data_treatment_theoretical

June 22, 2021

[1]:
```
import sys

# make sure all relevant libraries are installed
!{sys.executable} -m pip install pandas pyserial pyyaml
```

```
Requirement already satisfied: pandas in c:\users\kenne\appdata\local\packages\p
ythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-
packages\python39\site-packages (1.2.4)
Requirement already satisfied: pyserial in c:\users\kenne\appdata\local\packages
\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-
packages\python39\site-packages (3.5)
Requirement already satisfied: pyyaml in c:\users\kenne\appdata\local\packages\p
ythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-
packages\python39\site-packages (5.4.1)
Requirement already satisfied: pytz>=2017.3 in c:\users\kenne\appdata\local\pack
ages\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-
packages\python39\site-packages (from pandas) (2021.1)
Requirement already satisfied: numpy>=1.16.5 in c:\users\kenne\appdata\local\pac
kages\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-
packages\python39\site-packages (from pandas) (1.20.3)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\kenne\appdata\
local\packages\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\loca
l-packages\python39\site-packages (from pandas) (2.8.1)
Requirement already satisfied: six>=1.5 in c:\users\kenne\appdata\local\packages
\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-
packages\python39\site-packages (from python-dateutil>=2.7.3->pandas) (1.16.0)
WARNING: You are using pip version 21.1.1; however, version 21.1.2 is available.
You should consider upgrading via the 'C:\Users\kenne\AppData\Local\Microsoft\Wi
ndowsApps\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\python.exe -m pip
install --upgrade pip' command.
```

[2]:
```
# import all functions used for processing data
import pandas as pd
from SerialLogger import write_logs
from Analyzelogs import load_log, categorize_logfiles, calculate_all, \
 calculate_cost, extract_avg

# display values in dataframe table setting to 2 significant digit
```

```
pd.set_option('display.float_format', '{:.2E}'.format)
```

[ ]:
```
# run this cell each time you log primitive(s) to write the .log files
# from serial communication
write_logs("./test_results/")
```

[ ]:
```
# categorize device name here, do multiple if multiple devices.
#


root_dir = "./test_results" # TODO set to your workspace root or use relative␣
 ↪path
device_dir = "your_device_folder" # TODO set to the folder containing logfiles

device_dict = categorize_logfiles(f"{root_dir}/{device_dir})
```

[ ]:
```
# create index, the full name of each primitive being benchmarked
 #TODO fill in with your primitives
index = ["XOR" , "RNG"]

# create corresponding list to the index that contains the raw log name of␣
 ↪primitive
#TODO fill in with your own primitive log_type names matching index, add more␣
 ↪primitives
primitives = ["xor_256", "rng_256"]

# for each primitive in the list, calculate the avg
# for each primitive and store in a a list/column by the order given in␣
 ↪primitives
# so devices_avgs[0] = xor avg, devices_avg[1] = rng avg
device_avgs = extract_avg(device_dict, primitives)

# create primitive execution time table, column for each device
# TODO put in more devices if you want

primitives_avg = pd.DataFrame({"DEVICE" : device_avgs},
index=index
)
print(primitives_avg)

# store result in .file ./tables/primitives_avg.tex
primitives_avg.to_latex("./tables/primitives_avg.tex")
```

[ ]:
```
#calculate implemented scheme avg execution time
# just like primitive avg execution time

implemented_schemes_index = ["ECIES"]
implemented_schemes = ["ecies"]

device_avg_schemes = extract_avg(device_dict, implemented_schemes)

# assemble dataframe again, can also do this in previous cell
# all in one go instead.

implemented_schemes_avg = pd.DataFrame({"DEVICE" : device_avg_schemes},
index = implemented_schemes_index)
 print(implemented_schemes_avg)

 # store result in a .tex file
 implemented_schemes_avg.to_latex('./tables/implented_schemes_avg.tex')
```

[ ]:
```
# calcualte theoretical scheme avg

# index to use in assembled dataframe
scheme_index = ["Scheme1", "Scheme2"
 ]

# primitives and amounts they're used for each scheme
# stored as tuples in a list

scheme1_cost = [["SHA256",3], ["XOR", 2]]
scheme2_cost = [["SHA3-256", 7],  ["RNG"], 3]
# put these in a list for iteration
schemes = [scheme1_cost, scheme2_cost]

# calculate cost series using AVG execution time dataframe
# to lookup primitive execution time

device_theoretical_schemes = calculate_all(primitives_avg, schemes, "DEVICE")

# assemble dataframe/table by using columns as costs
# for each device, and scheme names as index

schemes_theoretical = pd.DataFrame({"DEVICE" : device_theoretical_schemes},
index=scheme_index)
print(schemes_theoretical)
 # store result
 schemes_theoretical.to_latex("./tables/theoretical_schemes_avg.tex")
```