Abstract—This is an innovative practice full paper. In past projects, we have successfully used a private TOR (anonymity network) platform that enabled our students to explore the end-to-end inner workings of the TOR anonymity network through a number of controlled hands-on lab assignments. These have satisfied the needs of curriculum focusing on networking functions and algorithms. To be able to extend the use and application of the private TOR platform into cryptography courses, there is a desperate need to enhance the platform to allow the development of hands-on lab assignments on the cryptographic algorithms and methods utilized in the creation of TOR secure connections and end-to-end circuits for anonymity.

In tackling this challenge, and since TOR is open source software, we identify the cryptographic functions called by the TOR algorithms in the process of establishing TLS connections and creating end-to-end TOR circuits as well tearing them down. We instrumented these functions with the appropriate code to log the cryptographic keys dynamically created at all nodes involved in the creation of the end to end circuit between the Client and the exit relay (connected to the target server).

We implemented a set of pedagogical lab assignments on a private TOR platform and present them in this paper. Using these assignments, students are able to investigate and validate the cryptographic procedures applied in the establishment of the initial TLS connection, the creation of the first leg of a TOR circuit, as well as extending the circuit through additional relays (at least two relays). More advanced assignments are created to challenge the students to unwrap the traffic sent from the Client to the exit relay at all onion skin layers and compare it with the actual traffic delivered to the target server.

Index Terms—project-based learning; problem-based learning; onion routing; TOR; undergraduate security course, anonymity, cryptography, Wireshark dissector.

I. INTRODUCTION

Over the past 15 years, we at this school have been progressively developing new undergraduate curricula for Computer Networking, Network and Information Security education spread across a diverse set of disciplines including, but not limited to, Computer Science, Telecommunication Sciences, Intelligence Analysis, and Information Technology. More specifically, our students have the opportunity to learn cryptography, design of computer networks, analysis of network traffic, and cyber-defense. In an effort to strengthen our students’ deep learning and mastery of the application of cryptography protocols and algorithms, we have embraced project-based and problem-based learning approaches [1]. According to [2], PBL is defined as a student-driven, teacher-facilitated approach in teaching and learning processes. The problem-based approach has been adopted for short and focused assignments, while the project-based learning approach is adopted for semester-long projects with cross-subject assignments and, in some cases, capstone projects. These assignments are to be tackled by a team of students where the instructor plays the role of a coach to guide and inspire rather than to instruct and lecture. While designing the assignments, specifically under the project-based learning (PBL) approach, we selected topics with integrative STEM education focus. The assignments integrate the concepts and practices of science and/or mathematics education with the concepts and practices of technology and engineering education [3]. For example, in [4], the author designed a set of programming projects and assignments aimed at familiarizing the students with the OpenSSL cryptography library. The projects were limited in scope to one or two cryptography concepts at-a-time. We extended on that approach by developing assignments that engage the students into a more integrated implementation of cryptographic mechanisms based on number theory algorithms, as well as exposing the students to production-level security source code. Such exposure was suggested to us in a number of conversations we had with current and potential employers in the government and the private cyber-security industry.

By selecting TOR network as the main focus of this paper, we have ensured that the students are engaged in real world opportunities on how anonymity is accomplished through unique application of multi-layer encryption and decryption cryptographic algorithms. The major challenge in designing these PBL assignments is the availability of a platform that would enable the students to conduct these explorations without the education limitations of and the possible interference with the public TOR network. Therefore, we decided to explore private TOR network platforms through which we have an end-to-end control over all the components of the TOR network and traffic. We researched a number of private TOR network platform implementations. In the experimentation described in [5]–[7], we found a lot of valuable high-level information about the overall workings of the TOR network. We also found in [8] a set of hands on assignments aimed at having the students to become acquainted with the inner workings of TOR connection establishment and circuits creation. However, these lack the desired exploration of cryptographic algorithms of achieving anonymity in the TOR network. In [9] and [10], we identified private network platforms with some enhanced
tools focusing on the cryptographic algorithms including the capability to decrypt and dissect TOR cells. Unfortunately, these were developed using older and outdated releases of TOR (tor-0.2.6.7 and older).

We adopted the private TOR network platform described in [8], and briefly introduced in section II. We enhanced the platform with cryptographic tools based on those described in [10] for a newer TOR network release, namely tor-0.3.3.0-alpha-dev. We focus on applying these enhancements in the design and development of several new PBL assignments. We believe that our enhancements to the TOR private network platform would enable other instructors to develop their own assignments for engaging their students into learning how content confidentiality is established and exploring new cryptographic algorithms projects.

We start in section II by stating the motivations behind the development of our pedagogical assignments. We also lay out some of the considerations influencing the design of the assignments. We also identify the prerequisites on behalf of the students in order to successfully learn from these assignments. Next, we discuss the implementation of the assignments in section III. We conclude the paper in section IV, and suggest some future extensions to this work.

II. MOTIVATION AND DESIGN CONSIDERATIONS

Over the past three years, we incorporated a comprehensive semester-project for our undergraduate Information Security course in which students implemented an advanced key-exchange protocol that employed a trusted key distribution center (KDC) and supported user authentication via digital signature. Table 1 shows the level of interest in this course among the students (the enrollment level at the end of the semester) and their performance in the semester-project.

<table>
<thead>
<tr>
<th>Semester</th>
<th>Class Size</th>
<th>Avg Project Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fall 2017</td>
<td>46 (2 sections)</td>
<td>96%</td>
</tr>
<tr>
<td>Fall 2018</td>
<td>28 (1 section)</td>
<td>86%</td>
</tr>
<tr>
<td>Fall 2019</td>
<td>48 (2 sections)</td>
<td>91%</td>
</tr>
</tbody>
</table>

TABLE 1: Students Enrollment and Grade

Early in the Spring semesters following the completion of the course, many students reported receiving relatively high-paying job offers from major corporations in the security field. In later recruiting events held at our university, some of these same students represented their employers recruiting for more fresh graduates. Many emphasized the strong impression their experience in security software development had on their employers. We also received suggestions from the hiring managers that we should consider exposing our students to production-level open-source security software in order to further prepare them for the cyber-security field.

In their 2019 report [11], the Enterprise Strategy Group with the Information Systems Security Association confirmed what we had heard from the hiring managers. The report also concluded that 23% of its member organizations did not believe the cyber-security team had been given the right level of training. Moreover, 71% of the member organizations indicated that the most effective methods for Knowledge, Skills, and Abilities (KSA) development is through attending face-to-face specific cyber-security training courses. In addition, the report concluded that the cyber-security skills shortage was not improving, and that 74% of the organizations had been either significantly impacted, or somewhat impacted by the global cyber-security skills shortage. Of these skills, development of security applications, and security analysis & investigation were among the top three skills in dire shortage.

The National Institute of Standards and Technology in their NICE publication [12] identified a comprehensive list of specialty areas to be addressed by any successful framework for cyber-security education. The following is part of a long list of specialty areas that are critical:

- the development and modification of security software,
- collecting and analyzing network information to identify and report security incidents,
- maintaining and reviewing the infrastructure security software, and
- gathering of evidence to mitigate possible or real-time threats.

We designed the new assignments in this paper to address some of these required skills that are in shortage. We believe that these new assignments will continue to exhibit the same level of enrollment and success among future students as shown in table 1. The following is a brief summary of the concepts covered and the skills to be gained by the student for each of the four new assignments included in this paper:

- In Assignment 1, the students learn how to analyze, and instrument an existing production-level security software application. Specifically, the students understand the role and the use of the Transport Layer Security (TLS) protocol widely used for a secure authenticated communication on the Internet today. They also become familiar with the TOR protocol which supports confidentiality, authentication, and anonymity to users of the Internet.
- In Assignment 2, the students learn the internal details of how the TLS protocol establishes secure communication channels, and the role of the master key in the context of Diffie-Hellman key exchange mechanism. The students learn how to use and modify tools intended to collect and analyze network traffic and information.
- In Assignment 3, students gain the skill of instrumenting security software. They modify the TOR source code to gather the necessary keys required to further analyze the TOR traffic on the network. This assignment emphasizes the skill of traffic analysis which is essential for any effort to detect and respond to security incidents.
- In Assignment 4, the students learn how to develop encryption and decryption software and maintain the necessary security OpenSSL contexts.

For the students to take part of the hands-on assignments presented in this paper, it is assumed that they have full access to the TOR Private Network depicted in Figure 1, and described in [8] and [5]. The private network is set up with a total of nine
virtual machines (VMs): two as TOR Directory Authorities (DAs), six as TOR Onion Routers or relays (ORs) and one as a TOR Client. The web server, although can run on a non-TOR machine, is installed and run on one of the DA machines. This is to minimize the size of the network, the storage requirement, and the performance impact on the host computer. We have enhanced the original platform to include the following:

- the ability to decrypt the TOR cells using a modified version of the patch created in [10], and
- the capability to display the decrypted TOR cells using a modified Wireshark TOR dissector written in lua [13] and described in [10]. The dissector is designed to perform the labeling of the TOR cells as well as the decryption of the TOR cells using a lua library known as lockbox [14].

![Network Architecture diagram for assignments](image)

Our TOR platform was originally built using tor-0.3.3.0-alpha-dev. Therefore, we designed and implemented our hands-on assignments based on that version. Recent versions of TOR, namely versions tor-0.4.x.x, have made significant changes to the structure of the code. We do not address those changes in this paper. The platform has been architected to enable instructors and/or students to design and/or conduct basic and advanced assignments depending on the requirements of their projects. Assignments 1 and 2 are examples of basic assignments, whereas Assignments 3 and 4 are relatively more advanced. All assignments discussed in this paper are designed and implemented to run on the TOR Client computer where the Onion Proxy resides. No traffic analysis or code changes are done at the Entry, Intermediate or Exit Onion Routers.

Since TOR code is written using the C programming language, it is expected that the students have a strong command of this language. More specifically, they must be fluent in topics such as pointers, dynamic memory, structures, macros, etc. We designed the assignments so that they build on top of each other, and thus provide continuity of the subject matter.

We assume the students are already familiar with cryptography concepts such as symmetric encryption, block modes of operation, secure hashing, and Diffie–Hellman (DH) key exchange schemes. The students should also be familiar with some cryptographic library, preferably the OpenSSL's crypto library. We require the students to work in teams of 2-to-3 each to emphasize the importance of peer-driven learning.

This will allow for a diverse group of institutions to use the lab assignments rather than limiting its application to a specific type of program. It is our intention to offer one or more of these assignments either as part of a module on anonymity or as semester projects. The module will include an in-depth introduction to the inner workings of TOR which will span at least two lecture sessions with a special emphasis on the cryptography topics on which the students will be trained.

### III. IMPLEMENTATION OF TOR CRYPTOGRAPHY ASSIGNMENTS

In this section, we describe the implementation of a set of cryptography-related hands-on assignments that would help undergraduate students understand how TOR achieves anonymity while browsing the Internet. However, we need to highlight few configurations that the students need to perform before attempting any of the assignments. These are to:

- disable the gzip compression on a browser to be used on the TOR Client say the Firefox browser [15],
- configure Wireshark so that a non-root user is able to capture packets on the target interface, and to own the Wireshark preferences file,
- place TOR-Cell-Dissector.lua [10] in the plugins folder under Wireshark,
- place the lockbox [14] in the lua/5.2 path, which can be created by the students directly or through the installation of lua [13], and
- re-launch Wireshark so that the lua dissector is loaded. TOR should be then listed under Wireshark protocols list.

#### A. Assignment 1: Neutralizing the Effect of TLS

In this assignment, we focus on having the students navigate through the TOR source code in order to include the ability to create a custom log file, and dump the pre-TLS traffic in that file. The students are asked to compare this traffic with the decrypted SSL output of Wireshark's built SSL dissector described in Assignment 2.

First, the students learn the steps needed to establish their own custom log file to monitor specific TOR events they deem of interest. This is accomplished via the following steps:

1) Define a new log severity level "LOG_STUDENT" in the file "src/common/torlog.h". Next, move the LOG_ERR severity to level 2, and create LOG_STUDENT at severity level 3. The reason for high custom severity level is to minimize the amount of information to be logged to the special log file set up for the purpose of the lab assignments. By default, that file will also include messages logged at the higher severity level LOG_ERR [16]. (Notice: the lower the number, the higher the severity level

2) Update the functions handling the severity levels to incorporate the new LOG_STUDENT severity. The students accomplish this by editing ""src/common/log.c"",...
and updating these three functions: sev_to_string(), should_log_function_name(), and parse_log_level().

3) Add a new log domain LD_STUDENT.TOR taking advantage of the concept of "log domain" to refer to an area of functionality inside TOR source code [17]. The students will use this log domain to refer to the new code we add to TOR. This involves modifying both "src/common/torlog.h" and "src/common/log.c".

4) Configure TOR to send log messages of the new LOG_STUDENT severity level to our own dedicated file by editing /usr/local/etc/tor/torrc. Events of interest in TOR’s source code can now be logged by calling: log_student(LD_STUDENT, ... ).

Next, we instruct the students to identify the TOR’s code responsible for sending and receiving data through a TLS connection. To aid the students, we provide them with a demo code illustrating the use of OpenSSL’s SSL_read() and SSL_write() library functions. The students are asked to locate the use of these functions in TOR’s code (hint: tortls.c) and add the necessary code to dump the data being exchanged through the TLS connection. Since such data may contain multiple TOR cells, a STUDENT_cell_parse() is repeatedly called to parse and dump one TOR cell at-a-time. When designing this parser, the students must take into consideration that not all cells have fixed-length, in which case a 2-byte "length" field precedes the payload data in variable-length cells. Students are also alerted that the cell layout has changed for link protocol versions 3 or higher. More specifically, the circuit ID field has been expanded to four bytes, up from two. Students are referred to Figure 2 for specific cell layout details. TOR provides a cell_command_to_string() function to decode the one-byte cell command which proves handy at this point. The cell command helps determine whether this is a fixed-length (i.e. 509 bytes) or a variable-length cell. The remainder of the cell content is then dumped to the log file. At the end, the cell-parsing function must keep track of any CELL_VERSIONS commands that may affect the width of future cell IDs. An excerpt of a sample log file is shown in Figure 3.

Fig. 2: TOR Cell Layout

As a sanity check, students are instructed to hold onto the output of this assignment so that they may use it to compare with the output of Wireshark built-in TLS decryption described in Assignment 2.

B. Assignment 2 : TLS payload, Master Key and Wireshark SSL Dissector

In this assignment, we focus on the modified patch to be applied to the tor-0.3.3.0_alpha_dev source code to dump the TLS master key for the TLS connection between the TOR Client and the entry OR. The students will then be able to use Wireshark built-in SSL dissector with the dumped TLS master key to decrypt the TLS connection payload. This patch is a modified version of the tor_0.2.7.6_Patch.patch published in [10]. The students learn how to apply the patch on the TOR source code running on the TOR Client. The patch is designed to dump the TLS master key for the TLS connection negotiated and established between the Client and the entry Onion Router (OR) for a given circuit or group of circuits. Also, it is capable for dumping the symmetric keys used for TOR cells encryption and decryption in the forward and backward directions. In this assignment, the students learn how to use the TLS master key and Wireshark built-in SSL decryption dissector to decrypt and display the decrypted TLS payload. Note that the decrypted TLS payload is an encrypted TOR cell or cells by the circuit symmetric keys. The students will be able to read the circuit ID and command identification allowing the TOR dissector to label the SSL application data packets as shown in Figure 6. However, it would not be possible to decrypt the TOR cell(s) by applying the TOR dissector without the circuit keys. In Assignment 1, we referred to the SSL decrypted payload as pre-TLS traffic. Note that the students will learn how to dump the circuit keys and dissect the TOR cells later in Assignments 3 & 4.

In this assignment, the students perform and execute the following steps:

1) Apply tor_0.3.3.3_alpha_dev_Patch.patch on the TOR source code of the TOR Client. Next, they compile, debug and resolve errors as appropriate and finally install TOR.

2) Launch Wireshark and start packet capturing on the TOR Client machine. Launch TOR on the same. Check that one or more circuits have been created using a custom python script (see an example in Figure 4). Note that each TOR
OR in a circuit is defined by its server fingerprint (20-byte hash tied to the secret id_key), nickname (such as T6) and its IP address (such as 192.168.1.6). Wireshark should display TLS/SSL packets. However, on some releases of Wireshark, the students may have to decode the transport port number 5000 as SSL so that Wireshark displays the captured packets with the correct TLS protocol designation.

3) Launch the browser (configured to make use of a TOR proxy) and access the default page of an apache2 web server running on the target server (using the HTTP protocol). Since Wireshark is set up to capture packets on the TOR private network through the deployment of the Host-only virtual network as shown in Figure 1, TLS/SSL wrapped and encrypted HTTP traffic at the TOR Client and the corresponding plaintext/decrypted HTTP traffic at the web server should be displayed on Wireshark. Close the browser on the TOR Client, stop the TOR process and stop packet capturing. Export the packets captured for analysis.

4) Find the dumped TLS master key in the info.log residing in the TOR log folder. For Wireshark to decrypt the TLS payload, it needs a master key file that contains the following information: CLIENT RANDOM "TLS Client random hexadecimal string". "The associated/related TLS master key in hexadecimal." The Client random hexadecimal string can be found in the Client Hello packet of the TLS connection establishment messages’ exchange exported above.

5) Prepare for the packet capture analysis in the next step. The students are asked at this point to reflect on questions such as: does the TOR process create multiple circuit under the same TLS/SSL connection between the Client and the entry OR? How would they filter all packets associated with the same TLS connection? When the browser is launched, do the associated packets have different source transport port number from the one used to establish the TLS/SSL connection? What type of TLS packets are used to carry the TOR cells?

6) Launch Wireshark, open the exported pcapng file. Select Wireshark preferences, select protocols and navigate to the SSL protocol item. The students should point to the master key file created in the previous step. Apply the following display filter:

\[
\text{ip.addr}==<\text{ip address of TOR client}> \\
\&\& \text{ip.addr}==<\text{ip address of Entry OR}> \\
\&\& \text{tcp.port}==<\text{TOR Client tcp port}> \\
\&\& \text{ssl.record.record_type}==23
\]

(23 designates an TLS/SSL Application Data). All the packets exchanged between the client and entry OR after the TLS connection establishment using the pair of random and master key strings should be displayed on Wireshark as being SSL decrypted. Note at the bottom of Wireshark windows, there should be a tab labeled "Decrypted SSL". This is next to the Frame tab as shown in Figure 5. From the SSL Decrypted strings, the students should be able to identify the circuit ID (the first 4 bytes) of the cell(s) carried within the TLS connection.

7) Continue analyzing the packet captured. Apply the TorDissector.lua provided in [10] without decryption of the TOR cells (addressed in Assignments 3 & 4). Once the lua dissector is loaded, the packets displayed in Figure 5 will be identified in terms of the TOR protocol, the associated TOR command, circuit ID and the cells are numbered as shown in Figure 6.
The students are asked to identify the cells with circuit ID 00000000 as shown in Figure 7 and the role of these cells. They are asked to explain how the TOR Dissector is able to figure out the command type using the tor specification documents. Next, they are asked to identify the number of unique circuit IDs beyond 00000000 that have been created under the TLS connection.

![Fig. 7: Cells with Circuit ID 00000000](image)

8) Compare the pre-TLS output of Assignment 1 as shown in Figures 3 and 8 with the output generated in this assignment as shown in Figures 5 and 6.

![Fig. 8: Pre-TLS Dump of Frame 179](image)

C. Assignment 3: Circuit Keys dump, TOR Cell Wireshark Dissector, Circuit Negotiation

It is assumed that Assignment 2 has been successfully completed by the students and they have:

- a copy of the exported pcapng file analyzed in Assignment 2 up to the point of labeling the pre-TLS (decrypted TLS/SSL) cell carrying packets, and
- a copy of the info.log.

Recall that Assignment 2 concludes without the decryption of the TOR cells exchanged by the TOR, the entry OR and beyond including the web server. This assignment focuses on creating the circuit_keys.log file to include the circuit symmetric keys dumped by the additional modifications to the TOR source code patch described in Assignment 2. This assignment also covers the use of Wireshark TOR dissector to decrypt, decode and display the TOR cells’ payload. They will compare the results of the HTTP exchange on the TOR Client which is decrypted by Wireshark TOR cell decryption and dissector with that captured at the exit OR towards the web server. The packets are captured and included in the pcapng exported in Assignment 2. Included in this assignment are the steps for negotiating and establishing TOR circuits.

In this assignment, the students perform and execute the following steps:

1) Find the dumped circuits symmetric keys (3 pairs per circuit) in the info.log residing in the TOR log folder. Place into a file named TorKeys.log. TOR on the Client may create multiple circuits under the same TLS connection. Note that there may be multiple sets of circuit keys as shown in Figure 9. Each statement includes the circuit ID, the direction (forward or backward/reverse), the digest and the key hexadecimal strings.

![Fig. 9: Example of Dumped Circuit Keys](image)

2) Navigate to the new TOR protocol page on Wireshark preferences and set up the path to the TorKeys.log file and select the 3 hops and the decryption checkbox options. Restart Wireshark and open the exported pcapng file (from Assignment 2). If the TLS master key and circuit keys log files contain the keys required for decrypting the TLS payload, decrypting the TOR cells payloads and dissecting the cells, then Wireshark will display the TLS packets associated with our TOR circuits and provide the decrypted and decoded TOR cells payload as shown in Figure 10.

![Fig. 10: Decrypted Cells - HTTP packets Carrying TOR Cells](image)

3) Modify the filter so that to add the HTTP packets as captured at the exit OR as shown in Figure 11.

4) Compare the HTTP GET request in Figures 10 and 11. They should match.

5) Continue your analysis to include the HTTP response under the HTTP 200 OK. The comparison would fail if we did not take the step of disabling the gzip compression earlier. This is because the TOR Dissector at this point assumes a pair consists of a forward key is used to encrypt/decrypt traffic travelling away from the Client, and the backward key is used for traffic travelling back towards the server.
point does not have the capability to resolve the gzip compression.

The students would then be asked to conduct a deeper analysis on the captured packets to identify the flows associated with the TLS connection and circuit establishment. They are to compare the flow with that expected based on the TOR specification documents [18] and [19]. Figure 12 shows the flow for the TLS connection between the Client and the Entry OR and the packets exchanged in the creation of the first circuit. See sequence diagrams in [19]. Figure 13 shows the flow for HTTP 200OK response from the web server (T4) to the exit OR (T7) to the middle/intermediate OR (T4) to the entry OR (T6) to the Client (T12). Note that the HTTP response from T4 to T7 requires 24 TOR cells to carry. Also note that T4 played 3 roles: DA, Web Server and intermediate OR by chance not by design.

D. Assignment 4: TOR Cell Parsing & Decryption

In this assignment, the students develop their own program to decode the TOR cells, and decrypt the payload of relay cells using the circuit keys dumped that are discussed earlier in in Assignment 3. We use OpenSSL [20] crypto library via the Envelope (EVP_) API. We are only interested in analyzing TOR traffic that are headed to or coming from the Entry onion router (OR). This assignment mainly consists of the following milestones:

1) Understanding the use of multiple encryption layers (a.k.a. skins) that are applied to Relay-type cells. The number of layers is determined by which relay along the established circuit is in communication with the TOR Client. Figure 15 provides the students with an illustration of how the three onion skins may exist in a 3-hop circuit. Students learn that the all TOR cell skins (encryption) are created when generating outgoing (a.k.a. forward) traffic. In contrast, the unwrapping of the encryption skins from incoming (a.k.a. backward) traffic. As relay cells travel forward, each intermediate relay unwraps only one skin to decide whether it is the target of that cell, or that it should send it forward to the next relay along the circuit. On the other hand, a relay will always add its own skin on top of backward traffic then sends it backward to the next relay router in the established circuit.

2) Maintaining the encryption contexts. Each skin uses two symmetric keys: one for each direction of traffic. Therefore, the students should maintain two arrays of cipher contexts as in:
Each context must be initialized with an all-zero 128-bit initial vector, and with the default engine implementation of the EVP_aes_128_ctr() algorithm. The context must use the appropriate symmetric key that were dumped in Assignment 3. For example, F_ctx[0] is initialized with the forward key associated with the outermost onion skin correlating the to the entry OR, while B_ctx[2] uses the backward key of the innermost onion skin correlating the Client to the exit OR.

3) Implementing the skin-unwrapping algorithm shown in Figure 16. The students shall apply this algorithm on the relay payload previously depicted in Figure 2. Initially, we simplify the decision to identify "recognized cell payloads" to merely testing that the Recognized field of the Relay header is equal to zero. Even though this is a necessary but not sufficient condition, in most cases this condition may be treated as sufficient [21]. We make such simplification to avoid overwhelming the students with more complicated details on actually maintaining the running digest of the relay cells. When a Relay-type cell is not recognized at a given onion skin level, we need to keep decrypting to peel off the next onion skin. It is worth noting that the number of skins to decrypt depends on which OR is in contact with the Client. For example, when the Client is communicating with the intermediate OR (i.e. the second step in the circuit), only the outer two skins are present. At this point, the students realize that since TOR uses AES in the Counter mode (CM), EVP_EncryptInit_ex() and EVP_EncryptUpdate() may also be called for decryption purposes. It’s worth noting that the AES in CM make use of an arbitrary number (counter); the counter is managed, in our case, by OpenSSL context.

4) Validating the unwrapping outcome. The students execute their own code on the same traffic previously analyzed in Assignment 3 and verify that the decrypted Relay cells’ payloads match. A sample run is shown in Figure 17. As part of this assignment, we scaffold the students to realize the need to decrypt all relay-type cells of a given direction in the same sequential order they are created. This is due to the fact that AES is applied in Counter mode of operation.

Fig. 16: Decrypting the Relay Cell’s Payload

We plan to extend this assignment in the future to have the students actually calculate the digest of the Relay cell payload, and incorporate the Integrity Relay header field to identify recognized Relay cells.

IV. CONCLUSIONS AND FUTURE WORK

We believe that the enhancements to the private TOR network platform would be valuable to those interested in learning and/or teaching the inner workings of the TOR algorithms. Preliminary assessment of the assignments with the help of a qualified colleague paves the road to offer them to undergraduate students. We plan to measure the impact of these assignments on the students’ knowledge base. Below is a brief list of anonymity-related topics that we see ourselves pursuing in the near future:

- Continue to assess and enhance the effectiveness of these assignments in the context of PBL,
- Continue to monitor the cyber-security community and industry requirements and needs in order to keep our approach and focus up-to-date,
- Implement full integrity validation in the skin-unwrapping algorithm of Assignment 4,
- Expand upon the network topology to include more Relays, and DAs to more accurately emulate the real-world environment, and
- Continue to assess the effectiveness of the platform as a tool in teaching anonymity algorithms including the creation and validation the authentication keys, digests and fingerprints.

V. ACKNOWLEDGMENT

The authors would like to thank Mgr. Michal Reinöhl for his work on patching the source code for tor-0.2.7.6 and providing the TOR dissector in LUA for Wireshark. Thanks to Dr. Steven Murdoch and Mr. Francesco Fatti for sharing with us copies of their early work. Also, a special thanks to Mr. Grant Hobar for making available to us the initial private TOR network virtual machines.
REFERENCES