CTF на Физтехе

Занятие 6

Хеширование

Примеры хеш-функций

- `1f3870be274f6c49b3e31a0c6728957f`
- `d0be2dc421be4fcd0172e5afceea3970e2f3d940`
- `$2a$10$/bWghfYQY6zG5GQLOtl5VuqYn.P0WwL3Q6sgmCu84fjzn2FGFbiUa`

Примеры хеш-функций

- `1f3870be274f6c49b3e31a0c6728957f (MD5)`
- `d0be2dc421be4fcd0172e5afceea3970e2f3d940 (SHA-1)`
- `$2a$10$/bWghfYQY6zG5GQLOtl5VuqYn.P0WwL3Q6sgmCu84fjzn2FGFbiUa (bcrypt)`

Криптографическая хеш-функция

- H(message) = hash
  - MD5(“Hello world!”) = 86fb269d190d2c85f6e0468ceca42a20
  - SHA-1(...) = d3486ae9136e7856bc42212385ea797094475802

- Свойства:
  - Стойкость к восстановлению прообраза
  - Стойкость к коллизиям первого рода
  - Стойкость к коллизиям второго рода

Коллизии

- Стойкость к коллизиям первого рода
  - для заданного сообщения M должно быть вычислительно невозможно подобрать сообщение N, для которого H(N) = H(M)

- Стойкость к коллизиям второго рода
  - должно быть вычислительно невозможно подобрать пару сообщений (M, N), имеющих одинаковый хеш

Применения

- Хеш-таблицы
- Контрольная сумма
- Торренты
- Цифровая подпись
- Имитовставка
- Хранение паролей
- Биткоины
- ...

Часто используемые хеш-функции

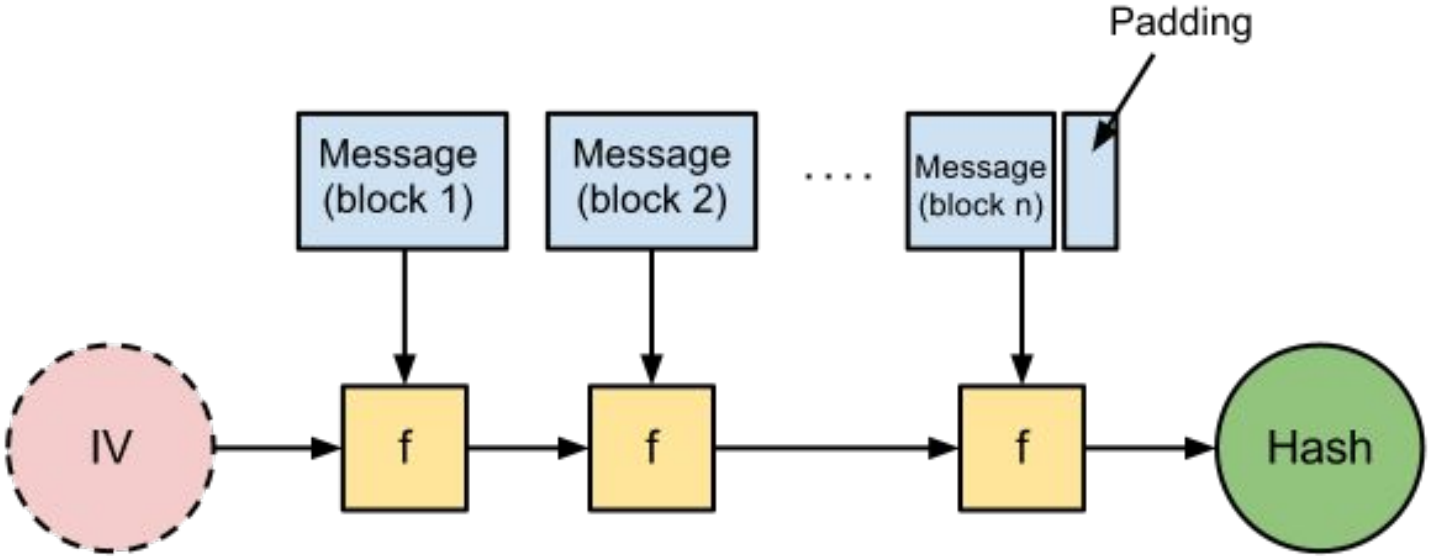
- MD5
- SHA-1
- SHA-2 (SHA-256, SHA-512, ...)
- SHA-3
- bcrypt
- PBKDF2
- ...

Сравнение хеш-функций

Comparison of SHA functions

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Rounds	Operations	Security (bits)	Example Performance[31] (MiB/s)
MD5 (as reference)		128	128 (4 × 32)	512	$2^{64} - 1$	64	And, Xor, Rot, Add (mod 2^{32}), Or	<64 (collisions found)	335
SHA-0		160	160 (5 × 32)	512	$2^{64} - 1$	80	And, Xor, Rot, Add (mod 2^{32}), Or	<80 (collisions found)	-
SHA-1		160	160 (5 × 32)	512	$2^{64} - 1$	80		<80 (theoretical attack[32] in 2^{61})	192
SHA-2	*SHA-224*	224	256 (8 × 32)	512	$2^{64} - 1$	64	And, Xor, Rot, Add (mod 2^{32}), Or, Shr	112	139
	SHA-256	256						128	
	SHA-384	384	512 (8 × 64)	1024	$2^{128} - 1$	80	And, Xor, Rot, Add (mod 2^{64}), Or, Shr	192	154
	SHA-512	512						256	
	SHA-512/224	224						112	
	SHA-512/256	256						128	
SHA-3	*SHA3-224*	224	1600 (5 × 5 × 64)	1152	Unlimited	24	And, Xor, Rot, Not	112	-
	SHA3-256	256		1088				128	
	SHA3-384	384		832				192	
	SHA3-512	512		576				256	
	SHAKE128	*d* (arbitrary)		1344				min (*d*/2, 128)	-
	SHAKE256	*d* (arbitrary)		1088				min (*d*/2, 256)	

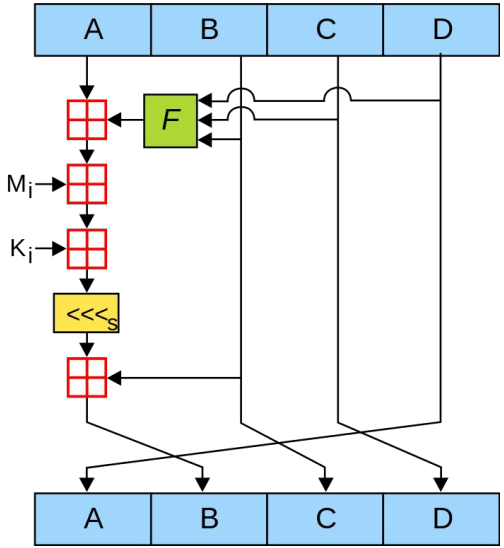
Процесс вычисления



MD5

1. Выравнивание: к сообщению дописывают единичиный бит (байт 0x80), а затем необходимое число нулевых бит, чтобы новый размер сообщения был сравним с 448 по модулю 512.
2. В оставшиеся 64 бита дописывают 64-битное представление длины данных до выравнивания.
3. Инициализируется внутреннее 128-битное состояние.
4. Данные разбиваются на блоки по 512 бит, которые обрабатываются последовательно. В процессе обработки модифицируется внутреннее состояние.

- Основная часть алгоритма обрабатывает 128-битные блоки данных (рисунок справа).
- https://en.wikipedia.org/wiki/MD5#Pseudocode



- A, B, C, D - блоки по 32 бита
- F - нелинейная функция
- Mi - 32-битный блок сообщения
- Ki - 32-битная константа

MAC (Message Authentication Code)

- Пусть Алиса и Боб оба знают секретный ключ key
- Алиса отправляет Бобу сообщение message
- Для проверки целостности вместе с сообщением message отправляется MAC = hash(key + message)
- При получении Боб может проверить MAC == hash(key + message)

MAC: вычисление

- Пусть secret = ‘secret’
- Пусть message = ‘data’
- Пусть hash = MD5
- Блок в 512 бит с выравниванием:

```
0000  73 65 63 72 65 74 64 61 74 61 80 00 00 00 00 00  secretdata......
0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0030  00 00 00 00 00 00 00 00 50 00 00 00 00 00 00 00  ........P.......
```

MAC: length extension attack

- Возьмем блок с выравниванем, припишем к нему фрагмент ‘append’
- Можно продолжить вычисление хеша, взяв в качестве начального внутренного состояния hash(key + message)

```
0000  73 65 63 72 65 74 64 61 74 61 80 00 00 00 00 00  secretdata......
0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0030  00 00 00 00 00 00 00 00 50 00 00 00 00 00 00 00  ........P.......
0040  61 70 70 65 6e 64                                append
```

Length extension attack

Original Data: count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo

Original Signature: 6d5f807e23db210bc254a28be2d6759a0f5f5d99

Desired New Data: count=10&lat=37.351&user_id=1&long=-119.827

&waffle=eggo&waffle=liege

signature = hash(secret + data)

Length extension attack

New Data: count=10&lat=37.351&user_id=1&long=-119.827
&waffle=eggo\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02\x28&waffle=liege

New Signature: 37c9dcaef9a370feb2daf97211a1bbb273812753

Length extension attack

- MD5, SHA-1, SHA-2 - уязвимы
- SHA-3 - нет

- HMAC = H(key1 || H(key2 || message))

Как хранить пароли пользователей?

Как хранить пароли пользователей?

- Использовать соль
  - salt = случайная строка
  - Вместо hash(password) хранить hash(salt + password) и salt
  - Разные соли для разных пользователей

- Использовать медленную хеш-функцию
  - Например bcrypt

bcrypt vs SHA-*

Хеш	Производительность
SHA-1	~20756000
SHA-256	~27492000
SHA-512	~13189000
bcrypt	~3160

bcrypt

- `$2a$10$/bWghfYQY6zG5GQLOtl5VuqYn.P0WwL3Q6sgmCu84fjzn2FGFbiUa`
  - `2a` - версия bcrypt
  - `10` - параметр стоимости, 2^10 раундов вычислений
  - `/bWghfYQY6zG5GQLOtl5Vu` - соль
  - `qYn.P0WwL3Q6sgmCu84fjzn2FGFbiUa` - хеш

Как ломать хеши?

Как ломать хеши?

- Варианты атак
  - Просто перебор
  - Перебор по словарю
  - Использование правил (rules)
  - Перебор по маскам
  - Радужные таблицы
- Инструменты
  - hashcat: https://hashcat.net/hashcat/
  - John the Ripper: http://www.openwall.com/john/

python + bash

```
>>> import hashlib
>>> m = hashlib.md5()
>>> m.update('Hello world!')
>>> m.hexdigest()
'86fb269d190d2c85f6e0468ceca42a20'
```

```
>>> import hashlib
>>> m = hashlib.sha1()
>>> m.update('Hello world!')
>>> m.hexdigest()
'd3486ae9136e7856bc42212385ea797094475802'
```

```
$ echo -n "Hello world!" | md5sum
86fb269d190d2c85f6e0468ceca42a20  -
$ echo -n "Hello world!" | sha1sum
d3486ae9136e7856bc42212385ea797094475802  -
```

Вопросы?