Password Security

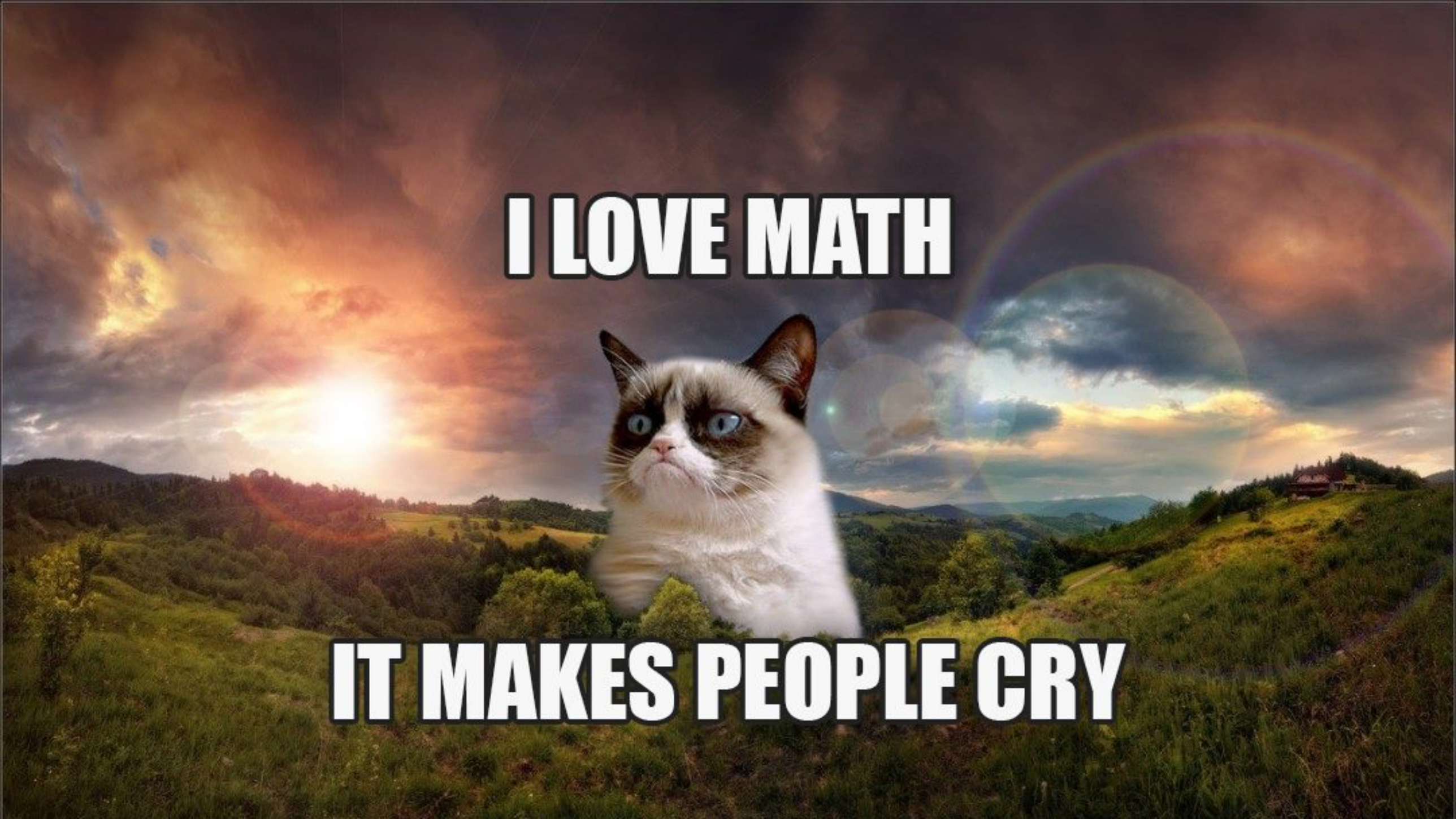
Emil Bay

CommodiTrader

Technical Founder, CommodiTrader

@emilbayes 🇩🇰

Johannes Bader, hacker_two, CC BY 2.0


I LOVE MATH
IT MAKES PEOPLE CRY

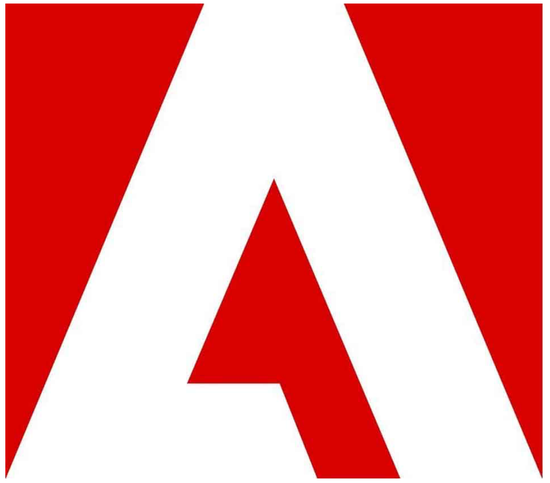
Passwords

Proves your identity

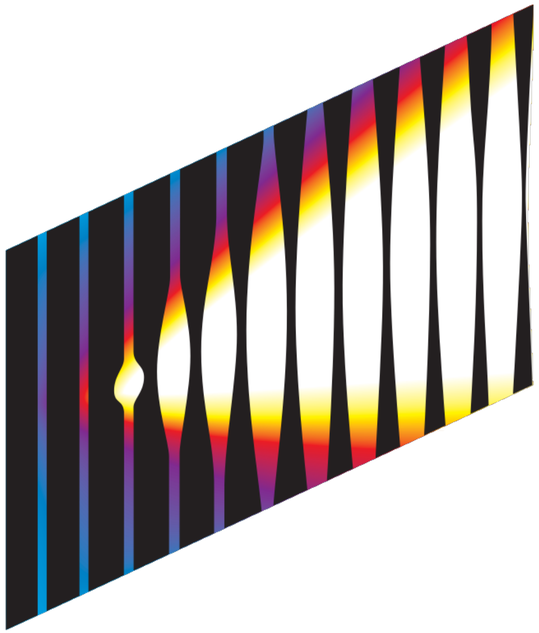
1. No security holes
2. Unique passwords
3. Safer storage

1. No security holes
2. Unique passwords
3. **Safer storage**


comcast




Bell



Johannes Bader, hacker_two, CC BY 2.0

Plaintext

username	password
emilbayes	secret
cat	AllHailTheLaserPo1nter
dog	squirrel
dog1993	squirrel

Problem

Plaintext!


SONY
PICTURES


Bell


comcast

Solution

Obscure Password

Hash

1. Deterministic
2. Pre-image resitant (one-way)
3. Second pre-image resitant (voluntary collisions)
4. Collision resistance (involuentary collisions)

$$\{0,1\}^* \mapsto \{0,1\}^n$$

~~MD5~~

~~SHA-1~~

SHA-2 (SHA-256 & SHA-512)

Blake2 🏎️

SHA-3 (Keccak)

bcrypt, scrypt

Argon2 💪

```
var crypto = require('crypto')

var hash = crypto.createHash('md5')
  .update(password)
  .digest()
```

username	password
emilbayes	secret
cat	AllHailTheLaserPo1nter
dog	squirrel
dog1993	squirrel

username	password
emilbayes	5ebe2294ecd0e0f08eab7690d2a6ee69
cat	5b85f85ee4254613ddb1a4de584d1eb8
dog	eac074b0503b45740d49b18eb659bcfc
dog1993	eac074b0503b45740d49b18eb659bcfc

username	password
emilbayes	5ebe2294ecd0e0f08eab7690d2a6ee69
cat	5b85f85ee4254613ddb1a4de584d1eb8
dog	eac074b0503b45740d49b18eb659bcfc
dog1993	eac074b0503b45740d49b18eb659bcfc

Problem

Tables

```
var words = //...

var rainbowTable = words.map(function (word) {
  return crypto.createHash('md5')
    .update(word)
    .digest()
})

var idx = indexOf(hash, rainbowTable)
var plaintext = words[idx]
```

Solution

Make identical passwords yield unique hashes

Salted Hash

Makes precomputation impractical
aka Rainbow Tables

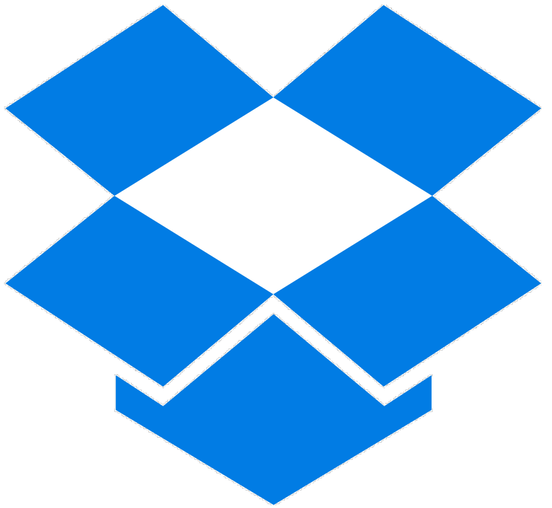
```
var crypto = require('crypto')

var salt = crypto.randomBytes(64)
var hash = crypto.createHash('md5')
  .update(salt)
  .update(':')
  .update(password)
  .digest()
```

username	password
emilbayes	5ebe2294ecd0e0f08eab7690d2a6ee69
cat	5b85f85ee4254613ddb1a4de584d1eb8
dog	eac074b0503b45740d49b18eb659bcfc
dog1993	eac074b0503b45740d49b18eb659bcfc

username	password	salt
emilbayes	863e11939ed9fb702e3173fb50c88d5f	b0…49
cat	f57c3079eb373d70aed39da8f5d3488f	cb…b7
dog	cb6dbdd525cb6e47db82c57119323975	4e…fd
dog1993	92cadf92421f44331bb231ed824e0eae	f1…cc

Problem

Too Efficient



Iterated Hash

NIST compliant

```
function iterHash (plaintext, iters) {
  var digest = hash(plaintext)

  for (var i = 0; i < iters; i++) {
    digest = hash(digest)
  }

  return digest
}
```

```
var crypto = require('crypto')

var salt = crypto.randomBytes(128)
var iter = 10000
var hashLen = 512
var algo = 'sha512'

crypto.pbkdf2(password, salt, iter, hashLen, algo, function (err, hash) {
  // ...
})
```

Problem

Parallelisable

KDF

*Key Derivation Function
Purposefully slow
Uses lots of memory and computation

Argon2

Winner of PHC

```
var sodium = require('sodium-native')

var hash = Buffer.allocUnsafe(sodium.crypto_pwhash_STRBYTES)
sodium.crypto_pwhash_str(hash, password,
                         sodium.crypto_pwhash_OPSLIMIT_INTERACTIVE,
                         sodium.crypto_pwhash_MEMLIMIT_INTERACTIVE)
```

Problem

Blocking

Solution

Async

```
npm install secure-password
```

```
var pwd = require('secure-password')()

var userPassword = Buffer.from('my-password')
pwd.hash(userPassword, callback)
```

libsodium

No `require('crypto')`

```
var sodium = require('sodium-native')
```

```
var sodium = require('sodium-universal')
```

```
var SecurePassword = require('secure-password')
```

```
var SecurePassword = require('secure-password')
```