Package ‘homomorpheR’

April 10, 2025

Type Package

Title Homomorphic Computations in R

Version 0.3

Date 2025-04-08

VignetteBuilder knitr

URL https://github.com/bnaras/homomorpheR

BugReports https://github.com/bnaras/homomorpheR/issues

Suggests knitr, rmarkdown, survival, dplyr, magrittr, digest

Imports R6, gmp, sodium

Description Homomorphic computations in R for privacy-preserving applications. Currently only the Paillier Scheme is implemented.

License MIT + file LICENSE

RoxygenNote 7.3.2

Encoding UTF-8

NeedsCompilation no

Author Balasubramanian Narasimhan [aut, cre]

Maintainer Balasubramanian Narasimhan <naras@stat.Stanford.EDU>

Repository CRAN

Date/Publication 2025-04-09 22:30:12 UTC

Contents

homomorpheR 2
PaillierKeyPair 3
PaillierPrivateKey 4
PaillierPublicKey 5
random.bigz 8

Index **9**

homomorpheR — *homomorpheR: Homomorphic computations in R*

Description

homomorpheR is a start at a rudimentary package for homomorphic computations in R. The goal is to collect homomorphic encryption schemes in this package for privacy-preserving distributed computations; for example, applications of the sort immplemented in package distcomp.

Details

At the moment, only one scheme is implemented, the Paillier scheme. The current implementation makes no pretense at efficiency and also uses direct translations of other implementations, particularly the one in Javascript.

For a quick overview of the features, refer to the vignettes in this package.

Author(s)

Maintainer: Balasubramanian Narasimhan <naras@stat.Stanford.EDU>

References

Homomorphic Encryyption

Paillier Encryption

See Also

Useful links:

- https://github.com/bnaras/homomorpheR
- Report bugs at https://github.com/bnaras/homomorpheR/issues

Examples

```
keys <- PaillierKeyPair$new(1024) # Generate new key pair
encryptAndDecrypt <- function(x) keys$getPrivateKey()$decrypt(keys$pubkey$encrypt(x))
a <- gmp::as.bigz(1273849)
identical(a + 10L, encryptAndDecrypt(a+10L))
x <- lapply(1:100, function(x) random.bigz(nBits = 512))
edx <- lapply(x, encryptAndDecrypt)
identical(x, edx)
```

`PaillierKeyPair` *Construct a Paillier public and private key pair given a fixed number of bits*

Description

Construct a Paillier public and private key pair given a fixed number of bits

Construct a Paillier public and private key pair given a fixed number of bits

Format

An `R6::R6Class()` generator object

Methods

`PaillierKeyPair$getPrivateKey()` Return the private key

Public fields

`pubkey` the public key

Methods

Public methods:

- `PaillierKeyPair$new()`
- `PaillierKeyPair$getPrivateKey()`
- `PaillierKeyPair$clone()`

Method `new()`**:** Create a new public private key pair with specified number of modulus bits

Usage:

`PaillierKeyPair$new(modulusBits)`

Arguments:

`modulusBits` the number of bits to use

Returns: a `PaillierKeyPair` object

Method `getPrivateKey()`**:** Return the private key

Usage:

`PaillierKeyPair$getPrivateKey()`

Returns: the private key

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`PaillierKeyPair$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

PaillierPublicKey() and PaillierPrivateKey()

Examples

```
keys <- PaillierKeyPair$new(1024)
keys$pubkey
keys$getPrivateKey()
```

PaillierPrivateKey *Construct a Paillier private key with the given secret and a public key*

Description

Construct a Paillier private key with the given secret and a public key

Construct a Paillier private key with the given secret and a public key

Format

An R6::R6Class() generator object

Public fields

`pubkey` the public key

Methods

Public methods:

- PaillierPrivateKey$new()
- PaillierPrivateKey$getLambda()
- PaillierPrivateKey$decrypt()
- PaillierPrivateKey$clone()

Method `new()`**:** Create a new private key with given secret `lambda` and the public key

Usage:

```
PaillierPrivateKey$new(lambda, pubkey)
```

Arguments:

`lambda` the secret

`pubkey` the public key

Method `getLambda()`**:** Return the secret lambda

Usage:

```
PaillierPrivateKey$getLambda()
```

Returns: lambda

Method `decrypt()`**:** Decrypt a message

Usage:

```
PaillierPrivateKey$decrypt(c)
```

Arguments:

`c` the message

Returns: the decrypted message

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PaillierPrivateKey$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`PaillierPublicKey()` which goes hand-in-hand with this object

`PaillierPublicKey` *Construct a Paillier public key with the given modulus.*

Description

Construct a Paillier public key with the given modulus.

Construct a Paillier public key with the given modulus.

Value

An `R6::R6Class()` generator object

Public fields

`bits` the number of bits in the modulus

`n` the modulus

`nSquared` the square of the modulus

`nPlusOne` one more than the modulus

Methods

Public methods:

- `PaillierPublicKey$new()`
- `PaillierPublicKey$encrypt()`
- `PaillierPublicKey$add()`
- `PaillierPublicKey$sub()`
- `PaillierPublicKey$add_real()`
- `PaillierPublicKey$sub_real()`
- `PaillierPublicKey$mult()`
- `PaillierPublicKey$clone()`

Method `new()`**:** Create a new public key and precompute some internal values for efficiency

Usage:

```
PaillierPublicKey$new(bits, n)
```

Arguments:

`bits` number of bits to use

`n` the modulus to use

Returns: a new `PaillierPublicKey` object

Method `encrypt()`**:** Encrypt a message

Usage:

```
PaillierPublicKey$encrypt(m)
```

Arguments:

`m` the message

Returns: the encrypted message

Method `add()`**:** Add two encrypted messages

Usage:

```
PaillierPublicKey$add(a, b)
```

Arguments:

`a` a message

`b` another message

Returns: the sum of `a` and `b`

Method `sub()`**:** Subtract one encrypted message from another

Usage:

```
PaillierPublicKey$sub(a, b)
```

Arguments:

`a` a message

`b` another message

Returns: the difference `a - b`

Method `add_real()`**:** Return the sum `a + b` of an encrypted real message `a`, a list consisting of a encrypted integer part (named `int`) and an encrypted fractional part (named `frac`), and a real number `a` using `den` as denominator in the rational approximation.

Usage:

```
PaillierPublicKey$add_real(den, a, b)
```

Arguments:

`den` the denominator to use for rational approximations

`a` the *real* message, a list consisting of the integer and fractional parts named `int` and `frac` respectively

`b` a simple real number

Method `sub_real()`**:** Return the difference `a - b` of an encrypted real message `a`, a list consisting of a encrypted integer part (named `int`) and an encrypted fractional part (named `frac`), and a real number `b` using `den` as denominator in the rational approximation.

Usage:

```
PaillierPublicKey$sub_real(den, a, b)
```

Arguments:

`den` the denominator to use for rational approximations

`a` the *real* message, a list consisting of the integer and fractional parts named `int` and `frac` respectively

`b` a simple real number

Method `mult()`**:** Return the product of two encrypted messages `a` and `b`

Usage:

```
PaillierPublicKey$mult(a, b)
```

Arguments:

`a` a message

`b` another message

Returns: the product of `a` and `b`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PaillierPublicKey$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`PaillierPrivateKey()` which goes hand-in-hand with this object

`random.bigz` *Return a random big number using the cryptographically secure random number generator from in the `sodium` package.*

Description

Return a random big number using the cryptographically secure random number generator from in the `sodium` package.

Usage

```
random.bigz(nBits)
```

Arguments

`nBits`	the number of bits, which must be a multiple of 8, is not checked for efficiency.

Index

homomorpheR, 2
homomorpheR-package (homomorpheR), 2

PaillierKeyPair, 3
PaillierPrivateKey, 4
PaillierPrivateKey(), *4*, *7*
PaillierPublicKey, 5
PaillierPublicKey(), *4, 5*

R6::R6Class(), *3–5*
random.bigz, 8