FlexList: Optimized Skip List for Secure Cloud Storage

Cryptography, Security, and Privacy Research Group, Koç University

Authors : Ertem Esiner, Adilet Kachkeev, Alptekin Küpçü, Ozan Okumuşoğlu, Öznur Özkasap

The cashlib, which is developed at Brown University, is extended by data structures (Skip List, Authenticated Skip List, Rank Based Authenticated Skip List, and FlexList),
DPDP and FlexDPDP protocols at Koç University Cryptography Laboratory.

If you have the extension folder only, all you need to do is to merge it with cashlib and accept overwrite for 3 files (namely test.cpp, makefile and GroupRSA.h).

This package includes:

- Skip list
- Authenticated Skip List
- Rank-Based Authenticated Skip List
- FlexList (with many optimization algorithms)
- FlexDPDP Protocol Implementation
- Helper Functions For The Protocol
- Necessary Helper Classes for Diff Operations and Network Implementations
- Server Client Implementation (optimized for Planetlab testing)

It also contains tests for usage and debug operations.

A) CASHLIB REQUIREMENTS

Cashlib does not require any installation but has some dependencies.
If those are not installed properly, the "make" command will give errors.

Therefore, you should first install the following:
1) Open SSL development library
2) Boost development libraries (Our Client-Server implementation also depends on Asio library as an addition to the previous cashlib implementation)
3) GNU Multi-Precision Arithmetic Library (GMP)
4) ANTLR development library
5) Git

B) UBUNTU REQUIRED PACKAGE DETAILS

Install the following packages using Synaptic Package Manager with any extra dependencies accepted during installation. No configuration will be necessary.

libssl-dev
libboost-all
libgmp-dev
libantlr-dev
git

The exact library package names might contain version numbers, and hence may differ slightly from what is provided above. Alternatively, you may install libboost-all instead of parts.

C) CASHLIB HOW TO DOCUMENT

In order to run the project, perform the following steps:

1) Go to directory where cashlib is copied.

2) Set your directory to ../cashlib/src then type make (make -j4 will run compilation on 4 cores).

3) When the make operation is done type: ./test .

You will see a list of tests. Some of them are just for data gathering, some are illustrations of protocol usage and some are prototypes.

D) CASHLIB TEST DESCRIPTIONS

Tests starting from 0 to 22 are basic cashlib tests: for more information please visit https://github.com/brownie.

Remaining test descriptions can be found below:

23: SL insert time test

24: SL remove time test

25: SL modify time test

26: SL search time test

- These 4 tests are meant to measure time spent for these operations.

27: Skip List Basic Test Menu

- This test allows you to test 3 types of skip list by adding one by one or letting the test try some random insert remove modify operations.

28: verifyInsertTestDEBUG

29: verifyDeleteTestDEBUG

- These tests are debugging functions that we use while implementing.

30: Rank-based Skip List bottom up insertion test

- Inserts some blocks to the skip list in a bottom up manner.

31: FlexList simple test

- Initial tests for a FlexList. First usage of the idea.

32: DPDP Test with FlexList

- The very same DPDP test using FlexList now. We used this test to have some data for our graphs. This tests can be used to understand how protocol works. This method can be thought as a documentation and not meant to be run now.

33: Leaf level connection

- With this test one can see how leaf level connection works. By adding some arbitrary nodes to the list, this method prints all leaf level nodes by traversing through the leaf level of the FlexList.

34, 35:> TO RUN A SERVER AND A CLIENT ON A LOCAL MACHINE:

-You should run the test on two different instances of linux terminal. In the first one you should run 34, and then on the other hand you should run 35 on which after the connection is established, you will see a list.

In the list you should first send the file to the server. The file you will send is situated in the folder called "clientFolder" and the name of the file is "aFileOf16000KiB". You can observe that this file will be divided into blocks in both client and the server (For the test to be done in the final implementation we eliminated this part of the code and use only one block to be pointed by all leaf level nodes to simulate the file thus we initiate the FlexLists accordingly at each side from the initiation). The client will also have some temp files for the blocks of the file. These temp files are also divided into blocks for tests being easier for us. Because when we change a block we will know for sure which block we are changing, ourselves (Even though the code have these implemented, they are mostly commented out, we have changed them for real life scenario tests. These can be reinitiated from the code with a bit of effort). After the send operation done, one can challenge his file and observe that it passes the verification. Later one can delete a block of the file from the "serverFolder" and challenge again to see it is not getting verified.

If the real codes are recovered and the simulation not used: !!!After doing one test, one should go to the client folder and delete all previous data except the file to be divided itself. Folder called "packet" and "aBlockOf16KiB" should also be kept. And one should also remove all previous files from "serverFolder".

The choice "Commit" should be called after making some changes to the file. The changes to the file are made using the temp versions of the blocks situated in the "clientFolder". You can add some new strings to the block and observe what diff algorithm outputs and then what kind of patches are sent to the server. We observed some bugs on this code because of googleDiffMatchPatch algorithms output sometimes do not match our requirements but the functions works well enough to observe how the protocol works. Later we may change our diff algorithm by changing BasicDiff class which was written as an adapter class.

36: Offline Diff Test - trials (alpha)

- Offline diff test uses files under "DiffTestFiles". The folder is ready for the first run and some files are provided for us to see what kind of diff operations are popping out of some changes.

37: Explicit multi hash calculation test

38: Multi hash calculation (time difference test) - outputs time values under "clientFoder"

- *In the above two tests, we show how the redundant hash calculations effect the time of multiple update operations.*

39: Multiple verify for many blocks test

40: Parallel Build for FlexList test

- *We show how a FlexList can be built by multiple cores.*

41: Tag calculation time test to compare with PDP - prints time for tag calculation (to check under your setting)

42: Multi verify vs single verify - outputs time values under "clientFoder"

43: Multi update method test

- *Some test cases to show how the multiple update operation works.*

44: Test Field: You can try small code segments by writing them down in this function.

E) Some Important Classes and Their Purposes

In this section of the document, purposes of some important classes can be found.

Following classes are mainly used for the Skip List data structure:

➔ **Class**: Node

Purpose: Class for definition of nodes in a skip list

➔ **Class**: AuthenticatedNode

Purpose: Class supports creation of authenticated nodes

➔ **Class**: RankBasedNode

Purpose: Definition and implementation of main methods of rank based nodes

➔ **Class**: ProofNode

Purpose: This is a class in which all elements required for a hash calculation for a particular node on a proof path is stored. It is also designed to be sent over the network.

➔ **Class**: SkipList

Purpose: Definition and implementation of essential methods for any skip list

➔ **Class**: AuthenticatedSkipList

Purpose: Definition and implementation of essential methods for an auth. skip list

➔ **Class**: RankBasedSkipList

Purpose: Definition and implementation of essential methods for a rank-based skip list

➔ **Class**: DPDPCONST

Purpose: This file defines DEFAULT values for different test cases

- Following classes are used for the Server-Client application of the project:

➔ **Class**: DPDPFlexProver

Purpose: This class is where the server operations are implemented. Server can respond to challenges through this class. Each client should have one instance of this object saved at the server side.

➔ **Class**: DPDPFlexVerifier

Purpose: This class is where the client operations are implemented. Client can challenge and verify through this class. Each client should have one instance of this object to verify the responses came from the prover.

➔ **Class**: PacketClient

Purpose: Packet that is created by the client and is sent to the server.

➔ **Class**: PacketBase

Purpose: This class is the base class for packets which will go through network

➔ **Class**: PacketServer

Purpose: Packet that is created by the server and is sent to the client.

➔ **Class**: ClientDPDPmain

Purpose: Connect to the host and port provided, initialize FlexDPDP variables, lets Client use the menu where she can send her file, challenge it and commit changes done.

➔ **Class**: ServerDPDPmain

Purpose: Waits for an initialization for a client, when initialization command arrives initialize FlexDPDP variables, lets Client use the menu where she can send her file, challenge her file and commit changes done.

➔ **Class**: DiffBasic

Purpose: This class is basically an adapter to GoogleDiffMatchPatch algorithm to be turned into a format that we wish, so that which any diff algorithm, on the format of DiffBasic, our system will do her job.

➔ **Class**: DiffSkipList

Purpose: This class takes the output of the diff algorithm used and turns it into block updates.

➔ **Class**: CommitProof

Purpose: This class is generated for ease of understanding the proofs generated for a series of updates.