Passwords in Servant

This is very quick

Cyrill Brunner

17th October 2022

What will I show you?

- Scrypt

Scrypt

`scrypt` takes 4 parameters:

- `N` - A power of 2, the “Cost factor”

Scrypt

`scrypt` takes 4 parameters:

- `N` - A power of 2, the "Cost factor"
- `r` - The "Block size factor"

Scrypt

`scrypt` takes 4 parameters:

- `N` - A power of 2, the "Cost factor"
- `r` - The "Block size factor"
- `p` - The "Parallelization factor"

Scrypt

`scrypt` takes 4 parameters:

- `N` - A power of 2, the "Cost factor"
- `r` - The "Block size factor"
- `p` - The "Parallelization factor"
- Desired key length

Scrypt

`scrypt` takes 4 parameters:

- `N` - A power of 2, the "Cost factor"
- `r` - The "Block size factor"
- `p` - The "Parallelization factor"
- Desired key length

Scrypt

`scrypt` takes 4 parameters:

- `N` - A power of 2, the "Cost factor"
- `r` - The "Block size factor"
- `p` - The "Parallelization factor"
- Desired key length

The memory used is defined by $N * r * p$. $r * p < 2^{30}$.

Scrypt

`scrypt` takes 4 parameters:

- `N` - A power of 2, the "Cost factor"
- `r` - The "Block size factor"
- `p` - The "Parallelization factor"
- Desired key length

The memory used is defined by $N * r * p$. $r * p < 2^{30}$. Defaults used in the `scrypt` library, using $log_2(N)$ instead:

Scrypt

`scrypt` takes 4 parameters:

- `N` - A power of 2, the "Cost factor"
- `r` - The "Block size factor"
- `p` - The "Parallelization factor"
- Desired key length

The memory used is defined by $N * r * p$. $r * p < 2^{30}$. Defaults used in the `scrypt` library, using $log_2(N)$ instead:

```
log_2(N) = 14
r        =  8
p        =  1
keyLen   = 64
-- from `defaultParams`
-- 128 * 8 * 2^14 = 16MB
```

Scrypt

```
do
  let password = Pass "myPassword"
      keyParams = defaultParams
```

Scrypt

```
do
  let password = Pass "myPassword"
      keyParams = defaultParams

  mySalt <- newSalt
```

Scrypt

```
do
  let password = Pass "myPassword"
      keyParams = defaultParams

  mySalt <- newSalt -- /dev/urandom, or CryptoAPI on Windows

  let encrypted = encryptPass keyParams mySalt password

  print encrypted
  -- EncryptedPass {
  --   getEncryptedPass = "14|8|1|..salt..|..hash.."
  -- }
```

Scrypt

```haskell
do
  let password = Pass "myPassword"
      keyParams = defaultParams


  encrypted <- encryptIO keyParams password
  -- uses `newSalt` internally

  print encrypted
  -- EncryptedPass {
  --   getEncryptedPass = "14|8|1|..salt..|..hash.."
  -- }
```

Scrypt

```
print $ verifyPass keyParams (Pass "myPassword") encrypted
-- (True, Nothing)
```

Scrypt

```
print $ verifyPass keyParams (Pass "myPassword") encrypted
-- (True, Nothing)
print $ verifyPass keyParams (Pass "MyPassword") encrypted
-- (False, Nothing)
```

Scrypt

```
print $ verifyPass keyParams (Pass "myPassword") encrypted
-- (True, Nothing)
print $ verifyPass keyParams (Pass "MyPassword") encrypted
-- (False, Nothing)

let strongerParams = fromJust $ scryptParams 16 8 1
```

Scrypt

```
print $ verifyPass defaultParams (Pass "myPassword") encrypted
-- (True, Nothing)
print $ verifyPass defaultParams (Pass "MyPassword") encrypted
-- (False, Nothing)

let strongerParams = fromJust $ scryptParams 16 8 1

print $ verifyPass strongerParams (Pass "myPassword") encrypted
-- (True, Just (EncryptedPass {
--     getEncryptedPass = "16|8|1|..salt..|..hash.."
-- }))
```