Cipher Tech Solutions, Inc.
407 North Highland Ave
Upper Nyack, NY 10960
www.CipherTechSolutions.com
Phone: 888-948-8324
Fax: 845-230-6632

Applicants,

Cipher Tech uses various technical challenges to assess the skills and aptitude of potential hires seeking software development positions within our company. You must successfully complete all of the challenges to be considered for the next round of interviews.

We understand these challenges may require varying levels of research. If you choose to use a code snippet or module directly from a different author, you are required to give credit to that author. This includes code segments and/or modules directly provided by professors and internet sources. These challenges are designed to test your technical knowledge and ability, so please ensure that the solutions you submit reflect your ability.

Your solutions for Challenges 1 & 2 will be evaluated in a Windows environment that does not have internet access. Your submission for Challenge 3 should avoid being OS dependent, though it will be evaluated in a Linux environment that does not have internet access. You may use different languages for different challenges, though please choose from the languages below.

Challenges 1 & 2: C, C++, C#, Java, Python
Challenge 3: PHP, JavaScript (Angular/jQuery preferred, not required), Java, Python

Each challenge will include what you are expected to return to us, but please note that your submission should **NOT** include compiled binaries for Challenges 1 & 2, this PDF, or any of the provided sample files.

We look forward to reviewing your responses!

- The Cipher Tech Technical Recruiting Team

Challenge 1: LFSR

A common technique for obfuscating data is to use exclusive-or (XOR) with some key; it is inexpensive and symmetric. A problem occurs when this is used on file formats like portable executable where there are many null bytes, since XORing nulls with the key ends up writing the key out.

A slightly more complex algorithm uses a Linear Feedback Shift Register (LFSR) to produce a key stream, which will be XORed with the data. A generic LFSR is:

> If F is the value which represents the LFSR feedback and S is the current state of the LFSR, the next state of the LFSR is computed as follows:
>
> if the lowest bit of S is 0: `S = S >> 1`
>
> if the lowest bit of S is 1: `S = (S >> 1) ^ F`

For this challenge, you'll be using an LFSR with the feedback value 0x87654321. The LFSR is initialized with a value and stepped to produce the key stream. The next key value is read from the LFSR after eight steps, with the actual key being the lowest byte of the current LFSR value.

For example, if the initial value of the LFSR is `0xFFFFFFFF`, then next value after stepping it eight times will be `0x9243F425`, meaning that the first key byte is `0x25`. If the first byte of the input is 0x41, the first byte of output will be 0x64.

Your task is to implement this algorithm (both LFSR and the XOR). We're only interested in algorithm implementation; other code will be discarded.

The function should adhere to one of following signatures:

C/C++	`unsigned char *Crypt(unsigned char *data, int dataLength, unsigned int initialValue)`
C#	`static byte[] Crypt(byte[] data, uint initialValue)`
Python	`Crypt(data, initialValue) # returns byte string`
Java	`static byte[] Crypt(byte[] data, long initialValue)`

Example Tests:

data	“apple”
dataLength	5
initialValue	0x12345678
result	“\xCD\x01\xEF\xD7\x30”

data	“\xCD\x01\xEF\xD7\x30”
dataLength	5
initialValue	0x12345678
result	“apple”

Submit: Source code containing your implementation of the LFSR-based encoding routine.

Challenge 2: KDB Files

During a forensic investigation, a new data storage file format is discovered with the extension KDB. In addition to being stored in a custom format, the contents are encoded with the LFSR algorithm seen in Challenge 1, so all of the data contained in the file format must also be decoded once extracted. Using the provided spec on the next page, create a program that extracts and decodes the enclosed data.

- Your program should accept a path to a KDB file via command line argument.
- Your program should report the extracted, decoded information to standard out with each entry (name and stored information) on a separate line.

Included is a sample KDB file named “store.kdb”. The initial value for the LFSR algorithm is 0x4F574154.

Submit: The source code of your solution **AND** a text file containing the standard out when your solution is run with store.kdb as its input.

KDB File Specification

HEAD		
MAGIC	**BYTE[6]**	**“CT2018”**
ENTRY_LIST *	**INT32**	**Pointer to the ENTRY_LIST**

ENTRY_LIST		
ENTRIES	**ENTRY[127]**	**Array of ENTRIES**

ENTRY		
NAME	**CHAR[16]**	**Null terminated string**
BLOCK_LIST *	**INT32**	**Pointer to the BLOCK_LIST**

BLOCK_LIST		
BLOCKS	**BLOCK[255]**	**Array of BLOCKS**

BLOCK		
SIZE	**INT16**	**Length of the BLOCK’s data**
DATA *	**INT32**	**Pointer to the BLOCK’s data**

DATA		
DATA	**BYTE[]**	**An array of bytes**

- All pointers are relative to the beginning of the file.
- The BLOCK’s size and all pointers are little endian.
- All KDB files begin with a HEAD at 0x0, which starts with the magic bytes “CT2018” and contains a pointer to the file’s ENTRY_LIST.
- The ENTRY_LIST is an array of up to 127 ENTRIES. The value 0xFFFFFFFF signifies the end of the ENTRY_LIST.
- Each ENTRY has an ASCII encoded, null-terminated name and a pointer to the ENTRY’s BLOCK_LIST.
- The BLOCK_LIST is an ordered array of BLOCKS. The value 0xFFFFFFFF signifies the end of the BLOCK_LIST.
- Each BLOCK contains a pointer to that BLOCK’s DATA and the size of that DATA.
- The DATA is an LFSR (from Challenge 1) encoded byte array. All BLOCKS from an ENTRY are combined then decoded to reveal the stored information.

Challenge 3: Web

There are two options for this challenge. It does not matter which you choose, but please complete only one of them.

Regardless of the option you choose to complete, the recommended stack is as follows:

- PHP
- JavaScript (preferably AngularJS)
- SQL (MySQL, PostgreSQL, MS SQL Server, or Oracle DB)

Option 1: Login Form

Nearly every web application utilizes a login system to protect proprietary or private resources and information. Your challenge is to implement a basic database-connected login form that allows a user to access a "Secure Page" after submitting a username and password to authenticate and create a session. Once logged in, the user should be redirected to the Secure Page. The Secure Page needs simply to state "This is the secure page" and display a functional logoff button that invalidates the user session.

- Create a login page
- Create the simple destination "secure" page with the desired text and a logoff button
- If an unauthenticated user attempts to access the secure page, they should be prompted to login, with an error message
- Provide appropriate validation and error messages as required
- Provide a logout mechanism that invalidates the user session
- Provide all necessary scripts to (re)create the database for testing, including any applicable tables, views, stored procedures, and/or sample data

Submit: The source code of your solution **AND** all related database scripts

Option 2: Code Submission

You may submit a previous or alternative project that you developed **independently** that demonstrates your experience with web technologies. There is no requirement for key functionality or style, but it should encompass at least one of the key areas that Cipher Tech and its customers focus on:

- Security and authentication
- Data validation and sanitization
- Data display, visualization, and analysis

Submit: All source code **AND** related SQL scripts **AND** associated files and/or data **AND** deployment instructions.

As a reminder, all code submitted must be developed exclusively by you.
As such, you must be the sole owner of the code.