Package: fhe (via r-universe)

July 19, 2026

Type Package

Title Fully Homomorphic Encryption

Version 0.7.0

Maintainer Louis Aslett <louis.aslett@durham.ac.uk>

Description A reference implementation of homomorphic encryption schemes. These are encryption schemes which allow limited operations (such as addition and multiplication) to be performed on the cipher text directly without decrypting first. In other words, if you encrypt the values 2 and 3 and ``add'' their cipher texts then decrypting renders 5. At present, the scheme of Fan and Vercauteren (2012) <https://eprint.iacr.org/2012/144> is provided. This package was originally named {HomomorphicEncryption} and available in source form. It is the package described in Aslett et al. (2015) <doi:10.48550/arXiv.1508.06574>, but due to a name collision created by a later package we have renamed to {fhe}.

License GPL-2

ByteCompile yes

Imports Rcpp (>= 1.0.12), methods, RcppParallel

LinkingTo Rcpp, RcppParallel

Depends R (>= 4.2.0), gmp

Suggests testthat, microbenchmark

RcppModules FandV

SystemRequirements GNU make; flint: flint-devel (rpm), libflint-dev (deb), flint (homebrew)

Encoding UTF-8

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libflint-dev libgmp3-dev make

Repository https://louisaslett.r-universe.dev

Date/Publication 2026-07-19 13:31:04 UTC

RemoteUrl https://github.com/louisaslett/fhe

RemoteRef HEAD

RemoteSha b7a2cbd750d27b8add11a0e3bd500dbfbd180336

Contents

`fhe-package` *Fully Homomorphic Encryption*

Description

This package provides easy to use implementations of some homomorphic encryption schemes. An encryption scheme is said to be homomorphic when certain functions can be applied directly to the cipher text in such a way that decrypting the result renders the same answer as if the function had been applied to the unencrypted data.

Details

Package:	fhe
Type:	Package
Version:	0.6.0
Date:	2024-03-01
License:	GPL-2

Author(s)

Louis Aslett

Maintainer: Louis Aslett <louis.aslett@durham.ac.uk>

References

Aslett, L. J. M., Esperança, P. M. and Holmes, C. C. (2015), A review of homomorphic encryption and software tools for encrypted statistical machine learning. Technical report, University of Oxford. https://arxiv.org/abs/1508.06574

Examples

```
# Generate cryptographic parameters
p <- pars("FandV")

# Create public/private keypair
keys <- keygen(p)

# Encrypt the values 2 and 3
ct1 <- enc(keys$pk, 2)
ct2 <- enc(keys$pk, 3)

# Homomorphically add the cipertexts together
ct <- ct1 + ct2

# Decrypt to 5, the result of applying + to plain messages
dec(keys$sk, ct)
```

`Arithmetic (+,-,*)` *Homomorphic operations on ciphertexts*

Description

These special operations overload the standard behaviour of the arithmetic operations to work instead homomorphically on ciphertexts.

Details

Some arithmetic operations have been overloaded allowing use of the standard operators on ciphertexts. Therefore, if `ct1` and `ct2` are two ciphertexts then the following will be handled automatically:

`ct1 + ct2`
`ct1 - ct2`
`ct1 * ct2`

Note that not all homomorphic encryption schemes will support all operations. Also, it is important to note that typically homomorphic operations cause an increase in the noise within a ciphertext. Once a certain number of operations have taken place the cipertext may no longer correctly decrypt. If a scheme is *fully* homomorphic, then it may be possible to apply a bootstrapping procedure which reduces the noise.

Value

A new ciphertext with the encrypted result of applying the operation to the messages held by the two original cipertexts.

Examples

```
p <- pars("FandV")
keys <- keygen(p)
ct1 <- enc(keys$pk, 2)
ct2 <- enc(keys$pk, 3)
ctAdd <- ct1 + ct2
ctSub <- ct1 - ct2
ctMul <- ct1 * ct2

# Decrypt to 5, -1 and 6: the result of applying +, - and * to plain messages
dec(keys$sk, ctAdd)
dec(keys$sk, ctSub)
dec(keys$sk, ctMul)
```

---

dec *Decrypt a ciphertext*

Description

This decrypts an integer message which has been encrypted under one of the homomorphic schemes supported by this package.

Usage

```
dec(sk, ct)
```

Arguments

sk	a private/secret key for any scheme as generated by the keygen. function.
ct	a ciphertext as produced from a call to enc.

Details

Note that the scheme specified by the private key, sk, and the ciphertext, ct, must match.

If a symmetric key scheme is being used, then the secret key should be provided for the sk argument.

Value

The decrypted integer message. If the value is in the range of a standard integer in R (-2147483647 to 2147483647) then an integer will be returned, otherwise a bigz big integer object from the gmp package will be returned.

Author(s)

Louis Aslett

See Also

enc to encrypt messages to ciphertexts which this function decrypts.

Examples

```
p <- pars("FandV")
keys <- keygen(p)
ct <- enc(keys$pk, 1)
dec(keys$sk, ct)
```

---

enc *Encrypt a message*

Description

This encrypts an integer message under one of the homomorphic encryption schemes supported by this package.

Usage

```
enc(pk, m)
```

Arguments

pk	a public key for any scheme as generated by the keygen. function.
m	an integer to be encrypted. Note that the permissable range of values for m is dependent on the scheme and the parameters of the scheme. m may even be resticted to as little as {0,1}, for example in binary encryption schemes.

Details

The scheme to use is determined by the public key which contains the parameter values for that scheme as part of it.

If a symmetric key scheme is being used, then the secret key should be provided for the pk argument.

Value

A ciphertext under the encryption scheme, encrypted using the public key provided.

Author(s)

Louis Aslett

See Also

keygen to create public/private keypairs; dec to decrypt the ciphertext generated by this function.

Examples

```
p <- pars("FandV")
keys <- keygen(p)
ct <- enc(keys$pk, 1)
dec(keys$sk, ct)
```

FandV *Fan and Vercauteren encryption scheme*

Description

The Fan and Vercauteren scheme is implemented in this package.

Details

Description of the scheme.

Examples

```
# Benchmark the performance of the scheme
library(microbenchmark)
p <- pars("FandV")
microbenchmark({ keys <- keygen(p) }, unit="ms")
microbenchmark({ ct1 <- enc(keys$pk, 2) }, unit="ms")
ct2 <- enc(keys$pk, 3)
microbenchmark({ ct1 + ct2 }, unit="ms")
microbenchmark({ ct1 * ct2 }, unit="ms")
microbenchmark({ dec(keys$sk, ct1) }, unit="ms")
```

---

keygen *Generate cryptographic keys*

Description

This function will generate cryptographic keys to enable encryption, decryption and possibly other operations (such as relinearisation or bootstrapping) for the homomorphic encryption schemes supported in this package.

Usage

```
keygen(p)
```

Arguments

p	a parameters object as produced by the pars function.

Details

The scheme to be used is determined by the type of the parameters object, p.

Value

A list object containing the keys will be returned

Depending on the scheme specified by the parameters object the exact structure will vary. For example, a symmetric key scheme will only return a private key, whereas a public key scheme will return a public and private keypair. Certain schemes may include additional keys for operations such as relinearisation or bootstrapping. The keys will be named pk (public), sk (private), rlk (relinearisation), bk (bootstrapping) in the list object if they are part of the scheme.

Author(s)

Louis Aslett

See Also

pars for generating the parameters for a scheme; enc for encrypting messages using the keys generated by this function.

Examples

```
p <- pars("FandV")
keys <- keygen(p)

# Look at public key and encrypt
keys$pk
ct <- enc(keys$pk, 1)

# Look at private key and decrypt
keys$sk
dec(keys$sk, ct)

# Obviously a different private key won't be able to decrypt
keys2 <- keygen(p)
keys2$sk
dec(keys2$sk, ct)
```

---

pars *Setup encryption scheme parameters*

Description

Use this function to create an encryption scheme parameters object.

Usage

```
pars(scheme, ...)
```

Arguments

scheme	the scheme for which to create a parameter object. Currently only Fan and Vercauteren's scheme is supported by specifying "FandV".
...	pass the specific options for the chosen scheme as named arguments to override any default values. See the details section for options for encryption schemes currently implemented.

Details

Currently only the scheme of Fan and Vercauteren ("FandV") is implemented.

For "FandV" you may specify:

d power of the cyclotomic polynomial ring (default 4096);

sigma the standard deviation of the discrete Gaussian used to induce a distribution on the cyclotomic polynomial ring (default 16.0);

qpow the power of 2 to use for the coefficient modulus (default 128);

t the value to use for the message space modulus (default 32768).

This function simply sets up the parameters which must be specified to use a particular encryption scheme. Using the scheme then requires generating cryptographic keys.

Author(s)

Louis Aslett

References

Fan, J., & Vercauteren, F. (2012). Somewhat Practical Fully Homomorphic Encryption. IACR ePrint. Retrieved from https://eprint.iacr.org/2012/144

See Also

parsHelp for automatic assistance selecting parameters which achieve a certain security level and multiplicative depth.

keygen to generate encryption keys using these parameters.

Examples

```
# Simplest example
p <- pars("FandV")
keys <- keygen(p)
ct <- enc(keys$pk, 1)
dec(keys$sk, ct)

# Change degree of cyclotomic polynomial used for ciphertext
```

```
p <- pars("FandV", d=2048)
keys <- keygen(p)
ct <- enc(keys$pk, 1)
dec(keys$sk, ct)
```

`parsHelp` *Encryption parameter selection*

Description

Use this function to help choose encryption parameters for your scheme.

Usage

```
parsHelp(scheme, ...)
```

Arguments

`scheme`	the scheme for which to get parameter help. Currently only Fan and Vercauteren's scheme is supported by specifying `"FandV"`.
`...`	pass the specific options for the chosen scheme as named arguments. See the details section for options for encryption schemes currently implemented.

Details

This function provides assistance with selecting parameter values for the encryption scheme you wish to use.

Currently only the scheme of Fan and Vercauteren (`"FandV"`) is implemented.

For `"FandV"` you may specify:

`lambda` the security level required (bits), default is 80;

`max` the largest absolute value you will need to store encrypted, default is 1000;

`L` the deepest multiplication depth you need to be able to evaluate encrypted, default is 4.

The security level in bits relates to the approximate number of operations which would be required to break the cipher text, i.e. $O(2^{\lambda})$ operations.

Ensuring the depth of multiplication operations you require is based on an overwhelming probability of being able to correctly decrypt after so many operations by roughly bounding the possible noise growth. In most situations it is very conservative and additional multiplications are possible, so this should be considered only a lower bound on the number of multiplies required. If computational performance is a concern this should be treated only as a good starting point from which the relevant parameters can be eased off, testing more thoroughly by simulation.

For the scheme of Fan and Vercauteren (`"FandV"`): the security level is based on the analysis of Lindner and Peikert (2011) which is known to be quite conservative; and the multiplicative depth bound is based on the analysis of Lepoint and Naehrig (2014). If the multiplicative depth lower bound is too conservative, then for computational performance reduce the qpow parameter from this recommended starting point.

Author(s)

Louis Aslett

References

Fan, J., & Vercauteren, F. (2012). Somewhat Practical Fully Homomorphic Encryption. IACR ePrint. Retrieved from https://eprint.iacr.org/2012/144

Lepoint, T., & Naehrig, M. (2014). A comparison of the homomorphic encryption schemes FV and YASHE. In *Progress in Cryptology–AFRICACRYPT 2014* (pp. 318-335).

Lindner, R., & Peikert, C. (2011). Better key sizes (and attacks) for LWE-based encryption. In *Topics in Cryptology–CT-RSA 2011* (pp. 319-339).

See Also

pars to manually choose parameters.

keygen to generate encryption keys using these parameters.

Examples

```
# Want 128-bit security with 6 multiplies deep
# to evaluate 7 factorial
p <- parsHelp("FandV", lambda=128, L=6)
keys <- keygen(p)
ct <- enc(keys$pk, 1:7)
dec(keys$sk, prod(ct))
factorial(7)

# ... but notice this is quite conservative
# The following will usually give the right answer too
ct <- enc(keys$pk, 1:8)
dec(keys$sk, prod(ct))
factorial(8)

# ... however, another multiply will usually fail:
ct <- enc(keys$pk, 1:9)
dec(keys$sk, prod(ct))
factorial(9)
```

---

saveFHE *Save and load keys and ciphertexts*

Description

Use these functions rather than the base R functions for saving and loading of ciphertexts and keys. Note that this is due to a limitation in Rcpp, so the save, save.image and saveRDS base R functions are overriden to warn if they are used incorrectly on ciper text objects.

Usage

```
saveFHE(object, file)

loadFHE(file)
```

Arguments

object	the ciphertext or keys to be saved
file	the filename to save (load) the object to (from)

Details

These functions provide a way to save ciphertexts and keys generated by this package to a file in such a way that they can later be restored to another R session (possibly on a different machine).

At present no compression is performed so for long-term storage or transmission over a network it would be advisable to gzip the files after creation.

Examples

```
p <- pars("FandV")
keys <- keygen(p)
ct1 <- enc(keys$pk, 2)
## Not run: saveFHE(ct1, "~/myCipherText.fhe")

# Start a new session or transfer to another machine
## Not run: ct1 <- loadFHE("~/myCipherText.fhe")
```

Vectors *Vectors of cipher texts*

Description

Vectors of cipher texts can be seemlessly created and manipulated in the same way one normally works with vectors in R.

Details

Vectors of cipher texts can be seemlessly created either at encryption time by passing a vector message argument to enc, or after encryption using the standard concatenation operator c:

```
enc(keys$pk, c(m1, m2, ...))
c(ct1, ct2)
```

Standard operations which can be used on regular vectors can also be used on vectors of cipher texts, where the scheme supports it:

```
ct1 + ct2
ct1 - ct2
ct1 * ct2
```

```
ct1 %*% ct2
sum(ct1)
prod(ct1)
```

As with regular scalar operations, note that not all homomorphic encryption schemes will support all vector operations. Also, it is important to note that typically homomorphic operations cause an increase in the noise within a ciphertext. Once a certain number of operations have taken place the cipertext may no longer correctly decrypt. If a scheme is *fully* homomorphic, then it may be possible to apply a bootstrapping procedure which reduces the noise.

Examples

```
p <- pars("FandV")
keys <- keygen(p)
ct1 <- enc(keys$pk, c(2,3))
ct2 <- enc(keys$pk, c(4,5))
ctConcat <- c(ct1, ct2)
ctAdd <- ct1 + ct2
ctSub <- ct1 - ct2
ctMul <- ct1 * ct2
ctIP <- ct1 %*% ct2

# Decrypt to the equivalent of the above operations applied to the vectors
# c(2,3) and c(4,5)
dec(keys$sk, ctConcat)
dec(keys$sk, ctAdd)
dec(keys$sk, ctSub)
dec(keys$sk, ctMul)
dec(keys$sk, ctIP)
```

Index