Hash-functions, HMAC

Хеш - функции

Хеш- функция

- Хеш- это функция, которая ставит в соответствие небольшой, фиксированного размера объем двоичных данных произвольному, сколь угодно большому объему входных данных. Здесь можно провести аналогию с отпечатком человеческого пальца, который мал по размеру, однако настолько характерен для своего обладателя, что с его помощью можно идентифицировать человека.

  Для любого конкретного хеша крайне трудно подобрать другие входные данные такие, что давали бы в результате точно такой же хеш.

  Криптографические хеш-функции используются в технологии цифровой подписи потому, что они позволяют эффективно выявлять нарушения целостности сообщения. Хеш называют также дайджестом сообщения.

Идеальная хеш – функция должна обладать следующими свойствами:

1. входные данные могут обладать произвольными размерами
2. выходные данные всегда обладают небольшим, фиксированным размером, вытекающим из используемого алгоритма
3. функция быстро вычисляется
4. ее трудно обратить (то есть это односторонняя функция);
5. вероятность возникновения коллизий невелика.

Что такое коллизия?

- Трудно найти два варианта входных данных, которые давали бы на выходе один и тот же хеш. Это может показаться странным на первый взгляд. Если вы можете иметь на входе данные произвольного размера, то число всевозможных вариантов входных данных бесконечно. Если при этом размер вычисляемого хеша фиксирован, то число возможных вариантов выходных данных конечно.
- Таким образом, очевидно, что должно существовать бесконечное число вариантов входных данных, дающих одинаковые хеши. Это похоже на парадокс.

- Однако, несмотря на бесконечное число вариантов, приводящих к коллизии, на самом деле найти хотя бы два варианта, дающих одинаковые хеши, чрезвычайно трудно!!!
  Таким образом, тот факт, что коллизии трудно найти, вытекает не из малого числа существующих коллизий (поскольку это число бесконечно велико). Коллизии трудно находить потому, что число вариантов сообщений, не приводящих к коллизиям, радикально больше.
- ***Безопасность хеша основывается на чрезвычайной трудности нахождения даже всего лишь одной пары вариантов сообщения, приводящей к коллизии!***

Хеш- алгоритмы, поддерживаемые в .NET

- Две наиболее часто используемые криптографические хеш-функции это **SHA1 (Secure Hash AIgorithm, хеширующий криптографический алгоритм**), опубликованный NIST в середине 1990х, **и MD5 (Message Digest, дайджест сообщения),** разработанный Р. Ривестом в начале 1990х. В добавление к этому было опубликовано несколько новых версий SHA. Также для целей, связанных с аутентификацией сообщений, важную роль играет алгоритм ключевого хеша. Все упомянутые алгоритмы поддерживаются в .NET Framework в форме классов, производных от Hash AIgorithm:

- **MD5;**
- **SHA1;**
- **SHA256;**
- **SHA384;**
- **SHA512;**
- **KeyedHashAlgorithm.**

Иерархия классов хеш-алгритмов

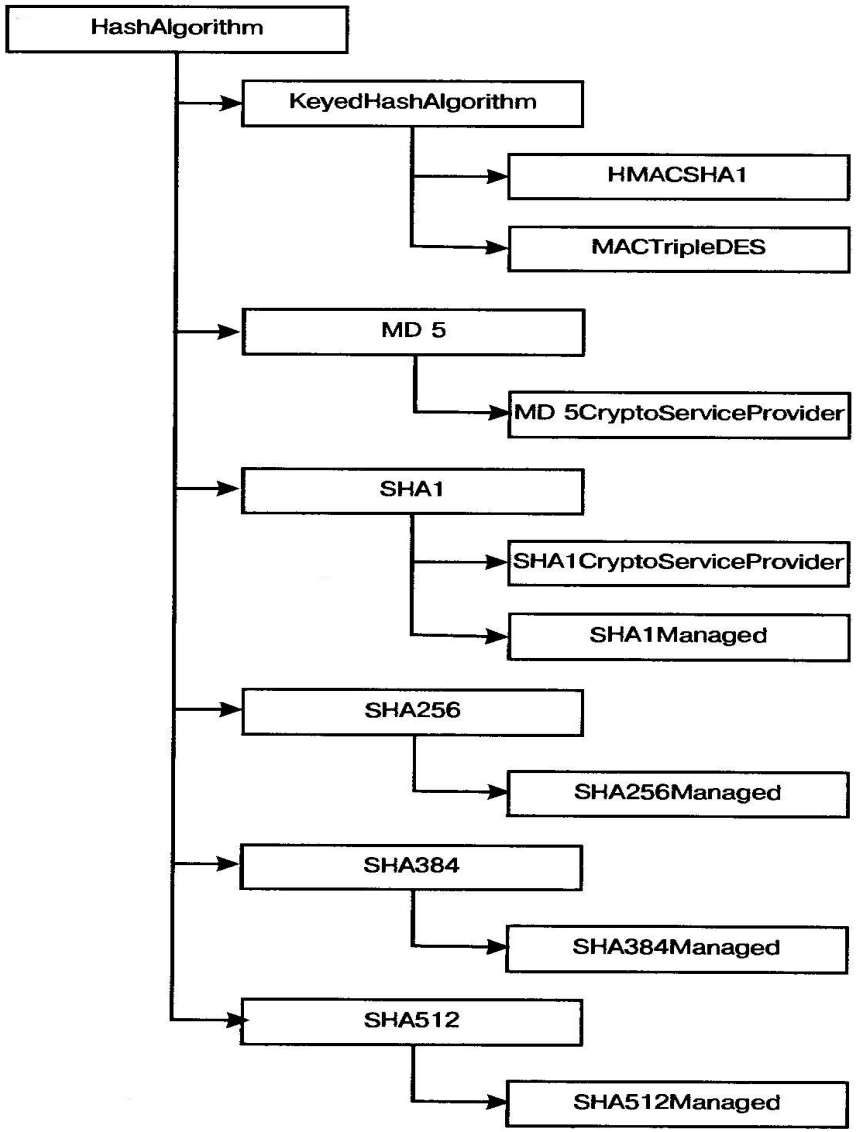


Рис. 5.1. Иерархия хеш-алгоритмов

Иерархию классов возглавляет абстрактный класс – HashAlgorithm, производный от класса Object. Производные абстрактные классы KeyedHashAlgorithm , MD5, SHAl, SHA256,SHA384, SHA512 представляют собой часто используемые криптографические алгоритмы

Класс HMACSHA1 и MACTripleDES

- Класс HMACSHA1 производит ключевой хеш KeyedHashessageAuthentication Code - код аутентификации сообщения при помощи ключевого хеша или HMAC) при помощи хеш-функции SHA-1.

  Класс MACTripleDES производит ключевой хеш HMAC при помощи шифрования TripleDES1, использованного в качестве хеш-функции.

- Ключевой хеш HMAC похож на цифровую подпись в том смысле, что его тоже можно использовать для верификации аутентичности и целостности сообщения, однако его отличие заключается в том, что для подтверждения обязательств его использовать нельзя.

Класс HashAlgorithm

- Класс HashAlgorithm обладает публичным свойством Hash, которое предоставляет собой байтовый массив, содержащий в себе вычисленный хеш.
- Публичное свойство HashSize содержит значение размера хеша в битах.
- Наиболее важный публичный метод класса HashAIgorithm – ComputeHash, возвращающий значение хеша в виде байтового массива, вычисленное для заданных во входном параметре байтовом массиве входных данных. В следующем коде иллюстрируется применение класса HashAIgorithm на примере конкретного производного класса, инкапсулирующего алгоритм SHAI. Предполагается, что входные данные уже существуют в форме байтового массива messageByteArray.

```
HashAlgorithm shal = new SHA1CryptoServiceProvider();
byte[] shalHash = shal.СотputeHash(messageByteArray);
```

MD5 и SHA

- Класс MD5 инкапсулирует алгоритм MD5, который производит 128- битовый хеш из входных данных произвольного размера.
- Этот алгоритм первоначально предназначался для приложений цифровой подписи, в которых хеш шифруется секретным ключом RSA.
- MD5 является расширением алгоритма MD4(Алгоритм Ривеста), при этом, работая немного медленнее, он обеспечивает более высокий уровень безопасности.
- Класс SHA1 инкапсулирует алгоритм SHA-1, который обрабатывает входные данные размера не более 264 бит, производя при этом хеш размером 160 бит.
- Алгоритм SHA-был принят в качестве стандарта NIST под названием SHS(Secure Hash Standart, стандарт безопасного хеша).
- В документе FIPS PUB 180-13 утверждается, что он основан на принципах алгоритма MD4. Таким образом, часто используемые алгоритмы MD5 и SHA-1 схожи друг с другом.

SHA256, SHA384 и SHA512

- Классы SHA256, SHA384 и SHA512 инкапсулируют набор родственных хеш-алгоритмов, которые производят хеши размером 256,384 и 512 бит .
- Эти новые алгоритмы, определенные документом FIPS PUB 180-21 в 2002 году, ведут свое происхождение от SHA-1. Они были созданы, поскольку со времени первоначальной публикации SHA-1 в 1993 году возникли опасения, что размер хеша в 160 бит не обеспечивает высоконадежной защиты на длительную перспективу. В то время как атака методом «грубой силы. на n-битовый симметричный алгоритм измеряется рабочим фактором, пропорциональным 2 в степени n, такая же атака на n-битовый криптографический хеш, измеряется рабочим фактором, пропорциональным 2 в степени n/2.

- Причина кроется в том, что атака «по дню рождения(birthday attack) на хеш считается успешной, если найдены всего два варианта входных данных, дающих одинаковые хеши. Следовательно, рабочий фактор атаки на 160- битовый алгоритм SHA-1 пропорционален, практически, всего лишь 280. Это достаточно большое число, и такая атака, вероятно, недоступна почти для всех современных противников, но в расчете на длительный период этот уровень безопасности, вероятно, уже недостаточен.

Класс KeyedHashAlgorithm

- Представляет собой хеш функцию, которая вычисляется не только с помощью входных данных, но еще и с помощью дополнительной информации(ключа). Алгоритм ключевого хеша представляет собой зависящую от ключа одностороннюю хеш-функцию. Такой механизм удобен для аутентификации сообщения, так как создать и верифицировать такой хеш может лишь владелец ключа.
- Алгоритм ключевого хеша обеспечивает одновременно контроль целостности и аутентификацию между сторонами, которые заранее обменялись ключами.

Пример реализации использования алгоритмов хеширования MD5 и Sha256 на языке программирования Python.

Programiz

Python Online Compiler

Learn Python App

main.py

Run

```python
import hashlib
text = str(input("Введите любой текст для шифрования "))
hash_md5 = hashlib.md5( text.strip().encode( 'utf-8'))
hash_sha256 = hashlib.sha256( text.strip().encode( 'utf-8'))
print("Ваш текст " + text + " в md5 " + hash_md5.hexdigest())
print("Ваш текст " + text + " в sha256 " + hash_sha256.hexdigest())
```

Shell

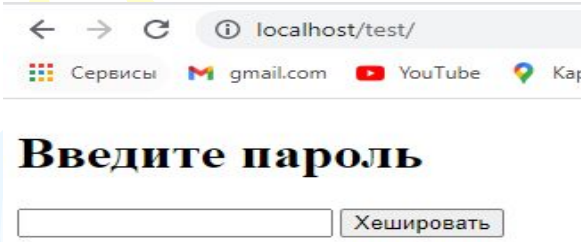
Clear

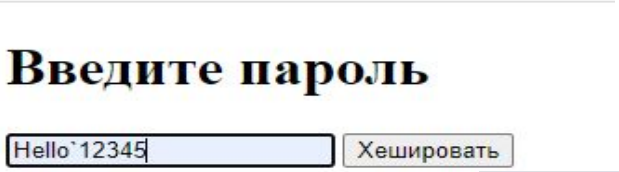
```
Введите любой текст для шифрования hash function
Ваш текст hash function в md5 0225c556a1d7ac76771a75e96c677a0d
Ваш текст hash function в sha256
    04ce4af53ae1b0f46451715ffc43092eab5b23330a6584462d0723732b147c4b
>
```

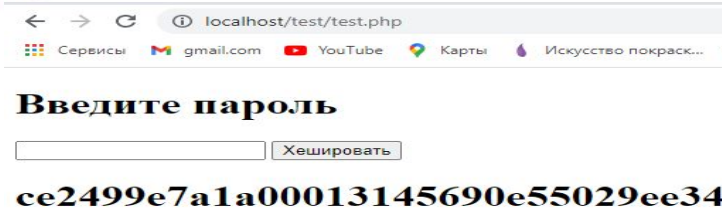
- Основным способом, гарантирующим безопасность хеша пароля, является использование «соли». Он основан на добавлении к паролю нескольких случайных символов и последующем хешировании результата.
- Для примера рассмотрим создание хеш-код MD5 в PHP использую несколько функций:

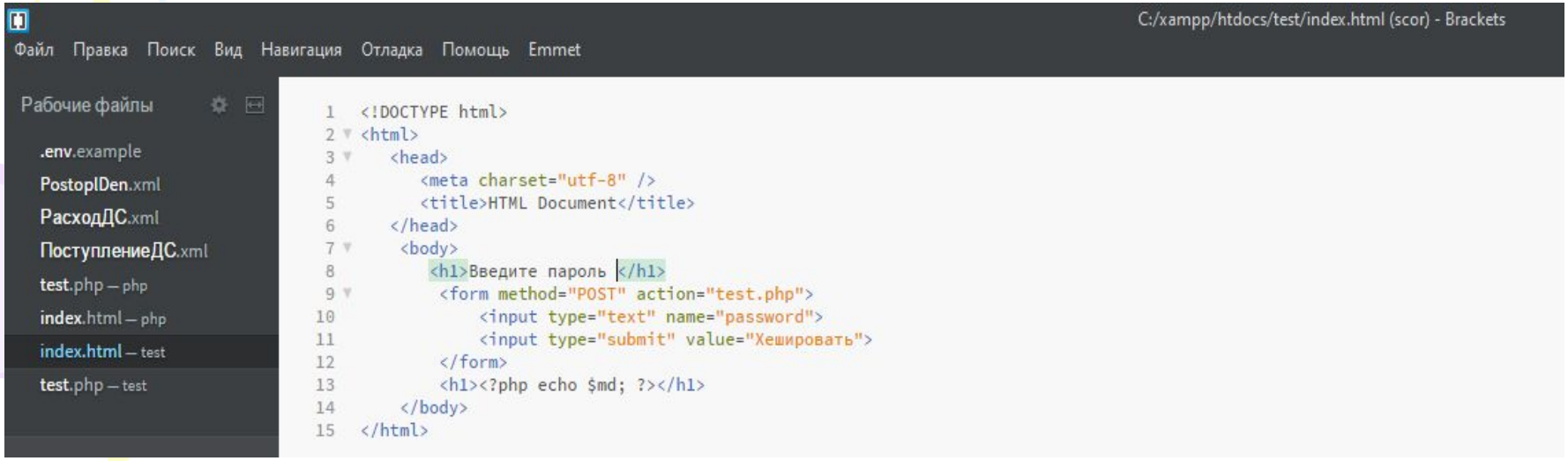
md5() – в качестве одного из параметров принимает значение «соли»;

crypt() – в отличие от предыдущей эта функция полностью автоматизирует весь процесс, в том числе и генерирование значения соли.









C:/xampp/htdocs/test/test.php (scor) - Brackets

Файл Правка Поиск Вид Навигация Отладка Помощь Emmet

Рабочие файлы

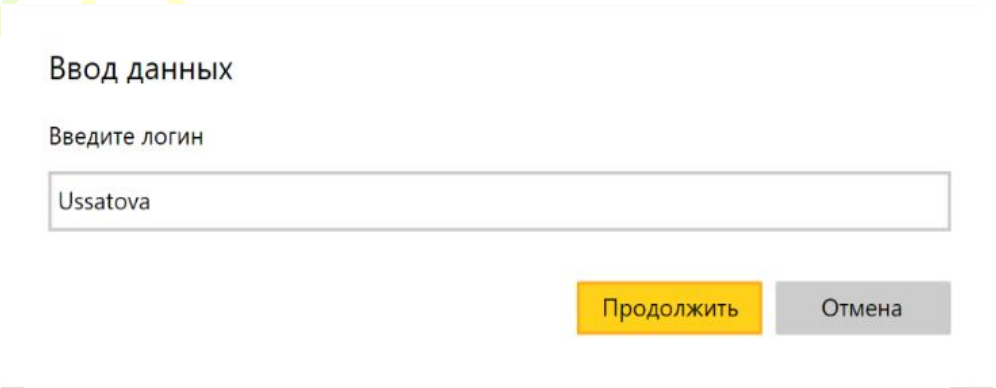
- .env.example
- PostopIDen.xml
- РасходДС.xml
- ПоступлениеДС.xml
- test.php – php
- index.html – php
- index.html – test
- test.php – test

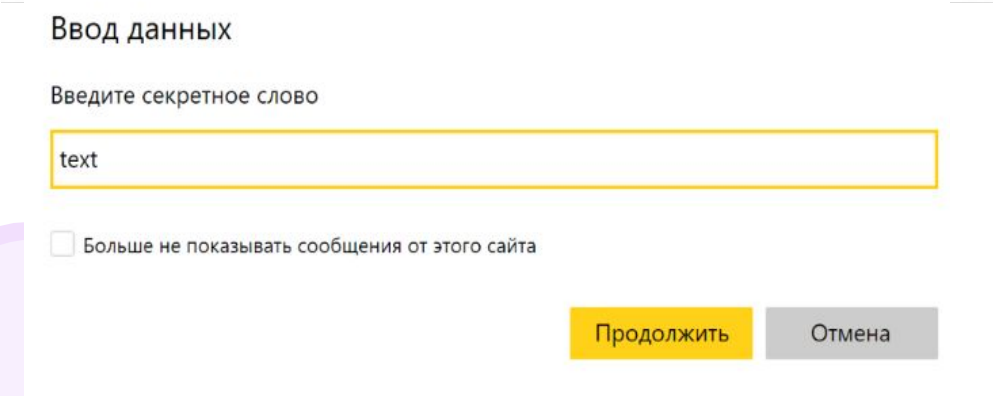
scor

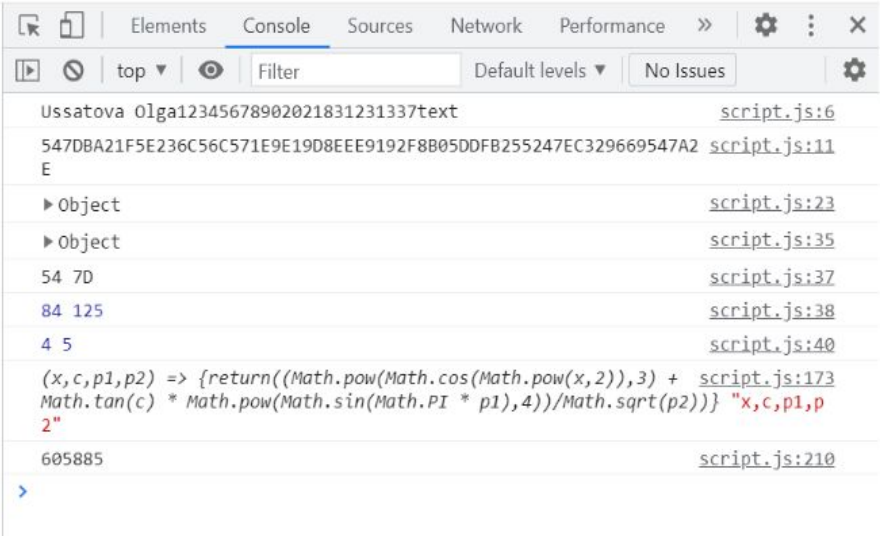
app

```php
<?php
if ($_POST["password"]) {
$pass = $_POST["password"];
function salt() {
    $s = '';
    $length = rand(7,12); // длина соли
    for($i=0; $i<$length; $i++) {
        $s .= chr(rand(33,126));//случайный символ из таблицы ASCII
    }
    return $s;
}
$md = md5(md5($pass).salt());
}
else {
  $md = "Вы не ввели пароль";
}
require_once('index.html');
?>
```

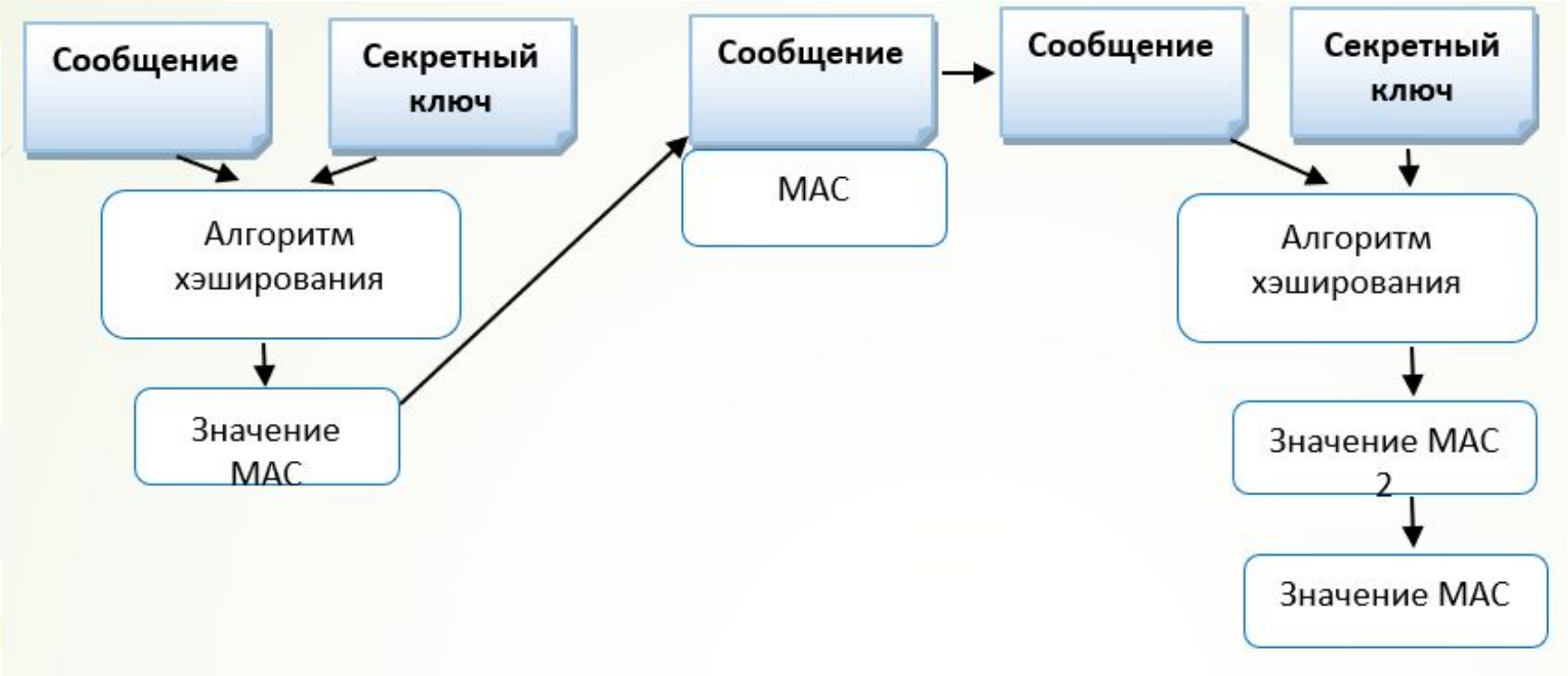
- Рассмотрим пример SHA256 в алгоритме двухфакторной аутентификации. Выбор тригонометрической функции для вычисления одноразового пароля осуществляется в соответствии с результатом полученной хеш–функции стандартов SHA256, где используются первые символы, которые будут являться индексами в таблице размерностью 256x256. По данному индексу будет выбрана функция и определены её параметры. По итогам вычисления в качестве одноразового временного пароля берутся цифры после запятой, начиная с 5 – й позиции и длиною в 6 цифр. Полученное число и будет временным паролем, которое необходимо ввести в приложение.







РАБОТА ФУНКЦИИ HMAC



- Функции MAC аналогичны криптографическим функциям hash, они удовлетворяют различными требованиями к безопасности. Несмотря на их сходство, реализуются они по-разному: обычно алгоритм генерации MAC основан на алгоритме генерации кода hash с расширением, которое использует закрытый ключ.
- Основное различие заключается в концептуальности: хэши используются для гарантии целостности данных, а MAC гарантирует аутентификацию целостности. Это означает, что хэш-код генерируется из сообщения без какого-либо внешнего ввода: то, что вы получаете, — это то, что можно использовать для проверки того, изменилось ли сообщение во время его перемещения. Вместо этого MAC использует закрытый ключ в качестве исходного кода для функции hash, которую он использует при генерации кода: это должно гарантировать получателю, что не только сообщение не было изменено, но и кто его отправил, это тот, которого мы ожидали: в противном случае злоумышленник не может знать закрытый ключ, используемый для генерации кода. HMAC - это код аутентификации сообщения на основе hash. Обычно это включает применение функции hash один или несколько раз к какой-либо комбинации общего секрета и сообщения.