Research on Techniques and Methods of Developing Cryptography Virtual Laboratory

Guihua Duan*, Yan Wang, Min Li, Yu Sheng, Jianxin Wang, Shigeng Zhang

School of Information Science and Engineering, Central South University, Changsha, Hunan, China

Abstract

To address the deficiencies of cryptography education, we design and implement a virtual experiment system named VESC (Virtual Experiment System for Cryptography), which is composed of a front end developed with HTML5 and a back end developed with Node.js and Docker. Based on the parameters input by the users, VESC uses the cross-browser vector graphics library Raphael to dynamically demonstrate the workflow of various cryptography algorithms to users in a step-by-step manner. VESC is easy to operate while having a friendly interface. After users submit their codes, VESC compiles the codes, executes them, and sends the results back to the users. Our experimental results show that VESC achieves high performance in a highly concurrent accessing environment. VESC is not only a virtual experimental platform for students, but also offers an assistant system for teachers to help students better understand the principles of complicated cryptographic algorithms and protocols as well as their applications.

Keywords: cryptography; virtual experiment; teaching assistant system; HTML5; node.js; docker

(Submitted on October 1, 2017; Revised on November 2, 2017; Accepted on November 23, 2017)

© 2017 Totem Publisher, Inc. All rights reserved.

1. Introduction

Cryptography is the core fundamental course of information security specialty, including classical theories and algorithms in cryptography, cryptographic protocols, key management and application, number theory, etc. In cryptography education, a teacher needs a powerful and easy-to-use experimental platform to help students understand complicated cryptographic algorithms. However, most existing experiment platforms lack the capability of step-by-step data processing and analysis in cryptography applications. Moreover, a few cryptographic algorithms deployed in these systems are publicly available. Hereby, we design and develop a VESC with the Browser/Server (B/S) structure. The computation of cryptographic algorithms is carried out in the browser, which reduces the workload of the server, and can effectively avoid the server security problems.

More specifically, the system has the following features:

- VESC includes a front end and a back end. The front end is used to demonstrate teaching and verification experiments, and the back end is mainly used to manage the system and to support the design experiments;
- It visualizes the abstract operation process, dynamically showing the overall process related to cryptographic algorithms;
- The users can repeatedly enter the input parameters. Some of the system parameters can be randomly generated within a predefined parameter range so that users can just click on the buttons to move into the next step;
- The algorithms are divided into a set of modules. Each module is demonstrated step-by-step using vector diagrams reflecting important data changes;
- In the encryption mode module, users can drag and drop components and connect them to implement the reuse of different pattern modules;

* Corresponding author.

E-mail address: duangh@csu.edu.cn.
The system administrator can create examination questions and users. The users can select the questions, and submit the answers and corresponding codes to the system online. The system will return the results as soon as possible;

Before compiling and running the code, Docker as a sandbox, restricts the execution time and memory of code. Hence, the system has good security.

The system can be provided to the teachers for teaching aids, to demonstrate a cipher algorithm principle or a working process for students in classrooms. The system can also be used as an independent experimental platform by students to intuitively learn cryptographic algorithms and deepen their understanding of cryptography.

2. Related Work

Cryptography virtual labs can be divided into three categories in structure: desktop applications, B/S and Client/Server (C/S). In desktop applications, Adamović et al. [1] used the third party tool CrypTool [15] to assist teaching in cryptography, which contains a large number of cryptographic algorithm experiments. Though CrypTool collects a number of cryptographic algorithm presentation tools, it is not convenient for users to implement it to assist teaching as it requires a host equipped with a variety of executable files.

Based on C/S structure, EAVS [6] is implemented by Java and SQL Server database. The client uses Swing control elements to form a friendly graphical interface. Students operate the client to validate cryptographic algorithm processes, and teachers maintain the back end. However, EAVS only implements some classical cryptography algorithms without any workflow display interface.

For B/S structure, Yen-Hung Hu et al. [5] propose a visualization tool for learning cryptography algorithms. They use JAVA applet to implement this tool, which allow users to utilize existing algorithms or create new ones, and to configure parameters and customize outputs. CSVE [10] is a communication system based on Common Object Request Broker Architecture (CORBA) technology and JavaBean components, and employs the matrix Lab (MATLAB) as the computing background. In [11], Java Applet is used to implement the client while the Web server side provides Java class files and transmits the JavaBean components and sub-categories to the client based on the user’s request. To carry out cryptography experiments, one needs to manually combine various components -- this procedure becomes very cumbersome and time consuming when an experimental demonstration of a cryptographic algorithm requires many components.

Some universities have also tried to develop its own software system to assist cryptography teaching. Asseisah et al. [2] designs and implements an interactive visualization DES algorithm demonstration system, which can graphically show the process of the DES algorithm workflow and the changes of the internal data in detail. AESvisual [8] is a part of cryptography visualization tools for the learning of the AES cipher. It includes VIGvisual, DESvisual, RSAvisual, etc. This desktop application tool involves the demo mode and the practice mode, which can run on Windows, MacOS and Linux. Gaffer et al. [4] try to use the virtual machine and the Ubuntu system to build a cryptography experimental platform. Users can use some commands of the Ubuntu system to implement the cryptographic algorithms and display the results.

Security Education (SEED) labs [3] on prebuilt VM images of Ubuntu has three exploration labs on cryptography: Secret-key encryption, One-way hash function and Public-key cryptography and students can launch a PKILabServer to go through the exercise. Richards et al. [9] developed a collaborative virtual computer laboratory (CVCLAB) which hosts a collection of virtual machines with unique tools. Cryptography is one topic of the new Cyber Security Laboratory. Students can access it via a web browser or a client interface. Crypto-Tutor [7] is a web application to learn modern cryptographic algorithms, and is implemented by AngularJS and Node.js [20]. It provides demos for each algorithm, which can be executed with customized inputs and configurations.

In summary, the existing experimental platforms can provide some cryptographic algorithm experiments, but they are not efficient at showing required specific data analysis and displaying dynamic process, and it is also short of cryptographic protocols and algorithms applications. Moreover, these systems do not have an experimental platform for hands-on programming exercises. In this paper, VESC is designed to solve these problems.

3. System design

3.1. The design of function structural

The main implementation for information security courses is about the mathematical foundation, classical cryptography, common modern cryptographic algorithms, and experiments of cryptographic protocols. With the above analysis, the
system designs cryptography experiments in the following three categories: the basic mathematical experiments, the cryptographic algorithms experiments, and the cryptographic protocols experiments.

The function design prototype of this system is shown in Figure 1.

Basic Mathematical experiment module displays data changes with mathematical formulas and tables. It includes three types of algorithms:
- Euclidean algorithm, the original proof of root, primality testing algorithm, fast exponentiation algorithm
- Threshold scheme: Shamir threshold scheme, Bloom threshold scheme
- Baby-Step and Giant-Step algorithm in discrete logarithm

Cryptographic algorithm experiment module shows algorithm process and intermediate data in the form of dynamic graphics. It includes two types of algorithms:
- Classical cryptographic algorithms: shift cipher, affine cipher, Vigenere cipher
- Modern cryptographic algorithms: sequence cipher(LFSR\RC4), symmetric cipher(DES\AES), asymmetric cipher(RSA\ElGamal\ECC), digital signature (RSA\DSA), hash algorithm(MD5\SHA)

Cryptographic protocols experiment module focuses on simulating the communication process on both sides. It includes three types of experiments:
- Encryption modes: ECB, CBC, OFB, CFB
- Key exchange protocols: Diffie-Hellman, EKE
- Identification protocols: SKEY, Fiat-Shamir

On the system function prototype, VESC includes a variety of cryptographic algorithms. To implement friendly interaction interfaces, we need to customize different functions for the different types of algorithms. General functions can include the following sections: interface display and parameter input. The basic mathematical experiment module customizes calculation process functions; the cryptography algorithm experiment module customizes corresponding encryption processes, key generation, encryption, decryption, signature, etc.; the cryptographic protocol encryption mode requires components combination functions and runs four encryption modes after users drag, drop and connect components.

3.2. The design of system architecture

Cryptography experiments involve a large number of mathematical operations. If some parameters are set incorrectly or there are too many concurrent accesses to this system, the server will slow down or even crash. In order to avoid this problem, two following strategies are usually adopted. One strategy is that, each parameter from requests is verified to be within the prescribed scope and another one is to conduct the calculation work in the sandbox. However, it is difficult to debug and is inconvenient for maintenance, and the sandbox mechanism makes the application consume more hardware resources, increasing the burden of the server. Therefore, in VESC, the computation of demonstration section is done in the browser. We only need to validate input parameters for the browser. Even if a user maliciously inputs improper parameter values, in the worst case, their own browser is frozen yet does not affect the security of the server. To reduce the overhead for faster responses, the parameters can be restricted to be relatively narrow.
VESC adopts the B/S structure. It divides users into two categories: teachers and students. Teachers use the system to demonstrate the principle and workflow of different encryption algorithms. This part of work is accomplished in the front end to relieve the burden of the server. The back end is used to execute codes and manage user information and examination questions. By separating teaching from programming, as shown in Figure 2, the server executes fewer algorithms and thus achieves better performance and higher efficiency, as well as enhancing its security.

The system uses two databases, one for storing management information and user-generated information, such as questions, users, code and results; another for storing session information and lifetime of requests, which makes the maintenance easy and achieves better interaction. Through the interaction with the database, the system handles creating, reading, updating, and deleting (CRUD) operations and real-time feedback to the user with the returned data.

Docker [16] is an open source container engine based on lightweight virtualization technology. Docker can be used to rapidly deploy applications within the container, with resource isolation and safety guarantees on kernel virtualization technology. Container virtualization technology has little performance loss with kernel-sharing. The isolation of the operating system layer is achieved by virtualization technology. When the container is running, it does not incur additional operating system overhead. Therefore, it can run one application within each container and different containers are isolated from each other. The creation and stop of a container is very fast (with latency of few seconds).

Docker uses the namespace technology to isolate resources, uses the cgroups technology to limit resource, and uses copy-on-write to improve file operation efficiency. Docker supports seccomp since version 1.10. Seccomp (secure computing mode) is a kernel feature of Linux for constraining the range of system call a process can use, which reduces the kernel attack and thus is widely used to build sandbox. To make a sandbox safer, Docker needs to limit the container capacity by configuring parameters when starting a container. With the above characteristics, the system chooses Docker to run untrusted code in some container as sandbox. Once a student submits his code, the server will send the code to the database and Docker. After the execution is finished, Docker will send back the code status and other result files to the server, and the server will send these files to the database as quickly as possible and feed the front end with the data from the files.

3.3. The design of operation process

The demonstration section

For an easy user interaction, users can choose a variety of cryptographic algorithms experiments, which are independent of each other. In this section, most experiments require the user to input some parameters and click some buttons to show the contents. The encryption experiment model has four algorithms, and the four experiments can be combined in different ways. So the four algorithms are implemented in a drag-and-drop way. Users can drag and drop components as they want onto the canvas and connect them on click. As shown in Figure 3, while the users input parameters or drag components, the system displays the algorithm flow vector diagram, data calculating process, etc.
The programming section

Students can fill each box according to the examination question. When they click the submit button, the server will receive their code. After executing the code in Docker and analyzing the results on the server, the results will be shown on the web pages of users. The results include pending and waiting, judging again, compiling, running and judging, format errors, not entirely correct, time expired, memory limit exceeded, output limit exceeded, runtime error, compile error, accepted, etc. Students just finish their answers and do not need to care about how the code works in the back end. The server verifies the correctness of the source code according to some predesigned test data. We set several groups of input data and output data. The input data will be sent to Docker and run with the code, and the actual output data will be sent to the server from Docker. Then the server compares the difference between the data outputted by the user’s program and the standard output samples, or checks whether the results of the user’s program meets the predetermined logical conditions.

The data flow in Docker is shown in Figure 4. Each code has its own id and folder. The system creates a project by saving the code to a file in the folder. Each directory will be loaded into the container when the container is created. When the container is started, it will compile and run the code. If the code execution process is timeout, the whole process will be hung and the container will turn to be waited and then be killed. If the code execution process is done in time, when Docker waits the container, it will not do the wait process actually and return a status different from the timeout condition. When the entire process is finished, the system will remove the code running environment, namely the container. However, the mounted project will be completely deleted only after the corresponding state files and other output files have been stored in the database.

4. System implementation

4.1. The development environment

The front end

All the data processing operations of this part are completed in the browser. We use WebStorm to develop multiple plug-ins and implement the corresponding functions. The system uses HTML5 to implement Web applications. We test the system efficiency on the Node.js server deployment in a highly concurrent environment.
The system chooses Bootstrap [13] as the front end framework, and carries out all of the experimental computing and data processing of cryptographic algorithms by JQuery [18], then displays mathematical formulas on the browser page by MathJax [19]. The displays of vector graphs and animation effects are implemented by Raphael [21]. JQuery-ui and Raphael are used to implement the components in drag-and-drop and the effect of the attachment in encryption mode. An algorithm in the system corresponds to an HTML, and at the same time corresponds to a customized JavaScript program; in the AES case, it corresponds to aes.html and aes.js, etc. The system uses CodeMirror [14] to optimize the user experience for code editing. CodeMirror is a versatile text editor implemented in JavaScript for the browser. It is specialized for editing code, and comes with a number of language modes and addons that implement more advanced editing functionality. The core library provides only the editor component, no accompanying buttons, auto-completion, or some other IDE functionality.

The back end

We use Node.js [20] as the server. Node.js is a JavaScript built on Chrome's V8 JavaScript engine. It uses an event-driven, non-blocking I/O model, which makes it lightweight and efficient, and responds quickly in highly concurrency environments. Express [17] is a minimal and flexible Node.js web application framework based on MVC implementation. With Express, the back end server can be built quickly. Docker is used as the sandbox to execute untrusted code in the system. Because the user data and question data have some overlapping fields, in order to perform time-efficient multi-table query when the amount of data is large, we use MySQL as the database because relational database is more efficient than non-relational database when processing cross-table query.

4.2. System function implementation

According to the analysis of operation process design in Section 2.3, the interaction of the teaching section mainly occurs between the user and the browser. The browser is mainly for Web page display and algorithm computation. Page display includes the dynamic display of the vector diagram and the data change after algorithm module. Users only need to input parameters and drag, drop and connect components to start the experiments.

Parameter input

The main function of the parameter input module is to enter parameters for the users and to verify the types and scopes of the parameters. If the parameter values are not correct, the system will directly prompt the users for parameter error. Meanwhile, according to the characteristics of the algorithms, some parameters of algorithm experiments can be randomly generated within predefined ranges. The verification and random generation of parameters are done in the browser.

Vector diagram display

On a Web page vector graphics and animations are shown by Raphael. In various cryptographic algorithms, the corresponding algorithm flow diagrams are also different. At the beginning, the system gets some parameters needed for vector graphics. Then, after cleaning the entire canvas, the system begins to draw the static vector graph of algorithm experiments. If there is a need for a flow chart of animation, the system will dynamically display vector graphics. Figure 5 shows the LFSR generation process of key stream with vector graphics operations.

Algorithm and data module

When implementing different cryptographic algorithms, the system divides the algorithms into several steps. Each step is encapsulated as an object. Each step of internal operations is independent of the other steps. Between two consecutive steps,
the output of the previous step is the input of the next step. This makes the whole process of experiment as a complete algorithm demo. Each step implements certain operations, such as vector graphics operations and/or mathematical operations. Figure 6 shows the calculation process of fast exponentiation algorithm with mathematical operations.

\[
\begin{array}{c|c|c}
 x & e & y \\
\hline
 5 & 3 & 1 \\
 5 & e^1 \mod 2 & x \mod 5 \mod 5 = 0 \\
 0 & e^1 = 0 & x \mod 5 \mod 5 = 0 \\
\end{array}
\]

Figure 6. Mathematical calculation in fast exponentiation algorithm

**Drag-and-drop mode**

In the encryption modes section, VESC implements component-based experimental design and component connection. A user can drag, drop and connect components together. As shown in Figure 7(a), users can realize the OFB mode that has fixed combinations and links. As shown in Figure 7(b), users can choose different components and link them in any combination within some necessary limitations. This mode is called customized mode.

On the toolbar, users can click the settings button to select the parameters for different encryption modes. The parameters include the key and text, format (hex and binary) and padding patterns (Zeros, PKCS5, ISO10126, ANSI923 and None). If users use the CFB or OFB modes, they need to select an extra parameter of left shift. In the customized mode, users set the secret key and the clear text by double clicking the components and filling in the pop-up input box. With a right mouse click, users can also easily delete the components and lines. The canvas can be zoomed in and out and cleaned out by the buttons on the toolbar and can be dragged around by mouse.

![Figure 7](image)

Figure 7. Encryption modes: (a) The OFB mode, and (b) The customized mode

In the customized mode, the in-degree and out-degree of all nodes are limited. The type of the source node and target node is appropriate. Because the component graph used in the experiment cannot be cyclic, the join graph must be a directed acyclic graph.

**Security configuration**

The interaction of the programming section occurs between the user and the server. Users can select different programming languages and CSS themes. When the server gets the code submitted by the user, it will create a file in a new folder to save the code. The project folder can be mounted into the container through the parameter `-v` when the container is started.

We use Docker on Windows 8.1 during the testing phase. However, Docker runs on an Ubuntu virtual system in the VirtualBox, and thus the performance on Windows is not as good as it on the native Ubuntu system. The future version of VESC will run on the Ubuntu operating system. On Windows host system, users also need to set the mount path of VirtualBox. The following parameters need to be set for Docker to enhance security.

- Set a normal user in the container. It would be better if the application in the container does not run as a root user. This will prevent programs in the container to access higher levels of access and manipulation. We set the `-u nobody` parameter.
• Restrict the container network. Containers should expose as few ports as possible. In general, the containers can communicate with each other even if the container port is closed. To avoid such cases, Docker service (Docker daemon) is required to specify the --icc = false flag, to close the communication between the containers.

• Limit memory of containers. Docker uses -m and --memory-swap flag to limit container memory. The -memory-swap flag indicates the total capacity of memory and swap memory. If the -m flag and --memory-swap flag are set to the same value, there will be no space for swap memory. If only -m flag is set, the total memory is two times of the value indicated in the -m parameter.

• Restrict restart. If the container keeps restarting, it will consume a lot of resources. We can use the --restart=on-failure:10 when starting the container. The number 10 can be replaced by any other number.

• Restrict file system. The restriction of file system is mainly to prevent the random-write (1scripts) attack in the container. We can use the --read-only parameter.

The system also has other methods to ensure its security. Authentication is used to sign up and sign in, and the appropriate permission is set for each user. When users submit their codes, in order to avoid multiple submissions in a short time, the back end server limits the rate of requests and increases the delay time of each request. To void command injection, we use child_process.execFile rather than child_process.exec because the latter is a bash interpreter rather than a program launcher. To prevent Cookie thefts, the HttpOnly flag can be set on cookies, which makes the cookies unreachable for Javascript.

5. System Performance Evaluation

5.1. The operating efficiency in Docker

We test a simple sample that adds two integers to get the result, as shown in Figure 8. In the folder from this submission, we can get the status:0 and the actual result is the same as the sample output. We can also get the compiling time and running time. There is little difference between the execution time in Docker and the execution time in the local host environment.

5.2. Comparisons with similar software

A comprehensive comparison is conducted among CrypTool, EVAS, VCL, AESvisual, CVCLAB, Crypto-Tutor, and VESC in terms of system structure or platform, number of experiments, dynamic demonstration, user-defined input and programming exercises. The results are shown in Table 1.

The VCL in [2] has at least three algorithms including DES, visualizing variants of DES and AES. AESvisual [8] is a series of cryptography visualization tools which include AESvisual, VIGvisual, DESvisual, RSAvisual, ECvisual, and SHAvisual. The cryptographic algorithms in CVCLAB include OpenSSL, some traditional and modern algorithms, public key infrastructure, and key exchange. Designing VESC, we take into account that the system should have good extensibility for adding new experiments of cryptography algorithms in the future. Now, we have implemented 29 experiments of cryptographic algorithms.
Research on Techniques and Methods of Developing Cryptography Virtual Laboratory

Table 1. Compared with the similar systems or tools (VCL is used if no name is specified)

<table>
<thead>
<tr>
<th>Systems /Tools</th>
<th>Structure/Platform</th>
<th>The number of experiments</th>
<th>Dynamic demonstration</th>
<th>User-defined input</th>
<th>Programming exercises</th>
</tr>
</thead>
<tbody>
<tr>
<td>CrypTool [15]</td>
<td>desktop application</td>
<td>52</td>
<td>partial</td>
<td>partial</td>
<td>-</td>
</tr>
<tr>
<td>EAVS [6]</td>
<td>C/S</td>
<td>4</td>
<td>-</td>
<td>√</td>
<td>-</td>
</tr>
<tr>
<td>VCL [11]</td>
<td>B/S (component based)</td>
<td>7</td>
<td>-</td>
<td>√</td>
<td>-</td>
</tr>
<tr>
<td>VCL [2]</td>
<td>desktop application</td>
<td>3+</td>
<td>√</td>
<td>√</td>
<td>-</td>
</tr>
<tr>
<td>AESvissual [8]</td>
<td>desktop application</td>
<td>6</td>
<td>√</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>VCL [4]</td>
<td>Ubuntu VM</td>
<td>7</td>
<td>-</td>
<td>√</td>
<td>execute preset programs</td>
</tr>
<tr>
<td>CVCLAB [9]</td>
<td>B/S(VM)</td>
<td>8+</td>
<td>-</td>
<td>√</td>
<td>execute preset programs</td>
</tr>
<tr>
<td>Crypto-Tutor [7]</td>
<td>B/S</td>
<td>6</td>
<td>-</td>
<td>√</td>
<td>-</td>
</tr>
<tr>
<td>VESC</td>
<td>B/S</td>
<td>29+</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>

5.3. Performance test

In the high concurrency environment, to test the system performance, we deploy VESC to the Node.js server, and use Apache Benchmark [12] to test its concurrent performance.

In the LAN environment, by simulating the number of concurrent users 20, 50, 100, 200, 500, 1000 and setting each user request to 100, we test the average response time for the test server at each request. The testing results are shown in Figure 9.

Simulating the number of multiple requests at a time 100, 200, 500, the Figure 9 shows the different percentages of the requests served within a certain time. When the number of concurrent requests is 100, the time is almost always in one second. In the other two situations, some requests can get responses after a short time.

![Figure 9. (a) System concurrency test (b) Server equalization test](image)

Figure 9(a) shows that the average response time is substantially proportional to the number of concurrent users, and the increase of concurrent users does not cause the surge of average response time. In Figure 9(b), the time increases steadily along with the requests within a time, which means the ability of the server to provide for each request in this concurrent situation is balanced when concurrent requests are less than 500. The results show that in this paper VESC can still have a good performance in a high concurrency environment.

6. Conclusions

In this paper, we propose a virtual experiment system of cryptography in B/S structure. With the user inputting parameters, the system dynamically presents the execution of algorithms and the data changes to the students, both as a teaching assistant system and an independent platform for student experiments. It has the following advantages: easy interaction, less overhead, support for concurrent access, and high efficiency.

The system has many CSS files and add-ons, which slightly affect its response time. In the future, we will further optimize the response delay by reducing unnecessary resource loading. Docker provides many security mechanisms.
Currently, we have implemented cryptographic algorithm experiments in the demonstration section, accomplished part of the back end and done some tests in the programming part. In the next step, we will continue to add other more algorithm experiments involved in cryptography courses and finish the code of back end. We will continue to improve the user-system interaction interface. Additionally, we need to do more research on the security.

References