Data Encryption and Password Remembrance Software

Charles Faulkner

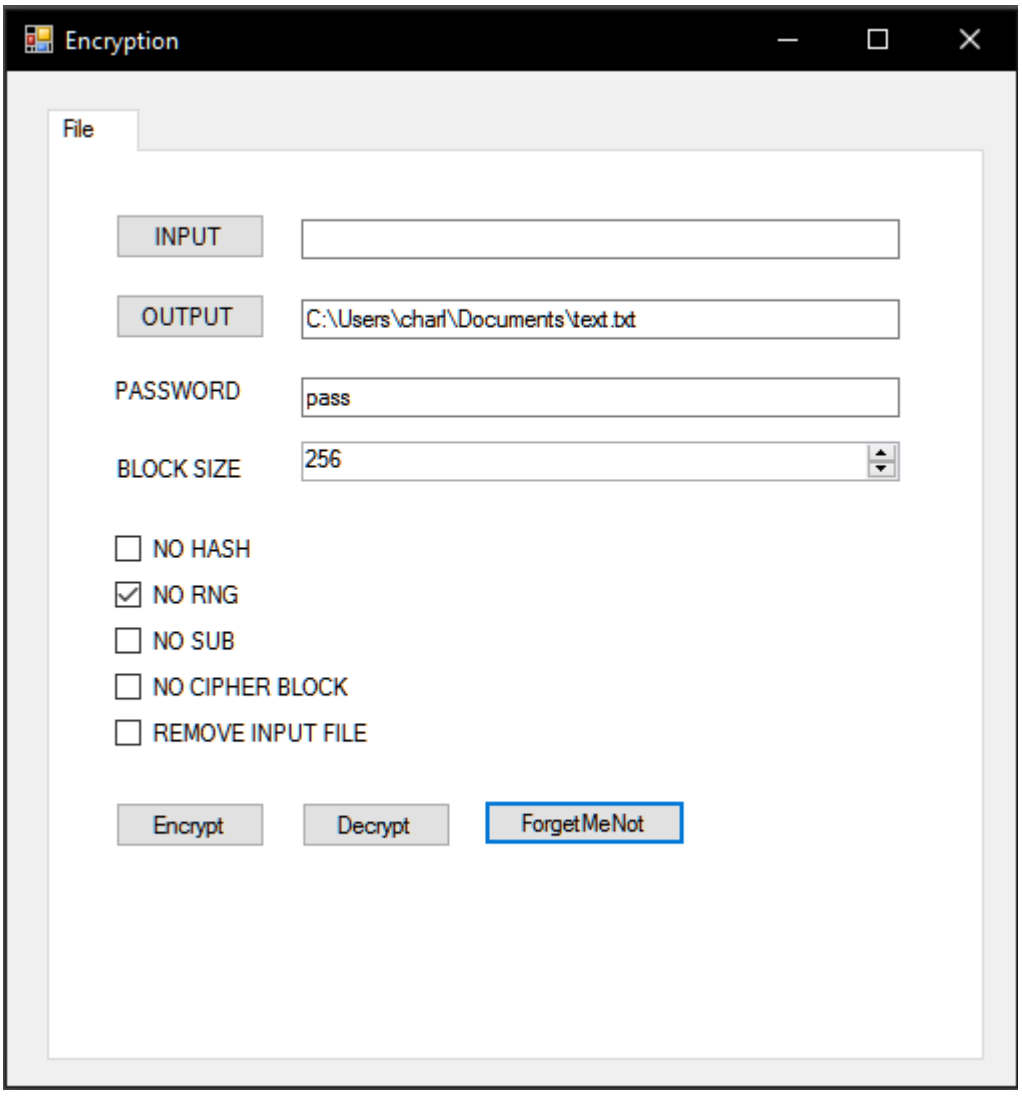
4/30/2017

Abstract

In today's society privacy and security is an important part of the technology we use. The best way to keep both those things intact is through encryption. These programs Encryption.exe and ForgetMeNot.exe are meant to protect entire files and account-username-password groups respectively. They use the Trust No One (TNO) level of security and allow customization of the encryption algorithm to ensure that documents are secure.

Table of Contents

Contents



Resources Used

The encryption software was written in C++. The GUI was written in C# and calls the programs with the appropriate arguments.

The CLHF shared library is being used for the encryption and decryption class. The CLHF library is my own proprietary library.

The programs are designed to work on the Windows 10 64-bit operating system.

Arguments

The argument list for ForgetMeNot.exe is a subset of the argument list for Encryption.exe

Both Programs

(-no_hash) Do not hash the password

(-no_rand) Do not use random number generation for the key

(-no_sub) Do not use the substitution box

(-no_cb) Do not use block cipher encryption

(-block_size=#) change the number of bytes in the block cipher

(-output=[filename]) set the output destination

(-pass=[password]) set the password

Encryption.exe Only

(-input=[filename]) sets the input filename

(-no_rm) do not remove the input file

(-dec) Decrypt the file

(-enc) Encrypt the file

Encryption.exe

The main function for Encryption.exe first parses the arguments. From there it goes one of two ways.

Encrypting:

```
CLHF::File_Encryptor output(output_file_name, key_, options, block_size);

std::ifstream input_file;
if (true == CLHF::Utility::GenerateFile(input_file_name, input_file)) {

	std::vector<uint8_t> buffer;
	//get data from input file
	CLHF::Utility::GetNextBytesFormFile(buffer, input_file, 10000);

	while (buffer.size() > 0) {
		//encrypt the data into the ouptut file
		output.encrypt(buffer);
		//get more data from the input file
		CLHF::Utility::GetNextBytesFormFile(buffer, input_file, 10000);
	}
} else {
	LOG_ERR("Failed to open input file.");
	return false;
}
```

Decryption:

```
CLHF::File_Decryptor input(input_file_name, key_, options, block_size);

std::ofstream output_file;

if (true == CLHF::Utility::GenerateFile(output_file_name, output_file)) {

	std::vector<uint8_t> buffer;
	//get decrypted data from file
	input.decrypt(buffer, 10000);
	while (buffer.size() > 0) {
		//write data to file
		output_file.write(reinterpret_cast<char*>(&buffer[0]), buffer.size());
		//get more data
		input.decrypt(buffer, 10000);
	}
} else {
	LOG_ERR("Failed to opon output file.");
}
```

ForgetMeNot.exe

The main function for this application parses the arguments like Encryption.exe. It then decrypts the file given and parses the data into account names, usernames, and passwords.

```
//vector for storing the account information
std::vector<usrpass> ls;
//read the file and get the unencrypted data as a string
std::vector<uint8_t> data;
loadData(data, filename, key_, options, block_size);
//parse the data into the vector
parse(data, ls);
//sort the data for dislpay
std::sort(ls.begin(), ls.end());
displayData(ls);
```

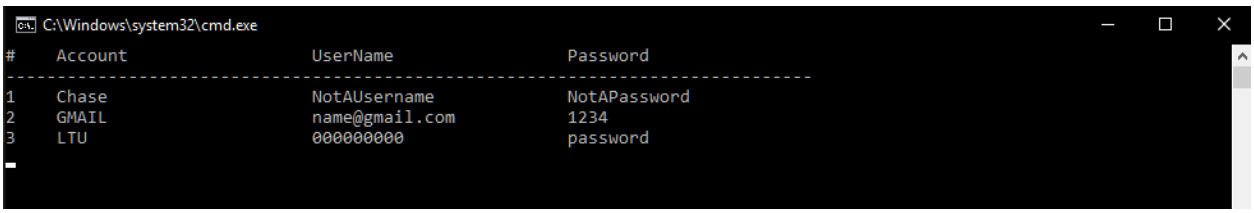
The inside of the loadData() function is a look that decrypts the data from the file until the there is no more data to decrypt. The parse() function takes the data and separates it based off of three delimitators.

```
const char accountDelim = 7;
const char usernameDelim = 8;
const char passwordDelim = 9;
```

The parsed data is stored in a vector of type userpass:

```
struct usrpass {
	usrpass(std::string acc, std::string usr, std::string pass) {
		account = acc;
		username = usr;
		password = pass;
	}
	std::string account;
	std::string username;
	std::string password;
	bool operator<(const usrpass& a) const { return account < a.account; }
};
```

These account-username-password groups are than output to the display.



Then the user us able to user commands to:

-add accounts

-remove accounts

-clear the screen

-quit

Encryption Algorithm

The algorithm for encryption is pretty straight forward:

```
for (uint32_t i = 0; i < buffer.size(); ++i) {

	buffer[i] = buffer[i] ^ key->GetNextByte();

	if (false == (options & NO_SUB)) {

		buffer[i] = sub_box[buffer[i]];
	}

	if (false == (options & NO_BC)) {
		buffer[i] ^= IV[IV_pos];
		IV[IV_pos] = buffer[i];
		IV_pos = (IV_pos + 1) % block_size;
	}
}
```

The algorithm encrypts one byte at a time. It XORs the byte with the key using the GetNextByte() method. It then puts the byte through the substitution box. The final step is to put it through the block cipher.

Decryption Algorithm

The algorithm for Decryption is the same as the Encryption but reversed.

```
for (uint32_t i = 0; i < buffer.size(); ++i) {

	uint8_t temp = buffer[i];

	if (false == (options & NO_BC)) {
		buffer[i] ^= IV[IV_pos];
		IV[IV_pos] = temp;
		IV_pos = (IV_pos + 1) % block_size;
	}

	if (false == (options & NO_SUB)) {

		buffer[i] = sub_box[buffer[i]];
	}

	buffer[i] = buffer[i] ^ key->GetNextByte();
}
```

Key

The Encryption and decryption algorithms both use keys to encrypt the data. These keys are derived from the password and the arguments (-no_hash) and (-no_rand).

In total, there are four keys:

1. Key that cycles through the password (-no_hash) (-no_rand)
2. Key that cycles through a hash of the password (-no_rand)
3. Key that uses a random number generator seeded by the password (-no_hash)
4. Key that uses a random number generator seeded by a hash of the password

The hash used is SHA 256 and the random number generator used is a Mersenne Twister.