

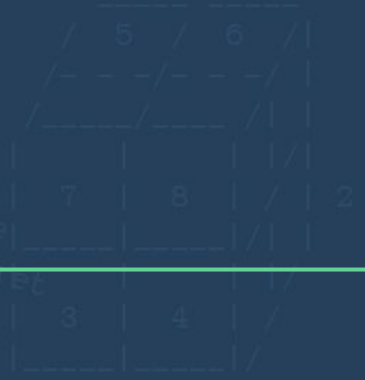
UX/UI

Buttons, Interactions, Navigation

```
* The coordinates (0, 0, 0) represents the octocube
*/
class GeoOctocube {
```

* Gets the sector from the (x, y, z) specified

* Sector will be:



```
* @param int $x the x coordinate
* @param int $y the y coordinate
* @param int $z the z coordinate
```

* @return int the number of the sector (0 if x =

```
static function get_sector ($x, $y, $z) {
```

Today

- Buttons
- Interactions
- Navigation

```
* The coordinates (0, 0, 0) represents the octocube
*/
class GeoOctocube {
```

* Gets the sector from the (x, y, z) specified

* Sector will be:

* <code>

		5	6	
		-	-	-
	7	8		2
	3	4		

* param int \$x the x coordinate
* param int \$y the y coordinate
* param int \$z the z coordinate

* @return int the number of the sector (0 if x =

```
static function get_sector ($x, $y, $z) {
```

```
* Common fields
format = format[0]
regionnumbers = [] * placeholder
keytype = "" * placeholder
```

```
* Ducky heuristic to distinguish known hosts from known hosts2.
is second field entirely decimal digits?
if (is_numeric($fields[1]))
```

```
* Treat all keys as host key
* Format: hostkey [keyid] [comment]
* (PUTT doesn't store the order of bits)
regionnumbers = map (long, $fields[1])
keytype = "key"
```

```
function get_sector($x, $y, $z) {
    $x = intval($x);
    $y = intval($y);
    $z = intval($z);
    if ($x < 0 || $x > 7 || $y < 0 || $y > 7 || $z < 0 || $z > 7) {
        return -1;
    }
    $sector = 0;
    for ($i = 0; $i < 8; $i++) {
        for ($j = 0; $j < 8; $j++) {
            for ($k = 0; $k < 8; $k++) {
                $sector++;
                if ($x == $i || $y == $j || $z == $k) {
                    $sector--;
                }
            }
        }
    }
    return $sector;
}
```

Menu navigation

```

require 'base32'
require 'cryptosphere'

module Cryptosphere
  # Blocks are the underlying convergent encryption primitive
  # Cryptosphere for data confidentiality. If you are looking for the secret sauce, welcome my friend, you have found the right place.
  # Blocks provide for both confidentiality and confidentiality for plaintexts between 0 bytes and 2^32 bytes (1 megabyte). Blocks are encrypted using a convergent cryptographic technique known as content hashing. With this approach, cryptographic hash of the file's contents is calculated using a symmetric key with an optional random hash keying parameter.
  # The symmetric key is derived, which is used to encrypt the file contents using a convergent symmetric cipher.
  # More specifics on the encryption, please see Blake2bXSalsa20poly1305.

  def self.encrypt(key, contents)
    # Generate a key for the encryption
    key = key || generate_key

    # Encrypt the contents
    encrypted_contents = encrypt_block(key, contents)

    # Generate a header for the encrypted contents
    header = generate_header(key, encrypted_contents)

    [header, encrypted_contents]
  end

  def self.decrypt(key, header_and_contents)
    # Extract the key and contents
    header, contents = header_and_contents

    # Decrypt the contents
    decrypted_contents = decrypt_block(key, contents)

    # Verify the header
    verify_header(key, header, decrypted_contents)

    decrypted_contents
  end

  def self.verify(key, header_and_contents)
    # Extract the key and contents
    header, contents = header_and_contents

    # Decrypt the contents
    decrypted_contents = decrypt_block(key, contents)

    # Verify the header
    verify_header(key, header, decrypted_contents)

    true
  end

  def self.verify_header(key, header, decrypted_contents)
    # Extract the key and contents
    key, contents = key, decrypted_contents

    # Decrypt the contents
    decrypted_contents = decrypt_block(key, contents)

    # Verify the header
    verify_header(key, header, decrypted_contents)

    true
  end

  def self.verify_contents(key, header_and_contents)
    # Extract the key and contents
    header, contents = header_and_contents

    # Decrypt the contents
    decrypted_contents = decrypt_block(key, contents)

    # Verify the header
    verify_header(key, header, decrypted_contents)

    true
  end

  def self.verify_key(key, header_and_contents)
    # Extract the key and contents
    header, contents = header_and_contents

    # Decrypt the contents
    decrypted_contents = decrypt_block(key, contents)

    # Verify the header
    verify_header(key, header, decrypted_contents)

    true
  end

  def self.verify_all(key, header_and_contents)
    # Extract the key and contents
    header, contents = header_and_contents

    # Decrypt the contents
    decrypted_contents = decrypt_block(key, contents)

    # Verify the header
    verify_header(key, header, decrypted_contents)

    # Verify the contents
    verify_contents(key, header, decrypted_contents)

    # Verify the key
    verify_key(key, header_and_contents)

    true
  end

  def self.generate_key
    # Generate a key for the encryption
    SecureRandom.random_bytes(32)
  end

  def self.encrypt_block(key, contents)
    # Encrypt the contents
    encrypted_contents = Crypto.encrypt(contents, key)

    encrypted_contents
  end

  def self.decrypt_block(key, contents)
    # Decrypt the contents
    decrypted_contents = Crypto.decrypt(contents, key)

    decrypted_contents
  end

  def self.verify_header(key, header, decrypted_contents)
    # Extract the key and contents
    key, contents = key, decrypted_contents

    # Decrypt the contents
    decrypted_contents = decrypt_block(key, contents)

    # Verify the header
    verify_header(key, header, decrypted_contents)

    true
  end

  def self.verify_contents(key, header, decrypted_contents)
    # Extract the key and contents
    header, contents = header, decrypted_contents

    # Decrypt the contents
    decrypted_contents = decrypt_block(key, contents)

    # Verify the header
    verify_header(key, header, decrypted_contents)

    # Verify the contents
    verify_contents(key, header, decrypted_contents)

    true
  end

  def self.verify_key(key, header_and_contents)
    # Extract the key and contents
    header, contents = header_and_contents

    # Decrypt the contents
    decrypted_contents = decrypt_block(key, contents)

    # Verify the header
    verify_header(key, header, decrypted_contents)

    # Verify the key
    verify_key(key, header_and_contents)

    true
  end

  def self.verify_all(key, header_and_contents)
    # Extract the key and contents
    header, contents = header_and_contents

    # Decrypt the contents
    decrypted_contents = decrypt_block(key, contents)

    # Verify the header
    verify_header(key, header, decrypted_contents)

    # Verify the contents
    verify_contents(key, header, decrypted_contents)

    # Verify the key
    verify_key(key, header_and_contents)

    true
  end

  def self.generate_header(key, encrypted_contents)
    # Generate a header for the encrypted contents
    header = generate_header(key, encrypted_contents)

    header
  end

  def self.verify_header(key, header, decrypted_contents)
    # Extract the key and contents
    key, contents = key, decrypted_contents

    # Decrypt the contents
    decrypted_contents = decrypt_block(key, contents)

    # Verify the header
    verify_header(key, header, decrypted_contents)

    true
  end

  def self.verify_contents(key, header, decrypted_contents)
    # Extract the key and contents
    header, contents = header, decrypted_contents

    # Decrypt the contents
    decrypted_contents = decrypt_block(key, contents)

    # Verify the header
    verify_header(key, header, decrypted_contents)

    # Verify the contents
    verify_contents(key, header, decrypted_contents)

    true
  end

  def self.verify_key(key, header_and_contents)
    # Extract the key and contents
    header, contents = header_and_contents

    # Decrypt the contents
    decrypted_contents = decrypt_block(key, contents)

    # Verify the header
    verify_header(key, header, decrypted_contents)

    # Verify the key
    verify_key(key, header_and_contents)

    true
  end

  def self.verify_all(key, header_and_contents)
    # Extract the key and contents
    header, contents = header_and_contents

    # Decrypt the contents
    decrypted_contents = decrypt_block(key, contents)

    # Verify the header
    verify_header(key, header, decrypted_contents)

    # Verify the contents
    verify_contents(key, header, decrypted_contents)

    # Verify the key
    verify_key(key, header_and_contents)

    true
  end

  def self.generate_key
    # Generate a key for the encryption
    SecureRandom.random_bytes(32)
  end

  def self.encrypt_block(key, contents)
    # Encrypt the contents
    encrypted_contents = Crypto.encrypt(contents, key)

    encrypted_contents
  end

  def self.decrypt_block(key, contents)
    # Decrypt the contents
    decrypted_contents = Crypto.decrypt(contents, key)

    decrypted_contents
  end

  def self.verify_header(key, header, decrypted_contents)
    # Extract the key and contents
    key, contents = key, decrypted_contents

    # Decrypt the contents
    decrypted_contents = decrypt_block(key, contents)

    # Verify the header
    verify_header(key, header, decrypted_contents)

    true
  end

  def self.verify_contents(key, header, decrypted_contents)
    # Extract the key and contents
    header, contents = header, decrypted_contents

    # Decrypt the contents
    decrypted_contents = decrypt_block(key, contents)

    # Verify the header
    verify_header(key, header, decrypted_contents)

    # Verify the contents
    verify_contents(key, header, decrypted_contents)

    true
  end

  def self.verify_key(key, header_and_contents)
    # Extract the key and contents
    header, contents = header_and_contents

    # Decrypt the contents
    decrypted_contents = decrypt_block(key, contents)

    # Verify the header
    verify_header(key, header, decrypted_contents)

    # Verify the key
    verify_key(key, header_and_contents)

    true
  end

  def self.verify_all(key, header_and_contents)
    # Extract the key and contents
    header, contents = header_and_contents

    # Decrypt the contents
    decrypted_contents = decrypt_block(key, contents)

    # Verify the header
    verify_header(key, header, decrypted_contents)

    # Verify the contents
    verify_contents(key, header, decrypted_contents)

    # Verify the key
    verify_key(key, header_and_contents)

    true
  end

  def self.generate_header(key, encrypted_contents)
    # Generate a header for the encrypted contents
    header = generate_header(key, encrypted_contents)

    header
  end
end

class GeoOctocube
  def initialize(x, y, z)
    @x = x
    @y = y
    @z = z
  end

  def get_sector
    # Gets the sector from the (x, y, z) specified
    # Sector will be:
    # <code>
    # | 5 | 6 |
    # | 7 | 8 | 2
    # | 3 | 4 |
    # |  |  |
    # @return int the number of the sector (0 if x = 0 and y = 0 and z = 0)
  end

  def get_sector(x, y, z)
    # Gets the sector from the (x, y, z) specified
    # Sector will be:
    # <code>
    # | 5 | 6 |
    # | 7 | 8 | 2
    # | 3 | 4 |
    # |  |  |
    # @return int the number of the sector (0 if x = 0 and y = 0 and z = 0)
  end
end

```

Buttons

- Form and function

- Form IS function
- Beauty is secondary

- Self-explanatory

- In menu - would you know what to press without the text?

```
 * The coordinates (0, 0, 0) represents the octocube
 */
class GeoOctocube {
```

```
 * Gets the sector from the (x, y, z) specified
```

```
 * Sector will be:
```



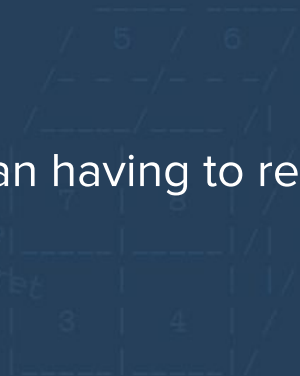
```
 * @param int $x the x coordinate
 * @param int $y the y coordinate
 * @param int $z the z coordinate
```

```
 * @return int the number of the sector (0 if x =
```

```
static function get_sector($x, $y, $z) {
```

Recognition vs. Recall

- Remembering things already in your head vs. picking the right answer
- Being in a store, trying to find the right shelf
 - Without visual aid
 - With clear signs
- Help player see what they are looking for, rather than having to remember which button to press



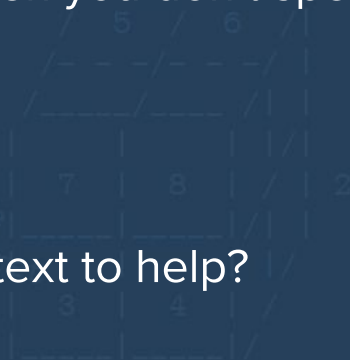
```
param int $x the x coordinate
param int $y the y coordinate
param int $z the z coordinate
@return int the number of the sector (0 if x =
static function get_sector ($x, $y, $z) {
```

An example:

If your little sister changes the language of your phone to Russian, would you find your way back to the right menu and change it back (given you don't speak Russian?)

Why/Why not?

How can you design a UI that you can navigate without text to help?



3G

9:42 AM



В В С РУССКАЯ СЛУЖБА

Главные новости

Последние новости

У мэрии Москвы произошла стычка из-за митингов



Сторонники и противники российского премьер-министра Владимира Путина устроили потасовки у...

Медведев встретится с лидерами неразрешенных партий



Уходящий президент России Дмитрий Медведев в понедельник, как ожидается, проведет редкую в...

Новым президентом Германии станет правозащитник из ГДР



Канцлер Германии Ангела Меркель заявила, что поддерживает кандидатуру



Новости



Популярное

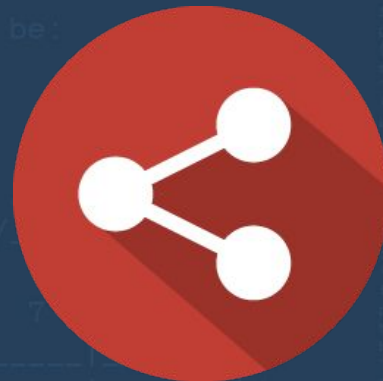


Разделы



Настройки

Common Imagery



require 'base32'
require 'cryptosphere'

```
class CryptoPrimitive
  attr_reader :key

  def initialize(key)
    @key = key
  end

  def encode(data)
    # Base32 encoding
    base32 = Base32.new
    base32.encode(data)
  end

  def decode(encoded_data)
    base32 = Base32.new
    base32.decode(encoded_data)
  end

  def validate(data)
    # Validation logic
  end
end
```

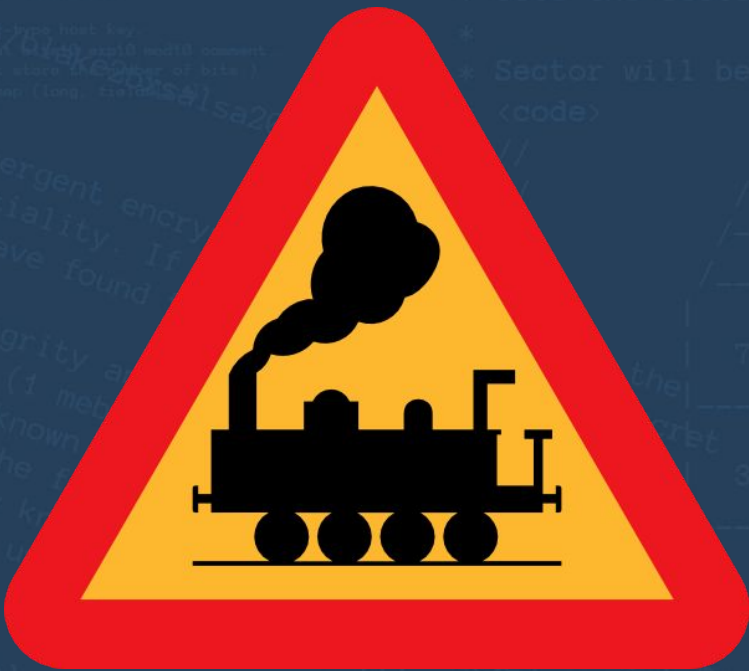
Blocks are the underlying
Cryptosphere for data
sausage, welcome my
Blocks provide
between 0 bytes
convergent encr
approach, a crypt
together with an op
symmetric key is de
authenticated symmetric
more specifics on the encryption, please see Blake2bXSalsa20p
key to use when calculating the ID of a block
matches the URI scheme for blocks
= "crypt_block"
dit on the size of
= 1_048_576

The coordinates (0, 0, 0) represents the octocube
*/
class GeoOctocube {

 /*
 * Gets the sector from the (x, y, z) specified
 *
 * Sector will be:
 *
 * <code>
 * //
 *
 * param int \$x the x coordinate
 * param int \$y the y coordinate
 * param int \$z the z coordinate
 * @return int the number of the sector (0 if x =
 */
 static function get_sector (\$x, \$y, \$z) {
 return (\$x % 8) * 8 + (\$y % 8) * 8 + (\$z % 8) * 8
 }
}

```
function get_sector ($x, $y, $z) {  
  $x = $x % 8;  
  $y = $y % 8;  
  $z = $z % 8;  
  return ($x * 8 + $y * 8 + $z * 8);  
}
```





```
require 'base32'
require 'cryptosphere'
module Cryptosphere
  # Blocks are the underlying convergent encryption
  # Cryptosphere for data confidentiality. If
  # sauce, welcome my friend, you have found
  Blocks provide for both data integrity and
  between 0 bytes and 1048576 bytes (1 megabyte)
  convergent encryption technique known as
  approach, a cryptographic hash of the
  together with an optional random key for
  symmetric key is derived, which is used
  authenticated symmetric cipher.
  # More specifics on the encryption, please see Blake2bXSalsa20
  # key to use when calculating the ID of a block
  # matches the URI scheme for blocks
  = "crypt-block"
  # More on the size of a block
  = 1_048_576

  # Common fields
  @total = fields[0]
  @signature = {} # placeholder
  @type = "" # placeholder

  # Dirty heuristic to distinguish known hosts from known hosts?
  # is second field entirely decimal digit?
  # if so, use it as a host key
  # Treat all fields as host key
  # Format: hostkey[0]:hostkey[1]:comment
  # (PUTTY doesn't store the size of bits)
  @signature = mac(long, fields)
  @type = ""

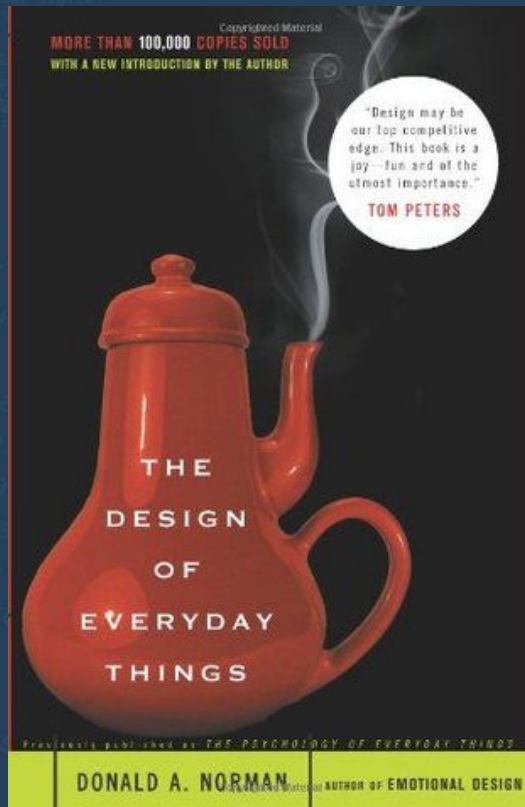
  # The coordinates (0, 0, 0) represents the octocube
  */
  class GeoOctocube {
    # Gets the sector from the (x, y, z) specified
    #
    # Sector will be:
    # <code>
    #
  }

  # The x coordinate
  # @param int $x the x coordinate
  # @param int $y the y coordinate
  # @param int $z the z coordinate
  # @return int the number of the sector (0 if x =
  #
  static function get_sector ($x, $y, $z) {
    5 6
    - -
    7 8
    3 4
  }
end
```


How to interact with an object



Reading tip:



Buttons

```
require 'base32'
require 'cryptosphere'

module Cryptosphere
  # Blocks are the underlying convergent encryption primitive
  # Cryptosphere for data confidentiality. If you are looking for the secret
  # sauce, welcome my friend, you have found the right place.

  # Blocks provide for both data integrity and confidentiality for plaintexts
  # between 0 bytes and 1048576 bytes (1 mebibyte). Blocks are encrypted using
  # a convergent cryptographic technique known as content hashing. With this
  # approach, the cryptographic hash of the file's contents is calculated using
  # together with an optional random hash key. This hash is used to generate a
  # symmetric key is derived, which is used to encrypt the file with an
  # authenticated symmetric cipher.

  # More specifics on the encryption, please see Blake2bXSalsa20poly1305
  # key to use when calculating the ID of a block
  # matches the URI scheme for blocks
  # = "crypt:block"
  # ID on the size of a block
  # = 1_048_576

  # Common fields
  @serial = fields[0]
  @regionnumbase = [] # placeholder
  @keytype = "" # placeholder

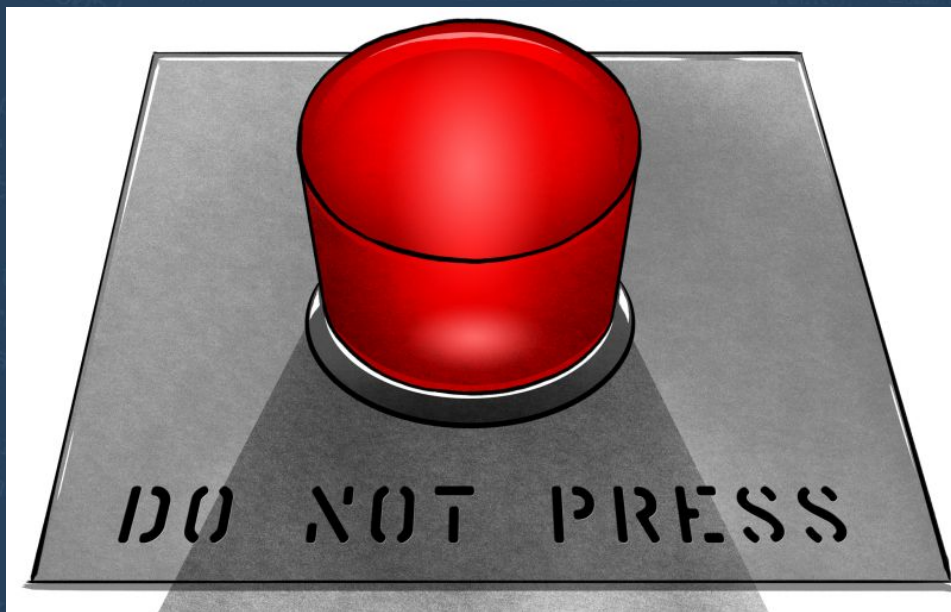
  # Ducky heuristic to distinguish known_hosts from known_hosts2.
  # Is second field entirely decimal digit?
  # @serial (r"^\d+$", fields[1])

  # Travel all ways, type host key
  # Format: hostkey @ip:port md5ib comment
  # (PUTT doesn't store the size of bits)
  @regionnumbase = mac(long: fields[0])
  @keytype = "raw"

  # The coordinates (0, 0, 0) represents the octocube
  */
  class GeoOctocube {
    /**
     * Gets the sector from the (x, y, z) specified
     *
     * Sector will be:
     *
     * <code>
     * //
     * //
     * // | 5 | 6 |
     * // | - | - |
     * // | - | - |
     * // | 7 | 8 | 2
     * // | - | - |
     * // | 3 | 4 |
     * // | - | - |
     * //
     * @param int $x the x coordinate
     * @param int $y the y coordinate
     * @param int $z the z coordinate
     *
     * @return int the number of the sector (0 if x =
     */
    static function get_sector($x, $y, $z) {
  
```

Buttons

- Protrude from surroundings
- Are pushed
- Binary (ON/OFF)



Mimic physical items



Tutorial: Super simple button

[This is where I start Photoshop]



