Хэширование паролей

Что это такое?

Вопрос защиты паролей пользователей является важным при проектировании системы, работающей с учетными записями.

Надежным способом защиты паролей является «соленое» хеширование пароля.

Хеш-алгоритмы являются **односторонними** функциями. Они превращают любое количество данных в «дактилоскопический отпечаток» фиксированной длины, который **не может быть обратим** (в отличие от шифрования, где при наличии ключа возможно обратное преобразование).

Хэширование паролей

Особенности хэширования

Также алгоритмы хэширования имеют такое свойство, что если входное значение изменяется даже совсем немного, полученный хеш-код полностью отличается от исходного:

```
hash("hello") = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
hash("hbllo") = 58756879c05c68dfac9866712fad6a93f8146f337a69afe7dd238f3364946366
hash("waltz") = c0e81794384491161f1777c232bc6bd9ec38f616560b120fda8e90f383853542
```

Это отлично подходит для защиты паролей, потому что мы хотим сохранить пароли в зашифрованном виде, невозможном для расшифровки (даже для самого администратора), и в то же время мы должны быть способны проверить то, что пароль пользователя корректен.

Хэширование паролей

Особенности хэширования

Следует отметить, что хеш-функции, используемые для защиты паролей, это не те же хеш-функции, которые относятся к структурам данных. Хеш-функции, используемые для реализации структур данных, таких как хеш-таблицы, разрабатываются для того, чтобы быть быстрыми, а не безопасными.

Только криптографические хеш-функции могут быть использованы для реализации хеширования паролей. Такие функции, как SHA256, SHA512, RipeMD и WHIRLPOOL И ДРУГИЕ являются криптографическими хеш-функциями.

Хэширование паролей

Схема хэширования без соли

Общая схема процесса регистрации аккаунта и аутентификации в системе учетных записей, основанной на хешировании (БЕЗ СОЛИ), выглядит следующим образом:

1.Пользователь создает учетную запись (логин+пароль);

2.Вычисляется хеш-код от пароля и сохраняется в базе данных вместе с логином.

3.Когда пользователь пытается войти в систему, логин и хеш-код, вычисленный от пароля, который он ввел, сравнивается с логинами и хеш-кодами реальных паролей из базы данных;

4.Если хеш-коды совпадают, пользователю предоставляется доступ. Если нет – пользователю сообщается, что он ввел неверные данные для входа в учетную запись.

Хэширование паролей

Важно знать!

Никогда не сообщайте пользователю, что именно не так: имя пользователя или пароль.

Всегда отображайте только общее сообщение, например, «**Неверное имя пользователя или пароль**».

Это не позволит злоумышленникам вычислить верные имена пользователей без знания их паролей.

Хэширование паролей

И все равно по хэшу подбирают пароль(

Несмотря на наличие алгоритмов хэширования есть много видов атак, позволяющих очень быстро восстановить пароли из простых хеш-кодов (по сути – ПОДОБРАТЬ к хэшу исходный пароль из уже сгенерированных табличных данных).

Как взламывается хеш

Самый простой способ взломать хеш-код – это попробовать подобрать пароль, вычисляя хеш-код для каждого предположения и проверяя, совпадает ли этот хеш-код со взламываемым.

Если хеш-коды одинаковые, то предположенный вариант является паролем. Двумя наиболее распространенными способами угадывания паролей является атаки по словарю и атаки полным перебором.

Хэширование паролей

И все равно по хэшу подбирают пароль(

Атаки по словарю:

```
1  Trying apple        : fail
2  Trying blueberry    : fail
3  Trying 1234567      : fail
4  ......
5  Trying letmein      : fail
6  Trying qwerty       : Success!
```

Атаки полным перебором:

```
1  Trying aaaa         : fail
2  Trying aaab         : fail
3  Trying aaac         : fail
4  Trying aaad         : fail
5  ......
6  Trying acdb         : fail
7  Trying acdc         : Success!
```

Хэширование паролей

crackstation.net

Сегодня существуют сгенерированные таблицы поиска, содержащие вычисленные хеш-коды паролей из словаря паролей для наиболее распространенных алгоритмов хэширования.

Умело реализованные таблицы поиска могут обрабатывать сотни поисков хеш-кодов в секунду, даже если они содержат много миллиардов хеш-кодов.

Пример онлайн-ресурса, определяющего пароль по хэшу (!без соли):

crackstation.net

Ресурс безопасный и бесплатный, но сложный пароль по хэшу конкретно этот ресурс не восстановит

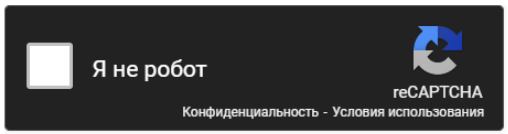
Хэширование паролей

crackstation.net

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

```
65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5
```



Crack Hashes

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5	sha256	qwerty

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

https://tools.seo-zona.ru/md5-sha1-hashing.html

Хэширование паролей

Предварительно сгенерировать хэш для строки, используя какой-либо распространенный алгоритм хэширования, можно здесь:

https://tools.seo-zona.ru/md5-sha1-hashing.html

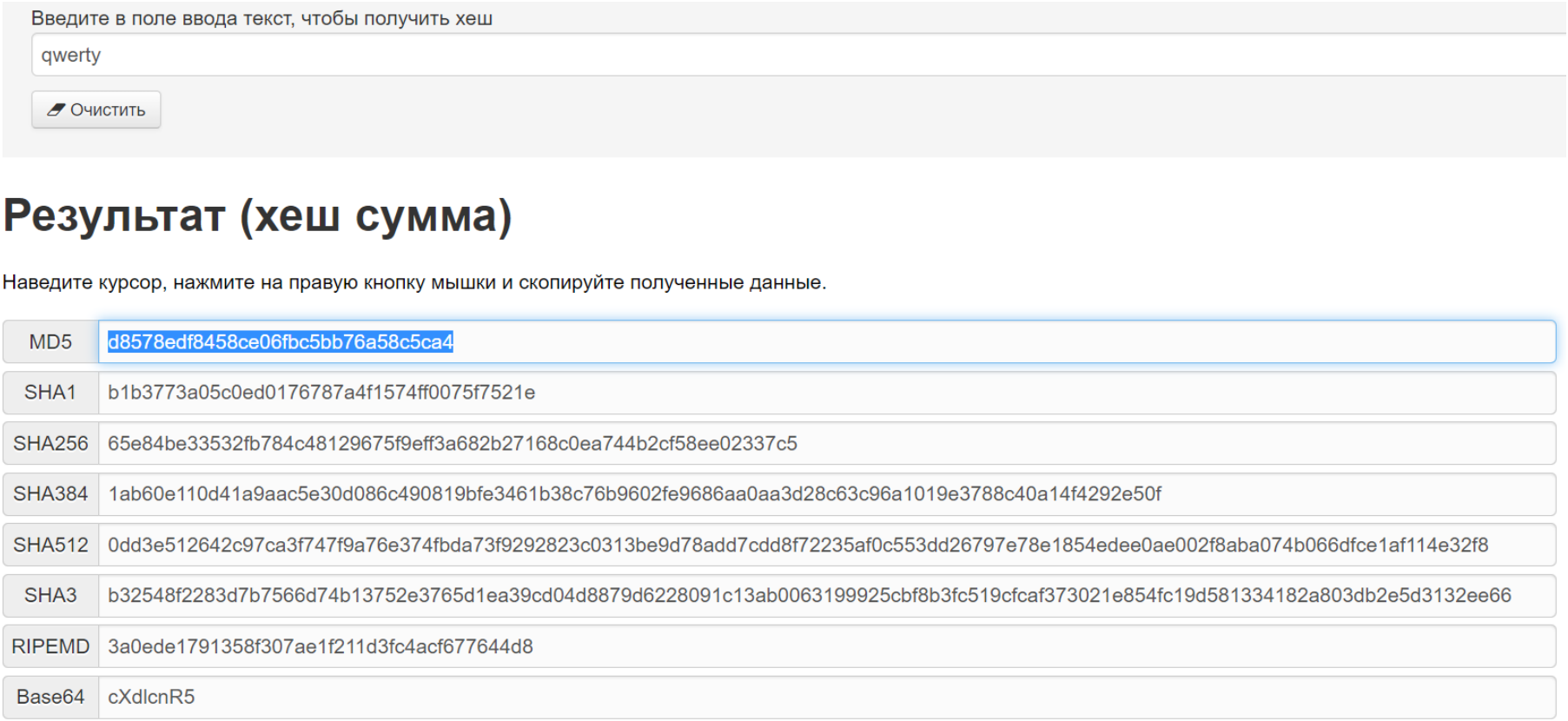
или здесь:

Это тоже безопасно и бесплатно

http://www.hashemall.com/

https://tools.seo-zona.ru/md5-sha1-hashing.html

Хэширование паролей



Хэширование паролей

СОЛЕНОЕ хэширование

Заранее сгенерированные таблицы не помогут, если применять СОЛЕНОЕ хэширование.

Мы можем добавить в конец пароля случайную строку, называемую «соль», перед операцией хеширования.

При этом, как показано в примере ниже, всякий раз получаются совершенно разные хэши из одного и того же пароля, но с добавлением различной соли:

```
hash("hello") = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
hash("hello" + "QxLUF1bgIAdeQX") = 9e209040c863f84a31e719795b2577523954739fe5ed3b58a75cff2127075ed1
hash("hello" + "bv5PehSMfV11Cd") = d1d3ec2e6f20fd420d50e2642992841d8338a314b8ea157c9e18477aaef226ab
hash("hello" + "YYLmfY6IehjZMQ") = a49670c3c18b9e079b9cfaf51634f563dc8ae3070db2c4a8544305df1b60f007
```

Злоумышленник не знает заранее, какая будет соль, поэтому он не может предварительно вычислить таблицу поиска или радужную таблицу.

Хэширование паролей

СОЛЕНОЕ хэширование

Чтобы проверить, корректен ли пароль, нам нужна соль, поэтому ее сохраняют в базе данных пользовательских аккаунтов вместе с логином и хеш-кодом.

Для каждого аккаунта генерируется своя случайная соль! Чтобы подобрать пароль, злоумышленнику (при условии получения доступа к базе данных) придется взять соль и добавить ее к распространенным/всем сгенерированным паролям (удовлетворяющим требованиям длины/допустимых символов конкретного ресурса), захэшировать эти данные (счет идет на миллиарды) с применением существующих алгоритмов и найти совпадение. И так для каждого интересующего аккаунта.

А это потребует очень много времени (хотя и возможно).

Хэширование паролей

Усложняем СОЛЕНОЕ хэширование

Чтобы добавить еще одну степень защиты и усложнить попытки злоумышленника подобрать пароль, применяют усложненные алгоритмы хэширования, например:

hash_pass_with_salt = sha256(sha256(**pass + salt**));

hash_pass_with_salt = sha256(sha256(**pass + salt**)+sha256(pass));

и другие в таком же духе.

Смысл усложнения заключается в том, что злоумышленнику нужно получить доступ к коду, прежде чем генерировать хэши для известных паролей по вашему алгоритму!

Хэширование паролей

Усложняем СОЛЕНОЕ хэширование

При создании авторского алгоритма с различными комбинациями хэширования не рекомендуется совмещать разные хэш-функци в одной (например, md5 и sha256 и т.д.).

Также не рекомендуется слишком усердствовать в усложнении алгоритма хэширования (не впадать в крайности).

Схема хэширования С СОЛЬЮ

Общая схема процесса регистрации аккаунта и аутентификации в системе учетных записей, основанной на хешировании С СОЛЬЮ выглядит следующим образом.

Регистрация пользователя:

1.Пользователь создает учетную запись (логин+пароль);

2.Генерируется случайная строка фиксированной длины (минимум 10 символов).

3.Вычисляется хеш-код по задуманному разработчиком алгоритму (в простейшем варианте – от комбинации пароль+соль, например sha256(**pass + salt**)).

4.В базе данных сохраняется логин, хэш-код, соль.

Хэширование паролей

Схема хэширования С СОЛЬЮ

Аутентификация пользователя:

1.Находим в базе данных совпадение по введенному логину (вот поэтому вводимые на этапе регистрации логины должны проходить проверку на уникальность!!!)

2.Для совпавшей по логину учетной записи из базы данных извлекаем соль и хеш-код пользователя

3.Добавляем соль к введенному паролю и вычисляем хеш-код с помощью той же самой функции (см. регистрация, шаг 3)

4.Сравниваем сгенерированный хеш-код для введенного пароля и соли с хеш-кодом из базы данных. Если они совпадают, пускаем пользователя в систему. В противном случае, «такой пользователь не найден».

Хэширование паролей

https://bit.ly/3fsc7ED

Прочитать про хэширование дополнительно можно здесь:

https://www.internet-technologies.ru/articles/solenoe-heshirovanie-paroley-delaem-pravilno.html

Короткая ссылка:

https://bit.ly/3fsc7ED

Хэширование паролей

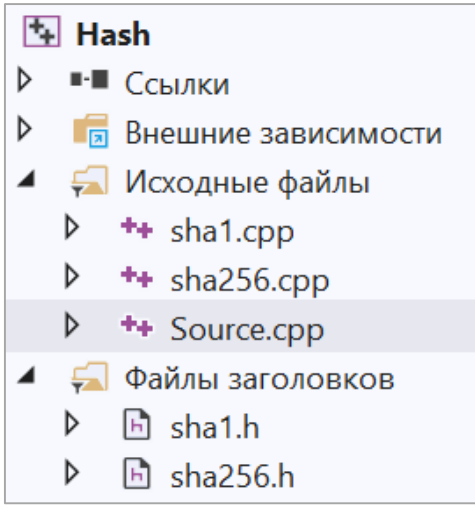
https://bit.ly/3tU52jV

Найти реализацию хэш-функций на C++ можно здесь:

http://www.zedwood.com/article/cpp-sha256-function

Короткая ссылка:

https://bit.ly/3tU52jV



Код проекта с интегрированными функциями SHA1 и SHA256 размещен на гугл-диске!!! Вы можете выбрать один из алгоритмов (лучше SHA256) и применить в своем проекте)

Хэширование паролей

Пример генерации соли и хэширования

sha1.h | sha256.cpp | sha256.h | sha1.cpp | Source.cpp*

Hash | (Глобальная область) | main()

```
1   #include <iostream>
2   #include <cstdlib>
3   #include <ctime>
4   #include <string>
5   #include "sha1.h"
6   #include "sha256.h"
7
8   using namespace std;
9
10  const int SALT_SIZE = 16; // длина соли
11  const int SYMBOLS_SIZE = 62; // набор символов, из которых генерируется соль
12
13  string generateSalt(int salt_size); // функция генерации соли
14
15  string getSymbolsForSalt(); // функция формирования набора символов
16  // вида: aA0bB1cC2dD3eE4fF5gG6hH7iI8jJ9kKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ
```

Хэширование паролей

Пример генерации соли и хэширования

```
18  int main()
19  {
20      setlocale(LC_ALL, "RUS");
21
22      string pass = "grape"; // для примера
23
24      cout << "sha1: " << sha1(pass) << endl;
25
26      cout << "sha256: " << sha256(pass) << endl;
27
28      string salt = generateSalt(SALT_SIZE); // генерируем соль
29      cout << "salt: " << salt << endl;
30
31      string hash_pass_with_salt = sha256(pass + salt); // простейшее хэширование с солью
32      cout << "hash_pass_with_salt: " << hash_pass_with_salt << endl;
33
34      hash_pass_with_salt = sha256(sha256(pass + salt)+sha256(pass)); // усложненный вариант (рекомендуется)
35      cout << "hash_pass_with_salt: " << hash_pass_with_salt << endl;
36
37      system("pause");
38  }
```

Хэширование паролей

Пример генерации соли и хэширования

```
40  string generateSalt(int salt_size)
41  {
42      string symbols = getSymbolsForSalt();
43      cout << symbols << endl;
44
45      srand(time(NULL));
46
47      string salt;
48      salt.reserve(salt_size);
49
50      for (int i = 0; i < salt_size; i++)
51      {
52          salt.push_back(symbols[rand() % SYMBOLS_SIZE]);
53      }
54
55      return salt;
56  }
```

Хэширование паролей

Пример генерации соли и хэширования

```
59  string getSymbolsForSalt()
60  {
61      string symbols;
62      symbols.reserve(SYMBOLS_SIZE);
63
64      char little_letter = 'a';
65      char big_letter = 'A';
66      char number = '0';
67      int i = 0;
68      for (int k = 0; k < 26; k++)
69      {
70          symbols.push_back(little_letter++);
71          symbols.push_back(big_letter++);
72          if (k < 10) symbols.push_back(number++);
73      }
74
75      return symbols;
76  }
```

Хэширование паролей

Пример генерации соли и хэширования

```
sha1: bc8a2f8cdedb005b5c787692853709b060db75ff
sha256: 0f78fcc486f5315418fbf095e71c0675ee07d318e5ac4d150050cd8e57966496
aA0bB1cC2dD3eE4fF5gG6hH7iI8jJ9kKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ
salt: SehBvARlru3xqNPU
hash_pass_with_salt: 4af00f78fa64b145e6962e1d35b647430925d6141e5a124659379eb187465628
hash_pass_with_salt: 57839c5baafe715af2d1c5f51561d3706680f7d4d05877af3268ff623d64716d
Для продолжения нажмите любую клавишу . . .
```