ABE Scalability

Cybersecurity practice to measure the scalability and complexity of Attribute-Based Encryption

Alejandro Pérez Bueno (100429952@alumnos.uc3m.es)

April 25th, 2022

Table of Contents

Introduction

The goal for this project is to get familiar with the `cpabe` tools for attribute-based encryption. We are asked to code an algorithm that creates various users and their secret keys made from a set of attributes, and later encrypts and decrypts a 5MB pdf file several times. In this practice we will try different combinations of the number of users, attributes and repetitions. The idea is to measure how long it takes to encrypt and decrypt the pdf file depending on these values.

Code Implementation

I implemented the algorithm for encryption in `C`, since it is what I'm most comfortable coding in. The project includes a `Makefile` with the necessary compliation rules. It will create the executable file `cp_abe` inside a `bin/` folder. This program always takes three arguments:

```
usage: cp_abe <n_users> <n_attributes> <n_repetitions>
```

Here is a quick overview of the functions I created:

Function	Description
`parse_args`	Reads arguments from `argv` and saves number of users, attributes and repetitions
`create_dirs`	Creates `tests/` folder where all users' folders will be, and runs `cpabe-setup` in `tests/master/`
`config_dirs`	Creates folder for every user and creates attributes for all of them
`get_time`	Returns current `epoch` time (seconds since 1970)
`get_str`	Adds given index to provided string (eg `"user_1"`, `"attr_3"`, `"file_n"`)
`wrap_cmd`	Joins up to three strings together (used to create command strings)
`crypt_pdf`	For every `n_repeat`, encrypts the pdf and then every user decrypts it
`encrypt_pdf`	Encrypts pdf file `file.pdf` with all attributes as `file-enc.pdf.cpabe`
`decrypt_pdf`	Decrypts pdf for a given user and saves it to the user's folder
`ft_putstr_fd`	Writes a custom string to a file descriptor with `write`
`ft_atoi`	Converts ascii to int. Reads a string and obtains the equivalent integer value
`ft_itoa`	Converts int to ascii. Reads an int and obtains the equivalent string value
`ft_strdup`	Returns allocated copy of a string
`ft_strjoin`	Joins two strings together in an allocated string
`ft_substr`	Returns allocated substring (copies n bytes from `start` of the given string)
`ft_strlen`	Returns length of a string
`ft_nbrlen`	Returns length of a number
`ft_strlcat`	Copies `n - 1` bytes of a string into another one
`ft_isspace`	Returns `1` if char is a form of space (same as `isspace`)
`ft_putchar_fd`	Writes a char to a file descriptor
`ft_putnbr_fd`	Writes int to a file descriptor

Here are the builtin functions I used and a quick description of what they do. Check their manpages for more information

Function	Description
`system`	Runs a command from the system (used mainly for cpabe commands)
`gettimeofday`	Returns `epoch` in a `timeval` struct
`open`	Opens a file to a file descriptor
`close`	Closes a file descriptor
`write`	Writes n bytes of memory to a file descriptor
`printf`	Prints string to `stdout`
`malloc`	Allocates bytes of memory to a given pointer
`free`	Frees allocated memory from a pointer
`chdir`	Changes the system's current working directory (same as `cd` in a shell)

- General Code description

The code of this practice is hopefully easy to read, but it is actually pretty straightforward. Here is a rough list of the instructions it goes over:

1. Reads arguments from `argv` (argument list) to save `n_usrs`, `n_attrs` and `n_rep`.
2. Deletes `tests/` folder (if present), creates `tests/master/` folder, runs `cpabe-setup` in it.
3. In the `tests/` folder, creates folder for every user (`user_1`, `...`, `user_n`), copies `pub_key` and creates `priv` key with their attributes (`attr_1`, `...`, `attr_n`) using `cpabe-keygen`.
4. Opens log file `log.txt` in the `tests/` folder where basic logging information will be saved.
5. Stores current time before starting encryption.
6. Repeats `n_rep` times the process of encrypting the file `file.pdf` with all attributes and then decrypting it for every user in their user folder as (`file_1.pdf`, `...`, `file_n.pdf`)
7. Stores current time after encryption.
8. Prints `end_time - start_time`, closes `log.txt` and exits

Testing the Algorithm

For this part, we will take a look at the time it takes to encrypt and decrypt a file. We retrieve 20 measurements and compute an average (mean). Then we'll make a graph to better visualize the results.

No. of Users	No. of Attributes	Avg. Execution Time (s)	Total Execution Time (s)
5	5	0.518	11
5	20	0.849	17
20	5	1.664	35
20	20	2.603	54

Note: these values are highly dependant on the processing power of the device running the program. It is only interesting to see the variations in time relative to each other, rather than the actual numbers.

- Key Sizes

To view the key sizes, I thought I'd use something like the following:

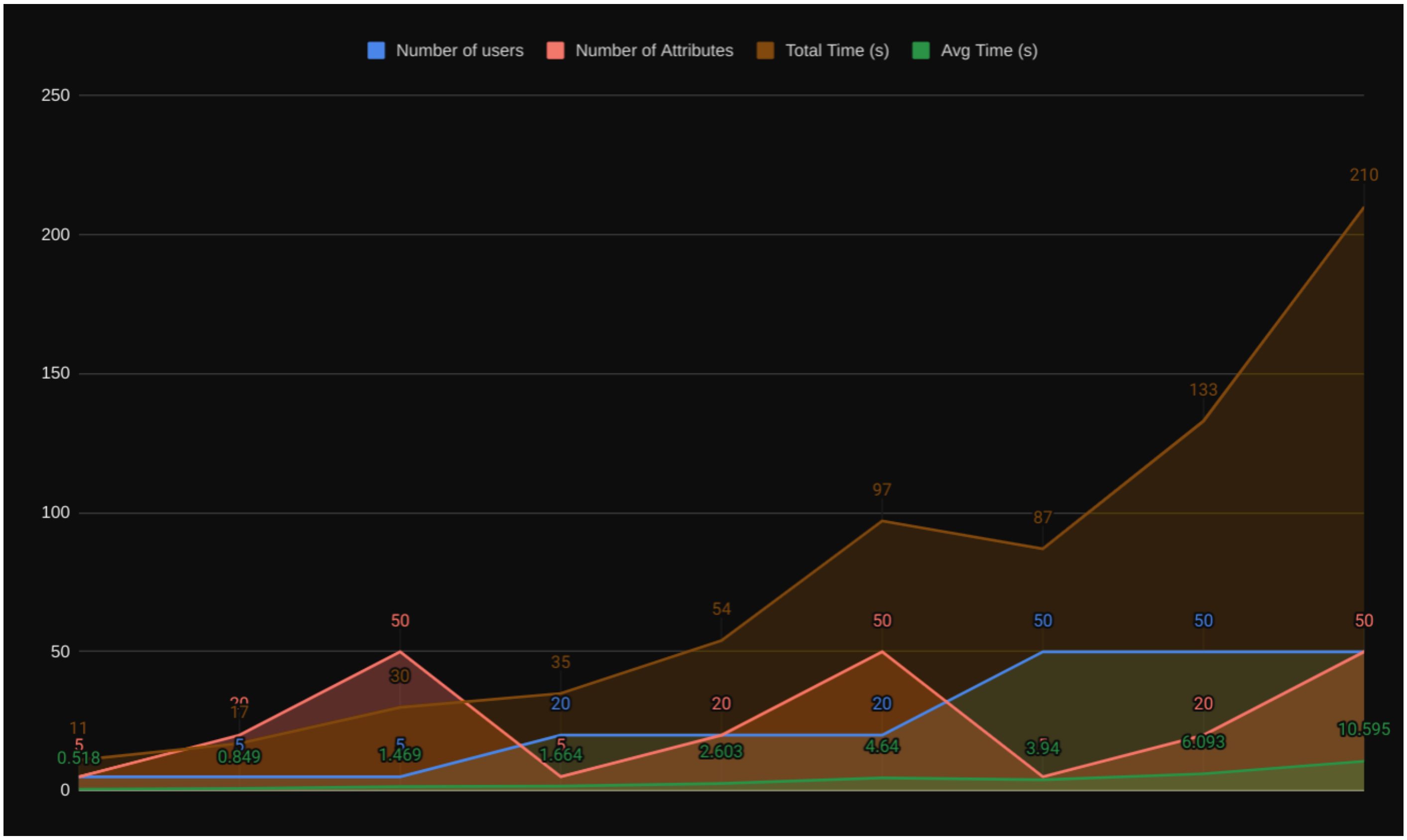
```
cat -e tests/master/master_key | wc -c
```

Master Key Size (bytes): 325

No. of Attributes	Secret Key Size
5	3288
20	12340
50	30771

As we can see, the key size increases very fast as the number of attributes goes up.

Graph



Note: I added a few extra rows of data to the graph for better visualization.

From the graph we can see a clear pattern. As expected, the more users and attributes, the longer it will take to encrypt and decrypt the file. However, we can see that changing the number of attributes doesn't affect the performance of the encryption nearly as much as increasing the number of users does. This is easily seen with the case of 5 users and 50 attributes, which roughly takes 30 (1.469s on average) seconds to finish. However, the inverse case of 50 users and 5 attributes per user takes more than double the time, taking almost 90 seconds (3.94s on average) to complete.

Thus, we can confidently say that it will be computationally less feasible to have 1k users than having 1k attributes per user.

How to Run the Program

- Installation

In order to run this practice, you must install some packages on your system. To build the packages, you must first install these dependencies:

```
sudo apt -y install make gcc g++ autoconf libc6 libpcre3 flex bison libgmp-dev \
libssl-dev libglib2.0-dev help2man
```

Once those dependencies are satisfied, follow these steps to build the required packages on your system (needs root/sudo)

```
# pbc
wget https://crypto.stanford.edu/pbc/files/pbc-0.5.14.tar.gz
tar zxvf pbc-0.5.14.tar.gz; cd pbc-0.5.14
autoconf
./configure
make
sudo make install
cd ..

# libbswabe
wget http://acsc.cs.utexas.edu/cpabe/libbswabe-0.9.tar.gz
tar zxvf libbswabe-0.9.tar.gz; cd libbswabe-0.9
./configure
make
sudo make install
cd ..

# cpabe
wget http://acsc.cs.utexas.edu/cpabe/cpabe-0.11.tar.gz
tar zxvf cpabe-0.11.tar.gz; cd cpabe-0.11
./configure --with-pbc-include=/usr/local/include/pbc --with-pbc-lib=/usr/local/lib
sed -e '67 s/\$1/\$1;/' policy_lang.y > temp
mv temp policy_lang.y
sed -e '89 s/help2man/help2man --no-discard-stderr/' Makefile > temp
mv temp Makefile
make LDFLAGS="-lgmp -lpbc -lcrypto -L/usr/lib/x86_64-linux-gnu -lglib-2.0 -lbswabe -lgmp"
sudo make LDFLAGS="-lgmp -lpbc -lcrypto -L/usr/lib/x86_64-linux-gnu -lglib-2.0 -lbswabe -lgmp" install
cd ..
```

To make things work, you might need to specify the proper path for the `LD_LIBRARY_PATH` environment variable:

```
export LD_LIBRARY_PATH=/usr/local/lib
echo "export LD_LIBRARY_PATH=/usr/local/lib" >> ~/.bashrc
echo "export LD_LIBRARY_PATH=/usr/local/lib" >> ~/.zshrc
```

- Usage

As previously mentioned, this project includes a Makefile with all the needed instructions, here's an overview of the commands you can use:

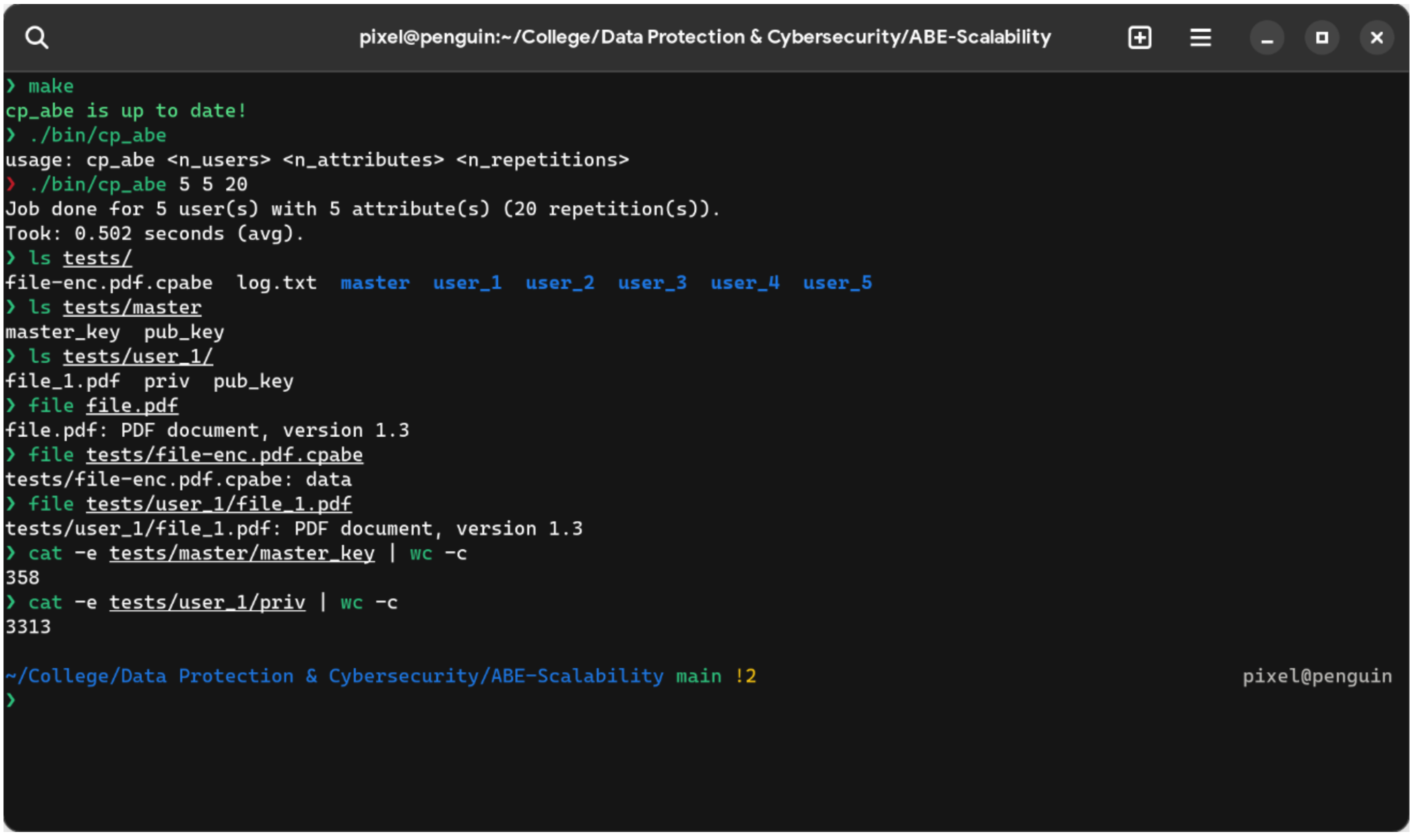
```
make/make all       compiles executable cp_abe to bin/ directory
make clean          cleans object files in obj/ directory
make fclean         calls clean rule and deletes cp_abe executable
make re              cleans up everything and compiles again
make norminette      Runs C linter (norminette) on all required files
```

The Makefile compliles the executable `cp_abe` to a folder called `bin/`. To run the file, specify the path to the executable:

```
./bin/cp_abe 5 5 20
```

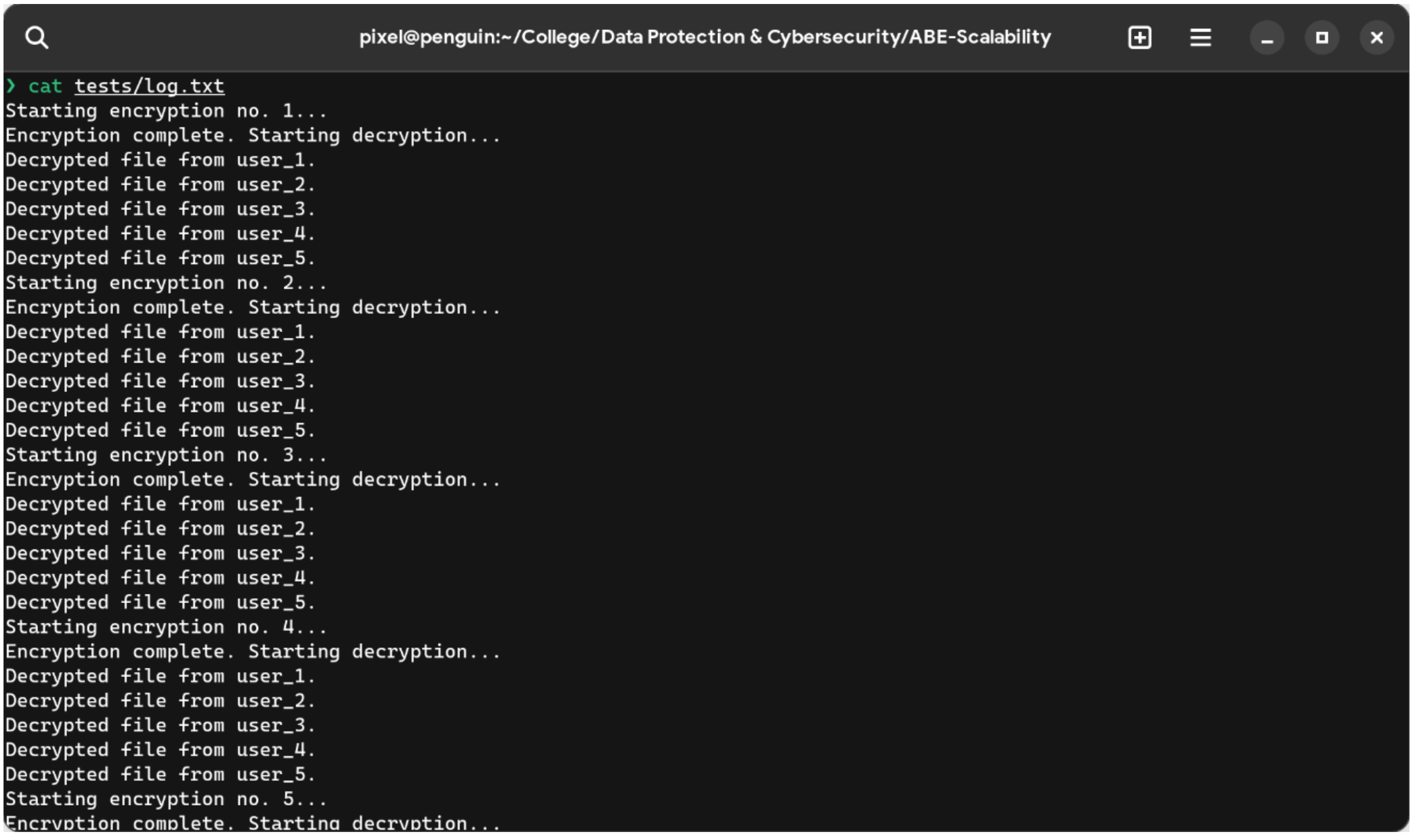
Note: if you get an error saying permission denied, enter `chmod +x ./bin/cp_abe` and try again

Example


pixel@penguin:~/College/Data Protection & Cybersecurity/ABE-Scalability

```
❯ make
cp_abe is up to date!
❯ ./bin/cp_abe
usage: cp_abe <n_users> <n_attributes> <n_repetitions>
❯ ./bin/cp_abe 5 5 20
Job done for 5 user(s) with 5 attribute(s) (20 repetition(s)).
Took: 0.502 seconds (avg).
❯ ls tests/
file-enc.pdf.cpabe  log.txt  master  user_1  user_2  user_3  user_4  user_5
❯ ls tests/master
master_key  pub_key
❯ ls tests/user_1/
file_1.pdf  priv  pub_key
❯ file file.pdf
file.pdf: PDF document, version 1.3
❯ file tests/file-enc.pdf.cpabe
tests/file-enc.pdf.cpabe: data
❯ file tests/user_1/file_1.pdf
tests/user_1/file_1.pdf: PDF document, version 1.3
❯ cat -e tests/master/master_key | wc -c
358
❯ cat -e tests/user_1/priv | wc -c
3313

~/College/Data Protection & Cybersecurity/ABE-Scalability main !2                    pixel@penguin
❯
```


pixel@penguin:~/College/Data Protection & Cybersecurity/ABE-Scalability

```
❯ cat tests/log.txt
Starting encryption no. 1...
Encryption complete. Starting decryption...
Decrypted file from user_1.
Decrypted file from user_2.
Decrypted file from user_3.
Decrypted file from user_4.
Decrypted file from user_5.
Starting encryption no. 2...
Encryption complete. Starting decryption...
Decrypted file from user_1.
Decrypted file from user_2.
Decrypted file from user_3.
Decrypted file from user_4.
Decrypted file from user_5.
Starting encryption no. 3...
Encryption complete. Starting decryption...
Decrypted file from user_1.
Decrypted file from user_2.
Decrypted file from user_3.
Decrypted file from user_4.
Decrypted file from user_5.
Starting encryption no. 4...
Encryption complete. Starting decryption...
Decrypted file from user_1.
Decrypted file from user_2.
Decrypted file from user_3.
Decrypted file from user_4.
Decrypted file from user_5.
Starting encryption no. 5...
Encryption complete. Starting decryption...
```

Summary

All in all, this project was fun to code and it helped me understand the basics of attribute-based encryption and how it scales with larger users and attributes per user :)

April 25th, 2022