

fx-check_multisig

Link

<https://github.com/bitcoin/bitcoin/blob/e25af11225d9d94ecf7068bf7a9a359268786fbe/contrib/binaries/verify.py#L267C1-L267C1>

▼ Code

```
def check_multisig(sums_file: str, sigfilename: str, args: argparse.Namespace) -> t.Tuple[int,
    retval, output = verify_with_gpg(sums_file, sigfilename)

    if args.verbose:
        log.info(f"gpg output:\n{indent(output)}")

    good, unknown, bad = parse_gpg_result(output.splitlines())

    if unknown and args.import_keys:
        # Retrieve unknown keys and then try GPG again.
        for unsig in unknown:
            if prompt_yn(f" ? Retrieve key {unsig.key} ({unsig.name})? (y/N) "):
                ran = subprocess.run(
                    ["gpg", "--keyserver", args.keyserver, "--recv-keys", unsig.key])

                if ran.returncode != 0:
                    log.warning(f"failed to retrieve key {unsig.key}")

            # Reparse the GPG output now that we have more keys
            retval, output = verify_with_gpg(sums_file, sigfilename)
            good, unknown, bad = parse_gpg_result(output.splitlines())

    return retval, output, good, unknown, bad
```

▼ Documentation

```
# check signature
#
# We don't write output to a file because this command will almost certainly
# fail with GPG exit code '2' (and so not writing to --output) because of the
# likely presence of multiple untrusted signatures.
```

Prompt

As a code documentation assistant, you are programmed to document at the function level of the given code snippet. Your approach involves placing comments directly under the def statement of the function. The output should be the entire code along with the documentation of

the function written. Code is as follows:

```
"""
def check_multisig(sums_file: str, sigfilename: str, args: argparse.Namespace) -> t.Tuple[int, str,
t.List[SigData], t.List[SigData], t.List[SigData]]:
    retval, output = verify_with_gpg(sums_file, sigfilename)

    if args.verbose:
        log.info(f"gpg output:\n{indent(output)}")

    good, unknown, bad = parse_gpg_result(output.splitlines())

    if unknown and args.import_keys:
        # Retrieve unknown keys and then try GPG again.
        for unsig in unknown:
            if prompt_yn(f" ? Retrieve key {unsig.key} ({unsig.name})? (y/N) "):
                ran = subprocess.run(
                    ["gpg", "--keyserver", args.keyserver, "--recv-keys", unsig.key])

                if ran.returncode != 0:
                    log.warning(f"failed to retrieve key {unsig.key}")

        # Reparse the GPG output now that we have more keys
        retval, output = verify_with_gpg(sums_file, sigfilename)
        good, unknown, bad = parse_gpg_result(output.splitlines())

    return retval, output, good, unknown, bad
"""
```

Responses

ChatGPT

```
import subprocess
import argparse
import typing as t

def check_multisig(sums_file: str, sigfilename: str, args: argparse.Namespace) -> t.Tuple[int, s
"""
    Verify digital signatures using GPG and retrieve unknown keys if needed.

    Args:
        sums_file (str): The file containing checksums to be verified.
        sigfilename (str): The file containing digital signatures.
        args (argparse.Namespace): Command-line arguments and options.
```

```

Returns:
    Tuple[int, str, List[SigData], List[SigData], List[SigData]]: A tuple containing:
    - retval (int): The return code from the GPG verification process.
    - output (str): The output message from the GPG verification process.
    - good (List[SigData]): List of successfully verified signatures.
    - unknown (List[SigData]): List of signatures with unknown keys.
    - bad (List[SigData]): List of signatures that failed verification.
"""
retval, output = verify_with_gpg(sums_file, sigfilename)

if args.verbose:
    log.info(f"gpg output:\n{indent(output)}")

good, unknown, bad = parse_gpg_result(output.splitlines())

if unknown and args.import_keys:
    # Retrieve unknown keys and then try GPG again.
    for unsig in unknown:
        if prompt_yn(f" ? Retrieve key {unsig.key} ({unsig.name})? (y/N) "):
            ran = subprocess.run(
                ["gpg", "--keyserver", args.keyserver, "--recv-keys", unsig.key])

            if ran.returncode != 0:
                log.warning(f"failed to retrieve key {unsig.key}")

    # Reparse the GPG output now that we have more keys
    retval, output = verify_with_gpg(sums_file, sigfilename)
    good, unknown, bad = parse_gpg_result(output.splitlines())

return retval, output, good, unknown, bad

```

GPT4

```

import argparse
import subprocess
import typing as t

# Function Documentation
#
# Function Name: check_multisig
#
# Purpose:
# This function checks multiple digital signatures on a file. It first verifies the file using G
#
# Parameters:
# - sums_file (str): The path to the file containing checksums.
# - sigfilename (str): The path to the signature file to verify.
# - args (argparse.Namespace): A namespace object containing various arguments. Relevant argumen
# - verbose (bool): Whether to log verbose output.
# - import_keys (bool): Whether to attempt key import for unknown signatures.
# - keyserver (str): The GPG keyserver URL to use for importing keys.
#

```

```

# Returns:
# - Tuple[int, str, list[SigData], list[SigData], list[SigData]]:
#   - int: The return value from the GPG verification process.
#   - str: The output from the GPG verification process.
#   - list[SigData]: A list of 'good' signatures (signatures that were successfully verified).
#   - list[SigData]: A list of 'unknown' signatures (signatures from keys not found in the keyri
#   - list[SigData]: A list of 'bad' signatures (signatures that failed verification).
#
# Exceptions:
# - This function may raise exceptions related to file handling or subprocess execution.

def check_multisig(sums_file: str, sigfilename: str, args: argparse.Namespace) -> t.Tuple[int, s
    retval, output = verify_with_gpg(sums_file, sigfilename)

    if args.verbose:
        log.info(f"pgp output:\n{indent(output)}")

    good, unknown, bad = parse_gpg_result(output.splitlines())

    if unknown and args.import_keys:
        # Retrieve unknown keys and then try GPG again.
        for unsig in unknown:
            if prompt_yn(f" ? Retrieve key {unsig.key} ({unsig.name})? (y/N) "):
                ran = subprocess.run(
                    ["pgp", "--keyserver", args.keyserver, "--recv-keys", unsig.key])

                if ran.returncode != 0:
                    log.warning(f"failed to retrieve key {unsig.key}")

            # Reparse the GPG output now that we have more keys
            retval, output = verify_with_gpg(sums_file, sigfilename)
            good, unknown, bad = parse_gpg_result(output.splitlines())

    return retval, output, good, unknown, bad

```

Bard

```

def check_multisig(sums_file: str, sigfilename: str, args: argparse.Namespace) -> t.Tuple[int, s
    """
    Checks the validity of a multi-signature file.

    Args:
        sums_file (str): The path to the sums file.
        sigfilename (str): The path to the signature file.
        args (argparse.Namespace): The arguments parsed from the command line.

    Returns:
        t.Tuple[int, str, t.List[SigData], t.List[SigData], t.List[SigData]]: A tuple containing
            the GPG output, a list of good signatures, a list of unknown signatures, and a list
    """

    retval, output = verify_with_gpg(sums_file, sigfilename)

```

```

if args.verbose:
    log.info(f"gpg output:\n{indent(output)}")

good, unknown, bad = parse_gpg_result(output.splitlines())

if unknown and args.import_keys:
    # Retrieve unknown keys and then try GPG again.
    for unsig in unknown:
        if prompt_yn(f" ? Retrieve key {unsig.key} ({unsig.name})? (y/N) "):
            ran = subprocess.run(
                ["gpg", "--keyserver", args.keyserver, "--recv-keys", unsig.key])

            if ran.returncode != 0:
                log.warning(f"failed to retrieve key {unsig.key}")

    # Reparse the GPG output now that we have more keys
    retval, output = verify_with_gpg(sums_file, sigfilename)
    good, unknown, bad = parse_gpg_result(output.splitlines())

return retval, output, good, unknown, bad

```

LLama2

```

def check_multisig(sums_file: str, sigfilename: str, args: argparse.Namespace) -> t.Tuple[int, s
"""
Check multi signature file.

Parameters:
- sums_file: Path to the file containing the sums to verify.
- sigfilename: Path to the signature file.
- args: Arguments object containing various flags and options.

Returns:
- retval: Return value from verify_with_gpg.
- output: Output from verify_with_gpg.
- good: List of valid signatures.
- unknown: List of signatures that couldn't be verified due to missing keys.
- bad: List of invalid signatures.
"""
retval, output = verify_with_gpg(sums_file, sigfilename)

if args.verbose:
    log.info(f"gpg output:\n{indent(output)}")

good, unknown, bad = parse_gpg_result(output.splitlines())

if unknown and args.import_keys:
    # Retrieve unknown keys and then try GPG again.
    for unsig in unknown:
        if prompt_yn(f" ? Retrieve key {unsig.key} ({unsig.name})? (y/N) "):
            ran = subprocess.run(
                ["gpg", "--keyserver", args.keyserver, "--recv-keys", unsig.key])

```

```

        if ran.returncode != 0:
            log.warning(f"failed to retrieve key {unsig.key}")

        # Reparse the GPG output now that we have more keys
        retval, output = verify_with_gpg(sums_file, sigfilename)
        good, unknown, bad = parse_gpg_result(output.splitlines())

    return retval, output, good, unknown, bad

```

StarChat

```

"""
This function checks whether a set of signatures on a file match using gpg. It takes as input th

The function first uses gpg to verify the signatures against the given public keys stored locall

Args:
    sums_file (str): Path to the signature file.
    sigfilename (str): Path to the signed file.
    args (argparse.Namespace): Command line arguments passed by the user.
Returns:
    tuple: A tuple containing four elements:
        1. An integer indicating success or failure of the operation.
        2. Output generated by running gpg.
        3. List of Signature objects representing valid signatures.
        4. List of Signature objects representing unknown keys.
        5. List of Signature objects representing invalid signatures.
"""

def check_multisig(sums_file: str, sigfilename: str, args: argparse.Namespace) -> \
    t.Tuple[int, str, t.List[SigData], t.List[SigData], t.List[SigData]]:
    """Check multisignature."""
    retval, output = verify_with_gpg(sums_file, sigfilename)

    if args.verbose:
        log.info(f"gpg output:\n{indent(output)}")

    good, unknown, bad = parse_gpg_result(output.splitlines())

    if unknown and args.import_keys:
        # Retrieve unknown keys and then try GPG again.
        for unsig in unknown:
            if prompt_yn(f"? Retrieve key {unsig.key} ({unsig.name})? (y/N) "):
                ran = subprocess.run(["gpg", "--keyserver", args.keyserver,
                                      "--recv-keys", unsig.key])

                if ran.returncode != 0:
                    log.warning(f"Failed to retrieve key {unsig.key}.")

        # Reparse the GPG output now that we have more keys

```

```
retval, output = verify_with_gpg(sums_file, sigfilename)
good, unknown, bad = parse_
```
