Государственное бюджетное общеобразовательное учреждение

средняя общеобразовательная школа № 471

Выборгского района Санкт-Петербурга

Индивидуальный проект

**«*Шифрование, кодирование и хеширование.*»**

Выполнил:

***Ефремов Никита Русланович***

Ученик 9 ,,Г’’ класса

Руководитель проекта Ф.И.О.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2022-2023 учебный год.

**Содержание**

[Введение 3](#_Toc120715440)

[Актуальность 3](#_Toc120715441)

[Цель 3](#_Toc120715442)

[Задачи 3](#_Toc120715443)

[Объекты исследования 3](#_Toc120715444)

[Предметы исследования 3](#_Toc120715445)

[Методика исследования 3](#_Toc120715446)

[Глава I 4](#_Toc120715447)

[Вступление 4](#_Toc120715448)

[Хеширование 4](#_Toc120715449)

[Коллизии хеш-функций 6](#_Toc120715450)

[Кодирование 7](#_Toc120715451)

[Опыт 8](#_Toc120715452)

[Шифрование 9](#_Toc120715453)

[AES 9](#_Toc120715454)

[Опыт 10](#_Toc120715455)

[Глава II 12](#_Toc120715456)

[Продукт 12](#_Toc120715457)

[Вывод 12](#_Toc120715458)

# Введение

## Актуальность

Проблема защиты информации остаётся актуальной до тех пор, пока не исчезает спрос на получение доступа к той или иной информации. История кибербезопасности начинается с появления первых атак на компьютеры (В 1989 году был создан первый компьютерный червь-самораспространяющийся вирус) и продолжается до сих пор.

## Цель

* Понять и изучить основные способы и алгоритмы шифрования данных.

## Задачи

* Узнать почему важно хранить информацию в безопасности.
* Узнать об основных видах шифрования информации (Encryption, encoding, hashing)
* Изучить наиболее популярные алгоритмы каждого из видов.
* Понять разницу между видами шифрования, а также разобраться, в каких ситуациях подойдёт той или иной вид шифрования.

## Объекты исследования

* Шифрование, кодирование и хеширование

## Предметы исследования

* Проблемы основных видов шифрования информации

## Методика исследования

* Анализ
* Опыт

# Глава I

## Вступление

Когда дело доходит до хранения своей информации в безопасности, существует три основных способа сделать это: **шифрование, кодирование и хеширование**.

Достаточно часто эти термины являются взаимозаменяемыми, но знать разницу между ними – **очень** важно. В этом исследовании мы узнаем об этих методах и о том, как они работают.

# Хеширование

* Начнём с самого интересного – с **хеширования**. Хеширование – это процесс преобразования входных данных в **уникальную** последовательность символов, причём этот процесс работает лишь в одну сторону (не является обратимым).
* Чаще всего хеширование используется для проверки целостности файлов, ведь если содержимое файла меняется хоть на **1 символ**, значение хеша поменяется **полностью**.
* Самая большая проблема хеширования – несмотря на то, что хеширования является необратимой функцией, получение доступа к данным всё ещё является возможным благодаря **базам данных хешей** или банальному **перебору**.

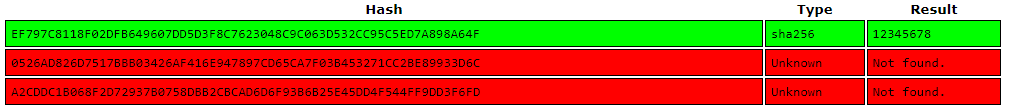
**Опыт**

* Допустим, мы **захешировали** свой пароль [**12345678**] благодаря алгоритму **SHA256**. Нам кажется, что он в безопасности, ведь количество хешей, которые данный алгоритм способен сгенерировать – **2 в 256 степени**. Это невероятно большое число (**~1.15\*10^77**).
* Кто будет тратить огромные вычислительные мощности на то, что-бы взломать наш пароль? Ответ на этот вопрос очень прост-**никто**. Взломщик просто воспользуется **базой данных хешей (например,** [**crackstation**](https://crackstation.net/)**’oм. Она хранит примерно 30% от всех хешей)**, ведь хеш нашего пароля уже есть в базе данных. Произошло это из-за того, что наш пароль не был достаточно **оригинальным**.
* Обойти это можно очень легко – добавить так называемую **криптографическую соль** в наши входные данные к хеш функции.
* **Криптографическая соль** – случайно сгенерированные данные, которые мы

используем вместе с нашими исходными данными в хеш функции. (Очень важно добавлять **соль** именно во входные данные **функции**, а не менять наши **изначальные** данные, которые мы хотим **защитить**, иначе теряется весь смысл использования **соли**). Рассмотрим **соль** в действии на примере базе данных, хранящей имена и **пароли** пользователей сайта и алгоритма хеширования **SHA256**.

* Достаточно часто у нескольких пользователей сайта **одинаковые** пароли. Тогда и **хеши этих паролей будут одинаковыми**. Но при хешировании мы можем добавить к каждому из паролей **разную** соль – тогда и хеши этих паролей будут полностью отличаться, а при **достаточной длине** и **хаотичности** соли полученные хеши уже не будут попадать в базу данных. Представим это в виде **таблицы**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Username** | **Password** | **Salt** | **Unsalted password hash** | **Salted password hash** |
| User1 | 12345678 | D;%yL9TS:5PalS/d | EF797C8118F02DFB649607DD5D3F8C7623048C9C063D532CC95C5ED7A898A64F | 0526AD826D7517BBB03426AF416E947897CD65CA7F03B453271CC2BE89933D6C |
| User2 | 12345678 | )<,-<U(jLezy4j>\* | EF797C8118F02DFB649607DD5D3F8C7623048C9C063D532CC95C5ED7A898A64F | A2CDDC1B068F2D72937B0758DBB2CBCAD6D6F93B6B25E45DD4F544FF9DD3F6FD |



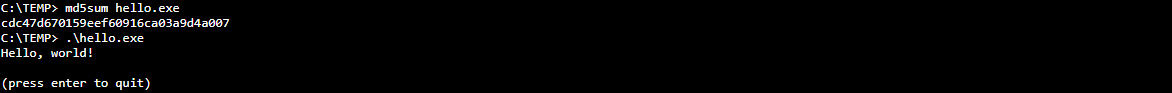
* Причём, даже если злоумышленник узнает значение нашей соли, он не сможет просто **,,вычесть’’** её из хеша, ведь **длина хеша всегда одинакова**.

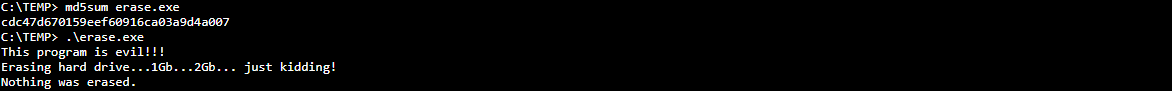
## Коллизии хеш-функций

* **Коллизии** – ещё одна проблема хеш функций; это когда несколько **различных** входных данных дают **одинаковое** хеш-значение. Происходит это по одной простой причине: количество возможных входных данных к хеш функции намного больше, чем количество хеш значений, которые способна сгенерировать хеш функция. На конец ноября 2022 года ещё не существует публично известных коллизий алгоритма **SHA256**, поэтому рассмотрим другой алгоритм: MD5. (Хеш этого алгоритма содержит **128 бит**, т.е существует **2^128** хешей (**~3.4\*10^38**)).

**Опыт**

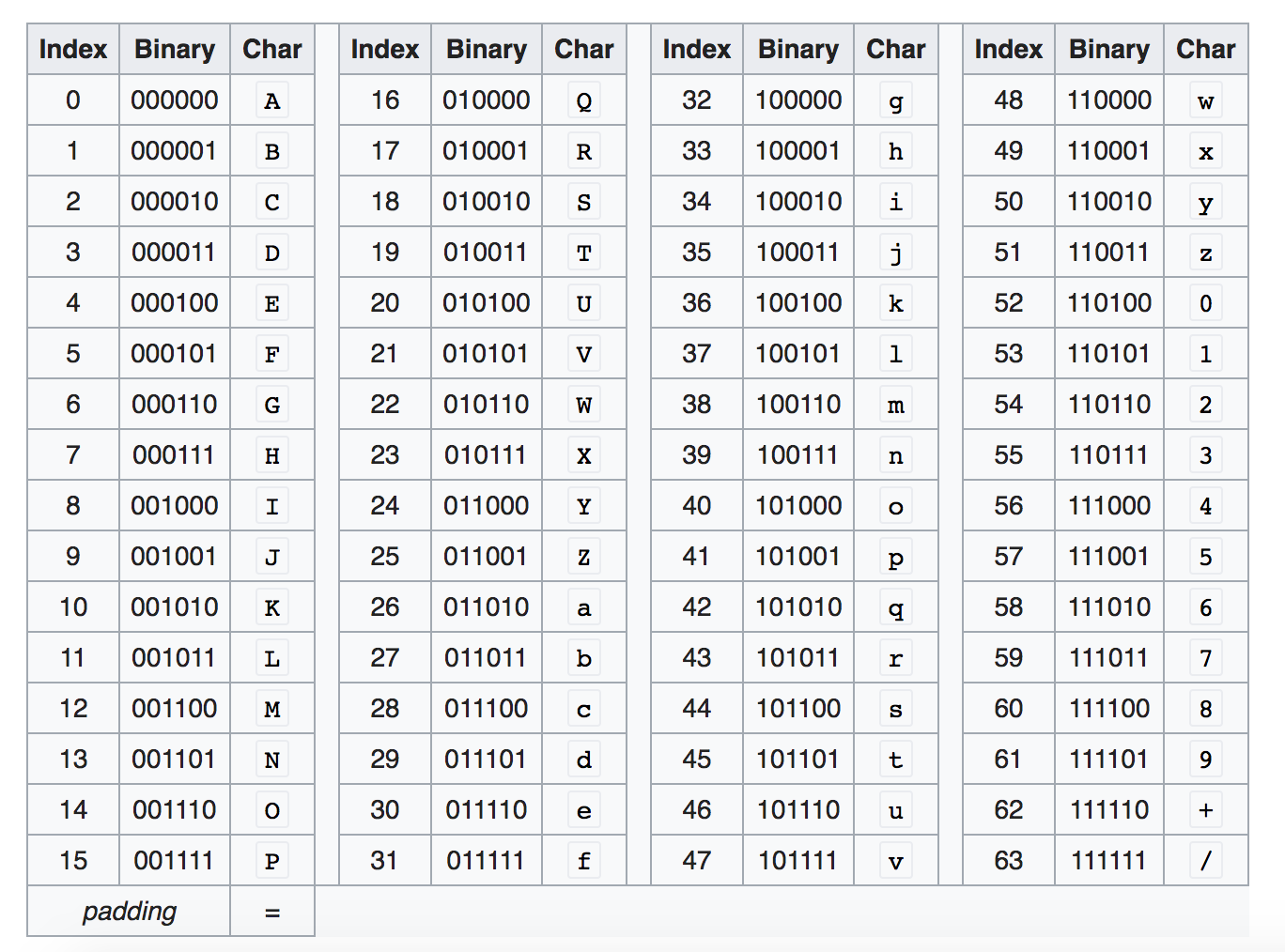
* Хеш файлов [**hello.exe**](https://www.mscs.dal.ca/~selinger/md5collision/hello.exe) и [**erase.exe**](https://www.mscs.dal.ca/~selinger/md5collision/erase.exe) полностью совпадает, однако при запуске этих файлов мы видим разные данные на выходе.





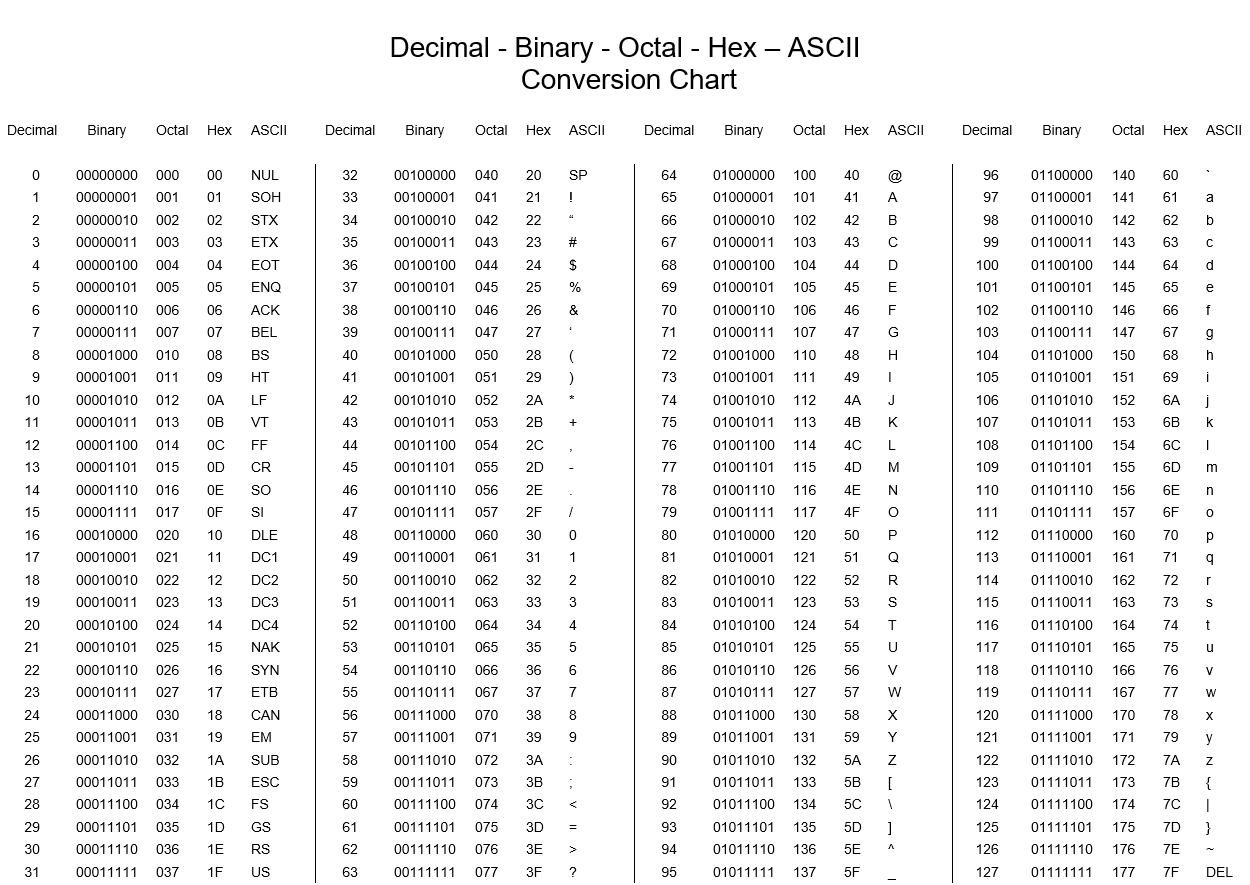
## Кодирование

* Второй объект исследования – **кодирование**. Если вы знаете больше одного языка или **системы счисления**, то он вам обязательно уже знаком. Фактически, кодирование – процесс ,,перевода’’ данных по заданной схеме для удобства в передаче/хранении/автоматической переработке. Это **обратимый** процесс, поэтому для защиты данных его использовать **не стоит**. Один из наиболее часто-используемых стандартов кодирования – **Base64**.
* **Base64** - схема по переводу из двоичной системы в символы **ASCII**. Она разработана для передачи данных, изначально находящихся в **двоичной** системе по каналам, имеющим **ограниченную** поддержку различных типов данных.
* Работает **Base64** достаточно просто: он разделяет двоичные данные на **6-битные** сегменты и представляет их в виде печатных символов **ASCII**. Происходит этот процесс в два этапа:
* **Первый этап** - ,,разбивание’’ **входных данных (байтов)** на **6-битные** блоки. Используется лишь **6 битов** ведь этого достаточно для обеспечения возможности печати и читабельности данных на выходе для человека (**2^6 = 64**, этого достаточно чтобы отобразить все строчные и заглавные буквы английского алфавита, цифры и знаки + и /, также используется 65 символ =, он используется когда последний сегмент двоичных данных не содержит в себе полных 6 битов)
* Затем идет перевод по схеме:



## Опыт

* Представим, что мы хотим отправить какому-либо интернет сервису, который принимает только ASCII символы свои данные. Для этого нам понадобится закодировать наши данные. Для этого мы используем Base64. Рассмотрим этот алгоритм шаг за шагом.
* Наше сообщение – например, ''hello, world!” сначала нужно будет перевести в двоичную систему.
* Символ “h” переведётся как 01101000 “e” как 01100101 и так далее.



* Затем, полученный результат (01101000 01100101 01101100 01101100 01101111 00101100 00100000 01110111 01101111 01110010 01101100 01100100 00100001) разобьём на сегменты из 6 битов каждый. Получаем список из 17 полных сегментов, последний же сегмент состоит лишь из 2 битов (['011010', '000110', '010101', '101100', '011011', '000110', '111100', '101100', '001000', '000111', '011101', '101111', '011100', '100110', '110001', '100100', '001000', **'01'**])
* Теперь, переведём каждый из получившихся сегментов по [схеме](#Схема6BitASCII): первый сегмент переведётся как a, второй – как G и так далее. Но что же делать с последним сегментом? Сначала разберёмся с тем, какая возможная длина может быть у этого сегмента. Она равна остатку от деления нашего исходного количества битов на 6 (размер сегмента). Так как наши входные данные всегда измеряются в байтах, то возможные длины последнего сегмента – 2 и 4, тогда для дополнения до полного (6-битного) сегмента нам потребуется добавить либо 4, либо 2 бита. **НОД** от 4 и 2 равен 2, тогда мы можем использовать один из символов (=) как 2 пустых бита. Нам не хватает 4 бита, поэтому добавим два знака = в конец. Полученный результат - aGVsbG8sIHdvcmxkIQ==. (Часть Q==　в конце образовалась благодаря тому, что наш последний сегмент = 01, для того, чтобы он стал полным 6 битным сегментом нам нужно добавить к нему 4 бита, получаем 010000, что по [схеме](#Схема6BitASCII) переводится как Q.
* Перевод из Base64 в ASCII символы очень прост – нужно лишь перевести каждый из символов в 6-битные сегменты, затем соединить их, разделить на 8-битные и перевести по [схеме](#Схема8BitASCII).

# Шифрование

* Третий объект исследования – **шифрование**. Шифрование – **обратимый** процесс преобразования информации для **защиты** от злоумышленников, но при этом предоставляющий доверенным лицам доступ. Существует два типа шифрования – **симметричный** и **ассиметричный**. Они различаются тем, что в симметричном используется один и тот-же ключ как для зашифровки, так и для расшифровки, в ассиметричном же они отличаются (при этом ключ для зашифровки является открытым, а для расшифровки – закрытым).
* Каждый из этих типов обладает как своими достоинствами, так и недостатками. Выбор применяемого метода зависит от целей.
* Например, когда данные отправляются к вебсайту по HTTPS протоколу, они зашифровываются по открытому ключу.
* Один из примеров – стандарт шифрования AES-256. Это симметричный алгоритм шифрования с 2^256 возможными вариантами ключей (**~1.15\*10^77**).
* Разберёмся, как работает этот стандарт шаг за шагом.

## AES

* Для того, чтобы понять, как устроен алгоритм AES нам сначала необходимо понять, как он передаёт информацию между каждым из шагов. Каждый блок состоит из 16 байтов, хранящихся в матрице 4 на 4, в которой каждая клетка хранит 1 байт информации.
* Матрица также известна как state array (массив состояний). Первый шаг алгоритма – добавить раундовый ключ. Затем, мы используем функцию XOR с нашим изначальным ключом как вторым аргументом на каждую из клеток матрицы.
* Второй шаг – так называемые sub-bytes (под-байты). На этом этапе мы конвертируем каждый байт массива состояний в шестнадцатеричную систему счисления, делёную на две равные части – строки и столбцы, затем сопоставляемые с полем подстановки для создания новых значений для конечного массива состояний.

Следующий этап – смещение рядов. Он пропускает первый ряд, затем перемещает все элементы второго ряда на одну позицию влево, элементы третьего ряда на 2 позиции влево, четвёртого на 3.

## Опыт

* Описанный процесс является достаточно сложным для понимания, поэтому лучше увидеть его в действии. Например, мы хотим зашифровать сообщение Testing message с помощью ключа Secret key examp. Как будет проходить данный процесс?
* Сначала нам необходимо преобразовать наши входные данные в шестнадцатеричную систему – получаем, что наше сообщение - 54 65 73 74 69 6e 67 20 6d 65 73 73 61 67 65, а ключ – 53 65 63 72 65 74 20 6b 65 79 20 65 78 61 6d 70.
* Сначала выполнится операция XOR над нашим текстом, используя ключ как второй аргумент к функции, затем выполнится сдвиг ячеек в рядах, потом полученная матрица умножится на матрицу-константу и такой цикл будет проходить, пока не дойдёт до финального раунда. В конце мы получим результат - A7352651D65E2E066EC1744AE018FFD0.

# Глава II

## Продукт

* Программа на языке программирования python (version 3.11), использующая модули hashlib, pyperclip, cryptography, string и random. для хеширования/кодирования (AES-256-CBC) данных пользователя. ([Github link](https://github.com/NikitaNightBot/InfProject/blob/main/Project.py))

## Вывод

* Каждый из описанных видов шифрования информации имеет как свои плюсы, так и минусы, именно поэтому криптография – наука, которая постоянно развивается, а специалисты-криптографы востребованы во всех компаниях, желающих защитить как свои данные, так и данные своих пользователей.

**Список использованной литературы**

* Грэм Дэниел Г. – Этичный хакинг. Практическое руководство по взлому.