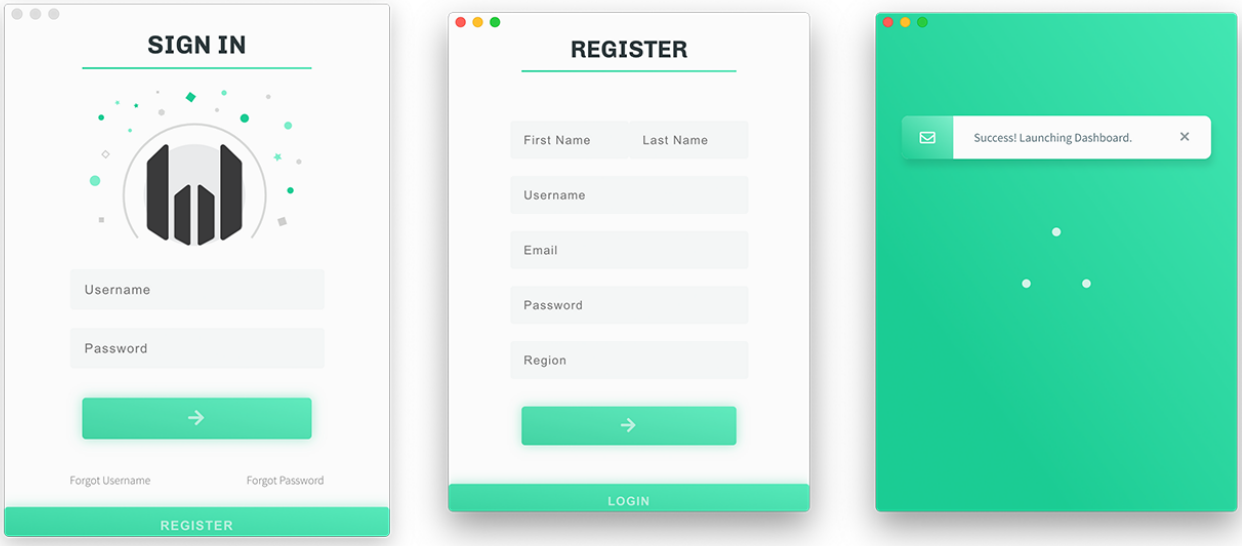

Figure 4: Access for desktop application using the Electron framework

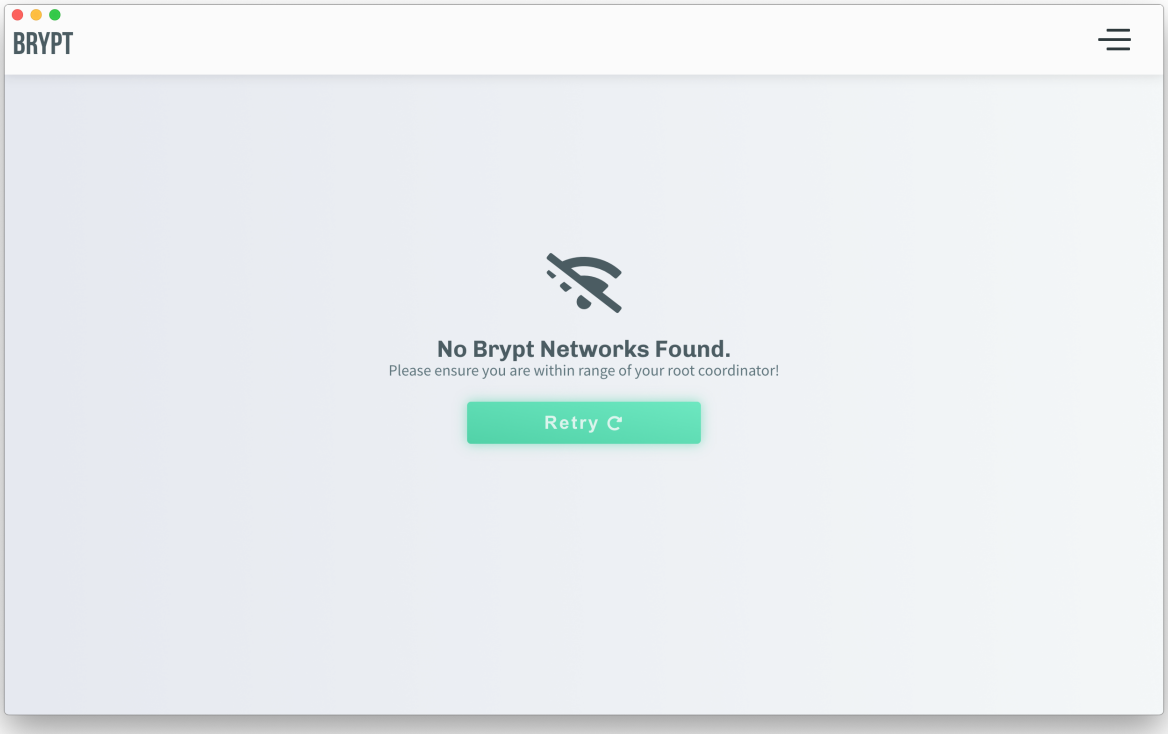

Figure 5: Brypt network access point not found

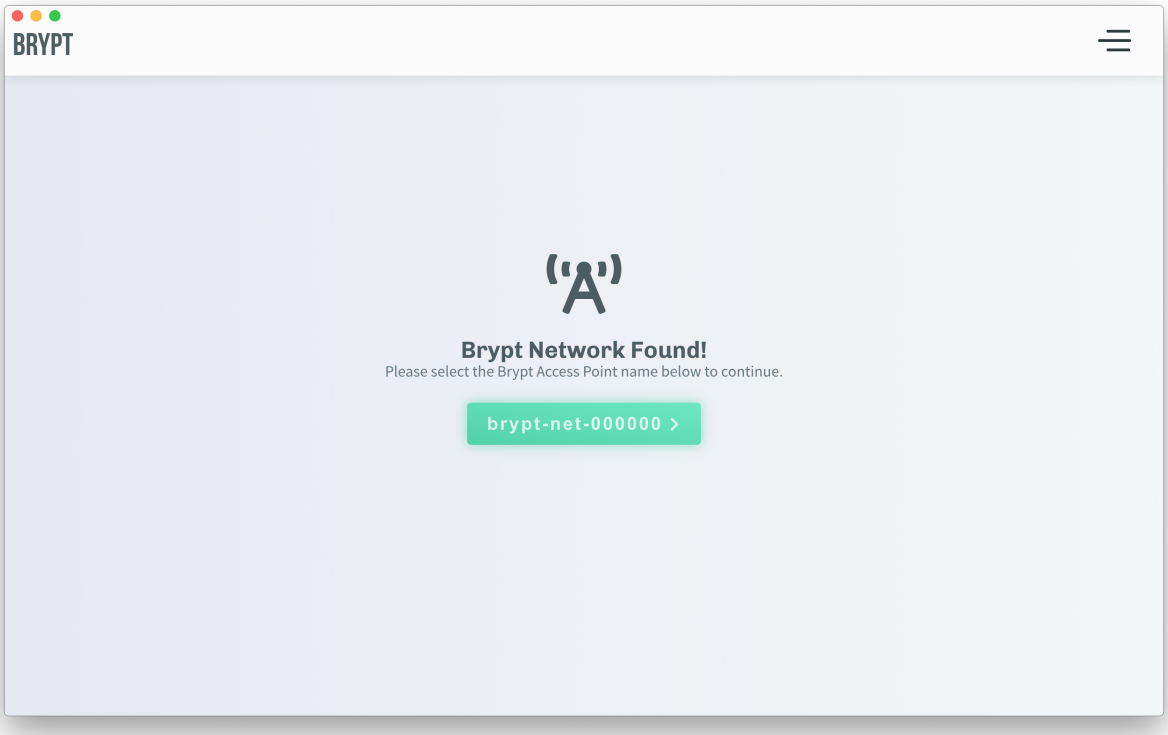

Figure 6: Brypt network access point found

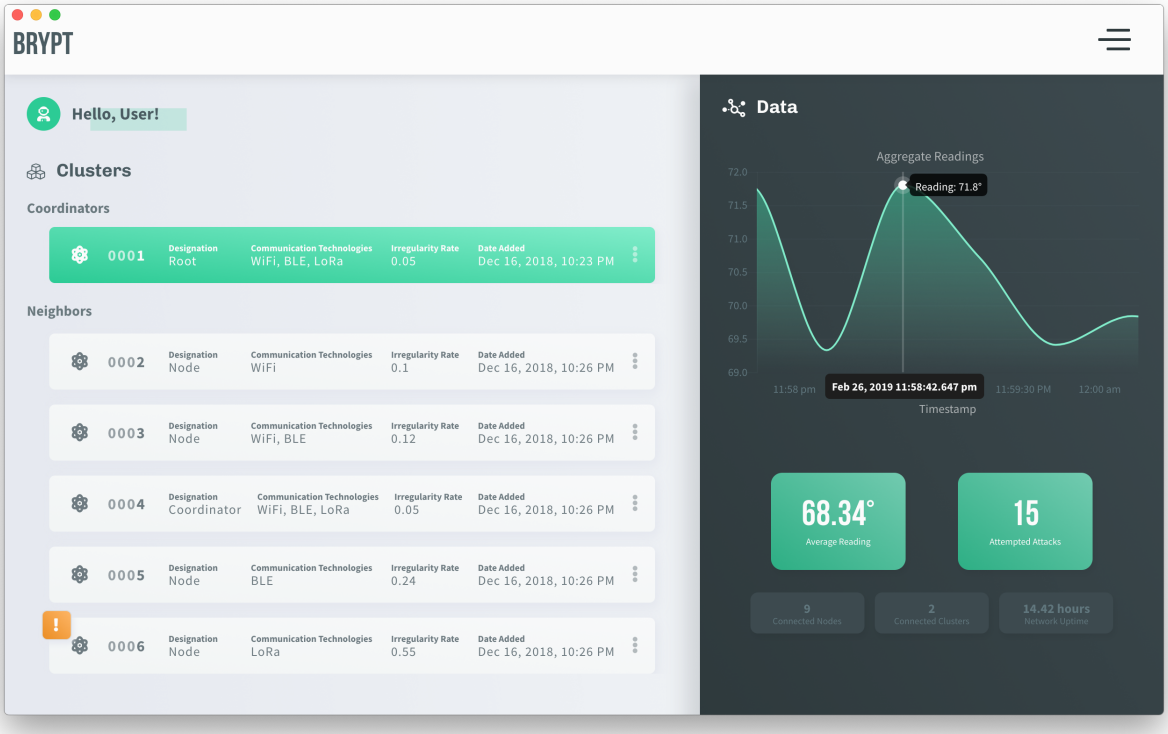

Figure 7: Brypt Desktop application using Electron framework

Figure 8: Clusters capable of being expanded

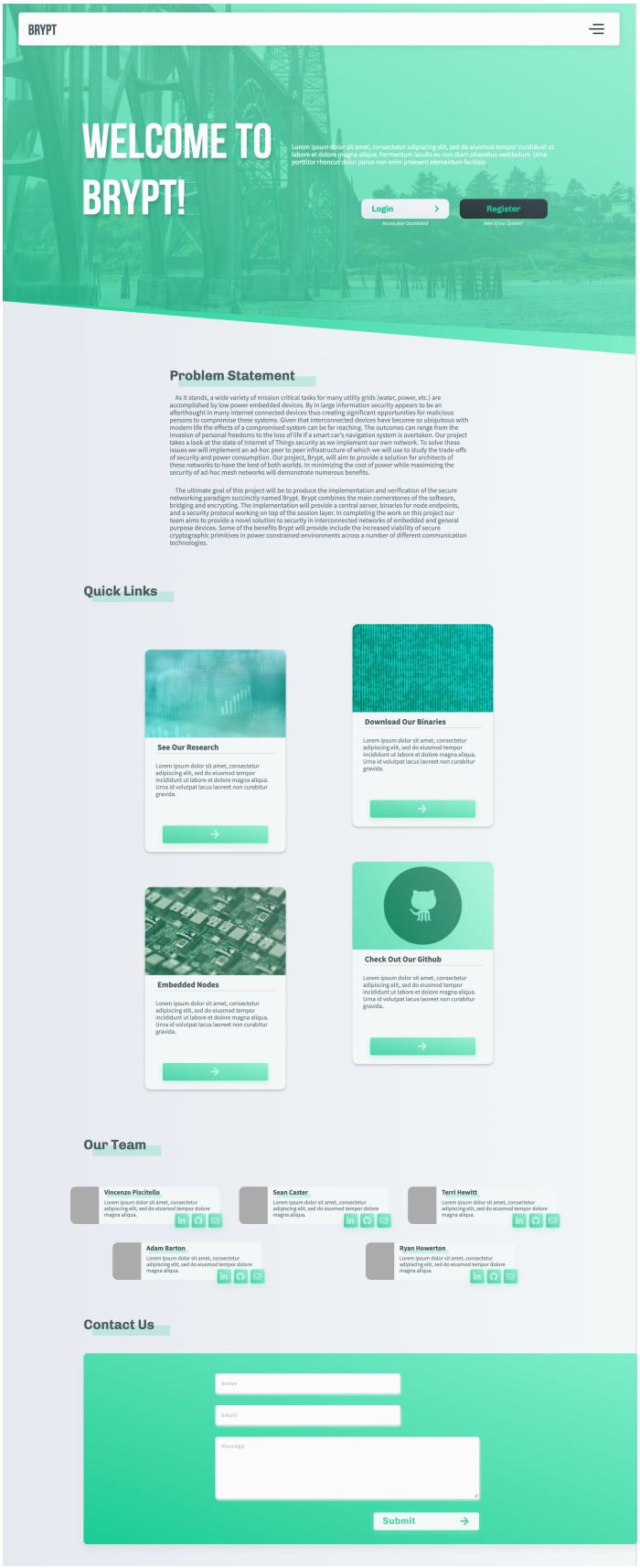

Figure 9: Online welcome page at www.brypt.com

6 Interesting Code Snippets

6.1 Desktop application supports Windows and OS X

```
'conditions': [
    [ 'OS=="win"', {
        'conditions': [
            ['target_arch=="x64"', {
                'variables': {
                'openssl_root%': 'C:/OpenSSL-Win64/openssl-1.1.0f-vs2017'
            },
            }, {
                'variables': {
                    'openssl_root%': 'C:/OpenSSL-Win32'
                },
            }],
        ],
        'defines': [
            'uint=unsigned int',
        ],
        'libraries': [
            '-l<(openssl_root)/lib64/libcryptoMT.lib',
            '-l<(openssl_root)/lib64/libsslMT.lib',
        ],
        'include_dirs': [
            "<!(node -e \"require('nan')\")",
            "<(openssl_root)/include64/openssl",
            "<(openssl_root)/include64",
        ],
    }, { # OS!="win"
        'libraries': [
            "-L/usr/local/opt/openssl/lib",
            "-L/usr/local/opt/openssl/include/openssl",
            "/usr/local/opt/openssl/lib/libcrypto.a",
            "-lcrypto",
            # "-lssl"
        ],
        'include_dirs': [
            "<!(node -e \"require('nan')\")",
            "/usr/local/opt/openssl/include/openssl",
            "/usr/local/Cellar/openssl/1.0.2q/include/openssl",
        ],
    }]
```

./source/binding.gyp

6.2 AwaitObject and AwaitContainer to delay responses to requests that require input from other nodes.

```
#ifndef AWAIT_HPP
#define AWAIT_HPP

#include <cstdio>
#include <cstring>
#include <string>
#include <unordered_map>
#include <openssl/md5.h>

#include "utility.hpp"
#include "json11.hpp"

typedef std::unordered_map<std::string, class AwaitObject> AwaitMap;
// const std::chrono::milliseconds AWAIT_TIMEOUT = std::chrono::milliseconds(500);
const std::chrono::milliseconds AWAIT_TIMEOUT = std::chrono::milliseconds(1500);

class AwaitObject {
    private:
        bool fulfilled;
        unsigned int expected_responses;
        unsigned int received_responses;
        class Message request;
        json11::Json::object aggregate_object;
        SystemClock expire;
```

```
    public:
        AwaitObject(class Message request, unsigned int expected_responses) {
            this->fulfilled = false;
            this->received_responses = 0;
            this->expected_responses = expected_responses;
            this->request = request;
            this->expire = get_system_clock() + AWAIT_TIMEOUT;
        }

        bool ready() {

            if (this->expected_responses == this->received_responses) {
                this->fulfilled = true;
            } else {
                if (this->expire < get_system_clock()) {
                    this->fulfilled = true;
                }
            }

            return this->fulfilled;
        }

        class Message get_response() {
            std::string data = "";

            if (this->fulfilled) {
                json11::Json aggregate_json = json11::Json::object(this->aggregate_object);
                data = aggregate_json.dump();
            }

            class Message response(
                this->request.get_destination_id(),
                this->request.get_source_id(),
                this->request.get_command(),
                this->request.get_phase() + 1,
                data,
                this->request.get_nonce() + 1
            );

            return response;
        }

        bool update_response(class Message response) {
            this->aggregate_object[response.get_source_id()] = response.get_pack();

            this->received_responses++;
            if (this->received_responses == this->expected_responses) {
                this->fulfilled = true;
            }

            return this->fulfilled;
        }


};

class AwaitContainer {
    private:
        AwaitMap awaiting;

        std::string key_generator(std::string pack) {
            unsigned char digest[MD5_DIGEST_LENGTH];

            MD5_CTX ctx;
            MD5_Init(&ctx);
            MD5_Update(&ctx, pack.c_str(), strlen(pack.c_str()));
            MD5_Final(digest, &ctx);

            char hash_cstr[33];
            for (int idx = 0; idx < MD5_DIGEST_LENGTH; idx++) {
                sprintf(&hash_cstr[idx * 2], "%02x", (unsigned int)digest[idx]);
            }

            return std::string(hash_cstr);
        }

    public:
```

```
        std::string push_request(class Message message, unsigned int expected_responses) {
            std::string key = "";

            key = this->key_generator(message.get_pack());
            std::cout << "== [Await] Pushing AwaitObject with key: " << key << '\n';
            this->awaiting.emplace(key, AwaitObject(message, expected_responses));

            return key;
        }

        bool push_response(std::string key, class Message message) {
            bool success = true;

            std::cout << "== [Await] Pushing response to AwaitObject " << key << '\n';
            bool fulfilled = this->awaiting.at(key).update_response(message);
            if (fulfilled) {
                std::cout << "== [Await] AwaitObject has been fulfilled, Waiting to transmit"
    << '\n';
            }

            return success;
        }

        bool push_response(class Message message) {
            bool success = true;

            std::string key = message.get_await_id();
            std::cout << "== [Await] Pushing response to AwaitObject " << key << '\n';
            bool fulfilled = this->awaiting.at(key).update_response(message);
            if (fulfilled) {
                std::cout << "== [Await] AwaitObject has been fulfilled, Waiting to transmit"
    << '\n';
            }

            return success;
        }

        std::vector<class Message> get_fulfilled() {
            std::vector<class Message> fulfilled;
            fulfilled.reserve(this->awaiting.size());

            for ( auto it = this->awaiting.begin(); it != this->awaiting.end(); ) {
                std::cout << "== [Await] Checking AwaitObject " << it->first << '\n';
                if (it->second.ready()) {
                    class Message response = it->second.get_response();
                    std::cout << response.get_data() << '\n';
                    fulfilled.push_back(response);
                    it = this->awaiting.erase(it);
                } else {
                    it++;
                }
            }

            return fulfilled;
        }

        bool empty() {
            return this->awaiting.empty();
        }

};

#endif
```

./source/await.hpp

7 Relevant Images

7.1 Brypt Architecture Diagram